Clustering Methods based on Deep Learning and Unimodality Testing

A Dissertation

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Georgios Vardakas

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

University of Ioannina
School of Engineering
Ioannina 2025

Advisory Committee:

- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Konstantinos Blekas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Christophoros Nikou**, Professor, Department of Computer Science and Engineering, University of Ioannina

Examining Committee:

- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Konstantinos Blekas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Christophoros Nikou**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Anastasios Tefas**, Professor, Department of Informatics, Aristotle University of Thessaloniki
- **George Vouros**, Professor, Department of Digital Systems, University of Piraeus
- **Konstantinos Skianis**, Assistant Professor, Department of Computer Science and Engineering, University of Ioannina
- **Athanasios Voulodimos**, Assistant Professor, School of Electrical and Computer Engineering, National Technical University of Athens

Dedication

«Αυτά τα δέντρα δε βολεύονται με λιγότερο ουρανό,
αυτές οι πέτρες δε βολεύονται κάτου απ' τα ξένα βήματα,
αυτά τα πρόσωπα δε βολεύονται παρά μόνο στον ήλιο,
αυτές οι καρδιές δε βολεύονται παρά μόνο στο δίκιο.»
— Γιάννης Ρίτσος, *Ρωμιοσύνη*

I dedicate this thesis to those who stand for justice, for free and public education and health care, for the right to dream without fear, for a society just, free and humane, where knowledge and opportunity belong to everyone.

I also dedicate this thesis to my family for their continuous support, and to my beloved partner, Evgenia, for her trust, love, and unwavering belief in me throughout all these years.

Acknowledgements

This thesis represents the outcome of an effort that has been as academic as personal. Its pages reflect not only research and ideas, but also the support, trust, and encouragement that I have received from many remarkable people along the way. Their presence made this path lighter, richer, and more meaningful. Before diving into the work itself, I want to express my heartfelt gratitude to those who supported me throughout this journey.

First, I would like to sincerely thank my advisor, Professor Aristidis Likas, for his invaluable guidance, trust, and encouragement throughout our collaboration, which began when I was still an undergraduate student. From the first day of our collaboration, back in 2019, until now, he has been consistently available and always eager to engage in fruitful scientific discussions. Beyond academic matters, he has also been a trusted mentor who offered me personal advice during difficult times. Our collaboration has played a crucial role in shaping the research skills and critical thinking that I possess today.

I am grateful to my advisory committee members, Prof. Konstantinos Blekas and Prof. Christophoros Nikou, for their guidance and support, and to the evaluation committee members for generously contributing their time and expertise to this thesis.

I would like to thank Senior Research Scientist Argyris Kalogeratos for insightful discussions and for his valuable assistance in presenting the UniForCE algorithm. I would also like to thank Assistant Professor John Pavlopoulos for our valuable collaborations over the years. I am also grateful to Dr. Prodromos Kolyvakis for kindly providing the implementation of Hartigan's dip test.

I would like to thank my friends and fellow colleagues Dr. Paraskevi Chasani, Ioannis Papakostas and Ioannis Georvasilis for accompanying me on this journey and for making our workplace a joyful and welcoming environment. Their friendship, discussions, collaboration, and daily interaction made working with such remarkable

individuals a truly valuable experience, one that helped me grow both as a researcher and as a person. I would like to especially thank Ioannis Papakostas for his valuable assistance in conducting experiments with the global kernel k-means++ algorithm and the soft silhouette criterion.

I would like to thank my childhood friend Grigorios Papigiotis for his unconditional support, enthusiasm, and genuine engagement in both scientific and non-scientific discussions. Our fruitful collaboration in Computational Astrophysics remains one of the most enjoyable research experiences I have been part of.

Moreover, I would like to express my heartfelt gratitude to my close friends for making everyday life more enjoyable and meaningful. I truly appreciate the time we spent together. In particular, I wish to thank Ilias Kleftakis, Giannis Zisis, Giannis Divas, Olga Skarlatou, Sokratis Gkrouidis, Theodore Tsoumanis, Sotiris and Dimitris Stratos, Christodoulos Giannakos, Konstantinos Kadoglou, Miltiadis Vasiliades, and Konstantina Kyriakoudi for their unwavering support and understanding.

I am deeply indebted to my beloved family. Their unconditional support, love, and care throughout my life is something I will always cherish. My mother, Athena, has been by my side through everything, offering strength and comfort in times of distress. My younger siblings, my brother Konstantinos and my sister Eleni-Anastasia, have shown me deep love and affection. Although they often looked up to me, they may not realize that they are far greater human beings than I will ever be. And finally, to my father and my personal hero, whose example taught me to demand more of myself, to not be afraid to face even the most difficult challenges life presents, to always offer a helping hand and to remain humble.

Last but not least, I owe a special thanks to my partner, Evgenia. Her unwavering love, care, support, and patience have accompanied me through every stage of my academic journey over the past ten years, from undergraduate studies to postgraduate and finally to PhD. Without her, this thesis would not have been possible.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Glossary

AE	Autoencoder
AEC	Autoencoder-based Clustering
AMI	Adjusted Mutual Information
ARI	Adjusted Rand Index
ClusterGAN	GAN-based clustering method with latent space partitioning
CVIs	Cluster Validation Indices
DCN	Deep Clustering Network
DCSS	Deep Clustering using the Soft Silhouette Score
DEC	Deep Embedding Clustering
DipEncoder	Autoencoder model using the Dip-test for clustering loss
Dip-test	Statistical test for unimodality
DNN	Deep Neural Network
GAN	Generative Adversarial Network
GKkM	Global kernel k-means
GKkM++	Global kernel k-Means++
GkM	Global k-means
GkM++	Global k-means++
IDEC	Improved Deep Embedding Clustering
IMLE	Implicit Maximum Likelihood Estimation
KkM++	Kernel k-means++
KL divergence	Kullback-Leibler divergence
Latent Space	Representation space learned by neural networks
LLM	Large Language Model
ML	Machine Learning
NIMLC	Neural Implicit Maximum Likelihood Clustering
Overclustering	Data partition with more clusters than expected

Pdip-means	Probabilistic Dip-means clustering
RBF	Radial Basis Function
RCC	Robust Continuous Clustering
RKkM	Kernel k-means with random uniform initialization
SMMP	Scalable Multi-modal Partitioning
t-SNE	t-distributed Stochastic Neighbor Embedding
UniForCE	Unimodality Forest for Clustering and Estimation

Abstract

Georgios Vardakas, Ph.D., Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2025.
Clustering Methods based on Deep Learning and Unimodality Testing.
Advisor: Aristidis Likas, Professor.

Data clustering is the process of partitioning a dataset into a finite set of groups, or clusters, such that data points within each cluster exhibit intra-cluster similarity, while those belonging to different clusters are characterized by inter-cluster dissimilarity. Clustering remains a challenging task due to the inherent complexity of uncovering meaningful structures within data. Revealing these hidden structures provides valuable insights and facilitates a deeper understanding of the underlying patterns.

This thesis concerns the development, implementation and evaluation of novel clustering methodologies mainly focused on three important problems: i) partitional clustering in both Euclidean and kernel spaces, ii) unimodality-based clustering, which incorporates the concept of unimodality into the clustering process, and iii) deep clustering, which leverages the representational power of deep learning methods.

We first introduce global k-means++, a method developed to address the initialization challenges inherent in the standard k-means algorithm. The approach integrates the incremental strategy of global k-means with the probabilistic center selection mechanism of k-means++, effectively combining the strengths of both techniques. The resulting synergy delivers high-quality clustering solutions while significantly reducing the computational cost typically associated with global k-means. Furthermore, we extend this concept from Euclidean to kernel space by proposing global kernel k-means++, an algorithm specifically designed to overcome the initialization problem in kernel k-means. The optimization effectiveness of both global k-means variants is thoroughly validated through extensive experimental evaluation.

Afterwards, we present UniForCE, a clustering method that simultaneously partitions data and estimates the number of clusters k. UniForCE introduces a novel notion of locally unimodal clusters, focusing on unimodality at local regions of the data density rather than in the entire cluster. By identifying unimodal pairs of neighboring subclusters, the method aggregates them into larger, statistically coherent structures via a unimodality graph. This flexible formulation enables the discovery of arbitrarily shaped clusters. A statistical test determines unimodal pairs, and clustering is achieved with automatic estimation k by detecting the number of connected components in the unimodality graph. Extensive experiments on synthetic and real datasets validate both the conceptual soundness of the method and its practical effectiveness.

Furthermore, we introduce the soft silhouette score, a generalization of the widely used silhouette measure that accommodates probabilistic cluster assignments. Building on this differentiable measure, we develop an autoencoder-based deep clustering method utilizing the soft silhouette score. Our method guides the learned latent representations to form clusters that are both compact and well-separated. This property is crucial in real-world applications, as simultaneously ensuring compactness and separability guarantees that clusters are not only densely packed but also clearly distinct from each other. We evaluate our method on a variety of benchmark datasets and against state-of-the-art methods to demonstrate that it outperforms established deep clustering approaches, highlighting the effectiveness of the soft silhouette score as a principled objective for improving the quality of learned latent representations.

Finally, we present the neural implicit maximum likelihood clustering, which is a neural-network-based approach that frames clustering as a generative task within the Implicit Maximum Likelihood Estimation framework. By adapting ideas from ClusterGAN, our method avoids several well-known shortcomings of GAN-based clustering while maintaining a simple and stable training objective. The method performs particularly well on small datasets, with experimental comparisons against both deep and conventional clustering algorithms underscoring its competitive potential. A notable strength of our method is its ability to capture diverse cluster geometries without requiring hyperparameter tuning. Experiments on synthetic datasets show that the method can successfully cluster both cloud-shaped and ring-shaped data.

Εκτεταμενη Περιληψη

Γεώργιος Βαρδάκας, Δ.Δ., Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2025.
Μέθοδοι Ομαδοποίησης βασισμένες στη Βαθιά Μάθηση και σε Έλεγχο Μονοτροπικότητας.
Επιβλέπων: Αριστείδης Λύκας, Καθηγητής.

Ομαδοποίηση ονομάζεται η διαδικασία διαμέρισης ενός συνόλου δεδομένων σε ένα πεπερασμένο σύνολο ομάδων, ή συστάδων, έτσι ώστε τα δεδομένα που εμπεριέχονται σε κάθε ομάδα να εμφανίζουν εσωτερική ομοιότητα, ενώ δεδομένα που ανήκουν σε διαφορετικές ομάδες να χαρακτηρίζονται από μεταξύ τους ανομοιότητα. Η ομαδοποίηση παραμένει μια απαιτητική διαδικασία λόγω της εγγενούς πολυπλοκότητας που παρουσιάζει η εύρεση ουσιαστικών δομών μέσα στα δεδομένα. Η ανάδειξη αυτών των κρυμμένων δομών παρέχει πολύτιμες πληροφορίες και διευκολύνει την πιο βαθιά κατανόηση των κρυμμένων προτύπων.

Η παρούσα διατριβή αφορά την ανάπτυξη, υλοποίηση και αξιολόγηση καινοτόμων μεθοδολογιών ομαδοποίησης, με κύρια εστίαση σε τρία σημαντικά ζητήματα: i) τη διαμεριστική ομαδοποίηση που βασίζεται στον αλγόριθμο k-means τόσο σε Ευκλείδειους όσο και σε χώρους που ορίζονται από συναρτήσεις πυρήνα (kernel functions), ii) την ομαδοποίηση βασισμένη στη μονοτροπικότητα, η οποία ενσωματώνει την έννοια της μονοτροπικότητας στη διαδικασία της ομαδοποίησης iii) τη βαθιά ομαδοποίηση, η οποία αξιοποιεί την ικανότητα δημιουργίας αποδοτικών αναπαραστάσεων.

Αρχικά παρουσιάζουμε τον global k-means++, μια μέθοδο που αναπτύχθηκε για να αντιμετωπίσει τις προκλήσεις αρχικοποίησης που είναι εγγενείς στον κλασικό αλγόριθμο k-means. Η προσέγγιση αυτή ενσωματώνει την επαναληπτική στρατηγική επίλυσης του αλγορίθμου global k-means με τον πιθανοκρατικό μηχανισμό επιλογής κέντρων του αλγορίθμου k-means++, συνδυάζοντας αποτελεσματικά τα

πλεονεκτήματα και των δύο τεχνικών. Η προκύπτουσα συνέργεια παρέχει λύσεις ομαδοποίησης υψηλής ποιότητας, μειώνοντας ταυτόχρονα σημαντικά το υπολογιστικό κόστος που συνήθως συνδέεται με τον αλγόριθμο global k-means. Επιπλέον, επεκτείνουμε την ιδέα αυτή από τον Ευκλείδειο χώρο σε χώρους χαρακτηριστικών που ορίζονται από συναρτήσεις πυρήνα, προτείνοντας τον global kernel k-means++, έναν αλγόριθμο σχεδιασμένο για να αντιμετωπίσει το πρόβλημα της αρχικοποίησης του κλασικού αλγορίθμου kernel k-means. Η ικανότητα ελαχιστοποίησης και των δύο προτεινόμενων παραλλαγών του global k-means επιβεβαιώνεται διεξοδικά μέσα από εκτεταμένη πειραματική αξιολόγηση.

Στη συνέχεια παρουσιάζουμε τον αλγόριθμο UniForCE, μια μέθοδο ομαδοποίησης που ταυτόχρονα διαμερίζει τα δεδομένα και παρέχει μία εκτίμηση για τον αριθμό των ομάδων. Ο UniForCE εισάγει την έννοια των τοπικά μονοτροπικών ομάδων, εστιάζοντας στη μονοτροπικότητα σε τοπικές περιοχές των δεδομένων αντί σε ολόκληρη την ομάδα. Με την αναγνώριση μονοτροπικών ζευγών γειτονικών υποομάδων, η μέθοδος τις συνενώνει σε μεγαλύτερες, στατιστικά συνεκτικές δομές μέσω ενός γράφου μονοτροπικότητας. Αυτή η ευέλικτη διατύπωση καθιστά δυνατή την ανακάλυψη ομάδων αυθαίρετου σχήματος. Επιπρόσθετα, προτείνεται ένα στατιστικό τεστ για τον καθορίσμο των μονοτροπικών ζευγών και η ομαδοποίηση επιτυγχάνεται μέσω του εντοπισμού των συνδεδεμένων συνιστωσών στον γράφο μονοτροπικότητας. Με τον τρόπο αυτό καθιρίζεται αυτόματα και ο αριθμός των ομάδων. Εκτεταμένα πειράματα σε συνθετικά και πραγματικά σύνολα δεδομένων επικυρώνουν τόσο την ορθότητα της μεθόδου όσο και την πρακτική της αποτελεσματικότητα.

Κατόπιν, εισάγουμε το soft silhouette, μια γενίκευση του ευρέως χρησιμοποιούμενου δείκτη silhouette, το οποίο υποστηρίζει πιθανοκρατικές αναθέσεις ομάδων στα δεδομένα. Βασιζόμενοι σε αυτό το διαφορίσιμο κριτήριο, προτείνουμε μια μέθοδο βαθιάς ομαδοποίησης με χρήση autoencoder, η οποία δημιουργεί αναπαραστάσεις που σχηματίζουν ομάδες που είναι ταυτόχρονα συμπαγείς αλλά και σαφώς διαχωρισμένες. Η μέθοδος αξιολογείται σε διάφορα σύνολα δεδομένων και συγκρίνεται με τις πλέον σύγχρονες μεθόδους. Φαίνεται να υπερτερεί έναντι καθιερωμένων προσεγγίσεων βαθιάς ομαδοποίησης, αναδεικνύοντας έτσι την αποτελεσματικότητα του κριτηρίου soft silhouette ως συνάρτηση στόχο για τη βελτίωση της ποιότητας των κρυμμένων αναπαραστάσεων.

Τέλος παρουσιάζουμε τη μέθοδο neural implicit maximum likelihood clustering,

μια προσέγγιση βασισμένη σε νευρωνικά δίκτυα, η οποία αντιμετωπίζει την ομαδοποίηση ως παραγωγική (generative) διαδικασία στο πλαίσιο της μεθόδου έμμεσης μεγιστοποίησης της πιθανοφάνειας (Implicit Maximum Likelihood Estimation). Προσαρμόζοντας ιδέες από τον αλγόριθμο ClusterGAN, η προτεινόμενη μέθοδος αποφεύγει αρκετές γνωστές αδυναμίες της ομαδοποίησης που βασίζεται στα Generative Adversarial Networks, διατηρώντας παράλληλα έναν απλό και ευσταθή κριτήριο εκπαίδευσης. Η μέθοδος επιτυγχάνει ιδιαίτερα καλά αποτελέσματα σε μικρά σύνολα δεδομένων, με πειραματικές συγκρίσεις τόσο έναντι μεθόδων βαθιάς όσο και συμβατικής ομαδοποίησης. Ένα αξιοσημείωτο πλεονέκτημα της μεθόδου είναι η ικανότητά της να επιλύει ποικίλες γεωμετρίες ομάδων χωρίς να απαιτείται ρύθμιση υπερπαραμέτρων. Πειράματα σε συνθετικά σύνολα δεδομένων δείχνουν ότι η μέθοδος μπορεί να ομαδοποιήσει επιτυχώς τόσο δεδομένα σε μορφή «νέφους» όσο και δεδομένα σε μορφή «δακτυλίου».

Chapter 1

Introduction

An enormous volume of new data is generated on a daily basis, leading researchers to develop methods to organize it, extract meaningful knowledge, and interpret the information. As the volume of data continues to grow rapidly, the task of uncovering valuable information becomes increasingly complex. An effective approach to managing data is to group them into sensible groups that are typically called clusters. This process creates a condensed representation of the information, revealing underlying similarities and differences, as well as hidden structures and patterns that might otherwise remain undetected.

Machine learning (ML) is the area of artificial intelligence that equips algorithms with the ability to learn from examples [5, 6]. It has emerged as a powerful tool, fundamentally changing how vast amounts of data are analyzed and interpreted by enabling algorithms to discover complex, non-obvious patterns within high-dimensional datasets [7, 8]. ML is typically divided into two main paradigms: *supervised learning* and *unsupervised learning*, with classification and clustering being their most representative problems, respectively. In classification, the goal is to assign a data point

x to one of a finite set of discrete class labels, that is, to appropriately categorize it. This is achieved by constructing a classifier, a parametric function that maps inputs to predicted class labels. To determine the parameters of this function, an inductive learning algorithm is employed, which minimizes an empirical risk objective over a finite labeled dataset $X = \{(x_i, y_i)\}_{i=1}^{N}$, where each x_i is a data instance and y_i is its corresponding class label. The goal of classification is to train a classifier on the labeled dataset X, enabling it to accurately predict the labels of previously unseen instances.

The acquisition of labeled data is often expensive and in many cases, impractical or even an impossible requirement. This thesis focuses on the clustering problem, a core and fundamental task in machine learning, where the dataset contains no labeled information and is defined as $X = \{x_i\}_{i=1}^{N}$, consisting solely of unlabeled instances. Data clustering is the process of partitioning a dataset into a finite number of groups, known as clusters, such that data points within each cluster share common characteristics and are different from those in other clusters [9]. Although its definition is intuitively simple, clustering remains a challenging problem due to the complexity of identifying meaningful structures in high-dimensional or noisy data. Discovering these hidden structures can provide valuable insight and enable deeper understanding of the underlying patterns. As a result, clustering has become a widely used technique across various domains in computer science such as data mining, pattern recognition, image segmentation and spatial database analysis, as well as in numerous scientific disciplines, including life sciences, medical research, and economics [10, 11].

Clustering typically relies on a similarity or dissimilarity measure, such as Euclidean distance, to describe the relationships between data points [9]. Clusters are then formed by applying a suitable clustering algorithm based on this measure. The choice of both the proximity measure and the clustering algorithm has a significant impact on the resulting partitioning; different choices can lead to drastically different outcomes in terms of cluster shapes, quality of solution, and even number of clusters. Unlike classification, where each training data point x_i is associated with a known class label, clustering is inherently unsupervised and subjective. There is no ground truth to indicate how the data should be grouped, making the clustering process highly dependent on the context and assumptions made. In most cases, even the number of clusters is not known a priori, further complicating the task. Consequently, evaluating the quality of a clustering solution is not straightforward and

often depends on the specific application and the goals of the analysis.

It is well known that most conventional clustering methods perform effectively on low-dimensional and relatively simple data, but often struggle with high-dimensional or complex modalities, such as images. To address these inherent limitations, recent research has turned to deep neural networks (DNNs) for clustering tasks within the broader framework of deep learning. DNNs are capable of learning rich and informative representations directly from raw data, minimizing the need for manual feature engineering [12]. Their exceptional nonlinear modeling capacity and architectural flexibility have been shown to enhance both supervised and unsupervised learning tasks [13, 14].

Although clustering was not the original focus of Deep Learning (DL) research, a growing body of work has adapted DNNs specifically for clustering purposes, giving rise to the *deep clustering* category of methods. These methods aim to leverage the representational power of neural networks to transform the input data into a latent space where the structure is more appropriate for clustering. In particular, they strive to produce *cluster-friendly* embeddings, where data points form compact and, ideally, well-separated clusters [15, 16, 17, 18, 19].

This thesis concerns the development, implementation and evaluation of novel (unsupervised) clustering methodologies mainly focused on three important and very active machine learning problems, namely: i) partitional clustering in both Euclidean and kernel spaces, ii) unimodality-based clustering and iii) deep clustering, which leverages the representational power of deep learning methods. In this Chapter, we describe these problems, along with a review of the related work. Afterward, we present the main contributions and the layout of the thesis.

1.1 Partitional Clustering

The typical form of clustering is the partitioning of a given dataset $X = \{x_1, \ldots, x_N\}$, $x_i \in \mathbb{R}^d$ into K disjoint clusters $\mathcal{C} = \{C_1, \ldots, C_K\}$ so that a specific criterion is optimized. The most widely used optimization criterion is the clustering error. It is defined as the within-cluster sum of squared distances between each data $x_i \in C_k$ to

its cluster center μ_k as defined in

$$E(C) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{1}_{C_k}(x_i) ||x_i - \mu_k||^2, \tag{1.1}$$

where $\mathbf{1}_{C_k}$ is the indicator function of the set C_k, while $M = \{\mu_1, \ldots, \mu_K\}$ is the set of K centers computed as the mean of the data points of each cluster. The number of ways in which a set of N objects can be partitioned into K non-empty groups is given by Stirling numbers of the second kind:

$$S(N, K) = \frac{1}{K!} \sum_{k=0}^{K} (-1)^k (K-k)^N \binom{K}{k}, \tag{1.2}$$

which can be approximated by $K^N/K!$ as $N \to +\infty$ [20]. It is evident from eq. 1.2 that a complete enumeration of all possible clusterings to determine the global minimum of eq. 1.1 is computationally prohibitive. It is worth mentioning that this non-convex optimization problem is NP-hard [21, 22] not only for two clusters ($K = 2$) [23], but also for two-dimensional datasets ($D = 2$) [24].

The k-means algorithm [25, 26] is a widely used method for minimizing clustering error. It is an iterative algorithm that has been extensively utilized in many clustering applications due to its simplicity and speed [27]. However, the k-means algorithm suffers from two main limitations:

1. **Sensitivity to Initial Centers**: The solution relies critically on the initial placements of the cluster centers. It is possible that, due to poor initialization, the k-means may converge to poor local minima of the clustering error.

2. **Linear Separability**: The resulting clusters are restricted to being linearly separable, limiting the algorithm's effectiveness in identifying complex cluster shapes.

In the remainder of this section, we present the standard k-means algorithm along with several variants that have been proposed in the literature to overcome these limitations.

1.1.1 k-means for clustering in Euclidean space

k-means

The *k-means* algorithm is computationally efficient, conceptually simple, and arguably the most widely used clustering method in the literature. However, its performance

is highly sensitive to the initialization of cluster centers. As a result, the k-means algorithm often converges to a *locally optimal solution* with respect to the clustering error (eq. 1.1). It is an iterative center-based clustering algorithm that starts with K cluster centers. In its purest form (Lloyd's algorithm [26]), the K centers are typically selected uniformly at random from the set of data points X. Then the two-step algorithmic procedure follows iteratively until convergence, which includes the data assignment and the center optimization steps. At the assignment step, every data point x_i is assigned to the cluster C_j with the nearest center μ_j:

$$j = \arg\min_k ||x_i - \mu_k||^2. \tag{1.3}$$

In the optimization step, each center is updated to the mean of all data points assigned to its cluster:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i. \tag{1.4}$$

This simple optimization methodology has been proven to be computationally fast and effective. The main disadvantage of the standard k-means algorithm is that it is a local optimization method with high sensitivity to the original starting positions of the cluster centers. Poor center selection usually leads to local minima of the clustering error. Therefore, to obtain near-optimal solutions using the k-means algorithm, several runs must be scheduled differing in the starting positions of the cluster centers. Among these runs, the solution with the lowest clustering error is naturally chosen and retained. However, it has been shown that this procedure generally does not produce satisfactory results [27].

It should be noted that various stochastic global optimization methods have been proposed, such as simulated annealing and genetic algorithms, to overcome the above problem, but have not gained wide acceptance. These types of methods involve several hyperparameters that are difficult to tune, for example, starting temperature, cooling, schedule, population size, and crossover/mutation probability and usually require a large number of iterations, which renders them prohibitive for large datasets. Consequently, stochastic global optimization methods are usually not preferred, but instead, methods with multiple random restarts are commonly used [9]. However, there are some promising genetic approaches in the literature, such as Recombinator-k-means [28].

k-means++

The *k-means++* [29] is probably the most widely used center initialization algorithm. It selects the cluster centers by sampling data points from a multinomial probability distribution that strives to effectively spread them away from each other. Specifically, the algorithm comprises two key steps: i) computing a probability distribution for center selection and ii) sampling a data point from this distribution as the initial cluster center position. The method iteratively recalculates the distribution in order to select subsequent cluster centers. This iterative process terminates when all K centers have been initialized.

Specifically, the method begins by choosing the first center position through uniform random selection from the dataset X. Then it computes the distances

$$d_i = \min_j ||x_i - \mu_j||^2, \tag{1.5}$$

of each data point x_i $(i = 1, \ldots, N)$ from its nearest center and forms the probability vector $P = (p_1, \ldots, p_N)$, where each component p_i is given by

$$p_i = Pr(\mu = x_i) = d_i \Big/ \sum_{j=1}^{N} d_j. \tag{1.6}$$

Then, it samples the next center position μ from the data using the multinomial distribution with probability vector P. This two-step procedure is repeated until the number of centers is equal to K. After the k-means++ center initialization procedure, the standard k-means algorithm proceeds with the assignment and optimization steps, described in eq. 1.3 and eq. 1.4 respectively. It is important to note that the k-means++ initialization is computationally efficient and guarantees that the expected clustering cost is at most a factor of $\mathcal{O}(\log k)$ larger than the optimal cost [29].

Global k-means

The *global k-means* [3] algorithm has also been proposed to effectively solve the k-means initialization problem and is considered one of the most widely used k-means initialization variants. It constitutes a deterministic global optimization method that employs the k-means algorithm as *local search procedure* and does not depend on the initialization of the centers or empirically adjustable parameters. In particular, several empirical studies have shown that both global k-means and k-means++ consistently outperform the standard k-means algorithm in terms of clustering quality [30, 31].

Algorithm 1.1 The global k-means [3]

Require: $X = \{x_1, \ldots, x_N\}$: Dataset
Require: K: Number of clusters
1: $\mu_1 \leftarrow \frac{1}{|X|} \sum_{x_n \in X} x_n; \quad M_1 \leftarrow \{\mu_1\}$
2: **for** $k = 2, \ldots, K$ **do**
3: **for all** $x_n \in X$ **do**
4: $\{(C_k^{(n)}, M_k^{(n)}, E(C_k^{(n)}))\} \leftarrow$ Run k-means with initial k centers positions $M_{k-1} \cup \{x_n\}$
5: **end for**
6: $(C_k, M_k) \leftarrow$ Solution with the minimum error $E(C_k^{\star(n)})$ among the N solutions $\{(C_k^{(n)}, M_k^{(n)}, E(C_k^{(n)}))\}, n = 1, \ldots, N$
7: **end for**
8: **return** solutions (C_k, M_k) for every $k \in \{1, \ldots, K\}$

The global k-means algorithm is outlined in Alg. 1.1. Importantly, instead of randomly selecting initial values for the cluster centers, the global k-means algorithm incrementally adds one new cluster center at each stage in an attempt to be optimally placed. To accomplish that, the global k-means algorithm solves a clustering problem with K clusters by sequentially solving every intermediate sub-problem with k clusters ($k \in \{1, \ldots, K\}$). In order to solve the problem with k clusters, the obtained solution with $k-1$ cluster centers is exploited.

Note that the set of k centers is denoted as M_k, the k clustering partition is denoted as C_k, and $E(C)$ denotes the clustering error, see eq. 1.1. The algorithm starts by solving the 1-means problem where the optimal position corresponds to the center of the dataset X (step 1 in Alg. 1.1). Then, it solves the 2-means problem by performing N executions of the k-means algorithm (steps 3-5 in Alg. 1.1). In each execution n, the first cluster center is always initialized at the optimal solution of the 1-means sub-problem, while the second center is initially set at the data point x_n ($n \in \{1, \ldots, N\}$). The best solution, with the lowest error (eq. 1.1), is obtained after the N executions of the k-means algorithm is considered the solution for the 2-means clustering sub-problem (step 6 in Alg. 1.1). Following the same incremental procedure, the solution for k_{th} clusters is obtained (steps 2-7 in Alg. 1.1). In general, for solving the k_{th} cluster sub-problem the procedure begins with the initialization of the $k-1$ centers at the center positions provided by the solution of the $(k-1)$ problem; then, the new k_{th} center is initialized at each data point x_n. The k-means algorithm is executed N times while retaining the best clustering solution. The global k-means algorithm is very successful in obtaining near-optimal solutions, but computationally expensive for large N as it involves $\mathcal{O}(NK)$ executions of k-means on the entire dataset.

It is important to note that the performance of random initialization methods, such

as k-means++, tends to deteriorate as the number of clusters K increases, especially compared to more robust strategies such as the global *k-means* algorithm. High values of K are often used in *overclustering* scenarios, where the goal is to uncover finer-grained substructures within the dataset. In general, a clustering of X is considered an overclustering when $K > k^\star$, where $k^\star$ denotes the true number of underlying clusters in X. In various fields such as speech recognition [32] and computational biology [33], overclustering is not only widely applied but often necessary to capture fine-grained data structure. For example, in computational biology, overclustering ensures that relevant cell types can be discovered even if an expected population is split into sub-populations. Moreover, overclustering is vital in more sophisticated clustering algorithms as an algorithmic step, particularly when dealing with non-convex data structures [34, 35]. In overclustering methods, obtaining solutions from the global k-means clustering algorithm is desirable. However, due to the computational burden associated with large values of N and K, computationally cheaper alternatives are practically utilized.

Although global k-means is widely acknowledged for its optimization capabilities, its prominent drawback lies in its high computational complexity. Specifically, each k clustering sub-problem necessitates $\mathcal{O}(N)$ k-means executions. Consequently, the practical application of the algorithm is predominantly confined to small datasets. This limitation has prompted researchers to explore heuristic techniques for selecting initial center candidates, with the dual objective of reducing computational complexity while maintaining clustering results of comparable quality to those obtained with global k-means. Nonetheless, it is essential to acknowledge that reducing the computational complexity of the global k-means algorithm results in the loss of its deterministic nature.

Fast global k-means and variants

The *fast global k-means* algorithm (FGKM) [3] constitutes an effort to accelerate the global k-means. Unlike global k-means, which necessitates $\mathcal{O}(N)$ k-means executions for each k sub-problem, the FGKM algorithm performs only a single k-means execution. Alg. 1.2 describes the FGKM clustering method.

In particular, for each value of k, the FGKM algorithm computes an upper bound (denoted as E_n) on the clustering error that would arise if the new center was initialized at the specific point x_n and k-means were executed until convergence. The

Algorithm 1.2 The fast global k-means [3]

Require: $X = \{x_1, \ldots, x_N\}$: Dataset
Require: K: Number of clusters
1: $\mu_1 \leftarrow \frac{1}{|X|} \sum_{x_n \in X} x_n; \quad M_1 \leftarrow \{\mu_1\}$
2: **for** $k = 2, \ldots, K$ **do**
3: $x_{i^\star} \leftarrow$ compute initial position of new ceneter (eq. 1.8)
4: $(C_k, M_k) \leftarrow$ Run k-means with initial k centers positions $\{M_{k-1}\} \cup \{x_{i^\star}\}$
5: **end for**
6: **return** solutions (C_k, M_k) for every $k \in \{1, \ldots, K\}$

upper bound is defined as $E_n \leq E - b_n$, where E is the error already found for the $k-1$ clustering sub-problem, while the value b_n is given by equation

$$b_n = \sum_{j=1}^{N} \max(d_{k-1}^j - ||x_n - x_j||^2, 0). \tag{1.7}$$

In eq. 1.7, the d_{k-1}^j is defined as the squared euclidean distance between x_j and its closest cluster center among the $k-1$ centers obtained so far. The initial position of the k-th cluster center is set to the data point $x_i^\star$ with minimum E_n or, equivalently, with maximum b_n:

$$i^\star = \arg\max_n b_n. \tag{1.8}$$

Several methods have been proposed that modify the global k-means or the fast global k-means to make it more efficient [30]. The efficient global k-means clustering algorithm (EGKM) [36] defines a normalized density function criterion for selecting the top candidate for the new cluster center. However, EGKM similarly to FGKM demands all pairwise distances $d(x_i, x_j)$, thus it needs $\mathcal{O}(N^2)$ distance computations and extra memory space. The fast modified global k-means algorithm (FMGKM) [37] defines a more sophisticated auxiliary function criterion for selecting the candidates compared to FGKM. Its main drawback is computational complexity because, at each iteration, it requires the entire or part of the affinity matrix computation, which also generally requires $\mathcal{O}(N^2)$. In addition, the FMGKM algorithm exhibits a secondary limitation due to the utilization of an auxiliary function criterion, which introduces a non-convex optimization problem. The fast global k-means clustering based on local geometric information [38] is an attempt to reduce the computational complexity of FGKM [3] and FMGKM [37] by leveraging local geometric structure information to decrease the distance computations required in each iteration. The method is claimed to require $\mathcal{O}(n_1 n_2)$ distance computations to select the new center candidate, where empirically $n_1 \ll N$ and $n_2 \ll N$. Fast global k-means clustering using cluster mem-

bership and inequality [39] is also an attempt to reduce the distance computations of the FGKM. It is claimed that the method requires $\mathcal{O}(Nn_1)$ distance computations to select the new center candidate, where empirically $n_1 \ll N$. It is crucial to emphasize that all related methods exhibit comparable clustering performance to FGKM and inferior to global k-means.

1.1.2 Kernel k-means for clustering in feature space

Kernel k-means

The *kernel k-means* algorithm [40] extends the typical k-means algorithm to another space of higher dimension, called feature space $\mathcal{F}$. This is achieved by mapping original data instances using a non-linear function $\phi : \mathcal{X} \to \mathcal{F}$. In the transformed space, the algorithm minimizes the clustering error as presented in the following equation (weighted version):

$$E(\mathcal{C}) = \sum_{i=1}^{N}\sum_{k=1}^{K} \mathbf{1}_{C_k}(x_i) w_i ||\phi(x_i) - m_k||^2, \tag{1.9}$$

$$\text{where } m_k = \frac{\sum_{i=1}^{N} \mathbf{1}_{C_k}(x_i) w_i \phi(x_i)}{\sum_{i=1}^{N} \mathbf{1}_{C_k}(x_i) w_i}. \tag{1.10}$$

In the above equation, the weights w_i are nonnegative scalars assigned to each data point x_i. The mapping $\phi(x_i)$ provides a representation of the data in the feature space. In the general case $\phi(x_i)$ is never computed explicitly. Instead, computations rely on the kernel matrix, denoted by $\mathbf{K} \in \mathbb{R}^{N\times N}$, where each element $\mathbf{K}_{ij} = \phi(x_i)^T\phi(x_j)$ represents the inner product similarity in the feature space between instance i and j. For the kernel matrix to be valid, it should be positive semidefinite [10, 41].

By utilizing the kernel trick [40, 41], the squared distances in eq. 1.9 can be computed without explicitly knowing the transformation ϕ, as shown in the following equation:

$$||\phi(x_i) - m_k||^2 = \mathbf{K}_{ii} - \frac{2\sum_{j=1}^{N} \mathbf{1}_{C_k}(x_j) w_j \mathbf{K}_{ij}}{\sum_{j=1}^{N} \mathbf{1}_{C_k}(x_j) w_j} + \frac{\sum_{j=1}^{N}\sum_{l=1}^{N} \mathbf{1}_{C_k}(x_j)\mathbf{1}_{C_k}(x_l) w_j w_l \mathbf{K}_{jl}}{\sum_{j=1}^{N}\sum_{l=1}^{N} \mathbf{1}_{C_k}(x_j)\mathbf{1}_{C_k}(x_l) w_j w_l}. \tag{1.11}$$

The kernel k-means does not explicitly use the transformation $\phi(x)$; therefore, the centers m_k of the clusters in the feature space cannot be computed directly. How-

ever, the values of the kernel matrix provide the information required to compute the feature space distance between instances and cluster centers. Kernel k-means can achieve non-linear cluster separation, thereby addressing the second k-means limitation. Unless specified otherwise, we assume the unweighted version of the algorithm, i.e. $w_i = 1$, $i = 1, \ldots, N$.

Algorithm 1.3 `Kernel` k`-Means`

Require: **K**: Kernel matrix
Require: K: Number of clusters
Require: $\{C_1, C_2, \ldots, C_K\}$: Cluster Initialization
Initialize $converged \leftarrow$ False
while not $converged$ **do**
  for $k = 1, \ldots, K$ **do**
    for all $x_i \in X$ **do**
      Compute $\|\phi(x_i) - m_k\|^2$ using eq. 1.11
    end for
  end for
  for all $x_i \in X$ **do**
    Assign x_i to the cluster with the nearest center $c^\star(x_i) \leftarrow \arg\min_k ||\phi(x_i) - m_k||^2$
  end for
  for $k = 1, \ldots, K$ **do**
    Update cluster $C_k \leftarrow \{x_i | c^\star(x_i) = k\}$
  end for
  if no change in cluster labels **then**
    $converged \leftarrow$ True
  end if
end while
return final partition $\mathcal{C}^\star = \{C_1, C_2, \ldots, C_K\}$ and clustering error $E(\mathcal{C}^\star)$ using eq. 1.9

It is known that given a positive semidefinite kernel matrix, the kernel k-means monotonically converges to a local minimum of clustering error in the feature space. In this case, the computational complexity of the algorithm is $\mathcal{O}(N^2\tau)$, where τ denotes the number of iterations to convergence [42]. It is important to point out that applying kernel k-means does not require the data vectors themselves, but only the values of the kernel matrix **K**. The detailed kernel k-means algorithm is described in Alg. 1.3. It should be emphasized that the kernel k-means requires an initial partition to be specified. At each iteration, the partition is updated in order to reduce the clustering error in feature space. It should be noted that in most software packages, the standard method for initializing kernel k-means is either random initialization [26] or random

instance labeling, also known as Forgy's method [43]. The former assigns cluster centers by selecting random positions in the feature space, while the latter initializes clusters by directly assigning each data point to a randomly chosen cluster. It is widely known that both strategies very often converge to poor suboptimal solutions.

Kernel k-means++

As previously mentioned, k-means++ [29] is a successful algorithm for selecting initial center positions in Euclidean space, offering provable guarantees. The intuition behind the k-means++ initialization algorithm is to select a set of well-dispersed initial centers throughout the dataset. This makes the algorithm less likely to converge to poor local minima, leading to improved clustering performance. Thus, we aimed to evaluate its performance in feature space clustering when used as an initialization method for kernel k-means. Therefore, we formulate the *kernel k-means++* [44] (KkM++) algorithm that can be derived from k-means++ using the feature space distance formulation described in Alg. 1.4.

Algorithm 1.4 `Kernel k-Means++ Initialization`

Require: $\mathbf{K}$: Kernel matrix
Require: K: Number of clusters
Initialize the set of cluster centers $M \leftarrow \{\}$
Initialize the clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ where $C_i \leftarrow \{\}$, $\forall i = 1, 2, \ldots, K$
Choose a data instance $\mu_1 \sim \mathcal{U}(X)$
$M \leftarrow M \cup \{\mu_1\}$
for $k = 2, \ldots, K$ **do**
  for all $x_i \in X$ **do**
    Compute distance $d_i \leftarrow \min_j d\left(\phi(x_i), \phi(\mu_j)\right)$ using eq. 1.12
  end for
  Choose a data instance $\mu_k \leftarrow x_i$ at random from X with probability $p_i \leftarrow \frac{d_i}{\sum_{j=1}^{n} d_j}$
  $M \leftarrow M \cup \{\mu_k\}$
end for
for all $x_i \in X$ **do**
  Assign x_i to the cluster with the nearest center $c^\star(x_i) \leftarrow \arg\min_k d\left(\phi(x_i), \phi(\mu_k)\right)$ using eq. 1.12
end for
for $k = 1, \ldots, K$ **do**
  Compute initial cluster $C_k \leftarrow \{x_i | c^\star(x_i) = k\}$
end for
return clustering initialization $\mathcal{C}^\star = \{C_1, C_2, \ldots, C_K\}$

In the following formulation, μ and m represent a cluster center in Euclidean and feature spaces, respectively. Specifically, KkM++ requires the number of clusters K and a kernel matrix $\mathbf{K}$ as input. The algorithm begins by initializing the first cluster center μ_1 by randomly selecting a data instance x_i from X (Step 3 in Alg. 1.4). For each data instance x_i, it computes the distance $d_i = \min_k ||\phi(x_i) - m_k||^2$, where m_k represents the closest cluster center in feature space, defined as $m_k = \phi(\mu_k)$ (Steps 6-8 in Alg. 1.4). The probability vector $P = (p_1, \ldots, p_N)$ is then defined, where each component p_i is given by $p_i = Pr(m = \phi(x_i)) = d_i \Big/ \sum_{j=1}^{N} d_j$ (Step 9 in Alg. 1.4). Since each cluster center $m_k = \phi(\mu_k) = \phi(x_j)$ is initialized by a sampled data instance x_j from the dataset X, the distance is computed using kernel matrix values:

$$d\left(\phi(x_i), m_k\right) = d\left(\phi(x_i), \phi(\mu_k)\right) = ||\phi(x_i) - \phi(x_j)||^2 = \mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj}. \tag{1.12}$$

Subsequently, the next center m is sampled from the dataset based on the multinomial distribution defined by the probability vector P, and the process is repeated until all K centers are initialized (Steps 5-11 in Alg. 1.4). Note that even though we do not have direct access to data instances $\phi(x)$ or cluster centers m in the feature space, the described process allows us to compute the distribution of distances between each data instance and center in the feature space, thus enabling us to sample the next cluster center directly from X.

Global kernel k-means

The *global kernel k-means* (GKkM) [45] is an extension of the global k-means algorithm for feature space clustering error minimization (eq. 1.9). GKkM employs kernel k-means as a local minimization procedure and operates incrementally, solving all subproblems for $k = 1, \ldots, K$ successively. The underlying idea is that a near-optimal solution for k clusters can be attained by first obtaining a near-optimal solution for $k-1$ clusters and then initializing the k_{th} cluster N times, with each initialization starting from a different data instance. Among the N solutions, the one with the smallest clustering error is selected for the k-clustering problem. GKkM presents very satisfactory optimization capabilities, provides all solutions for all $k \in \{1, \ldots K\}$, and is also deterministic since it does not depend on initialization. However, its primary limitation is the high computational complexity inherited from GkM. If the final number of clusters is K, then KN kernel k-means executions are required, resulting in an overall complexity of $\mathcal{O}(N^3 K\tau)$, assuming the kernel matrix has been precomputed [45].

Algorithm 1.5 `Global Kernel` k`-Means`

Require: $\mathbf{K}$: Kernel matrix

Require: K: Number of clusters

1: Initialize $\mathcal{C}_1^\star \leftarrow X$

2: **for** $k = 2, \ldots, K$ **do**

3: **for all** $x_n \in X$ **do**

4: $C'_k \leftarrow \{x_n\}$

5: $\mathcal{C}'_{k-1} \leftarrow \mathcal{C}^\star_{k-1} / \{x_n\}$

6: $\mathcal{C}'_k \leftarrow \mathcal{C}'_{k-1} \cup C'_k$

7: $(\mathcal{C}^n_k, E(\mathcal{C}^n_k)) \leftarrow$ Run `kernel` k`-means`$(\mathbf{K}, k, \mathcal{C}'_k)$ (Alg. 1.3)

8: **end for**

9: $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k)) \leftarrow$ Solution with the minimum error $E(\mathcal{C}^\star_k)$ among the N partitions $\mathcal{C}^n_k$

10: **end for**

11: **return** solutions $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k))$ for every $k \in \{1, \ldots, K\}$

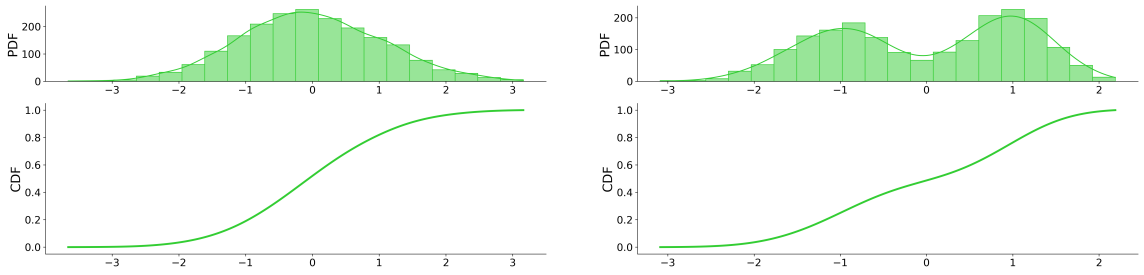
The details of GKkM are outlined in Alg. 1.5. Specifically, the algorithm requires the number of clusters K and a kernel matrix $\mathbf{K}$ as input. It should be noted that C_k refers to the k_{th} cluster in the partition, while $\mathcal{C}_k$ denotes the entire clustering solution for k clusters. At first, the method addresses the kernel 1-means subproblem by setting the clustering solution $\mathcal{C}_1$ to represent the entire dataset X (Step 1 in Alg. 1.5). Next, to tackle the kernel 2-means subproblem, kernel k-means is executed N times. In each of these N executions, the second cluster, C'_2, is initialized using a data instance x_n, which is removed from the previously obtained clustering solution $\mathcal{C}'_1$ (Steps 4 and 5 in Alg. 1.5). The initialization for two clusters is constructed by combining the solution $\mathcal{C}'_1$ with the single-point cluster C'_2 (Step 6 in Alg. 1.5). The Kernel k-means then further optimizes the clustering objective using this initialization (Step 7 in Alg. 1.5). After all data instances have been considered as possible initializations for the second cluster (i.e., kernel k-means has been executed N times), the lowest error solution is selected as the final solution for the kernel 2-means problem (Step 9 in Alg. 1.5). This process is repeated incrementally for every $k = 2, \ldots, K$, each time exploiting the solution with $(k-1)$ clusters. Finally, the algorithm provides solutions for all $k = 1, \ldots, K$.

1.2 Unimodality

In most real-world scenarios, we often work with data samples that are assumed to be generated from some unknown underlying distribution. From these samples, the goal is to infer various characteristics of the underlying distribution, which in turn provide insights into the internal structure of the system under investigation. One key statistical property of interest is whether the underlying distribution is unimodal or multimodal, that is, whether it has a single peak or multiple peaks [46]. These peaks, known as modes, often appear in the sampled data as cluster regions where a large number of data points are densely concentrated. This grouping behavior becomes apparent when examining a histogram of the data, where a high concentration of data points forms a "hill" around the mode. A unimodal underlying distribution suggests that the data originate from a single group or type of observation. In contrast, a multimodal distribution may indicate the presence of two or more distinct groups within the data, each exhibiting different behaviors or characteristics. For example, suppose that one measures the brightness of stars in a small region of a galaxy. If the histogram of star brightness shows two clear modes, this might mean there are two different groups of stars, i.e. one group that is older and dimmer and another that is younger and brighter. This kind of pattern can possibly tell us that the stars in that area did not all form at the same time. On the other hand, if the histogram has just one peak, it could mean that the stars are more similar to each other, possibly formed in the same star-forming event. In either case, observing unimodality or multimodality can raise new research questions and ultimately lead to a deeper understanding of the data under study. An illustrative example of univariate unimodal and bimodal distributions is presented in Fig. 1.1a and Fig. 1.1b, respectively.

1.2.1 Unimodality Definition

The notion of unimodality [47] is a statistical property characterizing a probability density function. A *univariate* probability density f is *unimodal* if there exists a point $m \in \mathbb{R}$, called *mode*, such that f is non-decreasing in $(-\infty, m)$ and is non-increasing in (m, ∞). Qualitatively, this means that f admits its maximum value at the mode m, and as we move away from m, it can only remain constant or decrease, with no other local maxima present. The unimodality property can also be characterized in terms of the Cumulative Distribution Function (CDF). Specifically, a CDF is unimodal



(a) Unimodal case.

(b) Bimodal case.

Figure 1.1: Histogram plots of a unimodal and a bimodal distribution (top row) and the corresponding CDF plots (bottom row). (a) The dataset corresponding to the left histogram has a dip value of 0.00 with a p-value of 1.00. (b) The dataset corresponding to the right histogram has a dip value of 0.02 with a p-value of 0.00.

if there exist two points x_l and x_u such that the function can be partitioned into three regions: (a) a convex part in $(-\infty, x_l)$, (b) a constant part on $[x_l, x_u]$, and (c) a concave part on (x_u, ∞). Note that the unimodality condition still holds if either the first two regions, i.e. (a) and (b) or the last two regions, i.e. (b) and (c), are absent.

Contrary, two modes would emerge if there is a *density gap*, i.e. a considerable drop of the density level among two regions of higher density. Specifically, a probability density function that does not exhibit unimodality is referred to as multimodal, indicating the presence of two or more modes. A typical example is a distribution with exactly two modes, which show up as separate peaks (local maxima) in the density plot and is called bimodal. Bimodal distributions frequently occur as combinations of two unimodal distributions, each having a single mode. For instance, combining two Gaussian distributions with identical variances but separated means typically produces a bimodal distribution.

Unimodality is a broad property of probability distributions that simply requires the distribution to have a single "peak" or mode. Because this requirement is very general, unimodal distributions can still take on a wide variety of shapes and densities. For example, distributions of many prominent families, such as the Uniform, Gaussian, Gamma and Beta, have this property. For the multivariate case, several definitions of unimodality have been proposed in the literature which are, however, not equivalent [48, 47].

1.2.2 Unimodality Testing

A reliable approach to assessing the unimodality of the data is through statistical procedures known as unimodality tests, which are designed to detect the presence of more than one mode in a distribution. In other words, these tests evaluate whether a dataset is likely to have been generated by a probability distribution with a single mode (peak). The concept of unimodality is closely linked to the grouping behavior of data points, that is, whether the observations tend to cluster around a single central region or are dispersed across multiple distinct regions.

Several statistical tests have been proposed to assess unimodality [49, 50]. In this thesis, we focus on one of the most prominent methods: *Hartigan's dip-test of unimodality* [50]. A comprehensive treatment of unimodality testing can be found in [51, 52, 53].

Hartigan's dip test [50] is a widely used statistical procedure for assessing the unimodality of a univariate dataset. Given a univariate sample $X = \{x_1, \ldots, x_n\}$, the test examines the underlying Empirical Cumulative Distribution Function (ECDF), defined as

$$F(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{x_i \leq x\}, \tag{1.13}$$

and determines whether the distribution exhibits a single or multiple modes. The dip statistic is computed as the maximum deviation between the ECDF and the closest unimodal distribution function, where "closest" is defined as the distribution that minimizes this maximum difference. Asymptotically, the uniform distribution serves as the least favorable unimodal case, and the distribution of the dip statistic is determined both theoretically (asymptotically) and empirically through sampling from the uniform distribution.

Given a set of real numbers $X = \{x_1, \ldots, x_n\}$, the dip test computes the dip value, denoted as $\text{dip}(X)$, which measures the departure of the empirical cumulative distribution function (ECDF) from unimodality. For two bounded distribution functions F and G, we define their distance as

$$\rho(F, G) = \max_x |F(x) - G(x)|. \tag{1.14}$$

Let $\mathcal{U}$ be the class of all unimodal distribution functions. Then, the dip statistic of a distribution function F is given by

$$\text{dip}(F) = \min_{G \in \mathcal{U}} \rho(F, G). \tag{1.15}$$

In other words, the dip statistic is defined as the smallest among the largest deviations between all CDF F and the CDFs belonging to the class of unimodal distributions. An appealing property of the dip statistic is its consistency: if X is a sample of n observations from F, then $\lim_{n\to\infty} \text{dip}(F_n) = \text{dip}(F)$. For the null hypothesis, the class of uniform distributions $\mathcal{U}$ is typically employed, since their dip values are stochastically larger than those of other unimodal distributions, such as those with exponentially decreasing tails. A further advantage of the dip test is that it is parameter-free.

To provide the method that the dip statistic is computed, we must introduce two key definitions. The *greatest convex minorant* (gcm) of a function F in $(-\infty, \alpha]$ is $\sup G(x)$ for $x \leq \alpha$, where the sup is taken over all functions G that are convex in $(-\infty, \alpha]$ and nowhere greater than F. Similarly, the least concave majorant (lcm) of a function F in $[\alpha, \infty)$ is defined as $\inf L(x)$ for $x \geq \alpha$, where the inf is taken over all functions L that are concave in $[\alpha, \infty)$ and nowhere less than F.

Given a dataset $X = \{x_1, \ldots, x_n\}$, where $x_i \in \mathbb{R}$, the dip statistic is computed as follows:

(i) Begin with $x_L = x_1$, $x_U = x_n$, $D = 0$.

(ii) Compute gcm G and lcm L for F in $[x_L, x_U]$; suppose the points of contact with F are respectively $g_1, \ldots, g_k$ and $l_1, \ldots, l_m$.

(iii) Suppose $d = \sup |G(g_i) - L(g_i)| > \sup |G(l_i) - L(l_i)|$ and that the sup occurs at $l_j \leq g_i \leq l_{j+1}$. Define $x_L^0 = g_i, x_U^0 = l_{j+1}$.

(iv) Suppose $d = \sup |G(l_i) - L(l_i)| > \sup |G(g_i) - L(g_i)|$ and that the sup occurs at $g_i \leq l_j \leq g_{i+1}$. Define $x_L^0 = g_i, x_U^0 = l_j$.

(v) If $d \leq D$, stop and set $D(F) = D$.

(vi) If $d > D$, set $D = \sup\{D, \sup_{x_L \leq x \leq x_L^0} |G(x) - F(x)|, \sup_{x_U^0 \leq x \leq x_U} |L(x) - F(x)|\}$

(vii) Set $x_U = x_U^0, x_L = x_L^0$ and return to (ii).

The dip test conceptually considers all $\frac{n(n-1)}{2}$ possible modal intervals $[x_L, x_U]$ defined by the n sorted observations. For each candidate interval, it constructs in $\mathcal{O}(n)$ time the corresponding gcm over $(\min X, x_L)$ and lcm over $(x_U, \max X)$. Fortunately, for a given sample X, the overall computation of the dip statistic requires only $\mathcal{O}(n)$

time. Beyond the dip value itself, the test also reports a measure of statistical significance in the form of a p-value. This p-value is estimated from b bootstrap replicates, each consisting of n independent observations sampled from the Uniform$[0, 1]$ distribution. Denoting the empirical distribution of the r-th replicate as U_n^r, the p-value is given by

$$P \;=\; \frac{1}{b}\sum_{r=1}^{b} \mathbf{1}\{\,\mathrm{dip}(X) \le \mathrm{dip}(U_n^r)\,\}\,, \tag{1.16}$$

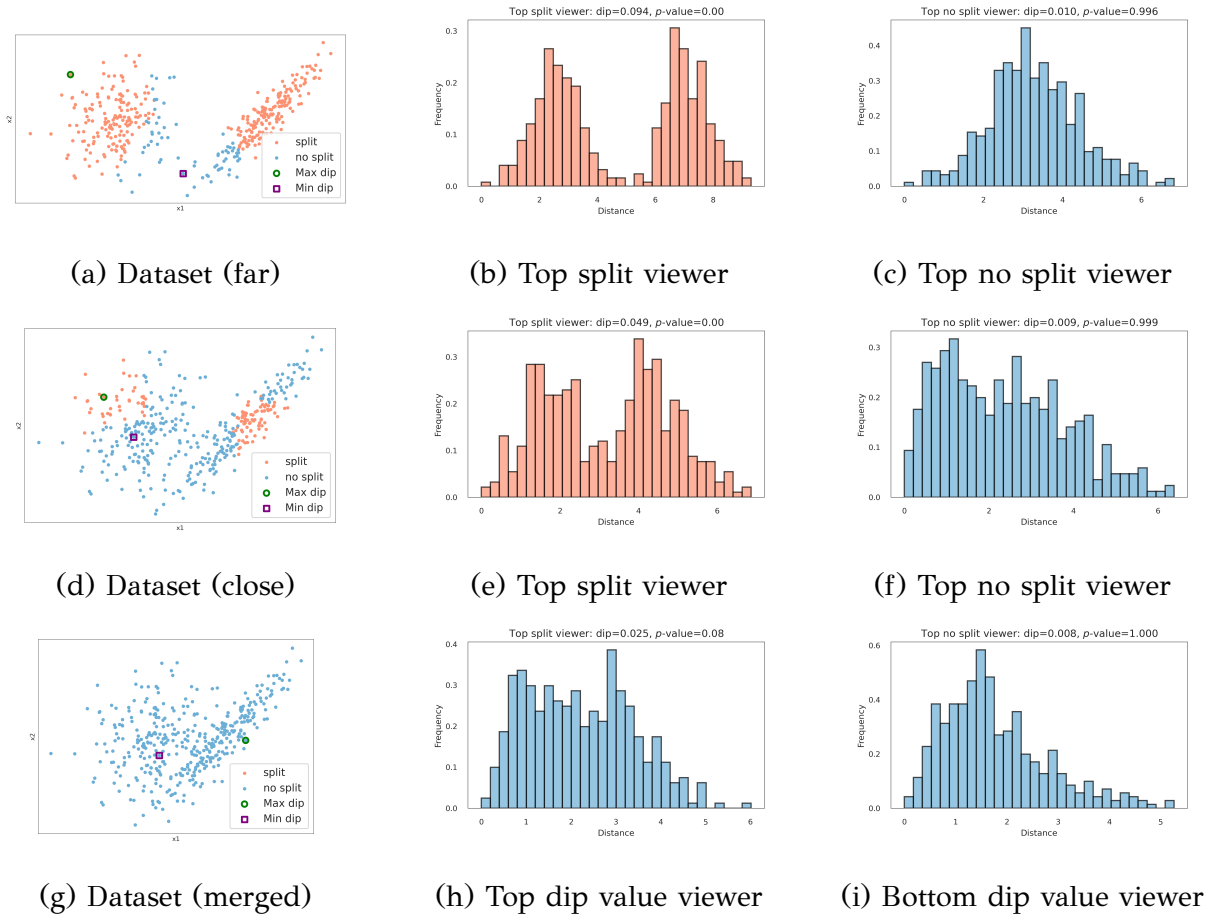
which represents the proportion of bootstrap replicates for which the dip statistic computed for datasets sampled from the uniform null model is greater than or equal to the dip of the dataset X. In other words, it measures how often a unimodal distribution generated under the null hypothesis appears at least as "multimodal" as the data. A small p-value therefore indicates that it is unlikely for the observed $dip(X)$ to be explained by a unimodal distribution, providing evidence against unimodality.

It should be noted that for each sample size n, the bootstrap distributions U_n^r are generated independently of the observed dataset X. Consequently, they need to be computed only once, together with their corresponding dip statistics $\mathrm{dip}(U_n^r)$. The hypotheses tested by the dip test are:

$$H_0 : X \text{ is unimodal}, \qquad H_a : X \text{ is multimodal}. \tag{1.17}$$

At a chosen significance level α, the null hypothesis H_0 is accepted if the computed p-value exceeds α. Otherwise, H_0 is rejected in favor of the alternative hypothesis H_a, indicating evidence of multimodality. An illustrative example of the dip value and the p-value for unimodal and bimodal datasets is shown in Fig. 1.1.

In the *multivariate* setting, the notion of unimodality is far less settled. Some definitions have been proposed in the literature, including *star-unimodality* and *generalized Anderson-unimodality* [47]. However, despite the existence of these mathematical formulations, assessing unimodality in higher dimensions remains challenging, and only a limited number of practical testing procedures have been developed. Given these open questions, we do not pursue the multivariate case further in this thesis.



(a) Dataset (far) (b) Top split viewer (c) Top no split viewer

(d) Dataset (close) (e) Top split viewer (f) Top no split viewer

(g) Dataset (merged) (h) Top dip value viewer (i) Bottom dip value viewer

Figure 1.2: Application of the dip-dist criterion on 2D synthetic datasets with two structures of 200 datapoints each. Split viewers are shown in red. (a) One uniform spherical and one elliptic Gaussian structure. (b), (c) Histograms of pairwise distances for the strongest and weakest split viewers for Dataset (far). (d) As the two structures move closer, the number of split viewers and the dip value decrease. (e), (f) Histograms of pairwise distances for the strongest and weakest split viewers for Dataset (close). (g) The structures are no longer distinguishable from each other. (h), (i) Histograms of pairwise distances for the strongest and weakest split viewers for Dataset (merged).

1.2.3 Unimodality-based clustering

The concept of unimodality and the use of unimodality testing play an important role in several machine learning domains, including clustering, density estimation, and feature selection. In data analysis, uncovering structural information in the data is crucial. For example, clustering is meaningful when a genuine cluster structure exists.

In this context, unimodality tests are particularly relevant [54]. A key notion here is *clusterability* [54], which refers to the extent to which a dataset exhibits inherent cluster structure. Assessment of clusterability should precede the application of clustering algorithms, as their success relies on the presence of such a structure. If clusterability is confirmed, the appropriate clustering algorithm can be selected. Conversely, when no clear structure is present, the clustering results are largely arbitrary and may be misleading, in which case clustering should not be applied.

For illustration, consider a dataset randomly generated from a single Gaussian distribution. Since the data contains only one coherent cluster, further division is meaningless. Nevertheless, most clustering algorithms (e.g. k-means with $k \geq 2$) would still partition the data into multiple clusters, even though no genuine multi-cluster structure exists. A unimodality test applied beforehand could reveal that the dataset forms a single homogeneous cluster, indicating that clustering would not be appropriate. Conversely, suppose that the data come from two Gaussian distributions with widely separated means. The resulting histogram would exhibit two distinct peaks, reflecting the presence of two clusters, and a unimodality test would correctly detect multimodality. Thus, running a unimodality test prior to clustering provides evidence of whether a meaningful cluster structure is present in the dataset. However, in the case of multidimensional data, unimodality testing faces significant challenges. Their performance becomes less predictable in high-dimensional real-world applications, and they often require reducing the data to a single dimension before the test can be applied.

Building on these ideas, top-down (divisive) clustering methods have been developed that explicitly rely on unimodality testing to guide the partitioning process. Such methods begin with the entire dataset treated as a single cluster and iteratively test each existing cluster for unimodality. If the test fails for a cluster, it is split into two subclusters, and the procedure continues until all clusters are decided as unimodal. Notable examples of this approach include the dip-means algorithm [55] and its extension, the projected dip-means algorithm [56], both of which simultaneously perform clustering and estimate the number of clusters. These methods illustrate how unimodality-based criteria can be incorporated directly into clustering frameworks. However, they also face important limitations: (i) testing unimodality for complex, high-dimensional data distributions can be computationally demanding and sensitive to noise, and (ii) clusters with arbitrary or non-convex shapes often cannot be identi-

fied, as they may not conform to unimodality assumptions. Despite these challenges, they remain important methods in the literature, and we present them in detail next.

Dip-means

The *dip-means* algorithm [55] combines Hartigan's dip test with the k-means framework to assess cluster homogeneity through the so-called *dip-dist* criterion. This criterion evaluates unimodality in a set of data points using only pairwise distances (or similarities). The idea is as follows: for a given data point (called a "viewer"), form a vector whose components are the distances from that viewer to all other data points. The distribution of the values in this distance vector can then reveal information about the underlying cluster structure. If the data points belong to a single cluster, the distribution of distances is expected to be unimodal. In contrast, if two well-separated clusters are present, the distribution will typically exhibit two distinct modes, each corresponding to distances to points in one of the clusters. Applying a unimodality test to the distance vector of a viewer therefore provides evidence about the unimodality of the cluster. Figure 1.2 illustrates the application of the dip-dist criterion on synthetic 2d datasets.

A key challenge arises from the dependence of the result on the choice of viewer. Intuitively, viewers located near cluster boundaries are more likely to produce distance vectors with clearly separated modes when multiple clusters exist. To address this issue, dip-means treats each data point as a viewer, applies the unimodality test to every corresponding distance vector, and classifies data points that reject unimodality as "split viewers". If the majority of viewers are identified as split viewers, the algorithm concludes that the cluster contains multiple substructures; otherwise, it is considered unimodal.

Dip-means operates as an incremental clustering algorithm built from three main components. The first is a local search procedure that, given a model with k clusters, optimizes the cluster parameters; this is implemented using k-means, where the cluster centers serve as model representatives. The second and central component applies the dip-dist criterion to determine whether a given subset of data contains evidence of multiple cluster structures. Finally, the third component is a divisive (bisecting) step that when a cluster is deemed multimodal, it splits into two subclusters.

The dip-means method is presented in Algorithm 1.6. Specifically, it requires as input the dataset X along with two parameters for the dip-dist criterion: the

Algorithm 1.6 Dip-means

Require: X (dataset)
Require: k_{init} (initial number of clusters)
Require: α (significance level)
Require: υ_{thd} (percentage of split viewers required for cluster to be considered as a split candidate)
Ensure: $score = \text{unimodalityTest}(c, \alpha, \upsilon_{\text{thd}})$ returns a score value for the cluster c
Ensure: $(C, M) = k\text{-means}(X, k)$ the k-means clustering
Ensure: $(C, M) = k\text{-means}(X, M)$ the k-means clustering initialized with model M
Ensure: (μ_L, μ_R) =splitCluster(c) that splits ta cluster c and returns two centers μ_L, μ_R
$k \leftarrow k_{\text{init}}$
$(C, M) \leftarrow k\text{-means}(X, k)$
while changes in cluster number occur **do**
  for $j = 1, \ldots, k$ **do**
    $score_j \leftarrow \text{unimodalityTest}(c_j, \alpha, \upsilon_{\text{thd}})$ *// compute the score for unimodality test*
  end for
  if $\max_j(score_j) > 0$ **then**
    $\text{target} \leftarrow \arg\max_j(score_j)$ *// index of cluster to be split*
    $(\mu_L, \mu_R) \leftarrow \text{splitCluster}(c_{\text{target}})$
    $M \leftarrow (M - \mu_{\text{target}}, \mu_L, \mu_R)$ *// replace the old centers with the two new ones*
    $(C, M) \leftarrow k\text{-means}(X, M)$ *// refine solution*
  end if
end while
return solution (C, M)

significance level α and a percentage threshold υ_{thd}, which specifies the minimum fraction of cluster members that must be split viewers to trigger a division. The algorithm begins from an initial partition with $k_{\text{init}} \geq 1$ clusters. At each iteration, all current k clusters are tested for unimodality. For each cluster c_j, the set of split viewers υ_j is identified, and the cluster is marked as a split candidate if the proportion of split viewers is greater than the threshold υ_{thd}. Multimodal clusters are assigned a non-zero score, while unimodal clusters receive a score of zero. Several scoring strategies are possible, for example, based on the fraction of split viewers or the cluster size. In the standard formulation, the score of a split candidate c_j is defined as the average dip statistic of its split viewers:

$$score_j = \begin{cases} \frac{1}{|\upsilon_j|} \sum_{x_i \in \upsilon_j} \text{dip}(F^{(x_i)}) & \text{if } \frac{|\upsilon_j|}{|c_j|} \geq \upsilon_{\text{thd}}, \\ 0 & \text{otherwise.} \end{cases} \tag{1.18}$$

To avoid overestimating the true number of clusters, only the candidate with the highest score is split in each iteration. Each split divides a cluster into two subclusters using a 2-means local search, initialized with a pair of sufficiently diverse centers μ_L and μ_R chosen from within the cluster, considering only its data points. The initial

centers are set as $(\mu_L, \mu_R) \leftarrow (x; \mu - (x - \mu))$, where x is a randomly selected cluster member and μ is the cluster center. In this way, μ_L and μ_R are placed symmetrically on opposite sides of μ at equal distances. The 2-means procedure can be repeated with different initializations of μ_L and μ_R to increase the chance of finding a good split. After each iteration, the solution is refined using k-means, which fine-tunes the partition into $k + 1$ clusters. The procedure terminates when no further split candidates are identified among the existing clusters.

Projected dip-means

The *projected dip-means* algorithm [56] (pdip-means) is an alternative to the dip-means algorithm that replaces the dip-dist criterion with a different one, called the projected dip, to assess cluster homogeneity. Like dip-dist, the projected dip relies on the dip test of unimodality, but it applies the test to different one-dimensional datasets. As previously noted, the dip-dist criterion operates on the pairwise distance matrix of the data. This makes it broadly applicable, even in cases where the original data objects are unavailable and only their distance matrix is provided. The pdip-means algorithm considers one-dimensional projections of the data along several axes and applies the dip test for unimodality on the projections.

Specifically, the projected dip (pdip) criterion to assess the homogeneity of a set of data vectors operates as follows: A set of L one-dimensional projections is first specified. For each projection j $(j = 1, \ldots, L)$, the corresponding projected values of the data vectors are computed, forming the one-dimensional set P_j. The dip test is then applied to P_j, yielding dip_j and $p\text{-value}_j$. In analogy to the dip-dist criterion, each projection can be viewed as a "point of view". If multimodality is observed for several projections, the dataset is considered multimodal; otherwise, it is considered unimodal. The pdip-means algorithm supports three types of one-dimensional projections:

1. Projections on each of the d original axes, i.e., applying the dip test to each column of the dataset.

2. PCA-based projections, where Principal Component Analysis is used to extract projections along each principal axis.

3. Random projections applied to randomly selected axes.

Algorithm 1.7 Pdip-means

Require: X (dataset)
Require: k_{init} (initial number of clusters)
Require: α (significance level)
Ensure: $projections$ = data_projection(c) return several data projections of cluster c
Ensure: dip_score = unimodalityTest(d, α) returns a dip score value for the dataset d
Ensure: $(C, M) = k$-means(X, k) the k-means clustering
Ensure: $(C, M) = k$-means(X, M) the k-means clustering initialized with model M
Ensure: (μ_L, μ_R) =splitCluster(c) that splits a cluster c and returns two centers μ_L, μ_R
$k \leftarrow k_{\text{init}}$
$(C, M) \leftarrow k$-means(X, k)
while changes in cluster number occur **do**
  for $j = 1, \ldots, k$ **do**
    $projections_{j,prj} \leftarrow$ data_projection(c_j)
    $dip\_score_{j,prj} \leftarrow$ unimodalityTest($projections_{j,prj}$, α) *// compute the score for unimodality test*
    $pdip\_score_j \leftarrow \max_{prj}(dip\_score_{j,prj})$
  end for
  if $\max_j(pdip\_score_j) > 0$ **then**
    target $\leftarrow \arg\max_j(pdip\_score_j)$ *// index of cluster to be split*
    $(\mu_L, \mu_R) \leftarrow$ splitCluster(c_{target})
    $M \leftarrow (M - \mu_{\text{target}}, \mu_L, \mu_R)$ *// replace the old centers with the two new ones*
    $(C, M) \leftarrow k$-means(X, M) *// refine solution*
  end if
end while
return solution (C, M)

In summary, to assess the homogeneity of a set of real-valued data vectors using the pdip criterion, the dip test is applied $2d + r$ times: once for each of the d columns of the data matrix, once for each of the d PCA projections, and r times for random projections. If at least one dip test indicates multimodality, the dataset is considered multimodal; otherwise, it is considered unimodal. In cases of multimodality, the largest dip value among the projections is taken as the dataset's multimodality score.

The detailed algorithm is presented in Alg. 1.7. Given a set of real-valued data vectors, the projected dip-means (pdip-means) algorithm can be derived from the original dip-means by replacing the dip-dist criterion with the projected dip (pdip) criterion. Thus, pdip-means is an incremental clustering algorithm that begins with a single cluster and iteratively adds new clusters through splitting based on the pdip criterion. More specifically, the pdip criterion is applied to each cluster in the current solution, classifying each cluster as either multimodal or unimodal. For clusters identified as multimodal, the maximum dip value (maxdip) from the dip tests is retained as the cluster's multimodality score. If one or more multimodal clusters exist, the cluster with the highest multimodality score is selected and split into two subclusters.

The number of clusters then increases to $k+1$, and k-means with $k+1$ clusters is applied to refine the partition. These steps are repeated until all clusters are found to be unimodal, at which point the algorithm terminates, since no further splits are warranted. It is important to note that the number of clusters is automatically determined, as also happens with the dip-means algorithm.

1.3 Deep Learning-based Clustering

Deep Learning (DL) has emerged as a powerful approach for extracting rich and meaningful representations from large-scale data without extensive reliance on manually engineered features [13, 57]. A deep neural network (DNN) is a computational model composed of multiple layers of interconnected processing units, or neurons, that learn hierarchical representations of data. Each layer applies a linear transformation followed by a non-linear activation function (e.g., sigmoid, tanh, ReLU), which enables the network to capture complex and highly non-linear relationships in the input. By stacking many such layers, DNNs progressively build more abstract and discriminative feature representations, allowing them to approximate intricate functions and achieve state-of-the-art performance in tasks such as computer vision, natural language processing, and speech recognition. Many DNN architectures incorporate an unsupervised learning stage, commonly referred to as *unsupervised pretraining* (e.g., autoencoders), which enables the model to capture more expressive and informative data representations. This strategy has been shown to significantly enhance the performance of subsequent supervised or unsupervised learning tasks.

Clustering is a well-studied problem with numerous proposed approaches, which can be generally classified as hierarchical (divisive or agglomerative), model-based (e.g. k-means [25], mixture models [5]) and density-based (e.g. DBSCAN [58], DensityPeaks [59]). *However, most of those methods are effective when the data space is low dimensional and not complex, i.e. image data.* Various feature extraction and feature transformation methods have been proposed to map the original complex data to a simpler feature space as a prepossessing step to address those limitations. Some of the methods include Principal Component Analysis [60], Non-negative Matrix Factorization [61], Spectral methods [62], and Minimum Density Hyperplanes [63].

More recently, deep neural networks (DNNs) have been employed for cluster-

ing in the context of deep learning. DNNs are used to learn rich and useful data representations from data collections without heavily relying on human-engineered features [12]. Notably, DNNs can improve the performance of both supervised and unsupervised learning tasks because of their excellent nonlinear mapping capability and flexibility [13, 14]. Although clustering has not initially been the primary goal of deep learning, several clustering methods have been proposed that exploit the representational power of deep neural networks; thus, the *deep clustering* category of methods has emerged. Such methods aim to improve the quality of clustering results by appropriately training neural networks to transform the input data and generate *cluster-friendly* representations, meaning that in the latent space the data will form compact, and hopefully, well-separated clusters [15, 16, 17, 18, 19].

1.3.1 Deep Clustering

In this thesis, we focus on models for unsupervised learning, with particular attention to those employed in deep clustering. Most deep clustering approaches share a common foundation: they first employ deep neural networks (DNNs) to learn meaningful data representations, which are then used as input for the clustering process [15]. A general deep clustering framework can be described through several key components: (1) the neural network architecture, (2) the deep features extracted for clustering, (3) the non-clustering loss, (4) the clustering loss, (5) the strategy for combining these losses, and (6) the mechanism for updating cluster assignments during network training. In the following subsections, we discuss these core concepts.

Neural Network Architectures

In most deep clustering methods, a DNN is employed to transform the input data into a latent representation that serves as the basis for clustering. Various DNN models have been utilized in the deep clustering framework [19]. Notable examples include Generative Adversarial Networks [4], Autoencoders [64], Graph Neural Networks [65] and regular DNNs have been utilized by methods such as ClusterGan [1], VaDE [66] and JULE [67]. A brief overview of these representative DNN models is provided below.

- **Multilayer Perceptron (MLP)**: This type of feedforward network consists of one or more hidden layers of neurons employing non-linear activation functions. In

this architecture, the output of each layer is passed as input to the subsequent layer, establishing a sequential flow of information through the network [68].

- **Autoencoder (AE)**: represent a specialized class of ANNs designed for unsupervised learning. They aim to extract efficient data representations without the need for labeled data. Their primary objective is to compress input data into a compact form while maintaining the ability to accurately reconstruct it. The core architecture of an AE consists of three components: the encoder, the bottleneck layer, and the decoder. The encoder progressively transforms raw input data into a lower-dimensional, informative representation through a series of hidden layers that capture key patterns and features. In the middle the bottleneck layer lies, also referred to as the latent space, which provides a condensed low-dimensional encoding of the input, preserving its essential characteristics. This encoding is then passed to the decoder, which incrementally expands it back to the original dimensionality. The decoder's hidden layers refine this reconstruction step by step, ensuring that the compressed representation is translated into a faithful approximation of the original input. Through this encoding–decoding process, AEs learn meaningful patterns within the data, enabling the discovery of essential features and structures [64]. Several types of AE exist, such as denoising, sparse, and variational AE. In this thesis, we focus on the typical AE model.

- **Convolutional Neural Network (CNN)**: Inspired by biological processes in which the connectivity patterns between neurons resemble the organization of the visual cortex, CNNs are a specialized form of multilayer perceptrons (MLPs) designed to process data with a regular grid structure, such as images. They are particularly effective in achieving invariance to translation, scaling, skewing, and other distortions [69, 13]. GANs are discussed in Section 1.3.3.

- **Generative Adversarial Network (GAN)**: A GAN consists of two competing neural networks, the generator ($\mathcal{G}$) and the discriminator ($\mathcal{D}$) that engage in a zero-sum game. The main goal of GANs is the generation of synthetic samples. Specifically, the objective of the generator is to produce synthetic samples that are indistinguishable from real data, while the discriminator attempts to correctly distinguish between real and generated samples. Through this adversarial process, both networks iteratively improve, ultimately leading the generator to

produce samples that are nearly indistinguishable from real data [4]. GAN's are discussed in detail in Section 1.3.3.

Deep Features

Leveraging the architecture of a DNN, the features used for clustering of the input data can be extracted from either a single layer or multiple layers of the network:

- **Single layer**: Features are obtained from a specific layer of the DNN. This approach is often advantageous due to its lower dimensionality, which simplifies computation and reduces complexity.
- **Multiple layers**: Features are derived from a combination of outputs across several layers. This strategy produces a richer representation, enabling the embedded space to capture more complex semantic information. As a result, it may lead to improved performance in similarity computations [70].

Non-Clustering Loss

The non-clustering loss comes purely from learning a deep representation of the data using DL methods, and it is independent of the clustering part of the procedure. Some possible options for non-clustering losses are [15]:

- **Absence of non-clustering loss**: in this case, the network model is only constrained by the clustering loss. The absence of a non-clustering loss can result in worse representations or even collapsing clusters [71].
- **Reconstruction loss**: if an autoencoder is used as DNN architecture, then the non-clustering loss is the reconstruction loss. Usually, the reconstruction loss is a distance measure $d(x_i, \hat{x}_i)$ between the input x_i to the autoencoder and the corresponding reconstruction $\hat{x}_i$. The most commonly used distance is the Euclidean:

$$\mathcal{L}_{AE}(x_i, \hat{x}_i) = \sum_{i}^{n} ||x_i - \hat{x}_i||^2 \tag{1.19}$$

  where n is the number of data, x_i is the input and $\hat{x}_i$ is the autoencoder output (reconstruction).

- **The min-max loss**: if the DNN model is a GAN architecture [4] then the non-clustering loss function is the following:

$$\mathcal{L}(\mathcal{D},\mathcal{G}) = \min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{x\sim p_{data}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z)))] \tag{1.20}$$

- **Additional data information** can be incorporated to formulate non-clustering losses that promote the extraction of domain-specific and application-relevant features.

Clustering Loss

Clustering loss functions guide neural networks to learn representations that are well-suited for clustering tasks [17, 16, 15]. These functions can generally be divided into two categories: explicit clustering losses, which directly produce cluster assignments after training, and implicit clustering losses, which improve the quality of learned representations but require an additional clustering step to generate the final clusters.

- **Explicit Clustering Loss**: Training with this type of loss enables cluster assignments to be derived directly, as the loss incorporates both the cluster centers and the clustering objective. Examples include k-means loss [72], cluster assignment hardening loss [73], agglomerative clustering loss [74], cluster classification loss [1, 75] and nonparametric maximum margin clustering [76].

- **Implicit Clustering Loss**: This type of loss encourages the network to learn representations suitable for clustering, but it does not itself provide cluster assignments. Thus, an additional clustering stage is required after training. Examples include locality-preserving loss [77], which enforces the preservation of local structures in the embedding space, and group sparsity loss [77], which exploits block-diagonal similarity matrices for representation learning.

The following are representative examples of explicit clustering loss functions [16]:

- **No clustering loss**: Even when a DNN is trained solely with non-clustering losses, the learned deep features can still be utilized for clustering after training. For instance, the network may transform the input data into a lower-dimensional representation, effectively performing dimensionality reduction.

Such representations can sometimes facilitate clustering; however, incorporating a clustering loss generally leads to superior results [73, 71].

- k-**means loss**: Also referred to as the clustering error, this loss encourages the learned representations to be cluster-friendly [71]. By minimizing the clustering error with respect to the DNN parameters, the distance between each data point and its assigned cluster centers is reduced in the latent space, thereby improving the quality of clustering when k-means is applied.

- **Agglomerative clustering loss**: Based on hierarchical clustering principles, this loss iteratively merges the two clusters with the highest similarity (or affinity) in the latent space until a predefined stopping criterion is reached [67].

- **Pseudo-label loss**: Cluster assignments generated during the update process can be treated as pseudo-labels and used as a classification loss within an additional network branch. This approach promotes the extraction of discriminative and meaningful features across network layers [1, 75].

Additionally, several types of implicit clustering loss functions have been proposed, some of them follow [16]:

- **Locality-preserving loss**: this loss target is to ensure the locality of the clusters by pushing nearby data points together [77]. The mathematical formulation is the following:

$$\mathcal{L}_{lp} = \sum_{i} \sum_{j \in N_k(i)} s(x_i, x_j) ||f(x_i) - f(x_j)||^2 \tag{1.21}$$

  where $N_k(i)$ is the set of k nearest neighbors of the data point x_i in the input space, $s(x_i, x_j)$ is a similarity measure between the points x_i and x_j, and $f(\cdot)$ is the nonlinear transformation from the input to the latent space implemented by a DNN.

- **Group sparsity loss**: it is inspired by spectral clustering. In this methodology, the block diagonal similarity matrix is exploited for representation learning [62]. Group sparsity is itself an effective feature selection method. As an example, in [77] the hidden units were divided into G groups, where G is the assumed number of clusters. Given a x_i, the obtained representation has the

form $\{f^g(x_i)\}_{g=1}^G$. Thus the loss can be defined as follows:

$$\mathcal{L}_{gs} = \sum_{i=1}^{N} \sum_{g=1}^{G} \lambda_g ||f^g(x_i)|| \tag{1.22}$$

where $\{\lambda_g\}_{g=1}^G$ are the weights of sparsity groups, defined as

$$\lambda_g = \lambda \sqrt{n_g} \tag{1.23}$$

where n_g is the group size and λ is a constant.

Combining The Non-Clustering and Clustering Losses

Consider a deep clustering procedure where both a clustering loss and a non-clustering loss are employed. An effective strategy for combining these two objectives is essential. The most common approach adopts the following formulation:

$$\mathcal{L}(\theta) = \mathcal{L}_n(\theta) + \lambda \mathcal{L}_c(\theta), \tag{1.24}$$

where $\mathcal{L}_c(\theta)$ denotes the clustering loss, $\mathcal{L}_n(\theta)$ denotes the non-clustering loss, and λ is a non-negative weighting parameter that balances the contribution of the two losses. The parameter λ thus serves as a hyperparameter in DNN training and may either remain fixed or vary according to a predefined schedule. Common strategies for setting or adjusting λ during training include [16]:

- **Joint training**: λ is assigned to a constant value so that both losses contribute simultaneously to the training process.
- **Variable schedule**: λ is dynamically adjusted according to a schedule. For example, training may begin with a low value of λ, which is gradually increased over successive epochs to emphasize the clustering objective more strongly as training progresses.
- **Pre-training and fine-tuning**: Training is carried out in two stages. In the first stage, λ is set to 0, and the model is trained exclusively with the non-clustering loss. In the second stage, λ increases to a positive value, and the model is fine-tuned using only the clustering loss. This approach leverages pre-training to learn general representations, though relying solely on the clustering loss in the second stage may degrade clustering performance.

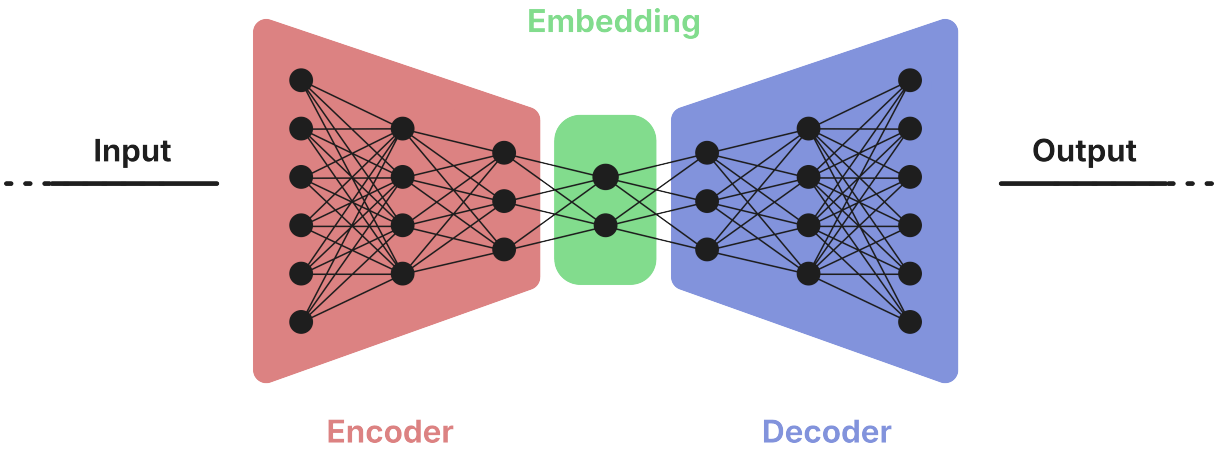


Figure 1.3: General autoencoder architecture. The encoder (shown in red, left) maps the input data to a lower-dimensional embedding space, while the decoder (shown in blue, right) reconstructs the input data from the embeddings.

1.3.2 Autoencoder-based Clustering

The majority of deep clustering methods rely on AE training [19]. AE-based deep clustering methodologies attempt to exploit the non-linear capabilities of the encoder and decoder models in order to assist in latent space [71]. To achieve this goal, novel objective functions have been proposed that integrate the typical AE reconstruction error with a clustering loss in order to train the AE network so that in the learned embedded space, the data will form more compact clusters (achieved through minimization of a clustering objective), while at the same time retaining the information of the original data (achieved by minimizing the AE reconstruction error).

An AE model consists of an encoder network $z = f_\theta(x)$ which given an input x provides embedding z, and the decoder network $\hat{x} = g_w(z)$ that provides the reconstruction $\hat{x}$ given the embedding z, as shown in Fig. 1.3. In the typical AE case, given a dataset $X = \{x_1, \ldots, x_N\}$, the parameters θ and w are adjusted by minimizing the reconstruction loss:

$$\mathcal{L}_{rec} = \frac{1}{N} \sum_{i=1}^{N} ||x_i - g_\theta(f_w(x_i))||^2. \tag{1.25}$$

A straightforward approach for AE-based clustering is to first train the AE using the reconstruction loss and then cluster the embeddings $z_i = f_w(x_i)$ using any clustering method. Thus, data projection and clustering are performed independently. However, it has been found that the better results are obtained if the embeddings are formed taking into account both reconstruction and clustering.

Therefore, *AE clustering framework* has emerged, where the goal is to create cluster-friendly embeddings z_i. To achieve this goal, the reconstruction loss is enhanced with an appropriately defined clustering loss $\mathcal{L}_{cl}$ resulting in a total loss of the form:

$$\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cl}, \tag{1.26}$$

where the hyperparameter λ balances the relative importance of the two objectives.

It should be noted that the minimization of the clustering loss enforces the formation of embeddings z_i with small cluster variance. An easily obtained trivial solution exists, where all data points x_i are mapped to embeddings z_i that are all very close to each other (ie. the encoder is actual a constant function). To avoid this trivial solution, the reconstruction loss is added that forces the embeddings z_i to retain the information of the original dataset. In essence, AE clustering methods strive to create embeddings that form compact clusters while keeping the characteristics of the original dataset. The most widely used AE-based clustering methods are summarized next.

Deep Embedding Clustering

Inspired by the t-SNE [78] algorithm, the *Deep Embedding Clustering* (DEC) [73] method has been proposed that optimizes both the reconstruction objective and a clustering objective. DEC transforms the data in the embedded space using an AE and then optimizes a clustering loss defined by the KL divergence between two distributions p_{ij} and q_{ij}: q_{ij} are soft clustering assignments of the data based on the distances in the embedded space between data points and cluster centers, and p_{ij} is an adjusted target distribution aiming to enhance the clustering quality by leveraging the soft cluster assignments. More specifically, q_{ij} is defined as

$$q_{ij} = \frac{(1 + ||z_i - m_j||^2/\alpha)^{-\frac{a+1}{2}}}{\sum_{j'}(1 + ||z_i - m_{j'}||^2/\alpha)^{-\frac{a+1}{2}}}, \tag{1.27}$$

where $z_i = f_w(x_i)$, m_j is a cluster center in the embedded space and α are the degrees of freedom. Furthermore, $p_{ij} = \frac{q_{ij}^2/f_{ij}}{\sum_{j'} q_{ij}^2/f_{ij}}$ (with $f_{ij} = \sum_i q_{ij}$) is the target probability distribution that aims to sharpen the cluster probability assignments. Finally, DEC performs pretraining to minimize the reconstruction loss and subsequently excludes the decoder part of the network, focusing solely on the clustering loss during the

training phase. The objective function of DEC is the following:

$$\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda \sum_{i=1}^{n} \sum_{j=1}^{k} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{1.28}$$

Improved Deep Embedding Clustering

A modification of the DEC method is the *Improved Deep Clustering with local structure preservation* (IDEC) [79]. Both DEC and IDEC optimize the same objective function 1.28, but differ in the training strategy of the autoencoder. Specifically, IDEC jointly minimizes the reconstruction and clustering losses during training, whereas DEC discards the decoder and focuses solely on the clustering loss. From eq. 1.27 and eq. 1.28 it is clear that the above approaches aim to minimize only an inner cluster distance loss.

Deep Clustering Network

Similar to DEC, the *Deep Clustering Network* (DCN) [71] jointly learns the embeddings and the cluster assignments by directly optimizing the k-means clustering loss in the embedded space. The optimized objective function is:

$$\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda \sum_{i=1}^{n} ||z_i - M s_i||^2, \tag{1.29}$$

where $z_i = f_w(x_i)$, M is a matrix containing the k cluster centers in the embedded space, and s_i is the cluster assignment vector for data point x_i with only one non-zero element. An analogous early work is AE-based data clustering (AEC) [80] which also aims to minimize the distance between embedded data and their nearest cluster centers.

DipEncoder

DipEncoder [81] uses a powerful statistical test in its definition of the clustering loss function, the dip-test of unimodality [82], which introduced in Section 1.2. Note that DipEncoder considers all cluster pairs to define the loss function and defines two terms. Specifically, the first term aims to enforce the unimodal nature of each cluster in the embedded space and is defined as:

$$\mathcal{L}_{uni}(a, b) = \frac{1}{2} \left(dip(\overline{Z}_{\alpha,\beta}) + dip(\overline{Z}_{\not{\alpha},\beta}) \right), \tag{1.30}$$

where α, β are two distinct clusters, $\overline{Z}_{\alpha,\not\beta}$ is the projected latent data values of cluster α onto the line passing through the cluster centers (similarly for $\overline{Z}_{\not\alpha,\beta}$), and $dip(\cdot)$ is the dip-value of the corresponding set. The second term of the clustering loss is responsible for enforcing multimodality of the union of two clusters α, β and is defined as:

$$\mathcal{L}_{multi}(a, b) = -dip(\overline{Z}_{\alpha,\beta}). \tag{1.31}$$

The complete form of the DipEncoder loss function is the following:

$$\mathcal{L}_{dip} = \frac{2}{k(k-1)} \sum_{\alpha=1}^{k-1} \sum_{\beta=a+1}^{k} \mathcal{L}_{uni}(a, b) + \mathcal{L}_{multi}(a, b), \tag{1.32}$$

where k is the number of clusters.

1.3.3 GANs-based Clustering

As mentioned previously, GANs [4] are based on a min–max optimization framework in which two networks compete in a zero-sum game. The generator ($\mathcal{G}$) aims to produce synthetic data, while the discriminator ($\mathcal{D}$) seeks to distinguish between real and generated samples. GANs have achieved remarkable success in a wide range of unsupervised learning tasks, and clustering, as a central unsupervised problem, naturally benefits from their capabilities. In particular, the latent space learned by GANs not only facilitates dimensionality reduction but also enables several novel applications. For example, perturbations in the latent space can be used to generate adversarial examples, which contribute to building more robust classifiers [83]. Similarly, compressed sensing approaches [84] leverage GANs by identifying latent vectors that minimize reconstruction error, while generative compression techniques [85] utilize the latent space for efficient data compression. These properties make GANs particularly well suited as a backbone for deep clustering methods. In the following, we first provide an overview of the general framework of GANs and then describe ClusterGAN [1], a methodology that extends the GAN architecture by incorporating an additional encoder network ($\mathcal{E}$).

GANs

GANs [4] aim to approximate the real data distribution by implicit sampling while simultaneously learning a mapping from a latent space $\mathcal{Z}$ to the input space $\mathcal{X}$. To

achieve this, the generator produces synthetic samples defined as $x = \mathcal{G}(z; \theta_g)$, where z is typically drawn from a prior distribution $z \sim \mathcal{N}(0, \sigma^2 I)$. The discriminator $\mathcal{D}$, acting as an adversary, attempts to distinguish real samples from those generated by $\mathcal{G}$, providing $\mathcal{D}(x; \theta_d)$, which represents the probability that x is real. Specifically, the generator's objective is to maximize the error of the discriminator, while the discriminator strives to minimize it. Training proceeds iteratively until the discriminator can no longer reliably differentiate between real and generated samples. Formally, the objective of GANs is expressed as:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))] \tag{1.33}$$

where p_{data} represents the distribution of real data, and p_z is the prior probability distribution of noise (usually a Gaussian distribution). Here, x and z are samples from $\mathcal{X}$ and $\mathcal{Z}$, respectively. The discriminator $\mathcal{D}$ outputs a real number in $[0, 1]$ indicating whether a sample is real $(\mathcal{D}(x) \to 1)$ or fake $(\mathcal{D}(x) \to 0)$ while the generator $\mathcal{G}$ produces synthetic samples intended to resemble real data as closely as possible.

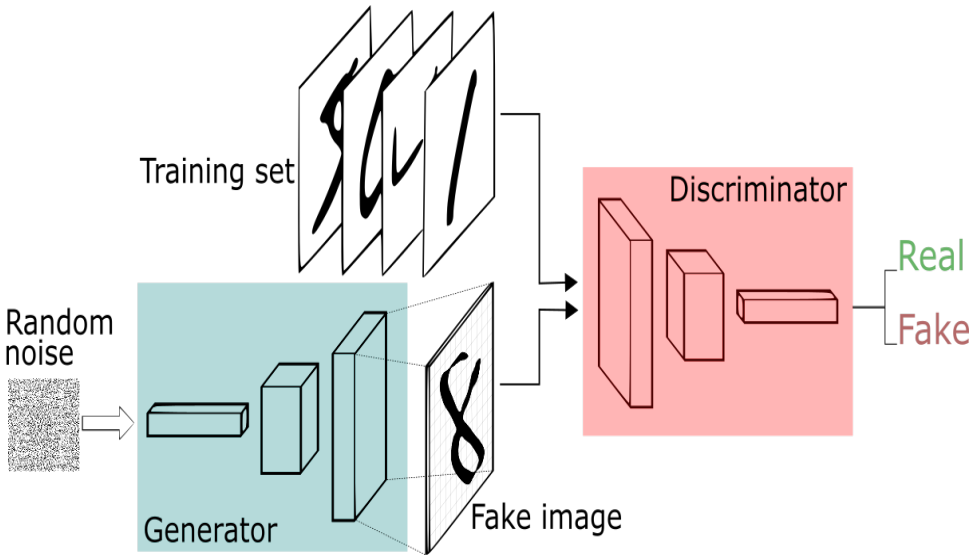


Figure 1.4: Basic GAN architecture and operation.

Figure 1.4 shows the architecture and operation of a typical GAN. Initially, the generator network receives a random noise vector as input and produces a synthetic sample as output. Both real samples from the dataset and generated samples are then fed into the discriminator network, which evaluates their authenticity. Finally, the discriminator outputs a value in the range $[0, 1]$, representing the probability that the input sample is real $(\mathcal{D}(x) \to 1)$ or fake $(\mathcal{D}(x) \to 0)$.

The complete training procedure for GANs is outlined in Alg. 1.8. For the gradient-based updates, any standard optimization method may be employed, with the Adam optimizer [86] being the most widely used choice.

Algorithm 1.8 Minibatch stochastic gradient descent training of GANs [4].

Require: X (dataset)

Require: $T, ITERATIONS$ (number of discriminator updates, number of iterations)

Require: $\mathcal{G}, \mathcal{D}$ (generator, discriminator)

$iteration \leftarrow 1$

while $iteration \leq ITERATIONS$ **do**

  $t \leftarrow 1$

  while $t < T$ **do**

    Sample minibatch of m noise samples $\{z_1, \ldots, z_m\}$ from noise prior $p_z(z)$.

    Sample minibatch of m examples $\{x_1, \ldots, x_m\}$ from data generating distribution $p_{data}(x)$.

    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log \mathcal{D}(x_i) + \log(1 - \mathcal{D}(\mathcal{G}(z_i))) \right]. \tag{1.34}$$

    $t \leftarrow t + 1$

  end while

  Sample minibatch of m noise samples $\{z_1, \ldots, z_m\}$ from noise prior $p_z(z)$.

  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - \mathcal{D}(\mathcal{G}(z_i))). \tag{1.35}$$

  $iteration \leftarrow iteration + 1$

end while

return The learned parameters θ_g, θ_d.

ClusterGan

ClusterGAN [1] presents an adaptation of the GAN framework [4] to address the task of data clustering. Its design integrates several key algorithmic ideas which can be summarized as follows:

- **Sampling prior**: ClusterGAN employs a latent prior that combines discrete and continuous variables, enabling the formation of clusters more naturally within the latent space.

- **Architecture**: In addition to the generator and discriminator networks, ClusterGAN introduces an encoder network dedicated to clustering. The encoder provides an inverse mapping from the data space $\mathcal{X}$ to the latent space $\mathcal{Z}$, i.e., $\mathcal{E} : \mathcal{X} \to \mathcal{Z}$.

- **Loss function**: Training jointly involves both the GAN and the encoder, guided by a clustering-specific loss. This ensures that the geometry of the projected latent space reflects the distance relationships among the latent variables z.

The first step in adapting the vanilla GAN [4] to enhance clustering performance in the latent space involves designing a more effective *sampling prior distribution* for the generator. ClusterGAN [1] achieves this by employing a latent prior composed of both discrete and continuous variables. This choice ensures that the resulting clusters remain well separated. Specifically, the latent variable is defined as $z = (z_n, z_c)$, where z_n is drawn from a multivariate Gaussian distribution ($z_n \sim \mathcal{N}(0, \sigma^2 I)$) and z_c is sampled from a one-hot distribution with K elements, corresponding to the number of clusters. To ensure that each mode generates samples exclusively from its corresponding class in the original data, the standard deviation σ must be chosen carefully. A small value, such as $\sigma = 0.10$, is typically used so that, with high probability, each dimension of the normal component satisfies $z_{n,j} \in (-0.6, 0.6) \ll 1.0$ for all j. Setting σ to such a small value is crucial, as it prevents overlap among the clusters in the latent space $\mathcal{Z}$ of the generator, thereby maintaining their separation.

Algorithm 1.9 Minibatch stochastic gradient descent training of ClusterGan [1].

Require: X (dataset)

Require: $T, ITERATIONS$ (number of discriminator updates, number of iterations)

Require: $\mathcal{G}, \mathcal{D}, \mathcal{E}$ (generator, discriminator, encoder)

$iteration \leftarrow 1$

while $iteration < ITERATIONS$ **do**

  $t \leftarrow 1$

  while $t < T$ **do**

    Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $\mathbb{P}_z$, where $z^{(i)} = (z_n^{(i)}, z_c^{(i)})$.

    Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $\mathbb{P}_x$.

    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_{\mathcal{D}}} \frac{1}{m} \sum_{i=1}^{m} \left[q(\mathcal{D}(x^{(i)})) + q(1 - \mathcal{D}(\mathcal{G}(z^{(i)}))) \right]. \tag{1.36}$$

    $t \leftarrow t + 1$

  end while

  Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $\mathbb{P}_z$, where $z^{(i)} = (z_n^{(i)}, z_c^{(i)})$.

  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_{\mathcal{G}}} \frac{1}{m} \left(-\sum_{i=1}^{m} q(\mathcal{D}(\mathcal{G}(z^{(i)}))) + \beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}(\mathcal{G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}(\mathcal{G}(z_c^{(i)}))) \right). \tag{1.37}$$

  Update the encoder by descending its stochastic gradient:

$$\nabla_{\theta_{\mathcal{E}}} \frac{1}{m} \left(\beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}(\mathcal{G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}(\mathcal{G}(z_c^{(i)}))) \right). \tag{1.38}$$

  $iteration \leftarrow iteration + 1$

end while

ClusterGAN extends the standard GAN architecture by incorporating an additional network called *Encoder*. Unlike the generator and discriminator, which are part of the vanilla GAN, the encoder is introduced specifically to enable clustering. Its primary role is to explicitly perform an inverse mapping from the data space of the generated samples back to the latent space, i.e., $\mathcal{E} : \mathcal{X}_g \rightarrow \hat{z}$, where $\hat{z}$ represents the estimated latent variables corresponding to a given data sample. This task is inherently challenging because the inverse mapping problem involves navigating a highly non-convex search space, since the generator is a neural network whose mappings

in the latent space $\mathcal{Z}$ can vary depending on initialization. By explicitly learning this inverse mapping, the encoder addresses this challenge efficiently, enabling the model to recover latent variables in a consistent manner. As a result, the encoder facilitates the clustering of samples in an unsupervised classification framework.

Figure 1.5 illustrates the architecture and basic operation of ClusterGAN. The generator takes as input the latent variable $z = (z_n, z_c)$, where z_c encodes the clustering information, and produces synthetic samples x_g. The discriminator receives both real samples (x_r) and generated samples (x_g), outputting the probability that each sample is real. At the same time, the encoder processes the generated samples and projects them back into the latent space, thereby recovering the latent representation.

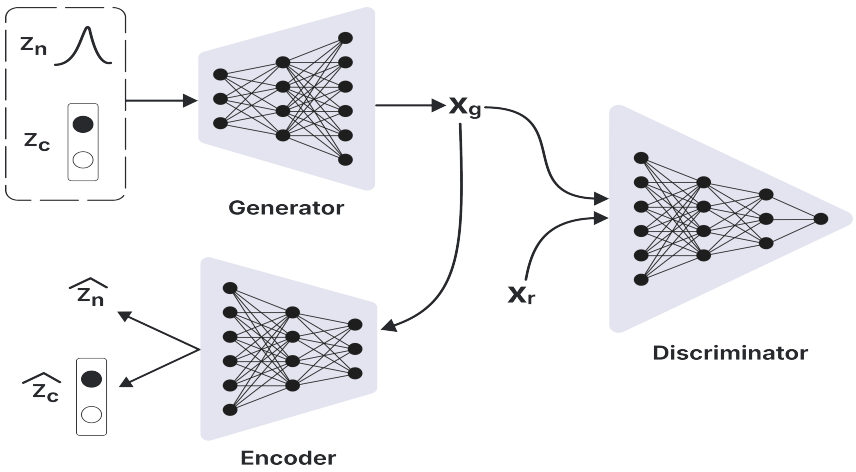


Figure 1.5: ClusterGan Architecture [1].

The ClusterGan's loss function is a combination of losses that contains the GANs' loss that targets the data generation and the clustering loss that aims at the data clustering:

$$\begin{aligned} &\min_{\theta_{\mathcal{G}},\theta_{\mathcal{E}}} \max_{\theta_{\mathcal{D}}} \mathop{\mathbb{E}}_{x\sim\mathbb{P}_x^r} q(\mathcal{D}(x)) + \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} q(1-\mathcal{D}(\mathcal{G}(z)))+ \\ &\beta_n \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} ||z_n - \mathcal{E}(\mathcal{G}(z_n))||_2^2 + \beta_c \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \mathcal{H}(z_c, \mathcal{E}(\mathcal{G}(z_c))) \end{aligned} \tag{1.39}$$

where $\mathcal{H}(.,.)$ is the cross-entropy loss, which is defined as $\mathcal{H}(X) = -\sum_x p(x)\log(p(x))$. The relative magnitudes of the regularization coefficients β_n and β_c enable a flexible choice to vary the importance of preserving the discrete and continuous portions of the latent code, where $q(.)$ is the quality function, given as $q(x) = log(x)$ for vanilla GAN [4], and $q(x) = x$ for Wasserstein GAN (WGAN) [87].

Specifically, the first two terms of the loss function (eq. 1.39) constitute the standard GAN objective, while the last two terms introduce the clustering loss. In particular, $\beta_n \underset{z\sim\mathbb{P}_z}{\mathbb{E}} ||z_n - \mathcal{E}(\mathcal{G}(z_n))||_2^2$, acts as a regularization of the continuous portion z_n of the latent space, which ensures the reversibility of the continuous portion of the latent space, i.e., $\mathcal{E} : \mathcal{G}(z_n) \rightarrow z_n$. Additionally, the term $\beta_c \underset{z\sim\mathbb{P}_z}{\mathbb{E}} \mathcal{H}(z_c, \mathcal{E}(\mathcal{G}(z_c)))$, penalizes the discrete portion of the latent code in order to guide the mappings of points belonging to the same class in $\mathcal{X}$ space to correspond to the same one-hot encoding when embedded in $\mathcal{Z}$ space. This regularization is the most important part to succeed in data clustering in the latent space. The complete training algorithm of ClusterGan is presented in Alg. 1.9.

1.4 Thesis Contribution

In this thesis, we investigate the clustering problem with emphasis on three main axes: (i) partitional clustering in both Euclidean and kernel spaces, (ii) unimodality-based clustering, and (iii) deep clustering, which leverages the representational power of deep learning methods. These research directions are not entirely independent, as they share certain conceptual overlaps. For instance, kernel-based clustering primarily aims to identify non-linearly separable clusters, a goal that deep clustering also pursues through the learning of richer, non-linear low-dimensional representations. The key distinction lies in the methodology; specifically in kernel-based clustering, the kernel function is predefined, whereas in deep clustering, the data representation is jointly learned during the training process. In the following, we summarize the main contributions of this thesis.

In Chapter 2, we introduce global k-means++ [88], designed to address the cluster initialization problem inherent in the standard k-means algorithm. The proposed method integrates the incremental strategy of global k-means with the probabilistic center selection mechanism of k-means++, thereby combining the strengths of both approaches. This synergy enables high-quality clustering results while substantially reducing the computational overhead typically associated with global k-means. Unlike previous efforts that focused on enhancing the fast global k-means algorithm, which provides a computationally cheaper approximation, our method directly improves the original global k-means formulation. We demonstrate that global k-means++ consti-

tutes a compelling alternative to both global k-means and k-means++, offering clustering solutions for all $k \in 1, \ldots, K$. This feature makes it particularly well-suited for model selection tasks where the number of clusters is unknown. In such scenarios, our approach exhibits notable efficiency, surpassing even non-incremental methods such as standard k-means and k-means++.

In a nutshell, we make the following contributions:

- We propose global k-means++, a novel clustering algorithm that combines the incremental strategy of global k-means with the probabilistic center selection mechanism of k-means++.
- We incorporate a hyperparameter that balances exploration and exploitation, ensuring more effective solutions to each clustering subproblem.
- We directly enhance the original global k-means formulation, in contrast to prior work that focused on approximations such as fast global k-means.
- We provide complete clustering solutions for all $k \in \{1, \ldots, K\}$, making our method particularly suitable for model selection tasks where the number of clusters is unknown.
- We demonstrate through extensive experiments that global k-means++ achieves clustering quality comparable to global k-means while substantially reducing computational cost.

In Chapter 3, we extend the concept of global k-means++ from Euclidean to kernel space. Specifically, we introduce global kernel k-means++ [44], an algorithm designed to address the initialization problem inherent in kernel k-means. Similarly to the global variants of k-means, our method incrementally solves all intermediate clustering subproblems for $k = 1, \ldots, K-1$, ultimately yielding the solution for K clusters. The algorithm achieves an effective trade-off between the strengths of global kernel k-means and kernel k-means++, delivering high-quality clustering at a much lower computational cost.

In a nutshell, we make the following contributions:

- We propose global kernel k-means++, an incremental clustering algorithm that conveys the concept of global k-means++ into the kernel space.

- We address the initialization sensitivity of kernel k-means by combining an incremental strategy with probabilistic center initialization.
- We demonstrate through experiments that the global kernel k-means++ achieves high clustering accuracy while maintaining low computational cost.

In Chapter 4, we present the Unimodality Forest method for Clustering and number of clusters Estimation (UniForCE) [89], which simultaneously performs clustering and estimates the number of clusters k. Our approach is grounded in a novel definition of *locally unimodal cluster*. Instead of requiring unimodality to hold over the entire cluster density, we study unimodality at a local level, within subregions of the cluster density. The method is motivated by the observation that unimodality often emerges when examining the union of neighboring subclusters. These unimodal pairs serve as the basis for aggregating smaller subclusters into larger, statistically coherent cluster structures in a bottom-up manner. A locally unimodal cluster, therefore, spans connected subregions of the data density that are linked through unimodal pairs in a single connected component of a unimodality graph. This formulation is flexible, allowing the discovery of arbitrarily shaped clusters, including both classical unimodal and convex structures. We also introduce a statistical test to determine unimodal pairs of subclusters and construct the *unimodality graph*, in which both clustering and estimation of k are performed by the computation of the unimodality spanning forest. We validate both the conceptual and algorithmic strengths of UniForCE through extensive experiments on synthetic and real-world datasets.

In a nutshell, we make the following contributions:

- We propose UniForCE, a clustering method that simultaneously clusters data and estimates the number of clusters k.
- We introduce a novel and flexible definition of a locally unimodal cluster, based on local unimodality in subregions of the data density.
- We design and implement a statistical procedure to test unimodality between high-dimentional subcluster pairs, ensuring that cluster formation is principled and data driven.
- We develop an online variant of Kruskal's algorithm to minimize the number of statistical tests required, improving efficiency without sacrificing clustering accuracy.

- We formalize the concept of a unimodality graph and show how clustering and number of clusters estimation can be performed by computing a unimodality spanning forest.
- We demonstrate that UniForCE can identify complex, non-convex, and arbitrary-shaped clusters without the need for hard-to-tune hyperparameters, offering a practical and robust solution.
- We empirically validate the effectiveness of UniForCE in both synthetic and real datasets, confirming its ability to deliver accurate and scalable clustering results.

In Chapter 5, we introduce the soft silhouette score [90], a generalization of the widely used silhouette measure that accommodates probabilistic cluster assignments. Building on this differentiable measure, we develop Deep Clustering with the Soft Silhouette Score (DCSS), an autoencoder-based framework specifically designed to optimize this objective [90]. Our method guides the learned latent representations to form clusters that are both compact and well-separated. This property is crucial in real-world applications, as simultaneously ensuring compactness and separability guarantees that clusters are not only densely packed but also clearly distinct from each other. We evaluate DCSS on a variety of benchmark datasets and against state-of-the-art methods to demonstrate that it outperforms established deep clustering approaches, highlighting the effectiveness of the soft silhouette score as a principled objective for improving the quality of learned latent representations.

In a nutshell, we make the following contributions:

- We propose the soft silhouette score, a probabilistic and differentiable generalization of the traditional silhouette measure.
- We develop DCSS, an autoencoder-based deep clustering framework that directly optimizes the soft silhouette score.
- We demonstrate that our method encourages latent representations to form both compact and well-separated clusters, a key property for reliable clustering.
- We showcase through extensive experiments on benchmark datasets that DCSS outperforms established deep clustering approaches, underscoring the practical value of the soft silhouette score as a natural clustering objective.

In Chapter 6, we present the Neural Implicit Maximum Likelihood Clustering (NIMLC) [91], a neural-network-based approach that frames clustering as a generative task within the Implicit Maximum Likelihood Estimation (IMLE) framework. By adapting ideas from ClusterGAN, NIMLC avoids several well-known shortcomings of GAN-based clustering while maintaining a simple and stable training objective. The method performs particularly well on small datasets, with experimental comparisons against both deep and conventional clustering algorithms underscoring its competitive potential. A notable strength of NIMLC is its ability to capture diverse cluster geometries without requiring hyperparameter tuning. Experiments on synthetic datasets show that under the same settings, NIMLC can successfully cluster both cloud-shaped and ring-shaped data. Taken together, these results highlight that incorporating generative modeling into IMLE provides a robust and versatile foundation for neural network–based clustering.

In a nutshell, we make the following contributions:

- We propose NIMLC, a novel clustering method that frames clustering as a generative modeling task within the Implicit Maximum Likelihood Estimation (IMLE) framework.
- We adapt ideas from ClusterGAN to design a training procedure that avoids the known instabilities and deficiencies of GAN-based clustering while retaining a simple and stable objective.
- We show that NIMLC performs particularly well on small datasets, where many deep clustering methods often struggle.
- We demonstrate that NIMLC can discover diverse cluster structures without the need for hyperparameter tuning, ensuring robustness and ease of use.
- We validate NIMLC through experiments on both synthetic and real datasets, showing its ability to effectively cluster cloud-shaped, ring-shaped, and other complex data structures.

1.5 Thesis Layout

The rest of this thesis is organized as follows. In Chapter 2, we propose global k-means++ [88], an effective relaxation of the global k-means algorithm that alleviates the initialization sensitivity of classical k-means while producing solutions comparable to global k-means, yet without its prohibitive computational demands. In Chapter 3, we extend this idea to kernel space and introduce global kernel k-means++ [44], an efficient error minimization method that leverages kernel functions to capture complex nonlinear cluster structures. In Chapter 4, we present UniForCE [89], a novel unimodality-based clustering framework that introduces the new concept of locally unimodal clusters. UniForCE not only detects complex cluster structures, but also automatically estimates the number of clusters, requires no difficult hyperparameter tuning, and achieves these results while maintaining statistical rigor and scalability. In Chapter 5, we introduce the soft silhouette score [90], a probabilistic and differentiable generalization of the traditional silhouette measure. Building on this formulation, we develop an autoencoder-based deep clustering optimization procedure that maximizes the soft silhouette score, thereby guiding the learning of latent representations that produce compact and well-separated clusters. In Chapter 6, we introduce NIMLC [91], a neural network–based clustering framework that utilizes implicit maximum likelihood estimation and delivers promising clustering results. Finally, Chapter 7 provides concluding remarks, summarizing the main contributions of this work and outlining directions for future research.

Chapter 2

The Global k-means++ Algorithm

2.1 Introduction

The k-means algorithm is highly sensitive to the initialization of cluster centers. As discussed in Chapter 1, Section 1.1, several variants of k-means have been proposed to address this issue. Among them, global k-means stands out for its strong optimization capabilities. However, as previously noted, its main limitation lies in its high computational cost.

In this Chapter [88], we propose the *global k-means++* clustering algorithm, which is an effective way of acquiring clustering solutions of comparable clustering errors to those that global k-means produces without its high computational cost. This is achieved by employing the effective center selection probability distribution of the k-means++ method. The global k-means++ algorithm is an attempt to *retain* the effectiveness of the incremental clustering strategy of the global k-means while *reducing* its computational demand using the efficient stochastic center initialization of the k-means++ algorithm.

The proposed algorithm proceeds incrementally by solving all intermediate sub-problems with $k \in \{1, 2, \ldots, K-1\}$ to provide the clustering solution for K clusters. The underlying idea of the proposed method is that an optimal solution for a clustering problem with K clusters can be obtained using a series of local k-means searches that are appropriately initialized. More specifically, at each local search, the $k-1$ cluster centers are always initially placed at their optimal positions corresponding to the clustering problem with $k-1$ clusters. However, the remaining k_{th} cluster center is placed at several starting positions within the data space that are randomly selected by sampling from the k-means++ probability distribution.

To evaluate the effectiveness of the proposed approach, an extensive series of experiments was conducted on a variety of benchmark datasets. The objective was to assess the clustering quality, robustness, and scalability of the method in comparison to well-established algorithms. Specifically, we benchmarked our approach against classical clustering techniques such as the standard k-means algorithm, k-means++, fast global k-means, and global k-means. These comparisons allowed us to highlight the strengths and limitations of each algorithm under different conditions, such as varying numbers of clusters, dataset complexity, number of data points, and dimensions. Quantitative evaluations were complemented with visualizations to provide further insight into the structure of the resulting clusters.

The rest of this Chapter is organized as follows. In Section 2.2, we present the proposed global k-means++ clustering algorithm. In Section 2.3, we provide extensive comparative experimental results. In Section 2.4 we discuss some key findings derived from the experimental results. Finally, in Section 2.5 summarizes the Chapter.

2.2 Global k-means++

Global k-means is a deterministic algorithm proposed to tackle the random initialization problem but it is computationally expensive. It partitions the data to K clusters by solving all k cluster sub-problems sequentially $k \in \{1, \ldots, K\}$. In order to solve the k clustering sub-problem, the method employs a strategy wherein the k-means algorithm is executed N times. Each algorithmic cycle considers all N data points as potential candidate positions for the new cluster center. This comprehensive exploration of candidate positions aims to identify the optimal center placement for the

new cluster center and drastically improves the clustering performance.

An approach to reduce the complexity of the global k-means algorithm is to consider a set of L data points as candidate positions for the new cluster center, where $L \ll N$. However, an effective candidate center selection strategy is required. Employing a random uniform selection method for the L candidates is not expected to be a viable choice. An ideal candidate selection method should meet the following requirements:

Efficiency requirements

i. Low computational complexity.

ii. Low space complexity.

Effectiveness requirement

i. Selection of a set of high-quality center candidates.

We consider as "high-quality" a set of centers assigned to separate data regions. In order to select such center candidates, prioritizing sampling from regions without cluster centers is crucial. Such strategy enhances the likelihood of choosing candidates that belong to different clusters while minimizing the risk of selecting redundant or sub-optimal center positions. For this reason, we have employed the k-means++ probability distribution as the candidate selection method. The effectiveness of this distribution in the initial placement of cluster centers makes it an excellent choice for addressing the aforementioned requirements.

In this Chapter, we propose the *global k-means++* algorithm, an effective relaxation of the global k-means method. Its main difference with the global k-means is that the proposed method requires only L executions of k-means for each k cluster sub-problem, where $L \ll N$. The primary goal of our proposed clustering method is to provide high-quality results that are comparable to those of the global k-means algorithm, while retaining low computational and space complexity. The complete global k-means++ method is presented in Alg. 2.1. As with any other global k-means variant, in order to reduce the computational requirements, we sacrifice determinism by using an effective stochastic initial center selection procedure.

The algorithm requires as input the dataset X, the number of clusters K, the number of candidates L to consider, as well as the sampling method S. Two possible

Algorithm 2.1 `Global` k`-means++`

Require: $X = \{x_1, \ldots, x_N\}$: Dataset
Require: K: Number of clusters
Require: L: Number of candidates
Require: $S \in \{\texttt{Batch}, \texttt{Sequential}\}$: Sampling method
1: $\mu_1 \leftarrow \frac{1}{|X|} \sum_{x_i \in X} x_i; \quad M_1 \leftarrow \{\mu_1\}$ *// Optimal initialization for $k = 1$*
2: $D \leftarrow \left(||x_1 - \mu_1||^2, \ldots, ||x_N - \mu_1||^2\right)$ *// Distance vector*
3: **for** $k = 2, \ldots, K$ **do**
4: **if** $S = \texttt{Batch}$ **then**
5: $\{c_1, \ldots, c_L\} \leftarrow$ `Batch Sampling`(X, M_{k-1}, L, D)
6: **else**
7: $\{c_1, \ldots, c_L\} \leftarrow$ `Sequential Sampling`(X, M_{k-1}, L, D)
8: **end if**
9: **for all** $c_\ell \in \{c_1, \ldots, c_L\}$ **do**
10: $\{(C_k^{(\ell)}, M_k^{(\ell)}, E(C_k^{(\ell)}))\} \leftarrow$ Run k-means with initial k centers positions $M_{k-1} \cup \{c_\ell\}$
11: **end for**
12: $(C_k, M_k) \leftarrow$ Solution with the minimum error $E(C_k^{\star(\ell)})$ among the L solutions $\{(C_k^{(\ell)}, M_k^{(\ell)}, E(C_k^{(\ell)}))\}$, $\ell = 1, \ldots, L$
13: **for all** $x_i \in X$ **do**
14: $D[i] \leftarrow \min_{\mu_j \in M_k} ||x_i - \mu_j||^2$ *// Pre-computed distances in step 12*
15: **end for**
16: **end for**
17: **return** solutions (C_k, M_k) for every $k \in \{1, \ldots, K\}$

Algorithm 2.2 `Batch Sampling`

Require: $X = \{x_1, \ldots, x_N\}$: Dataset
Require: M: Set of cluster centers
Require: L: Number of candidates
Require: D: Distance vector
1: Compute the probability vector $P = (p_1, \ldots, p_N)$, where $p_i = Pr(m_k = x_i) = d_i \Big/ \sum_{j=1}^{N} d_j$
2: $\{c_1, \ldots, c_L\} \leftarrow$ Sample without replacement L center candidates from dataset X by k-means++ probability vector P
3: **return** Set of initial center candidates $\{c_1, \ldots, c_L\}$

sampling strategies are considered, namely, *batch sampling* and *sequential sampling*. It should be noted that $D = (d_1, \ldots, d_N)$ represents the distance vector, where d_i is the distance of x_i to its closest center m_j (eq. 1.5).

Specifically, to solve a clustering problem with K clusters, the method proceeds as follows. Initially, it addresses the 1-means sub-problem by setting the center μ_1 as the mean of the entire dataset X and it initializes the distance vector D (steps 1-2 in Alg. 2.1). Then, to solve the 2-means sub-problem, it utilizes the 1-means solution as the first initial center position (obtained in step 1). To initialize the second center, a set of L candidate positions $\{c_1, \ldots, c_L\}$ is sampled using batch or sequential sampling (steps 4-8 in Alg. 2.1). The k-means algorithm is executed L times until convergence, one for each candidate position c_ℓ (steps 9-11 in Alg. 2.1), and the solution with

Algorithm 2.3 `Sequential Sampling`

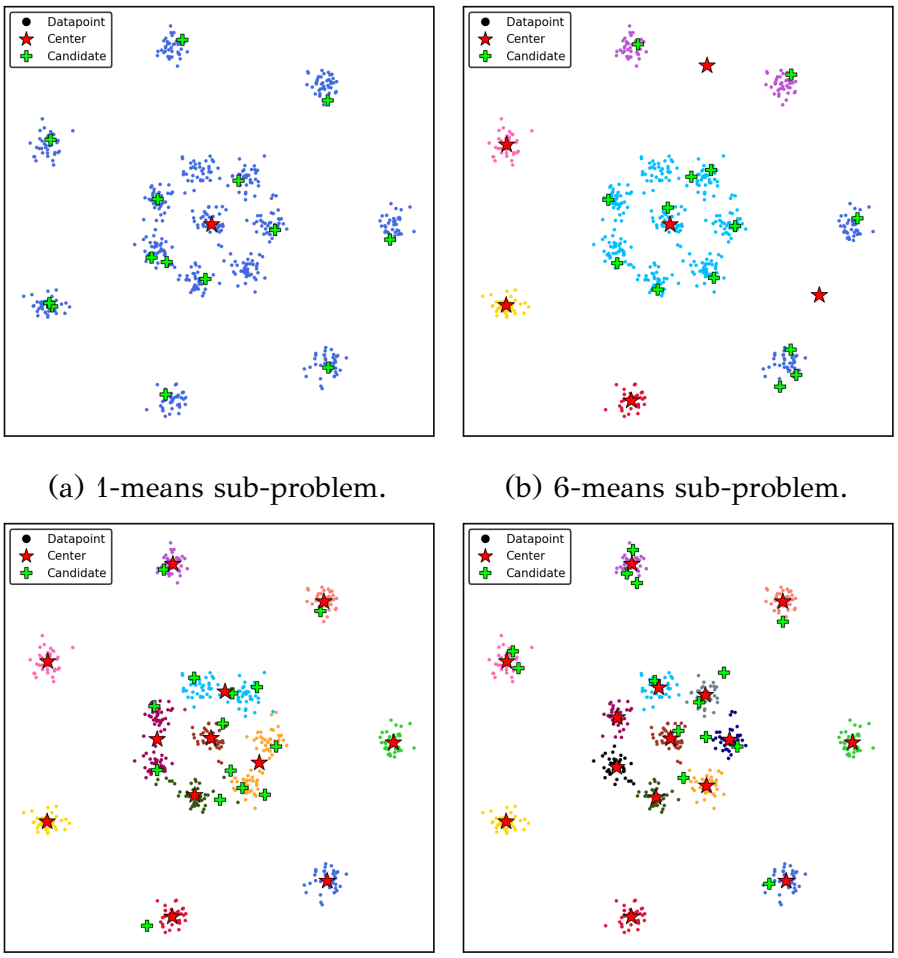
Require: $X = \{x_1, \ldots, x_N\}$: Dataset
Require: M: Set of cluster centers
Require: L: Number of candidates
Require: D: Distance vector
1: **for** $\ell = 1, \ldots, L$ **do**
2: Compute the probability vector $P = (p_1, \ldots, p_N)$, where $p_i = Pr\left(r_k = x_i\right) = d_i \Big/ \sum_{j=1}^{N} d_j$
3: $\{c_1, \ldots, c_{\ell-1}\} \cup \{c_\ell\} \leftarrow$ Sample without replacement one c_ℓ center candidate from dataset X by k-means++ probability vector P
4: **for all** $x_i \in X$ **do**
5: $d_i \leftarrow \min_{r_j \in M \cup \{c_1, \ldots, c_\ell\}} ||x_i - r_j||^2$
6: **end for**
7: **end for**
8:
9: **return** Set of initial center candidates $\{c_1, \ldots, c_L\}$

the minimum error is retained (step 12 in Alg. 2.1). Finally, the distance vector D is updated (steps 13-15 in Alg. 2.1), using the pre-computed center-to-data point distances of the best clustering solution. Following the same reasoning, for each $k = 2, \ldots, K$, it incrementally tackles the k cluster sub-problem by leveraging the solution of the previous $(k-1)$ cluster sub-problem. To solve for k clusters, it utilizes the center positions obtained from the $(k-1)$ sub-problem as initialization for the $k-1$ centers. It then uses batch or sequential sampling strategies, which we discuss next, to initialize the new k_{th} center. Finally, the algorithm returns the clustering solution for every $k \in \{1, \ldots, K\}$. Importantly, it should be noted that steps 9-11 of Alg. 2.1 can be executed concurrently or in parallel. This parallel execution allows for efficient computation and optimization.

We can sample candidates from the probability distribution P through either batch or sequential sampling, presented in Alg. 2.2 and Alg. 2.3, respectively. On the one hand, batch sampling does not require any additional computations as it utilizes only the pre-computed center-to-data point distances $D = (d_1, \ldots, d_N)$. First, it computes the probability vector $P = (p_1, \ldots, p_N)$, where p_i represents the selection probability of the data point x_i (step 1 in Alg. 2.2). The distribution P is formulated so that high selection probabilities are assigned to data points far from cluster centers, and small to near zero are assigned to data points near cluster centers. Then, a set of L candidates $\{c_1, \ldots, c_L\}$ are sampled without replacement by the constructed k-means++ probability vector P (step 2 in Alg. 2.2).

On the other hand, sequential sampling strategy selects one candidate c_ℓ at a time

using the distribution P (step 3 in Alg. 2.3). This procedure aims to select candidates that are far from: i) the set M consisting of the converged solutions of the $k-1$ centers, and ii) the $\ell-1$ candidates that already sampled. After each sampling the distance d_i of each data point x_i is updated accordingly (steps 4-6 in Alg. 2.3). This procedure is repeated the L times in total to create the set of candidates $\{c_1, \ldots, c_L\}$ (steps 1-7 in Alg. 2.3). Although the sequential sampling method incurs the cost of recalculating the minimum distance values, it provides a better spread of the L samples in the dataspace.



(a) 1-means sub-problem. (b) 6-means sub-problem.

(c) 12-means sub-problem. (d) 15-means sub-problem.

Figure 2.1: Illustration of a running instance of the algorithm applied to the "R15" dataset [2]. Circles denote the data points, the cluster centers are represented by red stars, while the center candidates are marked with green crosses.

In Figure 2.1, we present an illustration of a running example of the algorithm

using the 2-dimensional "R15" dataset [2].[1] The circles denote the data points, while their color indicates the cluster category that they belong to, asserted by the global k-means++ algorithm. With the red star, we represent the converged cluster centers for each clustering sub-problem k, while with the green cross-symbols, we show the initial candidate position of the next cluster center. In each sub-figure 2.1a-2.1d, the algorithm solves the current k-means sub-problem. Based on the current solution, the algorithm samples the next center candidates from the k-means++ distribution. The figure demonstrates that the method samples high-quality center candidates.

While the global k-means requires $\mathcal{O}(NK)$ k-means executions, the global k-means++ only requires $\mathcal{O}(LK)$, where generally $L \ll N$. Additionally, we have empirically observed that k-means converges very fast as k grows. The speed up in convergence is reasonable, since in order to solve each k cluster sub-problem, the method exploits the centers of $k-1$ cluster sub-problem that are already positioned sufficiently well. Therefore, the k-means does not require many iterations to converge.

It should be noted that the proposed method is a relaxation of the global k-means algorithm rather than an optimization of the FGKM similar to the methods discussed in the Chapter 1 in Section 1.1. Its simplicity and speed stem from the fact that the sampling strategies require none to minimal computations to select the L candidates, demonstrating superior computational complexity compared to other global k-means variants. In particular, the batch sampling strategy requires no additional distance computations since the center-to-data point distances necessary to define the probability vector P have already been computed by the k-means procedure. Meanwhile, the sequential sampling strategy requires only $\mathcal{O}(NL)$ distance computations for the selection process, which is the same as k-means++. Note also that solving the k-means problem with an incremental procedure has many advantages because, due to the unsupervised nature of the clustering problem, it is usually desirable to obtain solutions for different k that are evaluated using appropriate quality criteria (e.g. silhouette coefficient [92]) for selecting the appropriate number of clusters [93].

[1]The synthetic dataset is available in the following GitHub repository: https://github.com/deric/clustering-benchmark.git.

2.3 Empirical Evaluation

In this Section, we present our experimental study to evaluate the effectiveness of the proposed global k-means++ clustering method, both in batch (gl++ (b)) and sequential sampling (gl++ (s)) settings. Our study involved comparisons against other clustering methods, including global k-means (gl) [3], FGKM (fgl) [3], k-means++ (k-ms++) [29], scalable k-means++ (k-ms||) [94] and standard k-means with random uniform initialization (rnd) [26].[2] Our implementation of the global k-means++ clustering algorithm is available in the following GitHub repository: https://github.com/gvardakas/global-kmeans-pp.git.

Table 2.1: Descriptions of datasets.

Dataset	Type	Description	N	D	Source
Breast	Tabular	Characteristics of breast cancer tumors	569	30	[95]
MNIST	Image	Handwritten digits	60000	784	[96]
Pendigits	Tabular	Handwritten digits	7494	16	[95]
Wine	Tabular	Chemical analysis of wines	178	13	[95]

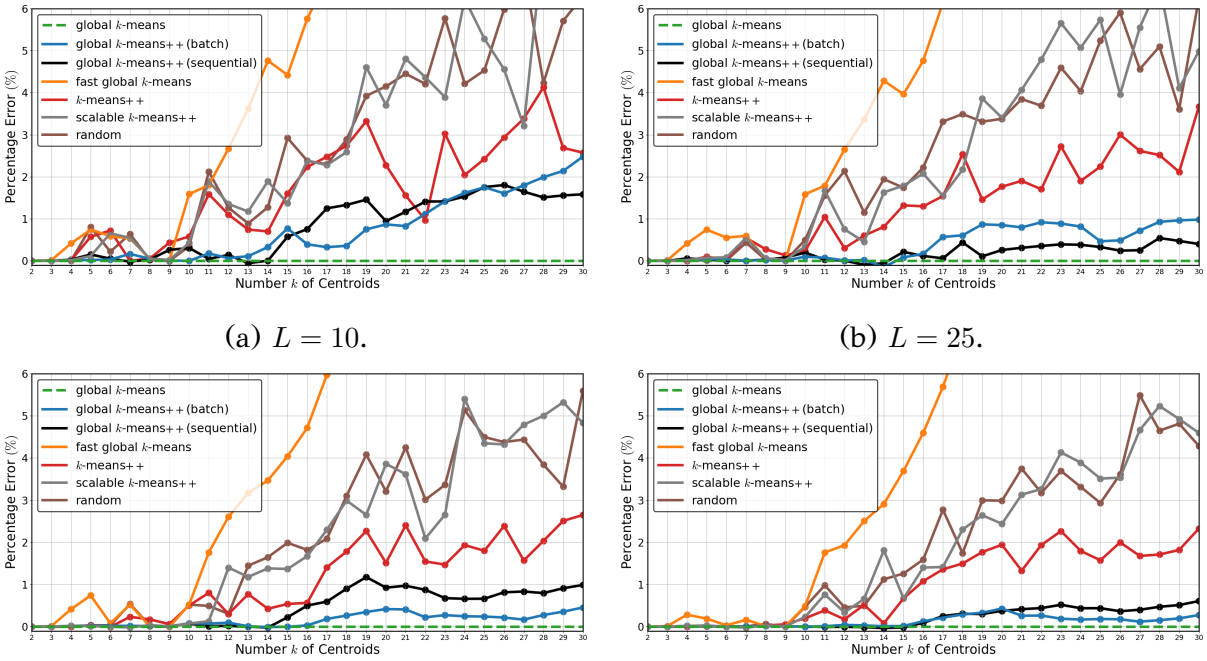
2.3.1 Datasets

In order to evaluate the proposed algorithm, a series of experiments have been conducted on various benchmark and publicly available datasets. Moreover, we deliberately selected datasets that exhibited a broad spectrum of data characteristics, encompassing factors such as the number of samples N and the data dimensionality D, the complexity, and the domain of origin. The description each dataset is presented in table 2.1. As a pre-processing step, we used min-max normalization to map the attributes of each real dataset to the $[0, 1]$ interval to prevent attributes with large ranges from dominating the distance calculations and avoid numerical instabilities in the computations [97].

[2]Experiments were carried on a machine with an Intel® Core™ i7-8700 CPU at 3.20 GHz and 16 GB of RAM.

[2]The time constraint is set to 7 days of execution while the available memory is 16 GB of RAM.

[3]The time constraint is set to 7 days of execution while the available memory is 16 GB of RAM.



(a) $L = 10$.

(b) $L = 25$.

(c) $L = 50$.

(d) $L = 100$.

Figure 2.2: Relative Percentage Error for the Breast, for different L values.

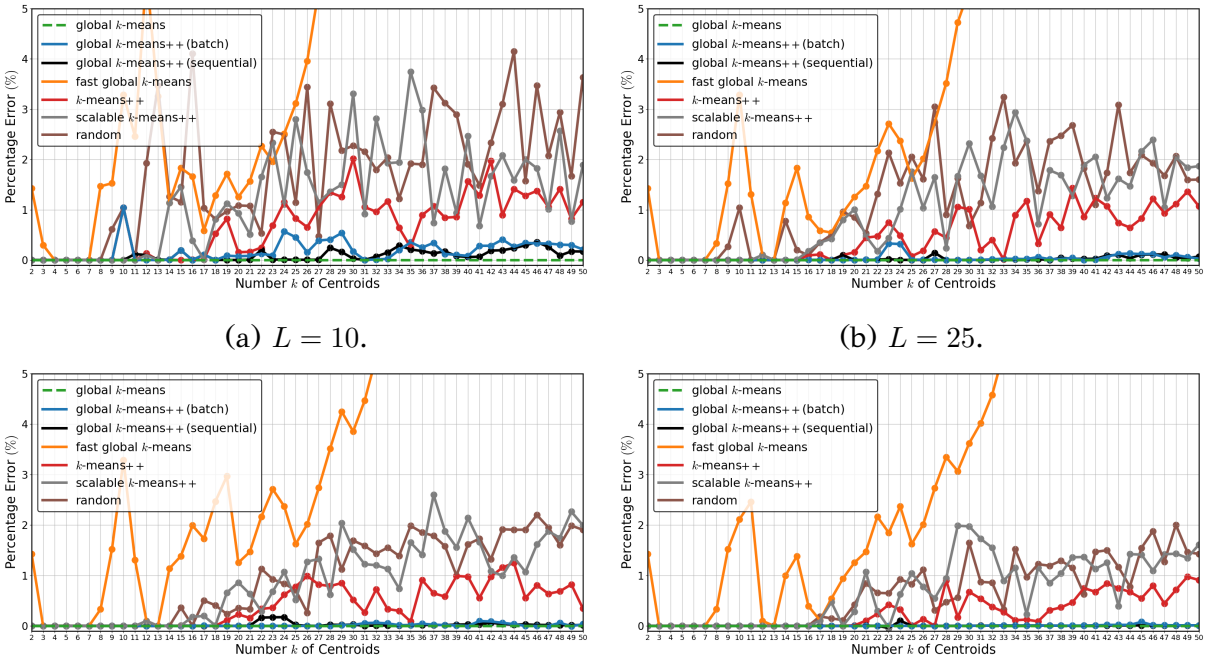
2.3.2 Evaluation

The evaluation of clustering methods encompasses various approaches, including internal and external metrics when the ground truth cluster labels are available. However, our investigation focuses solely on treating it as an optimization problem, with the primary objective being the minimization of clustering error. Consequently, we intentionally disregard any class labels in the data, as our emphasis lies not on assessing external clustering metrics or determining the number of clusters. Instead, we focus on clustering error $E(C_k)$ (eq. 1.1) as the performance measure for the different methods, enabling direct assessment of their error minimization capabilities.

Specifically, in order to evaluate the performance of the different methods, we have computed the relative Percentage Error, defined as

$$PE = \frac{E(C_k) - E(C_k^\star)}{E(C_k^\star)} \times 100\%, \tag{2.1}$$

where $E(C_k^\star)$ is the clustering error of the baseline method (global k-means), while $E(C_k)$ is the error provided by each of the compared methods (refer to Fig. 2.2, 2.3 and 2.4). Considering the MNIST dataset, the global k-means did not terminate in reasonable time due to its high computational complexity. In that case, we have used the difference in clustering errors, relative to the best-performing algorithm



(a) $L = 10$.

(b) $L = 25$.

(c) $L = 50$.

(d) $L = 100$.

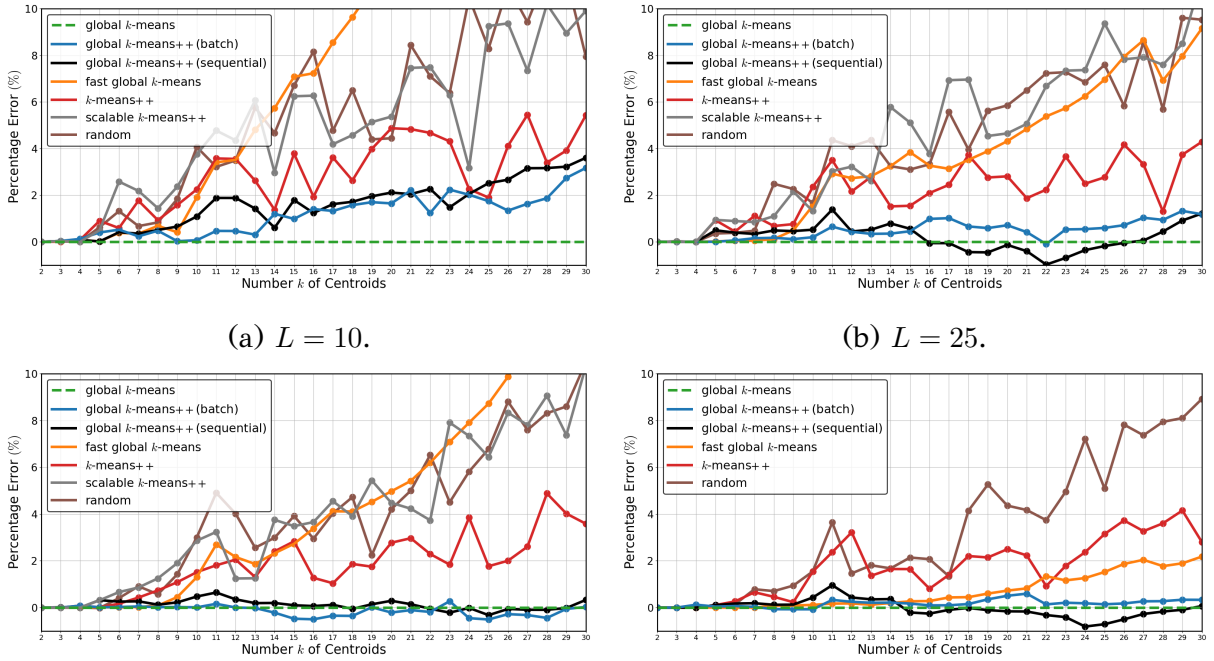
Figure 2.3: Relative Percentage Error for the Pendigits, for different L values.

to facilitate a transparent comparison (see Fig. 2.5). Additionally, we studied the convergence speed of each method for each k by presenting the CPU time required by each algorithm (Table 2.2) and the average number of iterations needed for k-means to converge (Fig. 2.6). In this way, we provide comprehensive information regarding each method's ability to minimize clustering error as well as its computational speed and efficiency.

2.3.3 Experimental Setup

In our experimental study, we executed the compared methods for a maximum number of clusters, denoted as K and evaluated all clustering solutions for $k \in \{1, \ldots, K\}$ in terms of clustering error. We chose the maximum number of clusters K based on the dataset size, with $K = 30$ for smaller datasets (Breast and Wine) and $K = 50$ for medium (Pendigits) and larger datasets (MNIST). For each dataset, we run the compared algorithms with L values of 10, 25, 50, and 100. This range of values allowed us to evaluate the impact of L on the performance of the clustering methods.

In the case of the randomly initialized methods, namely k-means++, scalable k-means++ and standard k-means, we have defined the parameter L to represent the



(a) $L = 10$.

(b) $L = 25$.

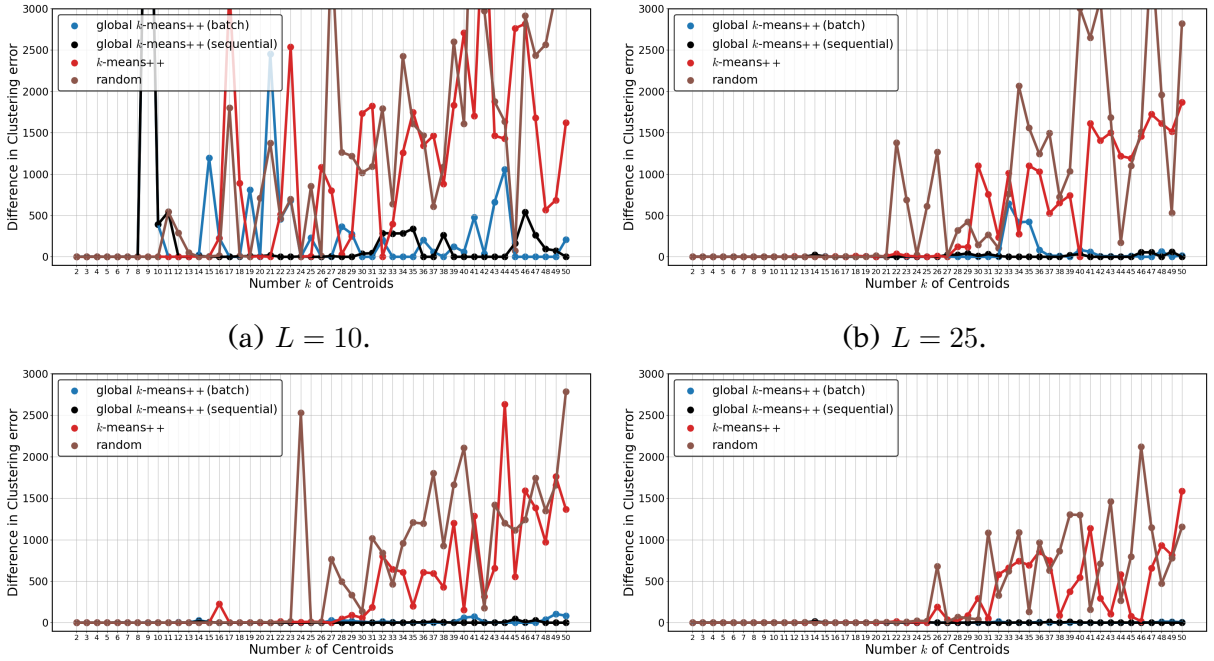
(c) $L = 50$.

(d) $L = 100$.

Figure 2.4: Relative Percentage Error for the Wine, for different L values.

number of restarts within each sub-problem k. However, the scalable k-means++ method has an additional hyperparameter l, which denotes the number of data points sampled in each iteration that we set equal to L. In contrast, for the incremental optimization methods of global k-means++ and FGKM, only a single execution was performed, considering L candidates. We implemented this approach to ensure a fair comparison with the randomly initialized methods, which do not utilize prior sub-problem solutions in their optimization procedure. It is worth noting that the FGKM [3], as presented in Chapter 1 and Section 1.1, originally considers only one candidate (eq. 1.8). However, in our study, we relaxed this limitation by allowing the method to select the top L candidates in each iteration that maximize eq. 1.7. In the following experiments, we evaluate the optimization capabilities of each method, the average number of iterations required, as well as the time needed for convergence to the clustering solution. In summary for each dataset, we conducted the following experiments:

- We selected $K = 30$ or $K = 50$ as the maximum number of clusters depending on the size of the dataset.
- We executed one run of global k-means, global k-means++, and FGKM. For the



(a) $L = 10$.

(b) $L = 25$.

(c) $L = 50$.

(d) $L = 100$.

Figure 2.5: Clustering Error Differences for the MNIST, for different L values.

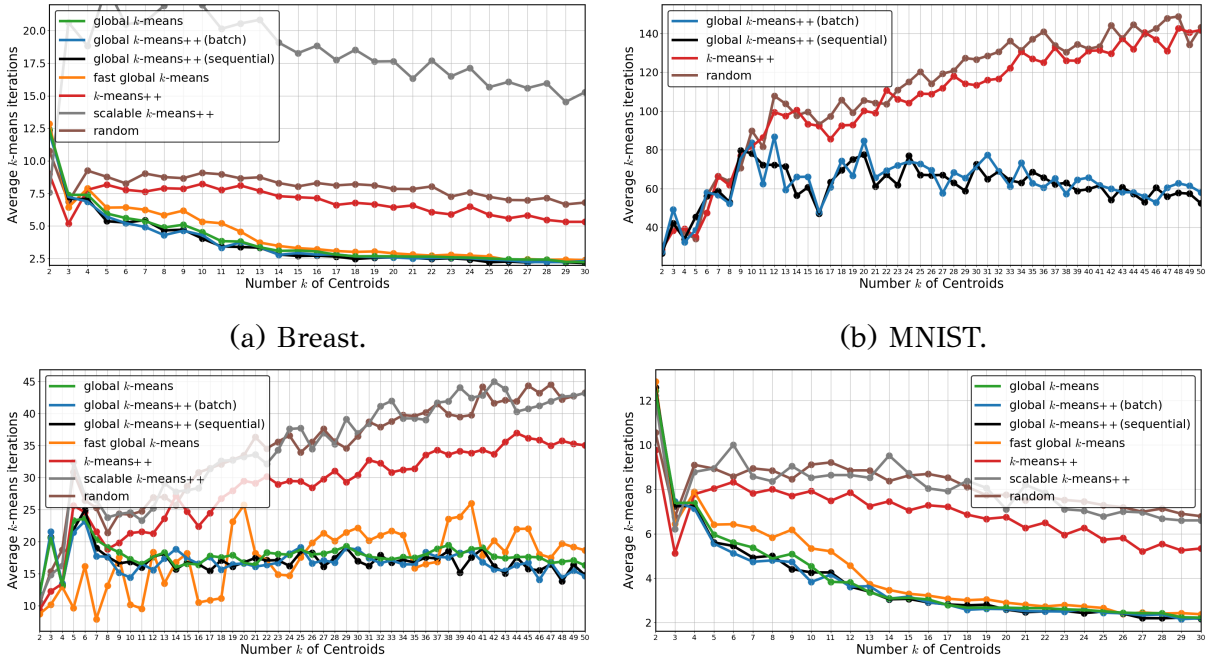
two latter methods, we considered L equal to 10, 25, 50 and 100 candidates in each k cluster sub-problem.

- The k-means++, scalable k-means++ and standard k-means methods were initialized L times for each $k = 1, \ldots, K$.

2.3.4 Results

Figures 2.2, 2.3, and 2.4 depict the relative percentage error between each method and the baseline algorithm, for the respective datasets: Breast, Pendigits, and Wine. However, in the case of the MNIST dataset, Figure 2.5 displays the difference in clustering error between each method and the best-performing algorithm. In the subfigures a, b, c, and d, we present the results of experiments considering different indicative values of L (i.e., 10, 25, 50, and 100) for the global k-means variants, k-means++, scalable k-means++, and the standard k-means algorithm. Moreover, Figure 2.6 provides the average number of iterations required by each algorithm to converge, while Table 2.2 presents the CPU time necessary to compute all K clustering solutions for each algorithm.

The results across all datasets demonstrate that both the batch and sequential



(a) Breast.

(b) MNIST.

(c) Pendigits.

(d) Wine.

Figure 2.6: Average number of k-means iterations.

versions of global k-means++ exhibit similar performance to global k-means, while clearly outperforming the FGKM, k-means++, scalable k-means++ and the standard k-means algorithm. The optimization capabilities of global k-means++ are particularly noteworthy, especially as k increases, where minimizing the clustering error becomes more challenging. As expected, the performance of global k-means++ improves with an increasing number of candidates, better approximating the performance of global k-means. Surprisingly, in the Wine dataset, global k-means++ outperformed global k-means in several k sub-problems (Fig. 2.4b-2.4d). The relative percentage error of global k-means++ was less than 1% when considering more than $L = 25$ candidates. In the Pendigits dataset 2.3, the baseline model and our algorithm coincide with high accuracy for $L = 25, 50, 100$. Notably, global k-means required four days of execution, while global k-means++ completed the task in significantly less time, ranging from 17 second up to almost 2 minutes, depending on the value of L. This indicates that we achieved comparable results in a fraction of the time. Last but not least, in the case of the MNIST dataset 2.5, when we consider $L = 25$ candidates or more, the global k-means++ consistently outperforms the other methods.

It should be noted that in the Wine dataset (2.4d), the FGKM algorithm with $L = 100$ candidates demonstrated promising results. However, its performance across

Table 2.2: CPU Time. Values marked by † and ‡ denote that the method could not be executed due to memory/time and method constraints, respectively.[3]

Dataset	L	gl	gl++ (b)	gl++ (s)	fgl	k-ms++	k-ms||	rnd
Breast	10	32.67s	**0.48s**	0.64s	0.88s	1.57s	5.03s	1.18s
	25		**1.30s**	1.52s	1.80s	4.10s	19.30s	3.04s
	50		**2.55s**	3.12s	3.40s	8.06s	1.10m	6.40s
	100		**5.46s**	5.86s	6.21s	15.03s	3.91m	12.31s
MNIST	10	†	**8.35m**	9.4m	†	29.81m	†	15.5m
	25		**25.58m**	30.23m		1.77h		39.73m
	50		**49.5m**	57.25m		3.68h		1.39h
	100		**1.58h**	1.81h		6.7h		2.70h
Pendigits	10	4d	17.50s	24.70s	45.73s	1.00m	41.18s	**7.30s**
	25		27.60s	41.57s	53.06s	2.51m	3.63m	**19.15s**
	50		43.18s	1.04m	1.13m	4.98m	13.88m	**38.17s**
	100		**1.15m**	1.78m	1.88m	10.00m	57.73m	1.23m
Wine	10	5.40s	**0.32s**	0.40s	0.36s	1.10s	3.09s	0.49s
	25		**0.76s**	0.95s	0.84s	2.77s	8.72s	1.21s
	50		**1.51s**	1.90s	1.66s	5.47s	22.13s	2.46s
	100		**3.06s**	3.81s	3.24s	10.77s	‡	5.10s

all datasets was inconsistent. It is evident that the FGKM algorithm cannot reliably approximate the clustering solution achieved by the global k-means algorithm. Even with an increasing number of candidates, the clustering solution provided by the FGKM significantly deviates from the quality attained by the global k-means++.

As expected, the k-means++ algorithm was successful for small values of k across all datasets. However, it becomes apparent that as k increases, the performance of the k-means++ algorithm deteriorates compared to global k-means and global k-means++. This is because the k-means problem becomes more challenging when it comes to selecting the initial center positions as k raises. Additionally, scalable k-means++ had inferior performance compare to k-means++ in most cases and especially when k is large. As expected, the standard k-means with the random uniform selection yielded the worst results.

Figure 2.6 presents the average number of k-means iterations performed by each method in all datasets and k sub-problems. It becomes evident that as k increases, the incremental methods require fewer iterations in each run of k-means due to improved initialization of the cluster centers. This advantage arises from the fact that

the previous $k-1$ centers have already been positioned in near-optimal solutions. It should be reminded that for each cluster number k, the number of executed k-means runs is $\mathcal{O}(Lk)$ for all compared methods, except for global k-means which executes k-means $\mathcal{O}(Nk)$ times. This means that even if global k-means tends to execute a smaller average number of k-means iterations compared to k-means++, this does not translate to faster execution times. Table 2.2 presents the CPU time required by each algorithm to compute all K clustering solutions. It turns out that if we want to find all k cluster solutions from $k=2$ until a cluster number K, global k-means++ is the faster approach.

2.4 Discussion

From the above experimental results, the following empirical conclusions can be drawn.

The Global k-means incremental method exhibits powerful optimization capabilities. However, due to its big computational requirements, it is often applied to smaller datasets. As shown in the experimental results, even for a medium size dataset like Pendigits global k-means required very large execution time. Nevertheless, it is a very powerful extension of k-means and has been widely used in the clustering literature [98]. It should be noted that the algorithm provides high quality solutions, however, it does not guarantee the discovery of the global optimum. As we have analyzed in the introduction, minimization of clustering is known to be a NP-hard problem. We have observed such a case in the Wine dataset where the global k-mean did not yield the best results.

FGKM constitutes a natural fast variant of the global k-means method which, in order to initialize the new cluster center, it selects the data point $x_{i^\star}$ that minimizes an upper bound of the final k-means clustering error. We should recall that we allowed the FGKM method to select the top L candidates that maximize the eq. 1.8 instead of just one as originally proposed. However, this candidate selection heuristic performs poorly even compared to the random initialization method. This suggests, that while FGKM is intuitively justified, it does not constitute a very effective strategy.

The k-means++ algorithm is considered as the state-of-the-art random initialization algorithm for solving the k-means clustering problem due to strong empirical per-

formance, theoretical guarantees of the solution quality, and simplicity [99, 100]. As expected, it demonstrated good performance and generally surpassed most methods, including the FGKM, the scalable k-means++, and of course k-means with random initialization. However, global k-means and global k-means++ had better performance overall, and this becomes more apparent as k gets larger.

The proposed Global k-means++ produced very satisfactory results. It demonstrates similar performance to the global k-means with a large reduction in computational complexity. Its candidate selection method chooses excellent candidates, as it is inspired by the successful k-means++ seeding strategy. Moreover, it provides the clustering solution for all intermediate values of k. Therefore, if we wish to obtain clustering solutions for different k, its time performance is even more impressive compared to the rest of the random initialization methods, which have to be executed separately for each k.

2.5 Summary

In this Chapter, we introduced the global k-means++ clustering algorithm, which is an effective relaxation of the global k-means algorithm that provides an ideal compromise between clustering error and execution speed. The basic idea of the proposed method is to take advantage of the superior clustering solutions that the global k-means algorithm can provide while avoiding its substantial computational requirements. The global k-means++ is an incremental clustering approach that dynamically adds one cluster center at each k cluster sub-problem. For each k cluster sub-problem, the method selects L data points as candidates for the initial position of the new center using the effective k-means++ selection probability distribution. The selection method is fast and requires no extra computational effort for distance computations.

Global k-means++ has been tested on various benchmark publicly available datasets and has been compared to the global k-means, the FGKM with multiple candidates, the k-means++, the scalable k-means++, and the standard k-means with random uniform initialization. The experimental results reveal its superiority against all method except global k-means. In this case, its performance is comparable but with a significant decrease in computational cost, thus establishing it as an effective relaxation.

Chapter 3

The Global Kernel k-means++ Algorithm for Efficient Clustering in the Feature Space

---

3.1 Introduction

As noted in Chapter 1 and Section 1.1, the k-means algorithm is widely used but suffers from several limitations. To address its second major limitation, which is the assumption of linear cluster separability, the kernel k-means algorithm was introduced [40, 101, 10, 45, 102]. The kernel k-means idea is that the data instances are first mapped from the input space to a higher-dimensional feature space using a nonlinear transformation. In this richer representation, the transformed data are expected to become linearly separable, making the use of the k-means algorithm more effective. Therefore, the k-means clustering error is minimized in this feature space, allowing for the identification of non-linearly separable clusters in the input space,

thus overcoming the second limitation. Spectral clustering constitutes an additional nonlinear approach to clustering, exploiting the eigenvectors of an affinity matrix constructed from the data [103, 104]. Notably, a direct relationship between kernel-based methods and spectral clustering has been established in [101].

However, introducing kernel-trick into the k-means procedure reintroduces the first limitation: *how to properly initialize the centers in feature space?* In this Chapter [44], inspired by the capabilities of the GkM++ method [88], we present the *global kernel k-means++* (GKkM++) algorithm, a novel approach to obtain superior kernel-based clustering solutions similar to those of GKkM at a lower computational cost. This improvement is achieved by integrating the effective center selection probability distribution of kernel k-means++ with the global kernel optimization strategy, allowing GKkM++ to efficiently explore the solution space while maintaining excellent optimization capabilities.

The optimization of the kernel k-means error is a long-standing obstacle and to the best of our knowledge, no recent methods have been proposed that specifically aim to optimize the kernel k-means objective. Instead, some researchers have focused on developing computationally cheaper kernels to accelerate the process and simplify the overall procedure [105]. Others have attempted to reformulate the kernel k-means optimization problem into a potentially simpler alternative [106, 102]. In any case, the global kernel k-means method is considered state of the art approach for optimizing the kernel k-means error. Additionally, the proposed Global Kernel k-Means++ method can also benefit from these advancements, such as leveraging more computationally efficient kernels.

To assess the effectiveness of the proposed approach, we conducted a comprehensive set of experiments on a diverse collection of benchmark datasets. The evaluation includes both synthetic and publicly available real-world datasets, and the proposed algorithm is compared against several established methods, including global kernel k-means, kernel k-means++, and kernel k-means with random uniform initialization (RKkM). Additionally, we examined its performance on the graph partitioning problem. The goal was to evaluate the clustering quality, robustness, and scalability of the method relative to these baseline algorithms. Our analysis considered a range of conditions, including varying numbers of clusters, dataset complexity, sample size, and dimensionality. Quantitative results were further supported by visualizations, providing deeper insight into the structure and coherence of the resulting clusters.

The rest of the Chapter is structured as follows. Specifically, in Section 3.2, we present the global kernel k-means++ algorithm, which offers enhanced optimization capabilities and reduced computational requirements compared to its predecessor, the GKkM method. In Section 3.3, we provide results on computational complexity. In Section 3.4, we present the results of an extensive comparative experimental study using both synthetic and real datasets. Additionally, we evaluate the proposed method in the context of the graph partitioning task (community detection). Finally, Section 3.5 summarizes the Chapter.

3.2 Global kernel k-means++

GKkM (Alg. 1.5) constitutes a deterministic method that aims to tackle the initialization issue of kernel k-means but at a high computational cost. It operates incrementally and provides feature space partitions into K clusters by sequentially addressing every k-cluster subproblem $k = 1, \ldots, K$. To address the problem with k clusters, N kernel k-means executions are implemented, where each of the N data instances is considered as the initial position for the new cluster center in the feature space. This exhaustive consideration of initial positions for the new cluster center results in an effective placement; thus, superior clustering solutions are obtained. However, exhaustive search limits the algorithm's applicability to relatively small datasets.

To preserve the optimization capabilities of the greedy sequential strategy of GKkM, and inspired by the GkM++ method, we introduce the *global kernel k-means++* (GKkM++) method for feature space clustering. The key difference is that, instead of evaluating every data instance $x_n \in X$ as a potential candidate for the new center, the proposed method considers only L appropriately selected candidates. Therefore, only L kernel k-means runs are required for each k-cluster subproblem, with $L \ll N$. GKkM++ achieves this by using an efficient candidate selection strategy that prioritizes sampling from regions without existing cluster centers in the feature space. Specifically, the method employs the KkM++ instance selection probability distribution to guide the selection process. The main objective of this approach is to obtain enhanced optimization capabilities at low computational complexity. The GKkM++ algorithm is presented in detail in Alg. 3.1.

The algorithm requires as input the kernel matrix $\mathbf{K}$, the final cluster number K,

Algorithm 3.1 `Global Kernel` k`-Means++`

Require: $\mathbf{K}$: Kernel matrix
Require: K: Number of clusters
Require: L: Number of candidates
Require: $S \in \{$`` `Batch' ``, `` `Sequential' ``$\}$: Sampling Method
1: Initialize $\mathcal{C}_1^\star \leftarrow X$
2: $D \leftarrow (||\phi(x_1) - m_1||^2, \ldots, ||\phi(x_N) - m_1||^2)$ using eq. 1.11 *// Distance vector*
3: **for** $k = 2, \ldots, K$ **do**
4: $\{c_1, \ldots, c_L\} \leftarrow$ `Candidate Selection`$(S, \mathbf{K}, L, D)$
5: **for all** $c_\ell \in \{c_1, \ldots, c_L\}$ **do**
6: $C'_k \leftarrow \{c_\ell\}$ *// Initialization of the new cluster C'_k using data instance c_ℓ*
7: $\mathcal{C}'_{k-1} \leftarrow \mathcal{C}^\star_{k-1} / \{c_\ell\}$ *// Exclude c_ℓ from the $k-1$ solution*
8: $\mathcal{C}'_k \leftarrow \mathcal{C}'_{k-1} \cup C'_k$ *// Initialization of clustering $\mathcal{C}'_k$*
9: $(\mathcal{C}^\ell_k, E(\mathcal{C}^\ell_k)) \leftarrow$ Run `kernel` k`-means`$(\mathbf{K}, k, \mathcal{C}'_k)$ (Alg. 1.3)
10: **end for**
11: $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k)) \leftarrow$ Solution with the minimum error $E(\mathcal{C}^\star_k)$ among the L partitions $\mathcal{C}^\ell_k$
12: **for all** $x_i \in X$ **do**
13: $D[i] \leftarrow \min_j(||\phi(x_i) - m_j||^2)$ using eq. 1.11 *// Pre-computed distances* in step 9
14: **end for**
15: **end for**
16: **return** solutions $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k))$ for every $k \in \{1, \ldots, K\}$

the number of candidates L, and the strategy S used to sample candidates: either *batch* or *sequential*. Note that $D = (d_1, \ldots, d_N)$ denotes the vector of distances d_i, with d_i being the distance of $\phi(x_i)$ from its nearest center m_j in the feature space. The method follows the steps outlined below. First, it computes the solution to the kernel 1-means subproblem by setting the first cluster to contain all data instances. It also initializes the distances d_i (steps 1-2 in Alg. 3.1). Then, in order to solve the 2-cluster subproblem, the kernel 1-means solution (obtained in step 1) is utilized as the first initial cluster. Then the set of L candidate instances $\{c_1, \ldots, c_L\}$ is formed through sampling using the batch or sequential strategy (steps 4 in Alg. 3.1) in order to determine candidate initialization for the second cluster. Since we do not assume direct access to $\phi(c_\ell)$, the GKkM++ method, similar to GKkM, initializes the candidate c_ℓ as a new cluster containing a single element, bypassing this problem (steps 6-8 in Alg. 3.1). Next, L executions of kernel k-means until convergence take place, one for each initial cluster $\{c_\ell\}$ (steps 5-10 in Alg. 3.1). From the L solutions found, we select the one with the minimum clustering error value (step 11 in Alg. 3.1).

Finally, we update the distances d_i (steps 12-14 in Alg. 3.1). In the same spirit, for each $k = 2, \ldots, K$, the method progressively solves the k-cluster subproblem by exploiting the solution of the $(k-1)$-cluster subproblem. To compute the solution with k clusters, the $k-1$ clusters are initialized using the already found partition of the $(k-1)$-cluster subproblem. Then, the L candidate instances are sampled (using either the batch or sequential strategy discussed next) to select the element that will be used to initialize the k_{th} cluster. The final output of the algorithm is a clustering solution for all $k = 1, \ldots, K$. It is important to note that steps 5-10 of Alg. 3.1 could be executed in parallel, providing significant speedup (up to L times faster). For further speed up kernel k-means execution, MapReduce schemes [107], and coreset [108] could also be applied.

The initial center candidates are sampled using the probability distribution P, which expresses the feature space distance of the instances from the closest cluster center. As presented in Alg. 3.2, two sampling strategies have been considered. *Batch sampling* (steps 2-3 in Alg. 3.2) is simpler since it exploits only the pre-computed distances d_i, $i = 1, \ldots, N$. Using these distances, the selection probability p_i for data instance x_i (step 2 in Alg. 3.2) is computed and the probability vector $P = (p_1, \ldots, p_N)$ is formed. Probability p_i is inversely proportional to distance d_i, so data instances far (in feature space) from the current cluster centers are more likely to be selected. Then, using the computed kernel k-means++ probability vector P, a set of L candidates $\{c_1, \ldots, c_L\}$ are selected by sampling without replacement (step 3 in Alg. 3.2).

Alternatively, a sequential sampling approach can be used in which each candidate instance is sequentially sampled. In this case, the distribution P is updated so that instances are selected to be distant not only from the current cluster centers but also from the already selected instances (step 7 in Alg. 3.2). Therefore, after each sampling, the distance d_i of each data instance $\phi(x_i)$ is updated accordingly (steps 9-11 in Alg. 3.2) to take into account the newly selected instance. After L sampling steps, the set $\{c_1, \ldots, c_L\}$ of candidates has been determined (steps 5-12 in Alg. 3.2). While the sequential sampling method requires recomputing the minimum distance values, it results in a more effective distribution of the L candidates within the feature space. It is important to note that at the beginning of the sequential sampling strategy, the relation $d_i = \min_{r_j \in M} ||\phi(x_i) - r_j||$ holds. Consequently, the computation in step 10 only needs to consider the set $\Phi_c^{(\ell)}$, which reduces the distance computation to $d_i = \min_{\ell}\{d_i, |\phi(x_i) - \phi(c_\ell)|^2\}$.

Algorithm 3.2 `Candidate Selection`

Require: **K**: Kernel matrix

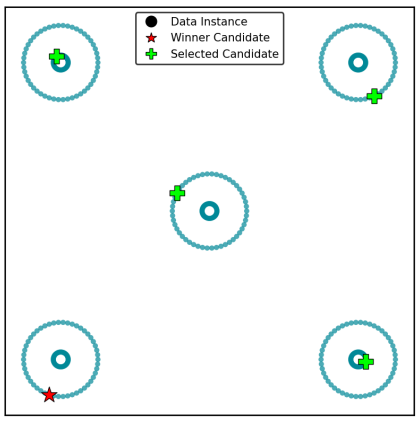
Require: L: Number of candidates

Require: $D = (d_1, \ldots, d_N)$: Vector of minimum instance to cluster center distances in feature space

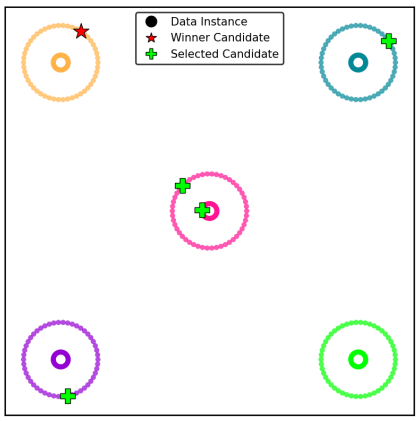
if $S =$ `'Batch'` **then**
  Compute the probability vector $P \leftarrow (p_1, \ldots p_N)$, where $p_i = d_i / \sum_{j=1}^{N} d_j$
  $\{c_1, \ldots, c_L\} \leftarrow$ Sample without replacement L candidates
else
  for $\ell = 1, \ldots, L$ **do**
    Compute the probability vector $P = (p_1, \ldots p_N)$, where $p_i = d_i / \sum_{j=1}^{N} d_j$
    $c_\ell \leftarrow$ Sample without replacement one candidate instance c_ℓ from X using P
    $\Phi_c^{(\ell)} \leftarrow \{c_1, \ldots, c_{\ell-1}\} \cup \{c_\ell\}$
    for all $x_i \in X$ **do**
      $d_i \leftarrow \min_{r_j \in M \cup \Phi_c^{(\ell)}} ||\phi(x_i) - r_j||$ using eq. 1.11
    end for
  end for
end if
return Set of initial center candidates $\{c_1, \cdots, c_L\}$

Figure 3.1 demonstrates a practical example of the proposed GKkM++ algorithm using a synthetic two-dimensional dataset with ten clusters arranged in five pairs of concentric rings. The dots represent the data instances, while the color indicates the respective cluster assignments determined by the algorithm. Green crosses mark the candidate positions for initializing the next cluster. Additionally, the red star denotes the winner candidate corresponding to the best initialization. In subfigures 3.1a-3.1d, the method addresses a kernel k-means subproblem with different number of clusters k. Based on each subproblem solution, the next center candidates are sampled from the KkM++ distribution. Specifically, in Figure 3.1a, the kernel 1-means subproblem is initially solved by considering the entire dataset as a single cluster (denoted in blue), and the method successfully samples the next candidate initial center positions across multiple distinct clusters in the dataset. This effective behavior is consistently observed in Figure 3.1b and Figure 3.1c until the final clustering with $k = 10$ clusters is achieved in Figure 3.1d.

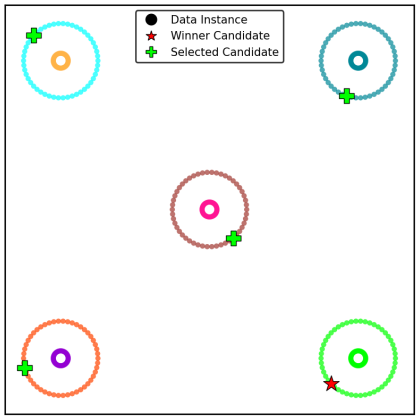
In Fig. 3.2 (left), we present the solution when the method is halted at $k = 8$. We observe that 6 of the 10 clusters have been correctly identified, while the remaining 4 require further refinement. At this stage, we expect data instances located farther from



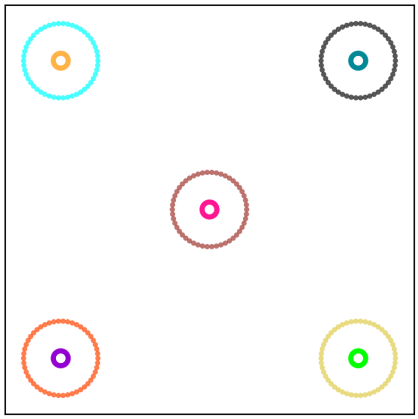
(a) Solution with $k = 1$ cluster.



(b) Solution with $k = 5$ clusters.



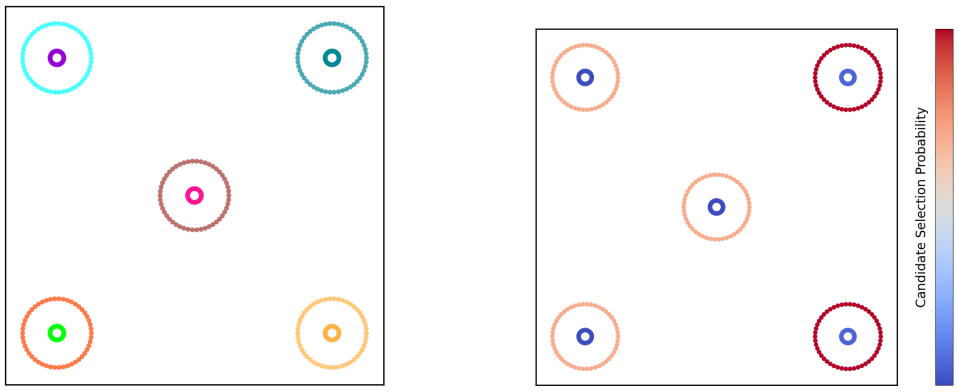
(c) Solution with $k = 8$ clusters.



(d) Final solution with $k = 10$ clusters.

Figure 3.1: Illustrative example of GKkM++ execution. Data instances are denoted with circles, red crosses indicate cluster centers and red star denotes the winner candidate corresponding to the best initialization.

any cluster center to be more likely to be selected than those closer to the centers. On the right side of Fig. 3.2, a heatmap illustrates the selection probability of each data instance as a candidate for initializing the next cluster in the $k = 9$ subproblem. In the heatmap, dark blue indicates low selection probability, while red indicates high selection probability. As observed, data instances in the two outer rings, which do not form a cluster independently, exhibit a significantly higher selection probability. Conversely, data instances in the inner rings have a low selection probability, as their proximity to a cluster center in the feature space reduces the likelihood of their selection. Lastly, the data instances in outer rings that already form distinct clusters have a lower selection probability than those in outer rings that have not been partitioned.



(a) Solution for $k = 8$ clusters.

(b) Heatmap with candidate selection probabilities.

Figure 3.2: Illustratve example of GKkM++ execution. Circles denote the data instances.

3.3 Complexity Analysis

Since GKkM++ is closely related to GkM++ and kernel k-means, it inherits their computational complexity. Specifically, the complexity of kernel k-means is $\mathcal{O}(N^2\tau)$, where τ represents the number of iterations until convergence. Assuming K clusters, GkM++ requires $\mathcal{O}(KL)$ executions of kernel k-means. Consequently, the computational complexity of GKkM++ is $\mathcal{O}(N^2KL\tau)$, assuming access to the kernel matrix K, where $L \ll N$. This is a significant speedup compared to GKkM, which has a $\mathcal{O}(N^3K\tau)$ complexity. It should be noted that we have empirically observed that kernel k-means tends to converge faster (in fewer iterations) as k increases in this sequential clustering framework. This speedup is reasonable because, when solving each k-cluster subproblem, the method utilizes the solution from the $(k-1)$-cluster subproblem, which is already well-positioned. As a result, the number of iterations τ required for convergence typically decreases as k increases, thereby improving the algorithm's overall efficiency.

As mentioned previously, the batch sampling strategy does not require extra distance computations. Sequential sampling adds $\mathcal{O}(NL)$ distance computations, similar to KkM++ for sampling the K cluster centers. It should be noted that an additional advantage emerges from the incremental solution of the kernel k-means problem: in many cases, the number of clusters is not given. Therefore, it is necessary to obtain solutions for a range of k values. Those solutions will be subsequently evaluated using clustering quality criteria (e.g. silhouette score [92], modularity [109], inclu-

sion [110, 111] etc.) for cluster number estimation [93].

3.4 Empirical Evaluation

We have conducted an extensive series of experiments to assess the performance of the proposed KkM++ and GKkM++ methods, the latter both with batch (b) and sequential (s) sampling strategy. The methods were compared against GKkM and RKkM.

The experimental evaluation is divided into three subsections. In subsection 3.4.1, we demonstrate the clustering performance of each algorithm on synthetic two-dimensional datasets that are not linearly separable. Subsection 3.4.2 focuses on evaluating the algorithms on graph partitioning tasks, while in subsection 3.4.3, real-world datasets are considered.

Table 3.1: Kernels used in our experimental evaluation.

Kernel Type	Kernel Function
Cosine Kernel	$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x_i} \cdot \mathbf{x_j}}{\|\mathbf{x_i}\| \|\mathbf{x_j}\|}$
Polynomial Kernel	$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \left(\gamma \left(\mathbf{x_i} \cdot \mathbf{x_j}\right) + c_0\right)^{deg}$
RBF Kernel	$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|_2^2\right)$

3.4.1 Synthetic Data Demonstration

At first, we considered two challenging synthetic datasets that are not linearly separable, as shown in Fig. 3.3 and Fig. 3.4, respectively. Specifically, the first synthetic dataset (Fig. 3.3) consists of nine pairs of concentric rings, each containing 50 data instances, resulting in a total of 900 points evenly distributed across eighteen clusters. The second dataset (Fig. 3.4) comprises three rings and six Gaussian distributions, with each cluster containing 50 data instances. This results in a total of 450 points uniformly distributed across all nine clusters. In both cases, the RBF kernel was employed. Algorithms such as k-means, which rely on identifying linearly separable clusters in the data space, are therefore unsuitable for these datasets.

In our synthetic data demonstration, the GKkM is executed once since it produces deterministic clustering results. In contrast, RKkM and KkM++ were employed 100

Table 3.2: Clustering error comparison of the evaluated methods on 2D synthetic datasets.

Dataset	RKkM	KkM++	GKkM	GKkM++ (b)	GKkM++ (s)
18 Rings	362.56	361.66	345.10	345.10	345.10
3 Rings & 6 Gaussians	128.02	127.21	121.75	121.75	121.75

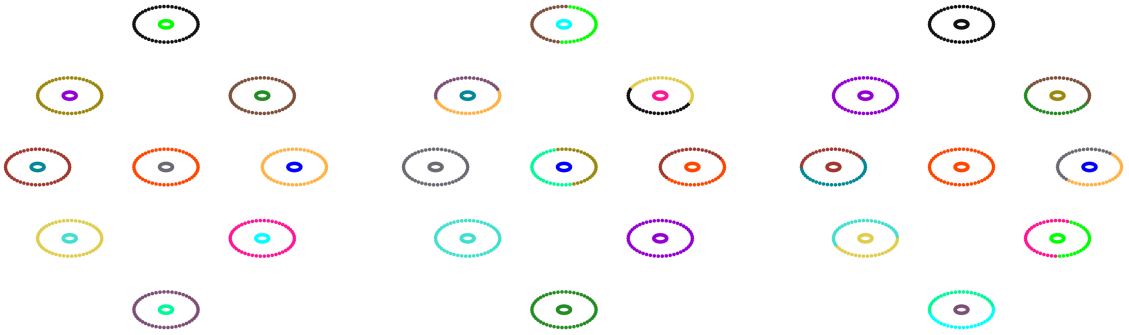
times, from which we report the minimum clustering error, similar to the experimental setup of survey [93]. Furthermore, for the variants of GKkM++, we set the number of cluster candidates to $L = 100$ (equal to the number of restarts used for the rest of kernel k-means variants). We conducted the experiment a single time, as practiced in [88]. Finally, the quality of the solutions produced by the algorithms is assessed through clustering error (eq. 1.9) and through visual inspection as shown in Figures 3.3 and 3.4.

In these challenging clustering tasks, GKkM accurately identifies all clusters. Similarly, GKkM++ with sequential and batch sampling also successfully partitions both datasets, demonstrating the effectiveness of both sampling procedures. The solutions produced by GKkM and GKkM++ variants achieve the lowest clustering errors, as shown in Table 3.2. On the other hand, RKkM and KkM++ fail to identify all clusters in both datasets correctly. However, as shown in Table 3.2, KkM++ outperforms RKkM in both synthetic datasets in terms of clustering error.

3.4.2 Graph Partitioning

Graph partitioning represents a different approach to data clustering. In this context, we are provided with a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of vertices and edges, respectively. Our goal is to partition the graph into disjoint clusters that satisfy specific conditions. Several objectives for graph partitioning have been proposed, including ratio association, ratio cut, normalized cut, and others. Spectral methods are commonly used to address these problems by computing the eigenvectors of the affinity matrix [62]. However, eigenvector computation is computationally intensive, requiring $\mathcal{O}(N^3)$ operations, and may become impractical for extensive graphs.

It is known that the kernel k-means objective is equivalent to graph cut objective once the kernel matrix is appropriately defined [101, 42, 112]. This proof is established



(a) GKkM++(b&s)/GKkM (b) RKkM (c) KkM++

Figure 3.3: Clustering results for the eighteen rings dataset.

(a) GKkM++(b&s)/GKkM (b) RKkM (c) KkM++

Figure 3.4: Clustering results for the three rings with six Gaussians dataset.

by formulating the problem as trace maximization, following a similar methodology to that in [113], where the k-means objective is also framed as trace maximization. Kernel k-means bypasses the requirement to compute eigenvectors; however, it cannot ensure an optimal solution due to its dependence on cluster initialization. Even when eigenvector computations are feasible, experiments in [42] indicate that kernel k-means can further enhance the clustering results obtained from spectral methods.

GKkM can be effectively utilized for relatively small graph partitioning tasks [45]. Naturally, KkM++ and GKkM++ can also address the graph partitioning problem, as they are built upon the foundations of kernel k-means and GKkM, respectively. In the following experiments, we demonstrate the performance of each clustering method on the graph partitioning task using three graphs of increasing difficulty (Table 3.3). We evaluate the performance of the compared methods for a maximum number of clusters (or communities), denoted by K, and consider each solution for $k = 1, \ldots, K$. In all experiments, we fix $K = 50$. To ensure consistency in the comparison, each subproblem with k clusters is solved using $L = 100$, where L represents either the number of restarts or the number of candidate solutions explored. Specifically, for KkM++ and RKkM, L defines the number of restarts. For GKkM++, L specifies the number of candidates selected. Finally, due to its deterministic nature, GKkM is

Table 3.3: Descriptions of utilized graphs.

Graph	Description	# nodes	# edges	Source
email-Eu-TD1	Email communication among first department members	309	1938	[114]
email-Eu	Email exchanges among all institution members	1005	16706	[114]
GRQC	Co-authorship in General Relativity and Quantum Cosmology	5242	14496	[114]

executed once.

Graph Datasets

To evaluate the effectiveness of the proposed algorithms, we conducted a series of experiments using freely accessible graphs. We intentionally selected graphs that exhibit diverse characteristics, including the number of nodes and edges (see Table 3.3).

The email-Eu-TD1 graph captures email communication exclusively among members of the first department within a European research institution, with edges indicating the sender-receiver relationships in both directions. In addition, email-Eu graph is constructed from anonymized email data collected from the same institution, reflecting all incoming and outgoing communications among its members. An edge (u, v) exists in this graph if person u has sent at least one email to person v, thus representing communication solely within the institution, while excluding interactions with external entities. Finally, the GRQC collaboration graph illustrates scientific collaborations among authors with papers in the General Relativity and Quantum Cosmology category. An undirected edge exists between authors i and j if they co-authored a paper together; if a paper has k co-authors, it generates a fully connected subgraph of k nodes.

Graph Partitioning Evaluation

Let us define $links(\mathcal{A}, \mathcal{B})$ as the cumulative weight of the edges between the nodes in sets $\mathcal{A}$ and $\mathcal{B}$ as $links(\mathcal{A}, \mathcal{B}) = \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{B}} \mathbf{A}_{ij}$, where $\mathbf{A}$ is the affinity matrix that captures the pairwise similarities among the vertices. Similarly, let $degree(\mathcal{A})$ represent the sum of the edge weights between the nodes in $\mathcal{A}$ and all vertices, expressed as $degree(\mathcal{A}) = links(\mathcal{A}, \mathcal{V})$, where $\mathcal{V}$ is the set of all vertices. Let $\mathbf{D}$ be the diagonal $|\mathcal{V}| \times |\mathcal{V}|$ degree matrix, where $\mathbf{D}_{ii} = \sum_{j=1}^{|\mathcal{V}|} \mathbf{A}_{ij}$. Next, we define the graph partitioning objectives optimized in the experimental section: the ratio association and

the normalized cut.

1. Ratio Association problem aims to maximize the internal connectivity of clusters in proportion to their size, and its objective function is presented in the following equation:

$$RA(G) = \max_{\mathcal{V}_1,\ldots,\mathcal{V}_M} \sum_{i=1}^{M} \frac{\text{links}(\mathcal{V}_i, \mathcal{V}_i)}{|\mathcal{V}_i|}. \tag{3.1}$$

To align the objective function of the weighted kernel k-means algorithm with the ratio association problem, we set $w_i = 1$, $w_j = 1$, $w_l = 1$ and $\mathbf{K} = \mathbf{A}$ of the eq. 1.9 and eq. 1.11. This makes the problem equivalent to the unweighted kernel k-means, where the affinity matrix is treated as the kernel matrix [42, 112].

2. Normalized Cut problem seeks to minimize the cut between clusters and the rest of the graph relative to the cluster's degree [115, 116]. This objective is widely used in graph partitioning, and its formulation is the following:

$$NC(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_M} \sum_{i=1}^{M} \frac{\text{links}(\mathcal{V}_i, \mathcal{V}/\mathcal{V}_i)}{\text{degree}(\mathcal{V}_i)}. \tag{3.2}$$

To transform the objective function of weighted kernel k-means to correspond to that of the normalized cut, we need to set $w_i = \mathbf{D}_{ii}$, $w_j = \mathbf{D}_{jj}$, $w_l = \mathbf{D}_{ll}$, and $\mathbf{K} = \mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}$ [42, 112].

The previously discussed definitions of the kernel matrix do not guarantee that it will be positive semidefinite, which is essential for its validity in our algorithms. Although being positive semidefinite is a sufficient condition, it is not necessary for the convergence of the examined algorithms. A solution to this problem is proposed in [42], which involves applying a diagonal shift to the kernel matrix. To address the ratio association problem, we define $\mathbf{K} = \lambda\mathbf{I} + \mathbf{A}$, where $\mathbf{I}$ represents the identity matrix and λ is a sufficiently large constant to guarantee that $\mathbf{K}$ is positive semidefinite. Additionally, to tackle the normalized cut problem, we formulate $\mathbf{K} = \lambda\mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}$. A notable point is that this adjustment to the kernel matrix does not alter the goal of the problem. However, as demonstrated in [42], it may adversely affect the performance of the algorithms if the shift λ is excessive [45].

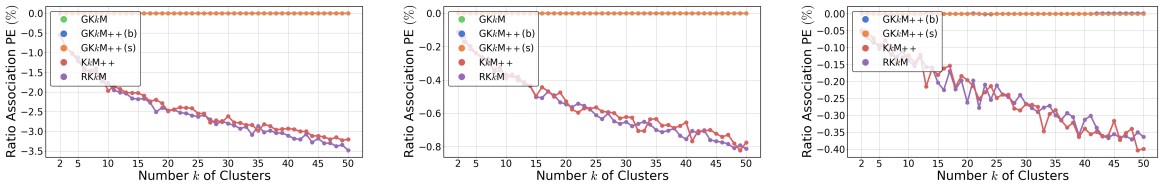
In our study, clustering is framed as an optimization problem. Our aim is to obtain solutions of minimum error in the feature space. Therefore, we evaluate the performance of each method using the clustering error. *Minimizing clustering error is equivalent to maximizing the ratio association (eq. 3.1) in the first case and minimizing the normalized cut (eq. 3.2) in the second case.* For performance comparison we calculate

the relative Percentage Error:

$$\mathrm{PE} = \frac{E(\mathcal{C}_k) - E(\mathcal{C}_k^\star)}{E(\mathcal{C}_k^\star)} \times 100\%, \tag{3.3}$$

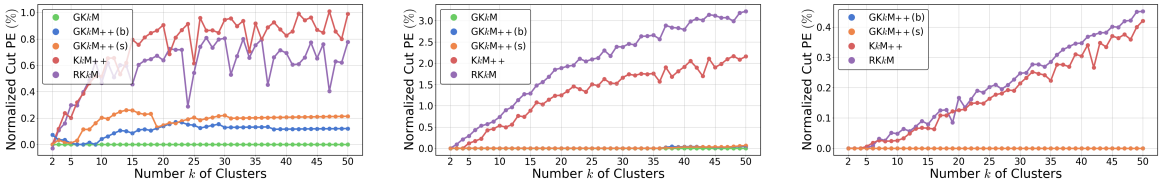
where $E(\mathcal{C}_k^\star)$ represents the clustering error of the baseline method, which we defined as GKkM, and $E(\mathcal{C}_k)$ denotes the error produced by each of the compared methods (Figs. 3.5, 3.6). However, in the case of GRQC where GKkM did not provide solutions in reasonable time due to its high computational burden, we utilized the GKkM++ with sequential sampling strategy as a baseline method.

Graph Partitioning Experimental Results



(a) email-Eu-TD1 (b) email-Eu (c) GRQC

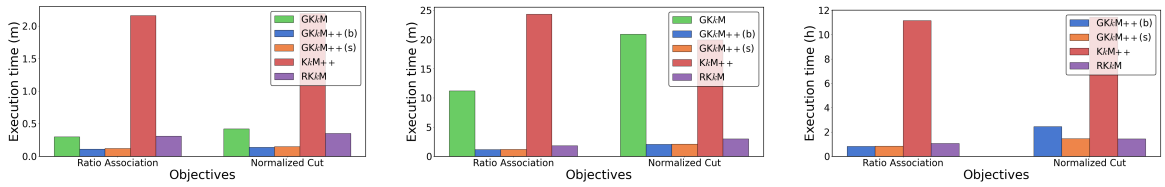
Figure 3.5: Relative Percentage Error in the ratio association objective across different graphs.



(a) email-Eu-TD1 (b) email-Eu (c) GRQC

Figure 3.6: Relative Percentage Error in the normalized cut objective across different graphs.

As it can be observed from the experimental results (Figs. 3.5, 3.6), GKkM++ produces results comparable to GKkM (with a Percentage Error smaller than 0.05 in most cases) which consistently achieves the best performance in both the ratio association and normalized cut objectives. Note that the ratio association objective is meant to be maximized (higher values are better), while the normalized cut objective should be minimized (lower values are better). GKkM failed to complete within a reasonable time on the GRQC graph due to its high computational demands, while



(a) email-Eu-TD1 (b) email-Eu (c) GRQC

Figure 3.7: CPU time comparison across different datasets and problems.

in such cases, GKkM++ produced the best outcomes. Additionally, the two GKkM++ variants consistently outperform KkM++ and RKkM. Notably, as the value of k increases, GKkM++ demonstrates progressively superior performance compared to the other two algorithms across both types of problems. This is particularly significant because higher values of k increase the complexity of optimizing the ratio association and normalized cut objectives.

In an attempt to compare KkM++ with RKkM on the ratio association objective, we observe that both algorithms exhibit similar behavior (Fig. 3.5). When addressing the normalized cut objective (Fig. 3.6), we notice that RKkM outperforms KkM++ on the email-Eu-TD1 graph. In the other two graphs, which contain a larger number of nodes and thus increase the complexity of the problem, KkM++ consistently delivers better graph partitioning results. Notably, in the email-Eu graph, as the value of k increases, the quality of KkM++ solutions becomes increasingly superior to that of RKkM.

Additionally, Figure 3.7 illustrates the CPU time of each algorithm to compute all K clustering solutions across the three graphs. Each subfigure shows the time required for each objective on the compared graphs to solve all clustering problems for $k = 1, \ldots, 50$. Specifically, we observe that regarding the ratio association objective, the GKkM++ variations require the least time in all graphs. For the normalized cut objective, GKkM++ outperforms all other algorithms on the email-Eu-TD1 and email-Eu graphs, while achieving comparable runtime performance to RKkM on the GRQC graph.

3.4.3 Real Datasets

We also conducted a series of experiments on several publicly available real-world datasets. We intentionally chosen datasets encompassing various characteristics, in-

cluding the number of samples (N), data dimensionality (d), complexity, and domain of origin. Table 3.4 provides a detailed description of each dataset.

Table 3.4: Descriptions of utilized datasets.

Dataset	Description	N	d	Source
Avila	Images of an XII century copy of the Bible	20867	10	[95]
Breast cancer	Characteristics of breast cancer tumors	569	30	[95]
Dermatology	Type of Erythematosquamous disease	366	34	[95]
Ecoli	Expression levels of proteins	336	7	[95]
Iris	Characteristics of Iris flower species	150	4	[95]
Olivetti faces	Face image dataset	400	4096	[117]
Pendigits	Handwritten digits	10992	16	[95]
Waveform-v1	Waveforms with multiple attributes	5000	21	[95]
Wine	Chemical analysis of wines	178	13	[95]

Min-max normalization in the [0, 1] range has been applied to each dataset to avoid numerical instabilities in the computations [118]. Additionally, for a more thorough investigation, we considered for each real dataset three different kernel functions, as presented in Table 3.1, which resulted in a total of 27 clustering problems.

Experimental Setup

In our experimental study on real-world datasets, we evaluate the compared methods both for the maximum number of clusters K and for all intermediate clustering solutions $k = 1, \ldots, K$. We set the maximum number of clusters $K = 50$ in all cases. To ensure a fair comparison, in each k subproblem, we executed the algorithms with $L = 100$, where L specifies the number of restarts or the number of candidates examined. In particular, for KkM++ and RKkM, we used L to control the number of restarts. In the case of GKkM++, L defines the number of candidates selected. On the other hand, GKkM is executed once, which is equivalent to running GKkM++ with the number of candidates $L = N$, where N is the total number of data instances.

Additionally, we evaluated the clustering performance using three kernel functions: Cosine, Polynomial, and RBF (Radial Basis Function), as shown in Table 3.1. For simplicity, in the Polynomial and RBF kernels, we set the γ hyperparameter for

each dataset using the following formula:

$$\gamma = \frac{1}{\sigma^2 d}, \tag{3.4}$$

where σ^2 is the variance and d the dimensionality. This equation accounts for both the variance of the data and the number of input features, ensuring an adaptive initialization of γ. This approach aligns with the standard γ initialization strategy of well-known software libraries such as scikit-learn [119]. However, more advanced techniques can also be applied [120]. Finally, for the Polynomial kernel, we set the degree deg and the coefficient c_0 to typical values, such as $deg = 3$ and $c_0 = 1$.

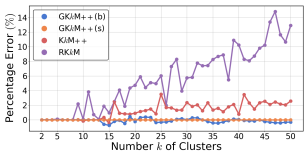
Evaluation

In this study, we propose kernel-based clustering methods that effectively minimize the feature space clustering error. Therefore, we use this error $E(\mathcal{C}_k)$ (eq. 1.9) for method comparison that provides a direct evaluation of the optimization capability of each method. More specifically, we calculate the relative Percentage Error (eq. 3.3), where $E(\mathcal{C}_k^\star)$ denotes the error corresponding to the baseline method which is GKkM (Fig. 3.8). However, due to high computational complexity, in some cases, GKkM did not provide solutions in a reasonable time. In such cases, we utilized the GKkM++ with sequential sampling strategy as the baseline method. Furthermore, for each method and k, we report the CPU execution time (Fig. 3.10) and the average number of kernel k-means iterations (Fig. 3.11). Such an evaluation approach illustrates both the error minimization capability of each method as well as its computational speed and efficiency.

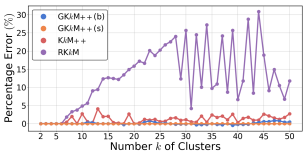
Experimental Results

GKkM++ exhibits performance comparable to that of GKkM with both batch and sequential sampling strategies across all datasets (Fig. 3.8). Notably, it clearly outperforms the GKkM method in several cases, which is more evident for large k values. It should be noted that GKkM did not terminate within a reasonable time frame in Avila, Pendigits and Waveform-v1 datasets due to its high computational complexity, where in these cases, the GKkM++ exhibited the best results. As the number of clusters k increases, the clustering problem becomes more challenging, and the performance difference between GKkM variants and the rest of the compared methods becomes

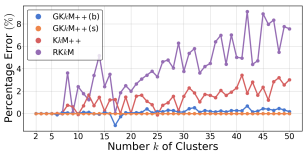
more profound. Solving the clustering problem with a larger number of clusters is crucial, as datasets inherently contain many clusters, such as Olivetti faces. Additionally, in most real-world scenarios, the number of clusters is unknown a priori. Therefore, the clustering problem should be addressed across a range of values for k, allowing us to determine the most suitable solution. In such cases, the minimization algorithm must produce good results for even larger k values.



(a) Cosine

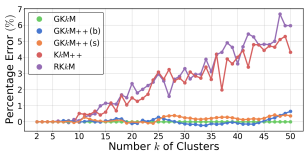


(b) Polynomial

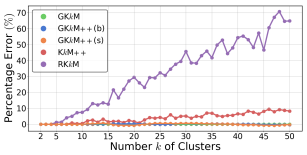


(c) RBF

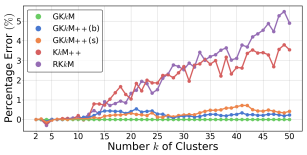
Avila



(a) Cosine

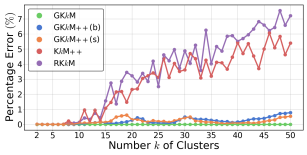


(b) Polynomial

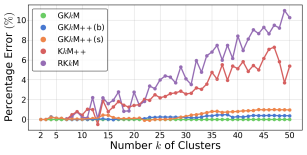


(c) RBF

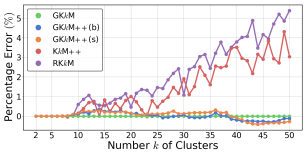
Breast cancer



(a) Cosine

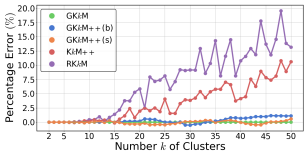


(b) Polynomial

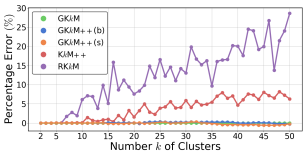


(c) RBF

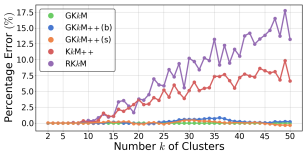
Dermatology



(a) Cosine

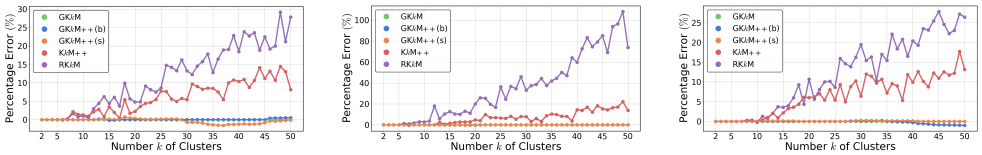


(b) Polynomial



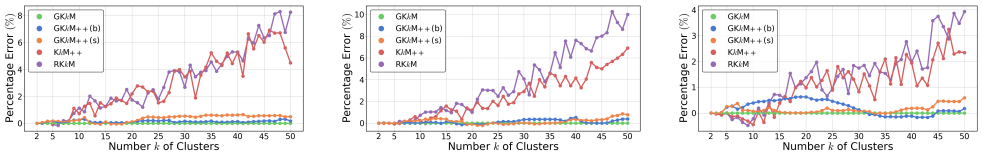
(c) RBF

Ecoli



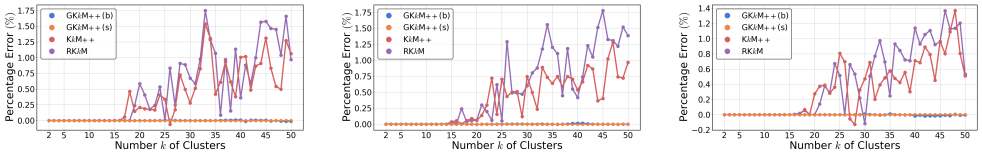
(a) Cosine (b) Polynomial (c) RBF

Iris



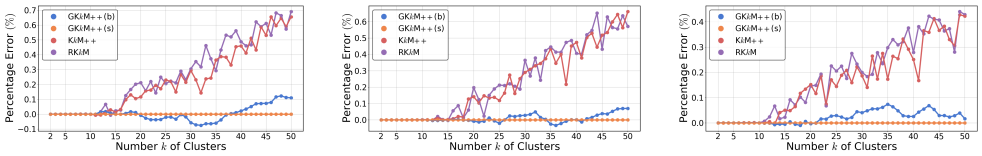
(a) Cosine (b) Polynomial (c) RBF

Olivetti faces



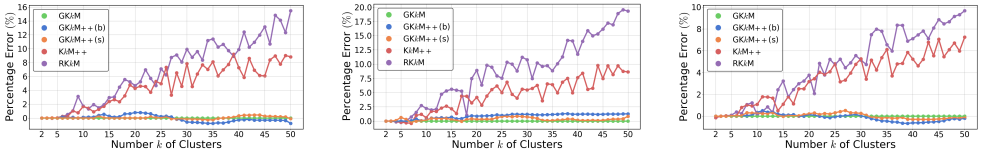
(a) Cosine (b) Polynomial (c) RBF

Pendigits



(a) Cosine (b) Polynomial (c) RBF

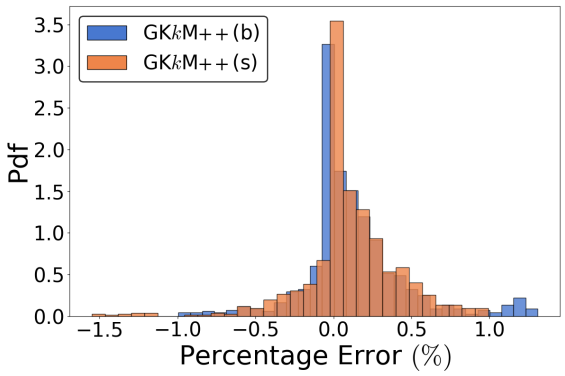
Waveform-v1



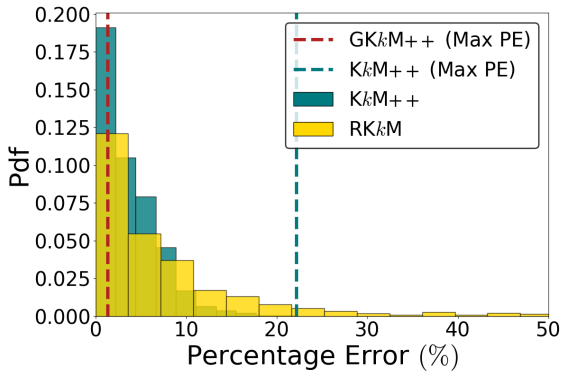
(a) Cosine (b) Polynomial (c) RBF

Wine

Figure 3.8: Comparison of the relative percentage error for each algorithm (relative to the GKkM method) across various datasets and kernel functions. Lower values indicate better clustering performance, with global optimization variants achieving the lowest error in most cases.



(a) Relative PE distribution of GKkM++, compared to GKkM, using both batch and sequential sampling strategies.



(b) Relative PE distribution of KkM++ and RKkM, compared to GKkM. The red and the blue dash lines indicate the maximum PE of the GKkM++ and KkM++ algorithms, respectively.

Figure 3.9: Distribution of relative percentage error for different clustering methods compared to GKkM. (a) Shows the performance of GKkM++ (using both batch and sequential sampling), which closely aligns with the GKkM method. (b) Compares KkM++ and RKkM, highlighting their higher error values.

Fig. 3.9 presents the relative PE of each method compared to GKkM accumulated across all $k = 2, \ldots, K$ subproblems and the 18 (out of 27) datasets where GKkM successfully converged within a reasonable time frame. Note that positive values indicate that the GKkM had superior performance, while negative values mean that the compared algorithm performed better.

In Fig. 3.9a, it is evident that GKkM++ demonstrates highly effective optimization capabilities. In most cases, GKkM++, using both batch and sequential strategies, converged to solutions with a PE of 0.5% or less. Even in the worst case, the maximum PE did not exceed 1.4% in our experiments. The plot emphasizes how closely GKkM++ approximates the performance of GKkM across various clustering subproblems. Interestingly, there are several instances where GKkM++ outperforms the solution of GKkM, and this improvement is more pronounced for the sequential sampling strategy (orange histogram) where there are cases in which the solution of GKkM++ had -1.5 PE compared to GKkM.

Moreover, it is clear that the GKkM variants consistently outperform both KkM++ and RKkM as shown in Fig. 3.9b. In the case of RKkM, the Percentage Error (PE)

reached as high as 100% (not included in the figure), whereas KkM++ exhibited a maximum PE of 22%. It should be noted that the dashed red line denotes the Maximum PE of GKkM++ algorithm. As anticipated, KkM++ significantly outperforms RKkM. However, it cannot match the optimization capabilities of the GKkM variants.

Figure 3.10 illustrates the time the CPU needs for each algorithm to compute each one of K clustering solutions for the datasets. Specifically, each subfigure presents the time required for each dataset and kernel to provide solutions for all $k = 1, \ldots, 50$. Overall, the GKkM++ variants demonstrate the highest efficiency, requiring the least execution time, often just a fraction of the time needed by the other methods. Some notable cases are the Avila, Waveform-v1 and Pendigits datasets in which the difference of GKkM++ is several days of execution compared to the second fastest algorithm. In these datasets, the GKkM failed to converge after weeks of execution due to the large number of data instances N.

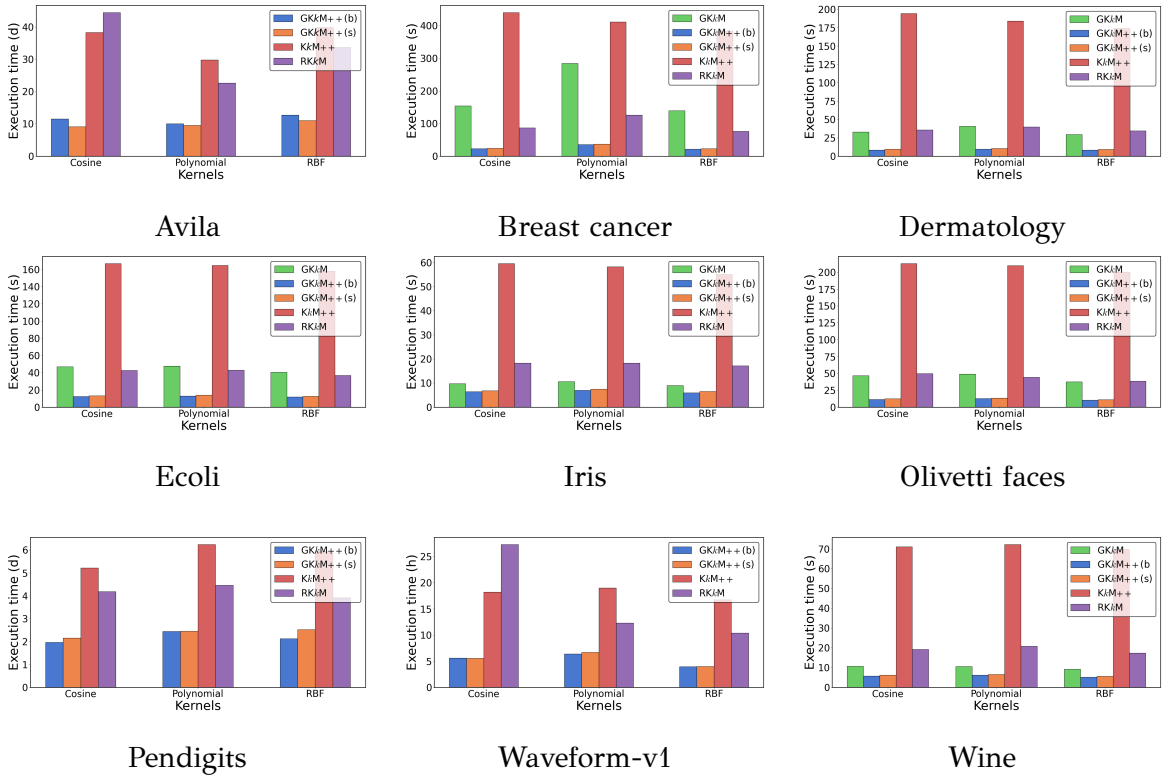
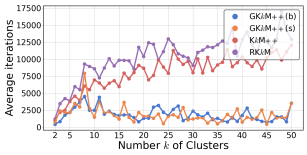


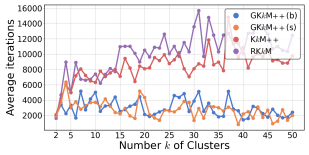
Figure 3.10: Comparison of CPU execution time required to compute all clustering solutions for different datasets and kernels. GKkM++ demonstrates significantly reduced computational cost compared to other methods.

This speedup in execution is also highlighted in Fig. 3.11, which shows the aver-

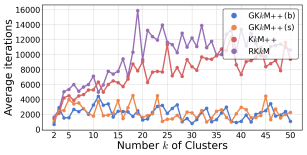
age number of kernel k-means iterations required for convergence by each method. As observed, the GKkM variants tend to require fewer iterations as k increases. This is mainly because, in each k subproblem, the $k-1$ clusters are already well partitioned. However, GKkM requires more iterations at lower values of k due to its exhaustive search nature, although its behavior is close to that of the GKkM++ variants. In contrast, KkM++ and RKkM generally require more iterations as k grows, or their iteration plateau is significantly higher than that of the global variants. This trend is particularly evident in the Avila, Breast cancer, Olivetti faces, Dermatology, Pendigits and Waveform-v1 datasets. Generally, it can be noticed that GKkM++ requires considerably fewer kernel k-means iterations in all cases.



(a) Cosine

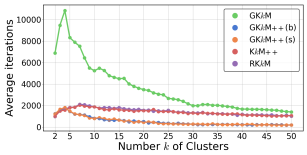


(b) Polynomial

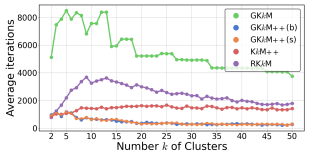


(c) RBF

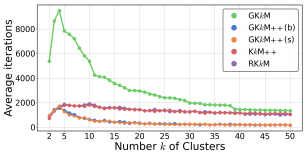
Avila



(a) Cosine

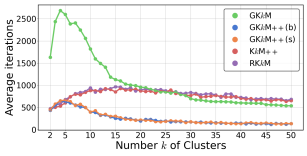


(b) Polynomial

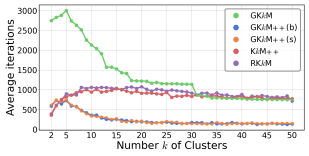


(c) RBF

Breast cancer



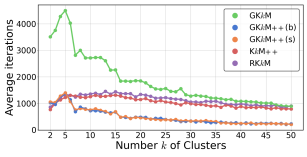
(a) Cosine



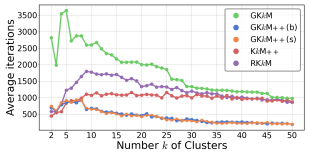
(b) Polynomial

(c) RBF

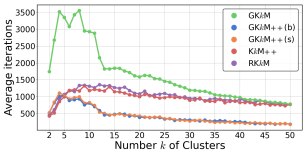
Dermatology



(a) Cosine

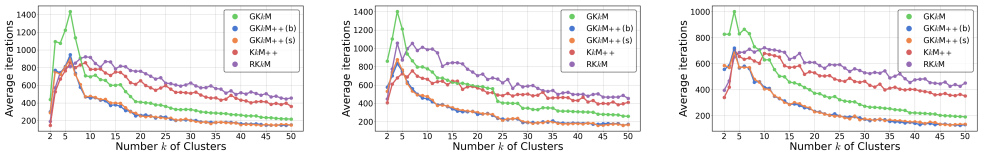


(b) Polynomial



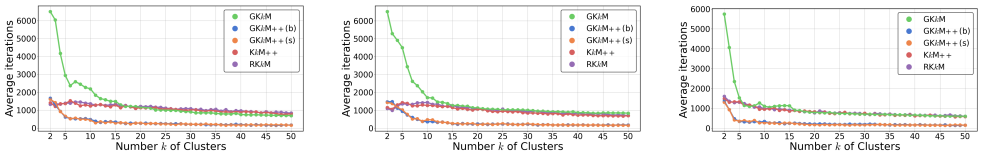
(c) RBF

Ecoli



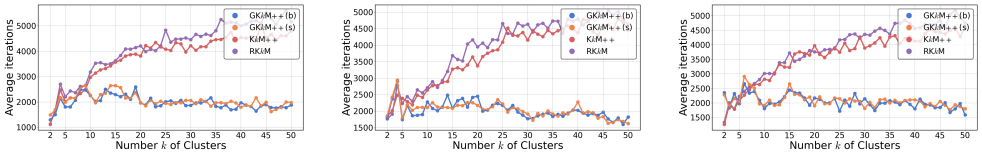
(a) Cosine (b) Polynomial (c) RBF

Iris



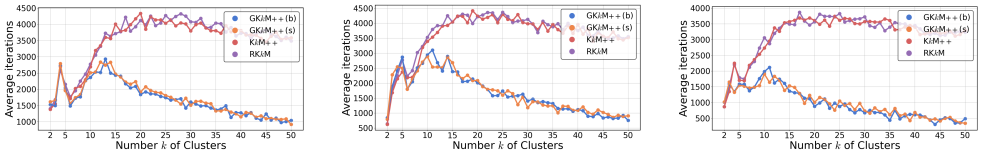
(a) Cosine (b) Polynomial (c) RBF

Olivetti faces



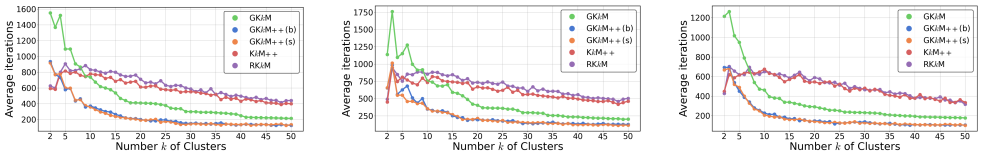
(a) Cosine (b) Polynomial (c) RBF

Pendigits



(a) Cosine (b) Polynomial (c) RBF

Waveform-v1



(a) Cosine (b) Polynomial (c) RBF

Wine

Figure 3.11: Comparison of the average number of iterations required for kernel k-means to converge across different datasets and kernel functions. GKkM++ requires fewer iterations as k increases.

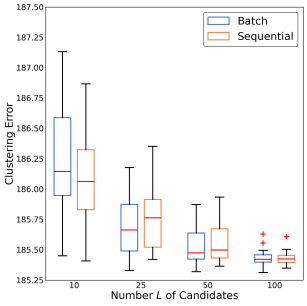
Sensitivity Analysis

In this section, we investigate the effect of the number of candidates L on the performance of the proposed method. Since L determines the number of candidates evaluated at each k-cluster subproblem, it directly influences clustering performance and computational efficiency. Specifically, a higher L value allows for a more extensive exploration of alternative solutions, increasing the possibility of selecting high-quality cluster initialization. Naturally, this comes at the cost of additional computational overhead.

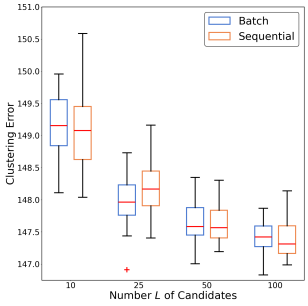
To analyze this trade-off, we conducted using various datasets and examining four values of L (10, 25, 50, 100). For each dataset and L value, 30 runs were conducted and, for each run, the clustering error attained for three different values of K (10, 25, 50) was used for our analysis. Note that both the sequential and batch sampling strategy were considered.

Figure 3.12 displays the influence of L on clustering performance, as measured by the clustering error. As expected, the increase of L leads to decrease in clustering error, since the exploration of more candidate centers improves the possibility of finding an optimal cluster placement. Additionally, as L grows, the variance of the clustering error decreases, indicating greater stability and robustness of the obtained solutions. Notably, there is no significant difference in clustering performance between the sequential and batch sampling strategies, suggesting that both selection methods are equally effective in the selected datasets.

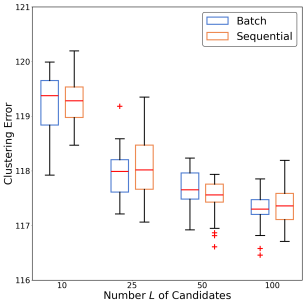
Figure 3.13 examines the impact of L on computational efficiency, measured by execution time. As expected, increasing L results in a higher computational cost due to the greater number of kernel k-means runs. It can also be observed that there is no significant difference in the execution time of the two sampling strategies.



$K = 10$

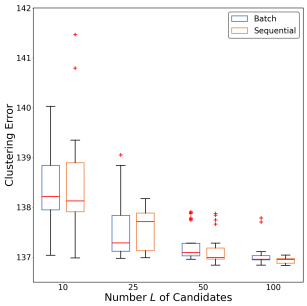


$K = 25$

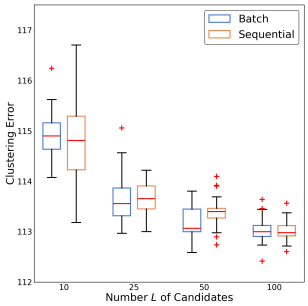


$K = 50$

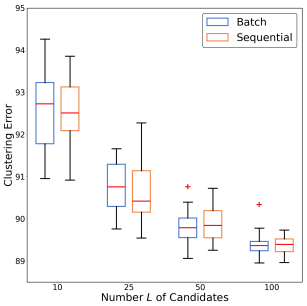
Breast cancer



$K = 10$

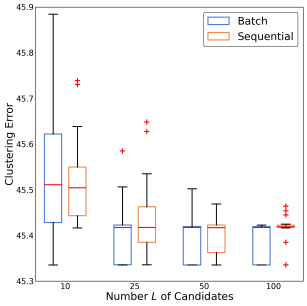


$K = 25$

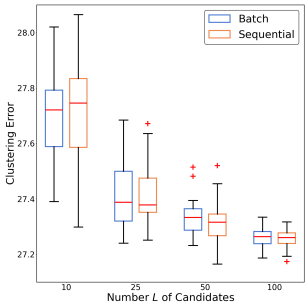


$K = 50$

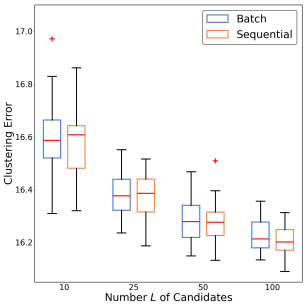
Dermatology



$K = 10$

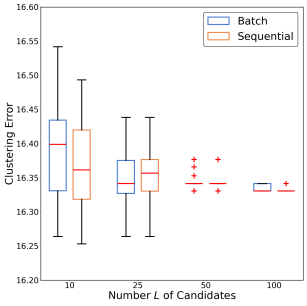


$K = 25$

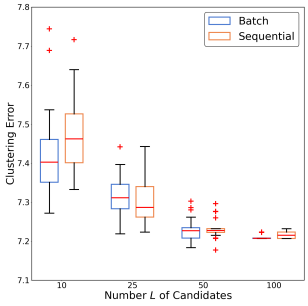


$K = 50$

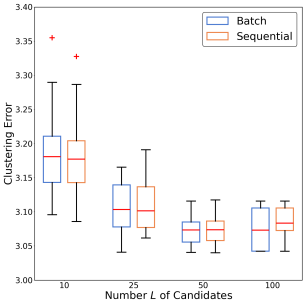
Ecoli



$K = 10$

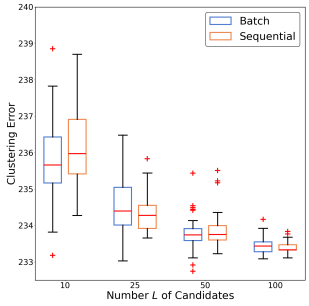


$K = 25$

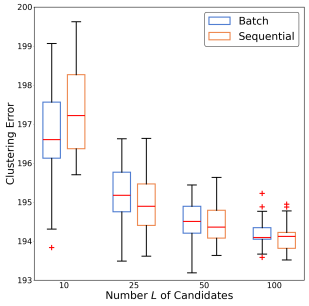


$K = 50$

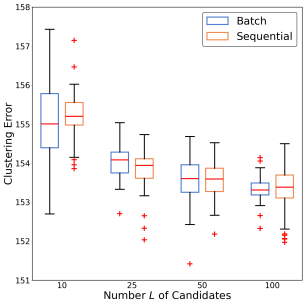
Iris



$K = 10$

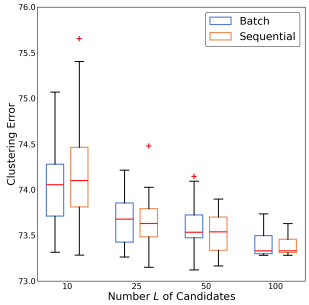


$K = 25$

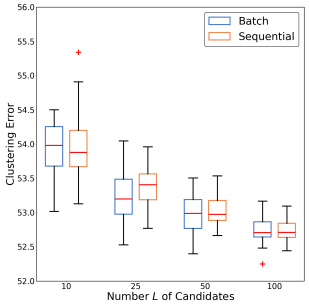


$K = 50$

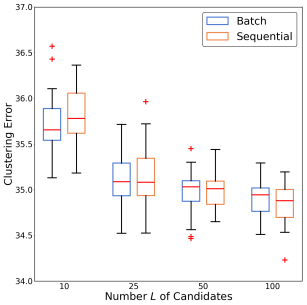
Olivetti faces



$K = 10$



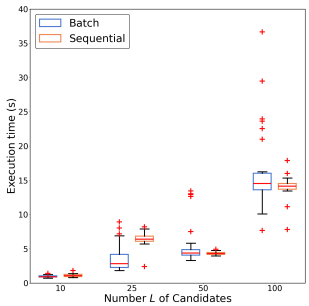
$K = 25$



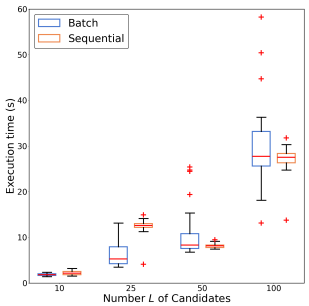
$K = 50$

Wine

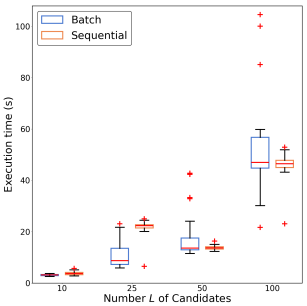
Figure 3.12: Effect of the number of candidates L on clustering performance of the proposed method for several datasets. For each dataset the clustering error statistics (over 30 runs) is presented for different values of L and number of clusters $K = 10, 25, 50$ using both the sequential and the batch sampling strategy.



$K = 10$

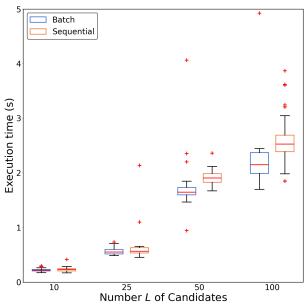


$K = 25$

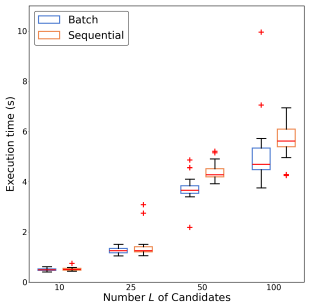


$K = 50$

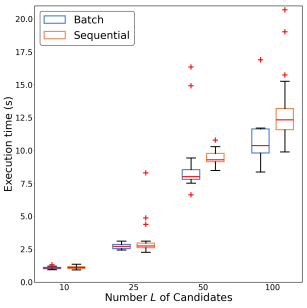
Breast cancer



$K = 10$

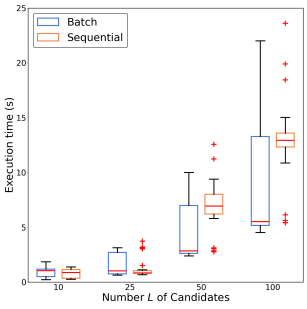


$K = 25$

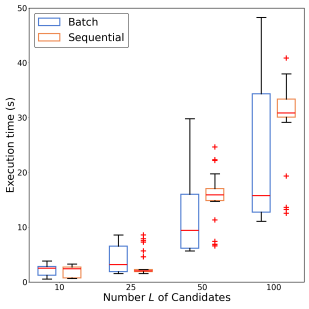


$K = 50$

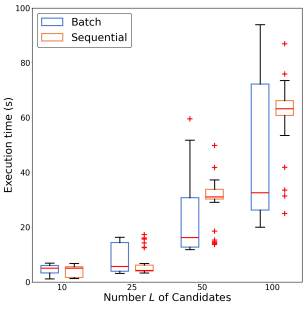
Dermatology



$K = 10$

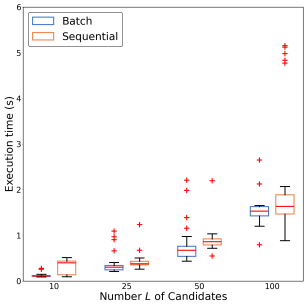


$K = 25$

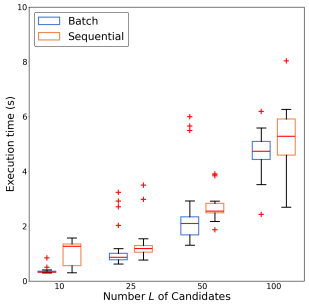


$K = 50$

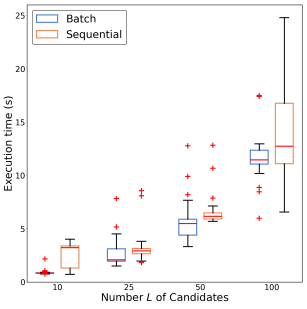
Ecoli



$K = 10$

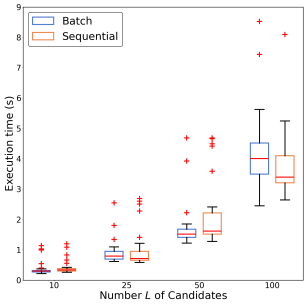


$K = 25$

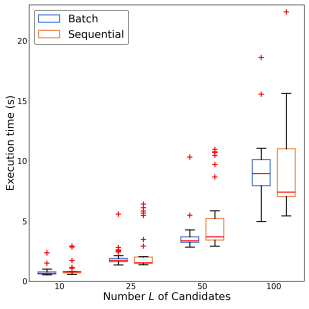


$K = 50$

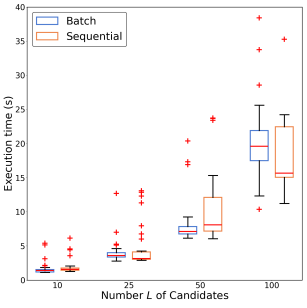
Iris



$K = 10$

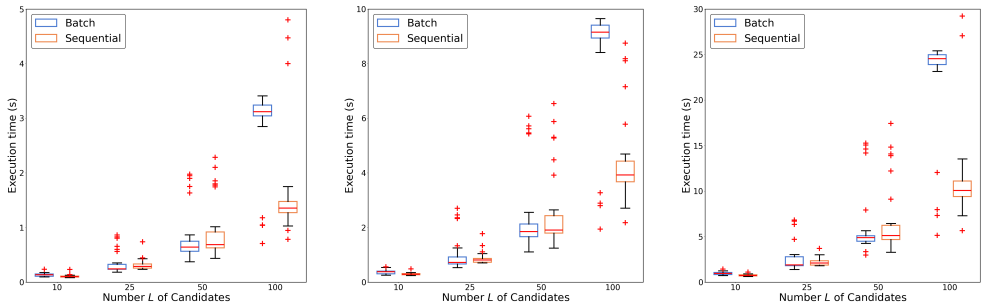


$K = 25$



$K = 50$

Olivetti faces



$K = 10$ $K = 25$ $K = 50$

Wine

Figure 3.13: Effect of the number of candidate initializations L on computational efficiency for several datasets. For each dataset the execution time statistics (over 30 runs) is presented for different values of L and number of clusters $K = 10, 25, 50$ using both the sequential and the batch sampling strategy.

3.5 Summary

In this Chapter, we introduced global kernel k-means++ (GKkM++), a novel clustering algorithm inspired by global k-means++, a method that combines the sequential clustering strategy of global k-means with the probabilistic initialization of k-means++ in Euclidean space. Our proposed GKkM++ extends this idea to a feature space using kernel methods, effectively achieving high-quality clustering results while significantly reducing computational cost.

Specifically, GKkM++ is an incremental clustering algorithm that extends the well-established global kernel k-means algorithm by incorporating the stochastic initialization strategy of kernel k-means++ to select L initial cluster candidates. To produce the solution with K clusters, it sequentially solves all intermediate subproblems for $k = 1, \ldots, K$, by sampling L initial cluster candidates at each k subproblem, where $L \ll N$. We presented two strategies for the sampling selection procedure: batch and sequential sampling. Specifically, the batch selection strategy samples L candidates at once without replacement. At the same time, sequential sampling selects L candidates one by one, also without replacement, updating the probability distribution accordingly at each sampling step. GKkM++ significantly reduces computational complexity while preserving superior minimization capabilities akin to those of traditional global

kernel k-means (GKkM), making it practical for addressing clustering problems in larger datasets, where global kernel k-means may not terminate within a reasonable time frame. Nonetheless, it is important to recognize that reducing the computational complexity of the GKkM algorithm by sampling initial cluster candidates comes at the cost of losing its deterministic nature.

We evaluated the proposed algorithm on synthetic and on several publicly available benchmark datasets and compared it to various methods, including global kernel k-means, kernel k-means++ and kernel k-means with random uniform initialization (RKkM). In all cases, GKkM++ has demonstrated its superior clustering performance and reduced computational cost. In addition, we evaluate its performance on the graph partitioning problem. Overall, the experimental results demonstrate that GKkM++ consistently achieves significantly better clustering optimization capabilities than KkM++ and RKkM. Furthermore, its performance is comparable to that of the GKkM method, with a maximum percentage error of less than 1.4% in the real datasets while achieving a maximum percentage error of less than 0.25% on the graph partitioning task. Surprisingly, in many cases, it even exceeds the performance of the GKkM. Overall, the GKkM++ variants demonstrate the highest efficiency, requiring the least execution time, often just a fraction of the time needed by the other methods.

Chapter 4

The UniForCE Algorithm for Clustering and Number of Clusters Estimation

---

4.1 Introduction

An essential question to think about before clustering a dataset is the following: *what is a meaningful cluster and how can it be represented?* The first part of the question concerns the *cluster assumption*, that is, the characteristics that a subset of data should exhibit in order to be considered a cluster. The second part, which is intertwined with the former, is how to represent mathematically a cluster of the assumed nature. *Model-based* assumptions consider a probabilistic model for each cluster, e.g. Gaussian or linear models. *Prototype-based* assumptions partition data around objects that

can be centroids [121], medoids [122], synthetic prototypes [123], or various other exemplars [124]. There are also several *density-based* assumptions, the most typical of which is the *density-level-based* that relies on several thresholds expressing the minimum local density level that a continuous region should have in order qualify as a cluster [125]. *Mode-seeking* approaches, on the other hand, use parametric density estimation to locate the area in a region where the density is maximized [126].

This Chapter [89] relates to a different type of density-based cluster assumption that focuses on the *density shape*. In most of the previous approaches, there are density shape assumptions that are either implicit (e.g. prototype-based assumptions lead to convex-shaped clusters) or consequential, but not preconditions (e.g. Gaussian mixture modeling would always fit with Gaussian components regardless of the validity of this assumption). The previously proposed explicit density shape cluster assumptions mainly concern Gaussianity [127, 128, 129]. Furthermore, the work in [55] was one of the first to introduce *unimodality*, described in Chapter 1 and Section 1.2, as an explicit assumption for clustering multivariate data. Unimodality was assessed by the proposed *dip-dist* criterion, a statistical methodology for unimodality testing of multivariate data that relies on multiple univariate unimodality tests (dip-tests) [50] performed on the distribution of pairwise distances between data points.

Once a clustering model has been selected, a clustering solution of the assumed properties is usually produced by an algorithm that optimizes an appropriate objective function. Algorithmic approaches include: k-means or expectation-maximization, agglomerative methods, incremental (divisive) algorithms that add clusters one by one, region growing or merging procedures, as well as hybrid approaches [11]. Among them, the *incremental density-shape-based* approaches have three notable advantages: i) they are robust as they employ well-founded statistical tests, ii) they offer a straightforward way to estimate automatically the number of the clusters k [127, 128, 129, 55, 56], and iii) also examine the *clusterability* of the data, since the absence of a multi-cluster structure is formally defined as the null hypothesis and can be statistically assessed [54].

Determining the number of clusters k during the optimization procedure is one of the most challenging problems in the field, especially in high-dimensions [54]. Most methods require k as input. Others claim to estimate k, but they essentially translate the problem into another, hopefully easier, problem, i.e. the tuning of their hyperparameters (e.g. [124, 126, 130, 131]). For estimating the number of clusters,

several internal cluster validation indices (CVIs) have been proposed, such as the silhouette, Davies–Bouldin, Dunn, and Calinski–Harabasz indices, among others [132]. Such CVIs value clusters that are compact and separate to each other. They define compactness in terms of intra-cluster variance, or distance of the data points to cluster centers, either of which favoring spherical or convex cluster structures. Moreover, measuring cluster separation requires at least two clusters, i.e. they are undefined for $k = 1$, and hence they cannot decide the clusterability of the input. For these reasons, they are not well-suited for guiding or evaluating clustering methods that aim to detect an unknown number of arbitrary-shaped clusters, as is the case in our work and the related literature [54].

It is important to note that an inadequate clustering assumption for a dataset makes it more likely that k will also be wrongly estimated, rendering the clustering result less informative. The existence of irregular cluster shapes is what makes most cluster assumptions fail. Typical approaches for dealing with this issue depart for the original dataspace. Spectral embeddings [133], deep data embeddings [134], hierarchical local embeddings [135, 136], or data-dependent distance metrics such as diffusion maps [137] or path-based metrics [138], they all aim at finding a new vector space to represent the data, where -hopefully- the clusters would be nicely-shaped and/or far from each other, hence typical methods will be able to recover them. It should be stressed that those lines of work overlook the discussion of cluster assumptions, while the estimation of k is usually beyond their scope. However, the divisive hierarchical method in [135] projects a data subset onto its principal direction and introduces a notion of clusterability based on contiguous clusters, i.e., sets that cannot be extended with additional data points without increasing their maximum in-set nearest-neighbor distance. It relies on criteria aiming to capture local density gaps rather than the overall shape of cluster density. Moreover, it does not offer a statistical test to assess the significance of the identified cluster structure. Finally, assuming that single clustering objectives are not sufficient, multi-objective approaches have been proposed combining objectives associated to different assumptions [139, 140].

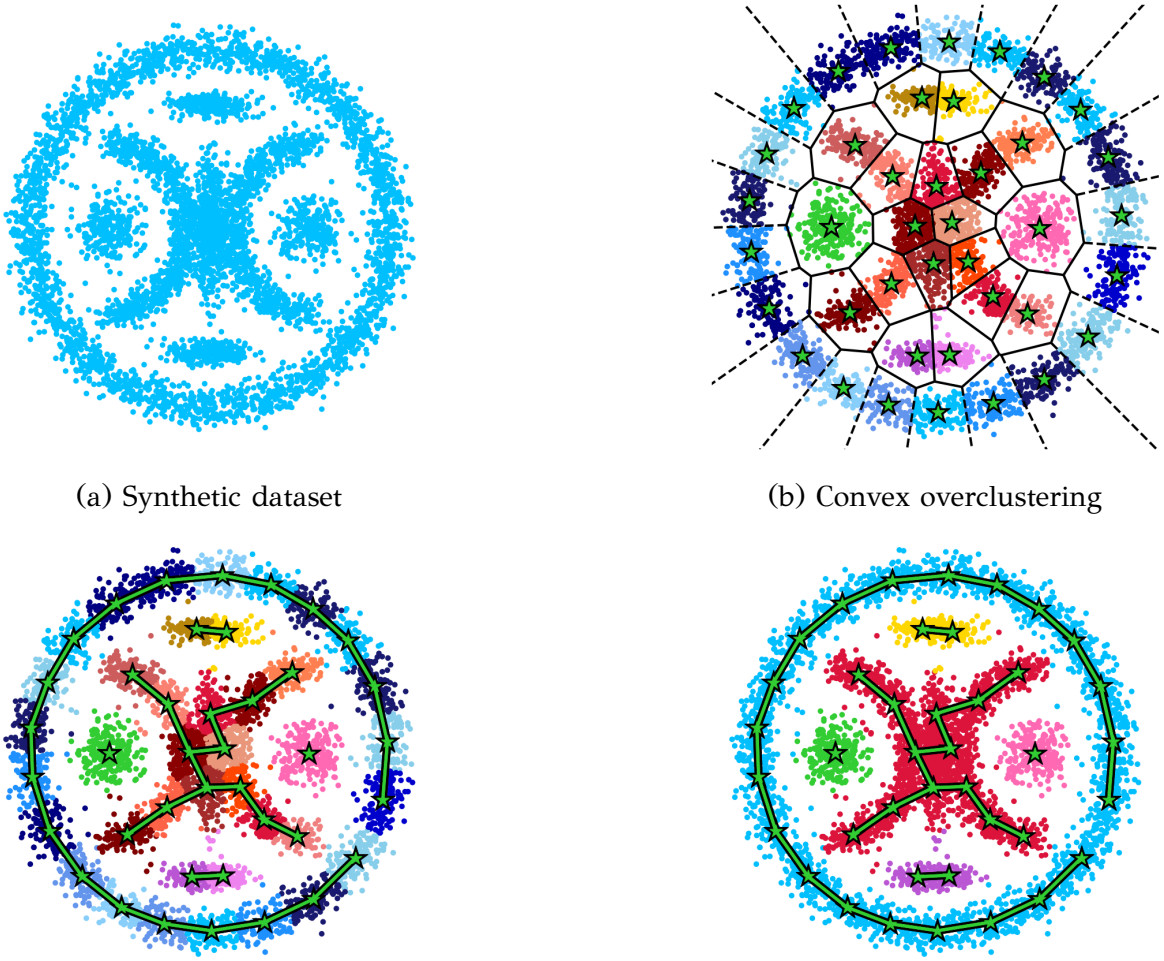
In this Chapter, we study clustering in the original dataspace, aiming at developing a clustering methodology that is flexible enough to identify irregular cluster shapes. One of the approaches that has been followed is to consider multi-prototypes as cluster representatives, and define variations of multiple-means clustering [141, 131]. Another recipe is to first employ *overclustering* to find small highly homogeneous *subclusters*,

and then try to combine them into larger and more complex cluster shapes, for instance via schemes that are density-based [142, 143, 144], a combination of the latter with graph-based approaches [145, 146], or other visualization-based schemes [147]. Agglomerating subclusters is a long-known approach, and one of the initial propositions was to use it for reducing the sensitivity and complexity of hierarchical clustering [148], but it can also be useful for discovering irregular-shaped clusters through a proper cluster linkage criterion. To some limited extent, this has also been explored using the unimodality criterion [149, 150], in application-oriented studies, and with simple merging criteria that would prevent the identification of irregular-shaped clusters.

We focus on the concept of unimodality and propose a flexible cluster definition called *locally unimodal cluster*. Such a cluster can be obtained by aggregating subclusters provided by an initial overclustering partition through a merging procedure that extends for as long as unimodality is locally preserved across pairs of subclusters. In order to examine this local property, we propose an effective statistical approach called *unimodal pair testing* that relies on the univariate dip-test for unimodality [50]. We exploit these elements to propose a cluster aggregation approach, the *Unimodality Forest for Clustering and Estimation* method (UniForCE), that boils down to: first, overclustering the dataset into small homogeneous subclusters lying in convex subregions, and then computing a *spanning forest* over the *unimodality graph* formed by the unimodal pairs of subclusters. Each spanning tree of the forest connects subclusters of the dataset that are aggregated in the same final cluster, and the number of trees is an estimate for the number of clusters. Therefore, a maximal locally unimodal cluster extends for as long as unimodality is locally preserved. This feature makes our definition flexible enough to identify typical unimodal as well as irregular-shaped clusters that are statistically significant. Fig. 4.1 illustrates the main steps of our method on a synthetic dataset. Our experimental study provides clear evidence that locally unimodal clusters can model sufficiently real and synthetic datasets, and that our algorithmic design allows the robust estimation of the number of clusters while effectively partitioning the data.

The rest of the Chapter is organized as follows. First, in Sec. 4.2, we define the locally unimodal cluster. In Sec. 4.3 we present the UniForCE clustering method and provide its computational complexity. In Sec. 4.4, we provide extensive experimental results and comparisons to real and synthetic datasets. Finally, in Sec. 4.5 we present

a discussion of the method's properties and limitations, while Sec. 4.6 summarizes the Chapter.



(a) Synthetic dataset

(b) Convex overclustering

(c) Spanning forest

(d) UniForCE clustering

Figure 4.1: **The UniForCE pipeline for locally unimodal clustering.** The steps followed by the proposed UniForCE clustering methodology are demonstrated on a synthetic dataset (Complex 2D, see Tab. 5.1 in Sec. 4.4). The input dataset is first overclustered into a large number of homogeneous subclusters lying in convex regions of the original dataspace. Then, based on pairs of subclusters that are jointly unimodal (unimodal pairs), a minimum spanning forest is computed, which provides a locally unimodal clustering with clusters as disconnected components.

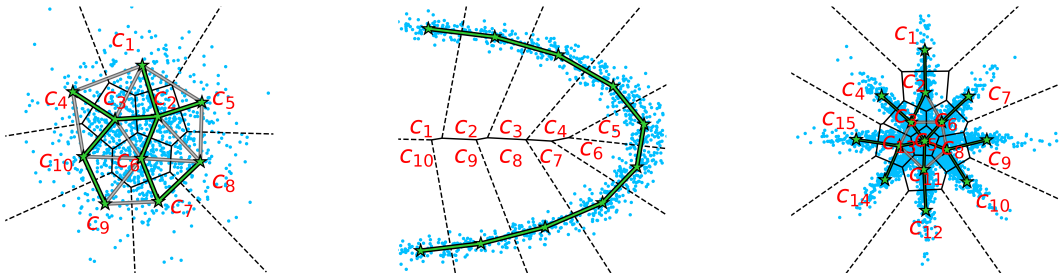


Figure 4.2: **Examples of locally unimodal clusters. a)** Spherical Gaussian density. **b)** Arc-shaped uniform density. **c)** Star-shaped density composed by 3 co-centric Gaussian ellipses. In each case, the data are overclustered in subclusters, and the computed unimodality graph includes edges (in green or gray color) between subclusters that are unimodal pairs. Any sequence of distinct subclusters corresponds to a path along which local unimodality is statistically confirmed. A spanning tree (green edges) is a subgraph of the unimodality graph that connects all the subclusters with the minimal number of edges.

4.2 Locally unimodal clusters

Our aim is to develop a clustering methodology that i) is able to capture complicated cluster structures, ii) can automatically discover the number of clusters, while at the same time iii) does not include any hard to tune hyperparameters. To this end, we introduce the concept of a *locally unimodal cluster* by formulating how unimodality extends across neighboring subregions that are part of the same larger structure. We define the locally unimodal cluster as follows:

Definition 4.1. Locally unimodal cluster. A data subset $C \subseteq X$ is a locally unimodal cluster, if there is a partition $\mathcal{C}^+ = \{c_1, \ldots, c_K\}$ of C into subclusters lying in convex subregions, such that for every pair (c_i, c_j) there exists a sequence S_{ij} of distinct subclusters, $S_{ij} = \{s_1 = c_i, s_1, \ldots, s_{n-1}, s_n = c_j\}$, where the union of any two successive subclusters $s_i \cup s_{i+1}$ is unimodal.

A clustering partition $\mathcal{C}$ of X is a *locally unimodal clustering* if every cluster of $\mathcal{C}$ is locally unimodal. The locally unimodal cluster definition is flexible enough to encompass not only typical unimodal clusters, but also arbitrary-shaped clusters. Fig. 4.2 presents examples of locally unimodal clusters, where each c_i is a subcluster

and an edge between subclusters indicates that their union is unimodal. Examining the Gaussian data density in Fig. 4.2a, we observe that there is a sequence S_{ij} between any subcluster c_i, c_j, represented as a path connecting them on the highlighted graph, e.g. $S_{1,9} = \{c_1, c_2, c_3, c_{10}, c_9\}$.

This cluster definition can be exploited in a bottom-up clustering framework that would start from an overclustering into a sufficient number of homogeneous subclusters c_i lying in convex subregions, which can be computed by a typical partitional algorithm such as the k-means. Then, a way to identify *unimodal pairs*, i.e. subcluster pairs whose union is unimodal, need to be defined. Two subclusters forming a unimodal pair are expected to lie close to each other, since the union of distant subclusters typically demonstrates a density gap that objects unimodality. The existence of unimodal pairs enables the union of small homogeneous subclusters to larger locally unimodal clusters, and this can be accomplished in a statistically sound manner. Our technique for deciding if two subclusters form a unimodal pair is presented in Sec. 4.3.

Once the initial overclustering partition is computed, we can define the corresponding *unimodality graph* having the subclusters as vertices and an edge between each unimodal pair of subclusters (see Fig. 4.2). Note that a path in the unimodality graph defines a sequence of subclusters such that successive subclusters in the sequence form unimodal pairs. We call such a path as *unimodal path*. Based on the above description, it is evident that the union of subclusters corresponding to any connected subgraph of the unimodality graph provides a locally unimodal cluster of arbitrary shape. This is due to the fact that the subgraph is connected, there exists such a unimodal path between any two subclusters. The connected components of the unimodality graph correspond to *maximal locally unimodal clusters* and define the clustering solution that provided by our method. The details of our method are described next.

4.3 Clustering based on local unimodality and the UniForCE algorithm

In this section, we present the proposed clustering methodology and an algorithm implementing it, called *Unimodality Forest for Clustering and Estimation* (UniForCE).

The methodology determines the maximal locally unimodal clusters by finding the connected components of the unimodality graph, as explained in the section Sec. 4.2. The general methodological framework is given in Alg. 4.1, and it comprises three main modules: *Overclustering*, *Unimodal pair testing*, and finally *Clustering by subcluster aggregation*. In the following subsections, we detail how UniForCE implements each of these steps.

Algorithm 4.1 The general UniForCE framework for locally unimodal clustering

Require: X (dataset)

Require: K' (number of subclusters, $K' \gg k^*$)

Require: M (minimum subcluster size)

Require: α (significance level)

1: **Overclustering**: Compute an initial overclustering of X into K' homogeneous subclusters lying in convex subregions. Eliminate small subclusters with less than M data points, and determine the final overclustering partition $\mathcal{C}^+$ into $K \leq K'$ subclusters.

2: **Unimodal pair testing**: Apply a statistical test to determine whether the union of two subclusters admits unimodality (with significance level α). This induces the unimodality graph G among the subclusters.

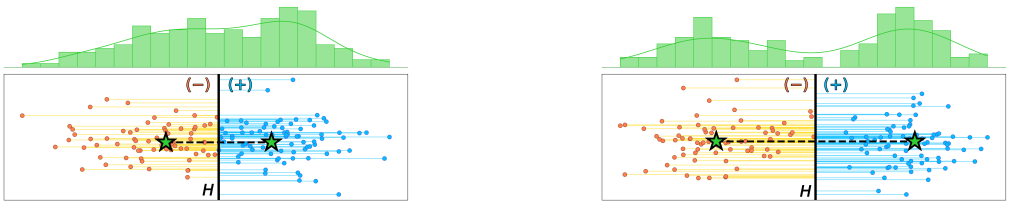
3: **Clustering by subcluster aggregation**: Compute the final clustering partition $\mathcal{C}$ by determining the connected components of the unimodality graph G of the overclustering $\mathcal{C}^+$.

4: **return** the locally unimodal clustering partition $\mathcal{C}$.

4.3.1 Overclustering

The overclustering is an essential initial step for our bottom-up strategy, since we intent examine unimodality in local data regions and then to infer the larger scale cluster structure. More specifically, the overclustering step oversegments the unknown optimal partition $\mathcal{C}^*$, which we seek to discover, in $K' \gg k^*$ homogeneous subclusters lying in convex subregions (Fig. 4.1b). Since k^* is unknown, the hyperparameter K of the method should be set to a sufficiently large value. The overclustering partition $\mathcal{C}^+$ will allow us to infer through a bottom-up aggregation the locally unimodal clusters (Fig. 4.1d).

An algorithm of the k-means family can be employed to obtain a suitable overclustering partition containing homogeneous clusters lying in convex subregions. It should be noted that for a large number of clusters, the performance of the standard k-means algorithm deteriorates, as it is merely improbable to draw a good random initialization for many centers simultaneously. To mitigate this problem, we



(a) **Unimodal case**: Two clusters forming a unimodal pair.

(b) **Multimodal case**: Two clusters forming a multimodal pair.

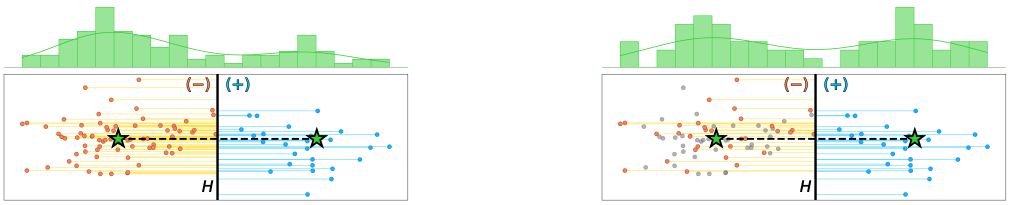
Figure 4.3: **Unimodal pair testing.** Two subclusters, c_i and c_j, appear in orange and blue, respectively, and their centers are shown as stars. The dotted line connects the two centers, while the rigged line is its perpendicular bisecting hyperplane H_{ij}. On the top, histograms present the density of the univariate set P_{ij}, containing the point-to-hyperplane signed distances, which we test for unimodality using the dip-test. **a)** Unimodal case: No density gap is observed between the subclusters, hence P_{ij} is decided as unimodal. **b)** Multimodal case: A considerable density gap is observed between the subclusters, hence P_{ij} is decided as multimodal.

use instead the global k-means++ algorithm [88], which is an incremental variant that exhibits robust clustering performance for large numbers of clusters.

Finally, in an additional preprocessing step (implemented by the function preprocess() in Alg. 5.1), subclusters containing very few data points get eliminated, and their data points get redistributed to the rest of the subclusters according to the k-means cluster assignment rule. As the subclusters increase in number, they become naturally more homogeneous, but their cardinality reduces. Since statistical testing lies at the heart of our methodology, namely the dip-test of unimodality [50], we need to enforce a minimum allowed subcluster size, let that be M, to ensure the validity of the test. Empirical evidence from prior work suggests that a threshold to keep the dip-test more reliable is the sample size to be greater that 50 data points [151]. As we describe in detail next, our method tests pairs or subclusters whether they are jointly unimodal, which justifies setting the minimum subcluster size to $M = 25$ data points.

4.3.2 Unimodal pair testing

A major step in our algorithm is the unimodal pair testing procedure that decides whether the union of two subclusters $c_{ij} = c_i \cup c_j$, c_i, $c_j \in \mathcal{C}^+$, results in a unimodal cluster. We exploit the fact that we know the centers μ_i and μ_j of the subclusters



(a) **Imbalanced clusters**: 75/25 size ratio (p-value ≈ 0.28).

(b) **Balanced clusters**: 25/25 size ratio (p-value ≈ 0.02).

Figure 4.4: **The imbalanced modes problem and our subsampling solution.** In this example, the subcluster pair has originally a size ratio of 75/25. **a)** The unimodal pair test fails to reject unimodality when all the data are considered. **b)** The problem is tackled when testing for unimodality the balanced subsample (data points shown in gray are discarded).

tested for merging. We first define the vector $r_{ij} = \mu_j - \mu_i$ connecting these centers, and the perpendicular bisecting hyperplane H_{ij} to r_{ij} passing through its midpoint. Formally, $H_{ij} : w_{ij}^\top x + b_{ij} = 0$, where $x \in \mathbb{R}^d$ is an input data point, w_{ij} is the vector of coefficients of the hyperplane, and b_{ij} is the intercept. More specifically, the equation of H_{ij} is:

$$(\mu_j - \mu_i)^\top x - \frac{1}{2}(\mu_j - \mu_i)^\top(\mu_i + \mu_j) = 0, \tag{4.1}$$

and the signed distance of a data point x to H_{ij} is given by:

$$d(x, H_{ij}) = \frac{(\mu_j - \mu_i)^\top x - \frac{1}{2}(\mu_j - \mu_i)^\top(\mu_i + \mu_j)}{||\mu_j - \mu_i||}. \tag{4.2}$$

Let P_{ij} the set that contains the values $d(x, H_{ij})$ for every $x \in c_{ij}$. We apply the Hartigans' dip-test [50] for unimodality to P_{ij} to decide whether it is unimodal with regards to a statistical significance level α. Illustrations of a successful and an unsuccessful unimodal pair test (when it decides for unimodality, and multimodality, respectively) are provided in Fig. 4.3. It is clear that if there is a density gap between the two subclusters, then the signed distances in P_{ij} will be multimodal, and therefore the dip-test is expected to reject unimodality.

The identification of imbalanced modes is a challenging aspect [151]. In our context, the dip-test employed in the unimodal pair test may fail to reject unimodality when one of the two subclusters is significantly smaller, even when the two associated modes are quite well-separated. To address this issue, our approach is to use a balanced subsample from the two tested subclusters. Specifically, all the data points

Algorithm 4.2 Unimodality pair test for two subclusters

Require: c_i, c_j (two subclusters)
Require: $L \leftarrow 11$ (odd number of Monte Carlo simulations)
Require: $\alpha \leftarrow 0.001$ (significance level)
if $|c_i| > |c_j|$ **then**
  Swap the indices $i \leftrightarrow j$
end if
Compute the centers of μ_i, μ_j of the two subclusters
Find the perpendicular bisecting hyperplane H_{ij} to the vector $r_{ij} = \mu_j - \mu_i$ connecting the two centers *// Eq. 4.1*
Compute the set P_i with the signed distances from H_{ij} for all samples in the smaller cluster c_i *// Eq. 4.2*
Initialize v as a zero vector with L elements *// Votes for unimodality*
for all $l \in [L]$ **do**
  Create c'_j by sampling $|c_i|$ elements uniformly at random without replacement from the larger cluster c_j
  Compute the set P_j with the signed distances from H_{ij} for all samples in c'_j *// Eq. 4.2*
  $P_{ij} \leftarrow P_i \cup P_j$ *// The set of all signed distances of $c_i \cup c'_j$ from H_{ij}*
  $p \leftarrow$ dip-test(P_{ij}) *// The p-value of the Hartigans' dip-test for unimodality*
  $v[l] \leftarrow \mathbb{1}\{p \geq \alpha\}$ *// Store the vote against or for unimodality, 0 or 1 respectively*
end for
$m \leftarrow \mathbb{1}\{\sum_{l=1}^{L} v[l] > \frac{L}{2}\}$ *// Compute the majority vote, either 0 or 1*
return m

are used from the smaller subcluster, let that be c_i, and a uniform random subsample $c'_j \subset c_j$ of equal size is drawn from the larger subcluster to produce the balanced set $c'_{ij} = c_i \cup c'_j$, $|c'_{ij}| = 2|c_i|$. Thus, the set P_{ij} will contain the signed distances to that hyperplane H_{ij} of only the data points of c'_{ij}, and this will be tested as before with the dip-test for unimodality. Note that the hyperplane H_{ij} is computed once using all the data and does not depend on subsampling. Fig. 4.4 illustrates this problem of wrongly accepting unimodality due to the imbalance of the subclusters in our specific context, and our workaround.

To account for imbalanced subcluster sizes, this procedure is repeated (Monte Carlo experiment) for an odd number of times L, and decide the success or failure of the unimodal pair test based on the majority of the results. The detailed algorithm for the unimodal pair test is presented in Alg. 4.2.

4.3.3 Finding connected components

The application of the unimodal pair testing procedure on subcluster pairs provides the unimodality graph of the initial overclustering partition. Finding the connected components of the unimodality graph is the next step in our methodology. We choose to represent each connected component by a spanning tree, which is the minimum structure required. Each spanning tree of the unimodality graph, called *unimodal span-*

ning tree, represents a maximal locally unimodal cluster, and the *unimodality spanning forest* provides the overall clustering partition. The UniForCE algorithm computes the unimodality spanning forest and uses an online graph construction procedure to minimize the number of the required unimodality tests. Specifically, we make two relaxations that make the computation much more efficient without affecting the clustering result. First, we remark that any spanning tree (not necessarily the minimum one) connecting the same set of subclusters of a given overclustering would produce the same clustering partition. Second, since unimodality extends locally across neighboring subclusters, we can use the proximity of between pairs of subclusters as a preference for the order in which pairs shall get tested for unimodality.

The exhaustive computation of the unimodality graph for K subclusters would require $\frac{K(K-1)}{2}$ unimodality tests. In order to reduce the computational cost, we use a simpler strategy by exploiting the above-mentioned preference order for testing pairs of (closely) neighboring subclusters. We start with a complete distance graph G, whose vertices are the subclusters and the edge weights W are the $\frac{K(K-1)}{2}$ pairwise Euclidean distances (using other alternatives adapted to the data is possible) between the centers of the subclusters. Then, we consider the proximity of two subclusters as an indicator for the possibility that unimodality extends across those subclusters, hence we test pairs of subclusters in pairwise proximity order.

The unimodal spanning forest approximation F is computed over G by also testing for spanning unimodality between pairs of vertices. Algorithmically, we compute F using a modification of the classical Kruskal's algorithm [152]. Initially, F consists of K trees, each with only one vertex. The edges of G, sorted in ascending weight order, indicate the order in which unimodality between pairs of vertices should be tested. When a test for a pair of vertices is successful, we add an edge in F connecting those vertices, and the number of trees (also clusters of the partition) is reduced by 1. While traversing this list of edge weights, we skip pairs of vertices that are already in the same spanning tree of F. Operating in the described way minimizes the number of unimodality tests that need to be performed without affecting the final clustering result.

Algorithm 4.3 The UniForCE algorithm for clustering and estimation of the number of clusters

Require: X (dataset)
Require: $K' \leftarrow 50$ (number of subclusters)
Require: $M \leftarrow 25$ (minimum subcluster size)
Require: $L \leftarrow 11$ (odd number of Monte Carlo Simulations)
Require: $\alpha \leftarrow 0.001$ (significance level)
$\{\mathcal{C}^+ = \{c_i\}_{i\in[K]}, \mu = \{\mu_i\}_{i\in[K]}\} \leftarrow$ preprocess(global k-means++$(X, K'), M)$ *// Overclustering into K subclusters, with more than M data points each*
Consider the subclusters' centers $\{\mu_i\}$ as graph vertices $\{V_i\}$, $i \in [K]$
Compute the distance graph G with edge weights $W_{ij} = \text{dist}(\mu_i, \mu_j)$, $i, j \in [K]$ *// dist$(\cdot,\cdot) \leftarrow$ Euclidean distance*
Initialize the unimodality spanning forest F with K singleton trees, one for each $V_i \in G$, $i \in [K]$
for each edge $(V_i, V_j) \in G$ in ascending order of distance W_{ij} **do**
    if belongInDifferentTrees(F, V_i, V_j) **and** isUnimodalPair(c_i, c_j, L, α) **then**
        Add the edge (V_i, V_j) in the unimodality spanning forest F
    end if
end for
for each unimodal spanning tree $T_j \in F$ **do**
    Create the cluster C_j with the vertices $V_i \in T_j$ *// Gather all data points of those subclusters*
end for
return the locally unimodal cluster partition $\mathcal{C} = \{C_1, \ldots, C_{|F|}\}$, and the estimated $k = |F|$

4.3.4 Complexity analysis

UniForCE, as all methods that rely on the dip-test, can benefit from a dramatic acceleration of the statistical tests. Instead of computing the dip statistics, we can make use of a lookup table with precomputed bootstrap dip statistics of Uniform samples[1], over a grid of sample sizes and significance levels. The grid needs to be sufficiently dense for the scale of the treated problem. Other approaches to accelerate the dip-test have appeared in the literature [153], offering directions for further refinements.

For the overclustering step, a computationally cheap choice is to use k-means++ [29]. However, in Sec. 4.3.1, we justified the use of the global k-means++ algorithm [88] by the fact that it takes $\mathcal{O}(QKNd)$ time, where Q is the number of candidates examined per incremental iteration to initialize the new cluster, K is the desired number of subclusters, N is the size of the dataset, and d is the dimensionality of the data. The number of candidates Q should be $\mathcal{O}(1)$, e.g. between 10 and 20. Thus, the overclustering step takes $\mathcal{O}(KNd)$ time. Postprocessing by removing very small subclusters from the overclustering partition and reassigning their elements takes $\mathcal{O}(KN)$ time.

In the subsequent analysis, we delve into the overall time complexity associated with the invocations of the isUnimodalPair function. The cost to compute the dip

[1] In our implementation we used the repository that is available at: https://pypi.org/project/diptest/.

Table 4.1: **The real datasets used in the experiments.** N is the number of data instances, d is the dimensionality, and k is the number of labeled classes (i.e. the ground-truth k^*). With '*', we mark an embeddings dataset obtained by training an autoencoder on the original dataset. Several of the used real datasets come from the UCI machine learning repository.

Dataset	Type	Description	N	d	k	Source
EMNIST Balanced Digits *	Vector	Handwritten digits	28000	10	10	[154]
EMNIST Balanced Letters *	Vector	Handwritten letters (A-J)	28000	10	10	[154]
EMNIST MNIST *	Vector	Handwritten digits	70000	10	10	[154]
HAR *	Vector	Sensor data from smartphones	10299	10	6	[95]
Isolet	Spectral	Speech recordings pronouncing letters	7797	617	26	[95]
Mice Protein Expression	Tabular	Expression levels of proteins	1080	77	8	[95]
Optdigits	Image	Handwritten digits	5620	8×8	10	[95]
Pendigits	Timeseries	Handwritten digits	10992	16	10	[95]
TCGA	Tabular	Cancer gene expression profiles	801	20531	5	[95]
Waveform-v1	Vector	Waveforms with multiple attributes	5000	21	3	[95]
YTF *	Vector	Face images from videos	2000	10	40	[155]
Complex 2D (synthetic)	Vector	Multiple structures inside a ring (Fig. 4.1)	5000	2	6	ours

statistic of a dataset of size n using Hartigans' dip-test is $\mathcal{O}(n)$ [50], provided that the values are sorted. However, since sorting is necessary, the time complexity for calling once the isUnimodalPair() function is $\mathcal{O}(n \log n)$. We can show that the total time complexity of computing the dip statistic for all unimodal pairs is of $\mathcal{O}(N \log N)$:

$$\sum_{\substack{i,j \in [K] \\ i<j}} (|V_i| + |V_j|) \log(|V_i| + |V_j|) \leq \Big(\sum_{i,j \in [K]} (|V_i| + |V_j|) \Big) \log N$$

$$\leq 2N \log N.$$

Note that the number of tests L is supposed to be $\mathcal{O}(1)$, e.g. between 1 and 11. The computation of the spanning forest F takes $\mathcal{O}(K^2 \log K)$ time. Constructing the final clustering takes $\mathcal{O}(N)$ time. Thus, the *total time complexity* of UniForCE algorithm is $\mathcal{O}(KNd + KN + N \log N + K^2 \log K)$.

4.4 Experimental evaluation

4.4.1 Experimental setup

Datasets. Tab. 5.1 summarizes the datasets that we used for experimental evaluation, which vary in size N, dimensions d, number of clusters k (this is the number of

labeled classes that we consider as the ground-truth value k^*), data type, and domain of origin.[2]

The datasets Optigits, Pendigits, EMNIST MNIST (EMNIST-M), and EMNIST Balanced Digits (EMNIST-BD) comprise handwritten digits, with 10 classes corresponding to the digits from 0 to 9. Optigits consist of images with a resolution of 8×8, while EMNIST-M and EMNIST-BD contain images with a higher resolution of 28×28. In contrast, Pendigits' data instances are represented by 16-dimensional vectors containing pixel coordinates. EMNIST-BL is a dataset with handwritten letters, with capital and non-capital characters, from which we selected the 10 classes corresponding to the letters A to J, that account for 28000 data points. The Isolet dataset is a collection of speech recordings containing the sound samples of spoken letters, represented by vectors of 617 spectral coefficients extracted from the speech signal. The TCGA is a collection of gene expression profiles obtained from RNA sequencing of various cancer samples. It includes 801 data instances, clinical information, normalized counts, gene annotations, and 6 cancer types' pathways. The Mice Protein Expression dataset consists of the expression levels of 77 proteins/protein modifications that produced detectable signals in the nuclear fraction of the cortex. It includes 1080 data points and 8 eight classes of mice based on genotype, behavior, and treatment features. The Waveform-v1 consists of 3 classes of generated waves with 5000 data points. Each class is generated from a combination of 2 of 3 'base' waves. The Human Activity Recognition (HAR) dataset consists of data recorded from smartphone accelerometers and gyroscopes as participants performed various activities such as walking, sitting, and standing. Each instance consists of a 560-dimensional feature vector. The dataset contains 10299 instances categorized into 6 activity classes. The YouTube Faces (YTF) dataset consists of face images extracted from videos of a wide range of individuals. For our subset, we randomly selected 40 individuals and sampled 50 images per person, yielding a total of 2000 face images.

EMNIST is an extended and more challenging MNIST dataset. Due to the high complexity of the three EMNIST versions and the YTF dataset, we used these datasets after creating a high-quality data embedding via an Autoencoder (AE). The architecture of the convolutional AE is highly used in literature for clustering purposes [156]. Specifically, the encoder part consists of 3 convolutional layers with channel numbers 32, 64, and 128, and kernel sizes of 5×5, 5×5, and 3×3, respectively. This is followed

[2]The UCI datasets are available at: https://archive.ics.uci.edu/datasets.

Table 4.2: **Summary of the experimental results.** The best values per dataset are shown in bold. Cases marked by † and ‡ indicate experiments that failed due to memory/time and method constraints, respectively.

	EMNIST-BD			**EMNIST-BL**			**EMNIST-M**			**YTF**		
Methods	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
X-means	357±10	0.37	0.06	286± 9	0.37	0.07	576±15	0.35	0.04	†	†	†
G-means	120± 5	0.44	0.15	139±11	0.41	0.12	265± 8	0.39	0.07	†	†	†
PG-means	42± 4	0.58	0.38	44± 4	0.56	0.34	48± 5	0.58	0.37	8±6	0.20	0.20
dip-means	7 or 8	0.60	0.53	5± 0	0.60	0.48	9± 0	0.75	0.72	8±0	0.43	0.15
pdip-means	7± 0	0.55	0.45	3± 0	0.45	0.23	5± 0	0.61	0.44	6±0	0.39	0.14
Mean Shift	†	†	†	†	†	†	†	†	†	49±0	0.92	0.76
HDBSCAN	12± 0	0.40	0.11	3± 0	0.02	0.01	**10**± 0	0.45	0.19	50±0	0.91	0.83
SMMP	†	†	†	†	†	†	†	†	†	28±0	0.88	0.77
RCC	82± 0	0.74	0.63	82± 0	0.52	0.35	†	†	†	221±0	0.65	0.50
UniForCE	**10**± 1	**0.87**	**0.87**	**12**± 1	**0.74**	**0.69**	13± 1	**0.84**	**0.85**	**39**±1	**0.94**	**0.89**
Ground-truth	10	–	–	10	–	–	10	–	–	40	–	–

	Optdigits			**Pendigits**			**Isolet**			**Waveform-v1**		
Methods	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
X-means	422± 9	0.34	0.05	1472±19	0.25	0.02	233± 4	0.49	0.16	10±0	0.32	0.22
G-means	57± 5	0.53	0.29	184±11	0.44	0.15	101± 6	0.57	0.33	12±0	0.31	0.21
PG-means	‡	‡	‡	25± 3	0.58	0.45	‡	‡	‡	4±1	0.45	0.50
dip-means	4± 0	0.38	0.27	**9**± 1	0.62	0.35	4± 0	0.30	0.14	4±0	0.42	0.29
pdip-means	12± 1	0.69	0.56	12± 1	0.69	0.43	16± 1	0.52	0.34	1±0	0.00	0.00
Mean Shift	73± 0	0.63	0.66	17± 0	0.65	0.51	†	†	†	11±0	0.81	0.91
HDBSCAN	9± 0	0.48	0.26	28± 0	0.69	0.54	3± 0	0.02	0.01	4±0	0.86	0.94
SMMP	13± 0	0.71	0.69	34± 0	0.25	0.13	6± 0	0.13	0.02	56±0	0.34	0.25
RCC	19± 0	**0.87**	**0.89**	46± 0	0.75	0.75	12± 0	0.53	0.20	**3**±0	**1.00**	**1.00**
UniForCE	**11**± 1	0.85	0.80	17± 1	**0.78**	**0.76**	**27**± 2	**0.71**	**0.41**	**3**±0	**1.00**	**1.00**
Ground-truth	10	–	–	10	–	–	26	–	–	3	–	–

	HAR			**TCGA**			**Mice Protein**			**Complex 2D**		
Methods	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
X-means	1536±20	0.15	0.01	20± 1	0.51	0.32	244± 2	0.27	0.05	1±0	0.00	0.00
G-means	99± 5	0.32	0.09	‡	‡	‡	32± 1	0.71	0.75	96±1	0.28	0.04
PG-means	2± 1	0.40	0.16	†	†	†	‡	‡	‡	23±2	0.42	0.23
dip-means	†	†	†	2± 0	0.36	0.24	5± 0	0.67	0.05	26±1	0.32	0.16
pdip-means	†	†	†	‡	‡	‡	9 or 10	**0.96**	**0.96**	6±0	0.37	0.23
Mean Shift	13± 0	0.60	0.49	†	†	†	6± 0	0.73	0.66	16±0	0.33	0.19
HDBSCAN	9± 0	0.52	0.39	5± 0	0.55	0.35	11± 0	0.93	0.95	7±0	0.90	0.95
SMMP	22± 0	0.16	0.08	4± 0	0.72	0.65	11± 0	0.43	0.21	10±0	0.07	0.06
RCC	118± 0	0.56	0.43	8± 0	0.84	0.85	54± 0	0.52	0.32	498±0	0.09	0.01
UniForCE	**6**± 1	**0.62**	**0.53**	**5 or 6**	**0.93**	**0.94**	**8**± 0	0.93	0.91	**6**±0	**0.98**	**0.99**
Ground-truth	6	–	–	5	–	–	8	–	–	6	–	–

by a two-layer MLP with 384 and 10 neurons, respectively. The decoder part of the AE is symmetrical with the encoder. LeakyReLU activates all intermediate layers of the AE with a slope equal to 0.1. We trained the AE for 100 epochs using the Adam optimizer with a constant learning rate of 0.001, batch size of 256 and with the default setting of $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Additionally, for the HAR dataset, we employed a widely used autoencoder architecture for feature extraction, following designs commonly adopted in the literature [73]. Specifically, the encoder consists of three fully connected layers with 500, 500, and 2000 neurons, respectively, followed by a latent space of 10 neurons. The decoder mirrors the encoder architecture symmetrically. All intermediate layers use LeakyReLU activation with a negative slope of 0.1. The autoencoder was trained using the same procedure described previously.

For the Mice Protein Expression dataset, we applied one-hot encoding to manage categorical values and to address the few missing data; we imputed the missing values by utilizing the mean values for each column. As a preprocessing step, we used min-max normalization to map the attributes of each dataset to the $[0, 1]$ interval to prevent attributes with large value ranges from dominating the distance calculations, and to also avoid numerical instabilities in the computations.

The 2-dimensional Complex 2D is the only synthetic dataset that we use in the first experimental part. It contains multiple clusters inside a ring (see Fig. 4.1), some of which are non-convex and pairwise non-linearly separable. More additional experiments on synthetic datasets are presented in Sec. 4.4.4.

Compared clustering methods. The performance of the UniForCE algorithm is compared with several methods that perform clustering and automatic estimation of the number of clusters.[3] The most related category of methods are those using statistical tests: X-means [127], G-means [128], PG-means [129], dip-means [55], and projected dip-means [56] (pdip-means). Additionally, we considered approaches that do not rely on statistical tests in their optimization procedure, such as HDBSCAN [130], RCC [157], SMMP [143], and Mean Shift [158]. HDBSCAN is a method that performs DBSCAN over varying epsilon values and integrates the results to find a clustering partition that gives the best stability over epsilon. Since, by design, we stay in the original data space, we do not consider approaches that integrate embedding and clustering, such as deep clustering methods. In all experiments, we fixed

[3]Machine specifications: Intel® Core™ i7-8700 CPU at 3.20 GHz and 16 GB of RAM.

our hyperparameters to $K = 50$ (number of initial subclusters), $\alpha = 0.001$ (statistical significance level), $M = 25$ (minimum subcluster size), and $L = 11$ (number of Monte Carlo simulations).

Evaluation measures. For evaluating how well a clustering partition matches the ground-truth label information, we compute the *Adjusted Mutual Information* (AMI) [159] measure defined as:

$$\text{AMI}(Y, C) = \frac{I(Y, C) - \mathbb{E}[I(Y, C)]}{\max\{H(Y), H(C)\} - \mathbb{E}[I(Y, C)]},$$

where Y denotes the vector of ground-truth labels, C denotes the vector of cluster labels produced by a clustering algorithm, I is the Mutual Information measure, H the entropy of a partition (either the ground-truth or the produced one), and $\mathbb{E}[\cdot]$ is the expected value. We also compute the *Adjusted Rand Index* (ARI) [160] measure as follows:

$$\text{ARI}(Y, C) = \frac{\text{RI}(Y, C) - \mathbb{E}[\text{RI}(Y, C)]}{\max(\text{RI}) - \mathbb{E}[\text{RI}(Y, C)]},$$

where RI is the Rand Index measuring the fraction of agreement between Y and C. Higher AMI and ARI values indicate that a clustering partition matches better with the ground-truth labels. We report the average values for k, AMI and ARI obtained from 30 executions of each method on each dataset. Care is needed when interpreting results concerning the estimation of the number of clusters k, since a correct estimation does not necessarily imply a correct clustering solution. Safer conclusions can be drawn by considering together the AMI measure and the estimated k.

4.4.2 Experimental results on real data

The experimental results are summarized in Tab. 4.2. First, we empirically confirm results known in the literature, that top-down methods such as X-means and G-means fail to capture the structure a dataset unless their assumptions are rather true. X-means exploits the BIC criterion, while G-means relies on statistical tests for Gaussianity. Their estimations of the number of clusters is one or even two orders of magnitude higher than the actual number of clusters in the data. Dip-means and pdip-means are also top-down approaches but they rely on unimodality tests and consistently outperform X-means and G-means providing better estimations for the number of clusters. Therefore, it is evident that the unimodality-based methods perform better compared to methods that make 'stricter' assumptions, such as Gaussianity. However, on datasets containing many clusters, both dip-means and pdip-means

terminate too early and fail to reasonably estimate the number of clusters. There are two distinct sources for this shortcoming: first, their approach for testing dataset unimodality (such as the dip-dist, which is a 'meta-test', and variations of it) is not very effective in the multivariate setting and, since the methods operate in a top-down fashion, there is high chance for false positive identification of unimodality; second, the structure of the true clusters may be complex, thus the classical unimodality assumption may not be valid. This is what we aim to capture with the proposed locally unimodal cluster definition. In addition, Mean Shift performed well only in the YTF dataset. HDBSCAN gave promising results in YTF, Waveform-v1 and Mice Protein, while its performance was poor in the remaining datasets due to low AMI and ARI values or false k detection. The SMMP method produce promising results only in YTF, Optidigits and TCGA datasets. The RCC method produced satisfactory results on the Optdigits, Waveform-v1 and TCGA datasets, yet it failed on the rest of the benchmark datasets because the k estimation is far from the ground truth labeling.

The UniForCE algorithm performed satisfactorily on all benchmark datasets. It produced high-quality estimates of the number of clusters and high-quality clusterings with a high AMI on all datasets. More specifically, on the EMNIST-BD, EMNIST-BL, YTF, Waveform-v1, Isolet, HAR, TCGA, and Complex 2D datasets, UniForCE outperformed the other methods in both estimating k and providing clustering solutions with high AMI and ARI. In addition, the UniForCE method had the best solutions for k in the Optdigits and Mice Protein datasets, while it was highly competitive in AMI and ARI. Finally, in the Pendigits and EMNIST-M datasets, the method had very high AMI and ARI scores with good performance on k, where it overestimated the ground truth labeling by a small margin. However, it should be noted that the detectable cluster structure is not always aligned with the number of classes labeled in the dataset.

Finally, Fig. 4.5 provides visualizations of UniForCE clustering results on 6 of the real datasets reported in Tab. 4.2. The visualizations are produced in an unsupervised manner by 2D t-SNE embeddings, which are then colored by the cluster labels decided by UniForCE. In addition, the associated AMI scores are shown in each case. The results provide clear evidence that the local unimodality clustering performed by UniForCE is meaningful as it identifies well the cluster structure of high-dimensional real data.

Table 4.3: **Sensitivity analysis of hyperparameters α, L, and M.** Results are reported for 30 experiments per setting, showing how variations in the significance level α, the number of Monte Carlo simulations L, and the minimum subcluster size M affect the clustering performance across six datasets. Performance is evaluated based on the number of clusters k, AMI, and ARI.

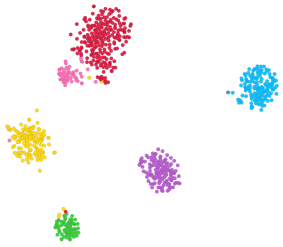
	EMNIST-BD			**EMNIST-BL**			**Pendigits**		
α	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
0.01	10±1	0.87	0.87	16± 1	0.70	0.67	19±1	0.77	0.75
0.001	10±1	0.87	0.87	12± 1	0.74	0.69	17±1	0.78	0.76
0.0001	10±0	0.87	0.87	11± 1	0.74	0.67	16±1	0.79	0.76
L	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
1	10±1	0.87	0.87	13± 1	0.73	0.66	17±1	0.78	0.75
5	10±0	0.87	0.88	13± 1	0.74	0.67	17±1	0.78	0.75
11	10±1	0.87	0.87	12± 1	0.74	0.69	17±1	0.78	0.76
M	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
10	10±1	0.87	0.87	12± 1	0.75	0.67	17±1	0.78	0.75
15	10±1	0.87	0.87	13± 1	0.74	0.67	17±1	0.78	0.75
25	10±1	0.87	0.87	12± 1	0.74	0.69	17±1	0.78	0.76
Ground-truth	10	–	–	10	–	–	10	–	–
	Waveform-v1			**TCGA**			**Complex 2D**		
α	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
0.01	4±1	0.89	0.92	6± 0	0.88	0.88	6±1	0.94	0.96
0.001	3±0	1.00	1.00	5 or 6	0.93	0.94	6±0	0.98	0.99
0.0001	3±0	1.00	1.00	5± 0	0.96	0.98	6±0	0.98	0.99
L	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
1	3±0	1.00	1.00	6± 1	0.91	0.92	6±0	0.98	0.99
5	3±1	1.00	1.00	5 or 6	0.91	0.92	6±0	0.97	0.98
11	3±0	1.00	1.00	5 or 6	0.93	0.94	6±0	0.98	0.99
M	k	AMI	ARI	k	AMI	ARI	k	AMI	ARI
10	3±0	1.00	1.00	5± 0	0.96	0.97	6±0	0.98	0.99
15	3±0	1.00	1.00	5 or 6	0.93	0.94	6±0	0.98	0.99
25	3±0	1.00	1.00	5 or 6	0.93	0.94	6±0	0.98	0.99
Ground-truth	3	–	–	5	–	–	6	–	–

(a) EMNIST-BD (AMI: .87)

(b) YTF (AMI: .94)

(c) Optdigits (AMI: .85)

(d) HAR (AMI: .63)



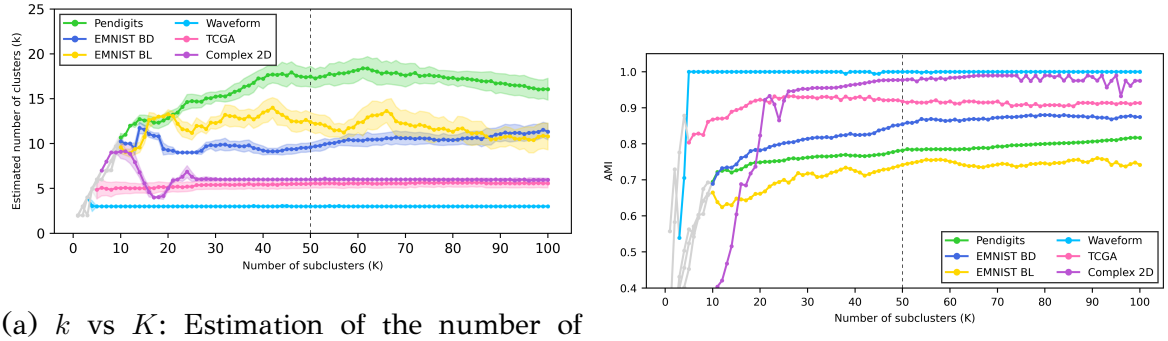
(e) TCGA (AMI: .93)

(f) Mice Protein (AMI: .93)

Figure 4.5: **t-SNE visualization of UniForCE clustering results on real datasets.** The embeddings are colored by the cluster labels decided by UniForCE for 6 of the real datasets.

4.4.3 Sensitivity study using real data

To complement our experimental analysis, we conducted a sensitivity study on the main hyperparameters of UniForCE. First, we study the influence of the overclustering resolution (K) in the performance of our method (which was fixed to $K = 50$ earlier) by experimenting within a range $K = 1, \ldots, 100$. The plots in Fig. 4.6a, showcase that the UniForCE method is quite robust with respect to the parameter K, except for the EMNIST-BL dataset, where the final number of clusters k increases as the initial number of clusters K increases. This may be an indication of the existence of a large number of substructures (much higher than the number of ground truth classes k^*). Finally, Fig. 4.6b provides a detailed view over the effect that the value of K has on the AMI measure. For completeness, we included for each curve a first part appearing in gray, which corresponds to when $K < k^*$, and therefore the cases where the initialization is not an overclustering, but rather an underclustering.



(a) k vs K: Estimation of the number of clusters (k) by the UniForCE method as a function of the number of starting subclusters (K).

(b) AMI vs K: Clustering performance (AMI score) by the UniForCE method as a function of the number of starting subclusters (K).

Figure 4.6: **Comparison of clustering results.** The gray part of each curve corresponds to clustering solutions where $K < k^*$ for a dataset.

In our sensitivity study we also analysed the influence of the significance level α, the number of Monte Carlo simulations L, and the minimum subcluster size M. To isolate the effect of each parameter, in each experiment we varied a single hyperparameter while keeping the others fixed to default values ($K = 50$, $\alpha = 0.001$, $L = 11$, $M = 25$). For each configuration, we performed 30 independent executions and evaluated the results using the estimated number of clusters k, AMI, and ARI. As shown in Tab. 4.3, lower values of α make UniForCE more conservative, often discovering fewer clusters, e.g. see the Pendigits and EMNIST-BL datasets. Regarding the parameter M, which sets the minimum subcluster size for applying the dip-test, our experiments show that the algorithm remains effective even for $M < 25$, where $M = 25$ is a value suggested by the literature [151]. In addition, the performance of the UniForCE method remains remarkably stable across different values of L. Overall, the algorithm performed consistently well across all the settings tested, further demonstrating its robustness and reliability under a wide range of parameter configurations.

4.4.4 Experimental results on synthetic data

To provide further insight into the UniForCE's clustering performance, we conducted additional experiments with synthetic 2D and 3D datasets that have been used in

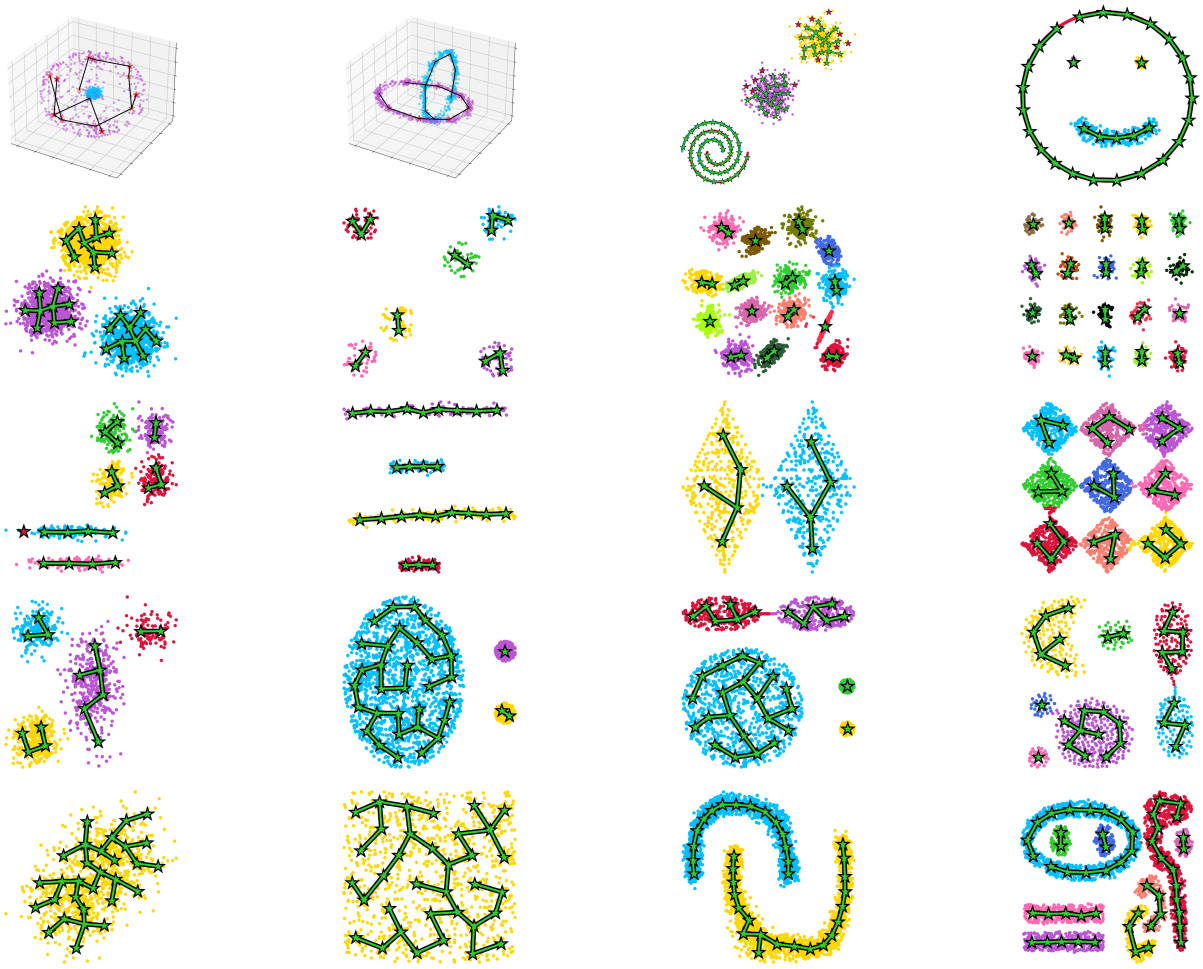


Figure 4.7: Clustering results using the UniForCE algorithm on a variety of 20 synthetic datasets.

the related literature[4]. Fig. 4.7 presents a panorama of 20 cases containing several typical and irregular shapes: Gaussian clusters, Uniform shapes, rings, rectangles, very elongated forms such as lines or 'moons', irregular shapes, and nested clusters. In some cases, the clusters are not linearly separable and/or they are imbalanced in terms of number of data points and spread size. The obtained results are impressive: the locally unimodal cluster definition seems versatile enough to capture the variety of data densities and shapes, and also the algorithm manages to find meaningful clustering solutions. In only few cases, e.g. those in subfigures $(1,3)$ and $(3,1)$, the postprocessing of the overclustering decided to eliminate very small initial subclusters. The centers of those subclusters appear as red stars. Such small initial subclusters occurs quite rarely in practice, and therefore the postprocessing of the overclustering

[4]They are available at: https://github.com/deric/clustering-benchmark.

is of minor importance.

4.5 Discussion and limitations

Our main contribution is the introduction of the local unimodality concept and the design of a tailored statistical unimodality test for pairs of subclusters. This in turn allows the definition of the unimodality graph that summarizes the density of the data with respect to local unimodality. The edge weights of this graph are the p-values of the associated local unimodality tests, while the given significance level α defines a threshold that eliminates edges with p-value $< \alpha$, and reveals the data clusters as connected components. In that sense, the clustering problem is already solved and what remains for an algorithmic scheme is to identify those connected components. The choice of how to achieve this latter does not affect the clustering result.

From an algorithmic point of view, UniForCE's scheme can be understood as finding *any spanning forest* in the thresholded unimodality graph. Note the resemblance to single-link agglomerative clustering, which would find the same top-level clusters (knowing the stopping k), however this can be understood as finding the minimum (or maximum) spanning forest in a distance (respectively similarity) graph. The advantage of the bottom-up algorithmic scheme we propose is that it computes only a sufficient subset of local unimodality tests instead of the whole unimodality graph. This is achieved by testing pairs of subclusters in ascending order with regards to the distance between their centers (taken as a rough indication for homogeneity), and, as clusters get aggregated, pairs of subclusters already belonging to the same cluster are not tested.

From a general standpoint, the identification of arbitrary-shaped clusters can be enabled mainly through data aggregation using a local cluster-merging criterion, which is the most crucial element of each method. Such a criterion allows a cluster to extend over a region for as long as a chosen property holds, without committing to any global cluster shape. This general strategy has been used extensively in the literature, either formalized as a single-link agglomerative, or as a density-based approach (DBSCAN and variants). The latter grows a cluster region toward a neighboring area using a local data density criterion. An advantage of this cluster-growing criterion is that it can also decide the termination of the process, hence estimates the

number of clusters. This is not the case for agglomerative clustering that needs an external termination criterion, typically not associated to the employed cluster-merging criterion.

Despite the numerous clustering algorithms proposed in this vein, the main limitation is that they use heuristic cluster-merging/-growing criteria whose parameters are particularly difficult to tune; e.g. they may rely on distances between clusters, simple features of local geometry, cluster variance, local density level, etc. UniForCE's novelty lies in that it uses the local unimodality to control the cluster aggregation, which is done in a statistically sound way and without involving external stopping criteria.

Deploying UniForCE in practice and limitations. In practice, finding a quality overclustering needs attention (Sec. 4.3.1), mainly because each subcluster is assumed to be a homogeneous segment of the data density that, without any further revision, will be eventually associated to one of the final clusters. The number of subclusters K obtained is determined by the user guess K', where $K' > K \gg k$, and the minimum size of admissible subcluster M. K' and M set together the resolution of the final clustering as clusters that are smaller than the initial K' subclusters or have less than M data points cannot be identified. Since our methodology relies on statistical testing on pairs of subclusters, their union needs to have sufficient data density. The proposed default value of $M = 25$ data points is so that $2M \geq 50$, which is the minimal sample size for reliable local unimodality testing using the dip-test [151]. In practice, this value not only is it sufficient for all the datasets we experiment with later, but our sensitivity analysis shows that the method is robust even for lower M values (see Sec. 4.4.2 and 4.4.3).

Although it is possible to drop the hyperparameter K' and ask for a maximal overclustering constrained only by a chosen small M value, that strategy would be computationally expensive for large datasets, and would also challenge the local unimodality testing by always testing very small samples. The role of K' is, to mitigate those risks, at the cost of a not so sensitive tuning. Provided $M = 25$, we provided evidence suggesting that setting K' hyperparameter requires way little sophistication: i) for different K' values UniForCE finds similar final data partitions; ii) UniForCE seems able to produce high quality results with a fixed K' value in several different datasets, for instance we used $K' = 50$. The sensitivity analysis for this choice is provided in Fig. 4.6.

An inherent limitation of identifying clusters based on the property of local unimodality is that it focuses on the local shape of the data density and hence neglects the level of density, as well as the abruptness of the density variation. Moreover, regarding the specifics of the UniForCE algorithm and the local statistical tests involved, intensive noise between two clusters could affect the overclustering by placing there subclusters that could eventually lead to a false merging of two clusters. In that sense, in such cases, our current implementation may underestimate the number of clusters. Issues related to noise and data density variation could be addressed at three levels: i) by a generic denoising preprocessing step before applying UniForCE; ii) by a method-specific postprocessing step over the overclustering to identify subclusters that are located at low density or remote areas; iii) by postprocessing the unimodality forest through inspection of certain links between subclusters.

4.6 Summary

In this Chapter, we presented the UniForCE clustering method that clusters and estimates the number of clusters k. Determining the number of clusters k during the optimization procedure is one of the most challenging problems in the field, especially in high-dimensions. Most methods require k as input, while others claim to estimate k, but they essentially translate the problem into another, hopefully easier, problem, i.e. hyperparameter-tuning.

The proposed approach is based on the novel definition of *locally unimodal cluster*. The main idea is that, instead of perceiving unimodality as a property that needs to hold for the whole cluster density, we proposed to study it at a local level, at subregions of the cluster density. We based our approach on the observation that unimodality may extend across pairs of neighboring subclusters when tested as a union. Such *unimodal pairs* enable the aggregation of small subclusters and the bottom-up formation of larger cluster structures in a statistically sound manner. Specifically, a locally unimodal cluster extends across subregions of the data density as long as there are unimodal pairs connecting them in a single connected component of the *unimodality graph*. The proposed locally unimodal cluster definition is flexible as it identifies arbitrary-shaped clusters, including typical unimodal or convex shapes.

As part of the proposed methodology, we have developed a statistical procedure

to identify unimodal pairs of subclusters by extending the functionality of the dip test to multidimensional data. Using the proposed statistical procedure, we built the unimodality graph in which both clustering and estimation of k can be addressed through the computation of a *unimodality spanning forest*. Each spanning tree of the forest connects subclusters of the dataset that are aggregated in the same final cluster. The number of trees is an estimate for the number of clusters that the UniForCE method provides. The strengths of the contribution's conceptual and algorithmic side have been validated with extensive numerical experiments on various real and synthetic datasets. Additionally, the sensitivity analysis highlights the robustness of the method with respect to the choice of the relatively few hyperparameters.

Chapter 5

Deep Clustering Using the Soft Silhouette Score

5.1 Introduction

As presented in Chapter 1 and Section 1.3, the vast majority of AE-based methods learn a representation in which individual clusters have small inner cluster variability. Most common approaches are the minimization of the k-means error, or the KL divergence between the soft clustering assignments and a target distribution. Such a representation has been shown to improve the clustering results in several scenarios. However, minimizing only the inner cluster distance is a suboptimal strategy. Our motivation is to formulate a deep clustering objective that simultaneously considers both the inner cluster distance and the outer cluster separation. This is achieved by optimizing the *soft silhouette* objective introduced in this Chapter [90].

Assessing the quality of a clustering solution is typically a challenging task. In this direction, several quality measures have been proposed which can be categorized as

external and internal measures [161]. External quality measures, as the name suggests, use additional information about the data as the ground truth labels. Well-known external evaluation measures include Normalized Mutual Information (NMI) [162], Adjusted Mutual Information (AMI) [159], Adjusted Rand Index and (ARI) [160, 163]. However, such measures are not applicable in real-world applications where the ground truth labels are absent. Internal quality measures, on the other hand, can be applied to the clustering problem since they are based solely on the information intrinsic to the data. Some typical internal clustering measures that take into account both cluster compactness and separation are the Dunn index [164], the Calinski-Harabasz index [165], the Davies-Bouldin index [166], and the silhouette [92]. In particular, the silhouette coefficient is the most widely used and successful internal validation measure [93].

The typical silhouette is considered as an effective clustering quality measure that combines both inter and intra cluster information. Specifically, silhouette rewards clustering solutions that exhibit both compactness within individual clusters and clear separation between clusters. However, it assumes a hard clustering solution, thus it cannot be used to evaluate probabilistic clustering solutions, unless they are transformed to discrete ones based on maximum cluster membership probability. In addition, the silhouette score cannot be efficiently used as a clustering objective for neural network training since it is not differentiable.

In this Chapter, in order to overcome the above limitations, we propose an extension of the silhouette score, called *soft silhouette* score, that evaluates the quality of probabilistic clustering solutions without requiring their transformation to discrete ones. Besides this obvious advantage, a notable property of soft silhouette is that it is differentiable with respect to cluster assignment probabilities. Assuming that such probabilities are provided by a parametric machine learning model, the soft silhouette score is used as a clustering objective function to train parametric probabilistic models using typical gradient-based approaches.

To this end, we propose a novel AE-based deep clustering methodology that directly provides cluster assignment probabilities as network outputs and exploits the soft silhouette score as a clustering objective. In this way, by training the network using soft silhouette, we achieve minimization of the inter-cluster variance, while at the same time maximizing the margin between clusters in the embedded space.

The rest of the Chapter is organized as follows. In Section 5.2 the soft silhou-

ette score introduced. Then in Section 5.3 we describe the proposed deep clustering methodology by presenting the model architecture, the corresponding objective function as well as the training method. Finally, in Section 5.4, we provide extensive experimental results and comparisons, while in Section 5.5, summarizes this Chapter.

5.2 The Soft Silhouette Score

5.2.1 Silhouette

The silhouette score [92, 167] is a measure utilized to assess the quality of a clustering solution. It assumes that a good clustering solution encompasses compact and well-separated clusters. Assume that we are given a partition $C = \{C_1, ..., C_K\}$ of a dataset $X = \{x_1, \ldots, x_N\}$ into K clusters. Let also $d(x_i, x_j)$ denote the distance between x_i and x_j.

The silhouette score computation proceeds by evaluating the individual silhouette score $s(x_i)$ of each data point x_i as follows. We first compute its average distance $a(x_i)$ to all other data points within its cluster C_I:

$$a(x_i) = \frac{1}{|C_I| - 1} \sum_{x_j \in C_I, i \neq j} d(x_i, x_j), \tag{5.1}$$

where $|C_I|$ represents the cardinality of cluster C_I, where $|C_I| > 1$. The value of $a(x_i)$ value quantifies how well the data point x_i fits within its cluster. A low value of $a(x_i)$ indicates that x_i is similar to its cluster members, suggesting that x_i is probably grouped correctly. Conversely, a higher value of $a(x_i)$ indicates that x_i is far from its cluster members.

The silhouette score also requires the calculation of the minimum average outer-cluster distance $b(x_i)$ for each data point $x_i \in C_I$ defined as

$$b(x_i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{x_j \in C_J} d(x_i, x_j). \tag{5.2}$$

A large $b(x_i)$ value indicates that the data point x_i significantly differs from data points in other clusters which is desirable.

The silhouette score of a data point x_i takes into account the requirements for small $a(x_i)$ and large $b(x_i)$ and is defined as:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}}. \tag{5.3}$$

It should be noted that $-1 \leq s(x_i) \leq 1$. A value close to 1 is achieved when $a(x_i)$ is small and $b(x_i)$ is high. This indicates that x_i has been assigned to a compact, well-separated cluster. In contrast, a value close to -1 suggests that x_i is more similar to points in other clusters than to points in its cluster, thus it has probably been assigned a wrong cluster label.

The total silhouette score for the whole partition C of the dataset X is obtained by aggregating the individual silhouette values through typical averaging:

$$S(X) = \frac{1}{N} \sum_{i=1}^{N} s(x_i). \tag{5.4}$$

The silhouette score [92] is not only suitable for (internal) clustering evaluation but also defines an intuitive clustering objective that rewards compact and well-separated clusters. As presented in Chapter 1 and Section 1.3, while several deep clustering objectives aim to provide compact clustering solutions, they do not optimize explicitly for cluster separability. Next, we introduce a probabilistic silhouette score, termed soft silhouette, which allows us to optimize for both compact and well-separated clusters.

5.2.2 Soft Silhouette

The soft silhouette score introduced below constitutes an extension of the typical silhouette score that assumes probabilistic cluster assignments instead of hard cluster assignments. More specifically, assume a dataset $X = \{x_1, \ldots, x_N\}$ partitioned into K clusters $C = \{C_1, \ldots, C_K\}$ and let $d(x_i, x_j)$ the distance between data points x_i and x_j. Let also $P_{C_I}(x_i)$ denote the probability that x_i belongs to cluster C_I. Obviously $\sum_{I=1}^{K} P_{C_I}(x_i) = 1$.

Assuming that x_i belongs to cluster C_I we define as:

- $a_{C_I}(x_i)$ the value of the distance of x_i to cluster C_I. This is actually a weighted average (expected value) of the distances of x_i to all other points $x_j \in X$ with weight the probability $P_{C_I}(x_j)$ (ie. that x_j belongs to the cluster of interest C_I)

$$a_{C_I}(x_i) = \frac{\sum\limits_{j=1}^{N} P_{C_I}(x_j) d(x_i, x_j)}{\sum\limits_{j=1, j\neq i}^{N} P_{C_I}(x_j)}. \tag{5.5}$$

- $b_{C_I}(x_i)$ the minimum value of the (expected) distance of x_i from the other

clusters C_J different from C_I

$$b_{C_I}(x_i) = \min_{J \neq I} \frac{\sum_{j=1}^{N} P_{C_J}(x_j) d(x_i, x_j)}{\sum_{j=1, j \neq i}^{N} P_{C_J}(x_j)} = \min_{J \neq I} a_{C_J}(x_i). \tag{5.6}$$

- $s_{C_I}(x_i)$ the conditional silhouette value for x_i given that it belongs to cluster C_I:

$$s_{C_I}(x_i) = \frac{b_{C_I}(x_i) - a_{C_I}(x_i)}{\max\{a_{C_I}(x_i), b_{C_I}(x_i)\}}. \tag{5.7}$$

Then the soft silhouette value $sf(x_i)$ of data point x_i is computed as the expected value of $s_{C_I}(x_i)$ with respect to its cluster assignment probabilities $P_{C_I}(x_i)$:

$$sf(x_i) = \sum_{I=1}^{K} P_{C_I}(x_i) s_{C_I}(x_i), \tag{5.8}$$

and the total soft silhouette score $Sf(X)$ of the partition is computed by aggregating (averaging) the individual scores $sf(x_i)$:

$$Sf(X) = \frac{1}{N} \sum_{i=1}^{N} sf(x_i), \tag{5.9}$$

It should be noted that, in the case of hard clustering, the cluster assignment probability vectors become one-hot vectors and the soft silhouette equations become similar to the typical silhouette equations.

It is obvious from the above equations that soft silhouette is differentiable with respect to the cluster assignment probabilities. Therefore, it can can be employed as a clustering objective function to be optimized in a deep learning framework. The major advantage of this objective is that it optimizes simultaneously both cluster compactness and separation. Such a deep clustering approach is presented next.

5.3 The DCSS method: Deep Clustering using Soft Silhouette

In this section we propose the Deep Clustering using Soft Silhouette (DCSS) algorithm, which belongs to the category of AE-based deep clustering methods that employ the soft silhouette as a clustering loss.

As described in Chapter 1 and Section 1.3 a typical AE-based deep clustering method employs an encoder network

$$z = f_w(x), \qquad f_w(\cdot) : \mathbb{R}^d \to \mathbb{R}^m,$$

that provides the latent representations (embeddings) z and a decoder network

$$\hat{x} = g_\theta(x), \qquad g_\theta(\cdot) : \mathbb{R}^m \to \mathbb{R}^d,$$

that reconstructs the outputs given the embeddings. The networks are trained to optimize a total loss that is the sum of the reconstruction loss and the clustering loss: $\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cl}$.

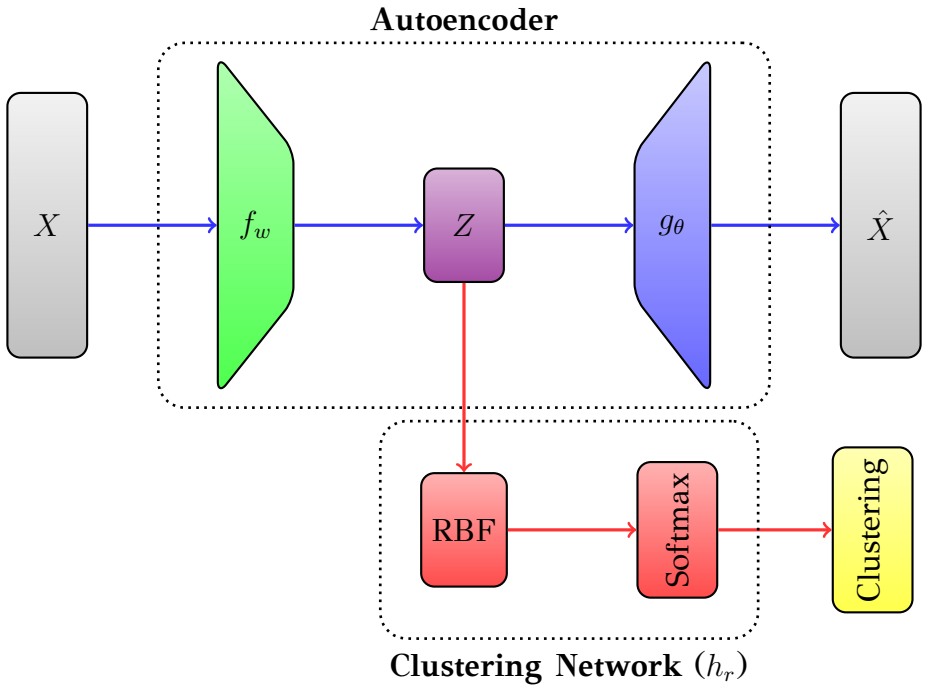


Figure 5.1: The proposed model architecture. The AE comprises the encoder f_w and the decoder g_θ. The data space is denoted as X, while the embedded space is represented by Z. The clustering network h_r consists of an RBF layer followed by a softmax layer.

In the proposed approach, the clustering loss will be based on the soft silhouette score, which requires the cluster assignment probabilities $p(x) = (p_1(x), \ldots, p_K(x))$ for an input x. For this reason, we enrich the AE-model with an additional network $h_r(z)$, called clustering network, that takes as input the embedding $z = f_w(x)$ of a data point x and outputs the cluster assignment probabilities $p_j(x) = h_{rj}(x)$ for $j = 1, \ldots, K$. Therefore, given the data set $X = \{x_1, \ldots, x_N\}$, the embedding $z_i = f_w(x_i)$ is first computed. Then the pairwise distances $d(z_i, z_j)$ and the cluster assignment probability vectors $p(x_i) = h_r(z_i)$ are specified, required for the soft silhouette computation.

The proposed three network architecture is illustrated in Fig. 5.1. It can be observed that the clustering network operates in parallel with the decoder network. For

an input vector x, the model provides the embedding vector z, the reconstruction vector $\hat{x}$, and the probability vector $p(x)$.

Based on experimentation with several alternatives, we have selected as a clustering network h_r an a Radial Basis Function (RBF) [168] model with a softmax output unit that provides the probability vector of the cluster assignments. The number of RBF units is set equal to the number of clusters K.

Soft silhouette is a criterion that should maximized in order to obtain solutions of good quality. Since a clustering loss is a quantity to be minimized, we take into account that $Sf \leq 1$ and define the clustering loss as follows:

$$\mathcal{L}_{cl} = 1 - Sf. \tag{5.10}$$

Note that $\mathcal{L}_{cl}$ is always positive and attains each minimum value when Sf is maximum ($Sf = 1$). Thus, the total loss for model training is specialized as follows:

$$\mathcal{L}_{AE} = \frac{1}{N}\sum_{i=1}^{N} ||x_i - g_\theta(f_w(x_i))||^2 + \lambda(1 - Sf(h_r(X))), \tag{5.11}$$

where $h_r(X) = \{h_r(x_1), \ldots, h_r(x_N)\}$ are the cluster assignment probability vectors. It should be noted that the pairwise distances $d(f_w(x_i), f_w(x_j))$ between the embeddings are also involved in the Sf computation. In this Chapter, the Euclidean distance has been used. Since $\mathcal{L}_{AE}$ is differentiable with respect to the model parameters w, θ, r it can be minimized using typical gradient-based procedures.

A technical issue that has emerged when training the model is that in many cases a trivial solution is attained where the output probabilities tend to be uniform (ie. equal to $1/K$) for many data points. To overcome this difficulty, we have included an additional term to the objective function that penalizes uniform solutions by minimizing the entropy of the output probability vectors. Thus, the entropy regularization term $\mathcal{L}_{reg}$ is defined as follows:

$$H(h_r(X)) = -\sum_{i=1}^{N}\sum_{j=1}^{K} h_{rj}(x_i)\log h_{rj}(x_i), \tag{5.12}$$

The final total loss that is minimized to train our model is:

$$\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda_1\mathcal{L}_{cl} + \lambda_2\mathcal{L}_{reg}, \tag{5.13}$$

Algorithm 5.1 Deep Clustering using Soft Silhouette algorithm (DCSS)

Require: X (dataset)

Require: K (number of clusters)

Require: λ_1, λ_2 (regularization hyperparameters)

1: Randomly initialize the w and θ AE parameters.

Stage 1: **Pretraining**

2: Pretrain the encoder f_w and decoder g_θ by minimizing the reconstruction error $\mathcal{L}_{rec}$ (eq. 1.25) through gradient based optimization for T_{pr} epochs. *// We employed batch training using the Adam optimizer.*

3: Apply k-means with K clusters to the learned representations $z = f_w(X)$.

4: Initialize the parameters of the clustering network h_r using the k-means result.

Stage 2: **Training**

5: Update the parameters θ, w and r by minimizing the total loss (eq. 5.13) until convergence through gradient based optimization to obtain $\theta^\star$, $w^\star$ and $r^\star$.

Stage 3: **Inference**

6: Compute the clustering solution $C = \arg\max softmax(h_{r^\star}(f_{w^\star}(X)))$. *// Data clustering.*

7: **return** the clustering solution C and the learned parameters $w^\star$, $\theta^\star$, $r^\star$.

and in more detail

$$\mathcal{L}_{AE} = \frac{1}{N}\sum_{i=1}^{N} ||x_i - g_\theta(f_w(x_i))||^2 + \lambda_1(1 - Sf(h_r(X))) \\ - \lambda_2 \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} h_{rj}(x_i)\log h_{rj}(x_i). \tag{5.14}$$

The details of the approach, called Deep Clustering using Soft Silhouette (DCSS), are summarized in Algorithm 5.1.

5.4 Experiments

In this section we present our experimental results on both real datasets and a synthetic dataset. In the first part, we demonstrate the representation learning capabilities of several methods compared to DCSS on a synthetic dataset. In the second part, we demonstrate the deep clustering capabilities of the DCSS method on several real datasets compared to the most widely used (AE-based) deep clustering methods that have been discussed in Chapter 1 and Section 1.3.

5.4.1 Synthetic Data Demonstration

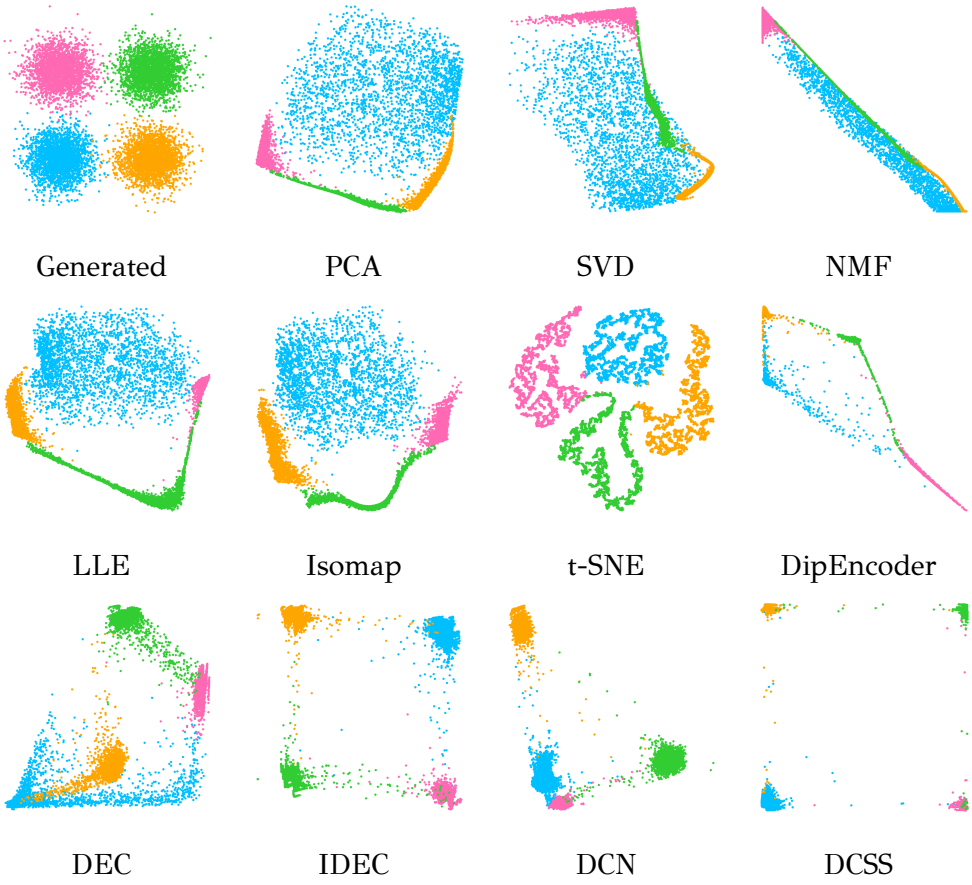


Figure 5.2: Synthetic demonstration of the representation learning capabilities of several methods. The generated 2-d dataset (top left) is hidden from the methods. Each method receives as input a 100-d dataset generated by non-linear transformations applied to the original 2-d data and provides a 2-d latent representation of the 100-d dataset, which is presented in the plots. Color indicates the true cluster labels.

We have relied on the synthetic dataset considered in [71] (for testing the DCN method) generated as follows. Let's suppose the observed high dimensional data points x_i exhibit a clear cluster structure in a two-dimensional latent space Z, ie. the latent vectors $z_i \in \mathbb{R}^2$ form compact and well-separated clusters. Given a latent vector z_i, the corresponding observation x_i is generated using the following transformation:

$$x_i = \sigma(U\sigma(Wz_i)), \tag{5.15}$$

where $W \in \mathbb{R}^{10\times 2}$ and $U \in \mathbb{R}^{100\times 10}$ are matrices whose entries are sampled from the Normal distribution $\mathcal{N}(0,1)$, and $\sigma(x)$ is the logistic function that introduces non-

linearity into the generation process. Given the observations x_i, recovering the original clustering-friendly domain in which z_i resides appears to be challenging.

We generated a set of latent vectors z_i belonging to four planar clusters, each with 2,500 samples (as shown in the first subfigure of Fig. 5.2) and we computed the corresponding observations x_i. The rest subfigures Fig. 5.2 demonstrate the two-dimensional projections provided using several dimensionality reduction methods given the observations x_i as input. More specifically, we present the solutions provided by Principal Component Analysis (PCA) [60], Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF) [61], Local Linear Embeddings (LLE) [169], Isomap [170], and t-SNE [78]. We also considered the deep clustering methods DEC [73], IDEC [79], DCN [71], DipEncoder [81], as well as our proposed DCSS method.

It is evident that the projection methods that do not optimize a clustering loss (first and second rows excluding DipEncoder) have failed to reveal the hidden two-dimensional latent structure. On the contrary, deep clustering methods (bottom row plus DipEncoder) demonstrated better performance. Specifically, DEC was able to recover three out of four latent clusters, however the fourth was scattered. DCN was able to reconstruct all of them, but the blue and red clusters are not sufficiently separated. IDEC was able to learn a very informative projection revealing the four cluster structure. DipEncoder was also able to reveal the four cluster structure, although some data points remain scattered. Superior are the results of the DCSS method, which was not only able to reveal the four clusters, but also sufficiently maximized their separation.

5.4.2 Datasets

Table 5.1 summarizes the benchmark datasets that we used for experimental evaluation, which vary in size n, dimensions d, number of clusters k, complexity, data type, and domain of origin. The subsequent paragraphs offer a more comprehensive overview of the datasets we employed and outline the preprocessing procedures we implemented for each of these datasets.

The datasets used in this study include the Pendigits (PEN), EMNIST MNIST (EMNIST), and EMNIST Balanced Digits (BD). These datasets consist of handwritten digits categorized into ten classes, each representing digits from 0 to 9. It is worth

Table 5.1: The datasets used in our experiments. N is the number of data instances, d is the dimensionality, and k denotes the number of clusters.

Dataset	**Type**	N	d	k	**Source**
EMNIST Balanced Digits	Image	28000	28×28	10	[154]
EMNIST MNIST	Image	70000	28×28	10	[154]
EMNIST Balanced Letters (A-J)	Image	28000	28×28	10	[154]
EMNIST Balanced Letters (K-T)	Image	28000	28×28	10	[154]
EMNIST Balanced Letters (U-Z)	Image	16800	28×28	6	[154]
Fashion MNIST	Image	70000	28×28	10	[171]
Kuzushiji MNIST	Image	70000	28×28	10	[172]
HAR	Tabular	10299	560	6	[173]
Pendigits	Tabular	10992	16	10	[95]
Waveform-v1	Tabular	5000	21	3	[95]
Synthetic	Tabular	10000	2	4	[71]

noting that the EMNIST dataset constitutes an extended and more challenging version of the MNIST dataset [96]. Both the E-MNIST and BD datasets comprise images with a resolution of 28×28 pixels. In contrast, Pendigits data instances are represented by 16-dimensional vectors containing pixel coordinates.

In addition, the EMNIST Balanced Letters (BL) dataset is included, featuring handwritten letters in both uppercase and lowercase forms, with a resolution of 28×28 pixels. The BL dataset has been divided into three mutually exclusive subsets. The first subset contains the letters A to J, the second includes the letters K to T, and the last subset contains the remaining letters U to Z. The first two subsets comprise 28000 data points each, distributed across 10 clusters, while the last subset consists of 16800 data points and 6 clusters. Furthermore, we used the Kuzushiji MNIST (K-MNIST) and the Fashion MNIST (F-MNIST) as additional datasets, offering a more challenging set of variations of the classic MNIST. Both include 70000 data points with a resolution of 28×28 pixel images. More specifically, K-MNIST features ten types of Japanese (Kuzushiji) symbols, while F-MNIST provides a diverse set of clothing items with ten types of objects.

The Human Activity Recognition with Smartphones (HAR) dataset was also considered. This dataset consists of data collected from the accelerometer and gyroscope

sensors of smartphones, sampled during a human activity. Specifically, each record in the dataset is a 560 feature vector with time and frequency domain variables. In addition, HAR consists of 6 classes of human activities which are the following: walking, walking upstairs, walking downstairs, sitting, standing, laying. Furthermore, the Waveform-v1 (WVF-v1) dataset was included, which consists of 3 classes of generated waves with 5000 data points. Each class is generated from a combination of 2 of 3 'base' waves. Finally, we also report results for the synthetic dataset described in the previous subsection.

In all datasets, we used min-max normalization as a prepossessing step, to map the attributes of each data point to the $[0, 1]$ interval to prevent attributes with large ranges from dominating the distance calculations and avoid numerical instabilities in the computations [27].

5.4.3 Neural Network Architectures

Determining optimal architectures and hyperparameters through cross-validation is not feasible in unsupervised learning problems. Therefore, we opt for commonly used architectures for the employed neural network models while avoiding dataset-specific tuning. Regarding tabular data, our approach involves adopting a well-established architecture, which consists of fully connected layers [174, 73]. The specific AE architecture that we used is the following:

$$x_d \rightarrow \text{Fc}_{500} \rightarrow \text{Fc}_{500} \rightarrow \text{Fc}_{2000} \rightarrow \text{Fc}_m \rightarrow \text{Fc}_{2000} \rightarrow \text{Fc}_{500} \rightarrow \text{Fc}_{500} \rightarrow \hat{x}_d,$$

where Fc_m stands for fully connected layer with m neurons and x_d represents a d-dimensional data vector.

In terms of image datasets, convolutional Neural Networks (CNNs) have demonstrated superior effectiveness in capturing semantic visual features. Consequently, we exploit a convolutional-deconvolutional AE to learn the embeddings for the image datasets. The AE architecture consists of three convolutional layers (encoder), one fully connected layer (emdedding layer), and three deconvolutional layers (decoder) [175, 176, 156]. More specifically, the architecture is the following:

$$x_{28\times 28} \rightarrow \text{Conv}^5_{32} \rightarrow \text{Conv}^5_{64} \rightarrow \text{Conv}^3_{128} \rightarrow \text{Fc}_m \rightarrow \text{Deconv}^3_{128} \rightarrow \text{Deconv}^5_{64} \rightarrow \text{Deconv}^5_{32} \rightarrow \hat{x}_{28\times 28},$$

where Conv_b^a (Deconv_b^a) denotes a convolutional (deconvolutional) layer with a $a \times a$ kernel and b filters, while the stride is always set to 2.

In the above encoder-decoder networks, the ReLU activation function is used [177], except for the embedded layer of the AE, where the Hyperbolic Tangent (tanh) function is used. The weights and biases are initialized using the He initialization method [178].

As already mentioned, in what concerns the clustering network, a Radial Basis Function (RBF) model with the number of hidden units equal to the number of clusters has been selected. The output of the RBF units is fed into a K output softmax activation function that provides the cluster assignment probabilities. After the AE pre-training, we initialize the centers of the RBF layer by using the k-means algorithm in the embedded space, while we initialize σ to a small positive value. The temperature parameter T of the softmax was set equal to $T = 20$.

Table 5.2: Performance results of the compared clustering methods.

		Method						
Dataset	**Measure**	k-mns	AE + k-mns	DCN	DEC	IDEC	DipEnc	DCSS
BD	NMI	0.48	0.72±0.01	0.75±0.02	0.80±0.03	0.82±0.01	0.77±0.06	**0.86**±0.04
	ARI	0.36	0.65±0.02	0.63±0.05	0.75±0.06	0.77±0.01	0.69±0.08	**0.80**±0.07
BL (A-J)	NMI	0.35	0.65±0.02	0.68±0.03	0.77±0.03	0.77±0.03	0.69±0.07	**0.80**±0.02
	ARI	0.25	0.55±0.02	0.56±0.05	0.69±0.06	0.69±0.05	0.58±0.09	**0.72**±0.04
BL (K-T)	NMI	0.51	0.73±0.02	0.77±0.03	0.84±0.01	0.84±0.02	0.81±0.05	**0.90**±0.02
	ARI	0.43	0.67±0.04	0.69±0.06	0.81±0.02	0.81±0.04	0.75±0.08	**0.87**±0.03
BL (U-Z)	NMI	0.47	0.64±0.01	0.64±0.02	0.68±0.02	0.67±0.02	0.64±0.04	**0.71**±0.04
	ARI	0.41	0.60±0.02	0.56±0.04	0.65±0.03	0.63±0.02	0.60±0.05	**0.68**±0.06
E-MNIST	NMI	0.48	0.75±0.01	0.83±0.03	0.84±0.03	0.85±0.02	0.84±0.04	**0.88**±0.04
	ARI	0.36	0.69±0.01	0.78±0.04	0.80±0.04	0.81±0.03	0.78±0.06	**0.83**±0.06
F-MNIST	NMI	0.51	0.61±0.02	0.62±0.01	0.57±0.01	0.59±0.01	0.61±0.02	**0.63**±0.02
	ARI	0.35	0.45±0.02	0.42±0.02	0.41±0.01	0.43±0.01	0.44±0.03	**0.45**±0.03
K-MNIST	NMI	0.44	0.54±0.00	0.53±0.01	0.56±0.02	0.57±0.01	0.59±0.02	**0.64**±0.01
	ARI	0.31	0.41±0.01	0.38±0.02	0.43±0.02	0.45±0.01	0.47±0.02	**0.49**±0.02
HAR	NMI	0.59	0.67±0.06	0.77±0.01	0.74±0.06	0.74±0.10	0.78±0.04	**0.81**±0.06
	ARI	0.46	0.60±0.09	0.70±0.02	0.66±0.08	0.65±0.12	0.67±0.05	**0.74**±0.09
PEN	NMI	0.69	0.68±0.02	0.73±0.03	0.73±0.03	0.75±0.03	0.75±0.01	**0.78**±0.02
	ARI	0.56	0.59±0.04	0.62±0.05	0.62±0.05	0.65±0.04	0.66±0.01	**0.68**±0.04
WVF-v1	NMI	0.74	0.84±0.13	0.95±0.08	0.93±0.12	0.89±0.09	**1.00**±0.00	**1.00**±0.00
	ARI	0.70	0.87±0.13	0.95±0.09	0.93±0.14	0.91±0.08	**1.00**±0.00	**1.00**±0.00
Synthetic	NMI	0.82	0.72±0.14	0.79±0.09	0.91±0.08	0.90±0.10	**0.94**±0.01	**0.94**±0.06
	ARI	0.83	0.70±0.19	0.75±0.14	0.92±0.12	0.91±0.11	**0.95**±0.01	**0.95**±0.05

5.4.4 Evaluation

It is important to mention that since clustering is an unsupervised problem, we ensured that all algorithms were unaware of the true clustering of the data. In order to evaluate the results of the clustering methods, we use standard external evaluation measures, which assume that ground truth clustering is available [161]. For all algorithms, the number of clusters is set to the number of ground-truth categories and assumes ground truth that cluster labels coincide with class labels. The first evaluation measure is the *Normalized Mutual Information* (NMI) [162] defined as:

$$NMI(Y, C) = \frac{2 \times I(Y, C)}{H(Y) + H(C)}, \tag{5.16}$$

where Y denotes the ground-truth labels, C denotes the clusters labels, $I(\cdot)$ is the mutual information measure and $H(\cdot)$ the entropy. The second metric used is the *Adjusted Rand Index* (ARI) [160, 163], which is a corrected for chance version of the Rand Index [179] that measures the degree of overlap between two partitions defined as:

$$ARI(Y, C) = \frac{RI(Y, C) - \mathbb{E}[RI(Y, C)]}{max\{RI(Y, C)\} - \mathbb{E}[RI(Y, C)]}, \tag{5.17}$$

where $RI(\cdot)$ denotes the Rand Index and $\mathbb{E}[\cdot]$ is the expected value.

5.4.5 Experimental Setup and Results

We have conducted a comprehensive performance analysis of the proposed DCSS method in comparison to well-studied deep clustering methods such as DCN [71], DEC [73], IDEC [79], and DipEncoder [81]. These methods are designed to facilitate the learning of a cluster-friendly embedded space, as also happens with our approach. Furthermore, we have evaluated the performance of k-means [26] both in the original space and in the embedded space (AE+k-means). The comparison with the latter approach quantifies the performance improvements achieved through the utilization of AE in the clustering procedure. At this point, it should be noted that for a fair comparison between the deep clustering methods, we used the same model architectures for all the methods, since we observed improved clustering results compared to those proposed in the original papers.

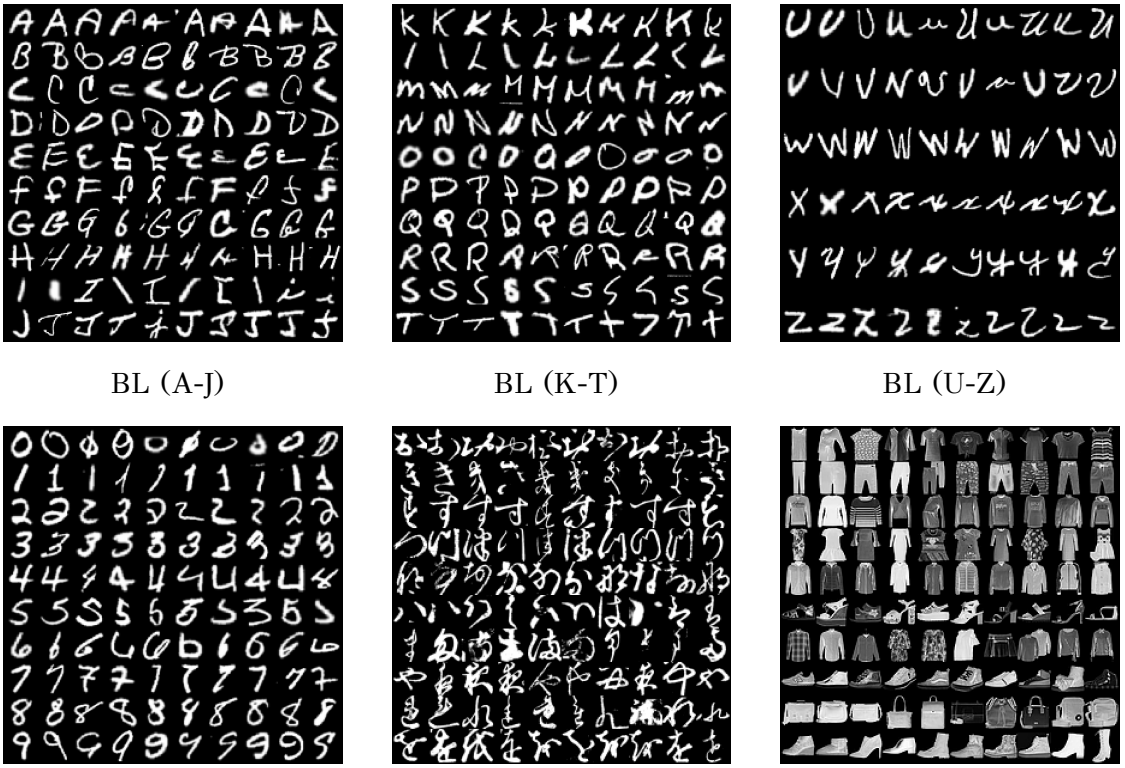
In experiments involving k-means, we initialized the algorithm 100 times and retained the clustering solution with the lowest mean sum of squares error. For the remaining methods, which integrate an AE model in the clustering procedure, we

conducted each experiment 10 times. In the context of deep clustering methods, an AE pre-training phase (ignoring the clustering loss) took place. For image datasets, we pretrained the AE for 100 epochs with a learning rate of 1×10^{-3}, while for tabular datasets, we extended the pre-training to 1000 epochs with a learning rate of 5×10^{-4}. During the pre-training phase, a small L_2 regularization of 1×10^{-5} was applied. In the training phase, the deep clustering models were trained for 100 epochs with a learning rate of 5×10^{-4} and without a regularization penalty. A fixed batch size of 256 was considered and the Adam optimizer [86] with the default settings of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ was used in both the pretraining and training phases. Additionally, there are several methodologies to tune the non-clustering (eq. 5.11) with the clustering loss (eq. 5.10) [15] during training. We choose the most simplistic and typical approach by setting our hyper-parameters to small values that balances the two losses. Specifically, we used $\lambda_1 = 0.01$ and $\lambda_2 = 0.01$. Finally, to initialize the centers of the RBF layer, we apply k-means in the embeddings of the pretrained AE, while σ was initialized to a small positive value.

In Table 5.2, we present the average performance in terms of NMI and ARI along with the standard deviation for each method and dataset. As anticipated, the clustering performance of k-means improves when projecting the data to a low-dimensional embedded space, as shown in the AE+k-means compared to k-means results. In general, the performance of DEC is better than that of DCN in all datasets except HAR and WVF-v1. At the same time, we can observe that IDEC slightly outperforms the DEC method in most cases, indicating that some improvement might be obtained by using the decoder part in the clustering optimization procedure. DipEncoder provides satisfactory results in general, outperforming DCN, DEC, and IDEC on tabular datasets. The proposed DCSS method demonstrates superior performance across all datasets. Concerning the NMI measure, there is a significant improvement ranging from 0.03 to 0.07 compared to the second best method for each dataset. In addition, the ARI measure also shows a similar improvement, ranging from 0.03 to 0.06. These results strongly indicate that soft silhouette constitutes an effective deep clustering objective function capable of providing compact and well-separated clusters.

Finally, in Figure 5.3, we provide image clustering examples using the DCSS method on datasets BL (A-J), BL (K-T), BL (U-Z), E-MNIST, K-MNIST, and F-MNIST. Images in the same row are assigned to the same cluster and are placed with decreasing cluster assignment probability, progressing from the leftmost (high

probability) to the rightmost (low probability) columns. It is obvious that more representative images are assigned higher probability values. The presented results are very satisfactory, taking into account that they have been obtained using an unsupervised learning method. Occasional assignment errors correspond to cases that are difficult to be discriminated even by visual inspection. For example, in the 5_{th} row of the E-MNIST images, it is not clear whether the rightmost image displays the 4 or the 8 digit.



BL (A-J) BL (K-T) BL (U-Z)

E-MNIST K-MNIST F-MNIST

Figure 5.3: Image clustering results on various datasets using the proposed DCSS method. In each sub-figure, rows correspond to different clusters. In each row the images are presented from left to right with decreasing cluster membership probability.

5.5 Summary

In this Chapter, we have proposed soft silhouette, an extension of the widely used silhouette score that accounts for probabilistic clustering assignments. Next, we have

considered soft silhouette as a differentiable clustering objective function and propose the DCSS deep clustering methodology that constitutes an AE-based approach suitable for optimizing the soft silhouette score. The proposed method has been tested and compared with well-known deep clustering methods on various benchmark datasets, yielding very satisfactory results. The experimental study indicates that soft silhouette constitutes as a more suitable deep clustering objective function capable of enhancing the learned representations of the embedded space for clustering purposes.

Chapter 6

Deep Clustering Based on Implicit Maximum Likelihood

6.1 Introduction

In this Chapter [91], we propose a neural clustering method called *Neural Implicit Maximum Likelihood Clustering* (NIMLC). It is a generative clustering method that relies on the recently proposed method of Implicit Maximum Likelihood Estimation (IMLE) [180]. This is an alternative approach to GANs [4], which is introduced in Chapter 1 and Section 1.3. In particular, given a set of data objects, the IMLE method uses a generator network that takes random input vectors and learns to produce synthetic samples. By minimizing an appropriate objective, the network is trained so that the distribution of samples resembles the data distribution. It has been shown that this training procedure maximizes the likelihood of the dataset without explicitly computing the likelihood.

In analogy to the ClusterGan [1] method, which exploits the GAN methodology to perform clustering, we have developed the NIMLC method, which relies on the

IMLE methodology to perform clustering. NIMLC utilizes two neural networks, the generator and the encoder. In contrast to ClusterGAN, the discriminator network is not needed. The generator network is fed by appropriately selected random samples (latent vectors) z belonging to K clusters and is trained to produce synthetic samples that resemble the objects of dataset X. The encoder network provides the partition of the dataset X into K clusters by learning the inverse map from the data space X to the latent space Z. Training of both networks is achieved by minimizing an appropriately defined objective function that involves the IMLE loss (data generation) and the reconstruction loss for the latent vectors z.

Note that the IMLE method does not suffer from mode collapse, vanishing gradients, or training instability that are frequently encountered in GAN training. Moreover, it does not require large datasets for training. Our aim is to exploit those nice IMLE properties for solving clustering problems through the development of the proposed NIMLC method.

The organization of the Chapter is the following. In Section 6.2 the IMLE method is first described and then the proposed NIMLC clustering method is presented and explained. Finally, Section 6.3 presents comparative experimental results on various datasets, while Section 6.4 summarizes this Chapter.

6.2 Neural Implicit Maximum Likelihood Clustering

The proposed NIMLC method relies on the data generation capabilities of the IMLE algorithm, which is summarized next.

6.2.1 Implicit Maximum Likelihood Estimation

Given a dataset $X = \{x_1, \ldots, x_n\}$ of d-dimensional vectors, the IMLE algorithm [180] trains a generative neural network $\mathcal{G}_\theta$ with m inputs, d outputs and parameter vector (weights) θ. This generator takes as input a random vector $z \in \mathbb{R}^m$ usually sampled from an m-dimensional Normal distribution and produces a sample $s^\theta \in \mathbb{R}^d$, i.e., $s^\theta = \mathcal{G}_\theta(z)$ (see Fig. 6.2a). IMLE trains the generator to generate synthetic samples s^θ that resemble the real data x_i. It is a simple generative method that, under certain conditions, implicitly maximizes the likelihood of the dataset, although the IMLE objective does not explicitly contain any log-likelihood term, and training neural

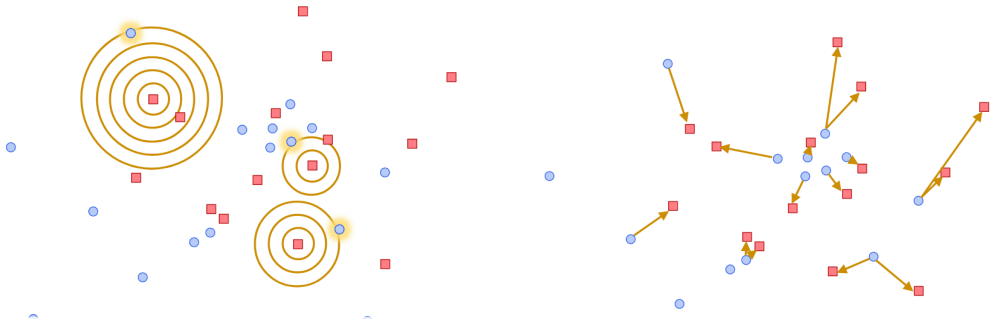
networks using maximum likelihood is considered a difficult task [181].

In each IMLE iteration, a sampling procedure takes place where a set of L random input vectors z_i (called latent vectors) are drawn from the Normal distribution $z_i \sim \mathcal{N}(0, \sigma^2 I_m)$ and used for the computation of the corresponding synthetic samples $s_i^\theta = \mathcal{G}_\theta(z_i)$ $(i = 1, \ldots, L)$. Then, for each real data example x_i $(i = 1, \ldots, N)$, its representative sample $r_i^\theta \in S^\theta$ is determined through nearest neighbor search (NNS) in S^θ based on Euclidean distance, i.e. $r_i^\theta = NNS(x_i, S^\theta)$. The generator parameters θ are updated in order to minimize the following IMLE objective function:

$$\hat{\theta}_{IMLE} = argmin_\theta \sum_{i=1}^{n} ||r_i^\theta - x_i||^2 \tag{6.1}$$

Figure 6.1 provides an illustration of the IMLE behavior.

The IMLE method exhibits several nice properties: it does not suffer from mode collapse, vanishing gradients, or training instability, unlike popular deep generative methods such as, for example, GANs [4]. Mode collapses do not occur, since the loss ensures that each data example is represented by at least one sample. Gradients do not vanish because the gradient of the distance between a data example and its representative sample does not become zero unless they coincide. Training is stable because the IMLE estimator is the solution to a simple minimization problem. Finally, it can be used both for small and large datasets.



(a) (b)

Figure 6.1: The data points are represented by squares and the samples by circles. (a) For each data point the nearest sample is found. (b) The generator is updated at each iteration so that the generated samples minimize the IMLE objective.

6.2.2 Cluster friendly input distribution

In the original IMLE method, the random input (latent) vectors z belong to a single cluster since they are drawn from a multivariate m-dimensional Normal distribution. This is not convenient for clustering. If we assume that the input vectors z are drawn from a mixture model, i.e., from K distinct distributions, then a clustering of the original dataset X could be obtained: each data point x_i can be assigned to the cluster to which its corresponding input vector z_i belongs to. Therefore, in the proposed method, the single Normal distribution is replaced by K non-overlapping distributions, with the k-th distribution responsible for the generation of the subset Z_k of input vectors assigned to cluster k. The most obvious first choice is a mixture of K m-dimensional Gaussian distributions. However, this choice requires the specification of the means and covariances of K Gaussian distributions so that they are well separated.

A more sophisticated mechanism for generating m-dimensional random vectors that belong to K disjoint clusters has been proposed in ClusterGan [1], where input vector z consists of two parts, i.e., $z = (z_n, z_c)$. The first part z_n is random vector (of dimension d_n) drawn from the Gaussian distribution: $z_n \sim \mathcal{N}(0, \sigma^2 I_{d_n})$. The second part z_c, is deterministic and specifies the cluster k to which z is assigned. Specifically, z_c is the one-hot encoding of the corresponding cluster k. Thus, for K clusters, the dimension of z_c is equal to K and, if z belongs to the k-th cluster, then $z_c = e_k$ where e_k is the k-th standard unit vector. Note that σ should be set to a small value so that clusters do not overlap.

In summary, in order to generate an input vector $z = (z_c, z_n)$ belonging cluster k, we set the z_c part equal to the one-hot encoding of k and draw the z_n part from $\mathcal{N}(0, \sigma^2 I_{d_n})$. By sampling an equal number of vectors for each cluster k, the set of random input vectors Z is created at each iteration which is partitioned into disjoint subsets Z_k, each one containing the random input vectors for cluster k ($k = 1, \ldots, K$).

Additionally, since $s^\theta = \mathcal{G}_\theta(z)$, the set S^θ of computed samples is partitioned into K disjoint clusters S_k^θ. Consequently, the original dataset X can be partitioned into K clusters by assigning each x_i to the cluster of its representative r_i^θ, i.e. if $r_i^\theta \in S_k^\theta$ then x_i is assigned to cluster k.

6.2.3 The IMLE loss from a clustering perspective

If we examine the IMLE objective function, we can observe its similarities with the k-means clustering loss. Specifically, if we generate exactly K samples $S_K^\theta = \{s_1^\theta, \ldots, s_K^\theta\}$ in each training epoch, where K is the number of clusters, we can treat those synthetic samples as cluster representatives (centroids). In this case, the IMLE objective coincides with the k-means objective ($\mathbb{1}_{C_k}$ is the indicator function):

$$\sum_{i=1}^{N} ||x_i - r_i^\theta||^2 = \sum_{i=1}^{N} ||x_i - NNS(x_i, S_K^\theta)||^2 = \sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{1}_{C_k}(x_i)||x_i - s_k^\theta||^2 \tag{6.2}$$

and IMLE can be considered as a clustering procedure that trains the generator to produce the cluster centers. The major difference between k-means and IMLE is that the k-means updates the centroids directly in order to minimize the clustering loss; on the contrary, the IMLE method updates the parameters θ of the generator.

An issue to be considered is how to specify the k input vectors z_k that will be used to generate the K samples so that each sample represents a different cluster. Since $z_k = (z_{nk}, z_{ck})$, a straightforward solution is to set $\sigma = 0$, thus $z_{nk} = 0$ for all k and $z_{ck} = e_k$ for all $k = 1, \ldots, K$. Then by feeding those z_k vectors as inputs to the generator, the synthetic samples s_k are provided as outputs which can be treated as cluster representatives. Training the generator this way using IMLE, we observed clustering behavior similar to k-means and that the generated K samples resembled the average data point of each cluster.

6.2.4 The NIMLC architecture

The proposed NIMLC approach is a modification of the IMLE method in order to achieve not only synthetic data generation but also clustering of the original dataset X. NIMLC combines ideas from IMLE and ClusterGAN. More specifically, it exploits the IMLE generator network that is fed with clustered input vectors z that follow the (z_n, z_c) representation proposed in ClusterGAN. Additionally, it employs a second network called encoder (originally proposed in ClusterGAN) that is trained to provide the cluster assignment for a data point x. It should be noted that, unlike ClusterGAN, NIMLC does not make use of a discriminator network since it is based on IMLE for synthetic data generation. The NIMLC architecture is presented in Figure 6.2b.

The generator $\mathcal{G}$ is trained to produce synthetic samples that resemble the real data x_i by minimizing the IMLE objective (eq. 6.1). It provides a mapping from the

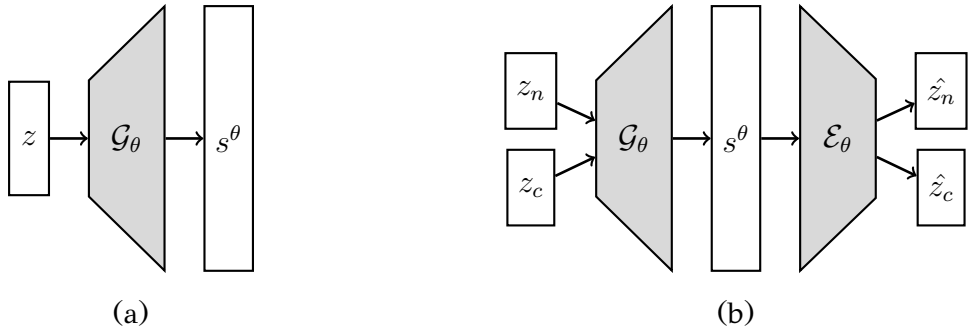


Figure 6.2: (a) IMLE general architecture. (b) NIMLC architecture.

latent space to the data space. The encoder $\mathcal{E}$ is trained jointly with the generator to implement the inverse mapping from the data space to the latent space. Thus, for an input x, it provides estimates of $\hat{z}_n$ and $\hat{z}_c$. The latter ($\hat{z}_c$) is computed using the softmax activation function (with K outputs) and provides a soft clustering assignment of the input x into K clusters.

In summary, the NIMLC architecture feeds an input vector $z = (z_n, z_c)$ to the generator, which produces a synthetic sample $s = \mathcal{G}(z)$. This sample is subsequently fed to the encoder, which provides the output $\hat{z} = \mathcal{E}(s)$. Note that the NIMLC network is actually an autoencoder since it takes an input z and provides as output an estimate $\hat{z}$ of z. After training, the encoder implements a clustering model providing soft clustering assignments $\hat{z}_c$ for any data point x.

6.2.5 The NIMLC objective function

The objective function used to train the NIMLC architecture consists of two parts. The first part concerns the generative process and is the IMLE error equal to $\sum_{i=1}^{n} ||r_i^{\theta_\mathcal{G}} - x_i||^2$ (eq. 6.1). Since NIMLC is an autoencoder, the second part of the objective function is the reconstruction loss of the autoencoder. This loss can be split into two terms. The first term is the reconstruction loss for the z_n part: $\sum_{i=1}^{n} ||z_{ni} - \hat{z}_{ni}||^2$. The second term is the reconstruction loss for the z_c part. Since z_c has the form of one-hot vector and $\hat{z}_c$ are probability vectors provided by the softmax function, the cross-entropy $\mathcal{H}(z_c, \hat{z}_c)$ between z_c and $\hat{z}_c$ is used as a loss function.

The complete objective function is presented below, where β_n and β_c are hyper-parameters adjusting the importance of each term.

$$J(\theta_{\mathcal{G}}, \theta_{\mathcal{E}}) = \sum_{i=1}^{n} ||r_i^{\theta_{\mathcal{G}}} - x_i||^2 + \beta_n \sum_{i=1}^{n} ||z_{ni} - \hat{z}_{ni}||^2 + \beta_c \sum_{i=1}^{n} \mathcal{H}(z_{ci}, \hat{z}_{ci}) \tag{6.3}$$

It should be noted that the first term depends only on the parameters $\theta_{\mathcal{G}}$ of the generator, while the rest two terms depend on the parameters of both the generator $\theta_{\mathcal{G}}$ and the encoder $\theta_{\mathcal{E}}$.

6.2.6 Slow paced learning

A critical hyperparameter of the NIMLC method is the standard deviation σ of the noise distribution used to generate the random part z_n of the input vectors. As mentioned earlier, when training the model with $\sigma = 0$, we have a very strict case with one generated sample per cluster. This sample can be considered as the representative of the corresponding cluster, and the obtained clustering results are on par with those of k-means. On the other hand, if σ is relatively large (e.g. $\sigma = 0.15$), the random input vectors per cluster are not very close. Therefore, it is possible for the generator to map the inputs of the same cluster to different regions in the data space, which negatively affects clustering performance. Moreover, we have observed that it is difficult to specify an appropriate value for σ.

In order to tackle this problem, we propose the following procedure:

- Start training with a small value of sigma, preferably $\sigma = 0$.
- In each training epoch increase σ by a small amount $\Delta\sigma$.
- Stop increasing when a max value σ_{max} is reached.

The intuition is that this slow-paced training procedure ([182], [183]) strives to learn and cluster the "easier" data points first, like those that are close to the cluster centers and then tries to learn and cluster "more difficult" data points away from the cluster centers. Thus, we initially start to explore the clustering solution space with no variability in the input space ($\sigma = 0$). In this way, we enforce only K samples to be generated and used to train the model. Then, at each training epoch, we add variability to the inputs by slowly increasing σ in order to incrementally capture complicated structures in the dataset.

Figure 6.3 provides an illustration of the generated samples for the Moons synthetic dataset as training proceeds and σ gradually increases. It is clear that the

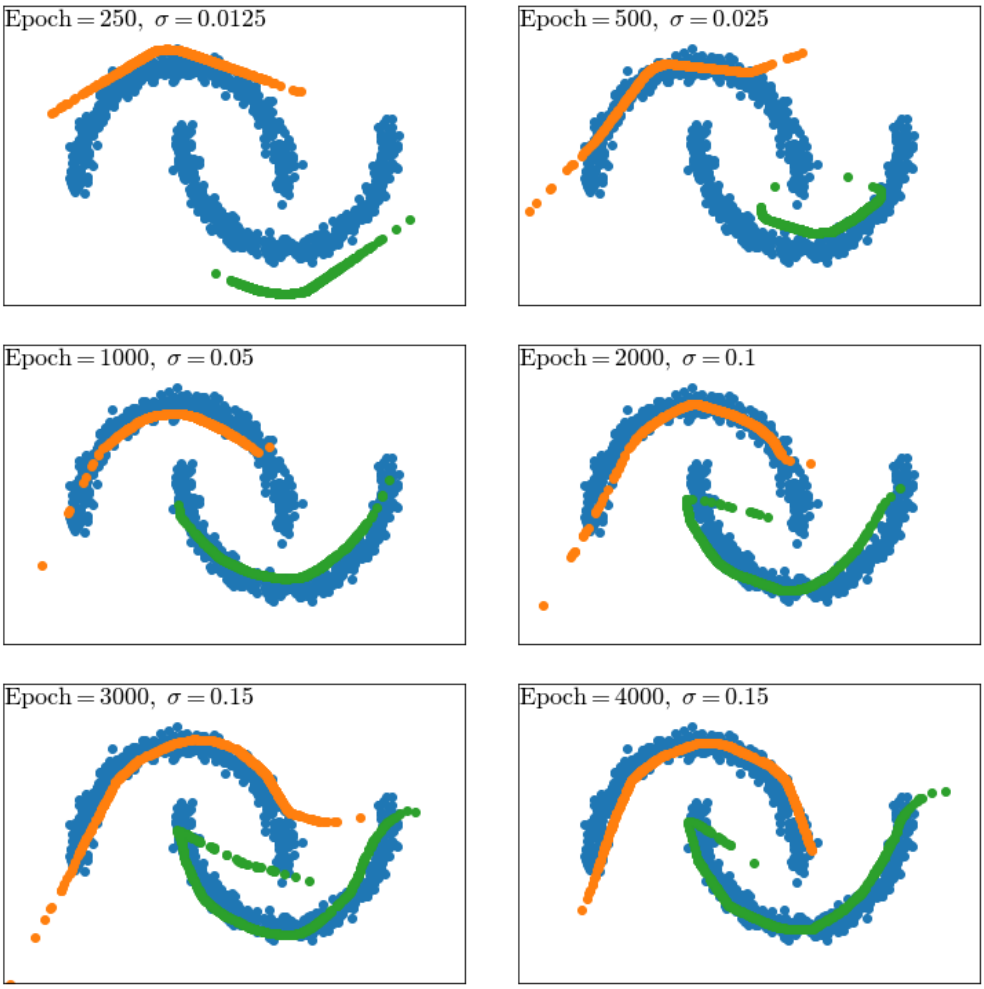


Figure 6.3: The evolution of generated samples for the Moons synthetic dataset as σ progressively increases.

model progressively succeeds in learning more complex data structures, generating high-quality samples and providing the correct clustering solution.

6.2.7 The NIMLC algorithm

The NIMLC method is summarized in Algorithm 6.1. At each epoch, a set of input vectors $Z = \{z_1, \ldots, z_L\}$ is generated, belonging to K clusters Z_k, $k = 1, \ldots, K$ of equal size. Each input vector $z_i = (z_{ni}, z_{ci})$ of Z_k is computed by sampling z_{ni} from $\mathcal{N}(0, \sigma^2 I_{d_n})$ and setting $z_{ci} = e_k$. We then feed the generator with the set of input vectors Z and the set of synthetic samples $S^{\theta_\mathcal{G}} = \{s_1^{\theta_\mathcal{G}}, \ldots, s_L^{\theta_\mathcal{G}}\}$ are generated at its output, i.e. $s_i^{\theta_\mathcal{G}} = \mathcal{G}(z_i)$. Then for each data batch X_b, we compute the nearest synthetic sample r_i for each $x_i \in X_b$, ie. $r_i^{\theta_\mathcal{G}} = NNS(x_i, S^{\theta_\mathcal{G}})$. Next, each r_i is fed as input to the

encoder that produces the reconstruction $\hat{z}_i = \mathcal{E}(r_i^{\theta_\mathcal{G}})$, where $\hat{z}_i = (\hat{z}_{ni}, \hat{z}_{ci})$. Then we update the parameters of the generator and the encoder using the gradients of the objective function (algorithm 6.1 steps 11 and 12). Finally, before proceeding to the next epoch, the standard deviation σ is updated.

Algorithm 6.1 Neural Implicit Maximum Likelihood Clustering

Require: X (dataset)

Require: K, L, T (number of clusters, number of samples, number of epochs)

Require: $\mathcal{G}, \mathcal{E}$ (generator, encoder)

Require: d_n (dimension of noise vector z_n)

Require: $\beta_n \leftarrow 1, \beta_c \leftarrow 1$ (regularization parameters)

Require: $\Delta\sigma \leftarrow 5 * 10^{-5}, \sigma_{max} \leftarrow 0.15$ (standard deviation increase, maximum standard deviation)

1: Initialize networks parameters $\theta_\mathcal{G}$ and $\theta_\mathcal{E}$.

2: $\sigma \leftarrow 0$

3: **for** $epoch \leftarrow 1$ to T **do**

4: Compute a set of L input vectors $Z = \{z_1, \ldots, z_L\}$ belonging to K equally sized subsets Z_k. The elements of each Z_k are computed as $z_i = (z_{ni}, z_{ci})$ where z_{ni} is drawn from $\mathcal{N}(0, \sigma^2 I_{d_n})$ and $z_{ci} = e_k$.

5: Compute samples $S^{\theta_\mathcal{G}} = \{s_1^{\theta_\mathcal{G}}, \ldots, s_L^{\theta_\mathcal{G}}\}$, where $s_i^{\theta_\mathcal{G}} = \mathcal{G}(z_i)$.

6: **for** $b \leftarrow 1$ to number of batches **do**

7: For each x_i in batch b find its nearest neighbour $r_i^{\theta_\mathcal{G}} \in S$ considering the Euclidean distance ($r_i^{\theta_\mathcal{G}} = NNS(x_i, S^{\theta_\mathcal{G}})$).

8: For each $r_i^{\theta_\mathcal{G}}$ compute $\hat{z}_i = \mathcal{E}(r_i^{\theta_\mathcal{G}})$, where $\hat{z}_i = (\hat{z}_{ni}, \hat{z}_{ci})$.

9: Update the generator parameters by descending their stochastic gradient:

$$\nabla_{\theta_\mathcal{G}} \left\{ \sum_{i=1}^{n} ||r_i^{\theta_\mathcal{G}} - x_i||^2 + \nabla_{\theta_\mathcal{G}} \left(\beta_n \sum_{i=1}^{n} ||z_{ni} - \hat{z}_{ni}||^2 + \beta_c \sum_{i=1}^{n} \mathcal{H}(z_{ci}, \hat{z}_{ci}) \right) \right\}$$

10: Update the encoder parameters by descending its stochastic gradient:

$$\nabla_{\theta_\mathcal{E}} \left(\beta_n \sum_{i=1}^{n} ||z_{ni} - \hat{z}_{ni}||^2 + \beta_c \sum_{i=1}^{n} \mathcal{H}(z_{ci}, \hat{z}_{ci}) \right)$$

11: **end for**

12: $\sigma \leftarrow \min(\sigma + \Delta\sigma, \sigma_{max})$

13: **end for**

14: **return** the final network parameters $\theta_\mathcal{G}^*$ and $\theta_\mathcal{E}^*$.

It should be noted that using IMLE for clustering has been introduced in our previous work [184]. However, that method did not make use of the encoder network. Instead, a two-stage nearest neighbor search was used to perform cluster assignments. Specifically, in the first stage, the centroid c_k of each subset S_k^θ was computed, and

then the x_i was assigned to the cluster l whose centroid c_l is nearest to x_i based on Euclidean distance. Additionally, in the second stage, instead of determining the representative sample for x_i through the nearest neighbor search over the entire set of samples S^θ, the nearest neighbor search was executed only to the specific subset S_l^θ that contains the samples of cluster l. The NIMLC method proposed herein includes two substantial improvements that lead to considerable performance enhancement. The first is the use of the encoder network that directly provides the cluster assignment for a given input x, while the second is the gradual increase of the noise variance σ that allows for slow-paced learning. Additionally, we exploit the generalization capability of the encoder network to cluster those data points that the generator could not learn sufficiently.

A computational overhead of our approach compared to deep clustering methods is related to nearest neighbor search. The training process involves several epochs where the algorithm must find the closest synthetic sample to each original data point, resulting in an $\mathcal{O}(NL)$ overhead in distance calculations. However, we observed that recalculating the nearest neighbors in every training epoch is unnecessary; reusing them for 5 to 10 epochs can significantly reduce the training time without compromising clustering performance.

6.3 Experiments

In order to evaluate the proposed clustering method (NIMLC), we conducted an experimental study using several synthetic and real datasets. We have compared NIMLC against ClusterGan [1] and the two most popular deep clustering methods, namely DCN [71] and DEC [73]. We also provide results using k-means [25, 26] and the density-based method of i-DivClu-D [185].

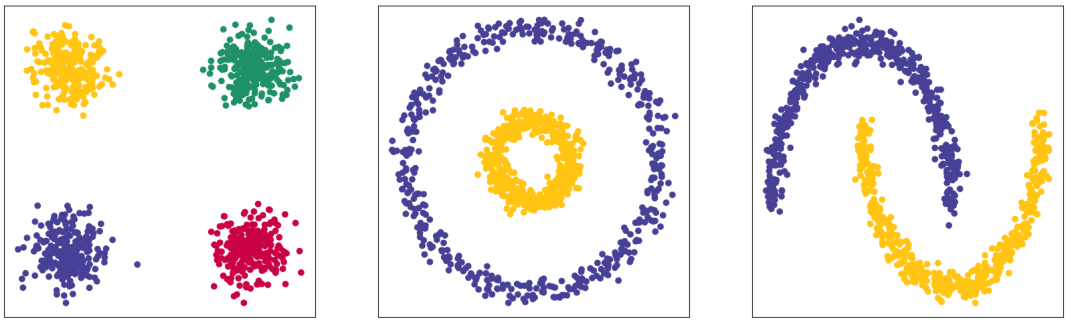
6.3.1 Synthetic datasets

We have used three synthetic two-dimensional datasets (Table 6.1) with known ground truth and different structures in order to assess the clustering capability of our method. The Gaussians dataset consists of four clusters (Figure 6.4a), while the Moons (Figure 6.4b) and the Rings (Figure 6.4c) consist of two clusters. The Gaussians dataset is easier to cluster compared to the other two datasets, whose structure

is more complex. It should be emphasized that it is difficult for a parametric method to be able to cluster both cloud-shaped (Gaussians) and ring-shaped (Rings) datasets.

Table 6.1: Description of synthetic datasets.

Dataset	# Points (N)	# Features (D)	# Clusters (K)
Gaussians	1000	2	4
Moons	1000	2	2
Rings	1000	2	2



(a) Gaussians (b) Rings (c) Moons

Figure 6.4: The synthetic datasets used in our experiments.

6.3.2 Real datasets

We further evaluated the method by including real datasets in our experimental study. For all datasets the number of clusters was set equal to the number of classes. As a pre-processing step, we used min-max normalization to map the attributes of each dataset to the $[0, 1]$ interval in order to prevent attributes with large ranges from dominating the distance calculation and avoid numerical instabilities in the computation [97]. The descriptions of the datasets that we included in our study are given below. For a summary, refer to Table 6.2.

• **10x_73k** [186] dataset consists of 73233 RNA-transcripts belonging to 8 different cell types. The dataset is sparse since the data matrix has about 40% zero values. Hence, we selected the 720 genes with the highest variances across the cells to reduce data dimensionality similar to [1].

• **Australian** [95] two-class dataset is composed of 690 credit card applications. A 14-dimensional feature vector describes each sample.

• **CMU** [95] contains grayscale facial images of twenty individuals captured with varying poses, expressions, and the presence or absence of glasses. The images are available in several resolutions, but for the purpose of our study, we have utilized the 128×120 resolution images.

• **Dermatology** [95] is a six-class dataset containing 366 patient records that suffer from six different types of Eryhemato-Squamous disease. Each patient is described by a 34-dimensional vector containing clinical and histopathological features.

• **E.coli** [95] includes 336 proteins from the E.coli bacterium, and seven attributes, calculated from the amino acid sequences, are provided. Proteins belong to eight classes according to their cellular localization sites.

• **Iris** [95] dataset contains three classes of 50 instances each, where each class refers to a type of iris plant. Each sample is described by a 4-dimensional vector, corresponding to the length and width of the sepals and petals in centimeters.

• **Olivetti** [187] is a face database of 40 individuals with ten 64×64 grayscale images per individual. For some individuals, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling/not smiling), and facial details (glasses/no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

• **Optical Recognition of Handwritten Digits** [95] dataset (ORHD) comprises a set of handwritten digits, with ten classes corresponding to each digit from 0 to 9. The resolution of each image is 8x8. For our experiment, we utilized the test set of this dataset, which consists of 1797 images.

• **Pendigits** [95] dataset consists of 250 writing samples from 44 different writers, a total of 10992 written samples. Each sample is a 16-dimensional vector containing pixel coordinates associated with a label from ten classes.

• **United States Postal Service** [188] dataset (USPS) is a collection of hand-written digits consisting of 7291 grayscale images. The dataset is organized into ten classes, each representing a digit from 0 to 9. Each digit is represented by a set of images, each of size 16×16 pixels.

• **Wine** [95] three-class dataset consists of 178 samples of chemical analysis of wines. A 13-dimensional feature vector describes each sample.

Table 6.2: Descriptions of real datasets.

Dataset	**#Points (N)**	**#Features (D)**	**#Classes (K)**	**Source**
10x_73k	73233	720	8	[186]
Australian	690	14	2	[95]
CMU	640	128×120	20	[95]
Dermatology	366	34	6	[95]
E.coli	336	7	8	[95]
Iris	150	4	3	[95]
Olivetti-Faces	400	64×64	40	[187]
ORHD	1797	8×8	10	[95]
Pendigits	7494	16	10	[95]
USPS	7291	16×16	10	[188]
Wine	178	13	3	[95]

6.3.3 Evaluation measures

It is important to mention that since clustering is an unsupervised problem, we ensured that all algorithms were unaware of the true clustering of the data. In order to evaluate the results of the clustering methods, we use standard external evaluation measures [161], which assume that ground truth clustering is available. For all algorithms, the number of clusters is set to the number of ground-truth categories [16] and assumes ground truth that cluster labels coincide with class labels. The first evaluation measure is *clustering accuracy* (ACC):

$$ACC(Y, C) = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}(y_i = m(c_i))}{n} \tag{6.4}$$

where $\mathbf{1}(x)$ is the indicator function, y_i is the ground-truth label, c_i is the cluster assignment generated by the clustering algorithm, and m is a mapping function which ranges over all possible one-to-one mappings between assignments and labels. This measure finds the best matching between cluster assignments from a clustering method and the ground truth. It is worth noting that the optimal mapping function can be efficiently computed by the Hungarian algorithm [189]. The second evaluation measure is *purity* (PUR). The same equation formulates purity as clustering accuracy (eq. 6.4), but their key difference is in the mapping function m. In this case, the mapping function of m greedily assigns clustering labels to ground truth categories in each cluster in order to maximize purity. The third evaluation measure is the

normalized mutual information (NMI) defined as [162]:

$$NMI(Y, C) = \frac{2 \times I(Y, C)}{H(Y) + H(C)} \tag{6.5}$$

where Y denotes the ground-truth labels, C denotes the clusters labels, I is the mutual information measure and H the entropy. The final evaluation metric is the *adjusted Rand Index* (ARI) [160, 163], which computes a similarity measure between two clustering solutions defined as the proportion of object pairs that are either assigned to the same cluster in both clusterings or to different clusters in both clusterings.

6.3.4 Implementation Details

Both the generator and the encoder were trained using the Adam optimizer [86] with learning rate $n = 3 * 10^{-4}$ and coefficients $b_1 = 0.5$ and $b_2 = 0.9$. We set $b_n = b_c = 1$, and $\Delta\sigma = 5 * 10^{-5}$ in all experiments. Additionally, the number of samples was set equal to 100 and 200 for small and big datasets, respectively. We used the same architectures for the two networks as the ClusterGan [1]. Specifically, the dimension of z_c is the set equal to the number of clusters. We used Leaky Relu activations (LRelu) with leak = 0.2 and Batch Normalization (BN). We used the same number of hidden layers and hidden neurons for all datasets. We present the detailed generator and encoder architectures in Table 6.3 and Table 6.4, respectively.

Table 6.3: Generator architecture for each dataset.

Dataset	**Input** (z_n, z_c)	**Hidden {1, 2}**	**Output** $(\hat{x})$
10x_73k	$(10, 8)$	FC 256 LReLU BN	FC 720 Sigmoid
Australian	$(5, 2)$	FC 256 LReLU BN	FC 14 Sigmoid
CMU	$(100, 20)$	FC 256 LReLU BN	FC 128×120 Sigmoid
Dermatology	$(5, 6)$	FC 256 LReLU BN	FC 34 Sigmoid
E.coli	$(4, 8)$	FC 256 LReLU BN	FC 7 Sigmoid
Gaussians	$(1, 4)$	FC 256 LReLU BN	FC 2 Sigmoid
Iris	$(2, 3)$	FC 256 LReLU BN	FC 4 Sigmoid
Moons	$(1, 2)$	FC 256 LReLU BN	FC 2 Sigmoid
Olivetti-Faces	$(10, 40)$	FC 256 LReLU BN	FC 64×64 Sigmoid
ORHD	$(100, 10)$	FC 256 LReLU BN	FC 8×8 Sigmoid
Pendigits	$(5, 10)$	FC 256 LReLU BN	FC 16 Sigmoid
Rings	$(1, 2)$	FC 256 LReLU BN	FC 2 Sigmoid
USPS	$(100, 10)$	FC 256 LReLU BN	FC 16×16 Sigmoid
Wine	$(2, 3)$	FC 256 LReLU BN	FC 3 Sigmoid

For ClusterGan, we used the proposed architecture and hyperparameters. In the case of the DCN and DEC, an extensive search for an autoencoder model was required

Table 6.4: Encoder architecture for each dataset.

Dataset	**Input** (x)	**Hidden {1, 2}**	**Output** $(\hat{z_n}, \hat{z_c})$
10x_73k	720	FC 256 LReLU BN	FC 18 LReLU for $\hat{z}$ and 8 Softmax for $\hat{z_c}$
Australian	14	FC 256 LReLU BN	FC 7 LReLU for $\hat{z}$ and 2 Softmax for $\hat{z_c}$
CMU	128×120	FC 256 LReLU BN	FC 120 LReLU for $\hat{z}$ and 20 Softmax for $\hat{z_c}$
Dermatology	34	FC 256 LReLU BN	FC 11 LReLU for $\hat{z}$ and 6 Softmax for $\hat{z_c}$
E.coli	7	FC 256 LReLU BN	FC 12 LReLU for $\hat{z}$ and 8 Softmax for $\hat{z_c}$
Gaussians	2	FC 256 LReLU BN	FC 5 LReLU for $\hat{z}$ and 4 Softmax for $\hat{z_c}$
Iris	4	FC 256 LReLU BN	FC 5 LReLU for $\hat{z}$ and 3 Softmax for $\hat{z_c}$
Moons	2	FC 256 LReLU BN	FC 3 LReLU for $\hat{z}$ and 2 Softmax for $\hat{z_c}$
Olivetti-Faces	64×64	FC 256 LReLU BN	FC 50 LReLU for $\hat{z}$ and 40 Softmax for $\hat{z_c}$
ORHD	8×8	FC 256 LReLU BN	FC 120 LReLU for $\hat{z}$ and 10 Softmax for $\hat{z_c}$
Pendigits	16	FC 256 LReLU BN	FC 15 LReLU for $\hat{z}$ and 10 Softmax for $\hat{z_c}$
Rings	2	FC 256 LReLU BN	FC 3 LReLU for $\hat{z}$ and 2 Softmax for $\hat{z_c}$
USPS	16×16	FC 256 LReLU BN	FC 120 LReLU for $\hat{z}$ and 10 Softmax for $\hat{z_c}$
Wine	3	FC 256 LReLU BN	FC 5 LReLU for $\hat{z}$ and 3 Softmax for $\hat{z_c}$

in order to obtain good results. We chose symmetrical encoder and decoder networks to simplify the architecture search problem. We resorted to an encoder architecture with three layers : $d - [2d, 3d] - d_z$, where d is the data space dimension and d_z is the latent space dimension. All layers are fully connected.

NIMLC and ClusterGan methods were executed for 5000 epochs, while DCN and DEC required 300 to 500 epochs of pretraining and 100 epochs of training with the clustering objective. Furthermore, for the methods that depend on initialization, we executed the neural approaches of NIMLC, ClusterGan, DCN, and DEC three times and the k-means algorithm ten times with k-means++ [29] initialization.Average performance results are provided. The i-DivClu-D method is deterministic and requires the number of nearest neighbors as a hyperparameter. In our experiments we set its value at the minimum number that resulted in a connected graph.

6.3.5 Results on synthetic datasets

In Table 6.5, we provide the average clustering performance of the compared methods for the synthetic datasets. All methods performed well when the dataset consisted of spherical, well-separated data clusters, as happens in the Gaussians dataset. In the Moons and Rings datasets, ClusterGan and DEC had a similar clustering performance as the k-means algorithm, while the DCN method performed better. On the other hand, the NIMLC method could perfectly solve the Moons dataset and had by a

Table 6.5: Experimental results on synthetic datasets. Bold numbers indicate the best average performance on each dataset.

Dataset	Algorithm	ACC	PUR	NMI	ARI
Rings	NIMLC	0.90	0.90	0.62	0.64
	ClusterGAN	0.52	0.52	0.00	0.00
	DCN	0.62	0.62	0.05	0.06
	DEC	0.52	0.52	0.00	0.00
	k-means	0.50	0.50	0.00	0.00
	i-DivClu-D	**1.00**	**1.00**	**1.00**	**1.00**
Moons	NIMLC	**1.00**	**1.00**	**1.00**	**1.00**
	ClusterGAN	0.86	0.86	0.43	0.53
	DCN	0.90	0.90	0.63	0.65
	DEC	0.86	0.86	0.44	0.53
	k-means	0.86	0.86	0.42	0.52
	i-DivClu-D	**1.00**	**1.00**	**1.00**	**1.00**
Gaussians	NIMLC	**1.00**	**1.00**	**1.00**	**1.00**
	ClusterGAN	**1.00**	**1.00**	**1.00**	**1.00**
	DCN	0.91	0.91	0.93	0.89
	DEC	0.98	0.98	0.94	0.96
	k-means	**1.00**	**1.00**	**1.00**	**1.00**
	i-DivClu-D	**1.00**	**1.00**	**1.00**	**1.00**

significant margin the best clustering performance on Rings, which is the most difficult of the three synthetic datasets. It should be stressed that the NIMLC method presents the unique capability of solving both the Gaussian and the Rings datasets by training the same neural architecture. It should be noted that the density-based i-DivClu-D method demonstrated perfect clustering performance in all three synthetic datasets.

Table 6.6: Experimental results on real datasets. Bold numbers indicate the best average performance for each dataset. Results marked by "*" are excerpted from the paper proposing the method.

Dataset	Algorithm	ACC	PUR	NMI	ARI
10x_73k	NIMLC	**0.81**	**0.84**	0.73	**0.69**
	ClusterGAN	NA	0.81*	0.73*	NA
	DCN	0.70	0.70	**0.74**	0.68
	DEC	0.67	0.72	0.72	0.57
	k-means	0.56	0.61	0.56	0.36
	i-DivClu-D	0.55	0.60	0.63	0.36
Australian	NIMLC	**0.86**	**0.86**	**0.43**	**0.50**
	ClusterGAN	**0.86**	**0.86**	**0.43**	**0.50**
	DCN	**0.86**	**0.86**	**0.43**	**0.50**
	DEC	0.77	0.77	0.25	0.31
	k-means	**0.86**	**0.86**	**0.43**	**0.50**
	i-DivClu-D	0.59	0.59	0.01	0.02
Dermatology	NIMLC	0.81	0.82	0.81	0.70
	ClusterGAN	0.72	0.77	0.68	0.56
	DCN	**0.83**	0.83	**0.88**	**0.81**
	DEC	0.77	0.86	0.86	0.74
	k-means	0.78	**0.87**	**0.88**	0.74
	i-DivClu-D	0.68	0.78	0.74	0.58
E.coli	NIMLC	**0.61**	0.76	0.54	**0.45**
	ClusterGAN	0.49	0.76	0.50	0.33
	DCN	0.52	0.70	0.46	0.34
	DEC	0.55	0.76	0.54	0.42
	k-means	0.60	**0.83**	**0.62**	0.43
	i-DivClu-D	0.53	0.78	0.53	0.33
Iris	NIMLC	0.95	0.95	0.83	0.85
	ClusterGAN	0.89	0.89	0.73	0.72
	DCN	0.93	0.93	0.83	0.80
	DEC	**0.97**	**0.97**	**0.91**	**0.92**
	k-means	0.89	0.89	0.74	0.72
	i-DivClu-D	0.93	0.93	0.83	0.80
Pendigits	NIMLC	**0.84**	**0.84**	**0.79**	**0.71**
	ClusterGAN	NA	0.77*	0.73*	NA
	DCN	0.72	0.72	0.69	0.56
	DEC	0.70	0.74	0.73	0.58
	k-means	0.68	0.71	0.69	0.54
	i-DivClu-D	0.70	0.73	0.71	0.57
Wine	NIMLC	0.93	0.93	0.80	0.80
	ClusterGAN	0.58	0.58	0.28	0.22
	DCN	0.93	0.93	0.80	0.80
	DEC	0.94	0.94	0.82	0.83
	k-means	**0.95**	**0.95**	**0.84**	**0.85**
	i-DivClu-D	0.92	0.92	0.77	0.77

6.3.6 Results on real datasets

Table 6.6 shows the clustering performance of the compared methods on tabular datasets, while Table 6.7 displays performance results on image datasets.

Table 6.7: Experimental results on real datasets. Bold numbers indicate the best average performance for each dataset. Results marked by "-" denotes the method was not able to learn the dataset.

Dataset	Algorithm	ACC	PUR	NMI	ARI
CMU	NIMLC	**0.82**	**0.84**	**0.92**	**0.80**
	ClusterGAN	-	-	-	-
	DCN	-	-	-	-
	DEC	-	-	-	-
	k-means	**0.82**	0.82	0.89	0.77
	i-DivClu-D	0.67	0.71	0.82	0.57
ORHD	NIMLC	0.84	0.84	0.78	0.73
	ClusterGAN	**0.85**	**0.85**	**0.84**	**0.78**
	DCN	0.79	0.79	0.75	0.64
	DEC	0.80	0.81	0.80	0.71
	k-means	0.80	0.80	0.74	0.67
	i-DivClu-D	**0.85**	**0.85**	0.80	0.72
Olivetti-Faces	NIMLC	**0.67**	**0.69**	**0.80**	**0.52**
	ClusterGAN	-	-	-	-
	DCN	-	-	-	-
	DEC	-	-	-	-
	k-means	0.57	0.63	0.78	0.44
	i-DivClu-D	0.47	0.5	0.67	0.29
USPS	NIMLC	0.70	0.70	0.58	0.55
	ClusterGAN	**0.74**	**0.78**	**0.71**	**0.63**
	DCN	0.69	0.76	0.70	0.60
	DEC	0.72	0.77	**0.71**	**0.63**
	k-means	0.68	0.75	0.64	0.56
	i-DivClu-D	0.56	0.65	0.62	0.40

The NIMLC method achieved excellent clustering performance on 10x_73k, Australian, CMU, Olivetti-Faces, and Pendigits, outperforming all other methods. Moreover, on Dermatology, E.coli, Iris, ORHD, USPS, and Wine datasets, the NIMLC method demonstrated comparable results with the best-performing method. It should

be stressed that the high dimensionality of data and the limited number of training samples resulted in training failures for ClusterGan, DCN, and DEC on CMU and Olivetti-Faces datasets. In contrast, the NIMLC method was able to learn these datasets effectively despite the small number of samples. Compared to k-means and the density-based clustering method (i-DivClu-D) our method also provides better or comparative results in all cases with the superiority being more clear on 10x_73k, Pendigits, Oliveti-Faces and USPS datasets. In summary, experimental results indicate that the proposed method constitutes a neural-based clustering method that performs well both for low dimensional and high dimensional data without requiring large number of samples as happens with typical deep clustering methods. Therefore it constitutes a viable alternative for both conventional and deep clustering approaches.

6.4 Summary

In this Chapter, we have proposed the NIMLC clustering method that is based on neural network training. NIMLC is a generative clustering approach that relies on the IMLE generative methodology to perform clustering. The NIMLC brings ideas from the ClusterGAN algorithm into the IMLE framework to overcome some of the GAN deficiencies. The method is based on a simple training objective, does not suffer from training instabilities, and performs well on small datasets. Experimental comparison against several deep clustering methods and conventional clustering methods illustrates the potential of the approach. A notable characteristic of the method is that, as shown in the experiments with synthetic data, it is able to cluster both cloud-shaped and ring-shaped data using the same hyperparameter setting.

Chapter 7

Conclusions and Future Work

7.1 Concluding Remarks

The objective of this thesis was the development, implementation and evaluation of novel (unsupervised) clustering methodologies. During the elaboration of the thesis, we mainly focused on three different axes: i) partitional clustering in both Euclidean and kernel spaces, ii) unimodality-based clustering, and iii) deep clustering, which leverages the representational power of deep learning methods.

Specifically, in Chapter 2, we introduced global k-means++, an approach designed to address the initialization problem inherent in the standard k-means algorithm. This method combines the incremental strategy of global k-means with the probabilistic center selection mechanism of k-means++. In doing so, it effectively balances the strengths of both algorithms, achieving high-quality clustering results while significantly reducing computational overhead. We argue that global k-means++ offers a compelling alternative to both global k-means and k-means++. Moreover, it provides complete clustering solutions for all $k \in 1, \ldots, K$, making it particularly well-suited for model selection tasks where the optimal number of clusters is unknown. In such scenarios, our method demonstrates notable efficiency, outperforming even non-incremental approaches like standard k-means and k-means++.

In Chapter 3, we introduced global kernel k-means++, which constitutes the extension of global k-means++ into feature space. The proposed method is designed to address the initialization problem inherent in the kernel k-means algorithm. Similarly to the global k-means++, the proposed algorithm incrementally addresses all intermediate subproblems for $k = 1, \dots K-1$, ultimately yielding the solution for K clusters. In conclusion, the proposed algorithm strikes an optimal balance between the strengths of both global kernel k-means and kernel k-means++, delivering high-quality clustering at a significantly lower computational cost.

In Chapter 4, we presented the UniForCE clustering method that clusters and estimates the number of clusters k. Our approach is based on the novel definition of locally unimodal cluster. The main idea is that, instead of perceiving unimodality as a property that needs to hold for the whole cluster density, we proposed to study it at a local level, at subregions of the cluster density. We based our approach on the observation that unimodality may extend across pairs of neighboring subclusters when tested as a union. Such unimodal pairs enable the aggregation of small subclusters and the bottom-up formation of larger cluster structures in a statistically sound manner. A locally unimodal cluster extends across subregions of the data density as long as there are unimodal pairs connecting them in a single connected component of the unimodality graph. The proposed locally unimodal cluster definition is flexible as it identifies arbitrary-shaped clusters, including typical unimodal or convex shapes. As part of the proposed methodology, we have developed a statistical procedure to decide on unimodal pairs of subclusters, and we built the unimodality graph in which both clustering and estimation of k can be addressed through the computation of a unimodality spanning forest. The strengths of our contribution's conceptual and algorithmic side have been validated with extensive numerical results using several real and synthetic datasets.

In Chapter 5, we introduced the soft silhouette score, a generalization of the widely used silhouette measure that accommodates probabilistic cluster assignments. Leveraging this differentiable measure, we developed the DCSS methodology and an autoencoder-based framework specifically designed to optimize the soft silhouette objective. Notably, the DCSS method guides the learned latent representations to form both compact and well-separated clusters. This property is crucial in real-world applications, as targeting both compactness and separability ensures that the resulting clusters are not only densely packed, but also distinct from each other. Experiments

on a variety of benchmark datasets show that DCSS performs competitively against established deep clustering approaches, underscoring the effectiveness soft silhouette as an objective for enhancing the quality of learned latent representations.

In Chapter 6, we presented the NIMLC clustering method, a neural-network-based approach that frames clustering as a generative task within the implicit maximum likelihood estimation framework. By adapting ideas from ClusterGAN, NIMLC avoids some of the known deficiencies of GAN-based clustering while retaining a simple, stable training objective. The method performs particularly well on small datasets, and experimental comparisons with both deep and conventional clustering algorithms underscore its competitive potential. A notable strength of NIMLC is its ability to accommodate diverse cluster geometries without tuning hyperparameters. Experiments on synthetic datasets demonstrate that the same settings allow NIMLC to effectively cluster both cloud-shaped and ring-shaped data. Taken together, these results suggest that the incorporation of generative modeling into IMLE offers a promising avenue for robust and versatile clustering in neural network-based systems.

7.2 Directions for Future Work

Finally, we outline several potential directions for future work, addressing open issues related to this thesis that merit further investigation.

For the global k-means++ method presented in Chapter 2 it would be interesting to investigate a dynamic method for appropriately selecting the number of candidates L in each k cluster sub-problem. This could potentially lead to a further speed-up of the algorithm. We could also adapt the global k-means++ algorithm into a semi-supervised setting by incorporating must-link or cannot-link constraints by utilizing the exact solver as referred to [190]. Additionally, we aim to test the method in real applications (such as in biology, speech recognition, face clustering, etc.) initially requiring a large number of clusters (overclustering solutions). Finally, it would be interesting to integrate the method into a deep framework for clustering of composite objects (graphs, images, text, etc.).

For the global kernel k-means++ method presented in Chapter 3 it would be interesting to investigate a technique for adapting the number of candidates L as k increases. We observed that fewer candidates are required to be examined for small

values of k since the problem is simpler. As k increases, the problem becomes more difficult, necessitating more extensive exploration using more candidates. We believe that dynamically tuning L in each k-cluster subproblem could further improve both the attained clustering error and the execution time of the method. In addition, we plan to investigate the estimation of the number of clusters in this algorithmic framework using criteria such as silhouette coefficient [92], modularity [109], and inclusion [110, 111]. Additionally, it can be used for image segmentation [191], where grouping pixels or regions with similar features is crucial, and bioinformatics [192] where analyzing gene expression patterns requires accurate non-linear clustering. In a graph partitioning framework, the method can be applied to social network analysis to detect communities in large-scale networks and to recommendation systems for clustering users based on shared preferences. Finally, we aim to extend this method to other kernel k-means variations such as fuzzy kernel k-means [193, 194] and use the method in real-world non-linear clustering applications.

A separate line of research addressed by the UniForCE method introduced in Chapter 4. Future work could explore its application across diverse domains to further evaluate its versatility. Additionally, investigating the potential advantages of integrating UniForCE within existing machine learning pipelines could reveal valuable comparative benefits. A promising future direction is the incorporation of the mudpod [195] into the UniForCE procedure. Specifically, mudpod is a recent criterion that leverages the dip statistic across random projections to assess multimodality. It will beneficial to be applied during the initial overclustering stage to validate that each subcluster is unimodal, an assumption on which UniForCE relies. We believe that the concept of local unimodality, i.e. the fact that unimodality can be tested and validated locally, could offer a foundation for future advancements in other unsupervised learning tasks, such as density estimation, dimensionality reduction, and data visualization. Finally, the development of internal cluster validity indices for the assessment of irregular-shaped clusters would help in understanding further the behavior of clustering methodologies.

Regarding the DCSS method introduced in Chapter 5, there are several directions for future work, such as improving the clustering results using data augmentation techniques since it is adopted as an effective strategy for enhancing the learned representations [175, 196]. In addition, more sophisticated models and training methodologies can be used, such as ensemble models [197, 198] or adversarial

learning [199, 200]. It is also possible to modify the learning procedure to incorporate self-paced learning [183], since learning the "easier" data first is expected to improve the clustering results [201, 202, 203]. However, our major focus will be on extending the DCCS algorithm for estimating the number of clusters by exploiting unimodality tests as happens in the dip-means [204] and DIPDECK [134] algorithms.

Finally, for the NIMLC method introduced in Chapter 6, future research could focus on a more detailed experimental investigation of its performance and its use in real-world applications. Additionally, we aim to consider a modification of the method in which the number of samples could vary at each epoch. In the same spirit, we could explore the possibility of adopting a self-paced approach similar to the one proposed in [183] for the online tuning of the parameter $\Delta\sigma$. Furthermore, it is interesting to study how the NIMLC approach could be integrated into a general methodology for estimating the true number of clusters in the dataset.

Building on the method-specific research directions discussed above, we can also identify broader avenues for future work in data clustering. One key limitation of current deep clustering methods is their lack of attention to the dimensionality of the latent space. A high-dimensional latent space can capture richer and more complex information, but it also risks encoding irrelevant details or noise. Conversely, a low-dimensional latent space produces more compact and manageable embeddings, but may fail to preserve important information. An interesting direction for future research is the development of methodologies that can automatically determine and adapt the appropriate latent dimensionality within deep clustering frameworks.

Relatively little research has addressed the problem of determining the number of clusters within deep clustering frameworks. As demonstrated in this thesis, this remains a crucial and challenging issue in data clustering. A promising direction for future work would be to integrate unimodality testing with deep learning approaches. One potential path is to build on the methodological foundations of UniForCE and embed them into a deep clustering framework, enabling automated cluster estimation. Such an approach could lead to the development of a Deep UniForCE method.

An exciting avenue for future research is the integration of large language models (LLMs) into text clustering tasks. LLMs provide rich contextual embeddings that capture deep semantic and syntactic information, far surpassing traditional word or sentence representations. Incorporating these embeddings into clustering frameworks could significantly improve the ability to group documents, sentences, or even multi-

modal content by meaning rather than surface similarity.

Another promising research direction lies in exploring practical applications of the proposed approaches across diverse domains. For instance, in computer vision, these methods could be applied to video summarization, enabling the automatic extraction of representative frames or segments that capture the essence of long video streams. Such capabilities would be valuable for content indexing, retrieval, and efficient video browsing. In the health domain, an especially compelling application is the analysis of genomic data. Applying clustering techniques to genome sequences could help uncover previously unrecognized genomic patterns or subgroups, potentially leading to the discovery of novel biomarkers, patient stratification strategies, or even insights into rare genetic conditions. These findings could support clinicians in identifying hidden structures in biological data that might otherwise remain unnoticed, thereby advancing personalized medicine and improving diagnostic procedure. Beyond these examples, the adaptability of the approaches suggests broader potential in fields such as social network analysis, natural language processing, and recommendation systems.

Bibliography

[1] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "Clustergan: Latent space clustering in generative adversarial networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4610–4617.

[2] C. J. Veenman, M. J. T. Reinders, and E. Backer, "A maximum variance cluster algorithm," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 9, pp. 1273–1280, 2002.

[3] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[5] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

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

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[8] S. J. Prince, *Understanding deep learning*. MIT press, 2023.

[9] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[10] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern recognition*, vol. 41, no. 1, pp. 176–190, 2008.

[11] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[12] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv preprint arXiv:1801.07648*, 2018.

[16] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

[17] G. Nutakki, B. Abdollahi, W. Sun, and O. Nasraoui, *An Introduction to Deep Clustering*, 01 2019, pp. 73–89.

[18] S. Zhou, H. Xu, Z. Zheng, J. Chen, J. Bu, J. Wu, X. Wang, W. Zhu, M. Ester *et al.*, "A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions," *arXiv preprint arXiv:2206.07579*, 2022.

[19] Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu, and L. He, "Deep clustering: A comprehensive survey," *IEEE transactions on neural networks and learning systems*, vol. 36, no. 4, pp. 5858–5878, 2024.

[20] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[21] V. Cohen-Addad and C. Karthik, "Inapproximability of clustering in lp metrics," in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2019, pp. 519–539.

[22] V. Cohen-Addad, C. Karthik, and E. Lee, "On approximability of clustering problems without candidate centers," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2021, pp. 2635–2648.

[23] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine learning*, vol. 75, no. 2, pp. 245–248, 2009.

[24] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar k-means problem is np-hard," *Theoretical Computer Science*, vol. 442, pp. 13–21, 2012.

[25] J. MacQueen *et al*., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[26] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[27] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the k-means clustering algorithm," *Expert systems with applications*, vol. 40, no. 1, pp. 200–210, 2013.

[28] C. Baldassi, "Recombinator-k-means: an evolutionary algorithm that exploits k-means++ for recombination," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 991–1003, 2022.

[29] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford InfoLab, Technical Report 2006-13, June 2006. [Online]. Available: http://ilpubs.stanford.edu:8090/778/

[30] A. Agrawal and H. Gupta, "Global k-means (gkm) clustering algorithm: a survey," *International journal of computer applications*, vol. 79, no. 2, 2013.

[31] P. Fränti and S. Sieranoja, "How much can k-means be improved by using better initialization and repeats?" *Pattern Recognition*, vol. 93, pp. 95–112, 2019.

[32] J. Ajmera and C. Wooters, "A robust speaker clustering algorithm," in *2003 IEEE Workshop on Automatic Speech Recognition and Understanding (IEEE Cat. No.03EX721)*, 2003, pp. 411–416.

[33] Y. Saeys, S. Van Gassen, and B. N. Lambrecht, "Computational flow cytometry: helping to make sense of high-dimensional immunology data," *Nature Reviews Immunology*, vol. 16, no. 7, pp. 449–462, 2016.

[34] Z. Wei and Y.-C. Chen, "Skeleton clustering: Graph-based approach for dimension-free density-aided clustering," in *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*.

[35] F. Nie, C.-L. Wang, and X. Li, "K-multiple-means: A multiple-means clustering method with specified k clusters," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 959–967.

[36] J. Xie, S. Jiang, W. Xie, and X. Gao, "An efficient global k-means clustering algorithm." *J. Comput.*, vol. 6, no. 2, pp. 271–279, 2011.

[37] A. M. Bagirov, J. Ugon, and D. Webb, "Fast modified global k-means algorithm for incremental cluster construction," *Pattern recognition*, vol. 44, no. 4, pp. 866–876, 2011.

[38] L. Bai, J. Liang, C. Sui, and C. Dang, "Fast global k-means clustering based on local geometrical information," *Information Sciences*, vol. 245, pp. 168–180, 2013.

[39] J. Z. Lai and T.-J. Huang, "Fast global k-means clustering using cluster membership and inequality," *Pattern Recognition*, vol. 43, no. 5, pp. 1954–1963, 2010.

[40] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[41] K.-R. Müller, S. Mika, K. Tsuda, and K. Schölkopf, "An introduction to kernel-based learning algorithms," in *Handbook of neural network signal processing*. Boca Raton, FL: CRC Press, 2018, pp. 4–1.

[42] I. S. Dhillon, Y. Guan, and B. Kulis, *A unified view of kernel k-means, spectral clustering and graph cuts*. E103 Westgate Building 288 N. Burrowes Rd. University Park, PA 16802: Citeseer, 2004.

[43] E. W. Forgy, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications," *biometrics*, vol. 21, pp. 768–769, 1965.

[44] G. Vardakas, I. Papakostas, and A. Likas, "Efficient error minimization in kernel k-means clustering," *Pattern Analysis and Applications*, vol. 28, no. 2, p. 107, 2025.

[45] G. F. Tzortzis and A. C. Likas, "The global kernel k-means algorithm for clustering in feature space," *IEEE transactions on neural networks*, vol. 20, no. 7, pp. 1181–1194, 2009.

[46] C. Loader, *Local regression and likelihood*. Springer Science & Business Media, 2006.

[47] S. Dharmadhikari and K. Joag-Dev, *Unimodality, Convexity, and Applications*. Elsevier Science, 1988.

[48] T. Dai, "On multivariate unimodal distributions (Master Thesis)," Ph.D. dissertation, University of British Columbia, 1989.

[49] B. W. Silverman, "Using kernel density estimates to investigate multimodality," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 43, no. 1, pp. 97–99, 1981.

[50] J. A. Hartigan and P. M. Hartigan, "The Dip Test of Unimodality," *The Annals of Statistics*, vol. 13, no. 1, pp. 70 – 84, 1985.

[51] M. C. Minnotte, *A test of mode existence with applications to multimodality*. Rice University, 1993.

[52] N. Fischer, E. Mammen, and J. S. Marron, "Testing for multimodality," *Computational statistics & data analysis*, vol. 18, no. 5, pp. 499–512, 1994.

[53] P. Chasani, "Machine learning methods based on unimodality testing," 2025.

[54] A. Adolfsson, M. Ackerman, and N. C. Brownstein, "To cluster, or not to cluster: An analysis of clusterability methods," *Pattern Recognition*, vol. 88, pp. 13–26, 2019.

[55] A. Kalogeratos and A. Likas, "Dip-means: an incremental clustering method for estimating the number of clusters," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[56] T. Chamalis and A. Likas, "The projected dip-means clustering algorithm," in *Hellenic Conf. on Artificial Intelligence*, 2018.

[57] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Icml*, 2011.

[58] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[59] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[60] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[61] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[62] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[63] N. G. Pavlidis, D. P. Hofmeyr, and S. K. Tasoulis, "Minimum density hyperplanes," *Journal of Machine Learning Research*, 2016.

[64] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[65] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[66] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: An unsupervised and generative approach to clustering," *arXiv preprint arXiv:1611.05148*, 2016.

[67] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5147–5156.

[68] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

[69] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[70] S. Saito and R. T. Tan, "Neural clustering: Concatenating layers for better projections," 2017.

[71] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*. PMLR, 2017, pp. 3861–3870.

[72] S. Theodoridis and K. Koutroumbas, "Chapter 11 - clustering: Basic concepts," in *Pattern Recognition (Fourth Edition)*, fourth edition ed., S. Theodoridis and K. Koutroumbas, Eds. Boston: Academic Press, 2009, pp. 595–625. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978159749272050013X

[73] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*. PMLR, 2016, pp. 478–487.

[74] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 407–416.

[75] C.-C. Hsu and C.-W. Lin, "Cnn-based joint clustering and representation learning with feature drift compensation for large-scale image data," *IEEE Transactions on Multimedia*, vol. 20, no. 2, pp. 421–429, 2017.

[76] G. Chen, "Deep learning with nonparametric clustering," *arXiv preprint arXiv:1501.03084*, 2015.

[77] P. Huang, Y. Huang, W. Wang, and L. Wang, "Deep embedding network for clustering," in *2014 22nd International conference on pattern recognition*. IEEE, 2014, pp. 1532–1537.

[78] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[79] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation." in *Ijcai*, 2017, pp. 1753–1759.

[80] C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, "Auto-encoder based data clustering," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 18th Iberoamerican Congress, CIARP 2013, Havana, Cuba, November 20-23, 2013, Proceedings, Part I 18*. Springer, 2013, pp. 117–124.

[81] C. Leiber, L. G. Bauer, M. Neumayr, C. Plant, and C. Böhm, "The dipencoder: Enforcing multimodality in autoencoders," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 846–856.

[82] J. A. Hartigan and P. M. Hartigan, "The dip test of unimodality," *The annals of Statistics*, pp. 70–84, 1985.

[83] A. Jalal, A. Ilyas, C. Daskalakis, and A. G. Dimakis, "The robust manifold defense: Adversarial training using generative models," *arXiv preprint arXiv:1712.09196*, 2017.

[84] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning*. PMLR, 2017, pp. 537–546.

[85] S. Santurkar, D. Budden, and N. Shavit, "Generative compression," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 258–262.

[86] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[87] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[88] G. Vardakas and A. Likas, "Global k-means++: an effective relaxation of the global k-means clustering algorithm," *Applied Intelligence*, vol. 54, no. 19, pp. 8876–8888, 2024.

[89] G. Vardakas, A. Kalogeratos, and A. Likas, "Uniforce: The unimodality forest method for clustering and estimation of the number of clusters," *Pattern Recognition*, p. 112357, 2025.

[90] G. Vardakas, I. Papakostas, and A. Likas, "Deep clustering using the soft silhouette score: Towards compact and well-separated clusters," *arXiv preprint arXiv:2402.00608*, 2024.

[91] G. Vardakas and A. Likas, "Neural clustering based on implicit maximum likelihood," *Neural Computing and Applications*, pp. 1–14, 2023.

[92] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[93] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern recognition*, vol. 46, no. 1, pp. 243–256, 2013.

[94] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[95] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[96] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[97] G. W. Milligan and M. C. Cooper, "A study of standardization of variables in cluster analysis," *Journal of classification*, vol. 5, no. 2, pp. 181–204, 1988.

[98] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023.

[99] O. Bachem, M. Lucic, H. Hassani, and A. Krause, "Fast and provably good seedings for k-means," *Advances in neural information processing systems*, vol. 29, 2016.

[100] D. Choo, C. Grunau, J. Portmann, and V. Rozhon, "k-means++: few more steps yield constant approximation," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1909–1917.

[101] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 551–556.

[102] D. Paul, S. Chakraborty, S. Das, and J. Xu, "Implicit annealing in kernel spaces: A strongly consistent clustering approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5862–5871, 2022.

[103] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 14, 2001.

[104] H. Jia, S. Ding, X. Xu, and R. Nie, "The latest research progress on spectral clustering," *Neural Computing and Applications*, vol. 24, pp. 1477–1486, 2014.

[105] J.-S. Wu, W.-S. Zheng, J.-H. Lai, and C. Y. Suen, "Euler clustering on large-scale dataset," *IEEE Transactions on Big Data*, vol. 4, no. 4, pp. 502–515, 2017.

[106] G. França, M. L. Rizzo, and J. T. Vogelstein, "Kernel k-groups via hartigan's method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4411–4425, 2020.

[107] N. Tsapanos, A. Tefas, N. Nikolaidis, and I. Pitas, "A distributed framework for trimmed kernel k-means clustering," *Pattern recognition*, vol. 48, no. 8, pp. 2685–2698, 2015.

[108] S. H.-C. Jiang, R. Krauthgamer, J. Lou, and Y. Zhang, "Coresets for kernel clustering," *Machine Learning*, pp. 1–16, 2024.

[109] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[110] N. Koufos and A. Likas, "The inclusion measure for community evaluation and detection in unweighted networks," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 1053–1056.

[111] N. Kornelakis and A. Likas, "The inclusion criterion for data clustering quality," in *Proceedings of the 13th Hellenic Conference on Artificial Intelligence*, 2024, pp. 1–4.

[112] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[113] H. Zha, X. He, C. Ding, M. Gu, and H. Simon, "Spectral relaxation for k-means clustering," *Advances in neural information processing systems*, vol. 14, 2001.

[114] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[115] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE, 1997, pp. 731–737.

[116] Shi, "Multiclass spectral clustering," in *Proceedings ninth IEEE international conference on computer vision*. IEEE, 2003, pp. 313–319.

[117] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 1994 IEEE workshop on applications of computer vision*. IEEE, 1994, pp. 138–142.

[118] G. W. Milligan and M. Cooper, "A study of standardization of variables in cluster analysis," *Journal of Classification*, vol. 5, pp. 181–204, 1988. [Online]. Available: https://api.semanticscholar.org/CorpusID:122116077

[119] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[120] S. Kitayama and K. Yamazaki, "Simple estimate of the width in gaussian kernel with adaptive scaling technique," *Applied Soft Computing*, vol. 11, no. 8, pp. 4726–4737, 2011.

[121] H.-H. Bock, *Clustering Methods: A History of k-Means Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 161–172.

[122] E. Schubert and P. J. Rousseeuw, "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms," *Information Systems*, vol. 101, p. 101804, 2021.

[123] A. Kalogeratos and A. Likas, "Document clustering using synthetic cluster prototypes," *Data & Knowledge Engineering*, vol. 70, no. 3, pp. 284–306, 2011.

[124] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[125] J. Jang and H. Jiang, "DBSCAN++: Towards fast and scalable density clustering," in *Int'l Conf. on Machine Learning*, 2019, pp. 3019–3029.

[126] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[127] D. Pelleg and A. W. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Int'l Conf. on Machine Learning*, 2000, p. 727–734.

[128] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16, 2003.

[129] Y. Feng and G. Hamerly, "PG-means: learning the number of clusters in data," in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19, 2006.

[130] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Trans. on Knowledge Discovery from Data*, vol. 10, no. 1, pp. 1–51, 2015.

[131] D. Li, S. Zhou, T. Zeng, and R. H. Chan, "Multi-prototypes convex merging based k-means clustering algorithm," *IEEE Trans. on Knowledge and Data Engineering*, 2023.

[132] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs, "NbClust: An R package for determining the relevant number of clusters in a data set," *Journal of Statistical Software*, vol. 61, no. 6, p. 1–36, 2014.

[133] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, 2007.

[134] C. Leiber, L. G. Bauer, B. Schelling, C. Böhm, and C. Plant, "Dip-based deep embedded clustering with k-estimation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 903–913.

[135] S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos, "Enhancing principal direction divisive clustering," *Pattern Recognition*, vol. 43, no. 10, pp. 3391–3411, 2010.

[136] S. Tasoulis, N. G. Pavlidis, and T. Roos, "Nonlinear dimensionality reduction for clustering," *Pattern Recognition*, vol. 107, p. 107508, 2020.

[137] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis, "Diffusion maps, spectral clustering and eigenfunctions of fokker-planck operators," in *Neural Information Processing Systems*, 2005, p. 955–962.

[138] A. Little, M. Maggioni, and J. M. Murphy, "Path-based spectral clustering: Guarantees, robustness to outliers, and fast algorithms," *Journal of Machine Learning Research*, vol. 21, no. 6, pp. 1–66, 2020.

[139] A. Mukhopadhyay, U. Maulik, and S. Bandyopadhyay, "A survey of multiobjective evolutionary clustering," *ACM Computing Surveys*, vol. 47, no. 4, 2015.

[140] S. Zhu, L. Xu, and E. D. Goodman, "Hierarchical topology-based cluster representation for scalable evolutionary multiobjective clustering," *IEEE Trans. on Cybernetics*, vol. 52, no. 9, pp. 9846–9860, 2022.

[141] F. Nie, C.-L. Wang, and X. Li, "k-multiple-means: A multiple-means clustering method with specified k clusters," in *ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining*, 2019, pp. 959–967.

[142] R. Mehmood, S. El-Ashram, R. Bie, H. Dawood, and A. Kos, "Clustering by fast search and merge of local density peaks for gene expression microarray data," *Scientific reports*, vol. 7, no. 1, p. 45602, 2017.

[143] J. Guan, S. Li, X. He, J. Zhu, J. Chen, and P. Si, "SMMP: A stable-membership-based auto-tuning multi-peak clustering algorithm," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 6307–6319, 2022.

[144] C. Li, S. Ding, X. Xu, H. Hou, and L. Ding, "Fast density peaks clustering algorithm based on improved mutual k-nearest-neighbor and sub-cluster merging," *Information Sciences*, vol. 647, p. 119470, 2023.

[145] D. Cheng, Q. Zhu, J. Huang, Q. Wu, and L. Yang, "Clustering with local density peaks-based minimum spanning tree," *IEEE Trans. on Knowledge and Data Engineering*, vol. 33, no. 2, pp. 374–387, 2021.

[146] Z. Wei and Y.-C. Chen, "Skeleton clustering: Graph-based approach for dimension-free density-aided clustering," in *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

[147] A. Peterson, A. Ghosh, and R. Maitra, "Merging k-means with hierarchical clustering for identifying general-shaped groups," *Stat*, vol. 7, 12 2017.

[148] Y. Zhao, G. Karypis, and U. Fayyad, "Hierarchical clustering algorithms for document datasets," *Data Mining and Knowledge Kiscovery*, vol. 10, pp. 141–168, 2005.

[149] A. Ioannidis, V. Chasanis, and A. Likas, "An agglomerative approach for shot summarization based on content homogeneity," in *Int'l Conf. on Machine Vision*, vol. 9445, 2015, p. 94451F.

[150] T. Chamalis and A. Likas, "Region merging for image segmentation based on unimodality tests," in *Int'l Conf. on Control, Automation and Robotics*, 2017, pp. 381–384.

[151] J. Ameijeiras-Alonso, R. M. Crujeiras, and A. Rodríguez-Casal, "Mode testing, critical bandwidth and excess mass," *TEST*, vol. 28, no. 3, pp. 900–919, 2019.

[152] J. Kleinberg and E. Tardos, *Algorithm Design (Sec. 4.6 & 4.7)*. Addison Wesley, 2006.

[153] L. G. Bauer, C. Leiber, C. Böhm, and C. Plant, "Extension of the dip-test repertoire-efficient and differentiable p-value calculation for clustering," in *SIAM International Conference on Data Mining*, 2023, pp. 109–117.

[154] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *Int'l Joint Conf. on Neural Networks*. IEEE, 2017, pp. 2921–2926.

[155] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 529–534.

[156] W. Guo, K. Lin, and W. Ye, "Deep embedded k-means clustering," in *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2021, pp. 686–694.

[157] S. A. Shah and V. Koltun, "Robust continuous clustering," *Proceedings of the National Academy of Sciences*, vol. 114, no. 37, pp. 9814–9819, 2017.

[158] D. Comaniciu and P. Meer, "Mean Shift: A robust approach toward feature space analysis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[159] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, vol. 11, no. 95, pp. 2837–2854, 2010.

[160] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[161] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz, "Internal versus external cluster validation indexes," *International Journal of computers and communications*, vol. 5, no. 1, pp. 27–34, 2011.

[162] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on neural networks*, vol. 20, no. 2, pp. 189–201, 2009.

[163] J. E. Chacón and A. I. Rastrojo, "Minimum adjusted rand index for two clusterings of a given size," *Advances in Data Analysis and Classification*, pp. 1–9, 2022.

[164] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," 1973.

[165] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[166] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[167] J. Pavlopoulos, G. Vardakas, and A. Likas, "Revisiting silhouette aggregation," in *International Conference on Discovery Science*. Springer, 2024, pp. 354–368.

[168] M. D. Buhmann, "Radial basis functions," *Acta numerica*, vol. 9, pp. 1–38, 2000.

[169] L. K. Saul and S. T. Roweis, "Think globally, fit locally: unsupervised learning of low dimensional manifolds," *Journal of machine learning research*, vol. 4, no. Jun, pp. 119–155, 2003.

[170] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[171] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[172] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *arXiv preprint arXiv:1812.01718*, 2018.

[173] E. Bulbul, A. Cetin, and I. A. Dogru, "Human activity recognition using smartphones," in *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2018, pp. 1–6.

[174] L. Van Der Maaten, "Learning a parametric embedding by preserving local structure," in *Artificial intelligence and statistics*. PMLR, 2009, pp. 384–391.

[175] X. Guo, E. Zhu, X. Liu, and J. Yin, "Deep embedded clustering with data augmentation," in *Asian conference on machine learning*. PMLR, 2018, pp. 550–565.

[176] Y. Ren, N. Wang, M. Li, and Z. Xu, "Deep density-based image clustering," *Knowledge-Based Systems*, vol. 197, p. 105841, 2020.

[177] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[178] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[179] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[180] K. Li and J. Malik, "Implicit maximum likelihood estimation," *arXiv preprint arXiv:1809.09087*, 2018.

[181] S. Mohamed and B. Lakshminarayanan, "Learning in implicit generative models," *arXiv preprint arXiv:1610.03483*, 2016.

[182] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[183] M. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," *Advances in neural information processing systems*, vol. 23, 2010.

[184] G. Vardakas and A. Likas, "Implicit maximum likelihood clustering," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2022, pp. 484–495.

[185] S. Tasoulis, N. G. Pavlidis, and T. Roos, "Nonlinear dimensionality reduction for clustering," *Pattern Recognition*, vol. 107, p. 107508, 2020.

[186] G. X. Zheng, J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, T. D. Wheeler, G. P. McDermott, J. Zhu *et al.*, "Massively parallel digital

transcriptional profiling of single cells," *Nature communications*, vol. 8, no. 1, pp. 1–12, 2017.

[187] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[188] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

[189] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 52, no. 1, pp. 7–21, 2005.

[190] V. Piccialli, A. R. Russo, and A. M. Sudoso, "An exact algorithm for semi-supervised minimum sum-of-squares clustering," *Computers & Operations Research*, vol. 147, p. 105958, 2022.

[191] Z. Khan and J. Yang, "Nonparametric k-means clustering-based adaptive unsupervised colour image segmentation," *Pattern Analysis and Applications*, vol. 27, no. 1, p. 17, 2024.

[192] R. Jothi, S. K. Mohanty, and A. Ojha, "Dk-means: a deterministic k-means clustering algorithm for gene expression analysis," *Pattern Analysis and Applications*, vol. 22, pp. 649–667, 2019.

[193] P. Das and A. Das, "A fast and automated segmentation method for detection of masses using folded kernel based fuzzy c-means clustering algorithm," *Applied Soft Computing*, vol. 85, p. 105775, 2019.

[194] D. Graves and W. Pedrycz, "Kernel-based fuzzy clustering and fuzzy clustering: A comparative experimental study," *Fuzzy sets and systems*, vol. 161, no. 4, pp. 522–543, 2010.

[195] P. Kolyvakis and A. Likas, "A multivariate unimodality test harnessing the dip statistic of mahalanobis distances over random projections," *arXiv preprint arXiv:2311.16614*, 2023.

[196] X. Deng, D. Huang, D.-H. Chen, C.-D. Wang, and J.-H. Lai, "Strongly augmented contrastive clustering," *Pattern Recognition*, vol. 139, p. 109470, 2023.

[197] S. Affeldt, L. Labiod, and M. Nadif, "Spectral clustering via ensemble deep autoencoder learning (sc-edae)," *Pattern Recognition*, vol. 108, p. 107522, 2020.

[198] Z. Hao, Z. Lu, G. Li, F. Nie, R. Wang, and X. Li, "Ensemble clustering with attentional representation," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[199] X. Yang, C. Deng, K. Wei, J. Yan, and W. Liu, "Adversarial learning for robust deep clustering," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9098–9108, 2020.

[200] N. Mrabah, M. Bouguessa, and R. Ksantini, "Adversarial deep embedded clustering: on a better trade-off between feature randomness and feature drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 4, pp. 1603–1617, 2020.

[201] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognition*, vol. 83, pp. 161–173, 2018.

[202] X. Guo, X. Liu, E. Zhu, X. Zhu, M. Li, X. Xu, and J. Yin, "Adaptive self-paced deep clustering with data augmentation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 9, pp. 1680–1693, 2019.

[203] K. Zhang, C. Song, and L. Qiu, "Self-paced deep clustering with learning loss," *Pattern Recognition Letters*, vol. 171, pp. 8–14, 2023.

[204] A. Kalogeratos and A. Likas, "Dip-means: an incremental clustering method for estimating the number of clusters," *Advances in neural information processing systems*, vol. 25, 2012.

Author's Publications

Journal Publications

J1. **G. Vardakas**, A. Kalogeratos, and A. Likas, "UniForCE: The Unimodality Forest method for Clustering and Estimation of the number of clusters," *Pattern Recognition*, p. 112357, 2025.

J2. **G. Vardakas**, I. Papakostas, and A. Likas, "Efficient error minimization in kernel k-means clustering," *Pattern Analysis and Applications*, vol. 28, no. 2, p. 107, 2025.

J3. G. Papigkiotis, **G. Vardakas**, A. Likas, and N. Stergioulas, "Universal description of a neutron star's surface and its key global properties: A machine learning approach for nonrotating and rapidly rotating stellar models," *Physical Review D*, vol. 111, no. 8, p. 083056, 2025.

J4. **G. Vardakas** and A. Likas, "Global k-means++: an effective relaxation of the global k-means clustering algorithm," *Applied Intelligence*, vol. 54, no. 19, pp. 8876–8888, 2024.

J5. J. Pavlopoulos, M. Konstantinidou, E. Perdiki, I. Marthot-Santaniello, H. Essler, **G. Vardakas**, and A. Likas, "Explainable dating of greek papyri images," *Machine Learning*, vol. 113, no. 9, pp. 6765–6786, 2024.

J6. **G. Vardakas** and A. Likas, "Neural clustering based on implicit maximum likelihood," *Neural computing and applications*, vol. 35, no. 29, pp. 21511–21524, 2023.

Conference Publications

C1. **G. Vardakas**, A. Karra, E. Pitoura, and A. Likas, "Counterfactual Explanations for k-means and Gaussian Clustering," *in IEEE 37th International Conference on*

Tools with Artificial Intelligence (ICTAI), 2025.

C2. **G. Vardakas**, A. Karra, E. Pitoura, and A. Likas, "Evaluating Clustering Quality in Centroid-Based Clustering Using Counterfactual Distances," *in International Conference AI for SCIENCE*, 2025.

C3. **G. Vardakas**, A. Karra, E. Pitoura, and A. Likas, "Linkage criteria for agglomerative clustering based on counterfactual distances." in *The Sixteenth International Conference on Information, Intelligence, Systems and Applications*, IEEE Press, 2025.

C4. A. Karra, **G. Vardakas**, E. Pitoura, and A. Likas, "Generating Counterfactual Explanations for Clustering Models Based on Their Equivalence to Classification Models," *in IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 85–100, 2025.

C5. J. Pavlopoulos, **G. Vardakas**, and A. Likas, "Revisiting silhouette aggregation," *in International Conference on Discovery Science*, pp. 354–368, 2024.

C6. J. Pavlopoulos, M. Konstantinidou, **G. Vardakas**, I. Marthot-Santaniello, E. Perdiki, D. Koutsianos, A. Likas, and H. Essler, "Explaining the chronological attribution of Greek papyri images," *in International Conference on Discovery Science*, pp. 401–415, 2023.

C7. **G. Vardakas** and A. Likas, "Implicit maximum likelihood clustering," *Proc. IFIP Int. Conf. Artificial Intelligence Applications and Innovations (AIAI)*, pp. 484–495, 2022.

Preprints

1. **G. Vardakas**, I. Papakostas, and A. Likas, "Deep clustering using the soft silhouette score: Towards compact and well-separated clusters," *arXiv preprint arXiv:2402.00608*, 2024.

Short Biography

Georgios Vardakas was born in Ioannina, Greece, in 1995. He received his Diploma in Computer Science and Engineering, as well as his M.Sc. in Data Science and Engineering, from the Department of Computer Science and Engineering at the University of Ioannina in 2020 and 2021, respectively. Since 2021, he has been a Ph.D. candidate in the same department. He has participated as a researcher in several projects funded by the National Strategic Reference Framework (NSRF) and the Hellenic Foundation for Research and Innovation (HFRI). He also received a scholarship through the NSRF 2014–2020 under the action Support of the Regional Excellence. In 2022, he completed a research internship at Archimedes Unit – Center for Research in Artificial Intelligence, Data Science and Algorithms. His research interests include machine learning, and neural networks, with a special focus on unsupervised learning, representation learning and clustering.