Clustering in Transformed Feature Spaces by Analyzing Distinct Modes

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von

Collin Leiber

München, den 11.01.2024

Erstgutachter:	Prof. Dr. Christian Böhm
Zweitgutachter:	Prof. Aristidis Likas, Ph.D.
Drittgutachterin:	Prof. Dr. Sibylle Hess
Vorsitz:	Prof. Dr. Albrecht Schmidt
Tag der Disputation:	16.04.2024

Eidesstattliche Versicherung

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

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

München, den 11.01.2024

Collin Leiber

Abstract

The ability to extract valuable information from data is becoming increasingly relevant due to the growing number of available data sets. Since annotating data is very costly, methods that provide this information without any annotations are of particular interest. However, so-called clustering methods, which identify groups of objects with similar characteristics, often run into problems when working with extensive data sets or data sets with numerous features. Therefore, we require new methods that return high-quality results even when working with large, high-dimensional data.

A common practice is to accompany the clustering process with a dimensionality reduction, leading to a simplified data set consisting only of those features that contain valuable information for the cluster analysis. If this dimensionality reduction occurs through a linear transformation, it is usually referred to as subspace clustering. Here, intermediate clustering results can significantly influence the resulting features. This ability is in contrast to a simple, preceding dimensionality reduction. When discussing subspace clustering, one must distinguish between traditional methods that define a separate subspace for each cluster and those that create a common feature space for all clusters. In this dissertation, we only deal with common subspace approaches, as they preserve the comparability between the clusters. One can go one step further and create multiple common subspaces for multiple clustering results, which is the case with so-called non-redundant clustering methods. Applying those approaches is often challenging because suitable parameters for the dimensionalities of the subspaces and the number of clusterings and clusters per clustering have to be defined. We tackle this problem by using the current modes in each subspace to encode the data and apply the Minimum Description Length principle. Here, the model with the minimum encoding cost is assumed to be the best solution.

Existing modes are further analyzed by utilizing statistical modality tests, such as the Dip-test of unimodality. The Dip-test outputs a Dip-value that describes the deviation of an empirical cumulative distribution function to an arbitrary unimodal distribution. Using a novel transformation function, we can further state how likely a unimodal or multimodal distribution is to be present, which helps to interpret the Dip-value better. Due to the differentiability of this function, it is used to identify projection axes that show a high degree of multimodality within the clusters. Combining these axes, one obtains a lower-dimensional feature space containing the main modes within the data set, where

each mode defines a separate cluster.

For complex data sets, there is a need for more powerful processes than linear transformations to obtain suitable feature spaces for clustering. Non-linear transformations using neural networks are often applied in these cases. The combination of clustering and neural networks is known as deep clustering. Such methods directly integrate the clustering approach into the optimization function of, e.g., an autoencoder. Once again, the problem arises of selecting a suitable value for the number of clusters. A solution is to overestimate the initial number of clusters deliberately and, based on the current modalities obtained by the Dip-test, decide whether two clusters can be merged. The embedding then adapts to the existing clustering structure. As a result, the number of clusters does not have to be known in advance but is learned by the algorithm. Further analyses show that the Dip-test can be directly used to optimize an autoencoder. For this purpose, we utilize the gradient of the Dip-test to train the parameters of an autoencoder and identify high-quality clustering results. Thus, no specific distribution function for the embedded data must be assumed apriori. The only assumption is that each cluster adopts a unimodal and each pair of clusters a multimodal structure, which is a relaxation compared to established methods.

The described approaches fulfill their goal of identifying informative structures in large, high-dimensional data sets without requiring any annotations. Compared to competitor methods, they achieve state-of-the-art results, while the parameterization is greatly simplified. All methods presented in this thesis are provided in our open-source package ClustPy.

Zusammenfassung

Die Fähigkeit wertvolle Informationen aus Daten zu extrahieren, wird aufgrund der wachsenden Anzahl an verfügbaren Datensätzen immer wichtiger. Da die Annotation von Daten sehr kostspielig ist, sind insbesondere Methoden interessant, die diese Informationen ohne vordefinierte Annotationen ausgeben. Sogenannten Clustering-Methoden, die automatisiert Gruppen von Objekten mit ähnlichen Charakteristiken identifizieren, stoßen jedoch häufig beim Arbeiten mit sehr großen Datensätzen oder Datensätzen mit einer Vielzahl von Merkmalen auf Probleme. Es werden daher neue Methoden benötigt, die auch bei großen, hochdimensionalen Daten qualitativ hochwertige Ergebnisse liefern.

Eine gängige Praxis ist es, den Clustering-Prozess mit einer Dimensionsreduktion zu kombinieren. Dies führt zu einem vereinfachten Datensatz, der nur aus denjenigen Merkmalen besteht, die wertvolle Informationen für die Clusteranalyse enthalten. Erfolgt diese Reduktion der Dimensionen durch eine lineare Transformation, so spricht man vom Subspace Clustering. Im Gegensatz zu einer einfachen, vorangehenden Dimensionsreduktion können beim Subspace Clustering die Zwischenergebnisse des Clusterings die resultierenden Merkmale erheblich beeinflussen. Bei der Betrachtung vom Subspace Clustering muss man zwischen traditionellen Methoden, die für jedes Cluster einen eigenen Unterraum definieren, und solchen, die einen gemeinsamen Merkmalsraum für alle Cluster erstellen, unterscheiden. In dieser Dissertation befassen wir uns nur mit Subspace Verfahren, die einen einzelnen Unterraum erstellen, da sie eine bessere Vergleichbarkeit zwischen den Clustern bieten. Man kann noch einen Schritt weiter gehen und mehrere solcher Unterräume für mehrere Clustering-Ergebnisse erstellen. Dies ist bei sogenannten nicht-redundanten Clustering-Methoden der Fall. Die Ausführung dieser Ansätze ist oft eine Herausforderung, da geeignete Parameter für die Dimensionalität der Unterräume und die Anzahl der Clusterings und Cluster pro Clustering definiert werden müssen. Wir gehen dieses Problem an, indem wir die aktuellen Modi in jedem Unterraum zur Kodierung der Daten verwenden und das Minimum Description Length Prinzip anwenden. Dabei wird angenommen, dass das Modell mit den geringsten Kodierungskosten die beste Lösung darstellt.

Vorhandene Modi werden ebenfalls durch statistische Modalitätstests, wie dem Dip-Test der Unimodalität, analyisert. Der Dip-Test liefert einen Dip-Wert, der den Abstand zwischen einer empirische Verteilungsfunktion zu einer beliebigen unimodalen

Verteilung angibt. Mithilfe einer neuartigen Transformationsfunktion können wir außerdem angeben, wie wahrscheinlich es ist, dass eine unimodale oder multimodale Verteilung vorliegt, wodurch der Dip-Wert besser interpretierbar ist. Aufgrund der Differenzierbarkeit dieser Funktion wird sie weiter dazu verwendet, Projektionsachsen zu identifizieren, die einen hohen Grad an Multimodalität innerhalb der vorhandenen Cluster aufweisen. Kombiniert man diese Achsen, erhält man einen niedrigdimensionalen Merkmalsraum, der die wesentlichen Modi innerhalb des Datensatzes enthält, wobei jeder Modus ein eigenes Cluster definiert.

Für komplexe Datensätze benötigen wir leistungsfähigere Verfahren als lineare Transformationen, um geeignete Merkmalsräume für das Clustering zu erhalten. Nichtlineare Transformationen unter Verwendung neuronaler Netze werden häufig in diesen Fällen angewandt. Diese Kombination aus Clustering und neuronalen Netzen wird als Deep Clustering bezeichnet. Solche Methoden integrieren den Clustering-Vorgang direkt in die Optimierungsfunktion bspw. eines Autoencoders. Auch hier stellt sich das Problem, einen geeigneten Wert für die Anzahl der Cluster zu wählen. Eine Lösung besteht darin, die anfängliche Anzahl der Cluster bewusst zu überschätzen und auf der Grundlage der aktuellen Modalitäten, die durch den Dip-Test ermittelt werden, zu entscheiden, ob zwei Cluster zusammengefasst werden können. Das Embedding passt sich dann an die vorhandene Clusterstruktur an. Dadurch muss die Anzahl der Cluster nicht im Voraus bekannt sein, sondern wird vom Algorithmus selbstständig erlernt. Weitere Analysen zeigen, dass der Dip-Test direkt zum Optimieren eines Autoencoders verwendet werden kann. Hierzu nutzen wir den Gradienten des Dip-Tests zum Trainieren der Parameter eines Autoencoders und für die Identifikation hochwertiger Clustering-Ergebnisse. Es muss keine spezifische Verteilungsfunktion für die transformierten Daten vorausgesetzt werden. Die einzige Annahme ist, dass jedes Cluster eine unimodale und jedes Clusterpaar eine multimodale Struktur annimmt. Dies stellt eine Reduzierung der Annahmen gegenüber etablierten Methoden dar.

Die beschriebenen Ansätze erfüllen ihr Ziel, informative Strukturen in großen, hochdimensionalen Datensätzen zu identifizieren, ohne dass Annotationen erforderlich sind. Im Vergleich zu konkurrierenden Methoden erzielen sie sehr gute Ergebnisse, wobei die Parametrisierung stark vereinfacht ist. Alle in dieser Arbeit vorgestellten Verfahren, werden durch unser Open-Source-Paket ClustPy bereitgestellt.

Acknowledgments

I want to thank the many great people who have accompanied me during my doctoral studies. Special thanks go to:

- My Ph.D. supervisor, Prof. Dr. Christian Böhm, who always supported me during my studies and was available in case I faced any problems. Thank you for making it possible for me to obtain my Ph.D. and for giving me all the freedom regarding the research content.
- My reviewers, Prof. Aristidis Likas, Ph.D., and Prof. Dr. Sibylle Hess, who agreed to review this thesis and spend their valuable time on it.
- Prof. Dr. Albrecht Schmidt for his willingness to act as chairman of the examination committee and for the pleasant atmosphere during the defense.
- Prof. Dr. Claudia Plant for the many valuable suggestions to improve my publications and the opportunity to cooperate with colleagues from the University of Vienna.
- My wonderful colleagues in Munich and Vienna for the numerous fruitful discussions. In particular, I would like to thank Lena Bauer, Dominik Mautz, Lukas Miklautz, Walid Durani, Mauritius Klein, and Benjamin Schelling, who helped me better understand many topics.
- Prof. Dr. Thomas Seidl, whose door was always open in case I needed any advice.
- Susanne Grienberger, Franz Krojer, and Joao Cardoso, who assisted me with many technical and administrative tasks.
- My family, my friends, and especially Bettina Chocholaty, who always built me up when things were not going well and without whose support this work would not have been possible.

Additionally, I would like to thank all the other amazing people with whom I have researched, taught, discussed, or simply had fun during this time.

Bibliographic Note

The topics discussed in this cumulative thesis have already been presented at prestigious scientific conferences and published in conference proceedings. The content is based on the following publications:

Main Publications

[LBS+21] Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. Dip-based deep embedded clustering with k-estimation. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 903–913. ACM, 2021.

[LMPB22] Collin Leiber, Dominik Mautz, Claudia Plant, and Christian Böhm. Automatic parameter selection for non-redundant clustering. In *Proceedings of the 2022 SIAM International Conference on Data Mining, SDM 2022, Alexandria, VA, USA, April 28-30, 2022*, pages 226–234. SIAM, 2022.

[LBN+22] Collin Leiber, Lena G. M. Bauer, Michael Neumayr, Claudia Plant, and Christian Böhm. The dipencoder: Enforcing multimodality in autoencoders. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 846–856. ACM, 2022.

[BLBP23] Lena G. M. Bauer, Collin Leiber, Christian Böhm, and Claudia Plant. Extension of the dip-test repertoire - efficient and differentiable p-value calculation for clustering. In *Proceedings of the 2023 SIAM International Conference on Data Mining, SDM 2023, Minneapolis-St. Paul Twin Cities, MN, USA, April 27-29, 2023*, pages 109–117. SIAM, 2023.

[LMPB23b] Collin Leiber, Lukas Miklautz, Claudia Plant, and Christian Böhm. Benchmarking deep clustering algorithms with clustpy. In *IEEE International Conference on Data Mining, ICDM 2023 - Workshops, Shanghai, China, December 4, 2023*, pages 625–632. IEEE, 2023.

Further Publications

[KLB23] Mauritius Klein, Collin Leiber, and Christian Böhm. k-submix: Common subspace clustering on mixed-type data. In *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part I*, volume 14169 of *Lecture Notes in Computer Science*, pages 662–677. Springer, 2023.

[MSL+23] Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch, Christian Böhm, and Claudia Plant. Non-redundant image clustering of early medieval glass beads. In *10th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2023, Thessaloniki, Greece, October 9-13, 2023*, pages 1–12. IEEE, 2023.

[LMPB23a] Collin Leiber, Lukas Miklautz, Claudia Plant, and Christian Böhm. Application of deep clustering algorithms. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, pages 5208–5211. ACM, 2023.

Contents

1 Introduction

> "We are drowning in information but starved for knowledge"
> - John Naisbitt (1982) [Nai82]

As the amount of available data is permanently growing, there is increasing interest in data mining methods capable of extracting valuable knowledge from this data. In order to grasp the potential offered by such approaches, one should consider that the amount of data generated per day was about 2.5 exabytes[1] in 2012 [MBD+12], and it is expected to increase to 463 exabytes per day by 2025 [RMR21]. Scientists cannot investigate this amount of data manually, so methods for automated data analysis are needed. Yet, even when considering modern machine learning approaches, a large amount of data does not automatically guarantee that valuable information is identified. Often, some amount of annotated data is required. The annotation of samples must usually be done by hand - often by domain experts - and can take multiple minutes per instance [SCF08]. Thus, it can become a very costly operation. Even powerful deep learning applications like ChatGPT[2] can only partially assist this manual work [ZZuH+23]. Therefore, unsupervised learning methods that do not require annotations play an essential role in data mining.

Numerous unsupervised clustering methods have been introduced in the past few decades [XT15]. "Clustering techniques attempt to group points in a multidimensional space in such a way that all points in a single group have a natural relation to one another and points not in the same group are somehow different" [DJ76]. These groups are also referred to as clusters. Here, the type of objective an algorithm pursues defines the kind of similarity measure, which leads to different categories of clustering methods. Examples are approaches based on partitions, densities, or hierarchies [XT15].

A problem faced by many clustering procedures is that modern data sets like texts, images, and videos are often large-scale and have numerous features. These characteristics can lead to the *Curse of Dimensionality* [Bel66]. This term describes the phenomenon that the more dimensions there are, the smaller the difference in distance between objects. As a result, the optimization functions of many clustering algorithms no longer produce satisfactory results [AHK01]. For this reason, the research field of *subspace clustering* has been established. Here, one tries to perform the clustering process in lower-dimensional

[1] 1 exabyte $= 10^{18}$ bytes
[2] `https://chat.openai.com/` (accessed 2024/01/10)

feature spaces by combining the clustering objective with a simultaneous reduction of dimensions [KKZ12]. In contrast to performing dimensionality reduction techniques, such as *Principal Components Analysis* (PCA) [Pea01], as a preprocessing step, in subspace clustering, intermediate clustering results can influence the resulting features. This feature selection can then lead to an improvement of the final clustering result. One can distinguish between *traditional subspace clustering* and *common subspace clustering* algorithms. The former define a separate subspace for each cluster, making a comparison of the clusters more difficult [GHPB14]. For this reason, we only deal with common subspace clustering in this thesis. Here, a single subspace is formed where the clustering process takes place. This subspace can help to better interpret the resulting clustering solution [MYPB17]. The idea of common subspace clustering can be taken a step further by creating multiple subspaces for multiple clustering results that describe different characteristics of the data. In this case, the term *non-redundant clustering* is often used.

While most subspace clustering methods employ linear transformations to define suitable subspaces, these operations are sometimes insufficient to analyze complex data sets [YFSH17]. For this reason, the research field of *deep clustering* has significantly grown in recent years. Here, a clustering objective is combined with the abstraction capabilities of a neural network, often in the form of an autoencoder [Bal87]. Thereby, non-linear relationships are recognizable by the clustering algorithm. In addition, batch-wise data processing enables the handling of larger data sets [Scu10]. Early deep clustering methods such as DEC [XGF16] have already shown promising results on image and text data sets. Since most deep clustering approaches define a single embedding for all clusters, they can be seen as a deep learning-based version of common subspace clustering.

Unfortunately, applying common subspace, non-redundant, and deep clustering methods usually requires parameters that can only be set with prior knowledge. These parameters heavily influence both the clustering results and the resulting subspaces. One example of such a parameter is the 'best' number of clusters, which is often unknown [HE03]. However, a simple parameterization is essential in the unsupervised domain as we want to avoid an expensive manual analysis of the given data set. The result is a trade-off between the ability to extract meaningful patterns from complex data sets and parameterization complexity, as illustrated in Fig. 1.1. Traditional clustering algorithms like k-Means [Llo82] only need clustering-specific parameters, subspace methods like FOSSCLU [GHPB14] often demand additional details regarding the feature reduction, and deep clustering approaches like DEC [XGF16] require information about the architecture and the optimization strategy. This thesis addresses the complex parameterization of methods that use feature transformations to analyze complex data sets. For this purpose, we are particularly interested in statistical modes. The modes of a data set are equal to the values in the feature space that occur most frequently or, in the case of continuous variables, the local maxima of the probability density function [Par62].

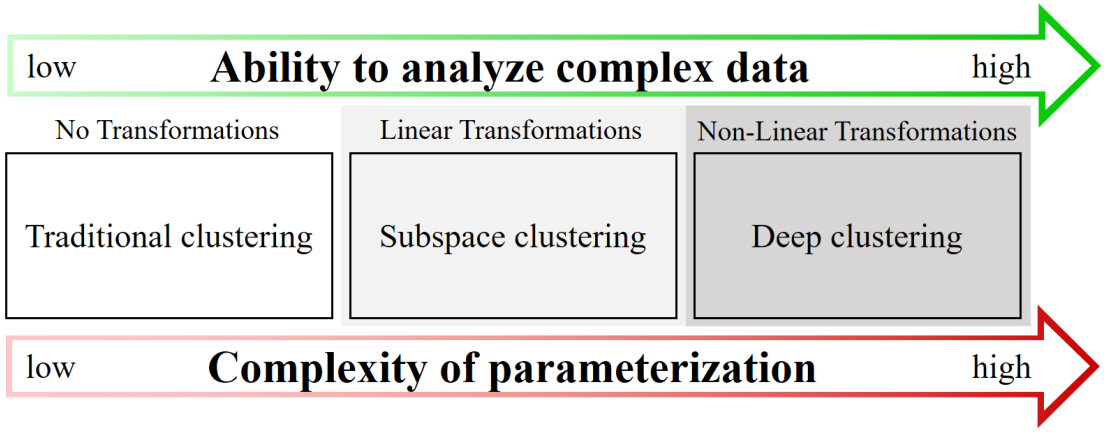


Figure 1.1: *When applying clustering methods that use feature transformations, it is crucial to recognize that while the ability to identify meaningful patterns in complex data increases, the parameterization's complexity increases simultaneously.*

Main Research Question

How can the modes in lower-dimensional feature spaces help simplify the parameterization of clustering methods, which handle complex data sets through linear or non-linear transformations?

1.1 Research Scope

This cumulative thesis tackles the stated research question by proposing novel common subspace clustering, non-redundant clustering, and deep clustering methods. These perform either a linear or non-linear transformation and are characterized by a simplified parameterization compared to competitor algorithms. We achieve this by analyzing the modes present in the resulting subspaces.

First, we present the algorithm AutoNR [LMPB22]. While the execution of most non-redundant clustering algorithms requires much prior knowledge, e.g., the number of clusterings and clusters per clustering, AutoNR automatically identifies sufficient values for the necessary parameters by utilizing the modes in each subspace to encode the data using the *Minimum Description Length* (MDL) [Ris78]. MDL is an information-theoretic strategy to choose the most sufficient model for a specific task. Employing a greedy heuristic, in which subspaces and individual clusters are split and merged, various models are obtained and evaluated based on their coding cost. The evaluated model with the lowest MDL cost will finally be returned. In addition, the encoding strategy can be used to identify samples not fitting the structures found, also referred to as outliers.

As an alternative to information-theoretic methods, modes can also be analyzed by

statistical modality tests. A test frequently used for unsupervised machine learning approaches, e.g., in [KL12, MP16, CL17, CL18, SBBP20], is the *Dip-test of unimodality* by Hartigan and Hartigan [HH85]. This test returns a so-called Dip-value that indicates how prominent the multimodality of the input samples is. As it does not require complex input parameters, it is of particular interest for clustering. Often, the Dip-value is not used directly, but the corresponding probability value, which indicates the likelihood that the input data shows a unimodal behavior. Precalculated look-up tables are usually used to obtain this value [HH85]. In [BLBP23], we substitute these tables by introducing a transformation function that calculates the corresponding probability value for each Dip-value and sample size. Further, the gradient of the Dip-test regarding a projection axis, as presented in [KL05], can be extended by this new function. These advantageous properties allow the development of the common subspace clustering algorithm Dip'n'Sub, which identifies clusters and a suitable subspace only by considering modalities. Neither the number of clusters nor the dimensionality of the subspace is needed as input parameter.

The number of clusters is not only a required parameter for clustering techniques that apply linear transformations but also for most deep clustering methods. We present the algorithm DipDECK [LBS+21], which offers a much simpler parameterization. Instead of an exact number of clusters, only an upper bound has to be defined. For this purpose, our method considers the modes present in the embedding of an autoencoder. Based on the Dip-test, clusters are merged if they show a unimodal structure. Thus, the number of clusters is successively reduced. The autoencoder is then able to adapt to the new structure of clusters.

We propose another innovative application of the Dip-test in our work on the DipEncoder [LBN+22]. Here, we integrate the Dip-test even further into the optimization of an autoencoder. For this purpose, we use the already mentioned differentiability of the Dip-test [KL05] to generate an embedding in which all pairs of clusters show a high degree of multimodality. At the same time, individual clusters should have a coherent, i.e., unimodal, structure. This process ensures a clear separation of the clusters. Unlike other methods, this approach does not require an assumption about the underlying data distribution within the embedding.

The implementations of the described procedures and relevant comparable algorithms are available via our open-source Python package ClustPy[3]. This package is used in [LMPB23b] to benchmark various deep clustering algorithms using a consistent setting. As a result, they can be compared in a meaningful way in order to highlight their strengths and weaknesses. Here, various parameterizations and their effects on the clustering result are examined. In the context of this benchmark, the DipEncoder [LBN+22] outperforms its competitors in many scenarios, emphasizing its applicability.

[3] `https://github.com/collinleiber/ClustPy` (accessed 2024/01/10)

1.2 Thesis Structure

The remainder of the thesis is structured as follows. In Chapter 2, we describe the foundations of the proposed approaches, including the Minimum Description Length, the Dip-test of unimodality, linear and non-linear feature transformations, as well as relevant clustering strategies. Chapter 3 then introduces our main contributions. A conclusion complemented by certain limitations and potential opportunities for future work is given in Chapter 4. The original publications on which this cumulative dissertation is based can be found in Appendix C. Appendix A and B contain additional experiments relating to these publications.

2 Foundations

In this thesis, we refer to several basics from the areas of information theory, statistical modality tests, linear transformations, non-linear transformations, and, in particular, clustering. The necessary information will be briefly presented in the following. Note that the details required to understand a specific topic are also discussed in the papers in Appendix C.

Before explaining the actual foundations, we define some relevant symbols in Tab. 2.1.

Table 2.1: *Definitions of the symbols used in this thesis.*

Symbol	**Definition**
$N \in \mathbb{N}$	Size of a data set
$d \in \mathbb{N}$	Dimensionality of a data set
$\mathcal{D} \subset \mathbb{R}^d$	A data set
$x_i \in \mathcal{D}$	A sample of data set $\mathcal{D}$
$k \in \mathbb{N}$	The number of clusters
$\mathcal{C}_j \subset \mathcal{D}$	Samples contained in cluster j
$Dip \in (0, 0.25]$	The Dip-value
$p_{Dip} \in [0, 1]$	The probability value associated with Dip
$\rho \in \mathbb{R}^d$	A projection axis
$m \in \mathbb{N}$	Dimensionality of the embedding/subspace
$\mathcal{B} \subset \mathcal{D}$	A batch of data
$\text{enc}(x_i) = z_i \in \mathbb{R}^m$	Embedding of x_i
$\text{dec}(z_i) = \hat{x}_i \in \mathbb{R}^d$	Reconstruction of x_i
$\lambda_1, \lambda_2 \in \mathbb{R}$	Weight of the reconstruction ($\mathcal{L}_{rec}$) and clustering ($\mathcal{L}_{clust}$) loss

2.1 Minimum Description Length

The Minimum Description Length (MDL) [Ris78] is a widespread information-theoretic strategy to solve model selection problems. It has been frequently used in clustering to identify outliers or a suitable number of clusters, amongst other purposes [BLS99, BFPP06, KKMC07, BFP08, GHPB14].

MDL is based on the idea that the more regularities can be found in a data set, the more it can be compressed [Grü05]. At the same time, a model achieving a higher compression

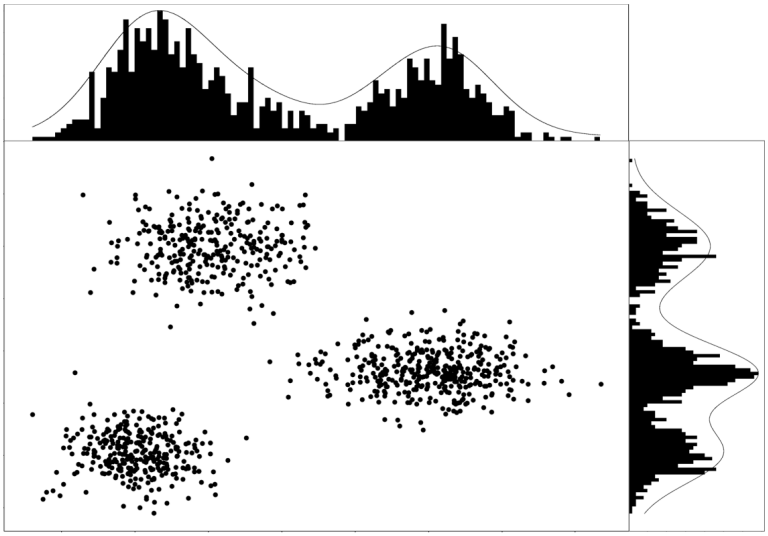
equals a better model, which automatically prevents overly complex models. "As such, MDL embodies a form of Occam's Razor, a principle that is both intuitively appealing and informally applied throughout all the sciences" [Grü05]. Usually, so-called two-part codes are used to evaluate the quality of a model [Lee01]. This means the encoding consists of the two terms $L(H)$ and $L(\mathcal{D}|H)$. $L(H)$ describes the number of bits necessary to encode the hypothesis [Grü05]. In the clustering domain, this can include the cluster centers, cluster assignments, and other cluster properties such as covariance matrices. $L(\mathcal{D}|H)$, in turn, specifies the number of bits that are necessary to encode the data using the described hypothesis [Grü05]. For this purpose, it is usually assumed that suitable probability distributions can represent the data. Here, *Shannon-Fano Coding* [Sha48] can be exploited to convert probabilities into code lengths, where a code length describes the number of bits necessary to encode specific information. It states that the code length L of a probability p equals $L(p) = \lceil -\log_2(p) \rceil$. However, for a floating point $x \in \mathbb{R}$, often no probability but only a probability density $\pi(x)$ is available. In this case, the code length can only be specified to a precision δ, where the code length equals $L(x) = -\log_2(\pi(x)) - \log_2(\delta)$ [Lee01]. In addition, integer values can be encoded using the *Universal Prior for Integers* [Ris83]. It implies that any integer $n \in \mathbb{N}$ can be encoded using $L(n) = \log^*(n) + \log_2(c)$ bits, where $\log^*(n)$ recursively calculates $\log^*(n) = \log_2(n) + \log^*(\log_2(n))$ until the term becomes negative and $c \simeq 2.865064$ is a constant. A good overview of possibilities for encoding data is given in [Lee01] and [Grü05].

Let us consider an example: Suppose we have a two-dimensional data set, as shown in Fig. 2.1a. We can see three modes suggesting different creation processes. Let us consider three different models that we can use to encode the data:

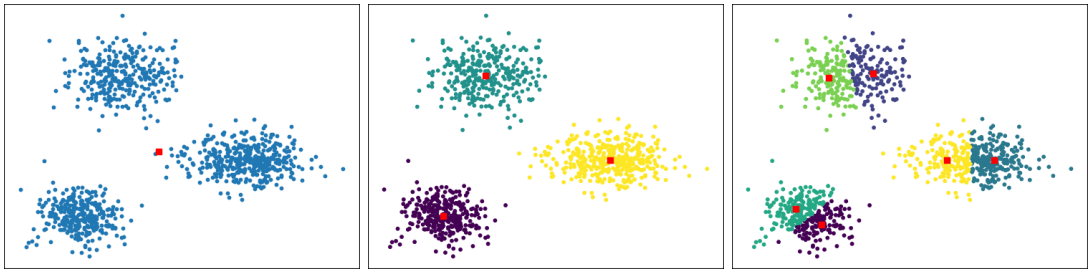
- **Model 1** - A single cluster (Fig. 2.1b): $L(H)$ is very cheap as we only have to encode a single cluster center - requiring the encoding of $d = 2$ values, one for each feature - and no cluster assignments. On the other hand, it will be challenging to find commonalities, i.e., a suitable distribution function with which to encode the data. Therefore, $L(\mathcal{D}|H)$ will likely be very large.

- **Model 2** - Three clusters (Fig. 2.1c): $L(H)$ is expected to be of moderate size since three cluster centers and the corresponding cluster assignments must be encoded. The data within each cluster can be encoded using a Gaussian distribution with an appropriate covariance matrix. This symmetric matrix requires $\frac{d(d+1)}{2}$ values to encode. Since this model should fit well, $L(\mathcal{D}|H)$ is supposed to be very small.

- **Model 3** - Six clusters (Fig. 2.1d): In this case, the clusters of 'Model 2' have been split in two. Due to these additional clusters, $L(H)$ approximately doubles. At the same time, it is unlikely that a much better distribution function will be found. Therefore, $L(\mathcal{D}|H)$ is supposed to be only slightly smaller.



(a) The input data set *consists of three Gaussian-distributed clusters. The histograms at the top and the right indicate the data distribution on the x- and y-axis, respectively.*



(b) Model 1: *A single cluster.* ***(c) Model 2:*** *Three clusters.* ***(d) Model 3:*** *Six clusters.*

Figure 2.1: *The images show a synthetic data set consisting of* 1000 *samples. The samples are distributed around three modes.* ***(a)*** *shows the raw data set,* ***(b)-(d)*** *show different variants to cluster this data set. The coloring indicates the cluster assignments and the red squares mark the positions of the cluster centers.*

From these considerations, it can be deduced that 'Model 2' is correctly assumed to be the best model. Further, we can see that MDL prevents overfitting [Grü05], as this would lead to a high hypothesis cost $L(H)$, as demonstrated by 'Model 3'.

2.2 The Dip-test of Unimodality

The Dip-test of unimodality by Hartigan and Hartigan [HH85] is a statistical test that rates the modality of one-dimensional samples. It returns a so-called Dip-value $Dip \in (0, 0.25]$ that indicates the deviation from a unimodal distribution. Accordingly, a small Dip represents a unimodal distribution, and a large Dip a rather multimodal distribution. For the calculation of Dip, no target distribution function, e.g., a Gaussian or Laplacian distribution, or other complex parameters need to be defined. This property distinguishes

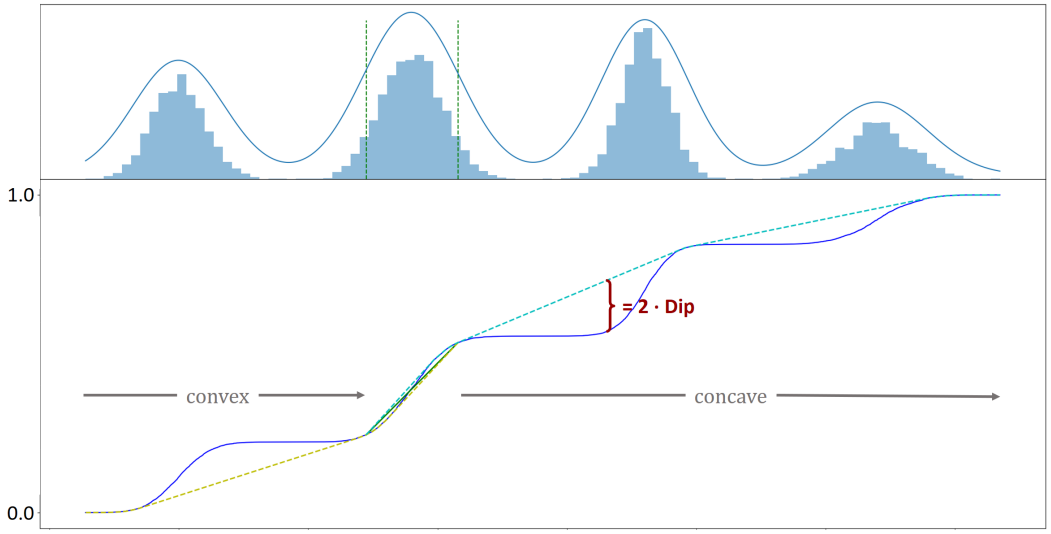


Figure 2.2: *Exemplary calculation of the Dip-value using a data set with* $10{,}000$ *samples drawn from* $\mathcal{N}(0,1) \cup \mathcal{N}(9,1.2) \cup \mathcal{N}(18,0.9) \cup \mathcal{N}(27,1.3)$*. Here,* $\mathcal{N}(a,b)$ *describes a normal distribution with mean* a *and standard deviation* b*. The upper part shows a histogram of the data, and the lower part shows the calculation of the Dip-value using the ECDF (blue line) and a fitted unimodal piece-wise linear function (yellow, green, and cyan line). The resulting Dip-value is equal to* $Dip = 0.081$*.*

the Dip-test from other statistical tests such as the *Anderson–Darling Test* [AD52] or the *Kolmogorov–Smirnov Test* [MJ51]. In addition, for a sorted data set, the Dip-test has a complexity of $O(N)$ [HH85], where N corresponds to the number of samples, which makes it attractive for applications concerning large data sets.

In order to calculate the Dip-value, the empirical cumulative distribution function (ECDF) of the data is considered, and the unimodal piece-wise linear function with the smallest maximum distance to the ECDF is determined. This distance is equal to two times the Dip-value and is illustrated in Fig. 2.2. The figure shows a data set consisting of four modes whose distribution is represented by the histogram shown in the upper part and whose ECDF corresponds to the blue line in the lower part. In general, a unimodal distribution is characterized by an ECDF being convex at first, followed by the steepest slope, and changing to a concave behavior afterward [MP16]. An example of a fitted unimodal piece-wise linear function is also visualized in Fig. 2.2. Here, the yellow-dashed line represents the convex part, the green line represents the steepest slope, and the cyan-dashed line represents the concave part. The steepest slope is synonymous with the primary mode of the data set [MP16]. A more detailed description of the computation of the Dip-value can be found in [HH85, KL05, MP16, LBN+22].

Despite the many good properties of the Dip-test, the resulting Dip-value Dip is often difficult to interpret since the magnitude depends heavily on the number of samples. The more samples there are, the smaller the resulting Dip concerning unimodal data sets.

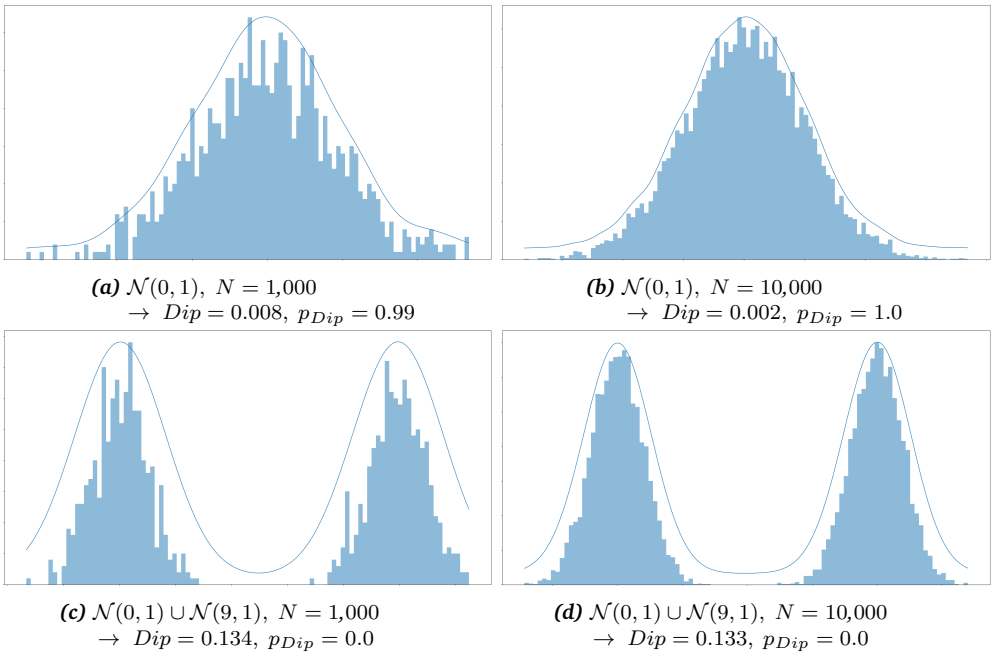
This behavior can be observed, for example, in Fig. 2.3a and 2.3b. Although the same distribution function is used for sampling in both cases and only the data size varies, the Dip-values with 0.008 and 0.002 differ significantly. In addition, the maximum Dip-value of 0.25 is a rather theoretical limit, which is rarely seen in practice. For example, neither the distribution in Fig. 2.3c nor Fig. 2.3d reaches such a high value, although a multimodal distribution is clearly present. It is, therefore, necessary to make a sensible assessment of a threshold to assume a multimodal distribution. For this purpose, one often does not use the Dip-value directly but the corresponding probability value p_{Dip}. This value indicates the likelihood of a unimodal distribution being present. It is calculated by sampling several times from a uniform distribution with the same sample size as the input data. For each sampling, the corresponding Dip-value is determined. The percentage of samplings with a larger Dip-value than that of the input data determines the resulting p_{Dip} [HH85]. A low Dip, therefore, leads to a high p_{Dip} and a high Dip to a low p_{Dip}. Fig. 2.3 shows that the probability value allows a clear and meaningful assessment of multimodality in all four cases.

Let us consider an example: We have a data set of size $N = 100$ with a Dip-value of $Dip = 0.1$ and want to obtain the corresponding probability value p_{Dip}. Therefore, we create ten artificial sample sets of the same size $N = 100$ using the uniform distribution. These have Dip-values of $\mathcal{A} = \{0.04, 0.05, 0.07, 0.07, 0.08, 0.09, 0.11, 0.12, 0.13, 0.14\}$ (arbitrary values). This results in the probability value $p_{Dip} = \frac{|\{Dip_a | Dip_a \in \mathcal{A} \wedge Dip_a > Dip\}|}{|\mathcal{A}|} = 0.4$. Note that the number of artificial sample sets would be much higher in a real scenario.

The uniform distribution is used for the sampling process as it is the “least-favorable” [HH85] or “worst case” [CAS19] unimodal distribution. In other words, it can be seen as a borderline case between unimodal and multimodal distributions. Note that when using the Dip-test, a statement can only be made as to whether a unimodal distribution is present or not. No conclusions can be drawn about the number of modes present. For instance, the Dip-value of the distribution shown in Fig. 2.2 is lower than that from Fig. 2.3d, although there are two additional modes and the same sample size is used.

2.3 Feature Transformations

When working with complex, high-dimensional data sets, data mining methods can struggle to produce high-quality results [AHK01]. This is due to computational issues [LCW+18] and the *Curse of Dimensionality* [Bel66], which causes distances in high-dimensional spaces to become increasingly similar. At the same time, the *Empty Space Phenomenon* arises, which states that the proportion of empty space between the data points increases as the number of dimensions increases [LCW+18, AHK01]. For these reasons, data mining methods are often performed on a dimensionality-reduced version of the original data set. This approach has “the advantages of improving learning per-



(a) $\mathcal{N}(0,1),\ N = 1{,}000$ $\rightarrow\ Dip = 0.008,\ p_{Dip} = 0.99$

(b) $\mathcal{N}(0,1),\ N = 10{,}000$ $\rightarrow\ Dip = 0.002,\ p_{Dip} = 1.0$

(c) $\mathcal{N}(0,1) \cup \mathcal{N}(9,1),\ N = 1{,}000$ $\rightarrow\ Dip = 0.134,\ p_{Dip} = 0.0$

(d) $\mathcal{N}(0,1) \cup \mathcal{N}(9,1),\ N = 10{,}000$ $\rightarrow\ Dip = 0.133,\ p_{Dip} = 0.0$

Figure 2.3: *Histograms of samplings from different distributions with a varying number of samples N and their corresponding Dip-value Dip as well as the probability value p_{Dip}. Here, $\mathcal{N}(a, b)$ describes a normal distribution with mean a and standard deviation b.*

formance, increasing computational efficiency, decreasing memory storage, and building better generalization models" [LCW+18]. Dimensionality reduction techniques can be divided into linear and non-linear transformations. Linear methods preserve the linear relationship of data points [CG15], which is not necessarily valid for non-linear methods.

Another possible categorization is into supervised and unsupervised techniques [LCW+18, NH19]. One might assume that only unsupervised methods are relevant in the context of clustering, as no predefined labels are known. While this is true for a preprocessed dimensionality reduction, intermediate clustering labels can be used to execute supervised transformation methods. This idea is taken up by *subspace clustering* algorithms, which are discussed in Sec. 2.4.3.

2.3.1 Linear Transformations

Linear feature transformations are essential methods for data mining and are often used for preprocessing a data set. Even simple methods such as random projections can be beneficial for machine learning approaches [Das00, BM01]. The best-known techniques are *Singular Value Decomposition* (SVD) and *Principal Component Analysis* (PCA) [Pea01]. These algorithms can determine the orthogonal projections on which the data exhibits the most significant variance. *Independent Component Analysis* (ICA) [JH91] is another

frequently used method. It identifies components that are statistically independent of each other. The Dip-test has also been used to identify suitable transformations for clustering [KL05, MP16, SP18, SBBP20]. For this purpose, the gradient of the Dip-value regarding a projection axis ρ [KL05] is often exploited. This gradient allows the usage of *Stochastic Gradient Descent* (SGD) to determine those projection axes on which the data shows the greatest multimodality, indicating the essential structures for clustering within the data set [KL05, MP16, SBBP20].

If the feature transformation should not take place as a preprocessing step, but during the clustering process, it is advisable to use supervised dimensionality reduction techniques instead of unsupervised ones. Such methods can make use of available label information, known initially or determined by a clustering algorithm, to improve the resulting feature space. A well-known representative is *Linear Discriminant Analysis* (LDA) [Fis38]. Here, the intra-cluster variance is minimized while the inter-cluster variance is maximized. Other supervised dimension reduction techniques are, for example, *Partial Least Squares* [Wol66] or *Neighborhood Component Analysis* [GRHS04]. For more information on linear transformations, see [CG15, LCW+18, NH19].

2.3.2 Non-linear Transformations Using Autoencoders

Non-linear transformations offer more flexibility to reduce the dimensionality of an input data set than linear transformations. A well-known group of non-linear methods are based on manifolds. These usually try to preserve the local neighborhoods of data points in low-dimensional spaces [TSL00]. Known representatives are *Isometric Feature Mapping* (ISOMAP) [TSL00], *t-Stochastic Neighbour Embedding* (t-SNE) [VdMH08], and *Uniform Manifold Approximation and Projection* (UMAP) [MH18]. In the context of this thesis, *autoencoders* (AEs) [Bal87] are of particular interest. These are unsupervised neural networks, which consist of two sub-modules: the *encoder* and the *decoder*.

In order to obtain a high-quality representation of the data, AEs usually only use small parts of the data at once [KMN+17], also referred to as batch-wise optimization. Assume we have a batch $\mathcal{B}$ of the entire data set $\mathcal{D}$, i.e., $\mathcal{B} \subset \mathcal{D} \subset \mathbb{R}^d$. The encoder maps each sample $x_i \in \mathcal{B}$ into a latent space consisting of m features, where usually $m < d$ applies. This mapping yields the embedding $z_i = \text{enc}(x_i), z_i \in \mathbb{R}^m$, which equals the lower-dimensional representation for subsequent tasks. The decoder then tries to retrieve the original information from the embedded features $\hat{x_i} = \text{dec}(\text{enc}(x_i)), \hat{x_i} \in \mathbb{R}^d$. Often, the architecture of the decoder corresponds to the inverted architecture of the encoder. The idea behind this neural network is to preserve sufficient information in the embedding, which serves as the bottleneck of the architecture, to allow a high-quality reconstruction. The quality of the transformation is assessed using an arbitrary differentiable loss function $\mathcal{L}_{rec}$ - often, the *Mean Squared Error* (MSE) is used - and the

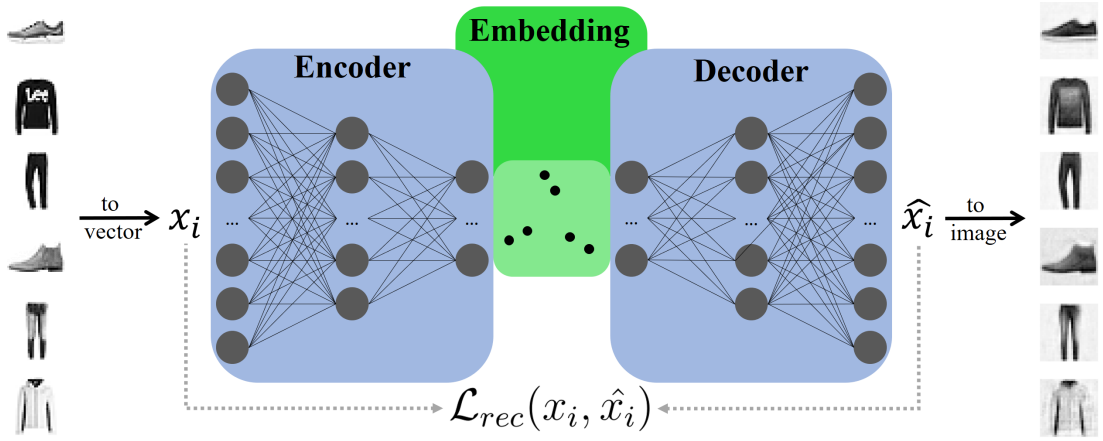


Figure 2.4: *Illustration of a typical feedforward autoencoder application. The input images (in this case, from the grayscale Fashion-MNIST data set [XRV17]) are first converted to a feature vector. The data is then mapped to a latent space representation consisting of two features ($m = 2$) by the encoder and finally converted back to the original feature space by the decoder. The similarity is evaluated by employing $\mathcal{L}_{rec}$. Ultimately, the resulting output images should be similar to the input images.*

network is updated by backpropagating the error.

$$\mathcal{L}_{rec}(\mathcal{B}) \overset{\text{often}}{:=} \text{MSE}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{x_i \in \mathcal{B}} ||x_i - \hat{x}_i||_2^2, \tag{2.1}$$

where $|\mathcal{B}|$ is the size of the batch $\mathcal{B}$. Various optimization techniques, such as classical SGD, ADAGRAD [DHS11], or ADAM [KB15], can be used for the backpropagation.

A schematic illustration of a simple feedforward AE is shown in Fig. 2.4. Due to the lower-dimensional embedding, there is a loss of information, and the input images cannot be fully recovered. This fact can be recognized, among other differences, by the missing lettering 'Lee' on the second input image, showing a sweater. However, this property is desired as it ensures the generalizability of the network. A variety of extensions are available that offer different improvements. For example, *Convolutional Autoencoders* [LBD+89] and *Vision Transformers* [DBK+21] are particularly suitable for the analysis of image data. These architectures eliminate the need to convert images into a vector representation, as displayed in Fig. 2.4. AEs can further be used for generative tasks, e.g., in the form of *Variational Autoencoders* [KW14].

2.4 Clustering

Clustering refers to unsupervised machine learning methods that autonomously divide data into groups, also called clusters. Consequently, unlike classification methods, no

predefined labels are required to train the model. Clusters can be defined based on different characteristics like distances to prototypes (e.g., k-Means [Llo82] or the *Expectation Maximization* (EM) algorithm [DLR77]), neighborhood information (e.g., DBSCAN [EKSX96] or Density Peak Clustering (DPC) [RL14]), or graph theory (e.g., spectral clustering [vL07] or k-Multiple-Means [NWL19]). A good overview is given in [XT15]. Since prototype-based methods are particularly relevant for this thesis, we discuss them in more detail below.

2.4.1 Prototype-based Clustering

The importance of prototype-based methods can be well demonstrated by looking at the top 10 "most influential data mining algorithms in the research community" [WKQ+08]. In this list, two clustering paradigms appear, namely k-means [Llo82] and fitting finite mixture distributions - usually Gaussian Mixture Models - using the EM algorithm [DLR77]; both are prototype-based methods. Procedures that belong to this group assign data points to clusters based on their position in relation to cluster centers. The best-known representative is Lloyd's variant of the k-Means algorithm [Llo82], which tries to group the data into k clusters. Here, the first step is to select $k \in \mathbb{N}$ data points randomly and use their coordinates to position the initial cluster centers $\mu \subset \mathbb{R}^d$, where $|\mu| = k$. Subsequently, all samples are assigned to the cluster corresponding to their closest cluster center ('assignment phase'). The cluster centers are then updated by setting their position to the mean value of the assigned data points ('update phase'). The assignment and update phases are repeated until convergence is achieved. Thus, k-Means tries to minimize the *within-cluster sum of squares* (WCSS) [Sch23], also called the *sum of the squared error* (SSE) [Jai10], regarding all clusters, i.e., the global variance.

$$WCSS(\mathcal{D}) = \sum_{j=1}^{k} \sum_{x_i \in \mathcal{C}_j} ||x_i - \mu_j||^2 \tag{2.2}$$

The advantages of k-Means are versatile. The algorithm is easy to understand and shows good runtime behavior [Jai10]. Furthermore, the parametrization process is simple, as only the number of clusters k needs to be specified as an input parameter. For these reasons, k-Means has already been extended by a variety of functionalities. For example, there are extensions for an improved selection of initial cluster centers [AV07, CKV13, XDLL15], the determination of outliers [CG13, GN17, HCK+05], fuzzy cluster assignments [DUN74, Bez81], a deterministic execution [LVV03, LLZ15], or the handling of mixed-type attributes [Hua97, AD07]. A disadvantage of k-Means is that the clusters can only take spherical shapes [Jai10]. The EM algorithm overcomes this limitation by using soft cluster assignments, which means that each point is not hard as-

signed to a single cluster but belongs to each cluster with a certain probability. Further, it optimizes cluster-specific covariance matrices rather than a single global variance. Thus, each cluster can adopt an individual distribution, which allows greater flexibility. No additional input parameters are required to achieve these advantages. However, the greater flexibility is accompanied by a higher runtime [XT15]. Another problem with k-Means and the EM algorithm is that they struggle to achieve good results on high-dimensional data sets because of the Curse of Dimensionality [Ass12, DHZS02].

2.4.2 Estimating the Number of Clusters

In the following, we discuss yet another drawback of most prototype-based methods. As already discussed, the number of clusters k is the only necessary input parameter for k-Means and the EM algorithm. However, a suitable number of clusters is often unknown in advance, so methods are necessary to determine this parameter automatically. Algorithms based on neighborhoods between samples, such as DBSCAN [EKSX96] or OPTICS [ABKS99], are able to do this. However, neighborhood-defining parameters, which are also not trivial to choose, are required to achieve good results with those algorithms.

When considering our research question of analyzing modes, mode-seeking approaches like Mean-shift [Che95], Medoid-shift [SKK07], Quick-shift [VS08], or DPC [RL14] represent another reasonable choice to estimate the number of clusters. Here, dense regions are determined by considering all samples within a specific area and applying a kernel function. The points with the highest density are then considered the modes of the data set, and the samples are assigned to the best matching mode. We want to refrain from using kernels and range parameters, which are often difficult to specify, especially in transformed feature spaces. For this reason, instead of density-based procedures, we investigate prototype-based methods that can intrinsically determine the number of clusters k.

A popular strategy is to look at the WCSS of k-Means results with different values for k and use the *elbow method* [Tho53] to pick the most appropriate solution. Here, the WCSS results are plotted with ascending k, and the result at the location of the most prominent kink is assumed to be the clustering result with the best number of clusters. However, this approach has some drawbacks. For example, there is not necessarily a distinct most prominent kink, so the final result depends on the selection procedure [Sch23]. In addition, the elbow method struggles to detect a very high number of clusters correctly [Sch23]. The *Gap Statistic* [TWH01] tackles these issues by comparing the k-Means results against uniformly distributed samplings. This approach has the advantage that the procedure directly provides a clear recommendation for the most appropriate number of clusters.

There are also extensions of prototype-based clustering methods that integrate the

estimation of k directly into their optimization strategy. An example is Bayesian k-Means [WK06]. This algorithm uses a *Maximization Expectation* strategy to estimate the hidden variables of a mixture model. By splitting and merging various clusters, it tries to identify a model with optimal log model evidence.

Other methods are based on information theory, with X-Means [PM00] being the most prominent representative. This algorithm starts with a low number of clusters and successively splits the current clusters into two. After each split, the result is evaluated using the *Bayesian Information Criterion* (BIC) [Sch78] or the *Akaike Information Criterion* (AIC) [Aka74] to decide whether the new clustering result is kept. BIC and AIC are information-theoretic criteria for model selection, similar to the MDL principle [Ris78]. The splitting mechanism of X-Means is repeated until no split leads to a better BIC/AIC score. The X-Means algorithm has been extended in [Ish05] to include a merging step to fine-tune the final clustering result. Approaches that start with a small number of clusters, which is successively increased, are also referred to as *top-down* methods. Another approach that uses information theory is presented in [BLS99]. It starts with an initial clustering solution with a high number of clusters. MDL is then used to decide which clusters to merge and which points to define as outliers. The process is repeated until the MDL cost cannot be further reduced. In contrast to X-Means, this approach is referred to as a *bottom-up* process.

Applying statistical tests is another strategy to intrinsically estimate the number of clusters. There are two main characteristics that distinguish such clustering processes - the type of statistical test and the strategy used to transform the data into a one-dimensional representation. The second characteristic is required due to the limitation of most statistical tests to one-dimensional data. The best-known algorithm that uses a statistical test to determine the number of clusters is G-Means [HE03]. It uses the Anderson–Darling statistical test [AD52] to check whether the data within a cluster is normally distributed. For this purpose, k-Means with $k = 2$ is executed within each cluster, and the data is projected onto the connecting line of the resulting cluster centers. If the test rejects a normal distribution, the two new clusters are kept. Otherwise, the original cluster is restored. The algorithm PG-Means [FH06] randomly creates several projection axes onto which the Gaussian Mixture Model, obtained by the EM algorithm, is projected. From these one-dimensional models, data is sampled and compared with the actual projected data using the Kolmogorov–Smirnov statistical test [MJ51]. If the test rejects a single projected model, a new cluster is added, and the process repeats.

DipMeans [KL12], ProjectedDipMeans [CL18], SkinnyDip [MP16], and the procedure proposed in [CL17] all utilize the Dip-test of unimodality [HH85] to estimate the number of clusters. The methods differ heavily in the one-dimensional transformation used. DipMeans [KL12] does not consider the coordinates of the data points, but for each sample, the distances to other samples in the same cluster. For each of these sets of distances, the

Dip-value and the associated probability value are obtained. If the proportion of objects indicating a multimodal distribution is higher than a predefined threshold, the cluster is split. ProjectedDipMeans [CL18] calculates the Dip-values concerning each feature of the data set and each principal component of PCA applied to the samples within a cluster. If a single projection indicates multimodality within a cluster, the most multimodal cluster is split. SkinnyDip [MP16] also considers the samples projected onto certain projection axes. In the case of detected multimodality, the clusters are recursively split until all of the sub-clusters are unimodally distributed on this axis. Here, the steepest slope, as identified by the Dip-test, defines the cluster boundaries. In [CL17], the Dip-test is utilized for image segmentation by initially overestimating the number of segments, i.e., clusters. Two neighboring cluster centers are then selected, and the distances between these centers to all pixels, i.e., samples, assigned to either of them are computed. Afterward, the Dip-test is used to decide whether these two sets are distributed unimodally. In this case, the clusters are merged. This process repeats until convergence is achieved. The employed transformation is thus similar to that proposed in [KL12].

In contrast to the methods mentioned above, M-Dip [CAS19] and SpecialK [HD19] are not limited to convex cluster shapes. M-Dip [CAS19] computes the k-nearest neighbor graph and splits a cluster if the path between two samples sees a significant drop in density. The specific threshold is obtained by sampling multiple times from a uniform distribution. This process is similar to calculating the probability value p_{Dip} of the Dip-test. SpecialK [HD19] uses probability bounds defined by the *Bernstein inequality* to decide whether data points are more likely to originate from one or two distributions. This decision is used as a cluster-splitting criterion to identify an appropriate number of clusters. Since SpecialK is not based on a prototype-based clustering approach but on spectral clustering [vL07], non-convex clusters can also be detected.

As all the procedures based on statistical tests, except the one presented in [CL17], start with a small number of clusters, they belong to the top-down approaches. Further, unlike those based on information theory, all of these methods require a significance threshold α as an input parameter that determines whether a hypothesis is accepted or rejected. Therefore, the parameter k is substituted by α, where α is often more intuitive to define than k.

2.4.3 Common Subspace Clustering

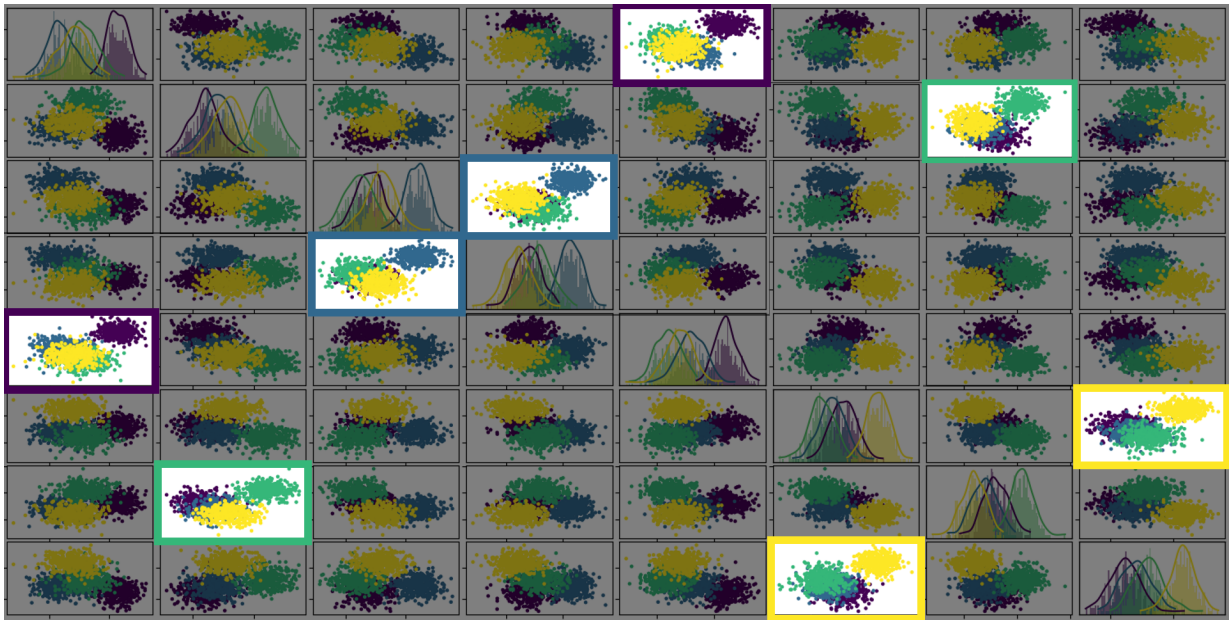
As pointed out in Sec. 2.3, many data mining techniques face issues processing high-dimensional data. In order to circumvent this problem, the data can be converted to a lower-dimensional feature space (subspace). In contrast to simply preprocessing the data set using a feature transformation technique and executing an algorithm in the resulting subspace, we want the objective function of the data mining approach to influence the

resulting feature space. Accordingly, the resulting subspace should be tailored to the specific clustering task. For example, the subspace should change depending on whether the user wants to identify three or five clusters. This field of research is referred to as *subspace clustering*. Such methods use intermediate cluster assignments to adjust the resulting subspace, and the subspace, in turn, influences the resulting cluster assignments.

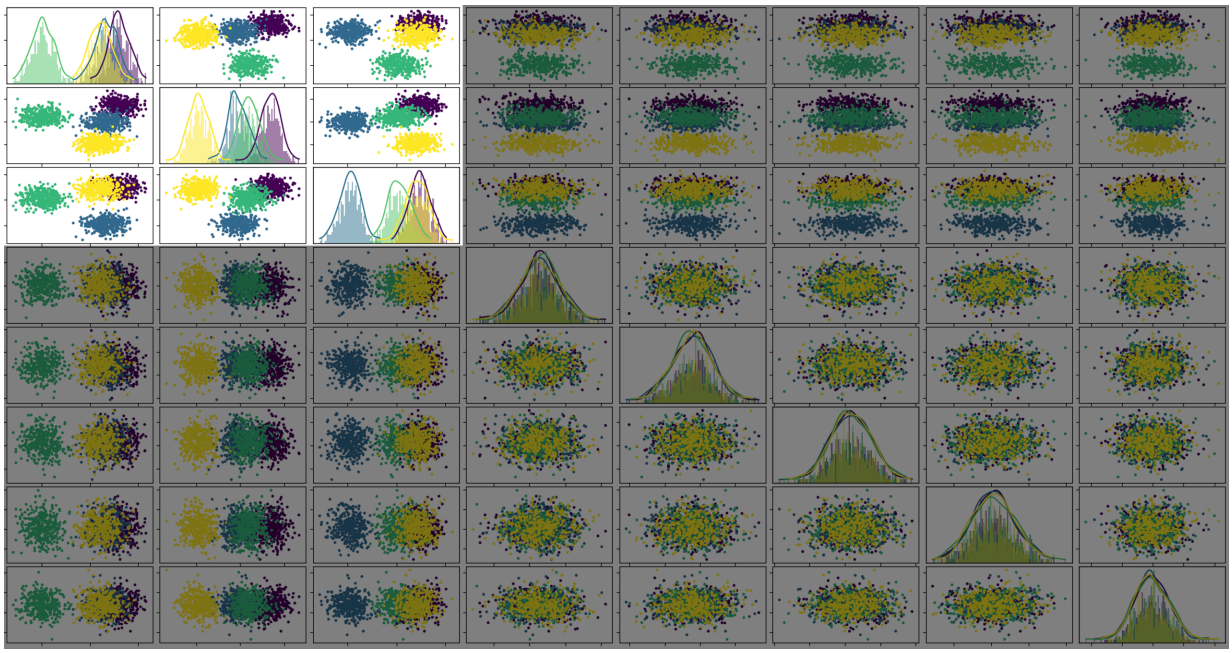
Here, a distinction can be made between traditional and common subspace clustering. While traditional subspace clustering has been extensively studied (e.g., in [PHL04, KKZ12, SGZC13]), common subspace clustering is often overlooked. In the case of traditional methods, each cluster usually gets its own axis-parallel or arbitrarily oriented subspace. Accordingly, a cluster is defined as $(\mathcal{O}, A)$, where $\mathcal{O}$ are the samples in that cluster and A is the set of associated features [SGZC13]. With common subspace clustering, a single shared subspace is defined for all clusters, which is usually hidden in an arbitrarily orientated version of the data. This way, "we can study not only the intra-cluster but also the inter-cluster relationships of objects" [GHPB14]. Thus, the common subspace facilitates a subsequent analysis of the results; one can visualize the clusters and examine the differences accordingly. Such visualizations are relevant in unsupervised learning to evaluate whether meaningful patterns have been identified.

Let us consider an example: In Fig. 2.5 we can see a potential result of a traditional (2.5a) and a common (2.5b) subspace clustering algorithm regarding a data set with eight features and four clusters. While in traditional subspace clustering, each cluster is assigned its specific features, e.g., features one and five for the purple cluster, in common subspace clustering, the goal is to map all clusters into a common feature space. This joint subspace makes the relationships between the clusters visible, allowing us to see in Fig. 2.5b that we only require three features to distinguish all clusters.

The most straightforward way to define a common subspace for all clusters is to use regular linear or non-linear dimensionality reduction techniques like PCA or ICA as a preprocessing step (see Sec. 2.3). However, in this case, we do not utilize intermediate clustering results and, therefore, ignore valuable information. For this reason, several methods have been proposed that use interim cluster labels to improve the concurrent optimization of the subspace. For example, ADR-EM [DHZS02] uses intermediate results of k-Means or the EM algorithm to enhance the subspace identified by SVD or QR decompositions [Par94]. LDA-k-Means [DL07] follows a similar strategy by using intermediate k-Means labels to execute LDA repeatedly. The algorithm FOSSCLU [GHPB14] combines the EM algorithm with rigid transformations, and SubKmeans [MYPB17] optimizes a k-Means-based loss through eigenvalue decompositions. For all these methods, the number of clusters k must be known beforehand. Additionally, apart from SubKmeans, they all require the size of the resulting subspace m as an input parameter. In the case of FOSSCLU, a procedure is proposed to determine these parameters automatically using MDL [GHPB14]. However, this is based on a brute-force search of all parameter combi-



(a) *Traditional subspace clustering. The colored border indicates the cluster that is considered in the corresponding features.*



(b) *Common subspace clustering.*

Figure 2.5: *Scatter matrix plot for a potential result of* ***(a)*** *traditional and* ***(b)*** *common subspace clustering on a synthetic data set with eight features and four clusters. The samples are colored using the ground truth labels, and relevant features are highlighted.*

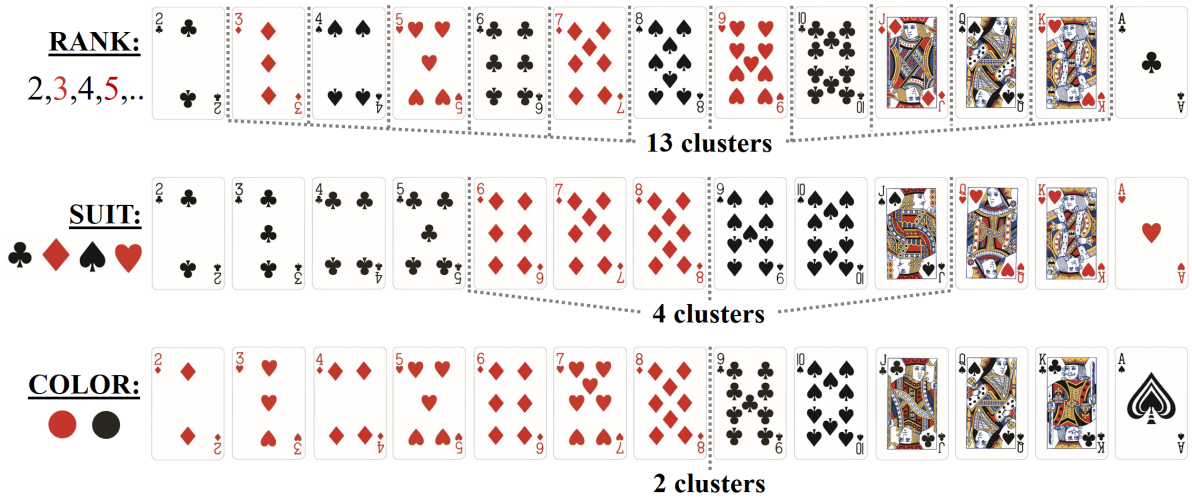


Figure 2.6: *Example use case of non-redundant clustering. The images belong to a deck of playing cards with* 52 *different cards that can be clustered by either rank* $(2, 3, \ldots, 10$*, Jack, Queen, King, Ace), suit (clovers, tiles, pikes, hearts), or color (black, red).*

nations within a given range, which can notably increase the runtime. Therefore, in this thesis (see Sec. 3.2), we analyze how a sophisticated common subspace algorithm can automatically determine sufficient values for both parameters k and m.

2.4.4 Non-Redundant Clustering

In some cases, it is not sufficient to identify a single clustering solution, but one would like to obtain several, each describing a different characteristic of the data set. Multiple clustering solutions can be useful to get better insights into a complex topic. *Non-redundant clustering* algorithms attempt to identify several clustering results that are as different as possible. Therefore, they impose specific non-redundant constraints. We want to show the benefit of such a constraint by means of an example. Assume we have decks of playing cards as shown in Fig. 2.6[4]. These cards can be clustered in multiple different ways, e.g., by rank $(2, 3, \ldots, 10$, Jack, Queen, King, Ace), suit (clovers, tiles, pikes, hearts), or color (black, red). Individually analyzing these clusterings is more accessible than a single global clustering result, consisting of 52 clusters. However, it is only of limited use to cluster by both suit and color since these results are partially redundant; tiles and hearts are always red, and pikes and clovers are always black. Therefore, it is sufficient to consider the 'rank' and the 'suit' clustering, which would be the desired outcome in most non-redundant clustering settings.

Methods for identifying non-redundant clusterings differ particularly in how non-redundancy is ensured. minCEntropy [NE10], for example, applies an optimization

[4]Source: `https://acbl.mybigcommerce.com/52-playing-cards/` (accessed 2023/12/02)

strategy based on conditional entropy, where the different clusterings should share a minimum amount of information. Dec-k-Means [JMD08] optimizes the k-Means loss of two different clustering results, which are simultaneously showing a low correlation. In the following, we focus on approaches that are based on feature transformations where each clustering obtains its own feature space. Often, these feature spaces come in the form of subspaces, where each feature belongs to one specific subspace. This process ensures that each clustering describes different characteristics of the data and can be interpreted as an extension of common subspace clustering.

Let us consider an example: If we execute a transformation-based non-redundant clustering algorithm on the playing cards from Fig. 2.6, two subspaces are conceivable; one for the rank, e.g., containing the information regarding the shown rank in the upper-left and lower-right corner, and one for the suit, e.g., containing the information regarding the color. A potential solution is illustrated in Fig. 2.7, where the two clustering results can be clearly distinguished. The first, capturing the information regarding the rank, is mapped to features one and two (framed in red), and the second, capturing the information regarding the suit, is mapped to features three and four (framed in blue). Here, the points are colored based on the labels from the first clustering result. Accordingly, the coloring of the clusters in the first subspace is homogeneous, while it is heterogeneous in the second subspace. This behavior is desired as in the second subspace, each cluster represents a suit, and 13 distinct ranks exist for each suit. We can look at the full-dimensional cluster centers to better understand the main characteristics of the subspaces. Four cluster centers of the first subspace (5, 9, Queen, and Ace) are shown on the left side. Here, the ranks of the cards are easily recognizable. The suit, however, is difficult to identify, and the color is dark red in all cases. The cluster centers of the second subspace are illustrated on the right side. Here, the ranks of the cards are not recognizable, but one can identify which cluster represents which suit.

A non-redundant clustering algorithm that is based on feature transformations is Orth [CFD07]. This procedure repeatedly executes k-Means within newly identified feature spaces orthogonal to existing clustering solutions. ISAAC [YMHP16] is a transformation-based method which uses *Independent Subspace Analysis* (ISA) [HH00], an extension of ICA, to identify multiple subspaces for clustering and executes the EM algorithm in those. ISA is also utilized by MISC [WWD+19], whereby kernel graph regularized semi-nonnegative matrix factorization (KGSNMF), an extension of SNMF [DLJ10], is used in the resulting subspaces to obtain the clustering solutions. The non-redundant clustering algorithm mSC [NDJ10] can detect non-spherical clusters in multiple subspaces by using the Hilbert-Schmidt Independence Criterion (HSIC) [GBSS05] in combination with spectral clustering [vL07].

Most of the existing transformation-based methods define the subspaces first and then fit a clustering solution within those subspaces. However, in this case, the clustering does

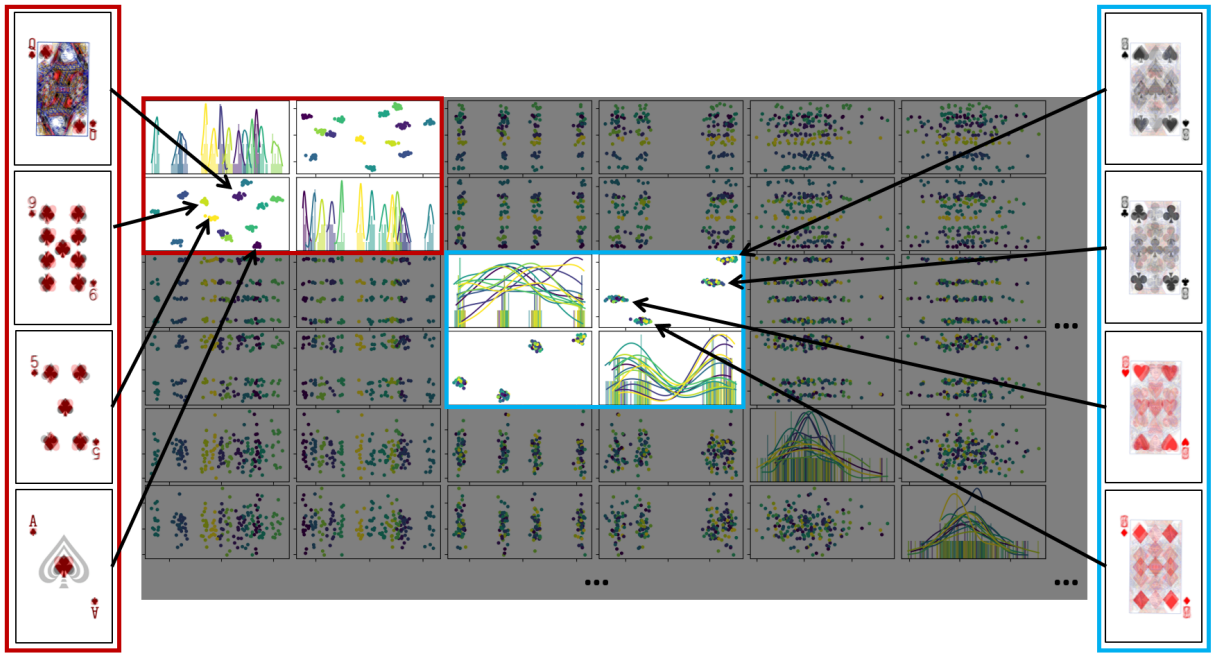


Figure 2.7: *Scatter matrix plot of a potential subspace-based non-redundant clustering result regarding a data set consisting of images showing playing cards (see Fig. 2.6). The first subspace consists of the first two features (framed in red) and represents the 13 distinct ranks. The second subspace consists of the third and fourth features (framed in blue) and represents the four different suits. The remaining features contain no relevant information and are not considered in detail. The playing cards next to the scatter matrix plot display four exemplary cluster centers within the first subspace (left) and the four cluster centers within the second subspace (right). All scatters are colored based on the labels of the first subspace.*

not influence the resulting features. In contrast, NrKmeans [MYPB18], which can be considered a generalization of the common subspace algorithm SubKmeans [MYPB17], updates the subspaces during the clustering process. In each iteration, the clustering results are obtained by running k-Means in each subspace. The subspaces are then optimized by performing eigenvalue decompositions before the next iteration starts. This idea was extended by NrDipMeans [MYPB20] to automatically identify an appropriate number of clusters in each subspace using the Dip-test. This process greatly simplifies the parameterization since only the number of subspaces needs to be specified. Of the methods mentioned, only ISAAC and MISC can automatically define the number of clusterings and clusters by applying MDL and, in the case of MISC, Bayesian k-Means [WK06].

In summary, most methods exhibit at least one of the following problems. They cannot simultaneously optimize the feature spaces and the clustering, which can have a negative impact on the clustering quality, or they require the number of subspaces and the number of clusters per subspace as input parameters, making applicability difficult. In this thesis (see Sec. 3.1), we tackle these issues.

2.4.5 Deep Clustering

Since the presentation of DEC [XGF16] in 2016, deep clustering (DC) has been an integral part of the data mining landscape. DC describes the combination of clustering and deep learning and enables the processing of large and high-dimensional data sets. The motivation is twofold. First, it has been shown that k-Means can be applied to large-scale data sets when utilizing SGD and batch-wise optimization [BB94, Scu10]. Further, the complexity of high-dimensional data can be reduced using the representation capabilities of deep learning procedures, which can counteract the Curse of Dimensionality. These advantages have led to the introduction of various DC approaches. A good overview is given in the surveys [ZXZ+22] and [RPY+22].

A simple way to distinguish different DC methods is to look at the architecture used. Commonly employed architectures are feedforward AEs (e.g., used by DEC [XGF16], IDEC [GGLY17], DCN [YFSH17], DeepECT [MPB19], DKM [FTG20], ACe/DeC [MBM+21]) and Convolutional AEs (e.g., used by DCEC [GLZY17], DEPICT [DHD+17], ENRC [MMA+20]). In addition, there are procedures using encoder-only models (e.g., JULE [YPB16], DeepCluster [CBJD18]) or generative models such as Variational AEs [KW14] (e.g., used by VaDE [JZT+17]) or *Generative Adversarial Networks* [GPM+14] (e.g., used by ClusterGAN [MALK19]). Since, in the context of this thesis, we are only dealing with approaches that use a simple feedforward AE, we limit ourselves to these architectures in the following. Note that applications using a feedforward AE can usually be combined with a Convolutional AE without major adjustments. For example, DCEC [GLZY17] is an extension of IDEC [GGLY17] that substitutes the feedforward AE with a Convolutional AE.

In addition to the reconstruction loss $\mathcal{L}_{rec}$ (see Sec. 2.3.2), AE-based DC methods typically define a clustering loss $\mathcal{L}_{clust}$. This loss is supposed to shape the embedding so that 'clustering-friendly' [YFSH17] representations are obtained, where clustering-friendly is synonymous with small distances between points within a cluster and large distances between points from different clusters. The combined loss function $\mathcal{L}_{total}$ looks as follows:

$$\mathcal{L}_{total}(\mathcal{B}) = \lambda_1 \mathcal{L}_{rec}(\mathcal{B}) + \lambda_2 \mathcal{L}_{clust}(\mathcal{B}), \tag{2.3}$$

where $\lambda_1 \in \mathbb{R}$ and $\lambda_2 \in \mathbb{R}$ are variables to weight the reconstruction and clustering loss. The purpose of this loss function is visualized in Fig. 2.8. Here, the general structure of the neural network is the same as that of a regular AE (see Fig. 2.4). However, one can see that the embedded data z_i is closer to other objects from the same cluster, which is achieved by the additional loss $\mathcal{L}_{clust}$.

In terms of the workflow, most DC procedures are similar. First, an AE is pretrained for a certain number of epochs, using only $\mathcal{L}_{rec}$. Afterward, a traditional clustering algorithm

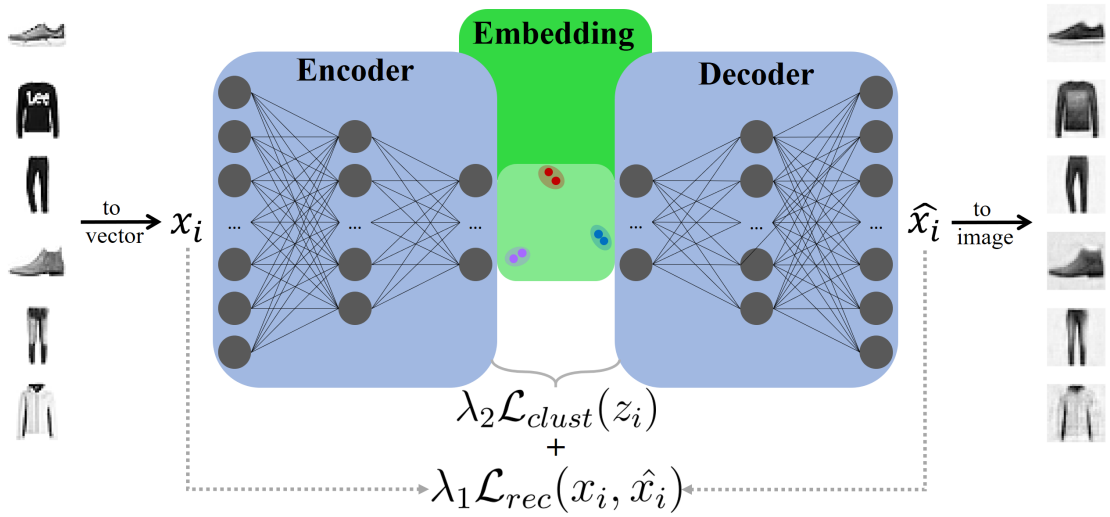


Figure 2.8: *Illustration of a typical deep clustering application. Compared to Fig. 2.4 (showing the regular feedforward autoencoder architecture), a new loss $\mathcal{L}_{clust}$ is introduced, which uses the embedded data z_i as input and provides a clustering-friendly structure in the embedding.*

is executed in the resulting embedding to obtain initial cluster labels. These labels form the starting point for the DC objective $\mathcal{L}_{clust}$, which improves the embedding and the final cluster labels. Therefore, DC algorithms can also be seen as deep learning methods for refining a given clustering result.

There is a wide variety of options for choosing $\mathcal{L}_{clust}$. DEC [XGF16] uses a loss function based on the Kullback-Leibler divergence. Here, the data distribution within the embedding, quantified by a kernel based on a Student's t-distribution, is compared with an auxiliary target distribution. An unique aspect is that DEC only uses $\mathcal{L}_{rec}$ for pretraining and λ_1 is set to 0 during the actual clustering process. The authors of IDEC [GGLY17] note that this can lead to a distorted embedding. Therefore, they suggest to use the objective of DEC, but setting $\lambda_1 > 0$ during the clustering optimization. As a result, the clusters obtained by DEC are typically much more compact than those identified by IDEC. Another popular strategy for choosing $\mathcal{L}_{clust}$ is to employ optimization strategies inspired by k-Means. For example, DCN [YFSH17] successively updates the data representation and the clustering parameters, i.e., the cluster centers and assignments. Accordingly, the clustering result and the embedding are not improved simultaneously but alternately. This step-wise optimization allows hard cluster assignments even though the objective function is not differentiable. DKM [FTG20] proposes a differentiable k-Means-like optimization. For this purpose, pseudo-hard labels are applied utilizing a parameterized softmax function.

In addition to these objectives, numerous overlaps exist with the topics discussed in the previous sections. For example, SCDE [DAMS19] can define a suitable number of

clusters for its deep clustering procedure. For this purpose, it uses a separate AE with a softmax layer at its core. ACe/DeC [MBM+21] combines DC and common subspace clustering by only using a subset of the embedding for clustering. Here, the features are softly assigned to the different subspaces, making the optimization differentiable. ENRC [MMA+20] uses a similar approach to obtain non-redundant clustering results in a deep learning setting. As for other DC algorithms, an initial - in this case non-redundant - clustering result is required, which is refined during the optimization.

A primary concern regarding the mentioned methods is the necessity of strong assumptions regarding the expected distributions within the embedding. In addition, the application of deep learning processes is accompanied by a large number of hyperparameters that can have a strong influence on the final clustering result. Both issues are addressed in this thesis (see Sec. 3.3, 3.4, and 3.5).

3 Contributions

In the following, we briefly present the main publications that form the basis of this cumulative thesis. Here, the essential contributions and their improvements compared to competitor algorithms will be highlighted. A detailed description of the proposals can be found in the corresponding papers in Appendix C. In addition, Appendix C contains a detailed division of work among the authors.

The publications share several mutual features, the most prominent being the inclusion of clustering. Moreover, all presented methods use a feature transformation to identify suitable feature spaces for clustering. Both linear and non-linear transformations are applied. Usually, such combinations of a feature transformation and a simultaneous clustering process involve various parameters that are difficult to define. Our methods show a simplified parameterization compared to other methods that pursue a similar objective. In particular, in the case of algorithms based on linear transformations, we try to identify a suitable number of relevant features and clusters automatically. Further, we do not force specific cluster structures such as spherical or Gaussian-based cluster shapes for algorithms based on non-linear transformations. In the case of DipDECK [LBN+22], this also allows an automatic determination of a suitable number of clusters.

We obtain these capabilities by analyzing modes in the transformed feature spaces. These modes are evaluated by using either the MDL principle (AutoNR [LMPB22]) or by employing the Dip-test (Dip'n'Sub [BLBP23], DipDECK [LBS+21], DipEncoder [LBN+22]). Tab. 3.1 offers an overview of the presented methods and their specific characteristics.

Table 3.1: *The table shows the similarities and differences of the algorithms discussed in this thesis. All methods use a feature transformation and can automatically analyze modes in the resulting feature spaces. For this purpose, either the MDL principle or the Dip-test is applied. This process can provide an automatic estimation of the number of clusters k.*

Algorithm	Linear transf.	Non-linear transf.	Estimates k	Dip-test	MDL
AutoNR [LMPB22]	✗		✗		✗
Dip'n'Sub [BLBP23]	✗		✗	✗	
DipDECK [LBS+21]		✗	✗	✗	
DipEncoder [LBN+22]		✗		✗	

3.1 AutoNR

As mentioned in Sec. 2.4.4, non-redundant clustering methods allow a more intuitive interpretation of a given data set, as instead of a single global clustering result, several ones are provided. This variety allows the users to select the most relevant clustering result regarding their task. In addition, many methods directly return associated subspaces, which helps to interpret the characteristics of the clusters better. However, the applicability becomes more complicated since the number of expected clusterings and clusters per clustering must be known. In a real-life scenario, setting these parameters requires extensive prior analysis.

We present the non-redundant clustering algorithm AutoNR [LMPB22], which can determine appropriate values for these parameters automatically. It extends NrKmeans [MYPB18] so that no input parameters are required to obtain the non-redundant clustering result and the associated subspaces. This simplified execution is achieved using the MDL principle to evaluate various non-redundant clustering solutions and identify the most suitable one. Our contribution consists of three main components: the encoding strategy, search heuristic and outlier detection.

Encoding strategy. First, we introduce a sophisticated strategy to encode the hypothesis and the data in a non-redundant setting. This strategy must fit the underlying clustering algorithm; otherwise, the encoding would not be able to evaluate the clustering result meaningfully. For example, it makes only limited sense to encode density-based clusters identified by DBSCAN [EKSX96] with the help of a Gaussian distribution. For clusters defined by the EM algorithm [DLR77], however, it would be a valid strategy. Note that AutoNR is based on NrKmeans [MYPB18], which in turn is based on a k-Means-like optimization. It has been shown that k-Means-based models can be encoded using a simplified Gaussian Mixture Model, where all clusters share a single-variance covariance matrix Σ, i.e., $\Sigma = \sigma^2 I$, where I is the identity matrix [BLS99]. Although such encodings have been studied intensively for traditional clustering, they are not trivially applicable to non-redundant clustering problems. Here, the transformation and subspace information must be encoded in addition to the clustering structures.

Search heuristic. The second component we propose is a suitable heuristic for searching the parameter space. To look at all possible parameter combinations is not expedient since the runtime would be too high. We present a heuristic that successively splits and merges the subspaces and the individual clusters in the subspaces. The best result is then determined based on the MDL encoding strategy.

Outlier detection. In addition to the number of clusterings and clusters per clustering, AutoNR is also able to detect outliers. This feature is based on the presented MDL encoding strategy, implying that no additional input parameters are necessary. To identify outliers, we check for each sample in each subspace whether it is favorable to

encode it as part of a cluster or as an isolated vector. Thus, we can recognize outliers subspace-specifically. In other words, outliers in one subspace do not necessarily have to be outliers in others. This property allows to fine-tune the individual clusters, leading to high-quality clusterings. Experiments on real-world data confirm that this can lead to improved clustering results.

Since AutoNR builds on NrKmeans [MYPB18], it can, unlike other parameter-free methods, optimize the clustering results and the corresponding subspaces simultaneously. Moreover, it shows an advantageous runtime behavior compared to those competitor algorithms.

Fig. 3.1 visualizes the result of AutoNR executed on the NrLetters data set [LMPB22], with additional outliers added. The original data set consists of $10{,}000$ 7×9 colored images, each showing a letter ('A', 'B', 'C', 'X', 'Y', 'Z') in the colors pink, cyan or yellow. In addition, one corner of each image (top left, top right, bottom left, bottom right) is highlighted in the same color as the letter, resulting in three clustering possibilities. The figure indicates that AutoNR correctly identifies all desired structures and all outliers after several splitting and merging steps. Details regarding the process by which this result was obtained are given in Appendix A.

Due to its strengths, AutoNR is used in [MSL+23] to analyze images of medieval glass beads. Here, although the deep clustering algorithm ENRC [MMA+20] obtains the final clustering solution, an initial non-deep, non-redundant clustering result is required. Since the parameters of the clusterings are initially unknown, and outliers are likely to be present in the data set, AutoNR is chosen for this purpose.

3.2 Dip'n'Sub

Various clustering methods use the Dip-test to decide whether a data distribution is uni- or multimodal (e.g., DipMeans [KL12], SkinnyDip [MP16], ProjectedDipMeans [CL18], StrDip [LZD+18], NrDipMeans [MYPB20], or the procedure proposed in [CL17]). A final statement is usually made by analyzing the probability value p_{Dip}. In order to obtain this value, the mentioned methods generally use a predefined look-up table, which specifies a probability value for multiple combinations of Dip-value Dip and sample size N. Hartigan and Hartigan introduced a table with 117 entries in [HH85]. This initial table was later extended to 546 entries[5]. Values not contained in the table are interpolated based on $\sqrt{N}Dip$ [HH85]. Nevertheless, even the enlarged table is constrained to data sets with a maximum of $72{,}000$ samples, which limits the applicability. In [BLBP23], we compute 307 pairs of Dip and p_{Dip} for 68 sample sizes to fit a sigmoid function that can determine valid probability values for all combinations of sample size N and Dip-value

[5] `https://cran.r-project.org/package=diptest` (accessed 2024/01/08)

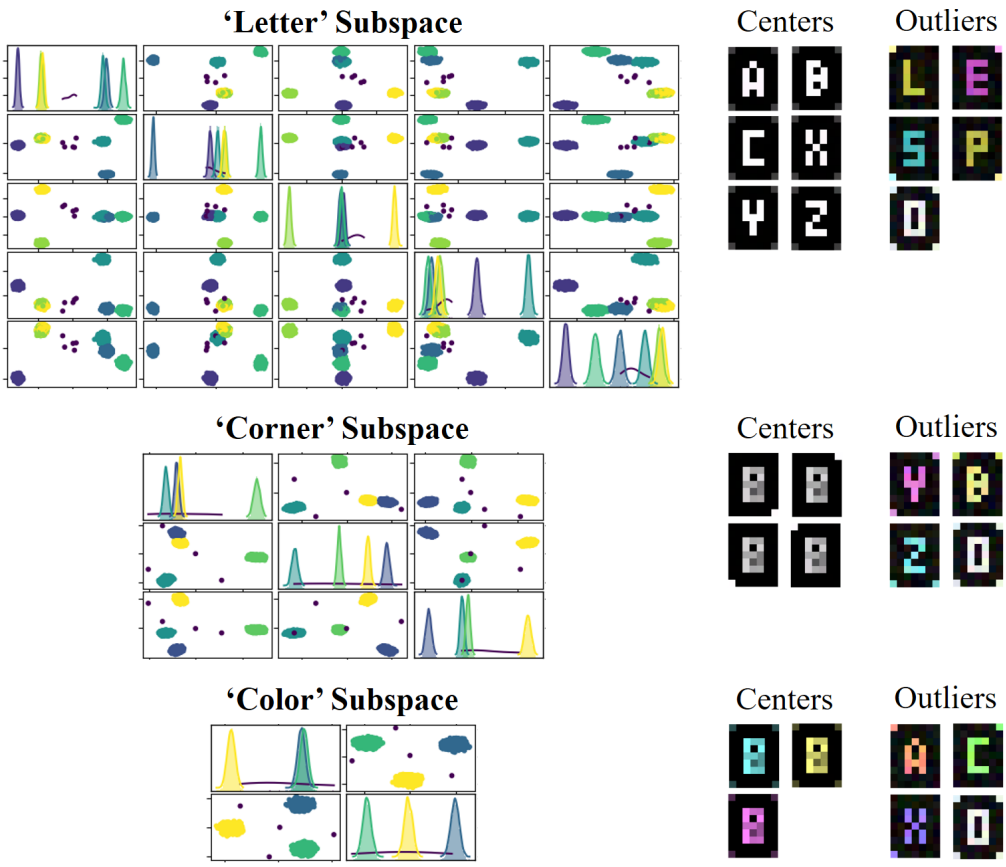


Figure 3.1: Result of AutoNR regarding the NrLetters data set: *The scatter matrix plots indicate that AutoNR successfully recognizes the three clustering possibilities hidden in the data set. These describe the shown letter ('A', 'B', 'C', 'X', 'Y', 'Z'), the marked corner (top left, top right, bottom left, bottom right), and the color of the letter (pink, cyan, yellow). Furthermore, AutoNR identifies the subspace-specific outliers, which were added to the data set and show a wrong letter, several marked corners, or a different color. No input parameters are necessary to achieve these clustering results. The coloring of the scatters corresponds to the labels in each subspace, and outliers are colored in purple.*

Dip. The resulting function is defined as:

$$\hat{p}(Dip, \hat{b}(N)) = 1 - \left[0.6(1 + 1.6e^{-\hat{b}(N)Dip+6.5})^{\frac{1}{1.6}} + 0.4(1 + 0.2e^{-\hat{b}(N)Dip+6.5})^{\frac{1}{0.2}}\right]^{-1}, \quad (3.1)$$

where $\hat{b}(N) = 17.30784\sqrt{N} + 12.04918$.

This function is continuously differentiable with respect to a projection axis ρ. Therefore, not only the gradient concerning the Dip-value, as explained in [KL05], but also the gradient regarding the probability value can be calculated. The following applies:

$$\nabla_\rho(\hat{p}(Dip, \hat{b}(N))) = \gamma \cdot \nabla_\rho(Dip), \quad (3.2)$$

where γ is a factor depending on N and Dip. In applications that consider the entire data set, such as DipExt [SBBP20], this gradient has no major advantage over directly applying $\nabla_\rho(Dip)$. Yet, it can be beneficial if a gradient regarding multiple probability values is to be calculated. In this case, the factor γ ensures that an intrinsic weighting of the individual gradients takes place.

To emphasize the usefulness of this aspect, we introduce the common subspace clustering algorithm Dip'n'Sub [BLBP23]. It uses SGD to successively search for those projection axes on which the existing clusters obtain the maximum average modality. The score S_p is to be minimized by these axes:

$$S_p = \frac{1}{N} \sum_{j=1}^{k} |\mathcal{C}_j| \hat{p} \left(\text{dip}(\overline{\mathcal{C}_j^\rho}), \hat{b}(|\mathcal{C}_j|) \right), \tag{3.3}$$

where $\mathcal{C}_j$ is the set of samples in cluster j, and $\text{dip}(\overline{\mathcal{C}_j^\rho})$ returns the Dip-value of $\mathcal{C}_j$ projected to ρ, i.e., $\overline{\mathcal{C}_j^\rho} = \{\rho^T x_i | x_i \in \mathcal{C}_j\}$. Subsequently, the clusters can be divided into sub-clusters based on the modes present on these projection axes. Here, we use a similar method as proposed in [MP16], which recursively executes the Dip-test to identify the steepest slope and, thus, the main mode in the projected data. This mode is regarded as a separate cluster and the process repeats without the newly identified cluster. In contrast to the procedure presented in [MP16], our subroutine TailoredDip can better capture the tails of distributions, leading to more precise cluster boundaries. The process terminates if no additional axis can be found on which more than $T\%$ of the samples are contained in a multimodal cluster. Here, T is a hyperparameter that the user has to set.

In summary, Dip'n'Sub automatically identifies a suitable subspace and the number of clusters using only the Dip-test. Since only modes on the identified axes are considered, and no further assumptions are made about the underlying distributions, our algorithm can recognize various convex cluster shapes. This ability distinguishes Dip'n'Sub from many comparison methods, which are often based on k-Means and can only recognize spherical clusters. A visualization of Dip'n'Sub's process is given in Fig. 3.2. Additionally, an experiment is conducted in Appendix B that indicates how Dip'n'Sub benefits from using the gradient regarding p_{Dip}.

3.3 DipDECK

While the automatic determination of a suitable number of clusters has been extensively examined in traditional clustering (see Sec. 2.4.2), the topic has yet to be addressed in deep clustering. Although it is easy to combine an AE with established k-estimation techniques by running these algorithms on the resulting embedding, they often do not

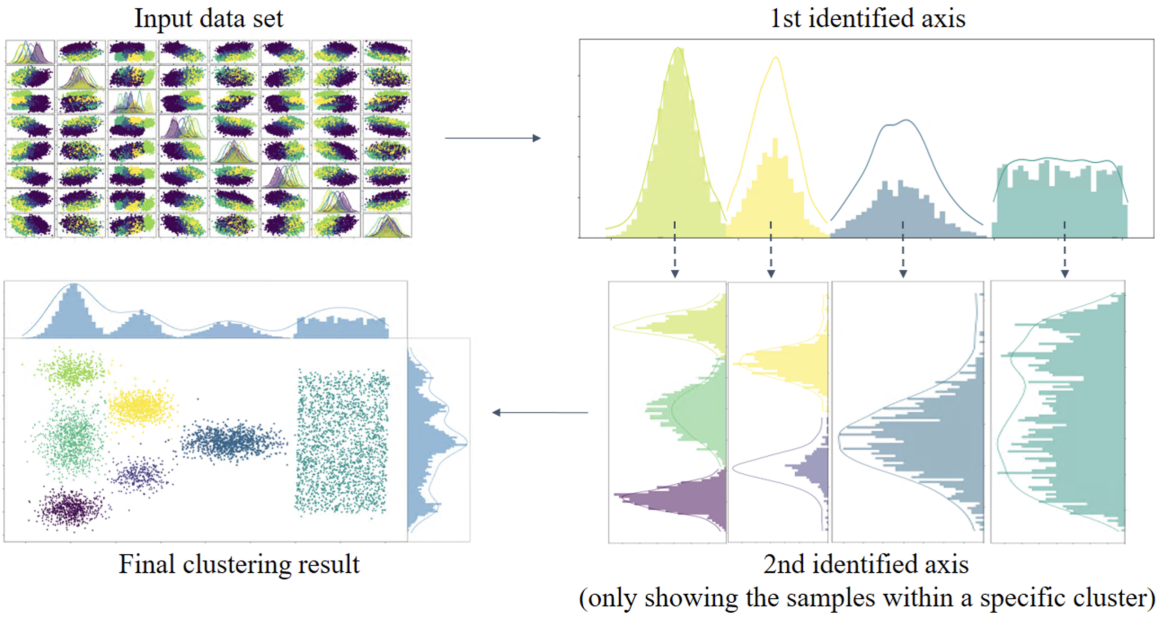


Figure 3.2: Process of Dip'n'Sub: *Detecting relevant structures in the input data set (scatter matrix plot - top left) is difficult. Dip'n'Sub performs SGD by using the gradient of Eq. 3.3 to identify a projection axis on which several modes and, thus, cluster-relevant structures are recognizable (histogram - top right). The subroutine TailoredDip then obtains the cluster assignments. The gradient of Eq. 3.3 is again used to find a second projection axis on which a maximum number of clusters exhibit multimodality (histograms - bottom right). As no further relevant projection axis can be identified, the final clustering result is returned (scatter plot - bottom left) containing clusters of various shapes. The first plot is colored according to the ground truth; the other three plots are colored using the labels identified by Dip'n'Sub.*

work satisfactorily on complex data sets. The main reason is that the optimization of the embedding and the estimation of the number of clusters run isolated from each other, which can lead to poor performance. If the initial embedding does not depict the desired patterns, the algorithms cannot recover from this error. This problem also applies to SCDE [DAMS19], one of the few clustering methods that uses a neural network to estimate the number of clusters. This algorithm first trains a regular AE and then uses the resulting embedding to determine the number of clusters by employing an additional softmax-based AE. Afterward, spectral clustering [vL07] with the estimated number of clusters is executed on the embedded data. Thus, the estimated number of clusters can not influence the final embedding. The process of first training an AE and then executing the actual clustering is also referred to as a multi-stage approach [ZXZ+22].

We propose the deep clustering algorithm DipDECK [LBS+21], which belongs to the iterative approaches [ZXZ+22], meaning that the embedding and the clustering are updated alternatingly. It starts with an overestimated number of so-called micro-clusters obtained by running k-Means on the embedding of a pretrained AE. During the optimiza-

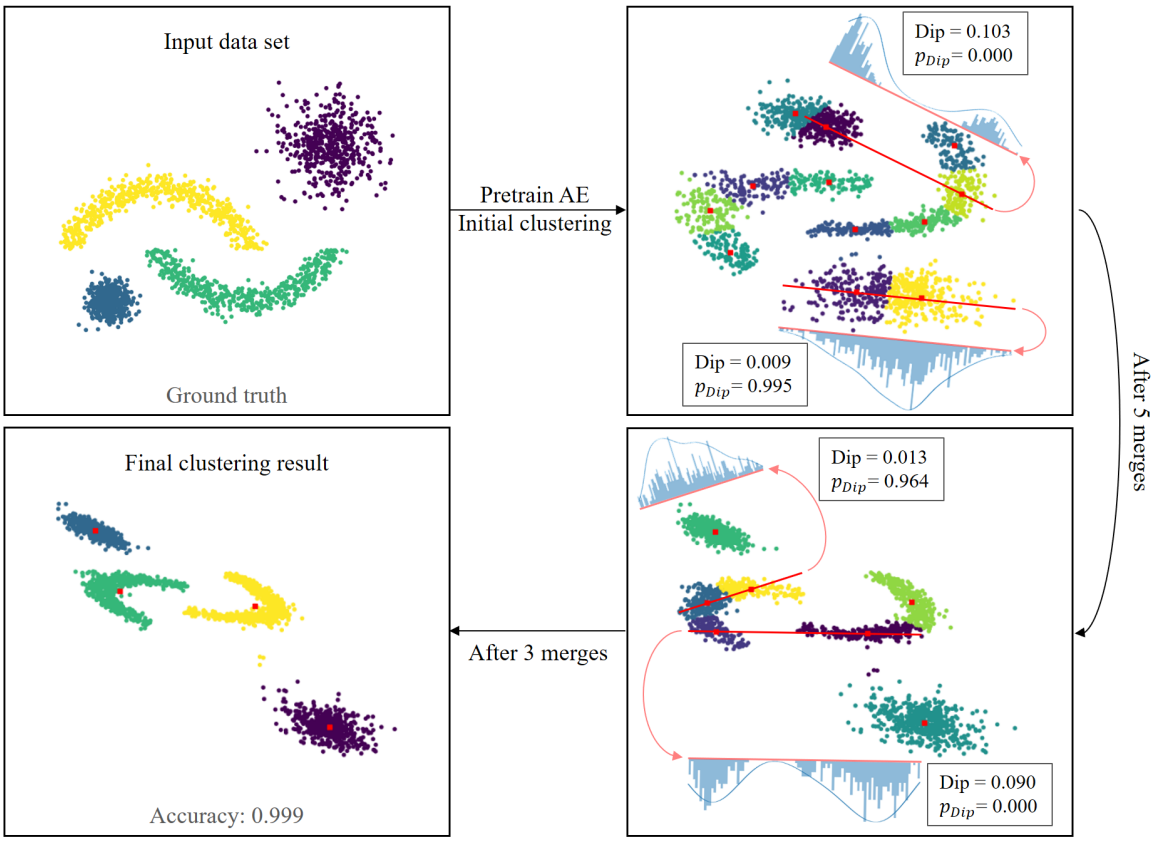


Figure 3.3: Process of DipDECK: *First, the input data set (top left) is used to pretrain the autoencoder. Afterward, the initial micro-clusters are obtained by running k-Means with an overestimated number of clusters on the embedded data set (top right). For each pair of micro-clusters, the Dip-value Dip and the corresponding probability value p_{Dip} are calculated by projecting the samples onto the connecting line between their cluster centers - the two red lines exemplify these projection axes, and the blue histograms visualize the projected data. Cluster pairs with high p_{Dip} are attracted until a merge occurs above a specified threshold. The third plot (bottom right) shows the embedding after five merges. The embedding is optimized until all clusters are clearly separated and no further merges occur (bottom left). The first plot is colored according to the ground truth; the other three plots are colored using the labels identified by DipDECK.*

tion process, DipDECK merges those micro-clusters likely to belong to a mutual structure. Here, the Dip-test measures the similarity between each pair of micro-clusters. We create the matrix $P \in \mathbb{R}^{k \times k}$, containing the probability values p_{Dip} of each pair of clusters projected onto the connecting line between their centers. Individual clusters receive a similarity value of 1, i.e., $\forall_{1 \leq j \leq k} : P_{j,j} = 1$. Finally, a k-Means-based loss function is used to optimize the embedding:

$$\mathcal{L}_{clust}(\mathcal{B}) = \frac{1 + \text{std}(D_C)}{\text{mean}(D_C)} \frac{1}{|\mathcal{B}|} \sum_{x_i \in \mathcal{B}} \sum_{j=1}^{k} \hat{P}_{\text{f}(x_i),j} ||\text{enc}(x_i) - \text{enc}(\mu_j)||_2^2, \tag{3.4}$$

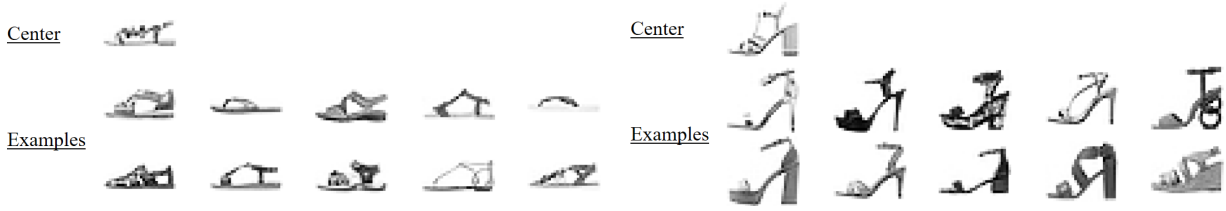
where D_C is the set of Euclidean distances between each pair of cluster centers, the function f(x_i) returns the cluster label of sample x_i, and $\hat{P}$ is a row-wise normalized version of P. The term mean(D_C) is included in the loss to prevent the AE from shrinking the scaling of the embedding to optimize $\mathcal{L}_{clust}$, and std(D_C) hinders the network from pushing a single easily separable cluster far away from the other clusters to increase mean(D_C). After updating the embedding for an epoch, the cluster labels and centers are updated in a k-Means-like fashion. These parameters are then used to update P. If the probability value of two clusters exceeds a specified threshold, a cluster merge is initiated and the AE can adapt to the new clustering structure in the following epoch.

The stated loss function uses the pair-wise probability values in P to strengthen the similarity and dissimilarity of existing structures. Strengthening relevant similarities can lead to subsequent merging operations. Thus, the clustering result and the embedding can influence each other, improving the final clustering result. Additionally, by combining several micro-clusters into one final cluster, DipDECK is more flexible regarding cluster shapes than comparison methods such as DEC [XGF16], IDEC [GGLY17], DCN [YFSH17], or DKM [FTG20]. This property can also be observed in Fig. 3.3, where DipDECK detects the correct number of clusters, even though the clusters differ in their shapes and extents.

By employing DipDECK, we can analyze complex data sets, e.g, Fashion-MNIST [XRV17], with little prior knowledge. In some cases, the number of identified clusters does not correspond to the number of classes contained in the ground truth. However, this is not necessarily a disadvantage but offers room for subsequent analyses, as interesting sub-structures may be identified. Note that "the task of clustering is to retrieve any natural grouping of the points in the data set, not necessarily the one encoded in the class label" [HD19]. It is reasonable that the algorithm returns unexpected solutions as it does not know contexts a human would consider common knowledge. Fig 3.4 shows an example regarding Fashion-MNIST, where DipDECK splits the ground truth class 'sandal' into the clusters 'flat sandal' (Fig. 3.4a) and 'high-heel sandal' (Fig. 3.4b) as they are comprehensibly different objects for a computer.

3.4 DipEncoder

While the gradient of the Dip-test has already been applied to perform linear transformations, e.g., in [KL05, MP16, SBBP20], it has yet to be considered for non-linear transformations. Studies are necessary to explore such potential optimization options. We introduce the DipEncoder [LBN+22], which combines the Dip-test's gradient with an AE to obtain an embedding in which all clusters are separated by adopting a multimodal structure. Simultaneously, each cluster must indicate a unimodal distribution to ensure no cluster is pulled apart. Since the Dip-test can only be applied to one-dimensional



(a) Cluster 'flat sandal'. *(b) Cluster 'high-heel sandal'.*

Figure 3.4: *The plots show two clusters of the Fashion-MNIST [XRV17] data set, as identified by DipDECK, which should be combined into a single cluster according to the ground truth. The cluster centers and ten samples from the clusters are illustrated.*

data, each combination of clusters is assigned an individual projection axis ρ. The loss function used to optimize the embedding looks as follows:

$$\mathcal{L}_{clust}(\mathcal{B}) = \frac{2}{k(k-1)} \sum_{a=1}^{k-1} \sum_{b=a+1}^{k} \frac{1}{2}(\text{dip}(\overline{Z}^{\mathcal{B}}_{a,\not b}) + \text{dip}(\overline{Z}^{\mathcal{B}}_{\not a,b})) - \text{dip}(\overline{Z}^{\mathcal{B}}_{a,b}), \tag{3.5}$$

where $\text{dip}(\overline{Z}^{\mathcal{B}}_{a,b})$ returns the Dip-value of all embedded samples of clusters a and b within batch $\mathcal{B}$ projected onto their corresponding projection axis $\rho_{a,b}$, i.e., $\overline{Z}^{\mathcal{B}}_{a,b} = \{\rho^T_{a,b} \cdot \text{enc}(x_i) \mid x_i \in \mathcal{B} \cap (\mathcal{C}_a \cup \mathcal{C}_b)\}$. This value should be maximized to ensure a highly multimodal structure when considering both clusters. The term $\text{dip}(\overline{Z}^{\mathcal{B}}_{a,\not b})$ returns the Dip-value of a subset of $\overline{Z}^{\mathcal{B}}_{a,b}$ by only considering the samples within cluster a, i.e., $\overline{Z}^{\mathcal{B}}_{a,\not b} = \{\rho^T_{a,b} \cdot \text{enc}(x_i) \mid x_i \in \mathcal{B} \cap \mathcal{C}_a\}$, and $\text{dip}(\overline{Z}^{\mathcal{B}}_{\not a,b})$ is defined analogously. Accordingly, these two values should be minimized to achieve a unimodal structure within the clusters.

When optimizing this loss function, not only the known gradient concerning the projection axis $\nabla_\rho \left(\text{dip}(\{\rho^T x_i \mid x_i \in \mathcal{D}\})\right)$ [KL05] is used but also the gradient regarding the data $\nabla_{x_i} \left(\text{dip}(\{\rho^T x_i \mid x_i \in \mathcal{D}\})\right)$. Since in traditional applications, the feature space is usually fixed, this gradient has so far remained unregarded. However, as the DipEncoder operates within an adaptable embedding, this gradient can provide the AE with beneficial information. By considering both gradients, the embedding can be improved in such a way that the positioning of the data and the projection axes are optimized.

The DipEncoder can be applied without making a precise statement about the within-cluster distribution of the data, which is in contrast to k-Means-based optimization functions, as used by DCN [YFSH17] and DKM [FTG20], and Kullback-Leibler-based optimization functions, as used by DEC [XGF16] and IDEC [GGLY17]. As a result, each cluster can have a distinct data distribution with an individual spread, adding flexibility to the resulting embedding.

As an iterative deep clustering approach, the clusters are adjusted after each epoch. Here, the cluster labels are updated by considering each pair of clusters and defining the

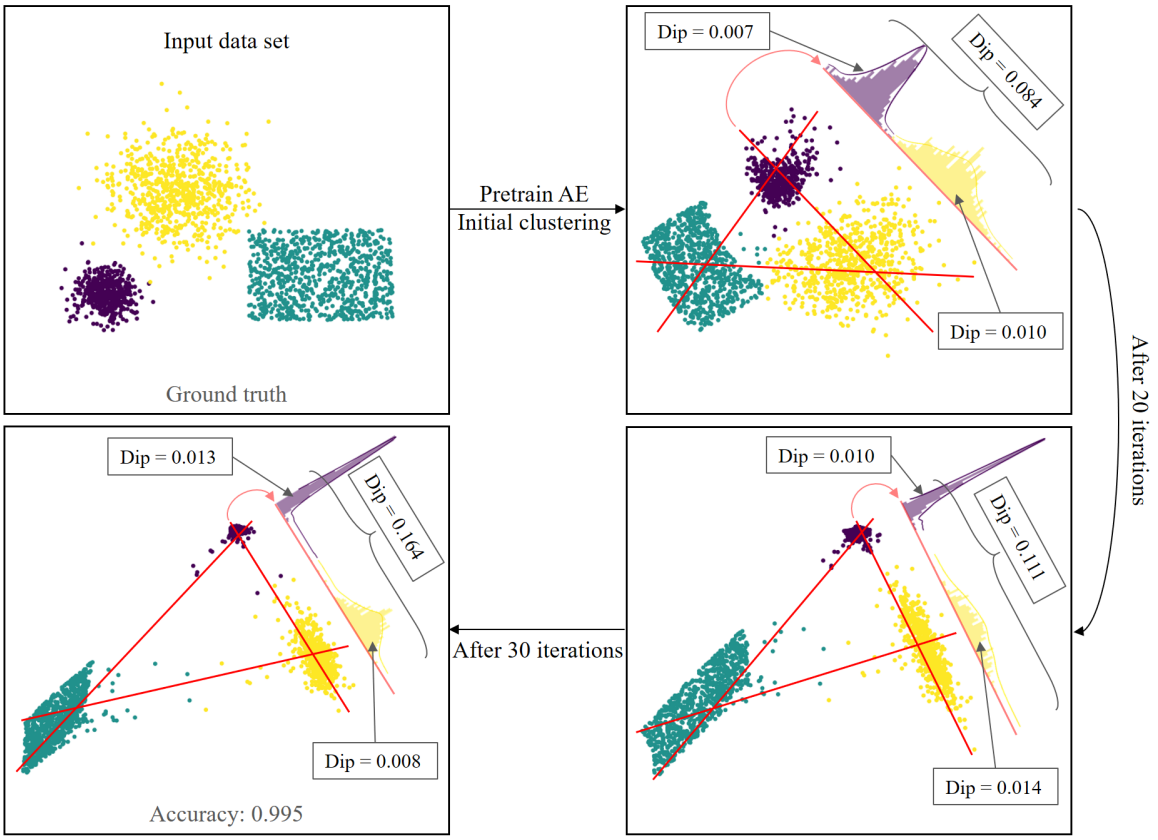


Figure 3.5: Process of the DipEncoder: *First, the input data set (top left) is used to pretrain the autoencoder. Afterward, the initial clusters are obtained by running k-Means on the embedded data set and updated by applying the described labels updating procedure of the DipEncoder (top right). The embedding is then updated by optimizing Eq. 3.5. The red line in the upper right shows the projection axis for the combination of the purple and yellow cluster, and the corresponding histogram visualizes its part of the loss term. After 20 iterations, the multimodality of the two clusters has increased from $Dip = 0.084$ to $Dip = 0.111$ (bottom right). After another 30 iterations, all clusters are clearly separated (bottom left). The first plot is colored according to the ground truth; the other three plots are colored using the labels identified by the DipEncoder.*

midpoint between the modes of the corresponding clusters on their common projection axis as the decision boundary. Thus, it is the first deep clustering algorithm whose clustering objective exclusively depends on the Dip-test and no other assumptions. k-Means is only required to obtain the initial cluster assignments, which can be directly updated using the strategy described.

Fig. 3.5 illustrates the optimization process of the DipEncoder. In this simple example, the algorithm is not only able to identify clusters of various shapes but also to preserve the original structures in the embedding. The DipEncoder has this ability because only modalities are taken into account, and the clusters are not forced into a predefined distribution.

3.5 The ClustPy Package for Benchmarking (Deep) Clustering Algorithms

All algorithms presented in this thesis are provided in our open-source Python package ClustPy[6]. Furthermore, ClustPy offers implementations of relevant comparison algorithms, methods for evaluating clustering techniques, and methods for loading certain benchmark data sets. Using ClustPy, we can compare clustering algorithms fairly and meaningfully, as the code base for all approaches is equivalent. This property is especially relevant for deep clustering algorithms as the implementations can vary heavily depending on the chosen deep learning framework.

In [LMPB23b], we use ClustPy to perform the first deep clustering benchmark using a unified framework. We compare the performances of the deep clustering algorithms DEC [XGF16], IDEC [GGLY17], DCN [YFSH17], ACe/DeC [MBM+21] and the DipEncoder [LBN+22] on various image data sets. The benchmark focuses on procedures that are comparable in their optimization goal, enabling us to make meaningful statements about their performances in different settings. Here, the influence of different AE architectures, hyperparameters, and augmentations is analyzed.

Some key findings are:

- In many scenarios, the commonly used AE architecture, as described in [XGF16], can be substituted with a simpler architecture without significantly influencing the clustering results.
- 5 and 10 are often good choices when setting the size of the AE's embedding m.
- The learning rate should not be set too large when optimizing a deep clustering loss, as this can negatively influence the final clustering result.
- As mentioned in Sec. 2.4.5, deep clustering algorithms require an initial clustering result, usually obtained by running k-Means. In our experiments, we notice that in the case of the MNIST data set [LBBH98], the EM algorithm can be a better choice, leading to a higher clustering accuracy. This finding is potentially relevant for other data sets as well.
- In general, deep clustering algorithms benefit notably from applying image augmentation. This confirms the results of other studies, e.g., [MPB19, MBM+21].
- The DipEncoder shows good overall performance. Especially with a simple AE architecture, the results are superior to those of most comparison algorithms. This

[6] `https://github.com/collinleiber/ClustPy` (accessed 2024/01/10)

behavior confirms our hypothesis that an analysis of the modes present in the embedding of an AE can lead to more flexible and high-quality clustering results.

In summary, we show that ClustPy offers a user-friendly way to evaluate novel clustering methods fairly. To this end, a tutorial with best practices on deep clustering using ClustPy was developed and presented at the 32nd ACM International Conference on Information and Knowledge Management (CIKM 2023) [LMPB23a].

4 Conclusion

In the following conclusion, we want to address some topics related to our contributions. First, we would like to discuss limitations to consider when working with the presented methods. Afterward, we exhibit which new research opportunities arise from these methods. Finally, we conclude the thesis with some last remarks.

4.1 Limitations

While the presented methods solve problems of existing approaches, they also suffer from certain limitations. For example, methods based on the Dip-test generally struggle to correctly detect modes when the number of points per cluster varies substantially. If one cluster is significantly larger than another, the Dip-test will always indicate a unimodal distribution. Accordingly, the clusters cannot be correctly distinguished from each other in an unsupervised manner. A Dip-test-based separation of the clusters would require a downsampling of the larger cluster. Such a strategy is discussed in [LBS+21]. Another challenge is the application of the Dip-test to non-continuous data. In this case, each expression of an attribute can be recognized as a separate mode, which can lead to undesirable effects during the clustering process. DipDECK [LBS+21] and the DipEncoder [LBN+22] avoid this problem by performing the Dip-test only on the embedded data, which typically consists of continuous values.

The Dip-test is not the only procedure that has difficulties when analyzing data sets with unbalanced cluster sizes. It is a general problem in deep learning applications since not the entire data set is used, but only random batches [ZXZ+22]. Thus, in the case of strongly varying cluster sizes, individual clusters may be underrepresented or even entirely missing in a batch, which, in turn, can heavily influence the final clustering result. This is also an issue when dealing with a large number of clusters. If the batch size is too small, there is a high probability that specific clusters are not included in a batch.

Another limitation is the parameterization of DipDECK [LBS+21] and the DipEncoder [LBN+22]. Although the presented methods omit some parameters that are difficult to choose, many hyperparameters are still necessary for all deep learning-based applications. For example, to train a neural network, a suitable architecture, optimizer, learning rate, and an optional learning rate scheduler must be chosen. In addition, the weight λ_1

for the reconstruction loss $\mathcal{L}_{rec}$ and λ_2 for the clustering loss $\mathcal{L}_{clust}$ are required for most deep clustering methods. These values are not trivially optimizable in unsupervised learning since - in contrast to supervised learning - no appropriate metric is available for evaluating a model. Due to the powerful non-linear transformation of the feature space, unsupervised metrics, such as the *silhouette score* [Rou87], which compares the within-cluster distances with the distances to other clusters, cannot be applied in a meaningful way. This results in a situation where it becomes difficult for domain experts to evaluate a clustering result. For this reason, the cluster centers within the original representation are often considered to analyze the identified clustering structure.

4.2 Future Work

The proposed approaches offer several opportunities for further research projects. We want to present some ideas briefly.

Combining AutoNR with methods other than NrKmeans: Currently, the framework described in [LMPB22] has been applied to NrKmeans [MYPB18] only. However, it is possible to transfer this to other non-redundant clustering algorithms, such as Orth [CFD07], without major changes. It would be interesting to see how potential procedures would perform and whether the findings would differ from those of AutoNR. In addition, the MDL encoding strategy could also be applied to common subspace methods such as SubKmeans [MYPB17], ADR-EM [DHZS02], or LDA-k-Means [DL07]. In this case, input parameters for the number of clusters and the dimensionality of the subspace would no longer be necessary, which would greatly simplify the application.

Parameter-free common subspace clustering on mixed-type data: AutoNR can be further extended by including the handling of mixed-type data sets. Currently, AutoNR is only able to process numeric values. However, in [KLB23], we demonstrate how common subspace clustering can work for mixed-type data sets. Since the numerical optimization of the proposed k-SubMix algorithm is based on SubKmeans [MYPB17], it is possible to extend our MDL encoding by a categorical part, which would allow an utterly parameter-free execution.

Combining the gradient regarding p_{Dip} with deep learning: In [BLBP23], we introduce a differentiable function to transform Dip-values to probability values. Further, we show how this function can be utilized to calculate a gradient regarding p_{Dip}. This gradient can be integrated into deep learning applications. For example, instead of maximizing the Dip-values between clusters, the DipEncoder [LBN+22] could also minimize the probability values. This process could result in an intrinsic weighting of the gradients, leading to improved clustering results.

Combining DipDECK with the clustering loss of the DipEncoder: Another research opportunity would be the combination of DipDECK [LBS+21] and the DipEncoder

[LBN+22]. Currently, the clusters in DipDECK are optimized using a k-Means-based loss function. The Dip-values between the clusters are only used for the merging operation and serve as a weighting of the Euclidean distances. It could be beneficial to integrate the Dip-test deeper into the procedure by using a clustering loss similar to that of the DipEncoder. An appropriate optimization could result in an increased flexibility of the embedding since modalities are evaluated instead of distances.

Substituting the Dip-test: Another possibility for further research would be to replace the Dip-test with more modern modality tests. For example, the *Folding Test of Unimodality* [SFTL18] or the *UU-Test* [CL22] could be used for this purpose. It would be interesting to check whether a gradient concerning these tests can be formulated, which could be integrated into appropriate subspace or deep learning applications. By substituting the Dip-test, the final clustering results can be compared with each other in order to discuss which test is best suited in which scenarios.

4.3 Closing Remarks

This thesis presents four methods that combine complex transformation-based clustering objectives, such as common subspace clustering, non-redundant clustering, or deep clustering, with simplified parameterizations compared to competitor algorithms. Here, different techniques are used to analyze the distinct modes in the resulting feature spaces. We want to emphasize that while the Dip-test has already been applied in various areas of machine learning, e.g., in [LZD+18] for clustering streaming data or in [CL17] for image segmentation, DipDECK and the DipEncoder are - to the best of our knowledge - the first algorithms that combine the Dip-test with deep learning. We hope that the presented innovations in the field of data mining will result in new research opportunities and that new, exciting insights can be gained. Furthermore, automatically determining the number of clusters will hopefully lead to opportunities to extract knowledge from previously unexamined data sources.

In order to accelerate the research progress, we further introduce the Python package ClustPy. This package provides a good basis for comparing clustering procedures and creating insightful benchmarks.

References

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 49–60. ACM Press, 1999.

[AD52] Theodore W Anderson and Donald A Darling. Asymptotic theory of certain" goodness of fit" criteria based on stochastic processes. *The annals of mathematical statistics*, pages 193–212, 1952.

[AD07] Amir Ahmad and Lipika Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data Knowl. Eng.*, 63(2):503–527, 2007.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

[Aka74] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[Ass12] Ira Assent. Clustering high dimensional data. *WIREs Data Mining Knowl. Discov.*, 2(4):340–350, 2012.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035. SIAM, 2007.

[Bal87] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987*, pages 279–284. Morgan Kaufmann, 1987.

[BB94] Léon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 585–592. MIT Press, 1994.

[Bel66] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[Bez81] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Springer, 1981.

[BFP08] Christian Böhm, Christos Faloutsos, and Claudia Plant. Outlier-robust clustering using independent components. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 185–198. ACM, 2008.

[BFPP06] Christian Böhm, Christos Faloutsos, Jia-Yu Pan, and Claudia Plant. Robust information-theoretic clustering. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 65–75. ACM, 2006.

[BLBP23] Lena G. M. Bauer, Collin Leiber, Christian Böhm, and Claudia Plant. Extension of the dip-test repertoire - efficient and differentiable p-value calculation for clustering. In *Proceedings of the 2023 SIAM International Conference on Data Mining, SDM 2023, Minneapolis-St. Paul Twin Cities, MN, USA, April 27-29, 2023*, pages 109–117. SIAM, 2023.

[BLS99] Horst Bischof, Ales Leonardis, and Alexander Selb. MDL principle for robust vector quantisation. *Pattern Anal. Appl.*, 2(1):59–72, 1999.

[BM01] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 245–250. ACM, 2001.

[CAS19] Pantelis Chronis, Spiros Athanasiou, and Spiros Skiadopoulos. Automatic clustering by detecting significant density dips in multiple dimensions. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 91–100. IEEE, 2019.

[CBJD18] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September*

8-14, 2018, Proceedings, Part XIV, volume 11218 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2018.

[CFD07] Ying Cui, Xiaoli Z. Fern, and Jennifer G. Dy. Non-redundant multi-view clustering via orthogonalization. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 133–142. IEEE Computer Society, 2007.

[CG13] Sanjay Chawla and Aristides Gionis. k-means-: A unified approach to clustering and outlier detection. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA*, pages 189–197. SIAM, 2013.

[CG15] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: survey, insights, and generalizations. *J. Mach. Learn. Res.*, 16:2859–2900, 2015.

[Che95] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, 1995.

[CKV13] M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst. Appl.*, 40(1):200–210, 2013.

[CL17] Theofilos Chamalis and Aristidis Likas. Region merging for image segmentation based on unimodality tests. In *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 381–384. IEEE, 2017.

[CL18] Theofilos Chamalis and Aristidis Likas. The projected dip-means clustering algorithm. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN 2018, Patras, Greece, July 09-12, 2018*, pages 14:1–14:7. ACM, 2018.

[CL22] Paraskevi Chasani and Aristidis Likas. The uu-test for statistical modeling of unimodal data. *Pattern Recognit.*, 122:108272, 2022.

[DAMS19] Liang Duan, Charu C. Aggarwal, Shuai Ma, and Saket Sathe. Improving spectral clustering with deep embedding and cluster estimation. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 170–179. IEEE, 2019.

[Das00] Sanjoy Dasgupta. Experiments with random projection. In *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford*

University, Stanford, California, USA, June 30 - July 3, 2000, pages 143–151. Morgan Kaufmann, 2000.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[DHD+17] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5747–5756. IEEE Computer Society, 2017.

[DHS11] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[DHZS02] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, and Horst D. Simon. Adaptive dimension reduction for clustering high dimensional data. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 147–154. IEEE Computer Society, 2002.

[DJ76] Richard C. Dubes and Anil K. Jain. Clustering techniques: The user's dilemma. *Pattern Recognit.*, 8(4):247–260, 1976.

[DL07] Chris H. Q. Ding and Tao Li. Adaptive dimension reduction using discriminant analysis and *K*-means clustering. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 521–528. ACM, 2007.

[DLJ10] Chris H. Q. Ding, Tao Li, and Michael I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):45–55, 2010.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–22, 1977.

[DUN74] JC DUNN. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *J. of Cybernetics*, 3:32–57, 1974.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[FH06] Yu Feng and Greg Hamerly. Pg-means: learning the number of clusters in data. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 393–400. MIT Press, 2006.

[Fis38] Ronald A Fisher. The statistical utilization of multiple measurements. *Annals of eugenics*, 8(4):376–386, 1938.

[FTG20] Maziar Moradi Fard, Thibaut Thonet, and Éric Gaussier. Deep *k*-means: Jointly clustering with *k*-means and learning representations. *Pattern Recognit. Lett.*, 138:185–192, 2020.

[GBSS05] Arthur Gretton, Olivier Bousquet, Alexander J. Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic Learning Theory, 16th International Conference, ALT 2005, Singapore, October 8-11, 2005, Proceedings*, volume 3734 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2005.

[GGLY17] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1753–1759. ijcai.org, 2017.

[GHPB14] Sebastian Goebl, Xiao He, Claudia Plant, and Christian Böhm. Finding the optimal subspace for clustering. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 130–139. IEEE Computer Society, 2014.

[GLZY17] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *Neural Information Processing - 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017,*

Proceedings, Part II, volume 10635 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2017.

[GN17] Guojun Gan and Michael Kwok-Po Ng. k-means clustering with outlier removal. *Pattern Recognit. Lett.*, 90:8–14, 2017.

[GPM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[GRHS04] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 513–520, 2004.

[Grü05] Peter Grünwald. Minimum description length tutorial. *Advances in minimum description length: Theory and applications*, 5:1–80, 2005.

[HCK+05] Ville Hautamäki, Svetlana Cherednichenko, Ismo Kärkkäinen, Tomi Kinnunen, and Pasi Fränti. Improving k-means by outlier removal. In *Image Analysis, 14th Scandinavian Conference, SCIA 2005, Joensuu, Finland, June 19-22, 2005, Proceedings*, volume 3540 of *Lecture Notes in Computer Science*, pages 978–987. Springer, 2005.

[HD19] Sibylle Hess and Wouter Duivesteijn. k is the magic number - inferring the number of clusters through nonparametric concentration inequalities. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*, volume 11906 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2019.

[HE03] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pages 281–288. MIT Press, 2003.

[HH85] John A Hartigan and Pamela M Hartigan. The dip test of unimodality. *The annals of Statistics*, pages 70–84, 1985.

[HH00] Aapo Hyvärinen and Patrik O. Hoyer. Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural Comput.*, 12(7):1705–1720, 2000.

[Hua97] Zhexue Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings Of 1st Pacific-Asia Conference on Knowledge Discouvery And Data Mining*, 1997.

[Ish05] Tsunenori Ishioka. An expansion of x-means for automatically determining the optimal number of clusters aˆeur" progressive iterations of k-means and merging of the clusters. In *IASTED International Conference on Computational Intelligence, Calgary, Alberta, Canada, July 4-6, 2005*, pages 91–96. IASTED/ACTA Press, 2005.

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

[JH91] Christian Jutten and Jeanny Hérault. Blind separation of sources, part I: an adaptive algorithm based on neuromimetic architecture. *Signal Process.*, 24(1):1–10, 1991.

[JMD08] Prateek Jain, Raghu Meka, and Inderjit S. Dhillon. Simultaneous unsupervised learning of disparate clusterings. *Stat. Anal. Data Min.*, 1(3):195–210, 2008.

[JZT+17] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1965–1972. ijcai.org, 2017.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KKMC07] Ivan O. Kyrgyzov, Olexiy O. Kyrgyzov, Henri Maître, and Marine Campedel. Kernel MDL to determine the number of clusters. In *Machine Learning and Data Mining in Pattern Recognition, 5th International Conference, MLDM 2007, Leipzig, Germany, July 18-20, 2007, Proceedings*, volume 4571 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2007.

[KKZ12] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Subspace clustering. *WIREs Data Mining Knowl. Discov.*, 2(4):351–364, 2012.

[KL05] Andreas Krause and Volkmar Liebscher. Multimodal projection pursuit using the dip statistic. 2005.

[KL12] Argyris Kalogeratos and Aristidis Likas. Dip-means: an incremental clustering method for estimating the number of clusters. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2402–2410, 2012.

[KLB23] Mauritius Klein, Collin Leiber, and Christian Böhm. k-submix: Common subspace clustering on mixed-type data. In *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part I*, volume 14169 of *Lecture Notes in Computer Science*, pages 662–677. Springer, 2023.

[KMN+17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[LBD+89] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989.

[LBN+22] Collin Leiber, Lena G. M. Bauer, Michael Neumayr, Claudia Plant, and Christian Böhm. The dipencoder: Enforcing multimodality in autoencoders. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 846–856. ACM, 2022.

[LBS+21] Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. Dip-based deep embedded clustering with k-estimation. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 903–913. ACM, 2021.

[LCW+18] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Comput. Surv.*, 50(6):94:1–94:45, 2018.

[Lee01] Thomas CM Lee. An introduction to coding theory and the two-part minimum description length principle. *International statistical review*, 69(2):169–183, 2001.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[LLZ15] Xv Lan, Qian Li, and Yi Zheng. Density k-means: A new algorithm for centers initialization for k-means. In *2015 6th IEEE international conference on software engineering and service science (ICSESS)*, pages 958–961. IEEE, 2015.

[LMPB22] Collin Leiber, Dominik Mautz, Claudia Plant, and Christian Böhm. Automatic parameter selection for non-redundant clustering. In *Proceedings of the 2022 SIAM International Conference on Data Mining, SDM 2022, Alexandria, VA, USA, April 28-30, 2022*, pages 226–234. SIAM, 2022.

[LMPB23a] Collin Leiber, Lukas Miklautz, Claudia Plant, and Christian Böhm. Application of deep clustering algorithms. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, pages 5208–5211. ACM, 2023.

[LMPB23b] Collin Leiber, Lukas Miklautz, Claudia Plant, and Christian Böhm. Benchmarking deep clustering algorithms with clustpy. In *IEEE International Conference on Data Mining, ICDM 2023 - Workshops, Shanghai, China, December 4, 2023*, pages 625–632. IEEE, 2023.

[LVV03] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. The global k-means clustering algorithm. *Pattern Recognit.*, 36(2):451–461, 2003.

[LZD+18] Yonghong Luo, Ying Zhang, Xiaoke Ding, Xiangrui Cai, Chunyao Song, and Xiaojie Yuan. Strdip: A fast data stream clustering algorithm using the dip

test of unimodality. In *Web Information Systems Engineering - WISE 2018 - 19th International Conference, Dubai, United Arab Emirates, November 12-15, 2018, Proceedings, Part II*, volume 11234 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2018.

[MALK19] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan: Latent space clustering in generative adversarial networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4610–4617. AAAI Press, 2019.

[MBD+12] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[MBM+21] Lukas Miklautz, Lena G. M. Bauer, Dominik Mautz, Sebastian Tschiatschek, Christian Böhm, and Claudia Plant. Details (don't) matter: Isolating cluster information in deep embedded spaces. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2826–2832. ijcai.org, 2021.

[MH18] Leland McInnes and John Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018.

[MJ51] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[MMA+20] Lukas Miklautz, Dominik Mautz, Muzaffer Can Altinigneli, Christian Böhm, and Claudia Plant. Deep embedded non-redundant clustering. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5174–5181. AAAI Press, 2020.

[MP16] Samuel Maurus and Claudia Plant. Skinny-dip: Clustering in a sea of noise. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1055–1064. ACM, 2016.

[MPB19] Dominik Mautz, Claudia Plant, and Christian Böhm. Deep embedded cluster tree. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 1258–1263. IEEE, 2019.

[MSL+23] Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch, Christian Böhm, and Claudia Plant. Non-redundant image clustering of early medieval glass beads. In *10th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2023, Thessaloniki, Greece, October 9-13, 2023*, pages 1–12. IEEE, 2023.

[MYPB17] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Towards an optimal subspace for k-means. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 365–373. ACM, 2017.

[MYPB18] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Discovering non-redundant k-means clusterings in optimal subspaces. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1973–1982. ACM, 2018.

[MYPB20] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Non-redundant subspace clusterings with nr-kmeans and nr-dipmeans. *ACM Trans. Knowl. Discov. Data*, 14(5):55:1–55:24, 2020.

[Nai82] John Naisbitt. Megatrends. *New York*, 17, 1982.

[NDJ10] Donglin Niu, Jennifer G. Dy, and Michael I. Jordan. Multiple non-redundant spectral clustering views. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 831–838. Omnipress, 2010.

[NE10] Xuan Vinh Nguyen and Julien Epps. mincentropy: A novel information theoretic approach for the generation of alternative clusterings. In *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 521–530. IEEE Computer Society, 2010.

[NH19] Lan H. Nguyen and Susan P. Holmes. Ten quick tips for effective dimensionality reduction. *PLoS Comput. Biol.*, 15(6), 2019.

[NWL19] Feiping Nie, Cheng-Long Wang, and Xuelong 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pages 959–967. ACM, 2019.

[Par62] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[Par94] Robert L Parker. *Geophysical inverse theory*, volume 1. Princeton university press, 1994.

[Pea01] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor.*, 6(1):90–105, 2004.

[PM00] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 727–734. Morgan Kaufmann, 2000.

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Autom.*, 14(5):465–471, 1978.

[Ris83] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, 11(2):416–431, 1983.

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

[RMR21] Mohd Soufhwee Bin Abd Rahman, Effendi Bin Mohamad, and Azrul Azwan Bin Abdul Rahman. Development of iot - enabled data analytics enhance decision support system for lean manufacturing process improvement. *Concurr. Eng. Res. Appl.*, 29(3), 2021.

[Rou87] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[RPY+22] Yazhou Ren, Jingyu Pu, Zhimeng Yang, Jie Xu, Guofeng Li, Xiaorong Pu, Philip S. Yu, and Lifang He. Deep clustering: A comprehensive survey. *CoRR*, abs/2210.04142, 2022.

[SBBP20] Benjamin Schelling, Lena Greta Marie Bauer, Sahar Behzadi, and Claudia Plant. Utilizing structure-rich features to improve clustering. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*, volume 12457 of *Lecture Notes in Computer Science*, pages 91–107. Springer, 2020.

[SCF08] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learning*, volume 1. Vancouver, CA:, 2008.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.

[Sch23] Erich Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead. *SIGKDD Explor.*, 25(1):36–42, 2023.

[Scu10] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 1177–1178. ACM, 2010.

[SFTL18] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. Are your data gathered? In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 2210–2218. ACM, 2018.

[SGZC13] Kelvin Sim, Vivekanand Gopalkrishnan, Arthur Zimek, and Gao Cong. A survey on enhanced subspace clustering. *Data Min. Knowl. Discov.*, 26(2):332–397, 2013.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[SKK07] Yaser Sheikh, Erum A. Khan, and Takeo Kanade. Mode-seeking by medoidshifts. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8. IEEE Computer Society, 2007.

[SP18] Benjamin Schelling and Claudia Plant. Diptransformation: Enhancing the structure of a dataset and thereby improving clustering. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 407–416. IEEE Computer Society, 2018.

[Tho53] Robert L Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[TSL00] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.

[VS08] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part IV*, volume 5305 of *Lecture Notes in Computer Science*, pages 705–718. Springer, 2008.

[WK06] Max Welling and Kenichi Kurihara. Bayesian k-means as a "maximization-expectation" algorithm. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 474–478. SIAM, 2006.

[WKQ+08] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael S. Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008.

[Wol66] Herman Wold. Estimation of principal components and related models by iterative least squares. *Multivariate analysis*, pages 391–420, 1966.

[WWD+19] Xing Wang, Jun Wang, Carlotta Domeniconi, Guoxian Yu, Guoqiang Xiao, and Maozu Guo. Multiple independent subspace clusterings. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5353–5360. AAAI Press, 2019.

[XDLL15] Qin Xu, Chris H. Q. Ding, Jinpei Liu, and Bin Luo. Pca-guided search for k-means. *Pattern Recognit. Lett.*, 54:50–55, 2015.

[XGF16] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 478–487. JMLR.org, 2016.

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

[XT15] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2:165–193, 2015.

[YFSH17] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3861–3870. PMLR, 2017.

[YMHP16] Wei Ye, Samuel Maurus, Nina C. Hubig, and Claudia Plant. Generalized independent subspace clustering. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 569–578. IEEE Computer Society, 2016.

[YPB16] Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5147–5156. IEEE Computer Society, 2016.

[ZXZ+22] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao Li, Jiajun Bu, Jia Wu, Xin Wang, Wenwu Zhu, and Martin Ester. A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions. *CoRR*, abs/2206.07579, 2022.

[ZZuH+23] Yiming Zhu, Peixian Zhang, Ehsan ul Haq, Pan Hui, and Gareth Tyson. Can chatgpt reproduce human-generated labels? A study of social computing tasks. *CoRR*, abs/2304.10145, 2023.

List of Figures

List of Tables

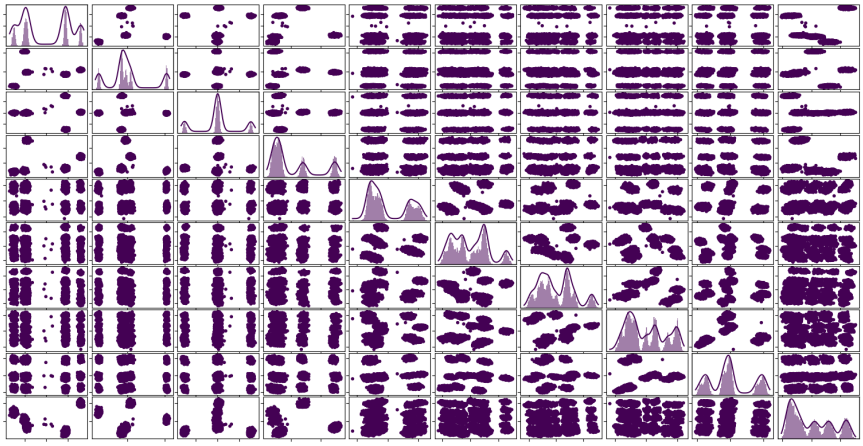
Appendix

A Process of AutoNR on the NrLetters data set

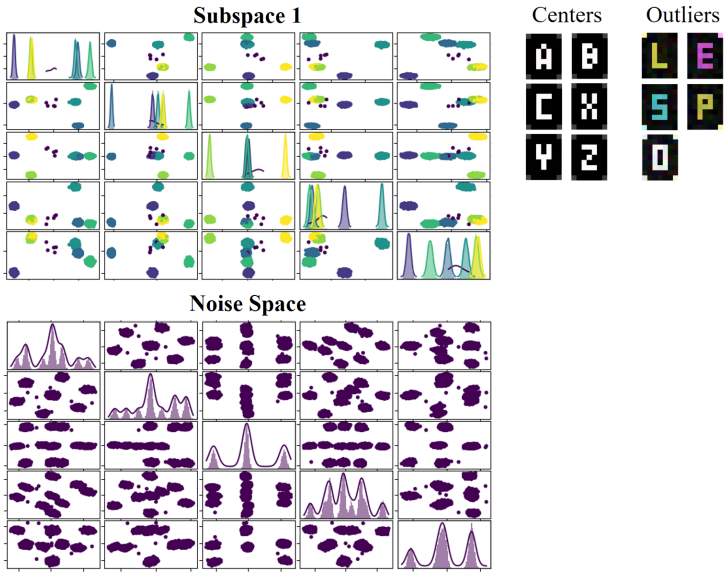
To illustrate the process of AutoNR, we run the algorithm on the NrLetters data set [LMPB22]. The expected clusterings describe the shown letter ('A', 'B', 'C', 'X', 'Y', 'Z'), the marked corner (top left, top right, bottom left, bottom right), and the color of the letter (pink, cyan, yellow). We further add eleven outliers to the data set. These show an incorrect letter ('E', 'L', 'O', 'P', 'S'), several marked corners, or an incorrect color (red, green, blue, white). All conditions apply to the outlier that illustrates a white 'O'. The final data set consists of $10{,}011$ 7×9 colored images.

In this example, we apply AutoNR on a dimensionally reduced version of the data set by utilizing PCA while keeping 90% of the data's variance. The clustering process is shown in Fig. A.1, where the main intermediate results are illustrated.

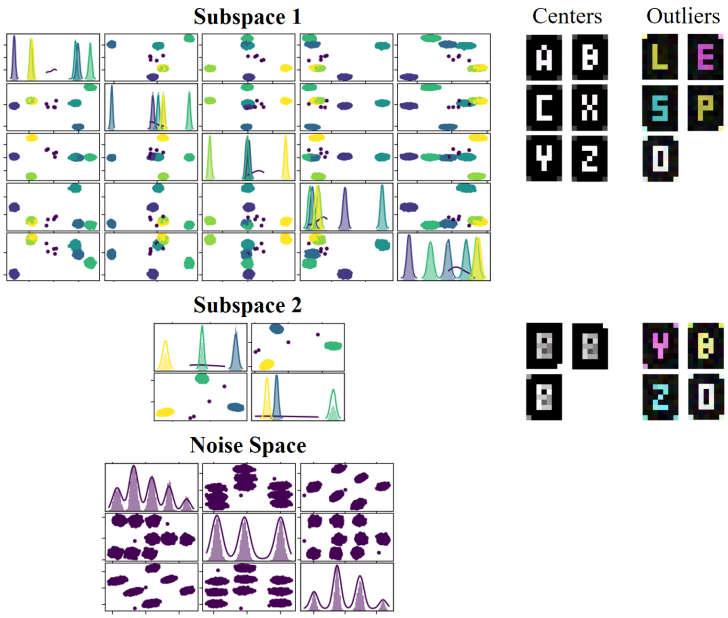
First, the input data set (Fig. A.1a) is split into two subspaces, the first containing six clusters describing the six letters and the second a single cluster (Fig. A.1b). We refer to subspaces with a single cluster as noise spaces. Next, the noise space is split into a subspace with three clusters and a new noise space (Fig. A.1c). Here, the newly created subspace contains information on whether the marked corner is on the left, top right, or bottom right. Therefore, AutoNR is not yet able to distinguish all four corners. Afterward, the noise space is split into a subspace containing three clusters and a new noise space (Fig. A.1d). These three clusters perfectly capture the color information of the images. Finally, AutoNR identifies another clustering with three clusters containing the missing information regarding the marked corner (Fig. A.1e). AutoNR further improves the clustering result by merging the second and the fourth subspaces (Fig. A.1f). The resulting subspace contains one cluster for each corner of the images. This indicates that AutoNR has the ability to 'repair' undesirable clustering results. In summary, AutoNR successfully identifies all clustering possibilities contained in the data set. Further, in all subspaces, the outliers are perfectly recognized.



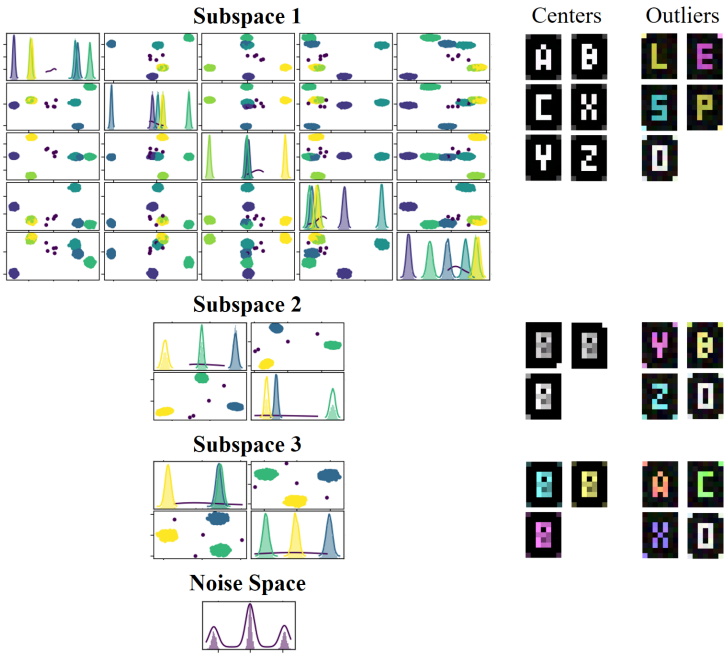
(a) The input data set.



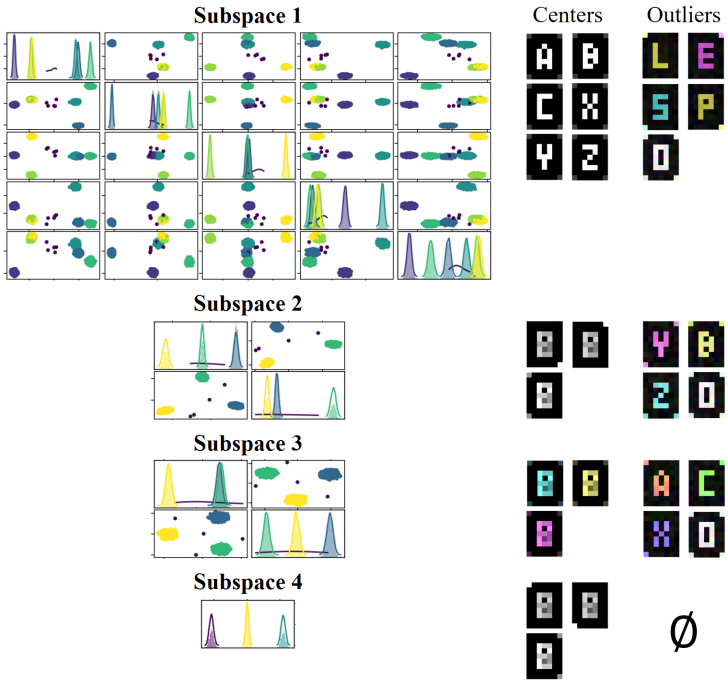
(b) Result after the first subspace split. AutoNR identifies the information regarding the shown letter.



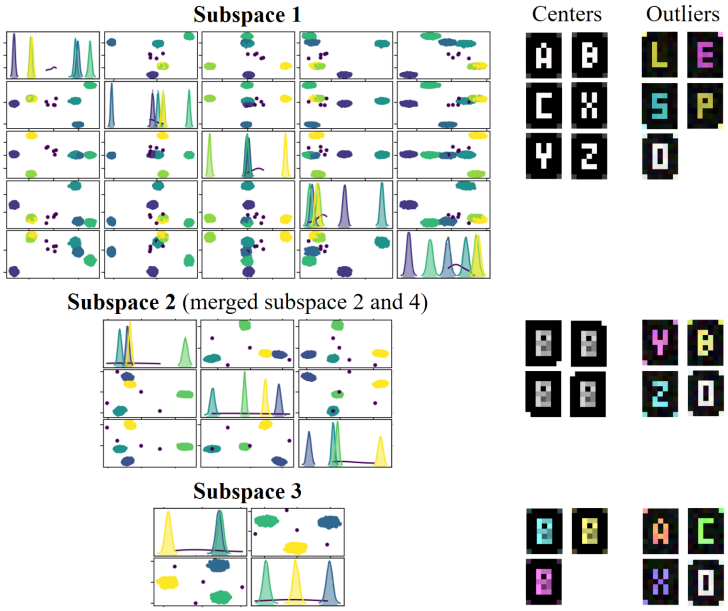
(c) Result after the second subspace split. AutoNR only partially identifies the information regarding the marked corner.



(d) Result after the third subspace split. AutoNR identifies the information regarding the color of the letters.



(e) Result after the fourth subspace split. AutoNR identifies the remaining information regarding the marked corner.



(f) Final result after merging subspaces 2 and 4. AutoNR combines the information regarding the marked corner contained in the two subspaces.

Figure A.1: Process of AutoNR regarding the NrLetters data set: *The visualizations illustrate the identified subspaces through scatter matrix plots and the cluster centers and outliers within those subspaces. After four subspace splits* ***(b)*** *-* ***(e)*** *and one merging of subspaces* ***(f)****, AutoNR perfectly captures the clustering information contained in the NrLetters data set. Furthermore, AutoNR identifies all outliers in the subspaces. The coloring of the scatters corresponds to the labels in each subspace.*

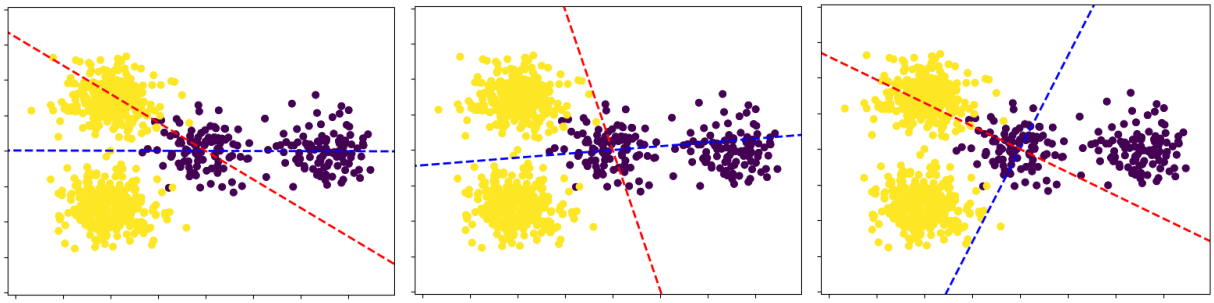
B Gradient regarding p_{Dip}

To examine the effect of our gradient regarding p_{Dip}, we consider the data set shown in Fig. B.2. We can see two clusters with 200 (purple) and 1000 (yellow) samples. Both clusters consist of two separate Gaussian distributions. It is easy to define a proper projection axis to split each cluster individually. This axis would be horizontal for the purple cluster and vertical for the yellow cluster. However, to obtain fewer features in the resulting subspace, we prefer to find a single projection axis that can be used to split both clusters. To evaluate the gradient of our transformation function, we analyze the difference between minimizing S_p, as defined in Eq. 3.3, and maximizing the score S_D:

$$S_D = \frac{1}{N} \sum_{j=1}^{k} |\mathcal{C}_j| \text{Dip}(\overline{\mathcal{C}_j^\rho}), \tag{B.1}$$

where $\text{Dip}(\overline{\mathcal{C}_j^\rho})$ returns the Dip-value Dip of $\overline{\mathcal{C}_j^\rho}$. The only difference is that Eq. 3.3 optimizes p_{Dip} while Eq. B.1 optimizes Dip. For the optimization of the projection axes, we use SGD with a learning rate of 0.01 and momentum of 0.95.

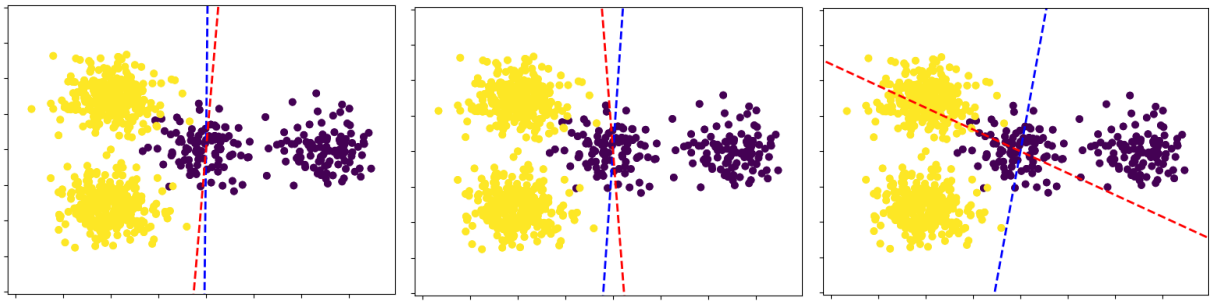
The results shown in Fig. B.2 indicate that using the gradient of Eq. 3.3 leads to a better, i.e., lower, S_p in all nine cases, and the gradient of Eq. B.1 leads to a better, i.e., higher, value of S_D in six cases. It can be concluded that both gradients are capable of improving their respective objective functions. However, Eq. 3.3 is better suited to the clustering task of Dip'n'Sub [BLBP23] since we want to identify multimodal axes regarding multiple clusters, which may vary heavily in size.



(a) *Horizontal starting axis - Iteration* 1.
S_D*-grad:* $S_D = 0.0309$, $S_p = 0.3461$
S_p*-grad:* $S_D = 0.0311$, $S_p = 0.1444$

(b) *Horizontal starting axis - Iteration* 10.
S_D*-grad:* $S_D = 0.0285$, $S_p = 0.7161$
S_p*-grad:* $S_D = 0.0762$, $S_p = 0.2459$

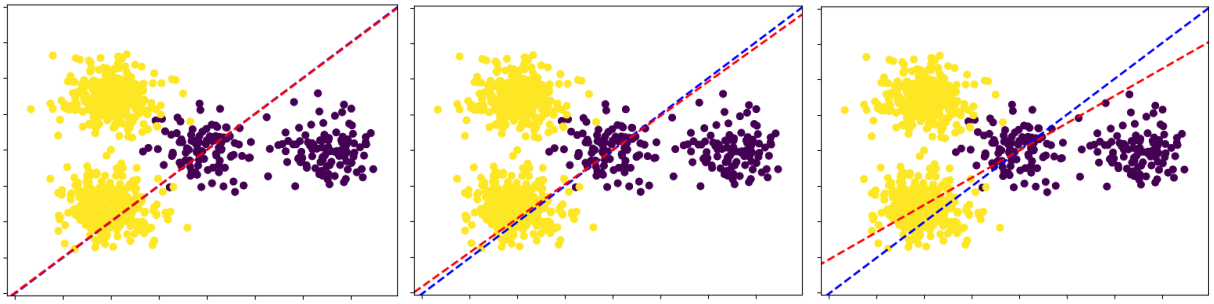
(c) *Horizontal starting axis - Iteration* 100.
S_D*-grad:* $S_D = 0.0788$, $S_p = 0.2476$
S_p*-grad:* $S_D = 0.0395$, $S_p = 0.0001$



(d) *Vertical starting axis - Iteration* 1.
S_D*-grad:* $S_D = 0.0804$, $S_p = 0.2299$
S_p*-grad:* $S_D = 0.0818$, $S_p = 0.1211$

(e) *Vertical starting axis - Iteration* 10.
S_D*-grad:* $S_D = 0.0808$, $S_p = 0.2249$
S_p*-grad:* $S_D = 0.0791$, $S_p = 0.2024$

(f) *Vertical starting axis - Iteration* 100.
S_D*-grad:* $S_D = 0.0826$, $S_p = 0.0581$
S_p*-grad:* $S_D = 0.0379$, $S_p = 0.0007$



(g) *Diagonal starting axis - Iteration* 1.
S_D*-grad:* $S_D = 0.0646$, $S_p = 0.0009$
S_p*-grad:* $S_D = 0.0646$, $S_p = 0.0008$

(h) *Diagonal starting axis - Iteration* 10.
S_D*-grad:* $S_D = 0.0647$, $S_p = 0.0012$
S_p*-grad:* $S_D = 0.0636$, $S_p = 0.0005$

(i) *Diagonal starting axis - Iteration* 100.
S_D*-grad:* $S_D = 0.0648$, $S_p = 0.0011$
S_p*-grad:* $S_D = 0.0569$, $S_p = 0.0000$

Figure B.2: *Results of SGD using the gradient regarding Eq. 3.3 (S_p-grad) to update the red projection axis and the gradient regarding Eq. B.1 (S_D-grad) to update the blue projection axis after* 1 *(left),* 10 *(middle), and* 100 *(right) iterations. For each scenario, we start with a horizontal (first row), a vertical (second row), and a diagonal (third row) projection axis. Values indicating a higher multimodality with respect to Eq. B.1 or Eq. 3.3 are underlined.*

C Appended Papers

The following section contains the contributed publications and, in the case of [LMPB22] and [BLBP23], their corresponding supplements. Further, we summarize additional information about the publications, such as the title, authors, abstract, the venue where it was published, the *Digital Object Identifier* (DOI), and the division of work among the authors.

C.1 Automatic Parameter Selection for Non-Redundant Clustering

Authors:
Leiber, Collin and Mautz, Dominik and Plant, Claudia and Böhm, Christian

Abstract:
High-dimensional datasets often contain multiple meaningful clusterings in different subspaces. For example, objects can be clustered either by color, weight, or size, revealing different interpretations of the given dataset. A variety of approaches are able to identify such non-redundant clusterings. However, most of these methods require the user to specify the expected number of subspaces and clusters for each subspace. Stating these values is a non-trivial problem and usually requires detailed knowledge of the input dataset. In this paper, we propose a framework that utilizes the Minimum Description Length Principle (MDL) to detect the number of subspaces and clusters per subspace automatically. We describe an efficient procedure that greedily searches the parameter space by splitting and merging subspaces and clusters within subspaces. Additionally, an encoding strategy is introduced that allows us to detect outliers in each subspace. Extensive experiments show that our approach is highly competitive to state-of-the-art methods.

Venue:
Proceedings of the 2022 SIAM International Conference on Data Mining (SDM). Society for Industrial and Applied Mathematics, 2022.
CORE ranking: A (http://portal.core.edu.au/conf-ranks/1727/)
Acceptance rate: 26.5%

DOI:
`https://doi.org/10.1137/1.9781611977172.26`

Thesis-Reference:
[LMPB22]

Division of Work:

Collin Leiber
Formulation of the MDL encoding strategy; Formulation of the main heuristic to identify the number of clusterings and clusters per clustering; Implementing the algorithm; Designing the experiments; Executing the experiments; Writing the majority of the paper; Creating the visualizations; Creating the presentation slides; Presenting at the conference

Dominik Mautz
Initial idea to combine MDL and non-redundant clustering; Regular discussions on the procedure; Discussions on relevant related work; Frequent suggestions for improving the presentation

Claudia Plant
Identification of relevant related work; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Christian Böhm
Providing information regarding MDL and writing of associated parts of the related work section; Discussions to improve the MDL encoding strategy; Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Automatic Parameter Selection for Non-Redundant Clustering

Collin Leiber* Dominik Mautz* Claudia Plant† Christian Böhm*

Abstract

High-dimensional datasets often contain multiple meaningful clusterings in different subspaces. For example, objects can be clustered either by color, weight, or size, revealing different interpretations of the given dataset. A variety of approaches are able to identify such non-redundant clusterings. However, most of these methods require the user to specify the expected number of subspaces and clusters for each subspace. Stating these values is a non-trivial problem and usually requires detailed knowledge of the input dataset. In this paper, we propose a framework that utilizes the Minimum Description Length Principle (MDL) to detect the number of subspaces and clusters per subspace automatically. We describe an efficient procedure that greedily searches the parameter space by splitting and merging subspaces and clusters within subspaces. Additionally, an encoding strategy is introduced that allows us to detect outliers in each subspace. Extensive experiments show that our approach is highly competitive to state-of-the-art methods.

1 Introduction

Several algorithms have been developed in order to cluster objects under certain conditions. Typically, these techniques deliver a single solution or multiple solutions in a hierarchical setup. However, often various distinct clusterings can be created by different characteristics of the given dataset. Take Figure 1 as an example. These low-resolution images can be grouped by the shown letter, the color, or the marked corner. This indicates that there are three different lower-dimensional subspaces, each with a unique clustering. Further, these partitionings are mutually non-redundant, for each object belongs to different clusters in different subspaces [16].

This diversity of clustering possibilities emerges especially from datasets in high-dimensional spaces. Non-redundant clustering algorithms can reveal coherences within the data, which would not be visible by a single clustering. Since each clustering has its individual subspace, the analysis of the results is facilitated by the lower dimensionality and usually also by to the lower number of clusters. In addition, the user can choose the best partitioning based on his specific objective without being limited to a single option. However, a problem with most available methods is that the user needs a lot of prior knowledge to set suitable input parameters.

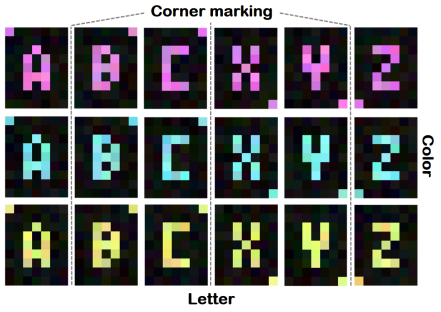


Figure 1: Images of the letters 'A', 'B', 'C', 'X', 'Y' and 'Z' in the colors pink, cyan, and yellow. In each image, one corner is highlighted in color. This results in three different clusterings with 6, 3, and 4 clusters.

In this paper, we describe a framework that utilizes the Minimum Description Length (MDL) to overcome the problem of specifying those parameters. MDL is used to solve model selection problems and for example is often applied in Boolean Matrix Factorization [18]. It is based on the idea that any regularity in the data can be used to compress the data [8]. The fewer bits are required to describe a model, the better the model is.

Our framework performs a sophisticated search in the parameter space to find a setting with which a high compression of the input dataset is achieved. Thus, it can automatically determine the number of subspaces and clusters per subspace. Existing parameter-free approaches (e.g. [24, 26]) first identify all subspaces and then the clusters within them. In contrast, we successively split and merge subspaces while simultaneously searching for the best number of partitions within these new subspaces. This gives us more flexibility in finding meaningful structures. Additionally, we introduce a feature that can identify outliers in each of the found subspaces. This feature is integrated into our MDL-based cost function and does not require additional input parameters. To illustrate the effectiveness of our framework we created the *AutoNR* algorithm which combines our proposal with the recently presented *Nr-Kmeans*

*LMU Munich. {leiber, mautz, boehm}@dbs.ifi.lmu.de
†Faculty of Computer Science, ds:UniVie, University of Vienna, Vienna, Austria. claudia.plant@univie.ac.at

[16] approach. Our major contributions are:

- We present a general MDL encoding for non-redundant clustering models that can be combined with centroid-based clustering algorithms.
- A parameter-free, greedy approach efficiently searches the parameter space to identify the number of subspaces and clusters within subspaces.
- Our procedure is extended to identify outliers in each existing subspace.

2 Related Work

We want to briefly review three relevant areas of research: non-redundant clustering, methods for determining the number of clusters and basic outlier detection extensions for Gaussian clustering approaches.

Non-Redundant Clustering: We only consider methods that can find non-redundant clusterings without knowing an initial clustering solution. Alternative clustering algorithms would require an additional parameter-free clustering approach that provides the input partitioning in a parameter-free setting.

Cui et al. [4] identify orthogonal clusterings by sequentially executing PCA and *k-means*. They propose two strategies: *orthogonal clustering (Orth1)* and *clustering in orthogonal subspaces (Orth2)*. *mSC* [19] creates non-redundant partitionings by combining spectral clustering with the Hilbert-Schmidt Independence Criterion to penalize similarity between different subspaces. *Nr-Kmeans* [16] is a generalization of the subspace clustering algorithm *SubKmeans* [15]. It assumes that a rotated feature space can be split into J axis-parallel subspaces, each containing an individual *k-means* clustering. An advantage of *Nr-Kmeans* is that it optimizes all subspaces and clusters in the subspaces simultaneously. *NrDipmeans* [17] is an extension of this method. Here, Hartigan's dip test [10] is used to detect multimodal structures in each subspace and thereby determine the number of clusters. The five algorithms mentioned so far all require the number of subspaces and, except for *NrDipmeans*, the number of clusters per subspace as input parameters. *ISAAC* [26] utilizes Independent Subspace Analysis (ISA) and MDL to find multiple subspaces in a parameter-free setting. It fits Gaussian Mixture Models (GMMs) within those statistically independent subspaces. The parameter-free algorithm *MISC* [24] also uses ISA and MDL to identify independent subspaces. Afterward, it performs kernel graph regularized semi-nonnegative matrix factorization to define clusters in each subspace.

Determine the Number of Clusters: A popular strategy to identify the number of clusters is the *X-means* algorithm [20]. It uses the Bayesian Information Criterion to rate different parametrizations of *k-means*. Starting with a small number of clusters, it repeatedly splits them up by creating two new centers along a randomly chosen vector. Ishioka et al. [13] extend this approach by introducing the idea of merging clusters. The *G-means* [9] algorithm exploits the Anderson-Darling statistical hypothesis test to check whether the objects within a cluster follow a Gaussian distribution. The approach starts with a single cluster and iteratively increases the number of clusters by splitting those whose data appears non-Gaussian. *PG-means* [5] projects the data and the learned model into a one-dimensional space and applies the Kolmogorov-Smirnov test to check if the projected model fits the projected data. This test is executed repeatedly with different projections. If any test fails, a new cluster will be added, and the approach repeats. Bischof et al. [2] use MDL as a pruning criterion by starting with a complex model and gradually reducing the number of clusters until optimal encoding costs are reached. Additionally, they use MDL to identify outliers in each iteration of their procedure. *FOSSCLU* [7] is a subspace clustering algorithm optimizing an orthonormal transformation that projects the data into a lower-dimensional subspace in which the *EM algorithm* is executed. The dimensionality of the subspace and the number of clusters are identified through MDL.

Most aboved-mentioned methods cannot be applied easily to non-redundant clustering, since the subspaces are constantly adapting to changing cluster structures. This particularly affects the search heuristics.

Outlier Detection: The identification of outliers usually requires additional input parameters. For example, *ODIN* [12] first creates a k-nearest neighbor graph and then counts the number of edges leading to a point. If this amount is smaller than a threshold, the point is an outlier. *ORC* [11] calculates the outlyingness of all points as their distance to their corresponding cluster centers divided by the maximum distance. If the outlyingness of a point is greater than a threshold, it will be removed from the dataset. *k-means--* [3] regards the l points farthest away from their corresponding centers as outliers in each iteration of *k-means*. These l points will not be used for the subsequent assignment and update step. *ODC* [1] and *KMOR* [6] both calculate the average distance d_{avg} between the objects and the cluster centers. If an object is at least $y \cdot d_{avg}$ away from all centers, with y being a parameter set by the user, it will be regarded as an outlier. One difference is that *KMOR* allows outliers to become inliers again.

For our parameter-free approach, it is important that no additional parameter for the outlier identification is introduced. Therefore, we integrate the outlier definition directly into our MDL-based cost function.

3 Parameter-free Non-Redundant Clustering

While algorithms like *Orth* [4], *mSC* [19], and *Nr-Kmeans* [16] work well in defining non-redundant clusterings, they still require extensive user knowledge of the input dataset to define the correct number of subspaces J and clusters per subspace k_j.

In the following we want to use MDL to solve this situation. All symbols used in this work are summarized and described in the supplement.

Usually, the subspace of a non-redundant clustering algorithm can be defined using an orthogonal transformation matrix $V \in \mathbb{R}^{d\times d}$ that rotates the given d-dimensional feature space and a projection matrix $P_j \in \mathbb{N}^{d\times m_j}$ that specifies its corresponding m_j dimensions after rotation. P_j has a 1 in its i-th row if the i-th dimension is contained in subspace j. All other values are 0. In addition, the sum of each column must be equal to 1. Consequently, in subspace j the input dataset $X \subseteq \mathbb{R}^d$ of size N changes to

$$X_j = \{xVP_j | x \in X\}, \tag{3.1}$$

where $X_j \subseteq \mathbb{R}^{m_j}$. The actual clustering then takes place in these subspaces. This is a common setting for many subspace and non-redundant clustering algorithms, like those presented in [4, 7, 15, 16, 19, 24, 26].

3.1 MDL Encoding First, we describe our MDL encoding strategy for non-redundant clusterings. In general, MDL tries to minimize $L(H)+L(D|H)$. $L(H)$ indicates the number of bits necessary to define the hypothesis, which in our case equals the rotation, subspace, and cluster parameters. $L(D|H)$ is the code length needed to encode the data under this hypothesis [8].

We can encode the natural numbers J, m_j, and k_j with the universal prior for integers, $L^0(n)$ [21].

$$L^0(n) = \log^*(n) + \log_2(c),$$

where $\log^*(n) = \log_2(n) + \log^*(\log_2(n))$ only involves the positive terms and $c \simeq 2.865064$.

The cluster centers in subspace j are generally represented by a vector consisting of m_j real values, each corresponding to one coordinate of the rotated feature space. To encode these values, we utilize a uniform distribution and apply Shannon-Fano coding [23]. By using the uniform distribution, no position in the feature space is favored. To implement this, we first calculate the maximum distance between all objects. In combination with the minimum coordinate of each rotated feature, we can form a hypercube that contains all points with certainty, regardless of the final rotation. The resulting probability density function (pdf) looks as follows:

$$\pi_{\text{uniform}}(r) = \frac{1}{\max(\text{dist}_X)},$$

where dist_X is the set of Euclidean distances between all objects, $\text{dist}_X = \{\sqrt{||x-y||_2^2} \mid x \in X \text{ and } y \in X\}$.

In order to correctly apply Shannon-Fano codes, we need to transform the result of a pdf to an actual probability. This can approximately be done by using a precision δ [14]. We assume that each object origins from a creation process that itself operates with a finite precision. Note, that the creation process of each feature can be different; therefore the precision can vary greatly. Since the multiplication by V creates a new space in which all features are mixed to some degree, we approximate δ by using the mean of the minimum feature-wise distances (ignoring zero values). The total code length is a combination of the pdf and δ [14].

$$L(r) = -\log_2(\pi(r)) - \log_2(\delta)$$

We use this to get the coding costs of cluster center $\mu_{j,i}$ in subspace j.

$$L(\mu_{j,i}) = -m_j \left(\log_2(\pi_{\text{uniform}}(\mu_{j,i})) + \log_2(\delta)\right) \tag{3.2}$$

To encode the cluster assignments, we assume that the probability of belonging to a particular cluster in subspace j is the same for all clusters, and therefore equals $\frac{1}{k_j}$. This way, we can again utilize Shannon–Fano coding and receive a code length of $-\log_2(\frac{1}{k_j})N$ [23].

Note that the costs of encoding N, d, V, and the features of the hypercube are always the same, regardless of which model we are analyzing. Therefore, these constant costs can be ignored.

The objects in a dataset can be encoded by choosing a suitable distribution function matching the clustering procedure. This could, for example, be a Gaussian Mixture Model (GMM) for *EM*-like algorithms. It is important that the distribution fits the underlying clustering algorithm, so that our parameter-search procedure does not work against that algorithm by trying to optimize something different. Once a suitable pdf (π_{obj}) has been identified, the coding costs of the objects in subspace j are as follows:

$$L(X_j) = \sum_{x\in X_j} \left(-\log_2(\pi_{\text{obj}}(x)) - m_j \log_2(\delta)\right) \tag{3.3}$$

Finally, the distribution-specific parameters have to be encoded. For a GMM, for example, these would be the $k_j d(d+1)/2$ values for the symmetric covariance matrices $\Sigma_{j,i}$ of the clusters in subspace j. Let's assume we have p_j such parameters in subspace j. Since we normally use a maximum likelihood estimator to calculate those values, the code length can be approximated by $\frac{p_j}{2}\log_2(N)$ [22, 14]. We use N because it is a well-known value that usually is sufficiently large.

In summary, the costs required to encode a complete non-redundant clustering model in bits are:

- number of subspaces: $L^0(J)$
- for each subspace j:
  - dimensionality: $L^0(m_j)$
  - number of clusters: $L^0(k_j)$
  - cluster centers: $\sum_{i=1}^{k_j} L(\mu_{j,i})$ (3.2)
  - cluster assignments: $-\log_2(\frac{1}{k_j})N$
  - objects: $L(X_j)$ (3.3)
  - distribution parameters: $\frac{p_j}{2}\log_2(N)$

3.2 Determine the Subspace and Cluster Count

We use this encoding strategy to develop a greedy procedure that identifies the number of subspaces and clusters within each subspace in a non-redundant clustering setting. For this, multiple models have to be computed and evaluated by their MDL costs. Theoretically, the number of subspaces can range from 1 to d, and the number of clusters within each subspace from 1 to N. This results in an extremely large search space.

Our proposed strategy repeatedly splits the feature space into smaller subspaces until the MDL costs cannot be lowered anymore. Within those subspaces, we search for the best amount of clusters. Since subspaces are examined separately, the execution time drops significantly due to the lower dimensionality of the analyzed space. We differentiate between an optional subspace with a single cluster, called *noise space*, and subspaces with more than one cluster, called *cluster spaces*. The pseudo-code of the process is shown in the supplement.

We start with a random V and a single *noise space*. Each iteration begins by sorting the *cluster spaces* from the best full-space result so far in descending order by their MDL costs. An optional *noise space* is added last. The ordering is used because costly subspaces offer the greatest potential for cost reduction through successive splitting operations. Only the corresponding subspace j is used for these operations. This is done by changing the input from X to the lower-dimensional X_j as defined in Eq. (3.1). Depending on the type of subspace, we execute a *noise* or *cluster space* split.

3.2.1 Noise Space Split In case of a *noise space* split, the first run is performed with a subspace containing two clusters and a new lower-dimensional *noise space*. In the following iterations, the number of clusters within the *cluster space* is constantly raised by one. Here, the projections P_j and the rotation matrix V from the last result are used as input for the next iteration. We also take the cluster with the largest dispersion and divide it into two by creating two new centers. For cluster i in subspace j, they are calculated as follows:

$$\mu_{j,\text{new}_{1\ /\ 2}} = \mu_{j,i} \pm \frac{\text{diag}(\Sigma_{j,i})}{m_j|C_{j,i}|},$$

where $|C_{j,i}|$ is the size of the cluster and $\Sigma_{j,i}$ is its covariance matrix. This allows us to use the centers as input too.

In addition to the execution with reused parameters, random parameterizations are applied to identify previously not found structures in the *noise space*. In the following, we consider only the outcome that produces the lowest MDL cost. If the newly created *noise space* does not change in two consecutive iterations, we can stop using random parameters. In this case, we assume that a good structure in the *cluster space* has been identified, but the number of clusters is yet uncertain. The *noise space* splitting procedure repeats until the MDL costs increase.

3.2.2 Cluster Space Split For *cluster spaces*, we restrict the search space of possible splits. We assume that no arising subspace may have more clusters than the original space and that there cannot be more clusters in the original space than the number of possible combinations of clusters in the two resulting spaces. This gives the following rules:

$$\max(k_{\text{split}_1}, k_{\text{split}_2}) \leq k_{\text{original}} \leq k_{\text{split}_1} \cdot k_{\text{split}_2} \quad (3.4)$$

In the beginning, both subspaces have a cluster count equal to the original space. Subsequently, we merge the two nearest cluster centers in both subspaces by replacing them with their mean. Additionally, the projections P_j and rotation matrix V are reused as inputs for the following iteration. This approach repeats until Eq. (3.4) is violated. The idea is that the MDL costs reach a local minimum if the subspace with the higher number of clusters is correctly identified. Hereafter, we fix k_j of the subspace that is more responsible for the low costs and successively reduce the number of clusters of the other subspace. This is repeated until Eq. (3.4) is violated, or the MDL costs increase. In the end, we output the result that produced the lowest overall costs.

3.2.3 Full-space Execution When the subspace splitting procedure converges, we check whether the sum of the costs of the two new subspaces is smaller than the costs of the original space. If this is not the case, we start to split the next of the sorted subspaces or, if there is no subspace left, we start the merging operation. This cost-control mechanism avoids unnecessary expensive clustering executions in the full-space. After a positive check, a full-space clustering execution, which reuses the parameters of the last full-space result, is performed. Here, the parameters of the original subspace are replaced with the ones from the two subspaces found by the splitting procedure. The cluster

centers can be brought to full dimensionality by using the cluster labels to calculate the mean of the assigned objects. The rotation matrix of the new subspaces can be converted into the full-space by using the following transformation formula [16]:

$$V_{\text{sub}}^{F}[a,b] = \begin{cases} V_{\text{sub}}[n,m], & \text{if } P_{\text{sub}} \text{ maps } a \text{ to } n \text{ and } b \text{ to } m \\ 1, & \text{if } a = b \text{ and not the first case} \\ 0, & \text{otherwise} \end{cases}$$

We update the rotation matrix V by calculating $V_{\text{new}} = V_{\text{old}} V_{\text{sub}}^{F}$. A single full-space execution is sufficient because all input parameters are known.

Some non-redundant clustering methods are not able to optimize all subspaces simultaneously. In this case, the full-space clustering execution can be skipped, and the model costs can be calculated directly using the combined parameters.

If the full-space solution has lower MDL costs than the best result found so far, a new iteration starts with the ordering of the subspaces. Otherwise, the result is discarded, and the next subspace is considered for splitting, or if no subspace is left, the merging process starts.

3.2.4 Cluster Space Merge The merging step is executed for each possible combination of *cluster spaces*. Again, the input dataset is transformed using Eq. (3.1) with the combined projection matrices. Since we analyze the merged space individually, no projection or rotation matrices are needed. In the first iteration, we divide the dataset into $k_{\text{original}_1} \cdot k_{\text{original}_2}$ clusters. We use all combinations of centers of the two subspaces as the input set of centers. In each iteration, the number of clusters decreases by one. For this purpose, the nearest centers are merged by replacing them with their mean. The procedure stops if either the MDL costs rise or the cluster count gets smaller than the maximum number of clusters of the original subspaces. This equals the reversed rules of the *cluster space* split from Eq. (3.4). If the MDL costs of the merged space are lower than the sum of the costs of the original subspaces, we perform a full-space execution as described above.

If merging led to a better result, the merging operation repeats with the newly defined subspaces. Otherwise, our approach terminates.

An illustration of the described procedure is shown in Figure 2. It visualizes the full-space results of our exemplary algorithm *AutoNR* (see Section 4) on a sample dataset. *AutoNR* first runs a *noise space* split on the input dataset and finds a subspace with four clusters. Next, it performs a *cluster space* split and splits the subspace into two subspaces with two clusters

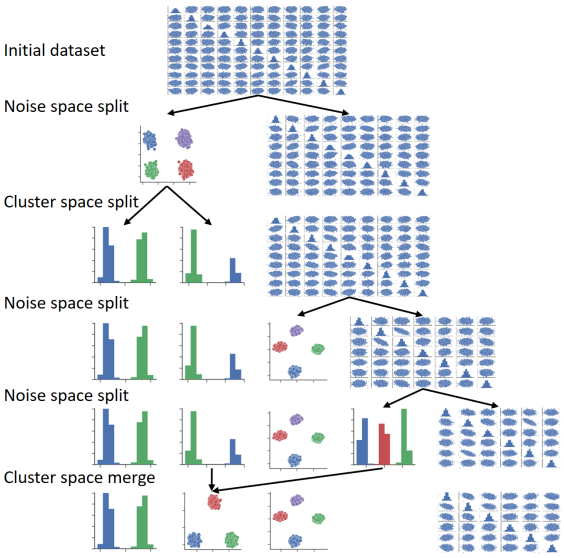


Figure 2: Example execution of *AutoNR* on a sample dataset ($d = 11$). The arrows indicate which subspaces are affected by an operation. After two *noise space* splits, a *cluster space* split and a *cluster space* merge, three *cluster spaces* are identified with four, three, and two clusters.

each. Then another *noise space* split is performed, and a *cluster space* with four clusters is found. The following *noise space* split provides a subspace with three clusters. Last, this is merged with one of the subspaces with two clusters, resulting in three *cluster spaces* with four, three, and two clusters.

3.3 Outlier Detection Many clustering approaches suffer from outliers. For this reason, we introduce an optional outlier detection by using the ability to rate clustering results by their MDL costs. In the case of non-redundant clusterings, it is crucial to re-determine the outliers with each update of the cluster structure because the definition can change significantly due to an update of the rotation matrix or the projections.

The cluster assignment and center update steps can be executed in the usual way. Afterward, we check for each point if it is cheaper to encode it using the distribution function of the corresponding cluster or separately as an outlier. The following must hold for each outlier o_j in subspace j:

$$L(X_j \setminus o_j) + L_{\text{total}}(o_j) < L(X_j).$$

We can encode a single outlier o_j in subspace j the same way we encode the cluster centers in Section 3.1.

$$L(o_j) = -m_j \left(\log_2(\pi_{\text{uniform}}(o_j)) + \log_2(\delta)\right) \tag{3.5}$$

In addition, we must consider that although we save one cluster assignment (requiring $-\log_2(\frac{1}{k_j})$ bits), we must specify the index of the specific outlier. This requires $\log_2(N)$ bits.

Alternatively, one could also specify for each point whether it is an outlier or not (requiring N bits) or interpret the outliers as an additional cluster and therefore encode the cluster assignments using $k_j + 1$ groups. However, since it can be assumed that there are significantly fewer outliers than inliers, this would greatly increase the MDL costs.

Overall, the following costs decide whether or not it is worth interpreting a point as an outlier:

$$L_{\text{total}}(o_j) = L(o_j) + \log_2(N) + \log_2(\frac{1}{k_j}).$$

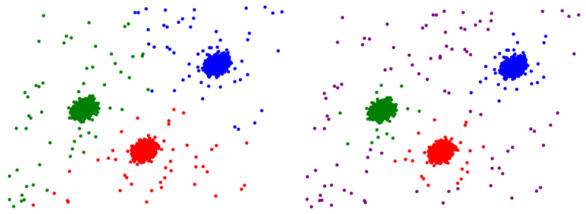
After we identified all the outliers O_j in subspace j, we update the cluster components (e.g. centers and covariance matrices). Furthermore, we must slightly adapt the MDL costs of our clustering model. For each subspace j, we need to state the number of outliers, requiring $L^0(|O_j|)$ bits. Next, we must use $|O_j| \log_2(N)$ bits to encode the indices, and Eq. (3.5) has to be added $|O_j|$ times to encode the actual points. In return, only $N - |O_j|$ points have to be regarded for the cluster assignments. The differences compared to the encoding summarized in Section 3.1 are for each subspace:

- number of outliers: $L^0(|O_j|)$
- cluster assignments: $-\log_2(\frac{1}{k_j})(N - |O_j|)$
- indices of the outliers: $|O_j| \log_2(N)$
- outliers: $\sum_{o_j \in O_j} L(o_j)$ (3.5)
- objects: $L(X_j \setminus O_j)$ (3.3)

This outlier detection procedure is user-friendly since no additional input parameters are necessary. The effectiveness can be seen in Figure 3. The image on the left clearly shows that many points are assigned to a cluster whose center is far away from the actual point. In contrast, our encoding strategy is able to correctly identify most of the outliers (marked in purple), as shown in the right image.

4 The AutoNR Algorithm

In this section, we create the example algorithm *AutoNR*. Therefore, we combine our parameter-free, non-redundant clustering framework with a specific non-redundant clustering algorithm. We exemplary choose *Nr-Kmeans* [16], because it uses a relatively simple *k-means*-based objective function. Furthermore, it updates all subspaces simultaneously what supports our idea of full-space clustering executions.



(a) Result without outlier detection. (b) Result with outlier detection (outliers in purple).

Figure 3: Clustering results of *AutoNR* with and without outlier detection on the second subspace of *syn3o*.

With *k-means*-based clustering, we know that only the distance to the nearest center matters. Here, we can assume that all clusters have an identical sphere of influence, which is the same in all directions. Statistically, this means that the data distribution in each subspace follows a GMM with a tied single-variance covariance matrix. This diagonal matrix has the same variance in each dimension and is valid for all clusters within a subspace, $\Sigma_j = \sigma_j^2 I_{m_j}$. Therefore, we can simplify the pdf of the multivariate normal distribution to obtain $L(X_j)$.

$$\pi_{\text{mvnd}}(x) = \frac{\exp\left(-\frac{1}{2}(x-\mu_{j,x})^T \Sigma_j^{-1}(x-\mu_{j,x})\right)}{\sqrt{(2\pi)^{m_j}\det(\Sigma_j)}}$$

$$\overset{\Sigma_j=\sigma_j^2 I_{m_j}}{=} \frac{\exp\left(-\frac{1}{2\sigma_j^2}||x-\mu_{j,x}||_2^2\right)}{\sqrt{(2\pi\sigma_j^2)^{m_j}}},$$

where $\mu_{j,x}$ is the center to which the object x is assigned to in subspace j. We utilize the maximum log-likelihood estimation to determine the variance σ_j^2.

$$\ln\left(\prod_{x\in X_j} \frac{\exp\left(-\frac{1}{2\sigma_j^2}||x-\mu_{j,x}||_2^2\right)}{\sqrt{(2\pi\sigma_j^2)^{m_j}}}\right)$$

$$= \sum_{x\in X_j} \ln\left(\frac{\exp\left(-\frac{1}{2\sigma_j^2}||x-\mu_{j,x}||_2^2\right)}{\sqrt{(2\pi\sigma_j^2)^{m_j}}}\right)$$

$$= -\frac{1}{2}\left(m_j N \ln(2\pi\sigma_j^2) + \frac{1}{\sigma_j^2}\sum_{x\in X_j}||x-\mu_{j,x}||_2^2\right) \quad (4.6)$$

By setting the derivative regarding σ_j^2 to zero we obtain:

$$\sigma_j^2 = \frac{1}{m_j N}\sum_{x\in X_j}||x-\mu_{j,x}||_2^2. \quad (4.7)$$

To receive the encoding costs of all objects in a subspace, we sum up the negative binary logarithm of the pdf of each point.

$$L_{\text{mvnd}}(X_j) = \sum_{x \in X_j} -\log_2 \left(\frac{\exp\left(-\frac{1}{2\sigma_j^2}||x-\mu_{j,x}||_2^2\right)}{\sqrt{(2\pi\sigma_j^2)^{m_j}}} \right)$$

$$= -\frac{1}{\ln(2)} \sum_{x \in X_j} \ln \left(\frac{\exp\left(-\frac{1}{2\sigma_j^2}||x-\mu_{j,x}||_2^2\right)}{\sqrt{(2\pi\sigma_j^2)^{m_j}}} \right)$$

$$\overset{(4.6)}{=} \frac{1}{2\ln(2)} \Big(m_j N \ln(2\pi\sigma_j^2) + \frac{1}{\sigma_j^2} \sum_{x \in X_j} ||x-\mu_{j,x}||_2^2 \Big)$$

$$\overset{(4.7)}{=} \frac{m_j N}{2\ln(2)} \left(1 + \ln\left(\frac{2\pi}{m_j N}\right) + \ln\Big(\sum_{x \in X_j} ||x-\mu_{j,x}||_2^2 \Big) \right)$$

The final step is to include δ.

$$L(X_j) = L_{\text{mvnd}}(X_j) - m_j N \log_2(\delta) \tag{4.8}$$

Using this encoding, the only distribution-specific parameter is σ_j^2, thus $p_j = 1$. Furthermore, we can use Eq. (4.8) to calculate the threshold of the distance between a point and its cluster center that determines whether it is an outlier. For clarity, we set $Y = \sum_{x \in X_j} ||x-\mu_{j,x}||_2^2$ and $Z = ||o_j - \mu_{j,o_j}||_2^2$.

$$L_{\text{total}}(o_j) < L(X_j) - L(X_j \setminus o_j)$$

$$\overset{(4.8)}{\Rightarrow} L_{\text{total}}(o_j) < \frac{m_j N}{2\ln(2)} \left(1 + \ln\left(\frac{2\pi}{m_j N}\right) + \ln\big(Y\big) \right)$$

$$- \frac{m_j (N-1)}{2\ln(2)} \left(1 + \ln\left(\frac{2\pi}{m_j (N-1)}\right) + \ln\big(Y - Z\big) \right)$$

$$\Rightarrow \overbrace{\frac{1}{(N-1)} \left(\frac{2\ln(2) L_{\text{total}}(o_j)}{m_j} - 1 - \ln\left(\frac{2\pi}{m_j N}\right) - \ln\big(Y\big) \right)}^{=:A}$$

$$< -\ln\big(N\big) + \ln\big(Y\big) + \ln\big(N-1\big) - \ln\big(Y - Z\big)$$

$$\Rightarrow Z > Y\big(1 - \frac{N-1}{N} \exp(-A)\big)$$

This formulation tells us: If the squared euclidean distance between a point and its center in subspace j is greater than $Y\big(1 - \frac{N-1}{N} \exp(-A)\big)$, then it is an outlier.

Another essential property of our encoding strategy is that it is scaling invariant. Assume that the entire dataset is multiplied by a factor f. Then, the precision δ naturally changes to $f\delta$. The following applies:

$$L(fX_j) = L(X_j) + \frac{m_j N}{2\ln(2)} \ln(f^2) - m_j N \log_2(f)$$

$$= L(X_j) + \frac{m_j N}{\ln(2)} \ln(f) - \frac{m_j N}{\ln(2)} \ln(f)$$

Since $\max(\text{dist}_X)$ changes analogously to δ, $L(f\mu_{j,i}) = L(\mu_{j,i})$ is also valid. All other costs are trivially independent of factor f.

The code, supplement and datasets can be downloaded at `https://dmm.dbs.ifi.lmu.de/downloads`.

5 Experiments

We conduct experiments on synthetic and real-world datasets. The information regarding the datasets and comparison algorithms can be found in the supplement. An analysis of the runtime can also be found there.

5.1 Quantitative Analyses We measure the performance of the algorithms using the ground truth labels of the datasets. In the non-redundant case, the prediction labels ($R^p \in \mathbb{N}^{N \times J_p}$) and the ground truth labels ($R^{gt} \in \mathbb{N}^{N \times J_{gt}}$) consist of multiple sets. Therefore, we report the prediction labels that best match the ground truth of each subspace. For subspace j this equals:

$$\text{score}_j(R^{gt}, R^p) = \max(\{\text{metric}(R_j^{gt}, R_i^p) \mid 1 \leq i \leq J_p\})$$

We use the *Normalized Mutual Information* (NMI) and the F1 score as metrics. Both produce values between 0 and 1 with an optimum result of 1. Most competitor algorithms are not able to process high-dimensional datasets in an acceptable time (<24h). Therefore, we conduct PCA to reduce the number of features while keeping 90% of the data's variance if $d > 50$. Furthermore, we apply standardization (zero mean and unit variance for all features) for *Wine*.

The results of the different experiments are shown in Table 1. We repeated each experiment ten times and added the average NMI and F1 scores ± the standard deviation. The NMI and F1 results often differ significantly. This is because the F1 score is more generous if the number of clusters is low.

From these results, we observe that *AutoNR* outperforms the other algorithms on almost all datasets. In particular, the comparison methods seem to have problems identifying additional good clusterings after a first high-quality subspace has been found. An example of this situation is *NRLetters*. All algorithms find a first clustering with an average NMI of > 0.89. However, the quality of the further clusterings decreases more and more for all methods except *AutoNR*. This shows the value of not initially fixing the subspaces. It is important to note that we are not influencing the order in which the algorithms identify the subspaces. If *AutoNR* does not achieve the best results, we are usually in second place (see *Fruits*, *ALOI* and *Optdigits*).

Outliers negatively affect the clustering results of *AutoNR* without outlier detection. This can clearly be seen in the case of *syn3o*. While we achieve perfect results for *syn3*, the quality suffers significantly due to the additional outliers added. Here, our outlier detection significantly improves the results in all three subspaces. This effect also applies to real-world datasets. For example, for *Wine*, the average NMI increases from 0.76 to 0.85 and the average F1 score from 0.81 to 0.90.

Table 1: Experimental results of our method without (*AutoNR-*) and with (*AutoNR+*) outlier detection against other algorithms on various datasets. The left side shows the NMI results in %, the F1 results in % are shown on the right. All experiments were run ten times, and the average result ± the standard deviation is stated. The best algorithm for each subspace is marked in bold. The † indicates that that algorithm could not process the dataset due to runtime constraints ($\dagger_R$) or memory issues ($\dagger_M$).

		NMI (%)						F1 (%)					
Dataset	**Subspace**	*AutoNR-*	*AutoNR+*	*NrDipmeans*	*ISAAC*	*MISC*	*FOSSCLU*	*AutoNR-*	*AutoNR+*	*NrDipmeans*	*ISAAC*	*MISC*	*FOSSCLU*
syn3 (N=5000, d=11)	1st (k_j=4)	**100 ± 0**	**100 ± 0**	96 ± 7	93 ± 15	89 ± 19	81 ± 1	**100 ± 0**	**100 ± 0**	94 ± 10	92 ± 18	86 ± 24	70 ± 2
	2nd (k_j=3)	**100 ± 0**	**100 ± 0**	87 ± 13	95 ± 12	92 ± 17	65 ± 9	**100 ± 0**	**100 ± 0**	85 ± 16	94 ± 15	89 ± 24	60 ± 5
	3rd (k_j=2)	**100 ± 0**	**100 ± 0**	80 ± 21	**100 ± 0**	92 ± 27	59 ± 9	**100 ± 0**	**100 ± 0**	79 ± 23	**100 ± 0**	92 ± 25	69 ± 7
syn3o (N=5150, d=11)	1st (k_j=4)	86 ± 10	**97 ± 0**	85 ± 10	86 ± 19	75 ± 18	78 ± 2	83 ± 18	**99 ± 0**	86 ± 12	86 ± 22	75 ± 22	69 ± 1
	2nd (k_j=3)	90 ± 10	**96 ± 0**	84 ± 12	90 ± 13	75 ± 14	62 ± 8	91 ± 17	**99 ± 0**	85 ± 16	89 ± 17	75 ± 19	59 ± 5
	3rd (k_j=2)	77 ± 20	**94 ± 0**	66 ± 17	77 ± 31	69 ± 30	55 ± 6	80 ± 27	**99 ± 0**	69 ± 21	81 ± 29	78 ± 27	66 ± 2
Fruits (N=105, d=6)	Species (k_j=3)	85 ± 9	83 ± 7	82 ± 6	21 ± 24	56 ± 14	**90 ± 9**	89 ± 9	87 ± 7	87 ± 7	56 ± 8	59 ± 16	**93 ± 8**
	Color (k_j=3)	17 ± 2	18 ± 1	17 ± 2	6 ± 3	13 ± 5	**19 ± 2**	47 ± 3	44 ± 5	47 ± 2	**49 ± 3**	42 ± 5	40 ± 3
ALOI (N=288, d=611)	Shape (k_j=2)	62 ± 4	64 ± 3	65 ± 3	**70 ± 8**	38 ± 8	58 ± 4	65 ± 2	65 ± 1	67 ± 1	**86 ± 6**	59 ± 5	60 ± 4
	Color (k_j=2)	62 ± 4	64 ± 3	**65 ± 3**	16 ± 11	36 ± 9	61 ± 6	65 ± 2	65 ± 1	**66 ± 2**	55 ± 5	58 ± 6	62 ± 9
DSF (N=900, d=400)	Body-up (k_j=3)	**100 ± 0**	**100 ± 0**	**100 ± 0**	71 ± 2	**100 ± 0**	73 ± 1	**100 ± 0**	**100 ± 0**	**100 ± 0**	60 ± 3	**100 ± 0**	65 ± 2
	Body-low (k_j=3)	**100 ± 0**	**100 ± 0**	71 ± 6	71 ± 2	**100 ± 0**	42 ± 0	**100 ± 0**	**100 ± 0**	73 ± 5	60 ± 3	**100 ± 0**	39 ± 1
CMUface (N=624, d=960)	Identity (k_j=20)	**68 ± 4**	64 ± 4	55 ± 0	20 ± 4	42 ± 10	57 ± 7	**38 ± 4**	34 ± 3	29 ± 0	12 ± 4	19 ± 5	29 ± 5
	Pose (k_j=4)	**35 ± 3**	33 ± 1	20 ± 0	3 ± 4	3 ± 3	21 ± 12	**45 ± 4**	42 ± 4	**45 ± 0**	34 ± 1	30 ± 5	41 ± 9
WebKB (N=1041, d=323)	Category (k_j=4)	32 ± 2	**34 ± 3**	16 ± 4	$\dagger_R$	$\dagger_R$	$\dagger_M$	50 ± 5	**58 ± 7**	51 ± 4	$\dagger_R$	$\dagger_R$	$\dagger_M$
	University (k_j=4)	56 ± 4	**57 ± 3**	24 ± 13	$\dagger_R$	$\dagger_R$	$\dagger_M$	51 ± 2	**52 ± 3**	45 ± 5	$\dagger_R$	$\dagger_R$	$\dagger_M$
NRLetters (N=10000, d=189)	Letter (k_j=6)	**100 ± 0**	**100 ± 0**	95 ± 7	87 ± 9	92 ± 8	82 ± 0	**100 ± 0**	**100 ± 0**	91 ± 12	78 ± 13	85 ± 11	67 ± 2
	Color (k_j=3)	**100 ± 0**	**100 ± 0**	60 ± 30	89 ± 18	87 ± 18	54 ± 18	**100 ± 0**	**100 ± 0**	66 ± 22	86 ± 24	82 ± 25	53 ± 16
	Corner (k_j=4)	**100 ± 0**	**100 ± 0**	58 ± 28	71 ± 11	70 ± 11	**100 ± 0**	**100 ± 0**	**100 ± 0**	59 ± 23	70 ± 11	70 ± 11	**100 ± 0**
Wine (N=178, d=13)	Type (k_j=3)	76 ± 5	**85 ± 3**	34 ± 11	0 ± 0	32 ± 23	84 ± 4	81 ± 6	**90 ± 4**	59 ± 5	0 ± 0	42 ± 29	**90 ± 4**
Optdigits (N=5620, d=64)	Digit (k_j=10)	73 ± 1	74 ± 1	39 ± 14	74 ± 7	**79 ± 1**	51 ± 2	54 ± 4	58 ± 4	39 ± 10	59 ± 10	**64 ± 4**	40 ± 2

Another positive aspect of *AutoNR* with outlier detection is its stability. While the other methods often have a high standard deviation (up to 0.30), the maximum standard deviation of *AutoNR+* across all tested datasets is 0.07. Overall, it can be seen that the outlier detection seldom leads to worse results. This means that only rarely are points wrongly identified as outliers. Therefore, we recommend always activating the outlier detection feature, which is essential for our claim that the procedure is parameter-free.

Based on the results on *Wine* and *Optdigits*, it can be assumed that *AutoNR* can also be used for datasets in which no non-redundant clusterings are suspected.

5.2 Qualitative Analyses First, we want to mention the high interpretability of the results. For this, we look at the cluster centers found in each subspace of *NRLetters* by *AutoNR*. These are illustrated in Fig. 4.

It can be seen that the first subspace describes the individual letters (4(a)). Here one should note that the colors have been completely extracted from the subspace. The colors are individually described in the second subspace (4(b)). In this case, however, the individual letters are no longer recognizable. The corner markings also receive their individual subspace (4(c)). Here, neither the colors nor the letters are represented. However, only one corner is highlighted, while all corners are slightly tagged in the other subspaces.

(a) 'Letter' subspace.

(b) 'Color' subspace.

(c) 'Corner' subspace.

Figure 4: The cluster centers in the three subspaces of *NRLetters* as identified by *AutoNR*.

Our method shows some problems in clustering *ALOI* correctly. Instead of identifying one subspace each for the shape and color, *AutoNR* finds a subspace with the clusters 'red box', 'green box', 'red ball', and 'green ball'. This is also the case with most competitor algorithms. Splitting this space into two would increase the MDL costs. However, if only a single clustering with four clusters is desired, *AutoNR* gives very good results.

The opposite is the case with the *Identity* subspace of *CMUface*. Here, splitting the subspace into two spaces reduces the MDL costs. Thus, *AutoNR* finds additional substructures in the data that can be used to increase the compression. If we were to merge the two subspaces again, we would get NMI values of over 0.85 and F1 values of over 0.72.

6 Conclusion

In this work, we introduced a framework to automatically determine the number of subspaces and clusters within subspaces in a non-redundant clustering setting. Additionally, we enhanced the quality of the clustering results by identifying outliers in each of the subspaces. Our framework is easily combinable with different non-redundant clustering approaches.

Experiments show that our exemplary algorithm *AutoNR* achieves state-of-the-art results. In particular, the outlier detection feature leads to results outperforming other parameter-free approaches. Since most operations are executed in lower-dimensional subspaces, it is significantly faster than most competitor algorithms for high-dimensional datasets.

At the moment, our framework is only able to handle orthogonal subspaces. Future efforts may attempt to combine our method with multi-view clustering algorithms (e.g. [25]). In this setting, dimensions can be used multiple times for several clustering solutions.

Acknowledgments

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

The present contribution is supported by the Helmholtz Association under the joint research school 'Munich School for Data Science - MUDS'.

References

[1] M. Ahmed and A. N. Mahmood, *A novel approach for outlier detection and clustering improvement*, in 8th IEEE ICIEA, IEEE, 2013, pp. 577–582.

[2] H. Bischof, A. Leonardis, and A. Selb, *Mdl principle for robust vector quantisation*, Pattern Analysis & Applications, 2 (1999), pp. 59–72.

[3] S. Chawla and A. Gionis, *k-means--: A unified approach to clustering and outlier detection*, in SIAM SDM, SIAM, 2013, pp. 189–197.

[4] Y. Cui, X. Z. Fern, and J. G. Dy, *Non-redundant multi-view clustering via orthogonalization*, in 7th IEEE ICDM, IEEE, 2007, pp. 133–142.

[5] Y. Feng and G. Hamerly, *Pg-means: learning the number of clusters in data*, in Advances in neural information processing systems, 2007, pp. 393–400.

[6] G. Gan and M. K.-P. Ng, *K-means clustering with outlier removal*, Pattern Recognition Letters, 90 (2017), pp. 8–14.

[7] S. Goebl, X. He, C. Plant, and C. Böhm, *Finding the optimal subspace for clustering*, in 14th IEEE ICDM, IEEE, 2014, pp. 130–139.

[8] P. Grünwald, *A tutorial introduction to the minimum description length principle*, Advances in minimum description length: Theory and applications, (2005), pp. 3–81.

[9] G. Hamerly and C. Elkan, *Learning the k in k-means*, vol. 16, MIT Press, 2004, pp. 281–288.

[10] J. A. Hartigan, P. M. Hartigan, et al., *The dip test of unimodality*, Annals of statistics, 13 (1985), pp. 70–84.

[11] V. Hautamäki, S. Cherednichenko, I. Kärkkäinen, T. Kinnunen, and P. Fränti, *Improving k-means by outlier removal*, in Scandinavian Conference on Image Analysis, Springer, 2005, pp. 978–987.

[12] V. Hautamaki, I. Karkkainen, and P. Franti, *Outlier detection using k-nearest neighbour graph*, in 17th IEEE ICPR, vol. 3, IEEE, 2004, pp. 430–433.

[13] T. Ishioka et al., *An expansion of x-means for automatically determining the optimal number of clusters*, in ICCI, vol. 2, 2005, pp. 91–95.

[14] T. C. Lee, *An introduction to coding theory and the two-part minimum description length principle*, International statistical review, 69 (2001), pp. 169–183.

[15] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Towards an optimal subspace for k-means*, in 23rd ACM SIGKDD, 2017, pp. 365–373.

[16] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Discovering non-redundant k-means clusterings in optimal subspaces*, in 24th ACM SIGKDD, 2018, pp. 1973–1982.

[17] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Non-redundant subspace clusterings with nr-kmeans and nr-dipmeans*, ACM Transactions on Knowledge Discovery from Data (TKDD), 14 (2020), pp. 1–24.

[18] P. Miettinen and J. Vreeken, *Model order selection for boolean matrix factorization*, in 17th ACM SIGKDD, 2011, pp. 51–59.

[19] D. Niu, J. G. Dy, and M. I. Jordan, *Multiple non-redundant spectral clustering views*, in ICML, 2010, pp. 831–838.

[20] D. Pelleg, A. W. Moore, et al., *X-means: Extending k-means with efficient estimation of the number of clusters.*, in ICML, vol. 1, 2000, pp. 727–734.

[21] J. Rissanen, *A universal prior for integers and estimation by minimum description length*, The Annals of statistics, 11 (1983), pp. 416–431.

[22] J. Rissanen, *Stochastic complexity and modeling*, The annals of statistics, (1986), pp. 1080–1100.

[23] C. E. Shannon, *A mathematical theory of communication*, Bell system technical journal, 27 (1948), pp. 379–423.

[24] X. Wang, J. Wang, C. Domeniconi, G. Yu, G. Xiao, and M. Guo, *Multiple independent subspace clusterings*, in AAAI, vol. 33, 2019, pp. 5353–5360.

[25] S. Yao, G. Yu, J. Wang, C. Domeniconi, and X. Zhang, *Multi-view multiple clustering*, in 28th IJCAI, IJCAI, ijcai.org, 2019, pp. 4121–4127.

[26] W. Ye, S. Maurus, N. Hubig, and C. Plant, *Generalized independent subspace clustering*, in 16th IEEE ICDM, IEEE, 2016, pp. 569–578.

Supplement - 'Automatic Parameter Selection for Non-Redundant Clustering'

Collin Leiber* Dominik Mautz* Claudia Plant† Christian Böhm*

1 Symbols

We use the following symbols in our paper as well as in the supplement:

Table 1: Description of the used symbols.

Symbol	Interpretation
$N \in \mathbb{N}$	Number of objects
$d \in \mathbb{N}$	Dimensionality of the feature space
$J \in \mathbb{N}$	Number of subspaces
$V \in \mathbb{R}^{d \times d}$	Orthogonal (rotation) matrix
$\delta \in \mathbb{R}$	The precision of the encoding
$I_{m_j} \in \mathbb{N}^{m_j \times m_j}$	$m_j \times m_j$ identity matrix
$X \subseteq \mathbb{R}^d$	Set of all objects
$k_j \in \mathbb{N}$	Number of clusters in subspace j
$m_j \in \mathbb{N}$	Dimensionality of subspace j
$P_j \in \mathbb{N}^{d \times m_j}$	Projection matrix of subspace j
$p_j \in \mathbb{N}$	Number of distribution-specific parameters in subspace j
$X_j \subseteq \mathbb{R}^{m_j}$	X projected to subspace j
$O_j \subseteq X_j$	Set of outliers in subspace j
$\mu_{j,i} \in \mathbb{R}^{m_j}$	Center of cluster i in subspace j
$\Sigma_{j,i} \in \mathbb{R}^{m_j \times m_j}$	Covariance matrix of cluster i in subspace j
$C_{j,i} \subseteq X_j$	Objects of cluster i in subspace j
$J_p \in \mathbb{N}$	Number of predicted subspaces
$Jgt \in \mathbb{N}$	Number of true subspaces
$R^p \in \mathbb{N}^{N \times J_p}$	Prediction labels matrix
$R^{gt} \in \mathbb{N}^{N \times J_{gt}}$	Ground truth labels matrix

2 Encoding the Constant Values

At this point, we would like to give a brief intuition on how the constant components of the encoding strategy we present in the paper could be handled.

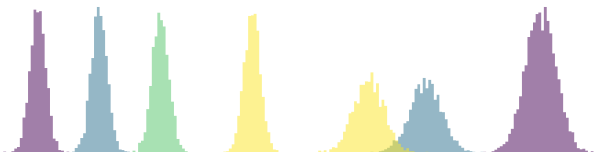
The number of objects N and dimensionality d can again be encoded using the natural prior for integers. Therefore, we need $L^0(N) + L^0(d)$ bits to encode these values.

In general real values r can be encoded by separately encoding the integer part $\lfloor r \rfloor$ and the decimal places [5]. Here, the precision δ is required for the decimal places. Hence, the following applies:

$$L(r) = L^0(\lfloor r \rfloor) - \log_2(\delta).$$

This can be used to encode the $d + 1$ values needed to define the hypercube. It can also be used to encode V. Since all the column vectors of V are orthonormal, they all have a length of 1. Therefore, all values of a column are less than or equal to 1 and we can consequently ignore $L^0(\lfloor r \rfloor)$. Moreover, since the orientation is indifferent, the last entry can be calculated using the first $d-1$ entries. Thereby each column loses one degree of freedom. Furthermore, the orthogonal property means that each following column loses an additional degree of freedom. Therefore, we can encode the first column using $-\log_2(\delta)(d-1)$ bits, the second with $-\log_2(\delta)(d-2)$ bits, and so forth. All in all, the code length of V is $-\log_2(\delta)\frac{d(d-1)}{2}$.

From these encodings, it is very easy to see that the values are actually constants that are independent of a particular clustering result.



(a) Subspace with 4 clusters. (b) Subspace with 3 clusters.

Figure 1: Histograms of two 1-dimensional subspaces.

3 Search Space Restrictions

In this section, we would like to justify our restrictions on the search space with an example.

In the paper, we say that we restrict the number of clusters in case of a *cluster space* split as follows:

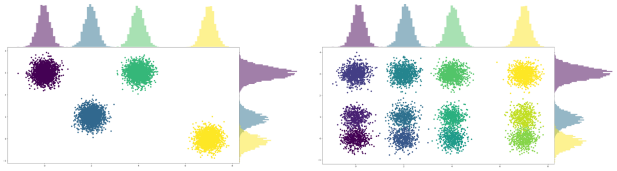
$$\max(k_{\text{split}_1}, k_{\text{split}_2}) \leq k_{\text{original}} \leq k_{\text{split}_1} \cdot k_{\text{split}_2}.$$

Also, we restrict the number of clusters for a *cluster space* merge with the inverted rule.

$$\max(k_{\text{original}_1}, k_{\text{original}_2}) \leq k_{\text{merge}} \leq k_{\text{original}_1} \cdot k_{\text{original}_2}$$

To better understand these rules, assume that we have the 1-dimensional subspaces shown in Figure 1 with 4 and 3 clusters, respectively. If we want to merge these subspaces, we will always get at least 4 clusters, since 4 clusters are already contained in the first subspace. Moreover, there are a maximum of 12 cluster combinations that can occur. Both extreme situations are illustrated in Figure 2.

*LMU Munich. {leiber, mautz, boehm}@dbs.ifi.lmu.de
†Faculty of Computer Science, ds:UniVie, University of Vienna, Vienna, Austria. claudia.plant@univie.ac.at



(a) Combination with the minimum number of 4 clusters.

(b) Combination with the maximum number of 12 clusters.

Figure 2: Two possible combinations of the two subspaces from Figure 1.

With a *cluster space* split, essentially the same applies, but in reverse. Here, no subspace may be created that already has more than the original number of clusters. Furthermore, $k_{\text{split}_1} \cdot k_{\text{split}_2}$ must be greater than or equal to the original number of clusters. If one of these two rules is not met, subspaces would be created that do not fit the structure of the original subspace.

These rules can also be applied to higher dimensional subspaces.

4 Pseudo-code

In order to determine the number of subspaces and clusters within subspaces for non-redundant clustering, the following steps are executed:

- *Noise Space* Split
- *Cluster Space* Split
- *Cluster Space* Merge

Additionally, we regularly combine model parameters to perform a full-space execution. To better understand how all these steps are linked, Algorithm 1 can be analyzed.

5 Implementation Details of AutoNR

We want to give additional information regarding the implementation of *AutoNR*.

Unfortunately, *Nr-Kmeans* introduced another parameter that has to be set by the user. The algorithm optimizes V and P_j through eigenvalue decompositions. Here, the eigenvectors represent the direction, and the signs of the eigenvalues E determine to which subspace the dimensions are assigned. Dimensions not matching the structure of any *cluster space* are assigned to the *noise space*. Consequently, the *noise space* will capture all dimensions corresponding to eigenvalues ≥ 0. Yet, the supplementary of [6] states that eigenvectors with a negative eigenvalue close to zero should also be assigned to the *noise space*. An example value is given in the respective publication. However, the optimal threshold changes depending on the input dataset. We want to avoid such hard thresholds in our approach. Therefore, we utilize the described encoding strategy to determine which dimensions should be contained in the cluster and which in the *noise space*.

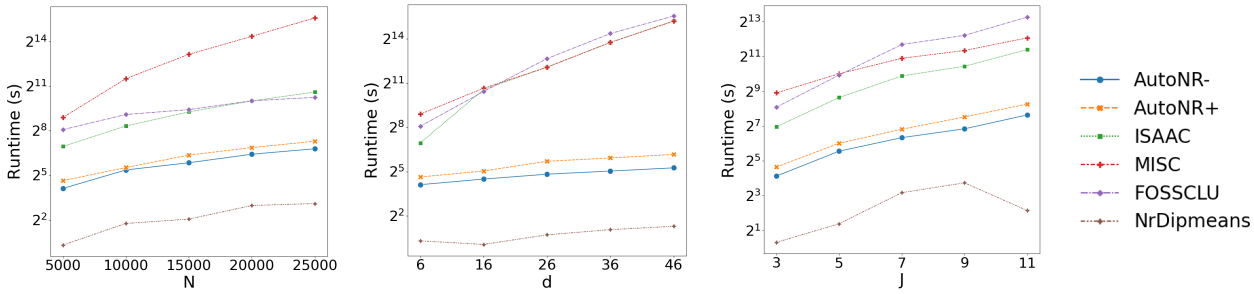
The rotation matrix V can be updated independently of the new subspace dimensionalities. Therefore, V can be calculated a priori and used in the process to define the new m_{cluster} and m_{noise}. The parameters present in the current iteration of *Nr-Kmeans* can be used to calculate the temporary MDL costs of the model. Since the cluster assignments, cluster centers, and scatter matrices stay constant during this operation, only those costs that depend on the subspace dimensionalities m_j and the projections P_j need to be considered. We start with a *cluster space* that only obtains the dimension corresponding to the lowest eigenvalue and a *noise space* containing the other $|E| - 1$ dimensions. The approach is repeated with a rising number of *cluster space* dimensions until the MDL costs exceed the result from the previous iteration or the dimensionality of the *cluster space* equals the number of negative eigenvalues. This means that an initial threshold is no longer necessary

We further utilize the initialization procedure of k-means++ [1] to seed the cluster centers.

6 Evaluation Setup

Datasets: *syn3* is a synthetic dataset with three subspaces containing 4, 3, and 2 clusters. Each cluster was created using a Gaussian distribution. For *syn3o*, we randomly added 150 uniformly distributed outliers in each subspace. We additionally created the *NRLetters* dataset. It consists of 10000 9×7 RGB images of the letters 'A', 'B', 'C', 'X', 'Y', and 'Z' in the colors pink, cyan, and yellow. Moreover, in each image, a corner pixel is highlighted in the color of the letter. This results in three possible clusterings. An extract of this dataset can be seen in the paper in Figure 1. *Wine* is a real-world dataset from the UCI[1] repository with three clusters. The UCI dataset *Optdigits* consists of 5620 8×8 images, each representing a digit. The *Fruits* [4] dataset was created using 105 images of apples, bananas, and grapes in red, green, and yellow. The images have been preprocessed, resulting in six attributes. The *Amsterdam Library of Object Image*[2] dataset (*ALOI*) contains images of 1000 objects recorded from different angles. For our analysis, we use a common subset of this data consisting of 288 images illustrating the objects 'box' and 'ball'

[1] https://archive.ics.uci.edu/ml/index.php
[2] http://aloi.science.uva.nl/



(a) Increasing N ($d = 6, J = 3$). (b) Increasing d ($N = 5000, J = 3$). (c) Increasing J ($N = 5000, d = 2J$). (d) Legend.

Figure 3: Scalability of *AutoNR* without (-) and with (+) outlier detection compared to its competitors. All tests are repeated ten times and the mean is stated.

in the colors green and red. *Dancing Stick Figures* [3] (*DSF*) is a dataset containing 900 20×20 images. It comprises two subspaces describing three upper- and three lower-body motions. *CMUface* is again taken from the UCI repository and is composed of 640 30×32 gray-scaled images showing 20 persons in four different poses (up, straight, left, right). Among those images, 16 show glitches resulting in 624 useful objects. The *WebKB*[3] dataset contains 1041 Html documents from four universities. These web pages belong to one of four categories. We preprocessed the data using stemming and removed stop words and words with a document frequency $< 1\%$. Afterward, we removed words with a variance < 0.25, resulting in 323 features.

Comparison Methods: We compare the results of *AutoNR* without (*AutoNR-*) and with (*AutoNR+*) outlier detection against the parameter-free algorithms *ISAAC* [9] and *MISC* [8] as well as *NrDipmeans* [7]. Furthermore, we extend the subspace clustering approach *FOSSCLU* [2] to iteratively identify new subspaces by removing the subspaces found in previous iterations. For *NrDipmeans* and *FOSSCLU* we have to state the desired number of subspaces. In case of *FOSSCLU* we need to define limits for m_j and k_j. We set those to $1 \leq m_j \leq 3$ and $2 \leq k_j \leq 10$. We wanted to set the upper bound of k_j to 20 for *CMUface*, so *FOSSCLU* would be able to determine all parameters correctly. Unfortunately, this leads to memory issues. Where required, *AutoNR* runs 15 executions of *Nr-Kmeans*. The significance for *NrDipmeans* is set to 0.01.

Experiments are conducted using the Scala implementations of *Nr-Kmeans* and *NrDipmeans* and the Matlab implementations of *ISAAC* and *MISC* as referenced in [6], [7], [9] and [8] respectively. Regarding *FOSSCLU*, we extend the Java version referenced in [2] as described above. *AutoNR* is implemented in Python.

[3] http://www.cs.cmu.edu/ webkb/

7 Runtime Analysis

We conduct runtime experiments on datasets with a rising number of objects N, dimensions d, and subspaces J. The created *cluster spaces* are always two-dimensional and contain three Gaussian clusters each.

All experiments are performed on a computer with an Intel Core i7-8700 3.2 GHz processor and 32GB RAM. The runtime results again correspond to the average of ten consecutive executions. The outcomes are illustrated in Figure 3.

The charts show that our approach is well applicable to high-dimensional datasets. The runtime increases only slightly with additional *noise space* dimensions (3(b)). *ISAAC* and *MISC* have to conduct an ISA which does not scale well to high-dimensional datasets. *FOSSCLU* has to perform Givens rotations multiple times, which is an expensive operation. On the other hand, our framework performs most steps in lower-dimensional subspaces where the overall dimensionality has no significant influence. If additional *cluster spaces* accompany a higher dimensionality, the runtimes of all algorithms behave similarly (3(c)). For large datasets, the differences in runtime are also much less prominent (3(a)). Only *MISC* needs significantly more time because a kernel graph regularized semi-nonnegative matrix factorization has to be performed.

Due to the additional operations required to calculate the outlier distance threshold for each subspace in each iteration, the execution of *AutoNR* with outlier detection expectably takes more time than without. Furthermore, the cluster centers and covariance matrices are updated after each outlier detection procedure.

NrDipmeans is the fastest in all experiments. However, it must be noted that *NrDipmeans* knows the correct number of subspaces and therefore does not need to run tests to identify J. Furthermore, in the case of $J = 11$, it settles with the initial two clusters in

Algorithm 1: Parameter search algorithm.

Function `main`(*dataset* X)
  V = initialize randomly
  R_{best} = Initial result (single noise space)
  // Sort by MDL costs, add noise space last
  sortedSpaces = sortSubspaces(R_{best})
  for $j \in$ *sortedSpaces* **do**
    $X_j = \{xVP_j | x \in X\}$
    if j is cluster space **then**
      // Split space into two cluster spaces
      j_{s_1}, j_{s_2} = clusterSpaceSplit(X_j, k_j)
    else if j is noise space **then**
      // Split space into cluster and noise space
      j_{s_1}, j_{s_2} = noiseSpaceSplit(X_j)
    // Check MDL costs
    if cost(j_{s_1}) + cost(j_{s_2}) < cost(j) **then**
      // Join parameters for full-space execution
      $V_{\text{tmp}}, P_{\text{tmp}}, \mu_{\text{tmp}}$ = joinParams($R_{\text{best}}, V, j_{s_1}, j_{s_2}$)
      R_{tmp}, V_{tmp} = fullSpace($X, V_{\text{tmp}}, P_{\text{tmp}}, \mu_{\text{tmp}}$)
      // Check full-space MDL costs
      if cost(R_{tmp}) < cost(R_{best}) **then**
        $R_{\text{best}} = R_{\text{tmp}}$; $V = V_{\text{tmp}}$
        go to line 5
  R_{best}, V = `merging`(R_{best}, V)
  if merging was successful **then**
    go to line 5
  return R_{best}
Function `merging`(R_{best}, V)
  for each pair (j_1, j_2) of cluster spaces **do**
    $X_{j_1,j_2} = \{xVP_{j_1,j_2} | x \in X\}$
    j_m = clusterSpaceMerge(X_{j_1,j_2}, k_{j_1}, k_{j_2})
    // Check MDL costs
    if cost(j_m) < cost(j_1) + cost(j_2) **then**
      // Join parameters for the full-space execution
      $V_{\text{tmp}}, P_{\text{tmp}}, \mu_{\text{tmp}}$ = joinParams(R_{best}, V, j_m)
      $R_{\text{tmp}}, V_{\text{tmp}}$ = fullSpace($X, V_{\text{tmp}}, P_{\text{tmp}}, \mu_{\text{tmp}}$)
      // Check full-space MDL costs
      if cost(R_{tmp}) < cost(R_{best}) **then**
        $R_{\text{best}} = R_{\text{tmp}}$; $V = V_{\text{tmp}}$
  if better result found **then**
    go to line 28
  return R_{best}, V

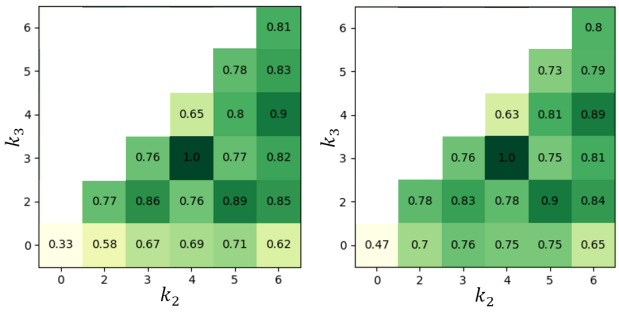


Figure 4: Results of various parametrizations of *Nr-Kmeans* on *NRLetters*. k_1 is set to 6. The left image shows the NMI, and the right the F1 results.

each subspace and does not invest time in finding better structures. Therefore, it seems to have problems running with a high J. *AutoNR*, on the other hand, almost always correctly identifies all clusters in all subspaces.

Our procedure could be further accelerated by, for example, parallelizing the multiple executions of *Nr-Kmeans* with identical parameters.

8 Comparison to *Nr-Kmeans*

We perform additional experiments using the original *Nr-Kmeans* algorithm, to show that the good experimental results are based on our proposal and not merely on the integration of *Nr-Kmeans*. The new results are shown in Table 2. As in the paper, we repeated each experiment ten times and added the average score ± the standard deviation to the table.

AutoNR returns superior results in most experiments, even though *Nr-Kmeans* already knows the correct number of subspaces and clusters for each subspace. Only for the non-redundant dataset *ALOI* does the original *Nr-Kmeans* perform better regarding the F1 score. This case, however, has already been mentioned in the paper. The biggest advantage of our application is the fact that it discovers structures one by one while preserving the ability to adjust already found subspaces. This gives great flexibility, so that possible errors can be compensated in a following iteration. Another advantage is the definition of the *noise space* using MDL (see supplement Section 5), as it can be seen with the datasets *CMUface*, *WebKB*, *NRLetters*, *Wine* and *Optdigits*.

One could argue that the multiple repetitions of *Nr-Kmeans* included in each run of *AutoNR* strongly favor our algorithm. However, we would like to counter this by stating that *Nr-Kmeans* by itself is often unable to achieve a perfect result just once (e.g., for *syn3*). In

Table 2: Results of *AutoNR-* and *AutoNR+* compared to our *Nr-Kmeans* version, where the dimensionality of the *noise space* is determined through MDL (see supplement Section 5), and the original *Nr-Kmeans* implementation on various datasets. The left side shows the NMI results in %, the F1 results in % are shown on the right. All experiments were run ten times, and the average result ± the standard deviation is stated. The best algorithm for each subspace is marked in bold.

Dataset	Subspace	NMI (%) *AutoNR-*	*AutoNR+*	*Nr-Kmeans* (MDL-based *noise space*)	*Nr-Kmeans* (Original)	F1 (%) *AutoNR-*	*AutoNR+*	*Nr-Kmeans* (MDL-based *noise space*)	*Nr-Kmeans* (Original)
syn3 (N=5000, d=11)	1st (k_j=4)	**100 ± 0**	**100 ± 0**	73 ± 20	59 ± 16	**100 ± 0**	**100 ± 0**	72 ± 18	61 ± 11
	2nd (k_j=3)	**100 ± 0**	**100 ± 0**	81 ± 13	75 ± 17	**100 ± 0**	**100 ± 0**	82 ± 12	78 ± 16
	3rd (k_j=2)	**100 ± 0**	**100 ± 0**	72 ± 22	75 ± 17	**100 ± 0**	**100 ± 0**	79 ± 16	81 ± 14
syn3o (N=5150, d=11)	1st (k_j=4)	86 ± 10	**97 ± 0**	62 ± 17	54 ± 13	83 ± 18	**99 ± 0**	64 ± 14	57 ± 10
	2nd (k_j=3)	90 ± 10	**96 ± 0**	67 ± 15	69 ± 11	91 ± 17	**99 ± 0**	73 ± 14	75 ± 9
	3rd (k_j=2)	77 ± 20	**94 ± 0**	56 ± 10	68 ± 13	80 ± 27	**99 ± 0**	63 ± 10	77 ± 13
Fruits (N=105, d=6)	Species (k_j=3)	**85 ± 9**	83 ± 7	70 ± 14	74 ± 11	**89 ± 9**	87 ± 7	78 ± 11	78 ± 11
	Color (k_j=3)	17 ± 2	**18 ± 1**	15 ± 2	16 ± 2	**47 ± 3**	44 ± 5	42 ± 3	44 ± 3
ALOI (N=288, d=611)	Shape (k_j=2)	62 ± 4	**64 ± 3**	47 ± 26	54 ± 30	65 ± 2	65 ± 1	73 ± 13	**77 ± 15**
	Color (k_j=2)	62 ± 4	**64 ± 3**	34 ± 0	31 ± 10	65 ± 2	65 ± 1	**66 ± 0**	**66 ± 0**
DSF (N=900, d=400)	Body-up (k_j=3)	**100 ± 0**	**100 ± 0**	70 ± 20	81 ± 26	**100 ± 0**	**100 ± 0**	77 ± 16	85 ± 20
	Body-low (k_j=3)	**100 ± 0**	**100 ± 0**	63 ± 22	56 ± 30	**100 ± 0**	**100 ± 0**	70 ± 17	68 ± 22
CMUface (N=624, d=960)	Identity (k_j=20)	68 ± 4	64 ± 4	**78 ± 6**	75 ± 7	38 ± 4	34 ± 3	**57 ± 9**	52 ± 9
	Pose (k_j=4)	**35 ± 3**	33 ± 1	28 ± 8	26 ± 6	**45 ± 4**	42 ± 4	41 ± 10	37 ± 10
WebKB (N=1041, d=323)	Category (k_j=4)	32 ± 2	**34 ± 3**	32 ± 3	30 ± 2	50 ± 5	**58 ± 7**	48 ± 3	48 ± 2
	University (k_j=4)	56 ± 4	**57 ± 3**	47 ± 8	45 ± 7	51 ± 2	52 ± 3	**54 ± 7**	52 ± 3
NRLetters (N=10000, d=189)	Letter (k_j=6)	**100 ± 0**	**100 ± 0**	85 ± 9	83 ± 9	**100 ± 0**	**100 ± 0**	78 ± 13	72 ± 12
	Color (k_j=3)	**100 ± 0**	**100 ± 0**	52 ± 29	39 ± 25	**100 ± 0**	**100 ± 0**	61 ± 22	52 ± 18
	Corner (k_j=4)	**100 ± 0**	**100 ± 0**	57 ± 26	48 ± 25	**100 ± 0**	**100 ± 0**	61 ± 23	51 ± 19
Wine (N=178, d=13)	Type (k_j=3)	76 ± 5	85 ± 3	**87 ± 2**	79 ± 15	81 ± 6	90 ± 4	**92 ± 2**	87 ± 10
Optdigits (N=5620, d=64)	Digit (k_j=10)	73 ± 1	**74 ± 1**	72 ± 2	70 ± 2	54 ± 4	58 ± 4	**66 ± 3**	64 ± 3

contrast, *AutoNR* often assigns the points to the correct clusters every time. This shows that the iterative identification of subspaces can be beneficial, with the effect becoming stronger the more subspaces there are.

9 Importance of Correct Parametrization

To better assess the importance of a correct parametrization of non-redundant clustering approaches, we perform another experiment. Suppose that we know that *NRLetters* comprises six different letters. We know nothing about the other clustering possibilities. Therefore, we try different parameters for *Nr-Kmeans* using a brute-force search. Here, we assume that no subspace contains more than six clusters. The NMI and F1 results can be seen in Figure 4. To arrive at a single number that indicates the quality of a non-redundant clustering result, we compute the average result over all three subspaces.

$$\text{score}(R^{gt}, R^p) = \frac{1}{J_{gt}} \sum_{1 \le j \le J_{gt}} score_j(R^{gt}, R^p),$$

where $score_j(R^{gt}, R^p)$ is the evaluation method as described in the paper and J_{gt} is the number of label sets in the ground truth. Each run is repeated ten times and the best result is added to the heatmap.

We see that the quality of the results deteriorates away from the optimum ($k_2 = 4, k_3 = 3$) even though we know the correct number of clusters in the first subspace. This shows the value of our framework, which achieved a perfect result in all ten iterations and that, without prior knowledge.

References

[1] D. Arthur and S. Vassilvitskii, *k-means++: The advantages of careful seeding*, in 18th ACM-SIAM SODA, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[2] S. Goebl, X. He, C. Plant, and C. Böhm, *Finding the optimal subspace for clustering*, in 14th IEEE ICDM, IEEE, 2014, pp. 130–139.

[3] S. Günnemann, I. Färber, M. Rüdiger, and T. Seidl, *Smvc: semi-supervised multi-view clustering in subspace projections*, in 20th ACM SIGKDD, 2014, pp. 253–262.

[4] J. Hu, Q. Qian, J. Pei, R. Jin, and S. Zhu, *Finding multiple stable clusterings*, Knowledge and Information Systems, 51 (2017), pp. 991–1021.

[5] T. C. Lee, *An introduction to coding theory and the two-part minimum description length principle*, International statistical review, 69 (2001), pp. 169–183.

[6] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Discovering non-redundant k-means clusterings in optimal subspaces*, in 24th ACM SIGKDD, 2018, pp. 1973–1982.

[7] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Non-redundant subspace clusterings with nr-kmeans and nr-dipmeans*, ACM Transactions on Knowledge Discovery from Data (TKDD), 14 (2020), pp. 1–24.

[8] X. Wang, J. Wang, C. Domeniconi, G. Yu, G. Xiao, and M. Guo, *Multiple independent subspace clusterings*, in AAAI, vol. 33, 2019, pp. 5353–5360.

[9] W. Ye, S. Maurus, N. Hubig, and C. Plant, *Generalized independent subspace clustering*, in 16th IEEE ICDM, IEEE, 2016, pp. 569–578.

C.2 Extension of the Dip-test Repertoire - Efficient and Differentiable p-value Calculation for Clustering

Authors:
Bauer, Lena G. M.* and Leiber, Collin* and Böhm, Christian and Plant, Claudia
** First authors with equal contribution*

Abstract:
Over the last decade, the Dip-test of unimodality has gained increasing interest in the data mining community as it is a parameter-free statistical test that reliably rates the modality in one-dimensional samples. It returns a so called Dip-value and a corresponding probability for the sample's unimodality (Dip-p-value). These two values share a sigmoidal relationship. However, the specific transformation is dependent on the sample size. Many Dip-based clustering algorithms use bootstrapped look-up tables translating Dip- to Dip-p-values for a certain limited amount of sample sizes. We propose a specifically designed sigmoid function as a substitute for these state-of-the-art look-up tables. This accelerates computation and provides an approximation of the Dip- to Dip-p-value transformation for every single sample size. Further, it is differentiable and can therefore easily be integrated in learning schemes using gradient descent. We showcase this by exploiting our function in a novel subspace clustering algorithm called Dip'n'Sub. We highlight in extensive experiments the various benefits of our proposal.

Venue:
Proceedings of the 2023 SIAM International Conference on Data Mining (SDM). Society for Industrial and Applied Mathematics, 2023.
CORE ranking: A (http://portal.core.edu.au/conf-ranks/1727/)
Acceptance rate: 27.4%

DOI:
https://doi.org/10.1137/1.9781611977653.ch13

Thesis-Reference:
[BLBP23]

Division of Work:

Joint work of the first authors
Development of the idea; Constant discussions on the procedure; Designing the experiments; Co-writing large parts of the paper

Lena G. M. Bauer
Main development of the function fit; Implementing the fitting functionality; Executing most experiments to evaluate the quality of the function fit and its runtime behaviour; Writing the majority of the related work regarding the Dip-test and bootstrapping, the methodology section regarding the table extension and the function fit, and the experiments section regarding reliable computation and computing time; Creating the visualizations regarding the function fit and Dip-p-value

Collin Leiber
Main development of the clustering procedure; Implementing the clustering algorithm; Creating the enlarged Dip-p-value table; Executing most experiments regarding the clustering algorithms; Writing the majority of the related work regarding Dip-test related data mining and common subspace clustering, the methodology section regarding Dip'n'Sub, and the experiments section regarding the Dip'n'Sub evaluation; Creating the visualizations regarding the clustering procedure; Creating the presentation slides and poster; Presenting at the conference

Christian Böhm
Formulating parts of the introduction; Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Claudia Plant
Identification of relevant related work; Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Extension of the Dip-test Repertoire - Efficient and Differentiable p-value Calculation for Clustering

Lena G. M. Bauer*,†,‡ Collin Leiber*,§ Christian Böhm† Claudia Plant†

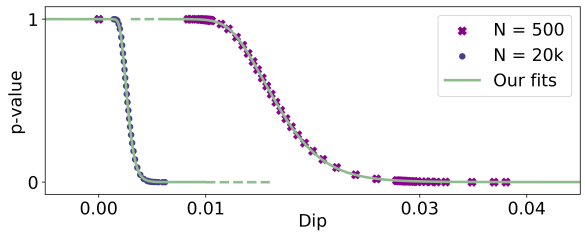
Abstract

Over the last decade, the Dip-test of unimodality has gained increasing interest in the data mining community as it is a parameter-free statistical test that reliably rates the modality in one-dimensional samples. It returns a so called Dip-value and a corresponding probability for the sample's unimodality (Dip-p-value). These two values share a sigmoidal relationship. However, the specific transformation is dependent on the sample size. Many Dip-based clustering algorithms use bootstrapped look-up tables translating Dip- to Dip-p-values for a certain limited amount of sample sizes. We propose a specifically designed sigmoid function as a substitute for these state-of-the-art look-up tables. This accelerates computation and provides an approximation of the Dip- to Dip-p-value transformation for every single sample size. Further, it is differentiable and can therefore easily be integrated in learning schemes using gradient descent. We showcase this by exploiting our function in a novel subspace clustering algorithm called Dip'n'Sub. We highlight in extensive experiments the various benefits of our proposal.

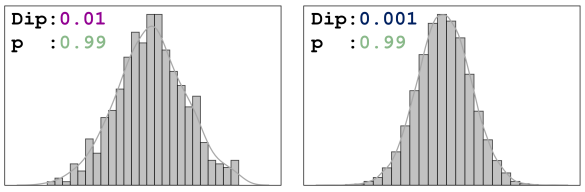
1 Introduction

One of the major goals in data mining is to automatically find meaningful patterns and structures in data. This is ideally done fast and with as few hyperparameters as possible. Here, the definition of statistical modes plays a decisive role in many approaches. Clustering methods like Meanshift [6] or Quickshift [28] try to find modes with the help of a predefined influence area and assign objects to these modes. Other procedures focus on evaluating whether a data set has a single or multiple modes. Examples are Hartigan's Dip-test [11], the Silverman test [27], the Folding Test [26] or the recently presented UU-test [5]. In clustering the assumption for these methods is that multiple modes indicate multiple clusters, while unimodality is a sign for a single cluster. The tests are basically parameter-free, which makes them particularly interesting for the data mining community. The most commonly used unimodality test in the clustering domain is probably the Dip-test. The input for the Dip-test is a one-dimensional sample and the test outputs a Dip-value $Dip \in (0, 0.25]$ that can be converted to a p-value, which we will term Dip-p-value throughout this work. The latter represents a probability for the sample to be unimodal and it is high for Dip-values close to zero and low for Dip-values close to its maximum of 0.25. Its benefits have already been exploited by several data mining techniques.

DipMeans [13], projected DipMeans [4], SkinnyDip [18], M-Dip [7], NrDipMeans [20] and DipDECK [16], for example, are all algorithms that use the Dip-test to



(a) Bootstrapped (Dip,p)-pairs and our fitted functions.



(b) $X \sim \mathcal{N}(0,1)$, $N = 500$. (c) $X \sim \mathcal{N}(0,1)$, $N = 20k$.

Figure 1: **(a)** Bootstrapped (Dip, p)-pairs for sample sizes $N = 500$ (purple) and $N = 20k$ (blue) and our fitted functions (green). The transformation function from Dip- to Dip-p-value strongly depends on the sample size. **(b)** and **(c)** Histograms of samples from a $\mathcal{N}(0, 1)$ normal distribution with sample size $N = 500$ and $N = 20k$. When applying the Dip-test on the two samples, their Dip-values differ with a factor of 10. The respective Dip-p-values are, however, 0.99 in both cases.

*Authors contributed equally.
†Faculty of Computer Science, ds:UniVie, University of Vienna, Vienna, Austria. {lena.bauer, christian.boehm, claudia.plant}@univie.ac.at
‡UniVie Doctoral School Computer Science.
§LMU Munich & MCML, Munich, Germany. leiber@dbs.ifi.lmu.de

estimate the number of clusters. Other algorithms like DipTransformation [25], DipExt [24] or the DipEncoder [15] utilise the Dip-test to create cluster-friendly lower-dimensional spaces. When executing these algorithms, a precise and efficient determination of the Dip- and Dip-p-value is of high relevance. In general, the Dip-value translates sigmoidal to the Dip-p-value. Figure 1a clearly shows the pattern. Importantly, we can observe that the specific curvatures and positions of the sigmoidal functions are heavily dependent on the number of samples N. Figures 1b and 1c show that the Dip-p-values p of a normal distribution $\mathcal{N}(0,1)$ with $N = 500$ and $N = 20k$ are both 0.99. This is in accordance to the expectation of the normal distribution to be unimodal irrespective of the sample size. The Dip-values *Dip*, however, differ by a factor of 10. Thus, considering the Dip-p-value is a far more robust choice when optimising for uni- or multimodality.

To the best of our knowledge, however, there exists no elegant method for the translation from Dip- to Dip-p-value. The state-of-the-art procedure is to utilise a look-up table with a limited amount of pairs of Dip- and Dip-p-values and use $\sqrt{N}$-interpolation to provide the missing pairs in between table values. While this approach has been used in multiple procedures (e.g., [4,13,16,18,20]), it shows drawbacks in several aspects. First, it is not differentiable and therefore harder to integrate into strategies such as stochastic gradient descent (SGD) and second, it is limited to the maximum bootstrapped sample size in the table. To resolve these shortcomings we propose a differentiable transformation function which - provided with a Dip-value and the number of samples - automatically calculates the corresponding Dip-p-value (see green lines in Fig. 1a). We showcase the practical value of our proposal for data mining research by developing a subspace clustering algorithm, that exploits the differentiability of our transformation function to identify a common subspace for all clusters in the data set. This helps to analyse relationships between clusters [10, 19] and distinguishes us from 'classical' subspace clustering algorithms (e.g., ORCLUS [1] or 4C [2]) which find a separate subspace for each cluster. Our idea uses the gradient of the Dip-p-value to identify projection axes where all clusters show a high degree of multimodality by performing SGD. This approach makes it particularly apt to rely on the Dip-p-value instead of the Dip-value as different cluster sizes bias the latter. Further, we present TailoredDip, an extended version of the one-dimensional clustering procedure UniDip [18], to cluster the data on those axes.

Our contributions can be summarised as follows:

- We provide a fully automatic translation from Dip- to Dip-p-value via an analytical function (Sec. 3.2)
- The translation is available for any data size N and provides reliable Dip-p-values irrespective of the underlying distribution (Sec. 4.1)
- Analyses show that our novel calculation is faster than previously used methods, in particular bootstrapping (Sec.4.2)
- We showcase how the differentiability of our function is useful for practical data mining applications by introducing our subspace clustering algorithm Dip'n'Sub (Sec. 3.3 and 4.3)

2 Related Work

2.1 The Dip-test The Dip-test is a statistical test for modality in a one-dimensional sample developed by Hartigan and Hartigan [11]. The test returns a so-called Dip-value *Dip* that specifies the distance between the Empirical Cumulative Distribution Function (ECDF) of the sample to an unimodal piece-wise linear function, i.e., a function that is convex up to the beginning of the steepest slope and concave thereafter. In this context, the area of steepest slope is often called *modal interval*. By definition $Dip \in (0, 0.25]$, where a *Dip* close to 0 indicates a unimodal distribution and a $Dip \gg 0$ indicates a multimodal distribution. The exact value not only depends on the specific distribution but also on the sample size (see Fig. 1b and 1c). For this reason, Dip-values are often not used directly, but the associated Dip-p-values. Here, the null hypothesis $\mathcal{H}_0$ states that the sample set is unimodal and the alternative hypothesis $\mathcal{H}_1$ is that there are at least two modes present in the data. The test does not make any assumptions about the data generating distribution. For any distribution with a single mode, whether it is Gaussian, Laplacian or t-distributed the test will not reject $\mathcal{H}_0$. The Dip-test can be calculated efficiently in $O(N)$ [11] on a sorted input of size N. Furthermore, [14] showed that the Dip-value is differentiable with respect to a projection axis. This enables the use of SGD. Due to these several benefits, the Dip-test has successfully been integrated in several data mining applications over the last decade.

2.2 Bootstrapping The Dip-test can convert a Dip-value to a p-value representing an evidence measure for the credibility of the null hypothesis. The original work by Hartigan and Hartigan [11] provides a table with bootstrapped Dip-values and corresponding Dip-p-values for samples of different sizes N. The table lists the relationships for 13 different sample sizes ranging between 4 and 200 with 9 (Dip, p)-pairs each. These were calculated by drawing N random samples from a uniform distribution several times. The Dip-test was

then performed on each of these sample sets. The percentage of sets with a Dip-value smaller than the input Dip-value thus gives the respective Dip-p-value. The idea is that the uniform distribution represents a borderline case between unimodal and multimodal distributions [11]. In previous publications authors used bootstrapping to enlarge Hartigan and Hartigan's table to a total of 21 sample sizes up to 72,000 data points with 26 (Dip, p)-pairs for each sample size. They use interpolation of $\sqrt{N}$ to get intermediate Dip-p-values. To the best of our knowledge this approach is the state-of-the-art when transforming Dip- to Dip-p-value.

2.3 Dip-test Related Data Mining Most clustering methods utilising the Dip-test are dedicated to the estimation of the number of clusters in a data set. Here, an essential component is the transformation of the data into a one-dimensional space where the Dip-test can be applied. One of the first approaches is DipMeans [13], which uses the Dip-test to determine if the distances between data points within a cluster are distributed unimodally. For projected DipMeans [4] the input for the Dip-test are the data points itself, after they have been projected onto projection axes. Another transformation is applied by M-Dip [7]. Here, the Dip-test is executed on the path of closely located points between two clusters. NrDipMeans [20] estimates the number of clusters in a non-redundant clustering setting by employing a Dip-based splitting strategy. The first clustering and k-estimation technique using the Dip-test in a deep learning context is DipDECK [16], which uses the test to decide whether clusters should be merged or not. SkinnyDip [18] determines clusters in highly noisy data sets by running its subroutine UniDip on each feature of each cluster. UniDip recursively executes the Dip-test to identify modal intervals (see Sec. 2.1) in the one-dimensional data set until all intervals themselves and the areas left and right (until the next interval) are considered unimodal. Finally, each modal interval is considered a cluster and all objects that do not fall within such an area are classified as noise. The idea of SkinnyDip is later used to cluster streaming data with StrDip [17]. All of the mentioned procedures need to decide whether samples are distributed unimodally or not and, therefore, use the same bootstrapped look-up table to convert Dip-values into probabilities.

An example for a Dip-based pre-processing algorithm is DipTransformation [25]. It scales and transforms a data set, such that the resulting feature space is suitable for k-means. In DipExt [24] this idea is extended by making use of the differentiability of the Dip-value with respect to the projection axis. Structure-rich features are extracted from the data by searching for suitable projection axes with SGD. The DipEncoder [15] uses the gradient of the Dip-value to train an autoencoder in such a way that each combination of clusters projected onto their specific projection axis is highly multimodal in the embedding.

2.4 Common Subspace Clustering Traditional subspace clustering algorithms like 4C [2] or ORCLUS [1] define an individual subspace for each cluster. In this setting, however, the inter-cluster relationships are difficult to analyse [10]. We would therefore like to identify a common subspace for all clusters. A simple possibility to find such subspaces are dimensionality reduction techniques such as Principal Component Analysis (PCA) [21], Independent Component Analysis (ICA) [12], Linear Discriminant Analysis (LDA) [9] or the already mentioned DipExt [24] algorithm. Since in these cases possible cluster assignments do not influence the final subspace, special common subspace clustering algorithms like LDA-k-means [8], FOSSCLU [10] and SubKmeans [19] have been developed. LDA-k-means and SubKmeans utilise LDA or an eigenvalue decomposition, respectively, in combination with the k-means objective to determine the subspaces. FOSSCLU combines the EM algorithm with rigid transformations.

3 Methods

In this section, we design a special sigmoid function converting Dip- to Dip-p-values. This function's differentiability is then exploited by our subspace clustering algorithm Dip'n'Sub.

3.1 Table Extension To increase the granularity level of the look-up table previously used in literature, we also use bootstrapping as proposed by [11]. We sample from uniform distributions with 100,000 repetitions to obtain a Dip-p-value table containing 307 (Dip, p)-pairs for 63 sample sizes up to a sample size of 150,000 data points. This look-up table provides a good basis, but it does not cover all values of sample sizes N. Therefore, we fit a logistic function $p(\cdot)$ to provide a differentiable solution for this issue. Fig. 3 visually captures Sec. 3.1 and Sec. 3.2.

3.2 Function Fit A good fitting function provides high flexibility to fit the different sigmoidal relationships between Dip- and Dip-p-value for all sample sizes N while using the smallest possible number of parameters. Due to the sigmoidal behaviour it is reasonable to approximate it with a generalised logistic function [22].

$$l(Dip) = d + \frac{a-d}{\left(c + h \cdot e^{-b \cdot Dip}\right)^{1/g}}$$

The parameters a and d represent the upper and lower asymptote, respectively, and need to be fixed as $a = 1$ and $d = 0$ in our application as we model probability values. In addition, $c = 1$ must hold for the function to actually be constrained by 1. Further, often $h = g$ applies (see e.g. [3]). To obtain a negative slope, b needs to be positive. Another requirement is that $g > 0$ holds. This essentially leaves the parameters $h = g$ and b, which mainly determine the curvatures. However, we also need to adjust the scale for the x-axis since the Dip-value is limited within $(0, 0.25]$. Therefore, we have to include a shift parameter in the exponential function. Finally, we want to account for highly different curvatures at the two asymptotes and therefore include a weighting, which is partly inspired by the work of [23]. Our final fitting function reads as follows:

$$p(x, \theta_N) = 1 - \Big[w_N \cdot (1 + h_N \cdot e^{-q_N \cdot x + s_N})^{1/h_N} + (1 - w_N) \cdot (1 + k_N \cdot e^{-r_N \cdot x + u_N})^{1/k_N} \Big]^{-1},$$

with the independent variable x and where $\theta_N = (w_N, h_N, k_N, q_N, r_N, s_N, u_N)$ is the set of parameters. We then optimise θ_N with respect to the mean squared error between the fitting function and our enlarged table values for each sample size N separately. We find that most of the parameters are approximately constant across sample sizes N. Thus, we set them to the mean value across all N to reduce the number of parameters. Hence, it is sufficient to set w_N, h_N, k_N, s_N and u_N as constants and further set $q_N = r_N = b_N$ as the one remaining parameter depending on the sample size N. The resulting optimal values for b_N are shown in Fig. 2 (teal stars). As they visually resemble a square root function of N, we model the parameter b_N as the following function of N:

$$b(N) = b_1 \cdot \sqrt{N} + b_2.$$

Note, that b_1 and b_2 are parameters independent of N. The final optimisation is to find b_1 and b_2 such that

$$\mathcal{E}(b_1, b_2) = \frac{1}{|S|L} \sum_{N \in \mathcal{S}} \sum_{i=1}^{L} (p_{Dip_i, N} - \hat{p}(Dip_i, \hat{b}(N)))^2$$

is minimal. Here, $\mathcal{S}$ is the set of all sample sizes N, where $|\mathcal{S}| = 63$ in the case of our enlarged table, and L is the number of (Dip, p)-pairs in the extended table, here $L = 307$. The optimal function within our optimisation scheme with respect to the mean squared error and with

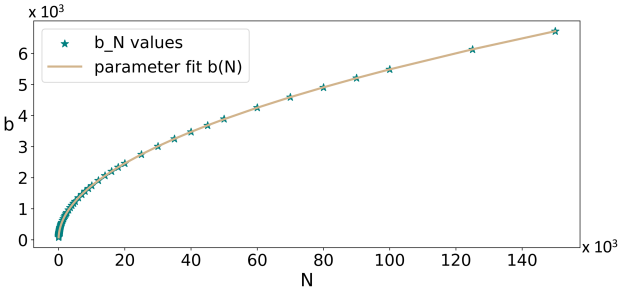


Figure 2: Our fitted function for the parameter b_N.

our bootstrapped table as data to be fitted is given by

$$\hat{p}(x, \hat{b}(N)) = 1 - \Big[0.6 \cdot (1 + 1.6 \cdot e^{-\hat{b}(N) \cdot x + 6.5})^{1/1.6} + 0.4 \cdot (1 + 0.2 \cdot e^{-\hat{b}(N) \cdot x + 6.5})^{1/0.2} \Big]^{-1},$$

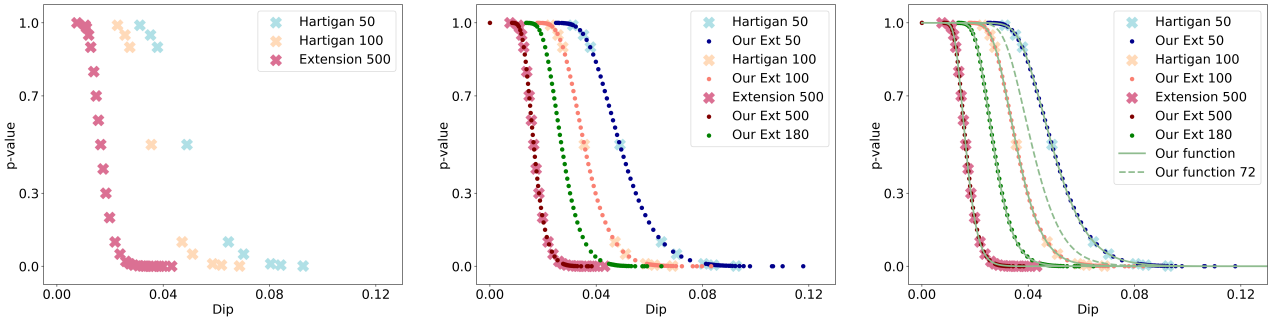
and

$$\hat{b}(N) = 17.30784 \cdot \sqrt{N} + 12.04918$$

where values for $\hat{b}_1 = 17.30784$, $\hat{b}_2 = 12.04918$ are rounded to five decimal places and correspond to the optimal estimators for b_1 and b_2. The concatenated function $\hat{p}(x, \hat{b}(N))$ is smooth and well-defined for all $N \in \mathbb{N}^+$ and all $x \in (0, 0.25]$.

Derivation: One of the benefits of our proposal is, that the fitted function is differentiable. This property is exploited in our later discussed subspace clustering algorithm. The Dip-test can only be applied to a one-dimensional sample, which is why we always consider a projection axis ρ for d-dimensional data sets X. Further, the data has to be sorted. Thus, the Dip-value is returned for the sorted and projected data $\overline{X}_\rho$. From [14] we know that we can calculate the gradient vector of the Dip-value on $\overline{X}_\rho$ with respect to the projection axis ρ. We will term this gradient $\nabla_\rho(Dip(\overline{X}_\rho))$. Details about the calculation of this gradient can be found in [14, 15, 18, 24]. By using our differentiable sigmoid function to calculate the Dip-p-value of the Dip, the Dip-p-value is also continuously differentiable as a concatenation of differentiable functions. The gradient is the following, where we define $D := Dip(\overline{X}_\rho)$ and $b := \hat{b}(N)$ for easier readability:

$$\nabla_\rho(\hat{p}(D, b)) = \big(- b \nabla_\rho(D) \big) e^{-bD+6.5} \cdot \Big[0.6(1 + 1.6e^{-bD+6.5})^{\frac{1}{1.6}} + 0.4(1 + 0.2e^{-bD+6.5})^{\frac{1}{0.2}} \Big]^{-2} \cdot \Big[0.6(1 + 1.6e^{-bD+6.5})^{\frac{-0.6}{1.6}} + 0.4(1 + 0.2e^{-bD+6.5})^{\frac{0.8}{0.2}} \Big]$$



(a) Examples of (Dip, p)-pairs from the original table [11] and an example of an extensions for $N = 500$

(b) We bootstrap with larger granularity for N as well as (Dip, p)-pairs.

(c) Our differentiable fitted function closes the remaining granularity gaps for all N and all (Dip, p)-pairs.

Figure 3: **(a)** Hartigan and Hartigan's original bootstrapped table only provides pairs of Dip- and Dip-p-values for 13 different sample sizes. This table has been extended to values of $N \geq 500$. **(b)** We enlarge the table for even more values of N as well as a larger granularity regarding (Dip, p)-pairs (for better visualisation, we down-sampled our table to every third point). **(c)** We close the remaining gaps by providing our fitted function, such as for $N = 72$, for which we do not have bootstrapped values.

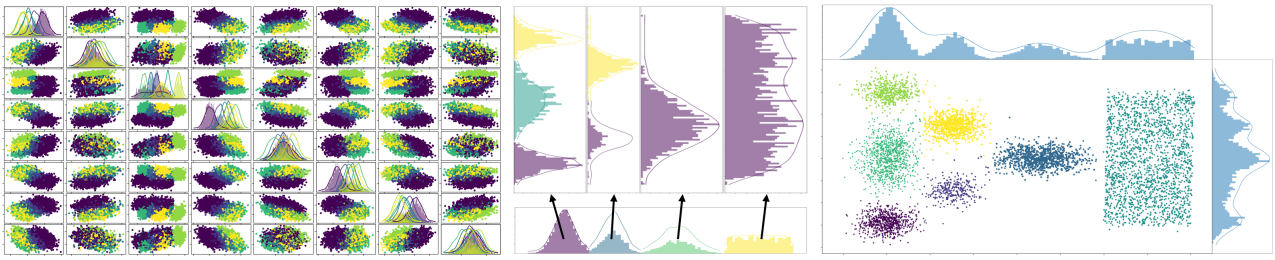
3.3 Dip'n'Sub We aim to show with a proof-of-concept that our differentiable Dip-p-value function has great value for the data mining community. Therefore, we present the subspace clustering algorithm Dip'n'Sub which is solely based on the Dip-test. It is able to automatically define a lower-dimensional subspace and also to derive the number of clusters. For this, only a significance threshold is necessary, which indicates whether a distribution is unimodal. We first propose TailoredDip, a new extension of UniDip [18].

A problem with UniDip is that the tails of distributions are very generously identified as outliers. TailoredDip is able to better capture those tails. We achieve this by checking the spaces between the clusters for additional structures after the regular UniDip algorithm has terminated. Therefore, we mirror the respective area between two clusters and calculate the Dip-p-value. If this indicates multimodal structures, we identify appropriate modes and assign those points to the best fitting neighbouring cluster. Further, if outlier detection is not desired we use the following strategy to assign them either to the left or the right cluster: Instead of simply defining the mid point between neighbouring clusters as a decision boundary, we choose the point that corresponds to the intersection of the ECDF and the line between the right limit of the left cluster and the left limit of the right cluster. This handles different tails more accurately. Details about TailoredDip as well as a pseudocode are given in the supplement (Sec. 1).

Now that we can find clusters in one-dimensional samples, let us consider the multidimensional case. Here, we use the fact that SGD can be used to find a projection axis on which a data set shows a minimum Dip-p-value. A naive approach would be to recursively select each cluster and, using the points of that cluster, find the projection axis on which those samples exhibit the greatest multimodality. The problem here is twofold. First, there is almost always some degree of multimodality in a set of objects, which means that one will identify a very high number of clusters. Second, the individual clusters are difficult to compare with each other, since each cluster forms its own subspace. Therefore, we want to successively identify those features in which as many objects as possible are contained in highly multimodal clusters. Thus, we recursively search for projection axes ρ that minimise the following term:

$$\frac{1}{N}\sum_{i=1}^{k} |C_i|\hat{p}\big(\mathrm{Dip}(\overline{C_i^\rho}), \hat{b}(|C_i|)\big), \tag{3.1}$$

where C_i are the samples in cluster i and $\overline{C_i^\rho}$ are the same samples projected to ρ and sorted afterwards, i.e. $\overline{C_i^\rho} = sort\{\rho^T c | c \in C_i\}$. To find these axes, we are inspired by [24]. We start with the q features that show the lowest sum of Dip-p-values weighted by their cluster sizes and use them as starting points for SGD with momentum. Further, we start at the first q components of a PCA. [24] has shown that $q = \log(d)$ is sufficient, where d is the original number of features. Using these $2q$ starting axes, we iteratively calculate the gradient with respect to all clusters. Here, we exploit that our fitted function enables us to directly calculate the gradient of Eq. 3.1. Having identified the best projection axis, we check whether more than



(a) The input dataset.

(b) The two projections identified by Dip'n'Sub.

(c) The final clustering result of Dip'n'Sub.

Figure 4: **(a)** Scatter matrix plot of an 8-dimensional synthetic data set (colours correspond to ground-truth labels). **(b)** The horizontal histogram below illustrates the first projection identified by Dip'n'Sub. The data is highly multimodal and therefore divided into 4 clusters. Dip'n'Sub now uses these cluster assignments to identify a second projection in which all clusters are as multimodal as possible. The second projection is shown vertically in the upper histograms, with respect to each existing cluster individually. It is easy to see that the first two clusters (purple and blue) are subdivided into 3 and 2 clusters, respectively. Thereafter, no multimodal third projection can be found. **(c)** The final clustering result of Dip'n'Sub reveals a clear separation of the clusters.

T% of the objects are contained in clusters considered multimodal on this axis, where T has to be set by the user. If this is the case, we apply TailoredDip to this axis. Each cluster is considered individually and divided into several sub-clusters. The clusters thereby form hypercubes in the final feature space. Our method Dip'n'Sub is presented in Algorithm 1 and an example of the subspace identification process is shown in Fig. 4.

4 Experiments and Results

We will show the several benefits of our proposal in three main experimental sections. First, we show, that our calculated Dip-p-values are as reliable as the ones with the look-up table. Then we present runtime experiments that prove our method to be efficient, not only in 'laboratory conditions', but also in practice when integrated in existing methods using Dip-p-values. Finally, we evaluate our subspace clustering algorithm Dip'n'Sub to showcase the integration of the gradient of the Dip-p-value in a practical data mining application.

Our supplement, codes, enlarged $(Dip,\ p)$-pairs table and the used data sets are available at: `https://dx.doi.org/10.6084/m9.figshare.21916752`.

4.1 Reliable Computation One important advantage of our fitted function is, that it provides Dip-p-values for all sample sizes N. Table 1 shows that our 'function' method is consistent with the look-up 'table' and 'bootstrap' methods as we produce basically the same Dip-p-values, not only for different unimodal distributions ($\mathcal{N}(4,1)$ = normal distribution with centre 4 and variance 1; $\mathcal{T}_{nc}(4,2,0,1)$ = non central student's t-distribution with 4 degrees of freedom, non-centrality parameter 2, centre 0 and scaling 1; $\mathcal{L}(0,2)$ = Laplace distribution with centre 0 and scaling 2), but also for multimodal distributions, which we create by combining samples of the same unimodal distribution, but with a different centre. In the supplement (Sec. 2.3) we show tables with a total of 23 distributions, where we observe the same behaviour.

Comparing to the enlarged bootstrapped table described in Sec. 3.1 our function performs better than the 'table' method, with the quality of both methods being measured in mean squared error (MSE). In this case, bootstrapping can be seen as some kind of ground truth, however it is not practical as runtime is a major issue, especially for large N. We will see this in more detail in the next section. Regarding the 'table' method, errors for sample sizes greater than $72,000$ had to be ignored for the calculation of this MSE, because it cannot provide any Dip-p-values in that case. While we achieve an MSE of $3.43 \cdot 10^{-6}$ for all N, the 'table' results in an MSE of $7.92 \cdot 10^{-6}$. To check how well our function generalises with respect to N, we calculate the MSE for a set of N chosen as the mean values between each two N of our enlarged table. Those 62 values were not used for our function fit. The result is an MSE of $3.14 \cdot 10^{-6}$ for 'function' and $8.12 \cdot 10^{-6}$ for 'table'. Hence, we outperform the 'table' method in both cases.

4.2 Computing Time In the first experiment, we sum up the runtimes of all the Dip-p-value calculations for the 23 distribution cases shown in the supplement (Table 1 is a selection of six of them). These are shown in Fig. 5. Note, that the y-scale is logarithmic. Note also the missing value for $N = 100k$ for the 'table'

Algorithm 1: The Dip'n'Sub algorithm

Input: data set X, significance α, threshold T
Output: labels

$k = 1; labels = [0, \dots, 0]; X_{fin} = []$
while *True* **do**
  $s = 1; \rho = \vec{0}$
  $Q = \log(d)$ features with lowest weighted p-values $\cup$ first $\log(d)$ components of PCA
  for *each* $\rho_{tmp} \in Q$ **do**
    Update ρ_{tmp} with SGD using Eq. 3.1
    s_{tmp} = value of Eq. 3.1 using ρ_{tmp}
    if $s_{tmp} < s$ **then**
      $s = s_{tmp}; \rho = \rho_{tmp}$
  $P = \{\text{p-value}(\text{Dip}(\overline{C_i^\rho}), |\overline{C_i^\rho}|) \mid i \in [1, k]\}$
  if $\frac{sum\{|C_i| \mid i \in [1,k] \wedge P_i < \alpha\}}{N} \geq T$ **then**
    for *each cluster* i **with** $P_i < \alpha$ **do**
      $labels_{new} = \text{TailoredDip}(\overline{C_i^\rho}, \alpha)$
      update *labels* using $labels_{new}$
    X_{fin} = combine X_{fin} and $\{\rho^T x | x \in X\}$
    X = keep features orthogonal to ρ
  else
    break
return $labels, X_{fin}$

method as here the calculation of Dip-p-values is not possible. We can observe, that the calculation of Dip-p-values is fastest with our function, although it should be noted that the speed-up relative to the 'table' method is only marginal and could be due to implementation details. The differences concerning the calculation time become of practical value, when the number of Dip-p-values to be calculated gets larger. In Table 2 we can see how the algorithms DipMeans, projected DipMeans and SkinnyDip have decreasing runtime, when our method is used instead of the look-up table or bootstrapping. Information about the data sets are given in the supplement (Sec. 2.2). As expected, our function method does not degrade the clustering performance. Across all three algorithms and all data sets, the normalised mutual information (NMI) remains stable with an average difference between 'table' and 'function' of $9.00 \cdot 10^{-3}$ and $1.17 \cdot 10^{-2}$ between 'function' and 'bootstrap'. Runtime is comparable to the 'table' method and improves notably compared to 'bootstrapping' as we save 50%, 92% and 99% for DipMeans, projected DipMeans and SkinnyDip, respectively.

4.3 Dip'n'Sub Evaluation We evaluate our algorithm Dip'n'Sub and competitors in terms of clustering performance using the normalised mutual information (NMI). This score attains values between 0 and 1, where 0 indicates a purely random label assignment and a value close to 1 is a perfect clustering result.

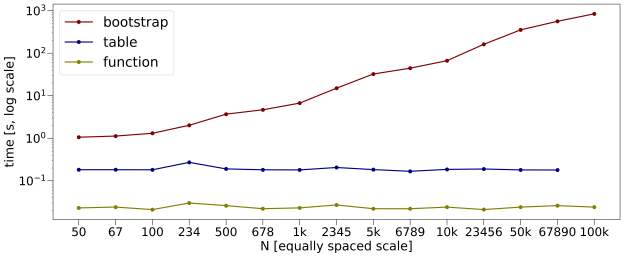


Figure 5: Runtime in seconds [s] on a logarithmic scale for the calculations of 100 Dip-p-values per sample size with the three methods 'bootstrap', 'table' and 'function' summed up over all 23 distribution scenarios as described in the supplement (Sec. 2.3).

Comparison Methods: We compare to multiple approaches that define a common subspace for all clusters. This includes dimensionality reduction methods like PCA, ICA or DipExt, which we combine with k-means, and the algorithms LDA-k-means, FOSSCLU and SubKmeans. Furthermore, since Dip'n'Sub is able to estimate the number of clusters, we compare with the Dip-based k-estimation methods DipMeans, projected DipMeans and SkinnyDip.

For PCA, we set the number of components such that 90% of the variance is preserved, and for ICA, it equals k. The significance level is set to 0.01 for all Dip-based methods. The range in which FOSSCLU can determine the number of subspace dimensions with MDL is defined as $[1, 5]$. All other parameters were set as described in the respective papers. Regarding Dip'n'Sub we set $T = 0.15$ and the significance to 0.01. For SGD, we choose a momentum of 0.95 and a step-size of 0.1 (for USER, ALOI, AIBO, SYMB, OLIVE) or 0.01 (for SYNTH, BANK, HTRU2, MICE, MOTE). Since none of the above data sets contains outliers, we assign all points to the best matching cluster by using the strategy described in Sec. 3.3.

Quantitative Analyses: Table 3 shows the results of our algorithm Dip'n'Sub and our competitors on a wide range of data sets (see supplement Sec. 2.2 for details). Note, that compared to the other subspace algorithms we do not know the ground truth number of clusters. Nevertheless, we are competitive compared to subspace and Dip-based k-estimation methods as we rank first 4 times and second 5 times in terms of NMI. On ALOI, we are slightly inferior, but we only need a single feature for our result. Overall, Dip'n'Sub achieves

Table 1: Dip-p-values for different unimodal (**left**) and multimodal (**right**) distributions with varying sample sizes N. All given values are averages for 100 random samples ± standard deviation. Respective first, second and third rows per distribution show Dip-p-values calculated with methods 'table' (T), 'function' (F) and 'bootstrapping' (B, 1000 repetitions). Dip-p-values for multimodal distributions are multiplied by 100; $*$: values obtained by $\sqrt{N}-$ interpolation, †: values not available.

unim. Distr.	Method	$N = 50$	$N = 234$	$N = 2345$	$N = 100k$	**multim. Distr.**	$N = 50$	$N = 234$	$N = 2345$	$N = 100k$
$\mathcal{N}(4,1)$	T	0.77 ± 0.24	$0.86 \pm 0.19^*$	$0.97 \pm 0.07^*$	†	$\mathcal{N}(4,1)$	8.94 ± 15.0	$0.06 \pm 0.23^*$	$0.00 \pm 0.00^*$	†
	F	0.77 ± 0.24	0.86 ± 0.19	0.97 ± 0.07	1.00 ± 0.02	$\cup$	8.83 ± 14.9	0.09 ± 0.25	0.00 ± 0.00	0.00 ± 0.00
	B	0.77 ± 0.24	0.86 ± 0.19	0.97 ± 0.07	1.00 ± 0.02	$\mathcal{N}(0,1)$	8.78 ± 15.0	0.06 ± 0.21	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}_{nc}(4,2,0,1)$	T	0.80 ± 0.21	$0.89 \pm 0.14^*$	$0.98 \pm 0.03^*$	†	$\mathcal{T}_{nc}(4,2,0,1)$	0.79 ± 2.20	$0.00 \pm 0.00^*$	$0.00 \pm 0.00^*$	†
	F	0.80 ± 0.21	0.90 ± 0.14	0.98 ± 0.03	1.00 ± 0.00	$\cup$	0.83 ± 2.07	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	B	0.80 ± 0.21	0.90 ± 0.14	0.99 ± 0.03	1.00 ± 0.00	$\mathcal{T}_{nc}(4,2,7,1)$	0.74 ± 2.17	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{L}(0,2)$	T	0.85 ± 0.19	$0.95 \pm 0.11^*$	$0.99 \pm 0.04^*$	†	$\mathcal{L}(0,2)$	24.1 ± 24.9	$2.36 \pm 7.87^*$	$0.00 \pm 0.00^*$	†
	F	0.85 ± 0.19	0.95 ± 0.11	0.99 ± 0.04	1.00 ± 0.00	$\cup$	23.9 ± 25.1	2.37 ± 7.82	0.00 ± 0.00	0.00 ± 0.00
	B	0.85 ± 0.19	0.95 ± 0.11	0.99 ± 0.04	1.00 ± 0.00	$\mathcal{L}(7,2)$	23.8 ± 24.8	2.34 ± 8.06	0.00 ± 0.00	0.00 ± 0.00

Table 2: Average NMI and runtime (RT - in seconds) results for DipMeans, p. DipMeans and SkinnyDip using the Dip-p-value calculation methods 'table' (T), 'function' (F) and 'bootstrap' (B) after 10 runs.

Dataset	DipMeans						p. DipMeans						SkinnyDip					
	NMI			RT			NMI			RT			NMI			RT		
	T	F	B	T	F	B	T	F	B	T	F	B	T	F	B	T	F	B
SYNTH	0.64	0.64	0.64	6.87	5.82	8.92	0.86	0.86	0.85	0.76	0.74	9.03	0.16	0.16	0.16	0.02	0.01	14.66
BANK	0.31	0.31	0.30	13.00	6.93	43.71	0.30	0.31	0.30	7.70	5.46	69.81	0.13	0.13	0.13	0.01	0.00	2.56
USER	0.00	0.00	0.00	0.05	0.02	0.08	0.34	0.34	0.35	1.54	1.13	11.37	0.15	0.15	0.15	0.00	0.00	0.68
HTRU2	0.00	0.00	0.00	28.87	28.32	29.03	0.17	0.17	0.17	2.20	2.18	19.05	0.08	0.08	0.08	0.04	0.04	40.78
ALOI	0.96	0.96	0.92	0.43	0.22	0.84	0.49	0.51	0.49	9.03	6.68	54.36	0.17	0.15	0.17	0.06	0.04	3.68
MICE	0.00	0.00	0.00	0.19	0.12	0.27	0.53	0.54	0.53	91.47	50.21	607.21	0.00	0.00	0.00	0.01	0.01	3.76
AIBO	0.00	0.00	0.00	0.09	0.04	0.14	0.28	0.33	0.27	20.02	5.33	129.60	0.02	0.02	0.02	0.02	0.01	3.29
MOTE	0.35	0.35	0.35	0.85	0.62	1.21	0.00	0.00	0.00	0.03	0.02	0.37	0.00	0.00	0.00	0.03	0.02	8.73
SYMB	0.82	0.82	0.82	1.50	1.15	2.23	0.70	0.74	0.70	36.70	5.64	73.69	0.02	0.02	0.02	0.11	0.10	5.16
OLIVE	0.50	0.50	0.50	0.06	0.04	0.16	0.64	0.52	0.64	1.26	0.15	3.08	0.04	0.04	0.04	0.03	0.02	0.39

a good ratio of NMI to the identified number of cluster-relevant features. Our method identifies rather small subspaces (maximum number of features is 3 for BANK, MICE and SYMB), which in combination with the good NMI values suggests that those features are particularly relevant for clustering. This can be interesting for a visual evaluation of the results especially in the unsupervised domain. Furthermore, our estimation of k is notably better than that of other Dip-based methods (which we outperform every time with regard to NMI except for OLIVE). While those procedures identify the correct number of clusters only once, we manage it in 50% of the cases. Especially projected DipMeans seems to heavily overestimate the number of clusters. This confirms our hypothesis that in many cases one can detect additional multimodal structures when looking at a single cluster. Therefore, we benefit from considering only those projection axes relevant to all clusters.

5 Conclusion

In this paper, we propose a differentiable function to translate Dip-values to Dip-p-values. This provides an automatic and fast translation for any desired sample size. We show that our method is effective as our Dip-p-values show lower squared errors than previously used look-up tables. Further, it is efficient regarding computing time. Finally, we underpin its practical relevance by integrating our function in the subspace clustering algorithm Dip'n'Sub. Here, we show how our proposal enables the use of gradient descent for the Dip-test's p-value. Future efforts may attempt to integrate those ideas into deep learning applications.

References

[1] C. C. Aggarwal and P. S. Yu, *Finding generalized projected clusters in high dimensional spaces*, SIGMOD, ACM, 2000, p. 70–81.

[2] C. Böhm, K. Kailing, P. Kröger, and A. Zimek, *Computing clusters of correlation connected objects*, SIGMOD, ACM, 2004, p. 455–466.

[3] L. Cao, P.-J. Shi, L. Li, and G. Chen, *A new flexible sigmoidal growth model*, Symmetry, 11 (2019), p. 204.

[4] T. Chamalis and A. Likas, *The projected dip-means clustering algorithm*, SETN, ACM, 2018.

Table 3: Maximum NMI results of different common subspace and Dip-based k-estimation algorithms after 10 runs. The resulting number of clusters and dimensions is given in brackets. Best result in bold, runner-up dotted. (k = number of clusters, d = data set dimensionality, KM = k-means, †: no results due to non-trivial errors).

Dataset (k/d)	Common Subspace Algorithms							Dip-based k-estimation Algorithms		
	Dip'n'Sub	PCA+KM	ICA+KM	DipExt+KM	LDA-KM	SubKM	FOSSCLU	DipMeans	p. DipMeans	SkinnyDip
SYNTH (7/8)	**0.97** (7/2)	0.87 (7/4)	0.38 (7/7)	0.69 (7/1)	0.88 (7/6)	0.87 (7/6)	0.90 (7/5)	0.64 (3/8)	0.92 (6/8)	0.16 (2/8)
BANK (2/4)	0.41 (7/3)	0.03 (2/2)	0.01 (2/2)	**0.83** (2/1)	0.01 (2/1)	0.03 (2/1)	0.01 (2/4)	0.32 (41/4)	0.32 (36/4)	0.13 (2/4)
USER (4/5)	0.52 (10/1)	0.43 (4/5)	0.03 (4/4)	0.40 (4/2)	0.49 (4/3)	0.46 (4/3)	**0.65** (4/2)	0.00 (1/5)	0.36 (33/5)	0.15 (2/5)
HTRU2 (2/8)	**0.38** (3/2)	0.03 (2/2)	0.30 (2/2)	0.03 (2/1)	0.28 (2/1)	0.03 (2/1)	0.32 (2/5)	0.00 (1/66)	0.18 (7/66)	0.08 (1/66)
ALOI (4/66)	0.98 (4/1)	**1.00** (4/35)	**1.00** (4/4)	**1.00** (4/5)	**1.00** (4/3)	**1.00** (4/3)	†	0.96 (5/66)	0.52 (69/66)	0.17 (16/66)
MICE (8/68)	0.55 (6/3)	0.27 (8/6)	0.39 (8/8)	**0.59** (8/3)	†	0.29 (8/7)	0.33 (8/5)	0.00 (1/68)	0.54 (278/68)	0.00 (1/68)
AIBO (2/70)	**0.68** (2/1)	**0.68** (2/20)	0.56 (2/2)	0.60 (2/3)	0.30 (2/1)	**0.68** (2/1)	0.52 (2/5)	0.00 (1/70)	0.34 (35/70)	0.02 (3/70)
MOTE (2/84)	0.37 (2/1)	0.30 (2/42)	0.37 (2/2)	**0.41** (2/2)	0.28 (2/1)	0.35 (2/1)	0.08 (2/5)	0.36 (3/84)	0.00 (1/84)	0.00 (1/84)
SYMB (6/398)	**0.84** (5/3)	0.80 (6/6)	0.79 (6/6)	0.65 (6/2)	0.80 (6/5)	0.80 (6/5)	†	0.82 (5/398)	0.74 (16/398)	0.02 (2/398)
OLIVE (4/570)	0.57 (4/2)	0.68 (4/4)	0.69 (4/4)	0.73 (4/58)	†	**0.75** (4/3)	0.16 (4/3)	0.50 (2/570)	0.68 (9/570)	0.04 (1/570)

[5] P. Chasani and A. Likas, *The uu-test for statistical modeling of unimodal data*, Pattern Recognition, 122 (2022), p. 108272.

[6] Y. Cheng, *Mean shift, mode seeking, and clustering*, IEEE transactions on pattern analysis and machine intelligence, 17 (1995), pp. 790–799.

[7] P. Chronis, S. Athanasiou, and S. Skiadopoulos, *Automatic clustering by detecting significant density dips in multiple dimensions*, in ICDM, IEEE, 2019, pp. 91–100.

[8] C. H. Q. Ding and T. Li, *Adaptive dimension reduction using discriminant analysis and* K*-means clustering*, in ICML, vol. 227, ACM, 2007, pp. 521–528.

[9] R. A. FISHER, *The statistical utilization of multiple measurements*, Annals of Eugenics, 8 (1938), pp. 376–386.

[10] S. Goebl, X. He, C. Plant, and C. Böhm, *Finding the optimal subspace for clustering*, in ICDM, IEEE, 2014, pp. 130–139.

[11] J. A. Hartigan and P. M. Hartigan, *The dip test of unimodality*, Ann. Statist., 13 (1985), pp. 70–84.

[12] A. Hyvärinen and E. Oja, *Independent component analysis: algorithms and applications*, Neural Networks, 13 (2000), pp. 411–430.

[13] 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, Curran Associates, Inc., 2012.

[14] A. Krause and V. Liebscher, *Multimodal projection pursuit using the dip statistic*, (2005).

[15] C. Leiber, L. G. M. Bauer, M. Neumayr, C. Plant, and C. Böhm, *The dipencoder: Enforcing multimodality in autoencoders*, in SIGKDD, ACM, 2022, p. 846–856.

[16] C. Leiber, L. G. M. Bauer, B. Schelling, C. Böhm, and C. Plant, *Dip-based deep embedded clustering with k-estimation*, SIGKDD, ACM, 2021, p. 903–913.

[17] Y. Luo, Y. Zhang, X. Ding, X. Cai, C. Song, and X. Yuan, *Strdip: A fast data stream clustering algorithm using the dip test of unimodality*, in WISE, Springer, 2018, pp. 193–208.

[18] S. Maurus and C. Plant, *Skinny-dip: Clustering in a sea of noise*, SIGKDD, ACM, 2016, p. 1055–1064.

[19] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Towards an optimal subspace for k-means*, SIGKDD, ACM, 2017, p. 365–373.

[20] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Non-redundant subspace clusterings with nr-kmeans and nr-dipmeans*, ACM Trans. Knowl. Discov. Data, 14 (2020), pp. 55:1–55:24.

[21] K. Pearson, *Liii. on lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2 (1901), pp. 559–572.

[22] F. J. RICHARDS, *A Flexible Growth Function for Empirical Use*, Journal of Experimental Botany, 10 (1959), pp. 290–301.

[23] J. H. Ricketts and G. A. Head, *A five-parameter logistic equation for investigating asymmetry of curvature in baroreflex studies.*, American journal of physiology. Regulatory, integrative and comparative physiology, 277 2 (1999), pp. R441–R454.

[24] B. Schelling, L. G. M. Bauer, S. Behzadi, and C. Plant, *Utilizing structure-rich features to improve clustering*, in ECML PKDD, Springer, 2020, pp. 91–107.

[25] B. Schelling and C. Plant, *Diptransformation: Enhancing the structure of a dataset and thereby improving clustering*, in ICDM, IEEE, 2018, pp. 407–416.

[26] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouët, *Are your data gathered?*, SIGKDD, ACM, 2018, p. 2210–2218.

[27] B. W. Silverman, *Using kernel density estimates to investigate multimodality*, Journal of the Royal Statistical Society: Series B, 43 (1981), pp. 97–99.

[28] A. Vedaldi and S. Soatto, *Quick shift and kernel methods for mode seeking*, in European conference on computer vision, Springer, 2008, pp. 705–718.

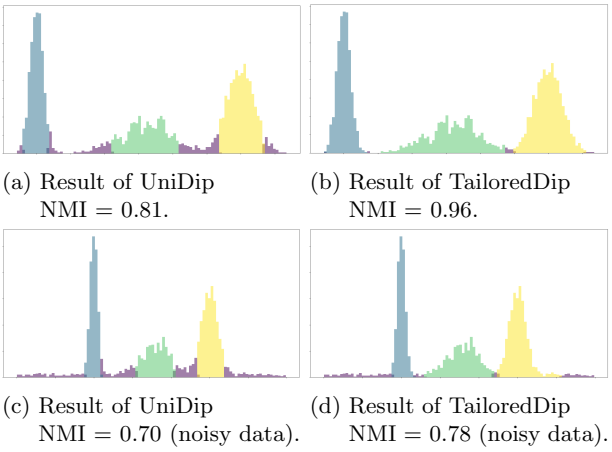
Supplement to 'Extension of the Dip-test Repertoire - Efficient and Differentiable p-value Calculation for Clustering'

Lena G. M. Bauer*,†,‡ Collin Leiber*,§ Christian Böhm† Claudia Plant†

1 TailoredDip

In the following we explain TailoredDip, which adds two extensions to the UniDip [7] algorithm. First, we show how a cluster can be expanded to include the tails of a distribution and then how outliers can be assigned to an appropriate cluster.

1.1 Capturing the Tails As mentioned in the paper, UniDip has problems correctly identifying the tails of distributions. These are usually labeled as noise. This behavior can be observed in Fig. 1a. The Gaussian clusters are not completely captured, but only the densest parts of the distributions. The same applies when uniformly distributed noise is added to the data (see Fig. 1c). TailoredDip is superior in capturing the tails of the distributions in both cases. This is also confirmed by the normalised mutual information (NMI) score and can be seen in Fig. 1b and 1d. We achieve this improvement by checking the spaces between the clusters for additional structures after the regular UniDip algorithm has terminated. Although these structures are no longer significant enough to be regarded as independent clusters by UniDip, they can still be part of a cluster. Therefore, we mirror the respective area between two clusters and calculate the Dip-p-value. If this indicates that there are still multimodal structures left, we again search for appropriate modes. In order to check whether a found structure matches the adjacent clusters, we apply a strategy that has been described in [6]. Here, the closest $2|S|$ samples of the respective cluster combined with the newly found structure S are used to calculate the Dip-p-value. If this value indicates unimodality, the structure will be added to that cluster and the process is repeated. The described procedure is shown in Algorithm 1. Since in our case a lot of Dip-p-values have to be calculated, a fast calculation of Dip-p-values is favourable.



(a) Result of UniDip NMI = 0.81.
(b) Result of TailoredDip NMI = 0.96.
(c) Result of UniDip NMI = 0.70 (noisy data).
(d) Result of TailoredDip NMI = 0.78 (noisy data).

Figure 1: Results of TailoredDip and UniDip on a sample data set consisting of three Gaussian clusters. The identified clusters are coloured in blue, green, and yellow respectively. Outliers are shown in purple.

1.2 Assigning Noise We also present a strategy for assigning outliers to clusters, paying attention to the different tails of the surrounding distributions. In terms of one-dimensional data, it makes sense to define a threshold between every two clusters, indicating whether an outlier is more likely to belong to the left or right cluster. A naive approach would now be to simply set the midpoint between the end of the left and the start of the right cluster. This strategy was chosen in [5], for example. However, this approach completely ignores the existing structures, since it is irrelevant whether a cluster ends abruptly (e.g. in case of an uniform distribution) or fades out slowly (e.g. in case of a normal distribution). To pay attention to these properties, we consider the Empirical Cumulative Distribution Function (ECDF), which is also used to calculate the Dip-value. Here, we draw a straight line from the last point of the left cluster to the first point of the right cluster. In Fig. 2 this is represented by the dotted red line. We now define the intersection of this line with the ECDF of the data as the cluster boundary. Looking at this point in Fig. 2 (left vertical line) we can see that it separates

*Authors contributed equally.
†Faculty of Computer Science, ds:UniVie, University of Vienna, Vienna, Austria. {lena.bauer, christian.boehm, claudia.plant}@univie.ac.at
‡UniVie Doctoral School Computer Science.
§LMU Munich & MCML, Munich, Germany. leiber@dbs.ifi.lmu.de

Algorithm 1: The TailoredDip algorithm

Input: one-dimensional data set X, significance α
Output: labels

1 // Get initial clusters by running UniDip
2 $labels, k = \text{UniDip}(X, \alpha)$
3 **for** $i = 0;\ i \leq k;\ i \mathrel{+}= 1$ **do**
4 **if** $i == 0$ **then**
5 X_{sub} = samples left of first cluster
6 **else if** $i == k$ **then**
7 X_{sub} = samples right of last cluster
8 **else**
9 X_{sub} = samples between cluster i and $i+1$
10 // Is X_{sub} uniformly distributed (only noise)?
11 X_{mirror} = mirror X_{sub}
12 p =p-value(Dip(X_{mirror}), $|X_{\text{mirror}}|$)
13 **if** $p < \alpha$ **then**
14 $labels_{\text{new}}, k_{\text{new}}$ =UniDip(X_{sub}, α)
15 X_{first} = combine cluster i with the first new cluster // *(ignore if $i == 0$)*
16 p_{first} = p-value(Dip(X_{first}), $|X_{\text{first}}|$)
17 X_{last} = combine cluster $i+1$ with the last new cluster // *(ignore if $i == k$)*
18 p_{last} = p-value(Dip(X_{last}), $|X_{\text{last}}|$)
19 **if** $i \neq 0$ *and* $p_{first} \geq \alpha$ *and* *($k_{new} \neq 1$ or $p_{first} \geq p_{last}$)* **then**
20 Update *labels* by adding all entries with $labels_{\text{new}} == 1$ to cluster i
21 **else if** $i \neq k$ *and* $p_{last} \geq \alpha$ *and* *($k_{new} \neq 1$ or $p_{last} > p_{first}$)* **then**
22 Update *labels* by adding all entries with $labels_{\text{new}} == k_{\text{new}}$ to cluster $i+1$
23 **if** *Cluster i or $i+1$ was updated* **then**
24 **go to** line 4
25 **return** *labels*

the tails of the two distributions much better than the naive strategy (right vertical line) since it better captures the higher standard deviation of the right cluster. If more than one intersection occurs, we choose the one closest to the midpoint between the clusters.

2 Additional Information for the Experiments

2.1 Computational Setup Dip'n'Sub as well as the algorithms DipMeans [4], projected DipMeans [1], SkinnyDip [7], DipExt [9], LDA-k-means [2] and SubKmeans [8] are all implemented in Python. Regarding FOSSCLU [3] we use the Java implementation as referenced in the paper. We conduct all runtime experiments on a machine with an Intel Core i7-5600U CPU with 2.60GHz and 8GB RAM. Further, we use Python 3.7 and in case of FOSSCLU, we use Java 8 due to compatibility issues.

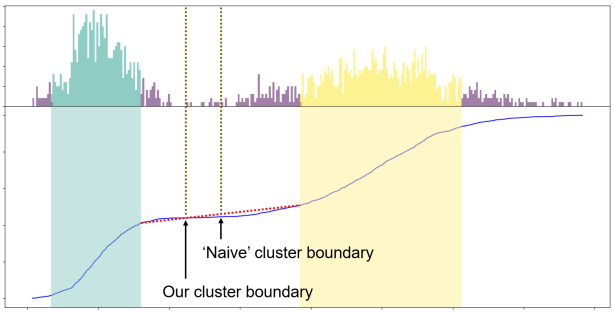


Figure 2: Visualisation of our strategy to assign outliers to the neighbouring clusters. [Top] A histogram of the data. The left (teal) cluster originates from a $\mathcal{N}(0,1)$ distribution with 700 samples, the right (yellow) cluster originates from a $\mathcal{N}(10, 2.5)$ distribution with 900 samples. The outliers are shown in purple. [Bottom] The ECDF of the data is shown in blue. The areas of the clusters are highlighted in their respective colours. The dotted red line indicates the connection line between the end of the teal and the beginning of the yellow cluster. The right vertical brown line marks the position of a naive cluster boundary, which corresponds to the centre of the red line. Our boundary corresponds to the intersection of the red line with the ECDF and captures the different tails of the two cluster much better.

2.2 Data Sets We conduct experiments on 9 real world data sets and one synthetic data set (the latter can be seen in the main paper in Fig. 4). Banknotes (BANK), User Knowledge (USER), HTRU2 and Mice Protein (MICE) are numerical data sets from the UCI repository[1]. SonyAIBO (AIBO), MoteStrain (MOTE), Symbols (SYMB) and OliveOil (OLIVE) are time series data sets[2], and ALOI[3] is an image data collection. ALOI was preprocessed as described in [10], resulting in 288 samples devided into 4 clusters. Other than ALOI, no data set did receive any pre-processing, except that features with a variance of 0 were removed. Note, that TailoredDip only works with continuous features, otherwise each value can be recognized as a separate mode. A summary of the data sets is given in Table 1.

[1] https://archive.ics.uci.edu
[2] https://www.timeseriesclassification.com
[3] https://aloi.science.uva.nl/

Table 1: Summary of the data sets (N = number of data points, d = dimensionality, k = number of clusters).

Dataset	N	d	k
SYNTH	6,300	8	7
BANK	1,372	4	2
USER	403	5	4
HTRU2	17,898	8	2
ALOI	288	66	4
MICE	1,077	68	8
AIBO	621	70	2
MOTE	1,272	84	2
SYMB	1,020	398	6
OLIVE	60	570	4

2.3 Interpolate Look-up Table We would like to briefly explain how missing values in the state-of-the-art look-up table are interpolated. Basically two interpolations must be performed. First, the values for the number of samples that lie below and above the input N must be searched for in the table. By using these two values we are able to interpolate all relevant (Dip,p)-pairs in relation to $\sqrt{N}$. In this interpolated array we search for the Dip-values that are below and above our input Dip to interpolate the Dip-p-value linearly.

2.4 Large Distribution Table In Tables 2 and 3 we show Dip-p-value calculations with the three methods 'table' (T), 'function' (F) and 'bootstrap' (B) for samples of 15 different sample sizes and a total of 23 distribution scenarios. In all cases, we can observe that our fitted function produces basically the same Dip-p-values as the other two methods. For this evaluation we first consider 8 different unimodal distributions:

- $\mathcal{N}(a,b)\ldots$ normal distribution with mean a and standard deviation b
- $\mathcal{T}(d,a,b)\ldots$ students-t distribution with d degrees of freedom, centre a and scaling b
- $\mathcal{L}(a,b)\ldots$ Laplace distribution with centre a and scaling b
- $\mathcal{U}(a,b)\ldots$ uniform distribution between a and b
- $\mathcal{G}(s,a,b)\ldots$ Gamma distribution with shape parameter s, centre a and scaling b
- $\mathcal{E}(a,b)\ldots$ exponential distribution with centre a and scaling b
- $\mathcal{B}(s,r,a,b)\ldots$ Beta distribution with shape parameters s and r, centre a and scaling b
- $\mathcal{T}_{nc}(d,c,a,b)\ldots$ non central students-t distribution with d degrees of freedom, non centrality c, centre a and scaling b

First, Table 2 shows results for these distributions, with only the listed distributions involved individually. We then generate 8 multimodal distributions by combining $\frac{N}{2}$ samples from one distribution with $\frac{N}{2}$ samples from the same distribution with a different centre. Additionally, we consider 7 cases, where we generate samples by choosing half the points from $\mathcal{N}(4,1)$ and the other half from one of the other unimodal distributions. These combinations always show multimodal structure. An exception is the case of samples from $\mathcal{N}(4,1) \cup \mathcal{T}_{nc}(4,2,0,1)$. We include this combination to have a relatively unambiguous case between unimodal and multimodal. Our function performs reliably in all cases as can be seen in Table 3.

References

[1] T. Chamalis and A. Likas, *The projected dip-means clustering algorithm*, SETN, ACM, 2018.

[2] C. H. Q. Ding and T. Li, *Adaptive dimension reduction using discriminant analysis and* K*-means clustering*, in ICML, vol. 227, ACM, 2007, pp. 521–528.

[3] S. Goebl, X. He, C. Plant, and C. Böhm, *Finding the optimal subspace for clustering*, in ICDM, IEEE, 2014, pp. 130–139.

[4] 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, Curran Associates, Inc., 2012.

[5] C. Leiber, L. G. M. Bauer, M. Neumayr, C. Plant, and C. Böhm, *The dipencoder: Enforcing multimodality in autoencoders*, in SIGKDD, ACM, 2022, p. 846–856.

[6] C. Leiber, L. G. M. Bauer, B. Schelling, C. Böhm, and C. Plant, *Dip-based deep embedded clustering with k-estimation*, SIGKDD, ACM, 2021, p. 903–913.

[7] S. Maurus and C. Plant, *Skinny-dip: Clustering in a sea of noise*, SIGKDD, ACM, 2016, p. 1055–1064.

[8] D. Mautz, W. Ye, C. Plant, and C. Böhm, *Towards an optimal subspace for k-means*, SIGKDD, ACM, 2017, p. 365–373.

[9] B. Schelling, L. G. M. Bauer, S. Behzadi, and C. Plant, *Utilizing structure-rich features to improve clustering*, in ECML PKDD, Springer, 2020, pp. 91–107.

[10] W. Ye, S. Maurus, N. Hubig, and C. Plant, *Generalized independent subspace clustering*, in ICDM, IEEE, 2016, pp. 569–578.

Table 2: Dip-p-values for different unimodal distributions with varying sample sizes N. All given values are averages for 100 random samples $\pm$ standard deviation. Respective first, second and third rows per distribution show Dip-p-values calculated with methods 'table' (T), 'function' (F) and 'bootstrapping' (B, 1000 repetitions); $*$: values obtained by $\sqrt{N}-$ interpolation, †: values not available.

Distr.	Meth.	$N = 50$	$N = 67$	$N = 100$	$N = 234$	$N = 500$	$N = 678$	$N = 1k$	$N = 2345$	$N = 5k$	$N = 6789$	$N = 10k$	$N = 23456$	$N = 50k$	$N = 67890$	$N = 100k$
$\mathcal{N}(4,1)$	T	0.77 ± 0.24	$0.81 \pm 0.21^*$	0.80 ± 0.20	$0.86 \pm 0.19^*$	0.91 ± 0.14	$0.94 \pm 0.13^*$	0.94 ± 0.09	$0.97 \pm 0.07^*$	0.99 ± 0.03	$0.98 \pm 0.05^*$	0.99 ± 0.04	$0.99 \pm 0.01^*$	1.00 ± 0.02	$1.00 \pm 0.01^*$	†
	F	0.77 ± 0.24	0.81 ± 0.21	0.81 ± 0.20	0.86 ± 0.19	0.91 ± 0.14	0.94 ± 0.13	0.95 ± 0.09	0.97 ± 0.07	0.99 ± 0.03	0.98 ± 0.05	0.99 ± 0.04	1.00 ± 0.01	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.02
	B	0.77 ± 0.24	0.81 ± 0.21	0.81 ± 0.20	0.86 ± 0.19	0.91 ± 0.14	0.94 ± 0.13	0.95 ± 0.09	0.97 ± 0.07	0.99 ± 0.03	0.98 ± 0.05	0.99 ± 0.04	1.00 ± 0.01	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.02
$\mathcal{T}(4,0,1)$	T	0.78 ± 0.22	$0.85 \pm 0.19^*$	0.84 ± 0.20	$0.88 \pm 0.18^*$	0.94 ± 0.10	$0.95 \pm 0.09^*$	0.97 ± 0.07	$0.97 \pm 0.07^*$	0.99 ± 0.04	$0.99 \pm 0.03^*$	0.99 ± 0.02	$1.00 \pm 0.01^*$	1.00 ± 0.00	$1.00 \pm 0.01^*$	†
	F	0.78 ± 0.22	0.85 ± 0.19	0.85 ± 0.20	0.88 ± 0.18	0.94 ± 0.10	0.96 ± 0.09	0.97 ± 0.07	0.97 ± 0.07	0.99 ± 0.04	0.99 ± 0.03	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.01	1.00 ± 0.00
	B	0.78 ± 0.22	0.85 ± 0.19	0.85 ± 0.20	0.88 ± 0.18	0.94 ± 0.10	0.96 ± 0.09	0.97 ± 0.07	0.97 ± 0.07	0.99 ± 0.04	0.99 ± 0.02	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.01	1.00 ± 0.00
$\mathcal{L}(0,2)$	T	0.85 ± 0.19	$0.88 \pm 0.18^*$	0.92 ± 0.11	$0.95 \pm 0.11^*$	0.98 ± 0.04	$0.98 \pm 0.04^*$	0.99 ± 0.01	$0.99 \pm 0.04^*$	1.00 ± 0.00	$1.00 \pm 0.00^*$	1.00 ± 0.00	$1.00 \pm 0.00^*$	1.00 ± 0.00	$1.00 \pm 0.00^*$	†
	F	0.85 ± 0.19	0.89 ± 0.18	0.92 ± 0.11	0.95 ± 0.11	0.98 ± 0.04	0.99 ± 0.04	0.99 ± 0.01	0.99 ± 0.04	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	B	0.85 ± 0.19	0.89 ± 0.18	0.92 ± 0.11	0.95 ± 0.11	0.98 ± 0.04	0.99 ± 0.03	1.00 ± 0.01	0.99 ± 0.04	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
$\mathcal{U}(0,2)$	T	0.52 ± 0.29	$0.50 \pm 0.28^*$	0.46 ± 0.28	$0.53 \pm 0.29^*$	0.52 ± 0.28	$0.54 \pm 0.31^*$	0.48 ± 0.30	$0.50 \pm 0.28^*$	0.56 ± 0.28	$0.54 \pm 0.29^*$	0.52 ± 0.32	$0.50 \pm 0.30^*$	0.53 ± 0.28	$0.51 \pm 0.30^*$	†
	F	0.52 ± 0.30	0.51 ± 0.28	0.46 ± 0.28	0.53 ± 0.30	0.52 ± 0.29	0.54 ± 0.31	0.48 ± 0.30	0.49 ± 0.29	0.56 ± 0.28	0.54 ± 0.30	0.52 ± 0.32	0.50 ± 0.30	0.53 ± 0.28	0.51 ± 0.30	0.56 ± 0.28
	B	0.52 ± 0.29	0.51 ± 0.28	0.46 ± 0.28	0.53 ± 0.30	0.52 ± 0.29	0.54 ± 0.31	0.48 ± 0.30	0.50 ± 0.29	0.56 ± 0.28	0.53 ± 0.30	0.52 ± 0.32	0.50 ± 0.30	0.53 ± 0.28	0.51 ± 0.31	0.56 ± 0.28
$\mathcal{G}(2,-1,1)$	T	0.76 ± 0.23	$0.78 \pm 0.22^*$	0.81 ± 0.21	$0.88 \pm 0.14^*$	0.91 ± 0.14	$0.91 \pm 0.11^*$	0.93 ± 0.13	$0.97 \pm 0.08^*$	0.98 ± 0.05	$0.98 \pm 0.05^*$	0.99 ± 0.02	$0.99 \pm 0.02^*$	1.00 ± 0.01	$1.00 \pm 0.00^*$	†
	F	0.76 ± 0.24	0.79 ± 0.22	0.81 ± 0.21	0.88 ± 0.14	0.91 ± 0.14	0.92 ± 0.11	0.94 ± 0.13	0.97 ± 0.08	0.99 ± 0.05	0.99 ± 0.05	0.99 ± 0.02	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00
	B	0.76 ± 0.23	0.79 ± 0.22	0.81 ± 0.21	0.88 ± 0.14	0.91 ± 0.14	0.92 ± 0.11	0.94 ± 0.13	0.98 ± 0.08	0.99 ± 0.05	0.99 ± 0.05	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
$\mathcal{E}(0,1)$	T	0.72 ± 0.26	$0.79 \pm 0.21^*$	0.79 ± 0.23	$0.86 \pm 0.18^*$	0.92 ± 0.11	$0.95 \pm 0.10^*$	0.94 ± 0.11	$0.98 \pm 0.05^*$	0.99 ± 0.03	$0.99 \pm 0.02^*$	1.00 ± 0.01	$1.00 \pm 0.00^*$	1.00 ± 0.00	$1.00 \pm 0.00^*$	†
	F	0.72 ± 0.26	0.79 ± 0.21	0.80 ± 0.23	0.86 ± 0.18	0.93 ± 0.11	0.95 ± 0.10	0.94 ± 0.11	0.98 ± 0.05	0.99 ± 0.04	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	B	0.72 ± 0.26	0.79 ± 0.21	0.79 ± 0.23	0.86 ± 0.18	0.93 ± 0.11	0.95 ± 0.10	0.94 ± 0.11	0.98 ± 0.05	0.99 ± 0.04	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
$\mathcal{B}(2,2,1,1)$	T	0.65 ± 0.25	$0.66 \pm 0.25^*$	0.73 ± 0.25	$0.81 \pm 0.18^*$	0.82 ± 0.22	$0.84 \pm 0.21^*$	0.84 ± 0.21	$0.90 \pm 0.16^*$	0.96 ± 0.07	$0.96 \pm 0.09^*$	0.98 ± 0.05	$0.98 \pm 0.05^*$	0.99 ± 0.02	$0.99 \pm 0.01^*$	†
	F	0.65 ± 0.25	0.67 ± 0.26	0.73 ± 0.26	0.82 ± 0.18	0.82 ± 0.22	0.84 ± 0.21	0.85 ± 0.21	0.90 ± 0.16	0.96 ± 0.07	0.96 ± 0.09	0.98 ± 0.05	0.99 ± 0.05	0.99 ± 0.02	1.00 ± 0.01	0.99 ± 0.02
	B	0.65 ± 0.26	0.66 ± 0.25	0.73 ± 0.25	0.81 ± 0.19	0.82 ± 0.22	0.84 ± 0.21	0.84 ± 0.21	0.90 ± 0.16	0.96 ± 0.07	0.97 ± 0.09	0.98 ± 0.05	0.99 ± 0.05	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.02
$\mathcal{T}_{nc}(4,2,0,1)$	T	0.80 ± 0.21	$0.78 \pm 0.23^*$	0.85 ± 0.17	$0.89 \pm 0.14^*$	0.96 ± 0.06	$0.94 \pm 0.10^*$	0.95 ± 0.08	$0.98 \pm 0.03^*$	0.99 ± 0.04	$0.99 \pm 0.02^*$	0.99 ± 0.01	$1.00 \pm 0.01^*$	1.00 ± 0.00	$1.00 \pm 0.00^*$	†
	F	0.80 ± 0.21	0.78 ± 0.23	0.85 ± 0.17	0.90 ± 0.14	0.96 ± 0.06	0.95 ± 0.10	0.96 ± 0.08	0.98 ± 0.03	0.99 ± 0.04	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	B	0.80 ± 0.21	0.78 ± 0.23	0.85 ± 0.17	0.90 ± 0.14	0.96 ± 0.06	0.95 ± 0.10	0.96 ± 0.08	0.99 ± 0.03	0.99 ± 0.04	0.99 ± 0.02	1.00 ± 0.01	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00

Table 3: Dip-p-values for different combinations of distributions with varying sample sizes N. All given values are averages for 100 random samples $\pm$ standard deviation. Respective first, second and third rows per distribution show Dip-p-values calculated with methods 'table' (T), 'function' (F) and 'bootstrapping' (B, 1000 repetitions). Values are multiplied by 100. $*$: values obtained by $\sqrt{N}-$ interpolation, †: values not available.

Distr.	Meth.	$N = 50$	$N = 67$	$N = 100$	$N = 234$	$N = 500$	$N = 678$	$N = 1k$	$N = 2345$	$N = 5k$	$N = 6789$	$N = 10k$	$N = 23456$	$N = 50k$	$N = 67890$	$N = 100k$
$\mathcal{N}(4,1)$	T	8.94 ± 15.0	$6.97 \pm 14.1^*$	4.95 ± 10.4	$0.06 \pm 0.23^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	*F*	8.83 ± 14.9	7.03 ± 14.2	4.87 ± 10.3	0.09 ± 0.25	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(0,1)$	B	8.78 ± 15.0	6.81 ± 13.9	4.97 ± 10.5	0.06 ± 0.21	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}(4,0,1)$	T	21.3 ± 22.7	$16.7 \pm 23.3^*$	9.11 ± 16.0	$1.25 \pm 3.85^*$	0.01 ± 0.03	$0.00 \pm 0.01^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	21.0 ± 22.9	16.8 ± 23.6	9.01 ± 16.0	1.26 ± 3.74	0.01 ± 0.04	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}(4,4,1)$	B	21.0 ± 22.9	16.8 ± 23.7	8.87 ± 15.9	1.21 ± 3.85	0.00 ± 0.03	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{L}(0,2)$	T	24.1 ± 24.9	$23.2 \pm 25.7^*$	16.4 ± 23.3	$2.36 \pm 7.87^*$	0.02 ± 0.08	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	23.9 ± 25.1	23.3 ± 26.0	16.2 ± 23.4	2.37 ± 7.82	0.03 ± 0.10	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{L}(7,2)$	B	23.8 ± 24.8	23.1 ± 25.9	16.4 ± 23.5	2.34 ± 8.06	0.02 ± 0.10	0.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{U}(0,2)$	T	0.07 ± 0.09	$0.01 \pm 0.01^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	0.15 ± 0.13	0.04 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{U}(3,2)$	B	0.08 ± 0.13	0.01 ± 0.03	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{G}(2,-1,1)$	T	1.00 ± 5.80	$0.18 \pm 1.28^*$	0.01 ± 0.04	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	1.07 ± 5.78	0.21 ± 1.23	0.02 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{G}(2,5,1)$	B	0.98 ± 5.48	0.17 ± 1.11	0.01 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{E}(0,1)$	T	0.08 ± 0.36	$0.00 \pm 0.02^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	0.11 ± 0.37	0.01 ± 0.03	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{E}(4,1)$	B	0.09 ± 0.42	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{B}(2,2,1,1)$	T	7.05 ± 14.1	$4.69 \pm 10.4^*$	1.05 ± 2.26	$0.03 \pm 0.13^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	7.03 ± 14.0	4.72 ± 10.4	1.10 ± 2.11	0.05 ± 0.16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{B}(2,2,2,1)$	B	6.80 ± 14.0	4.68 ± 10.4	1.00 ± 2.07	0.04 ± 0.18	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}_{nc}(4,2,0,1)$	T	0.79 ± 2.20	$0.32 \pm 2.17^*$	0.01 ± 0.06	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	0.83 ± 2.07	0.37 ± 2.16	0.03 ± 0.08	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}_{nc}(4,2,7,1)$	B	0.74 ± 2.17	0.36 ± 2.42	0.01 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	19.0 ± 23.1	$9.19 \pm 12.4^*$	7.03 ± 13.7	$0.35 \pm 0.89^*$	0.00 ± 0.01	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	18.8 ± 23.2	9.16 ± 12.4	6.92 ± 13.6	0.38 ± 0.85	0.01 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{T}(4,0,1)$	B	18.7 ± 23.1	9.35 ± 12.6	6.90 ± 13.6	0.33 ± 0.88	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	59.4 ± 32.6	$55.5 \pm 32.0^*$	54.0 ± 30.9	$44.7 \pm 32.9^*$	25.4 ± 28.0	$21.6 \pm 23.9^*$	13.0 ± 17.6	$0.89 \pm 3.11^*$	0.01 ± 0.05	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	59.5 ± 32.9	56.0 ± 32.3	54.0 ± 31.2	44.7 ± 33.2	25.2 ± 28.2	21.4 ± 24.1	12.8 ± 17.6	0.94 ± 3.05	0.02 ± 0.07	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{L}(0,2)$	B	59.4 ± 32.8	56.0 ± 32.3	54.2 ± 31.1	44.6 ± 33.2	25.4 ± 28.3	21.4 ± 23.9	12.7 ± 17.6	0.85 ± 2.91	0.01 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	24.2 ± 26.4	$20.0 \pm 23.3^*$	11.2 ± 18.4	$1.14 \pm 2.95^*$	0.03 ± 0.18	$0.00 \pm 0.04^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	24.0 ± 26.6	20.1 ± 23.5	12.1 ± 18.4	1.17 ± 2.86	0.04 ± 0.19	0.01 ± 0.05	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{U}(0,2)$	B	23.8 ± 26.5	20.1 ± 23.5	11.0 ± 18.4	1.08 ± 2.91	0.03 ± 0.16	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	28.8 ± 28.3	$28.2 \pm 28.3^*$	28.0 ± 28.5	$9.39 \pm 16.1^*$	3.37 ± 8.94	$1.07 \pm 3.24^*$	0.13 ± 0.55	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	28.6 ± 28.5	28.4 ± 28.7	27.9 ± 28.7	9.33 ± 16.1	3.32 ± 8.86	1.05 ± 3.05	0.15 ± 0.54	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{G}(2,-1,1)$	B	28.6 ± 28.5	28.5 ± 28.8	28.0 ± 28.6	9.30 ± 16.1	3.31 ± 8.94	1.02 ± 3.00	0.13 ± 0.56	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	17.6 ± 23.2	$9.91 \pm 14.5^*$	9.01 ± 16.6	$1.26 \pm 4.10^*$	0.05 ± 0.33	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	17.5 ± 23.3	9.94 ± 14.6	8.94 ± 16.6	1.29 ± 4.03	0.07 ± 0.31	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{E}(0,1)$	B	17.5 ± 23.4	9.91 ± 14.7	8.79 ± 16.6	1.26 ± 4.00	0.05 ± 0.30	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	22.3 ± 25.4	$19.4 \pm 25.5^*$	10.3 ± 15.5	$1.51 \pm 3.26^*$	0.02 ± 0.05	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	0.00 ± 0.00	$0.00 \pm 0.00^*$	†
∪	F	22.1 ± 25.5	19.5 ± 25.8	10.1 ± 15.4	1.51 ± 3.09	0.03 ± 0.07	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{B}(2,2,1,1)$	B	22.1 ± 25.4	19.8 ± 25.8	10.3 ± 15.4	1.47 ± 3.17	0.01 ± 0.05	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
$\mathcal{N}(4,1)$	T	69.8 ± 26.9	$67.7 \pm 27.0^*$	73.9 ± 22.9	$82.8 \pm 19.0^*$	82.9 ± 20.5	$88.4 \pm 15.8^*$	90.6 ± 12.5	$95.5 \pm 7.56^*$	96.9 ± 8.89	$97.2 \pm 5.51^*$	98.5 ± 4.34	$99.1 \pm 2.97^*$	99.7 ± 0.44	$99.8 \pm 0.22^*$	†
∪	F	69.9 ± 27.2	68.3 ± 27.2	74.1 ± 23.1	83.1 ± 19.0	83.1 ± 20.6	88.7 ± 15.8	90.8 ± 12.5	95.7 ± 7.54	97.0 ± 8.91	97.4 ± 5.52	98.7 ± 4.36	99.3 ± 2.97	99.9 ± 0.43	1.00 ± 0.13	1.00 ± 0.06
$\mathcal{T}_{nc}(4,2,0,1)$	B	69.8 ± 27.1	68.0 ± 27.3	73.7 ± 23.4	83.0 ± 19.2	83.2 ± 20.3	88.8 ± 15.9	90.8 ± 12.6	95.8 ± 7.60	97.0 ± 9.11	97.4 ± 5.59	98.8 ± 4.29	99.4 ± 2.94	99.9 ± 0.36	1.00 ± 0.08	1.00 ± 0.02

C.3 Dip-based Deep Embedded Clustering with k-Estimation

Authors:
Leiber, Collin* and Bauer, Lena G. M.* and Schelling, Benjamin and Böhm, Christian and Plant, Claudia
** First authors with equal contribution*

Abstract:
The combination of clustering with Deep Learning has gained much attention in recent years. Unsupervised neural networks like autoencoders can autonomously learn the essential structures in a data set. This idea can be combined with clustering objectives to learn relevant features automatically. Unfortunately, they are often based on a k-means framework, from which they inherit various assumptions, like spherical-shaped clusters. Another assumption, also found in approaches outside the k-means-family, is knowing the number of clusters a-priori. In this paper, we present the novel clustering algorithm DipDECK, which can estimate the number of clusters simultaneously to improving a Deep Learning-based clustering objective. Additionally, we can cluster complex data sets without assuming only spherically shaped clusters. Our algorithm works by heavily overestimating the number of clusters in the embedded space of an autoencoder and, based on Hartigan's Dip-test - a statistical test for unimodality - analyses the resulting micro-clusters to determine which to merge. We show in extensive experiments the various benefits of our method: (1) we achieve competitive results while learning the clustering-friendly representation and number of clusters simultaneously; (2) our method is robust regarding parameters, stable in performance, and allows for more flexibility in the cluster shape; (3) we outperform relevant competitors in the estimation of the number of clusters.

Venue:
KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021, ACM, 2021.
CORE ranking: A (`http://portal.core.edu.au/conf-ranks/26/`)*
Acceptance rate: 15.5%

DOI:
`https://doi.org/10.1145/3447548.3467316`

Thesis-Reference:
[LBS+21]

Division of Work:

Joint work of the first authors
Development of the idea to combine k-estimation with deep clustering; Constant discussions on the procedure; Implementing the algorithm using pair-programming; Designing the quantitative experiments; Executing the quantitative experiments; Co-writing large parts of the paper

Collin Leiber
Formulation of the optimization and cluster merging strategy; Refinement of the initial loss function by considering the inter-cluster distances; Setting-up the pipeline for the comparison algorithms; Designing the robustness experiments; Executing the robustness experiments; Writing the majority of the related work regarding k-estimation approaches and the robustness experiments; Creating the visualizations regarding the robustness experiments and the confusion matrices; Creating the poster and parts of the video; Co-presenting at the conference

Lena G. M. Bauer
Development of the idea to check the pair-wise Dip-values and formulation of an initial Dip-based loss function; Designing the qualitative experiments; Executing the qualitative experiments; Writing the majority of the related work regarding deep clustering and the qualitative experiments; Creating the visualizations regarding the methodology, the synthetic dataset and the qualitative experiments; Creating the presentation slides and parts of the video; Co-presenting at the conference

Benjamin Schelling
Initial idea to combine Hartigan's Dip-test with deep learning; Regular discussions on the procedure; Providing information regarding the Dip-test and writing of the associated part of the related work section; Helping to design the experiments; Writing large parts of the experiments section

Christian Böhm
Identification of relevant related work; Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Claudia Plant
Identification of relevant related work; Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Dip-based Deep Embedded Clustering with k-Estimation

Collin Leiber*
LMU Munich & MCML
Munich, Germany
leiber@dbs.ifi.lmu.de

Lena G. M. Bauer*
ds:UniVie
University of Vienna, Vienna, Austria
lena.bauer@univie.ac.at

Benjamin Schelling
Faculty of Computer Science
University of Vienna, Vienna, Austria
benjamin.schelling@univie.ac.at

Christian Böhm
LMU Munich & MCML
Munich, Germany
boehm@dbs.ifi.lmu.de

Claudia Plant
Faculty of Computer Science &
ds:UniVie
University of Vienna, Vienna, Austria
claudia.plant@univie.ac.at

ABSTRACT

The combination of clustering with Deep Learning has gained much attention in recent years. Unsupervised neural networks like autoencoders can autonomously learn the essential structures in a data set. This idea can be combined with clustering objectives to learn relevant features automatically. Unfortunately, they are often based on a k-means framework, from which they inherit various assumptions, like spherical-shaped clusters. Another assumption, also found in approaches outside the k-means-family, is knowing the number of clusters a-priori. In this paper, we present the novel clustering algorithm DipDECK, which can estimate the number of clusters simultaneously to improving a Deep Learning-based clustering objective. Additionally, we can cluster complex data sets without assuming only spherically shaped clusters. Our algorithm works by heavily overestimating the number of clusters in the embedded space of an autoencoder and, based on Hartigan's Dip-test - a statistical test for unimodality - analyses the resulting micro-clusters to determine which to merge. We show in extensive experiments the various benefits of our method: (1) we achieve competitive results while learning the clustering-friendly representation and number of clusters simultaneously; (2) our method is robust regarding parameters, stable in performance, and allows for more flexibility in the cluster shape; (3) we outperform relevant competitors in the estimation of the number of clusters.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Computing methodologies** → **Cluster analysis**; *Dimensionality reduction and manifold learning*; *Neural networks.*

KEYWORDS

Deep Clustering; Dip-test; Estimating the number of clusters

*First authors with equal contribution.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467316

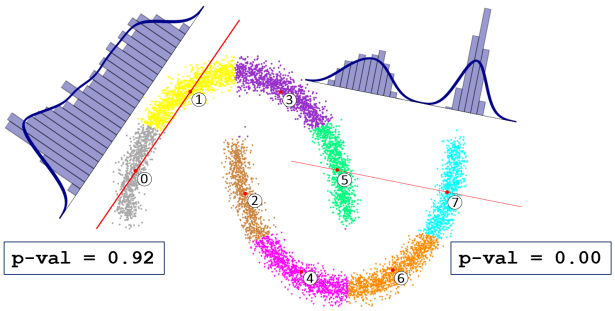


Figure 1: Main idea of our Dip-based clustering. The depicted two moons data set contains two true clusters. We overestimate the number of clusters by executing k-means with $k_{\text{init}} = 8$. If we draw a straight line between two k-means centroids (red) and project the data points assigned to either one of the clusters onto this line, the Dip-test can be applied to the projected points. The blue histograms show the distribution of the projected points of clusters 0 and 1 on the left side and clusters 5 and 7 on the right side. High Dip-p-values indicate unimodality. Thus, according to the Dip-test, clusters 0 and 1 should be merged, while 5 and 7 should not.

ACM Reference Format:
Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. 2021. Dip-based Deep Embedded Clustering with k-Estimation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467316

1 INTRODUCTION

Finding patterns in large amounts of unlabelled data is one of the major data mining research branches. The goal is to partition the data into groups of similar data points. However, in practice, it is often not known how many clusters there are.

There is a (wide) range of potential methods for traditional clustering algorithms to address this problem. Many of them are based on the k-means framework like X-means [23] or Dip-means [16]. Some approaches like PG-Means [8] are EM-based, which allows for more flexibility in the cluster shape. However, in these frameworks, they automatically inherit the Gaussian cluster assumption. While

this might be accurate for some data sets, it is too restrictive for others, resulting in an arbitrary clustering. There are, of course, other clustering approaches that are able to determine the number of clusters in a data set automatically and are not limited to a Gaussian cluster shape. One of the best known is the density-based method DBSCAN [7]. It can, just like some variants of Spectral Clustering-based approaches [31], estimate the number of clusters and is very flexible regarding the cluster shape. These approaches, however, trade a comparably easy to understand parameter - the number of clusters - for more complicated parameters (e.g., the neighbourhood range or the number of neighbours). The detected number of clusters is, to a large extent, controlled by these parameters. Thus, these methods substitute one parameter with others.

The major drawback for all of the mentioned methods is that their performance becomes unsatisfactory for modern data sets consisting of large and high dimensional data such as images, videos, and text. While run time and memory issues could be resolved with high-performance implementations, other problems such as the curse of dimensionality remain since many methods are based on the Euclidean distance. The trend for these types of data sets has become to cluster them with Deep Learning (DL) approaches. For this task, most methods use an autoencoder to learn a cluster-friendly lower-dimensional representation of the data such that the clustering can be executed in sufficient run time and to overcome the curse of dimensionality. Thus, methods to estimate the correct number of clusters should ideally be integrated into these types of approaches such that they are compatible and can exploit the benefits of DL.

Until now, to the best of our knowledge, there is no DL-based method to cluster a data set while simultaneously estimating the correct number of clusters. All existing strategies are built on traditional clustering approaches that do not scale well to high-dimensional and/or large data sets.

To address the above problems, we propose **DipDECK** - **Dip**-based **D**eep **E**mbedded **C**lustering with **k**-estimation. It simultaneously improves the estimation of the number of clusters k, the cluster assignments, and the data embedding. We overestimate the number of clusters in the embedded space of an autoencoder by k_{init} and use Hartigan's Dip-test [12], a statistical test for modality in one-dimensional samples, to identify clusters that share structural similarities. It returns a Dip-value, which translates to a p-value that is equal to the probability of a sample's unimodality. We define a clustering loss that forces the autoencoder to push clusters sharing a high Dip-p-value together, resulting in compact cluster shapes. These can then be combined into a common cluster. We induce the assumption that $k \leq k_{\text{init}}$. However, this is a significant relaxation compared to when k needs to be given as a fixed value. Fig. 1 serves as an illustration of the idea behind our Dip-based clustering. Assume the depicted data points represent a two-dimensional autoencoder embedding of a higher-dimensional data set. We can see how the Dip-test indicates which sub-clusters are connected. Additionally, it shows that we can even use this strategy to identify non-convex cluster shapes.

Our contributions can be summarised as follows:

- We introduce a novel deep clustering method that operates oblivious of the correct number of clusters k in the data. Despite being at the disadvantage of not knowing k, we achieve competitive clustering results regarding NMI on a wide range of benchmark and additional data sets.
- Although partly centroid-based by implicitly taking a k-means like loss term into account, our method is more flexible regarding the cluster shape.
- We exploit the Dip-test, a statistical test for unimodality, which quantifies the structure within a data set. The Dip-test is introduced for the first time to DL.
- Since we outperform relevant competitors in estimating the number of clusters on various data sets, our method can also be used merely to estimate this value. This estimation can then be used in combination with other clustering methods.

2 RELATED WORK

We introduce DipDECK, a method for k-estimation in a DL context. Thus, we have two relevant research branches: DL approaches for clustering and estimation methods for k. Since our method is based on the Dip-test, we also describe it in more detail in this section.

Deep Clustering. The first branch consists of DL methods created for clustering. For this, we focus on the essential and foundational methods which have been established as some of the most important Deep Clustering (DC) methods. These include DEC [28], IDEC [10] and DCN [29], which are all centroid-based approaches without excessive tuning of the autoencoder or exploitation of domain knowledge such as augmentation for images. These are closely relatable to us; the same autoencoder can be used for comparisons, and they also have a degree of kinship to k-means. VaDE [15] embeds the probabilistic clustering problem into a variational autoencoder framework. Although this strategy is different to the other mentioned methods, there is still a relation because it models the data generative procedure by a Gaussian Mixture Model and is therefore also Gaussian-based. Other DC approaches like ClusterGAN [22], JULE [30], DEPICT, [9] or DTI [21] are already rather distant or partly use extensive techniques (e.g. CNNs or augmentation). In our work, we do not focus on topics such as autoencoder architecture optimisation or exploitation of domain knowledge in, e.g., the form of augmentation. These directions of research are orthogonal to our approach of optimising the clustering objective and simultaneous k-estimation.

Estimating k. The second, more important, branch is k- estimation methods. Most of them are based on k-means and proceed in the following way: They start with a low initial number of clusters k_{init} (usually a single cluster) and then apply a criterion to determine whether a cluster should be split. Thus, their most significant distinction is the applied criterion, for which they use, e.g., the Bayesian Information Criterion (X-means [23]), the Dip-test (Dip-means [16], respectively its continuation pDip-means [3]), Anderson-Darling hypothesis test (G-means [11]) or the Kolmogorov-Smirnov test (PG-means [8]). They either stop determined by their criterion, or the value of the maximal accepted k_{max} is reached. Some approaches (e.g. [1]) start with k_{max} and merge clusters until the termination condition is reached, i.e. the applied criterion is satisfied, or k equals k_{min} (usually a single cluster). Our approach also follows the route of merging clusters to find a suitable value of k. The advantage of starting with a large k is that each micro-cluster can preserve some

Table 1: Description of the used symbols.

Symbol	Interpretation
$d \in \mathbb{N}$	Dimensionality of the original feature space
$m \in \mathbb{N}$	Dimensionality of the embedded space
$k \in \mathbb{N}$	Number of clusters
$X \subseteq \mathbb{R}^d$	Set of all objects
$C_i \subseteq X$	Objects assigned to cluster i
$\mathcal{B} \subseteq X$	A mini-batch
$\mu_i \in C_i$	Cluster centre of cluster i
$\mu_i^{\text{km}} \in \mathbb{R}^m$	$i-$th k-means cluster centre
$x \in X$	A single object of the data set
$d_{i,j} \in [0, 0.25]$	Dip-value of the combined clusters i and j
$p_{i,j} \in [0, 1]$	Dip-p-value of $d_{i,j}$ (0 if $d_{i,j} \approx 0.25$; 1 if $d_{i,j} \approx 0$)
$T \in [0, 1]$	Dip-p-value threshold

underlying structure in the embedded space. With our approach, the shape of the Dip-connected micro-clusters is - with the help of the autoencoder - transformed, and they start to merge depending on the Dip-test until we have only clearly separated clusters transformed into roughly spherical shapes.

Another research branch covers postprocessing methods that can be used to enhance an existing clustering solution. An example is RIC [2] which utilises MDL to purify and improve the given clustering structures. This improvement includes a refinement of the number of clusters. These methods are, however, orthogonal to our goal of optimising the initial clustering objective.

The only DC-related approach that includes a strategy for cluster number estimation is SCDE [6]. However, they use an additional softmax autoencoder network for the cluster estimation and do not integrate the estimation as an automated process during DC. Instead, they use the estimated amount of clusters to execute Spectral Clustering on the embedded samples. Furthermore, the embedded space dimension of this additional softmax network is systematically chosen as twice the number of ground truth clusters - precisely the number we are trying to estimate. Another DC approach that does not need k as an input parameter is DeepECT [20]. In contrast to SCDE and DipDECK, DeepECT does not estimate a concrete k but returns a hierarchical clustering structure which can be further analysed to choose an appropriate number of clusters afterwards.

Dip-test. In order to identify coherent patterns, our approach makes use of the Dip-test [12]. The Dip-test is a statistical test developed in the 1980s by Hartigan and Hartigan to measure modality. It returns a Dip-value $Dip \in [0, 0.25]$, which, when close to zero, indicates unimodality. The corresponding Dip-p-value, which indicates how likely it is that a sample set is unimodal, is then close to 1. Larger Dip-values indicate that the data set contains at least two modes, which will yield a Dip-p-value almost equal to zero. The advantages are the run time and the fact that it is parameter-free. It has, until now, never been used in DL. There are some approaches in traditional clustering that make use of it [16], and it has recently gained interest in the data mining community [3, 19, 24, 25].

3 DIP-BASED DEEP EMBEDDED CLUSTERING

In this section, we describe our method Dip-based Deep Embedded Clustering with k-estimation (DipDECK). It utilises an autoencoder [17] to simultaneously estimate the number of clusters and define

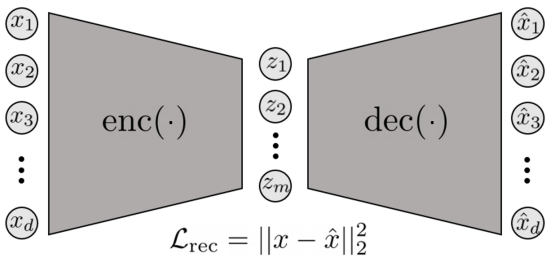


Figure 2: Autoencoder network architecture. The clustering is performed on the data points $z = \mathbf{enc}(x)$.

cluster assignments in the embedded space. All symbols used in this work are described in Table 1.

An autoencoder is a two-part unsupervised neural network that consists of an encoder and a decoder network. The encoder embeds the input data into a latent, usually lower-dimensional space. The decoder, on the other hand, tries to reconstruct the embedded data into its original state. The autoencoder can learn the properties of the embedded space by minimising the reconstruction loss $\mathcal{L}_{rec}$. Usually, the mean squared error is used for this. For a mini-batch $\mathcal{B}$, this loss reads as follows:

$$\mathcal{L}_{rec} = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} ||x - \text{dec}(\text{enc}(x))||_2^2, \tag{1}$$

where enc(·) describes the data after applying the encoder, dec(·) is the result of the decoder, and $||\cdot||_2^2$ denotes the squared Euclidean distance. Fig. 2 illustrates the autoencoder architecture.

In general, our method requires two main parameters: First, an initial number of clusters k_{init}, which should be significantly larger than the expected value, and a threshold T for the Dip-p-value, which determines whether two clusters should be merged.

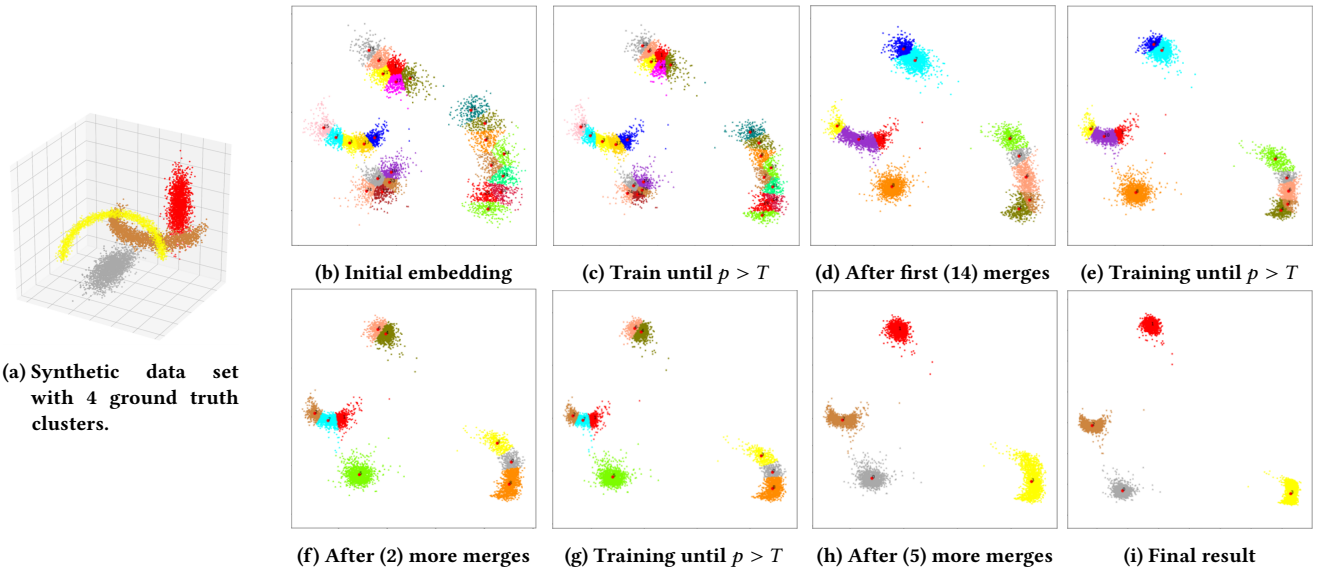
In our experiments, we use a simple feed-forward network architecture. However, other domain-specific architectures may be used as well. Afterwards, we execute k-means in the embedded space with the overestimated number of clusters to get the initial cluster centres and cluster assignments. Since we want to optimise the positions of the cluster centres simultaneously to the data embedding, we use the objects within the data set that are closest to the k-means centres (μ^{km}) as actual cluster centres.

$$\mu_i = \underset{x \in C_i}{\arg\min} \left(\left\| \text{enc}(x) - \mu_i^{\text{km}} \right\|_2^2 \right) \tag{2}$$

We apply the Dip-test to obtain the Dip-values for each pairwise combination of clusters i and j in the embedded space. Since the input of the Dip-test must be one-dimensional, we use the dot product $(\cdot)$ to project each point assigned to either one of the two clusters onto the connection line of the corresponding centres μ_i and μ_j. Normalisation is not necessary because the Dip-test is scale-invariant.

$$C_{i,j}^{1d} = \{\text{enc}(x) \cdot (\text{enc}(\mu_i) - \text{enc}(\mu_j)) \mid x \in C_i \cup C_j\}$$

This one-dimensional ($1d$) data set can then be used to calculate the Dip-value and consequently the Dip-p-value. Fig. 1 illustrates this idea.



(a) Synthetic data set with 4 ground truth clusters.

(b) Initial embedding (c) Train until $p > T$ (d) After first (14) merges (e) Training until $p > T$

(f) After (2) more merges (g) Training until $p > T$ (h) After (5) more merges (i) Final result

Figure 3: (a) A synthetic 3d data set to visualise our method. (b) k-means is executed on the 2d-embedding with $k_{\text{init}} \gg k_{\text{true}}$. (c) The autoencoder is trained until a Dip-p-value is found larger than the threshold T leading to (d) merging of clusters. (e) Training resumes, and the autoencoder repositions the clusters according to their Dip-p-value until T is reached again and (f) merging is instantiated. This process repeats itself in (g) and (h) until training finds no more clusters which need to be merged. The final result i) shows four almost perfectly separated clusters (NMI=0.99) in the embedded space.

The Dip-test might sometimes identify two groups as unimodal even though there is a large spatial distance between them if they differ greatly in size. To account for this, we calculate a second Dip-value that includes only the closest points of the larger cluster to the centre of the smaller cluster and the complete smaller cluster. Ultimately, the larger of the two Dip-values (and therefore the smaller Dip-p-value) is used. The intuition is that the transition between the clusters must also be unimodal. This idea is described in detail in Appendix C.

Furthermore, we define that each cluster with itself receives a Dip-value of 0 and therefore a Dip-p-value of 1. Since $p_{i,j} = p_{j,i}$, we get the following symmetric Dip-p-value matrix:

$$P = \begin{pmatrix} 1 & p_{1,2} & \cdots & p_{1,k} \\ p_{2,1} & 1 & \cdots & p_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k,1} & p_{k,2} & \cdots & 1 \end{pmatrix}$$

We normalise P by dividing each entry of the i-th row by the sum of the respective i-th row. The resulting matrix is termed $\hat{P}$.

Now we can start to optimise the autoencoder in a mini-batch fashion. We first encode all contained objects and all the cluster centres for each batch $\mathcal{B}$ in the data set. Then, we update the cluster assignments in a k-means fashion by assigning each point to the closest centre.

We define our novel clustering objective $\mathcal{L}_{clu}$ as

$$\mathcal{L}_{clu} = \frac{(1 + \text{std}(D_C))}{\text{mean}(D_C)} \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \sum_{i=1}^{k} \hat{P}_{c_x,i} ||\text{enc}(x) - \text{enc}(\mu_i)||_2^2, \quad (3)$$

where c_x is the label of the cluster x is assigned to, std$(\cdot)$ and mean$(\cdot)$ are the standard deviation and the mean of a set, respectively. D_C is the set of Euclidean distances between all cluster centres,

$$D_C = \left\{ \sqrt{||\text{enc}(\mu_i) - \text{enc}(\mu_j)||_2^2} \mid i \in [1, k-1] \text{ and } j \in [i+1, k] \right\}.$$

The row-wise normalisation of P ensures that the innermost sum of Eq. 3 is an affine sum. This means that the weights $\hat{P}_{c_x,i}$ sum up to 1 over all clusters i for a fixed x. The intuition is to force the network to strongest push a data point enc(x) to the centre it is assigned to - because 1 is the maximal possible Dip-p-value- but also proportionately pushing it in the direction of those centres μ_i whose points - according to the Dip-test- share a unimodal distribution with the points of the cluster of enc(x). Since the respective Dip-p-value $\hat{P}_{c_x,i}$ will then be large, the network will try to minimise the loss by reducing the respective distances to these centres.

The division by mean(D_C) is used to hinder the autoencoder of just reducing the embedding scale in order to minimise $\mathcal{L}_{clu}$. However, mean(D_C) could still be kept sufficiently large even if the network reduces the scale if it simultaneously pushes individual clusters far away. To prevent this, we include the term std(D_C).

Finally, we calculate $\mathcal{L}_{rec}$ using Eq. 1 and define the full loss function of DipDECK as

$$\mathcal{L} = \mathcal{L}_{rec} + \mathcal{L}_{clus}.$$

After each epoch, we update all labels by assigning each embedded point to its closest embedded centre to acknowledge the new autoencoder structure. Furthermore, we update the cluster centres similar to Eq. 2, but this time, we choose the points closest to the embedded cluster means instead of the k-means centres.

Algorithm 1: Pseudocode of DipDECK

Input: data set X, starting number of clusters k_{init}, Dip-p-value threshold T, number of epochs n
Output: *labels*, k

$k = k_{\text{init}}$
AE = pretrained autoencoder
(*kmCentres*, *labels*) = K-Means(AE.encode(X), k)
centres = find closest points to *kmCentres* (Eq. 2)
DipMatrix = calculate pairwise Dip-p-values of the clusters in the embedded space
$i = 0$
while $i < n$ **do**
  for $\mathcal{B}$ ***in*** X **do**
    if $i \neq 0$ **then**
      update *labels* of $\mathcal{B}$
    calculate $\mathcal{L} = \mathcal{L}_{rec}$ (Eq. 1) + $\mathcal{L}_{clu}$ (Eq. 3) for $\mathcal{B}$
    optimise AE using $\mathcal{L}$
  update all *labels*, *centres* (Eq. 4) and the *DipMatrix*
  i++
  // Start merging process
  while *max(DipMatrix)* $\geq T$ **do**
    k--
    merge clusters with highest Dip-p-value $\rightarrow$ add the new centre (Eq. 5) to *centres* and overwrite *labels*
    update the *DipMatrix*
    $i = 0$
return *labels*, k

$$\mu_i = \underset{x \in C_i}{\arg\min} \left(\left\| \text{enc}(x) - \frac{1}{|C_i|} \sum_{y \in C_i} \text{enc}(y) \right\|_2^2 \right) \tag{4}$$

Afterwards, the Dip-p-value matrices P and $\hat{P}$ are updated, and the cluster merging process starts.

3.1 Merging Process

To merge two clusters, we first check whether the maximum Dip-p-value in P is larger than a specified Dip-p-value threshold T. If so, the number of clusters will be reduced by one and the two clusters i and j that produce the corresponding Dip-p-value will be merged. Therefore, we assign the same label to all points assigned to these clusters, and a new centre will be created by using the closest point to the weighted mean of the old centres.

$$\mu_{new} = \underset{x \in C_i \cup C_j}{\arg\min} \left(\left\| \text{enc}(x) - \frac{|C_i|\text{enc}(\mu_i) + |C_j|\text{enc}(\mu_j)}{|C_i| + |C_j|} \right\|_2^2 \right) \tag{5}$$

With this new centre, we can then update the Dip-p-value matrices P and $\hat{P}$. In the end, we again check if the maximum Dip-p-value in P is larger than our threshold. If this is the case, we repeat the merging process. Otherwise, we reset the epoch counter and start optimising the autoencoder. In the first following epoch, we will not update the labels of the batches to allow the autoencoder to adjust to the new cluster structures and compress clusters that occupy a large space due to the prior merging.

Algorithm 1 shows the complete procedure of DipDECK.

Fig. 3 shows the procedure applied to a three-dimensional data set with four true clusters - two arbitrarily oriented moons and two arbitrarily oriented elongated Gaussian clusters. While this data set could probably be clustered successfully without the help of an autoencoder (e.g., with Spectral Clustering), it serves as a good illustration example to follow our clustering process in a visualisable two-dimensional embedded space.

4 EXPERIMENTAL EVALUATION

We evaluate our approach regarding several aspects. In Sec. 4.1, we compare DipDECK to various k-estimation methods while also considering the quality of the resulting clustering. Additionally, we compare the performance of related DC approaches. We then explore the interpretability of our found clusters in more detail (Sec. 4.2). In several robustness tests, we investigate the influence of our parameters on the results (Sec. 4.3).

Evaluation Metrics. For the quantitative evaluation, we consider the estimated number of clusters k and the normalised mutual information (NMI) [26]. NMI ranges in $[0, 1]$, where 1 indicates a perfect and 0 an arbitrary result. It is commonly used in the evaluation of unsupervised tasks.

Data sets. We conduct experiments on the image data sets USPS [14], MNIST [18], Fashion-MNIST (F-MNIST) [27], Kuzushiji-MNIST (K-MNIST) [4], Optdigits [5] and GTSRB [13], as well as the numerical data sets Pendigits [5] and Letterrecognition [5]. A detailed description of the data sets can be found in the Appendix A. All image data sets are reshaped into a one-dimensional vector and preprocessed by a channel-wise z-transformation. The numerical data sets are preprocessed using a feature-wise z-transformation.

Experimental Setup. We set the autoencoder dimensions to d-500-500-2000-m-2000-500-500-d. This is equal to the settings described in [28]. For the Dip-test, we have to project the data from dimension m to 1. The larger m, the more information gets lost due to this projection. Therefore, we choose $m = 5$. Furthermore, we use the ADAM optimiser and a constant learning rate of 0.001 for the pre-training as well as 0.0001 for the actual clustering process in all experiments. We set the initial k_{init} in all cases to 35, as this value is an overestimation for all correct k in the considered data sets. The batch size is set to 256, the Dip-p-value threshold to 0.9, the number of epochs for the pre-training to 100 and for the clustering process to 50. Our implementation of DipDECK is based on PyTorch (https://pytorch.org/) and can be downloaded at https://dmm.dbs.ifi.lmu.de/downloads.

We use the same parameters as for DipDECK if possible for the DC algorithms DEC, IDEC, DCN and VaDE. One exception is m, which we set to 10 since this is the dimensionality used in their respective papers. Additionally, we set the number of epochs for the clustering process to 150. All algorithm-specific parameters, including those for the non-DC algorithms, are set using the recommended values specified in the respective papers. For the algorithms X-means, G-means, PG-means, Dip-means, pDip-means in combination with an autoencoder (AE+), we use the same network architecture as for DipDECK, including $m = 5$. As these methods

Table 2: The resulting NMI values of various methods on different data sets. Additionally, for algorithms that can determine the number of clusters, this value is given. Best estimation of clusters and best NMI are highlighted in bold. Values marked with † could either not be executed due to memory constraints or aborted after 24h. All results are mean ± std of 10 executions.

Method	USPS		MNIST		F-MNIST		K-MNIST	
	k	NMI	k	NMI	k	NMI	k	NMI
Ground truth	10	-	10	-	10	-	10	-
DipDECK (ours)	**9.4 ± 0.47**	**0.846 ± 0.02**	**11.2 ± 0.50**	**0.889 ± 0.01**	**12.2 ± 0.75**	**0.679 ± 0.01**	15.8 ± 0.94	**0.658 ± 0.01**
X-means	35.0 ± 0.00	0.607 ± 0.01	35.0 ± 0.00	0.551 ± 0.00	35.0 ± 0.00	0.512 ± 0.00	35.0 ± 0.00	0.505 ± 0.00
G-means	35.0 ± 0.00	0.608 ± 0.00	35.0 ± 0.00	0.550 ± 0.00	35.0 ± 0.00	0.511 ± 0.00	35.0 ± 0.00	0.503 ± 0.00
PG-means	2.4 ± 0.63	0.136 ± 0.07	2.1 ± 0.79	0.175 ± 0.09	4.1 ± 1.93	0.312 ± 0.11	2.4 ± 0.76	0.135 ± 0.05
Dip-means	4.0 ± 0.00	0.438 ± 0.00	†	†	†	†	†	†
pDip-means	35.0 ± 0.00	0.617 ± 0.01	35.0 ± 0.00	0.554 ± 0.00	35.0 ± 0.00	0.511 ± 0.00	35.0 ± 0.00	0.502 ± 0.00
AE+X-means	2.0 ± 0.00	0.293 ± 0.01	18.1 ± 18.1	0.620 ± 0.10	24.4 ± 3.57	0.570 ± 0.01	2.1 ± 0.29	0.072 ± 0.05
AE+G-means	35.0 ± 0.00	0.669 ± 0.01	35.0 ± 0.00	0.686 ± 0.01	35.0 ± 0.00	0.550 ± 0.01	35.0 ± 0.00	0.569 ± 0.01
AE+PG-means	3.9 ± 1.24	0.379 ± 0.12	3.6 ± 1.22	0.453 ± 0.10	2.8 ± 0.71	0.387 ± 0.05	2.6 ± 0.76	0.103 ± 0.07
AE+Dip-means	6.8 ± 0.57	0.617 ± 0.02	†	†	†	†	†	†
AE+pDip-means	4.9 ± 1.31	0.519 ± 0.07	8.0 ± 1.60	0.705 ± 0.04	5.8 ± 0.57	0.522 ± 0.01	**12.4 ± 3.05**	0.474 ± 0.09
DEC	-	0.805 ± 0.02	-	0.847 ± 0.01	-	0.607 ± 0.03	-	0.551 ± 0.01
IDEC	-	0.811 ± 0.02	-	0.867 ± 0.01	-	0.641 ± 0.02	-	0.553 ± 0.02
DCN	-	0.748 ± 0.02	-	0.844 ± 0.02	-	0.617 ± 0.02	-	0.522 ± 0.04
VaDE	-	0.749 ± 0.03	-	0.806 ± 0.03	-	0.642 ± 0.02	-	0.558 ± 0.01

	Optdigits		Pendigits		Letterrecognition		GTSRB	
	k	NMI	k	NMI	k	NMI	k	NMI
Ground truth	10	-	10	-	26	-	5	-
DipDECK (ours)	**10.4 ± 0.76**	0.858 ± 0.02	14.0 ± 0.85	**0.817 ± 0.00**	**20.6 ± 1.80**	**0.491 ± 0.03**	**5.2 ± 0.87**	**0.612 ± 0.06**
X-means	35.0 ± 0.00	0.709 ± 0.01	35.0 ± 0.00	0.703 ± 0.01	35.0 ± 0.00	0.406 ± 0.01	35.0 ± 0.00	0.426 ± 0.01
G-means	35.0 ± 0.00	0.715 ± 0.01	35.0 ± 0.00	0.702 ± 0.01	35.0 ± 0.00	0.404 ± 0.01	35.0 ± 0.00	0.420 ± 0.01
PG-means	1.1 ± 0.29	0.024 ± 0.07	3.0 ± 1.41	0.335 ± 0.18	1.7 ± 0.61	0.068 ± 0.06	†	†
Dip-means	1.0 ± 0.00	0.000 ± 0.00	**10.4 ± 0.47**	0.691 ± 0.02	1.0 ± 0.00	0.000 ± 0.00	1.0 ± 0.00	0.000 ± 0.00
pDip-means	35.0 ± 0.00	0.709 ± 0.01	35.0 ± 0.00	0.705 ± 0.01	35.0 ± 0.00	0.418 ± 0.01	35.0 ± 0.00	0.427 ± 0.01
AE+X-means	12.8 ± 1.40	0.804 ± 0.01	35.0 ± 0.00	0.719 ± 0.01	2.0 ± 0.00	0.104 ± 0.02	2.2 ± 0.38	0.296 ± 0.01
AE+G-means	35.0 ± 0.00	0.730 ± 0.01	35.0 ± 0.00	0.711 ± 0.01	35.0 ± 0.00	0.476 ± 0.01	35.0 ± 0.00	0.455 ± 0.01
AE+PG-means	2.6 ± 1.61	0.290 ± 0.13	4.9 ± 1.93	0.534 ± 0.09	1.8 ± 0.93	0.048 ± 0.07	1.4 ± 0.47	0.048 ± 0.07
AE+Dip-means	1.0 ± 0.00	0.000 ± 0.00	9.4 ± 0.63	0.689 ± 0.01	1.0 ± 0.00	0.000 ± 0.00	1.0 ± 0.00	0.000 ± 0.00
AE+pDip-means	1.0 ± 0.00	0.000 ± 0.00	**10.4 ± 1.36**	0.695 ± 0.03	1.0 ± 0.00	0.000 ± 0.00	1.0 ± 0.00	0.000 ± 0.00
DEC	-	**0.885 ± 0.02**	-	0.763 ± 0.01	-	0.404 ± 0.03	-	0.555 ± 0.02
IDEC	-	0.864 ± 0.02	-	0.753 ± 0.01	-	0.443 ± 0.03	-	0.577 ± 0.06
DCN	-	0.857 ± 0.02	-	0.723 ± 0.02	-	0.450 ± 0.02	-	0.528 ± 0.04
VaDE	-	0.743 ± 0.04	-	0.718 ± 0.02	-	0.108 ± 0.02	-	0.200 ± 0.02

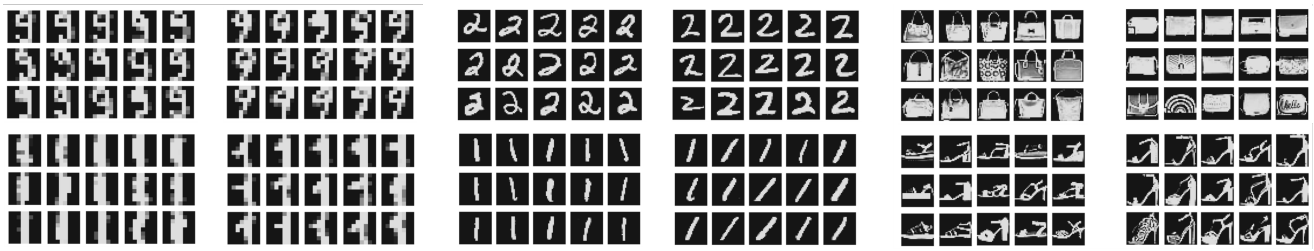
expect a maximal value of k, we set it to 35 to allow a fair comparison with our method, which starts with $k_{\text{init}} = 35$. We can not compare to SCDE because no source code is available.

4.1 Quantitative Experiments

The evaluation results can be seen in Table 2 (ARI results in Appendix B). Classical approaches like X-means often have trouble estimating a reasonable value for k. In most cases, they either stop close to their starting value of 1, i.e. they cannot find a decent cluster structure at all, or continue splitting clusters until they reach their termination condition, i.e. have found more than 35 clusters. In fact, their estimation of k deviates by more than 5 from the ground truth in all but 10 out of 73 experiments (excluding the k=1-estimations on GTSRB). Another drawback is that some experiments were stopped because they exceeded 24 hours, and for many approaches, a single run still took more than an hour. The autoencoder embedding does not lead to a notable improvement for our k-estimation competitors. Since the embedding here needs to be learned oblivious of the respective method, this shows that it is crucial to simultaneously integrate cluster number estimation into the actual DC process. DipDECK gives reliably good estimations for k and, at the same time, impressive clustering results, which rival, if not also surpass, other DC approaches.

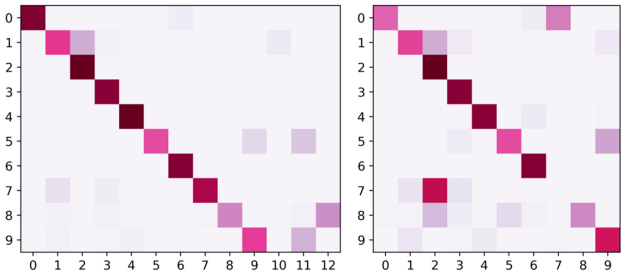
We wish to discuss two cases in more detail to highlight the characteristics of our approach.

4.1.1 Number of clusters. There are two cases where DipDECK's estimations do not seem to be the best ones at first glance. The first is Pendigits. Here, we estimate an average of $k = 14.0$, which does not seem impressive compared to the results of, e.g. Dip-means which estimates $k = 10.4$. However, one has to consider the quality of the found clusters as well. While our k-estimation is worse, our



(a) Subclasses in Optdigits (b) Subclasses in MNIST (c) Subclasses in F-MNIST

Figure 4: Meaningful substructures as found by DipDECK. Ground truth labels are equal for all top and all bottom images respectively in each subfigure (a), (b), (c). (a) Optdigits: [top] The digit '9' can be written with a round or a straight lower part and [bottom] '1' either as a straight line or with an additional skewed serif on top. (b) MNIST: [top] The digit '2' is either written with a loop or as a mirrored 'S'. [bottom] Without augmentation strategies, straight digits and rotated digits are split. (c) [top] For Fashion-MNIST, the class 'bags' can be split into bags with and without straps and the class 'sandal' [bottom] into flat sandals and such with high heels.



(a) DipDECK result (b) Dip-means result

Figure 5: Confusion matrices of DipDECK and Dip-means applied to Pendigits. The rows stand for the ground truth, while columns show the predicted clusters. Darkness of a square represents the number of data points in the cluster.

NMI is by far better with 0.817 vs 0.691. This shows that while we find more clusters, the ones we find better represent the ground truth. An obvious hypothesis follows: our approach partitions the ground truth clusters into multiple parts, while Dip-means splits the data set contrary to the ground truth. We verify this with the confusion matrices. A typical result of DipDECK with an NMI of 0.807 and 13 found clusters can be seen in Fig. 5a. The last two rows of the confusion matrix show that two ground-truth clusters are split into two. Furthermore, column 10 shows us a cluster consisting of less than 100 points, which can be easily dismissed. There are, of course, various data points assigned to the wrong clusters, but a comparison to the confusion matrix of Dip-means (Fig. 5b) shows us how much more helpful the result of DipDECK is.

We assume that DipDECK's overestimation of k on Pendigits is due to the structure found in a cluster, i.e. it splits clusters into meaningful sub-clusters. For example, in MNIST, digits are written in distinct styles. Splitting a cluster along these lines is reasonable, although not corresponding to the ground truth. In Pendigits, we can observe that clusters are split. However, we cannot verify that this split is based on a particular style since the data points are not images that can be visualised in a natural way. It is, however, possible for data sets as the mentioned MNIST. In Sec. 4.2, we take a closer look and see that DipDECK does indeed split along the lines of such meaningful sub-clusters.

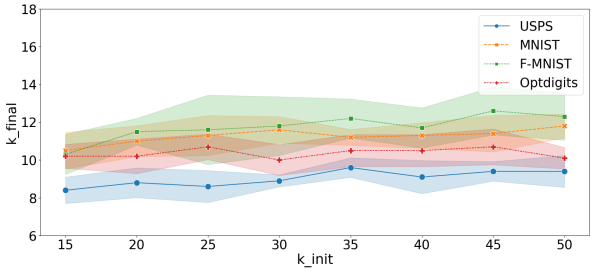
4.1.2 Variance. The second result, where DipDECK does not seem too impressive at first glance, is K-MNIST. The estimated k is relatively large compared to AE+pDip-means, but here again, the NMI (0.658 vs 0.474) shows us that the result of DipDECK consists of far purer clusters. Besides the NMI, there is a second factor to consider here. The variance of the k-estimation of AE+pDip-means is surprisingly large (> 3), with k-values in the range from 5 to 16. Contrary to that, DipDECK's standard deviation on K-MNIST is below 1, as it is on all data sets (except Letterrecognition with 1.80). DipDECK is very stable in this regard, and its results are reliable.

To substantiate the claim that our found clusters are very close to the ground truth, we tested against various DC approaches, in particular those comparable in their architecture. The results can be seen in Table 2. These methods are given the correct number of clusters as a parameter, which is usually unknown in unsupervised settings. DipDECK only needs a rough estimate of the cluster number but still performs better than all of these methods on all but one data set. This supports our claim that more initial micro-clusters are beneficial for the training of the autoencoder. DipDECK is the only tested DC procedure that utilises this idea.

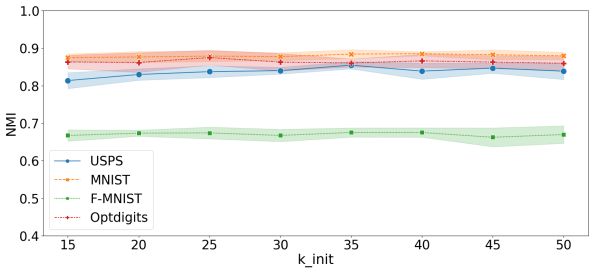
All of this shows us how important it is to evaluate k together with NMI. k alone only gives the number of clusters, while NMI estimates how 'correct' these clusters are. In Table 2, one can see that DipDECK has the best estimate of k in most cases and the best NMI results in all but one case. More importantly, DipDECK shows the best results in the combined evaluation of both - the k-estimation and the clustering task.

4.2 Qualitative Experiments

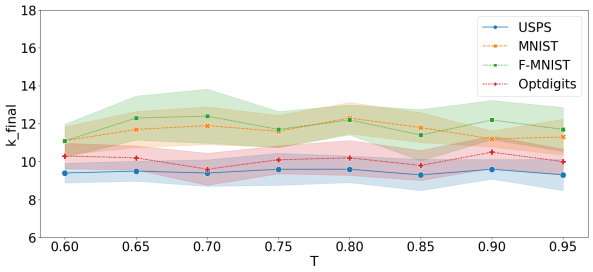
On some data sets (e.g. Pendigits), there is a slight tendency for DipDECK to overestimate the correct number of clusters (see Table 2). As stated in Sec. 4.1.1, this might be due to meaningful sub-clusters. In the case of data sets like Optdigits, MNIST or Fashion-MNIST, the resulting clusters might reveal substructures within the



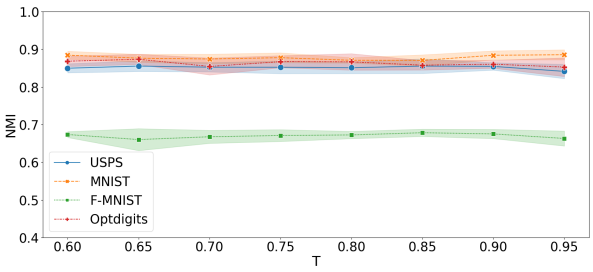
(a) k_{final} with increasing k_{init} ($T = 0.9$)



(b) NMI with increasing k_{init} ($T = 0.9$)



(c) k_{final} with increasing T ($k_{\text{init}} = 35$)



(d) NMI with increasing T ($k_{\text{init}} = 35$)

Figure 6: The robustness tests show the flexibility of the main input parameters of DipDECK. All tests are repeated 10 times, and all data sets contain 10 ground truth clusters. Points indicate the mean value, while the coloured area displays the standard deviation.

data set that appear reasonable to human perception. In Fig. 4, the clustering results of DipDECK on these data sets are shown. As can be observed, we indeed find certain meaningful sub-clusters.

Such a run of DipDECK, where the k for Optdigits was estimated as 12, is shown in Fig. 4a. Our algorithm extracted substructures that show two styles for the digits '1' and '9', respectively.

Fig. 4b shows a similar case for MNIST. The top panel shows that two styles for the digit '2' could be detected. The lower panel shows a well-known effect in the MNIST data set. The digit '1' is written as a straight line in both sub-clusters but with a different rotation. The same could sometimes be observed for the digit '5'. However, this can be easily overcome with augmentation strategies, where the network structure learns explicitly to be invariant regarding certain functions such as rotation or shifts.

F-MNIST is the most complex of the three considered data sets. Clustering results with DipDECK- as with other methods - often are mixed clusters containing objects of two to three ground truth classes. Apart from that, also meaningful substructures separating a ground truth cluster can be found. Fig. 4c shows how DipDECK separates 'bags' and 'sandals' into subclasses.

In Appendix D we show a similar analysis for K-MNIST.

4.3 Robustness Experiments

An essential aspect of every method is its stability regarding its (hyper-)parameters. For our method, this entails the DipDECK-specific parameters of the starting value k_{init} and its Dip-p-value threshold T. The correct number of clusters for the data sets in our experiments in Sec. 4.1 range from $k_{\text{true}} = 5$ to $k_{\text{true}} = 26$. For all experiments, we used a starting value of $k_{\text{init}} = 35$, and DipDECK could successfully estimate the correct k, thus making a solid first point for DipDECK's stability regarding k_{init}.

Table 3: DipDECK on different subsets of USPS. The first i digits were extracted from USPS (e.g. 0, 1, 2 and 3 for 4-USPS). The average of 10 runs is given (mean ± std).

Data set	k	NMI	Data set	k	NMI
1-USPS	1.0 ± 0.00	1.000 ± 0.00	6-USPS	5.9 ± 0.67	0.859 ± 0.02
2-USPS	2.0 ± 0.00	0.978 ± 0.00	7-USPS	6.7 ± 0.63	0.858 ± 0.02
3-USPS	3.1 ± 0.29	0.895 ± 0.04	8-USPS	7.4 ± 0.47	0.866 ± 0.02
4-USPS	4.2 ± 0.570	0.862 ± 0.06	9-USPS	8.7 ± 0.44	0.862 ± 0.01
5-USPS	5.0 ± 0.00	0.887 ± 0.02	10-USPS	9.4 ± 0.47	0.851 ± 0.02

4.3.1 ***k_{init}.*** To further investigate the stability regarding k_{init}, we conducted the following experiment. We executed DipDECK on all subsets of the USPS data set consisting of the first i digits, i.e., i-USPS consists of the digits $0, 1, \ldots, i-1$. If k_{init} significantly influences DipDECK, it will have problems estimating the correct number of clusters for data sets with similar characteristics but different correct k. The results can be seen in Table 3. DipDECK finds exactly as many clusters as there are in the data set. For 5-USPS as an example, it finds the correct k in 10 out of 10 runs with an average NMI of 0.89. The only difference in these data sets is the number of clusters. The characteristics of the data sets are the same, as well as the starting value of $k_{\text{init}} = 35$. Independently of k_{init}, DipDECK managed to estimate the correct k, whether it was 1 or 10.

Further stability results can be taken from Fig. 6a. We chose four of the data sets from our main experiments in Table 2 and executed DipDECK with a starting k_{init} ranging from 15 to 50. Despite this

wide range of values, DipDECK shows almost no difference regarding the final k it converges to, with only a very slight increase in k with increased k_{init}. Optdigits, as an example, ranges from an average found k_{final} from 10.0 to 10.7 and USPS from 8.4 to 9.6. In earlier sections, we have already seen how important it is to keep the NMI score in mind when evaluating k-estimates. These can be seen in Fig. 6b and are consistently good. DipDECK finds reliable estimates for k while keeping NMI stable, showing us its capability of finding the correct k independently of the chosen k_{init}.

4.3.2 ***Dip-p-value threshold T.*** The second parameter set by us is the Dip-p-value threshold chosen as 90%. This value is a commonly used threshold regarding significance, but we, nevertheless, also tested DipDECK for different thresholds. The results can be seen in Fig. 6c. Following the experiments, the merging process and the final number of clusters is relatively unaffected by this value. When evaluating various runs of DipDECK in more detail, it becomes evident that the Dip-test is rarely close to a medium value. If two separate clusters are tested, the Dip-p-value is often no higher than 10%, but if a cluster is cut in two and its parts are tested together, the Dip-p-value often surges above 99%. This, again, shows the impressive stability of DipDECK regarding its parameters.

5 CONCLUSION

In this paper, we introduced DipDECK, a novel DC algorithm with an implicit estimation of the number of clusters. To the best of our knowledge, we are the first DC approach with a simultaneous k-estimation. Existing methods in this field heavily rely on the given number of clusters in a data set. In this regard, our algorithm offers considerable relaxation, as one can heavily overestimate this number. Based on the Dip-test connecting structures are identified, and the corresponding clusters are merged.

Furthermore, DipDECK is more flexible regarding the shape of the clusters, while most of the previously proposed DC methods are k-means based and inherit the Gaussian cluster assumption. This is also true for many classical k-estimation methods such as X-means or Dip-means. Additionally, they are limited in their scalability and are, therefore, no longer adequate for modern large and high-dimensional data sets.

In extensive experiments, we demonstrate that our cluster performance and our k-estimation results most often outperform relevant k-estimation and DC baselines on various data sets. We further show that our results are highly stable across multiple runs and robust regarding our two parameters.

DipDECK offers several possibilities to extend this work. The bottom-up like cluster number estimation provides a reasonable basis for a hierarchical clustering algorithm, especially since we could observe that our substructures in case of overestimation are often meaningful for human perception and pure in terms of NMI.

ACKNOWLEDGMENTS

We owe our gratitude to Dominik Mautz and Lukas Miklautz for their support during the creation of this work.

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

REFERENCES

[1] Horst Bischof, Aleš Leonardis, and Alexander Selb. 1999. MDL principle for robust vector quantisation. *Pattern Analysis & Applications* 2, 1 (1999), 59–72.
[2] Christian Böhm, Christos Faloutsos, Jia-Yu Pan, and Claudia Plant. 2006. Robust information-theoretic clustering. In *SIGKDD*. 65–75.
[3] Theofilos Chamalis and Aristidis Likas. 2018. The Projected Dip-Means Clustering Algorithm. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*.
[4] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. 2018. Deep learning for classical Japanese literature. *arXiv preprint arXiv:1812.01718* (2018).
[5] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[6] L. Duan, C. Aggarwal, S. Ma, and S. Sathe. 2019. Improving Spectral Clustering with Deep Embedding and Cluster Estimation. In *ICDM*. 170–179.
[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *SIGKDD*, Vol. 96. 226–231.
[8] Yu Feng and Greg Hamerly. 2007. PG-means: learning the number of clusters in data. In *Advances in neural information processing systems*. 393–400.
[9] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE international conference on computer vision*. 5736–5745.
[10] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved Deep Embedded Clustering with Local Structure Preservation. In *IJCAI*.
[11] Greg Hamerly and Charles Elkan. 2004. Learning the k in k-means. In *Advances in neural information processing systems*. 281–288.
[12] J. A. Hartigan and P. M. Hartigan. 1985. The Dip Test of Unimodality. *Ann. Statist.* 13, 1 (03 1985), 70–84. https://doi.org/10.1214/aos/1176346577
[13] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. 2013. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *IJCNN*. IEEE, 1–8.
[14] Jonathan J. Hull. 1994. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence* 16, 5 (1994), 550–554.
[15] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *IJCAI*. 1965–1972. https://doi.org/10.24963/ijcai.2017/273
[16] Argyris Kalogeratos and Aristidis Likas. 2012. Dip-means: an incremental clustering method for estimating the number of clusters. In *Advances in Neural Information Processing Systems*.
[17] Yann Lecun. 1987. *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)*. Universite P. et M. Curie (Paris 6).
[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[19] Samuel Maurus and Claudia Plant. 2016. Skinny-dip: clustering in a sea of noise. In *SIGKDD*. 1055–1064.
[20] Dominik Mautz, Claudia Plant, and Christian Böhm. 2019. Deep embedded cluster tree. In *ICDM*. IEEE, 1258–1263.
[21] Tom Monnier, Thibault Groueix, and Mathieu Aubry. 2020. Deep Transformation-Invariant Clustering. In *NeurIPS*.
[22] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. 2019. ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. In *AAAI*. 4610–4617.
[23] Dan Pelleg and Andrew W. Moore. 2000. X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters. In *ICML (ICML '00)*.
[24] B. Schelling, L. Bauer, S. Behzadi Soheil, and C. Plant. 2020. Utilizing Structure-rich Features to improve Clustering. In *ECML-PKDD 2020*.
[25] B. Schelling and C. Plant. 2018. DipTransformation: Enhancing the Structure of a Dataset and Thereby Improving Clustering. In *ICDM*.
[26] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *Journal of Machine Learning Research* (2010).
[27] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:cs.LG/1708.07747 [cs.LG]
[28] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In *ICML (JMLR Workshop and Conference Proceedings)*.
[29] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *ICML*.
[30] Jianwei Yang, Devi Parikh, and Dhruv Batra. 2016. Joint Unsupervised Learning of Deep Representations and Image Clusters. In *CVPR*.
[31] Lihi Zelnik-Manor and Pietro Perona. 2005. Self-Tuning Spectral Clustering. In *Advances in Neural Information Processing Systems*.

APPENDIX

In the appendix, we present details about the data sets used in the experiments (Appendix A), add ARI results for our quantitative experiments (Appendix B) and give a detailed explanation of how we deal with imbalanced micro-cluster sizes (Appendix C). We also take a closer look at the sub-structures found in K-MNIST (Appendix D).

A DATA SETS

A summary of our used data sets is given in Table 4.

Table 4: Information regarding the used data sets.

Data set	# Samples	# Features	# Classes
USPS	9298	256	10
MNIST	70000	784	10
F-MNIST	70000	784	10
K-MNIST	70000	784	10
Optdigits	5620	64	10
Pendigits	10992	16	10
Letterrecognition	20000	16	26
GTSRB	7860	3072	5

USPS [14]: Greyscale image data set consisting of 9298 handwritten digits (0 to 9) with a size of 16×16 pixels.

MNIST [18]: Greyscale image data set consisting of 70000 handwritten digits (0 to 9) with a size of 28×28 pixels.

Fashion-MNIST (F-MNIST) [27]: Greyscale image data set consisting of 70000 articles from the *Zalando* online store. Each sample belongs to one of 10 product groups and has a size of 28×28 pixels.

Kuzushiji-MNIST (K-MNIST) [4]: Greyscale image data set consisting of 70000 Kanji characters (10 different characters, each representing one column of hiragana) with a size of 28×28 pixels.

Optdigits [5]: This 8×8 image data set consists of 5620 samples, each representing a digit (0 to 9). Each pixel depicts the number of marked pixels within a 4×4 block of the original 32×32 bitmaps.

Pendigits [5]: This data set consists of 10992 vectors of length 16, representing 8 coordinates. The coordinates were taken from the task of writing digits (0 to 9) on a tablet.

Letterrecognition [5]: This data set consists of 20000 data samples whereby each sample represents one of the 26 capital letters in the English alphabet. All samples are composed of 16 numerical stimuli describing the respective letter.

GTSRB [13]: Image data set containing 32×32 images of German traffic signs. Each image contains 3 colour channels. We used a subset of 5 different signs ('Speed limit (50)', 'No passing', 'Ahead only', 'Right of way', 'Attention'), resulting in 7860 images.

B ARI RESULTS

In addition to the quantitative evaluation via NMI, we calculated the corresponding ARI values. ARI also ranges from 0 to 1, where 1 marks a perfect result. The results can be seen in Table 5.

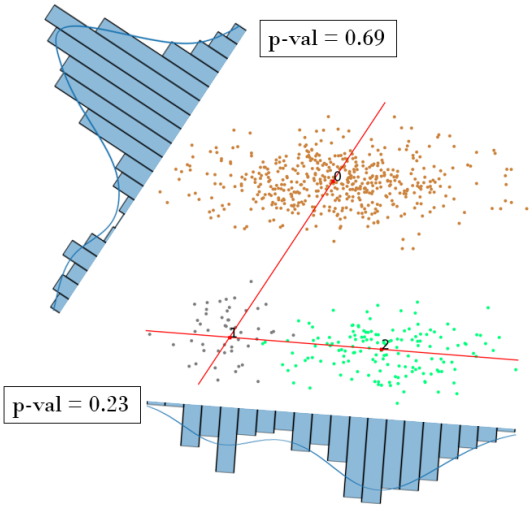


Figure 7: Synthetic data set consisting of three Gaussian clusters containing 500 (brown), 50 (grey) and 150 (green) objects. The left histogram shows the distribution of clusters 0 (brown) and 1 (grey) projected to the connection line (red) between their respective centres. The lower histogram shows the analogous distribution of clusters 1 (grey) and 2 (green). The Dip-p-values of those distributions are 0.69 **for clusters 0 and 1 and** 0.23 **for clusters 1 and 2.**

C IMBALANCED CLUSTERS

We mentioned in Sec. 3 that there are problems applying the Dip-test if two clusters are very different in size. To analyse this, we created a two-dimensional data set with three Gaussian clusters of different sizes as an example. The setting is displayed in Fig. 7. Cluster 0 (brown) contains 500, cluster 1 (grey) 50 and cluster 2 (green) 150 points. It can be seen that cluster 1 lies relatively close to cluster 2. It is conceivable that these two clusters belong together. Cluster 0 and cluster 1, on the other hand, should not be merged, as a clear separation can be observed. However, the Dip-p-values indicate the opposite. With a value of 0.69, cluster 1 is said to have a higher affiliation with cluster 0 than with cluster 2 (Dip-p-value of 0.23). This means that in the case of DipDECK, contrary to the natural perception, the autoencoder is more likely to move cluster 1 closer to cluster 0 than to cluster 2. This is because cluster 1 is so small relative to cluster 0 that it is perceived merely as outliers and not as a second mode. In the histogram of cluster 1 and cluster 2, on the other hand, two distinct modes can be seen, which prevents a higher value. When the autoencoder starts moving the clusters towards each other, these effects are amplified.

To counteract this behaviour, in addition to the Dip-value of the fully combined clusters, we calculate the Dip-value of the transition from one cluster to another if the sizes differ significantly. For this, we define a threshold S. If $|C_j| > S|C_i|$, we calculate a second Dip-value by only taking the $S|C_i|$ points closest to μ_i from C_j in combination with all points from C_i. This gives us two different Dip-values, from which we choose the maximum.

Table 5: The resulting ARI values of the experiments as described in the Sec. 4

Method	USPS	MNIST	F-MNIST	K-MNIST	Optdigits	Pendigits	Letterrec.	GTSRB
DipDECK (ours)	**0.834 ± 0.05**	**0.860 ± 0.02**	**0.494 ± 0.02**	**0.519 ± 0.02**	0.833 ± 0.02	**0.740 ± 0.01**	**0.221 ± 0.03**	0.518 ± 0.09
X-means	0.315 ± 0.01	0.271 ± 0.01	0.230 ± 0.01	0.263 ± 0.01	0.415 ± 0.01	0.430 ± 0.02	0.155 ± 0.01	0.150 ± 0.01
G-means	0.313 ± 0.00	0.270 ± 0.01	0.232 ± 0.01	0.259 ± 0.01	0.432 ± 0.02	0.423 ± 0.02	0.154 ± 0.01	0.148 ± 0.01
PG-means	0.041 ± 0.05	0.008 ± 0.04	0.139 ± 0.06	0.067 ± 0.02	0.007 ± 0.02	0.189 ± 0.11	0.011 ± 0.01	†
Dip-means	0.292 ± 0.00	†	†	†	0.000 ± 0.00	0.537 ± 0.02	0.000 ± 0.00	0.000 ± 0.00
pDip-means	0.333 ± 0.02	0.261 ± 0.00	0.227 ± 0.00	0.248 ± 0.01	0.391 ± 0.01	0.409 ± 0.01	0.161 ± 0.01	0.148 ± 0.00
AE+X-means	0.143 ± 0.00	0.409 ± 0.09	0.362 ± 0.03	0.030 ± 0.03	0.752 ± 0.01	0.517 ± 0.02	0.023 ± 0.01	0.239 ± 0.01
AE+G-means	0.436 ± 0.02	0.430 ± 0.02	0.300 ± 0.02	0.327 ± 0.01	0.460 ± 0.02	502 ± 0.01	0.217 ± 0.01	0.178 ± 0.01
AE+PG-means	0.157 ± 0.09	0.142 ± 0.27	0.206 ± 0.05	0.055 ± 0.04	0.09 ± 0.06	0.347 ± 0.11	0.003 ± 0.08	0.022 ± 0.04
AE+Dip-means	0.537 ± 0.04	†	†	†	0.000 ± 0.00	0.564 ± 0.01	0.000 ± 0.00	0.000 ± 0.00
AE+pDip-means	0.395 ± 0.10	595 ± 0.07	0.304 ± 0.10	0.348 ± 0.08	0.000 ± 0.00	0.581 ± 0.04	0.000 ± 0.00	0.000 ± 0.00
DEC	0.716 ± 0.02	0.755 ± 0.04	0.441 ± 0.04	0.411 ± 0.01	**0.850 ± 0.05**	0.621 ± 0.03	0.152 ± 0.02	0.489 ± 0.02
IDEC	0.733 ± 0.03	0.807 ± 0.02	0.475 ± 0.03	0.410 ± 0.03	0.811 ± 0.04	0.634 ± 0.02	0.197 ± 0.03	**0.520 ± 0.07**
DCN	0.642 ± 0.03	0.790 ± 0.03	0.449 ± 0.03	0.360 ± 0.05	0.829 ± 0.04	0.587 ± 0.04	0.209 ± 0.02	0.437 ± 0.05
VaDE	0.605 ± 0.08	0.716 ± 0.05	0.480 ± 0.02	0.401 ± 0.02	0.621 ± 0.06	0.586 ± 0.05	0.023 ± 0.01	0.102 ± 0.02

Table 6: Impact of different choices of S on the Dip-p-values from the example displayed in Fig. 7.

S	1	2	3	4	5	∞
$p_{0,1}$	0.0012	0.0082	0.0240	0.0544	0.1039	0.6892
$p_{1,2}$	0.0196	0.0879	0.2307	0.2307	0.2307	0.2307

In total, there are three possible cases to determine the Dip-value:

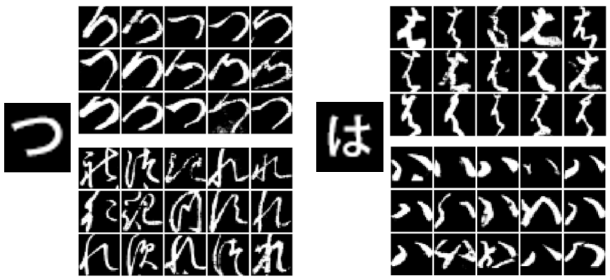
$$d_{i,j} = \begin{cases} \max\{\mathrm{Dip}(C^{1d}_{i,j}), \mathrm{Dip}(C^{1d}_{i,j_S})\}, & \text{if } |C_j| > S|C_i| \\ \max\{\mathrm{Dip}(C^{1d}_{i,j}), \mathrm{Dip}(C^{1d}_{i_S,j})\}, & \text{if } |C_i| > S|C_j| \\ \mathrm{Dip}(C^{1d}_{i,j}), & \text{otherwise} \end{cases}$$

Here $C^{1d}_{i_S,j}$ only contains the $S|C_j|$ nearest points of C_i to μ_j in the embedded space, and C^{1d}_{i,j_S} is defined analogously.

To see the impact of the choice of S, Table 6 can be inspected, which shows the effects using the example data set described above. 2 seems to be a good compromise to solve the described problem while not preventing interesting cluster connections, like the one between clusters 1 and 2. Note, however, that all of the S values (except ∞) lead to Dip-p-values that correspond to our perception of the relative connections between the clusters 0, 1 and 2.

D K-MNIST QUALITATIVE EXPERIMENTS

K-MNIST seems to be one of the data sets where DipDECK does not perform too well, as the average estimated number of clusters is with 15.8 significantly larger than the desired amount of 10. However, as with MNIST, F-MNIST, and Optdigits, we can again show that DipDECK finds meaningful subdivisions within the ground truth clusters. For example, consider the letter 'tsu' (displayed in Fig. 8a). We can see that the first identified cluster in the top well represents the modern counterpart. The lower cluster on the other hand looks very different. In this case it seems reasonable to perform a separation. The same applies to the character 'ha' (displayed in Fig. 8b). Here, the objects from the upper cluster show certain similarities to the second sign of the modern counterpart. The objects of the lower cluster, on the contrary, often consist of two single dashes. The examples show once again that splitting ground truth clusters can lead to additional insights into a data set in some cases.



(a) The character 'tsu' **(b) The character 'ha'**

Figure 8: Found substructures of DipDECK within the characters 'tsu' and 'ha' in the K-MNIST data set. The single characters on the left show the modern hiragana counterparts. On the right we see two different styles of writing found by DipDECK.

C.4 The DipEncoder: Enforcing Multimodality in Autoencoders

Authors:
Leiber, Collin and Bauer, Lena G. M. and Neumayr, Michael and Plant, Claudia and Böhm, Christian

Abstract:
Hartigan's Dip-test of unimodality gained increasing interest in unsupervised learning over the past few years. It is free from complex parameterization and does not require a distribution assumed a priori. A useful property is that the resulting Dip-values can be derived to find a projection axis that identifies multimodal structures in the data set. In this paper, we show how to apply the gradient not only with respect to the projection axis but also with respect to the data to improve the cluster structure. By tightly coupling the Dip-test with an autoencoder, we obtain an embedding that clearly separates all clusters in the data set. This method, called DipEncoder, is the basis of a novel deep clustering algorithm. Extensive experiments show that the DipEncoder is highly competitive to state-of-the-art methods.

Venue:
KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14-18, 2022, ACM, 2022.
CORE ranking: A (http://portal.core.edu.au/conf-ranks/26/)*
Acceptance rate: 15.3%

DOI:
https://doi.org/10.1145/3534678.3539407

Thesis-Reference:
[LBN+22]

Division of Work:

Collin Leiber
Development of the idea; Formulation of the optimization strategy and the loss function; Formulation of the label update procedure; Implementing the algorithm; Designing the experiments; Executing the experiments; Writing the majority of the paper; Creating the visualizations; Creating the presentation slides, poster and video; Presenting at the conference

Lena G. M. Bauer
Frequent discussion on the procedure; Checking the formulas and refinement of the mathematical notation; Helping to design parts of the experiments

Michael Neumayr
Frequent discussion on the procedure; Creating a supervised prototype for dimensionality reduction; Designing parts of the experiments regarding supervised learning; Writing parts of the related work regarding the dimensionality reduction techniques and parts of the calculation of the Dip-value in the methodology section

Claudia Plant
Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

Christian Böhm
Frequent discussion on the methodology; Frequent suggestions for improving the presentation; Refinement of the final draft; Mentoring and supervision

The DipEncoder: Enforcing Multimodality in Autoencoders

Collin Leiber
LMU Munich
Munich, Germany
leiber@dbs.ifi.lmu.de

Lena G. M. Bauer
Faculty of Computer Science
UniVie Doctoral School Computer Science
ds:UniVie
University of Vienna, Vienna, Austria
lena.bauer@univie.ac.at

Michael Neumayr
LMU Munich
Munich, Germany
michael.neumayr@campus.lmu.de

Claudia Plant
Faculty of Computer Science
ds:UniVie
University of Vienna, Vienna, Austria
claudia.plant@univie.ac.at

Christian Böhm
Faculty of Computer Science
University of Vienna, Vienna, Austria
christian.boehm@univie.ac.at

ABSTRACT

Hartigan's Dip-test of unimodality gained increasing interest in unsupervised learning over the past few years. It is free from complex parameterization and does not require a distribution assumed a priori. A useful property is that the resulting Dip-values can be derived to find a projection axis that identifies multimodal structures in the data set. In this paper, we show how to apply the gradient not only with respect to the projection axis but also with respect to the data to improve the cluster structure. By tightly coupling the Dip-test with an autoencoder, we obtain an embedding that clearly separates all clusters in the data set. This method, called DipEncoder, is the basis of a novel deep clustering algorithm. Extensive experiments show that the DipEncoder is highly competitive to state-of-the-art methods.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Computing methodologies** → **Cluster analysis**; *Dimensionality reduction and manifold learning*; *Neural networks.*

KEYWORDS

Hartigan's Dip-test, Deep Clustering, Dimensionality reduction

ACM Reference Format:
Collin Leiber, Lena G. M. Bauer, Michael Neumayr, Claudia Plant, and Christian Böhm. 2022. The DipEncoder: Enforcing Multimodality in Autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539407

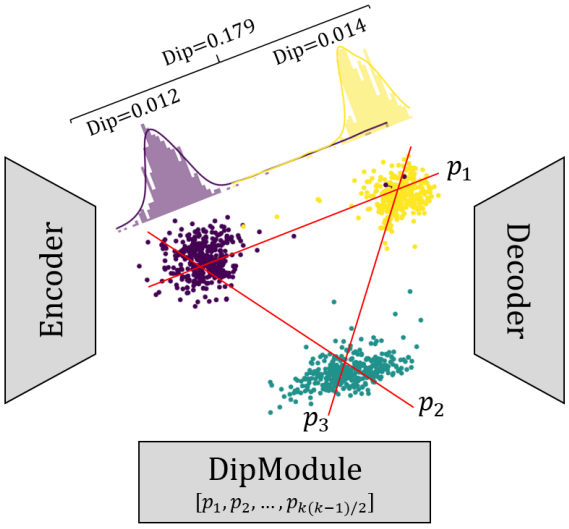


Figure 1: Architecture of the DipEncoder. The illustration shows the two-dimensional result of the DipEncoder on a subset of the Optdigits data set. We can see that each combination of clusters receives its own projection axis within the DipModule. The histogram depicts the purple and yellow clusters projected onto their corresponding projection axis p_1. In addition, the figure shows the Dip-values of the purple cluster (0.012), the yellow cluster (0.014), and the combination of the two (0.179). From this, we can conclude that both clusters are unimodal while the combination is multimodal.

1 INTRODUCTION

The interest in analyzing large amounts of high-dimensional data such as images, videos or texts increased significantly in recent years. Since such data sets are highly complex both in terms of their interpretability and the time it takes to process them, particular analysis methods are usually required.

An established strategy to handle high-dimensional data is to run a dimensionality reduction technique before the desired analysis method. The most common technique is probably the Principal Component Analysis (PCA) [10]. With the growing availability of computing power, Neural Networks (NNs) have also become more popular. Due to their high abstraction capabilities, they offer a range of powerful analysis options. Special architectures can even favor the performance concerning certain data types (e.g.,

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539407

Convolutional Neural Networks (CNNs) [23] for image data). A strategy that combines dimensionality reduction with NNs is to make use of an autoencoder (AE) [1]. This feed-forward NN learns a lower-dimensional embedding of the input data set. One can then execute further analysis procedures on this embedding.

In recent years, the field of clustering has increasingly taken up this idea. Corresponding procedures, also referred to as Deep Clustering (DC) methods, usually use an AE to learn an embedding in which the actual clustering procedure is applied, allowing the clustering objective to be updated simultaneously to the embedding. Hereby, the runtime can drop significantly due to the lower dimensionality and we counteract the curse of dimensionality while at the same time using the abstraction capabilities of the AE.

In this paper, we present the DipEncoder. This NN combines an AE with Hartigan's Dip-test of unimodality [14]. The Dip-test is a parameter-free statistical test that returns a Dip-value within the interval $(0, 0.25]$, which specifies the multimodality of a one-dimensional data set. Dip-values close to 0 indicate unimodality of the input samples, while larger values indicate that the samples contain at least two modes. Our goal is to use this test to create an embedding that clearly separates different groups of data. In other terms, we want to achieve an embedding that shows high modality between each combination of clusters. Figure 1 illustrates this idea by the high Dip-value of 0.179 between the purple and yellow cluster. However, we cannot simply maximize the Dip-value since we have to be careful that this process does not pull one of the clusters apart to achieve a multimodal structure. Therefore, we also want to minimize the modality within the separate clusters. In Figure 1 this corresponds to the small Dip-values of 0.012 for the purple and 0.014 for the yellow cluster. Since the Dip-test can only process one-dimensional samples, we create individual projection axes for each combination of clusters and store them within the so-called DipModule. These axes are represented in Figure 1, for example, by the red line $p1$ between the purple and yellow cluster. Using this architecture, we can leverage the gradient of the Dip-test to optimize the projection axes simultaneously to the embedding. We further use this idea to show how we can use the Dip-test to update the cluster labels. Building on this, we present a novel DC algorithm based solely on the Dip-test which does not require clustering-specific parameters other than the number of clusters k.

Our main contributions can be summarized as follows:

- First, we present the previously unused gradient of the Dip-value with respect to the data.
- We show how to use the gradients of the Dip-value in combination with an AE for supervised dimensionality reduction.
- Based on this, we develop a procedure that updates the cluster labels using only the Dip-test.
- This method is extended to create a novel Deep Clustering algorithm that does not require clustering-specific parameters apart from the number of clusters.

2 RELATED WORK

In the following, we describe a few methodologies that underlie our proposal. First, we describe the Dip-test in detail as it is an essential part of our procedure. Then, we briefly discuss dimensionality reduction techniques and basic Deep Clustering algorithms.

The Dip-test: The Dip-test [14] of unimodality is a statistical test that measures how multimodal a given one-dimensional data set is. It returns a Dip-value $\text{dip} \in (0, 0.25]$, which indicates unimodality if it is close to zero. Since the Dip-test is parameter-free, we do not have to state any underlying distribution function. Therefore, $\text{dip} \approx 0$ regardless of whether we execute the Dip-test on samples from a single Gaussian, Laplacian, uniform, or any other unimodal distribution. On the other hand, $0 \ll \text{dip} \leq 0.25$ if we have samples from distributions with multiple distinctive modes. Hartigan and Hartigan showed how to efficiently calculate the Dip-value on a sorted data set of size N with a complexity of $O(N)$ [14]. For this purpose, a modal interval is used, which indicates the steepest slope in the empirical cumulative distribution function (ECDF). These characteristics make the Dip-test very interesting for the Data Science community, which is why it has already been used for various purposes.

Since no distribution of the data has to be assumed, it is particularly suitable for unsupervised learning. One of the first clustering algorithms using the Dip-test is Dip-means [19] which aims at identifying the number of clusters. Here, the Dip-test is calculated on the distances between all objects within a cluster. A new cluster is added if the distances do not show a unimodal distribution. Projected Dip-means [2] pursues the same objective but projects the data onto projection axes and applies the Dip-test on these projected one-dimensional values. SkinnyDip [28] recursively analyzes the features of a data set by interpreting each modal interval returned by the Dip-test as a cluster. This idea is continued by StrDip [27] to cluster streaming data. Nr-Dipmeans [30] attempts to determine the number of clusters in a non-redundant clustering setting. Therefore, each cluster is split into two, and the objects are projected onto the line connecting the two new centers. The initial cluster is kept if the Dip-test indicates unimodality; otherwise, the procedure continues with the two new clusters. The first clustering algorithm that combines the Dip-test with Deep Learning is DipDECK [25]. Here, clustering is done in the embedded space of an AE. Initially, the technique heavily overestimates the number of clusters. If the data points within two microclusters show high unimodality, they are merged, and the embedding can be further updated.

DipTransformation [34] shows that the Dip-test can also be used for data preprocessing. It uses the Dip-test to transform and scale a data set so that its most important features are highlighted. [28] and [33] go one step further. They take advantage of the fact that we can deduce the gradient of the Dip-value with respect to the projection axis, as shown in [22]. This allows them to identify cluster-friendly subspaces by picking out those projections which yield the highest multimodality. The differentiability is a convenient feature of the Dip-test. We want to extend this idea by additionally using the gradient with respect to the data.

Dimensionality reduction techniques: Methods that reduce the dimensionality of a data set fall into two main categories: unsupervised and supervised.

Principal Component Analysis (PCA) [10] is probably the best-known dimensionality reduction technique and is a common preprocessing step when analyzing a data set. It rotates the features space such that only the components with the highest variances remain. Independent Component Analysis (ICA) [18] is not a dimensionality reduction technique per definition. However, it also

finds a lower-dimensional basis of the data by searching for statistically independent components. t-Distributed Stochastic Neighbor Embedding (t-SNE) [35] tries to preserve local structures in high-dimensional data sets while reducing the dimensionality. Therefore, it converts distances in the original feature space into probabilities of whether an object would pick another as a neighbor. Uniform Manifold Approximation and Projection (UMAP) [31] pursues a similar goal. However, it uses a more extensive mathematical basis by leveraging Riemannian geometry. Another way to perform a dimensional reduction is to use an autoencoder (AE) [1]. This unsupervised NN consists of two parts, an encoder enc(·) that transforms the input to the embedding and a decoder dec(·) that tries to restore the data from the embedding to its original state. This basic idea is also presented in Figure 1, where we have a two-dimensional embedding. The AE is usually trained by a batch-wise optimization of the reconstruction loss $\mathcal{L}_{rec}$.

$$\mathcal{L}_{rec}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} ||x - \text{dec}(\text{enc}(x))||_2^2, \tag{1}$$

where $\mathcal{B} \subseteq X$ is one batch of the data set $X \subseteq \mathbb{R}^d$ and $||\cdot||_2^2$ denotes the squared Euclidean distance.

Until now, we only discussed unsupervised techniques. However, some methods use known cluster labels to achieve a lower dimensionality. One such supervised approach is the Linear Discriminant Analysis (LDA) [9]. LDA identifies a subspace by minimizing the intra-cluster variance while maximizing the inter-cluster variance. Another approach is Partial Least Squares (PLS) [36]. This method searches for structures in the data set that maximize the covariance with the labels.

We present the DipEncoder, which also falls into the group of supervised dimensionality reduction techniques since we want to improve the embedding of an AE by arranging objects of a common group unimodally and those of different groups multimodally.

Many of the mentioned methods have already been successfully combined with data-mining approaches to, for instance, create subspace or non-redundant clustering algorithms. Examples are Orth [5] (using PCA), LDA-k-means [6] (using LDA) or generally DC algorithms (using AEs).

Deep Clustering: Since our final product is a Deep Clustering (DC) algorithm, we want to briefly discuss corresponding procedures. One of the first methods that combine a simple AE with a centroid-based clustering objective is DEC [38] and its successor IDEC [13]. They optimize their network by minimizing the Kullback-Leibler divergence between soft cluster labels and an auxiliary target distribution. While DEC uses the reconstruction loss of the AE only for pretraining, IDEC integrates it into the cost function of the primary clustering method. DCN [39] is more oriented towards the original k-means algorithm and therefore uses hard cluster labels. The hierarchical clustering algorithm DeepECT [29] provides multiple levels of labels, which can then be analyzed at a later stage. Other methods introduce specific AE architectures to better handle certain types of data. For example VaDE [17] uses a Variational Autoencoder (VAE) [21] to solve a probabilistic clustering objective. DEKM [12], JULE [40] and DEPICT [7] all use CNNs, which greatly increases their processing power on image data. ClusterGAN [32] applies yet another clustering strategy by employing Generative Adversarial Networks (GANs) [11].

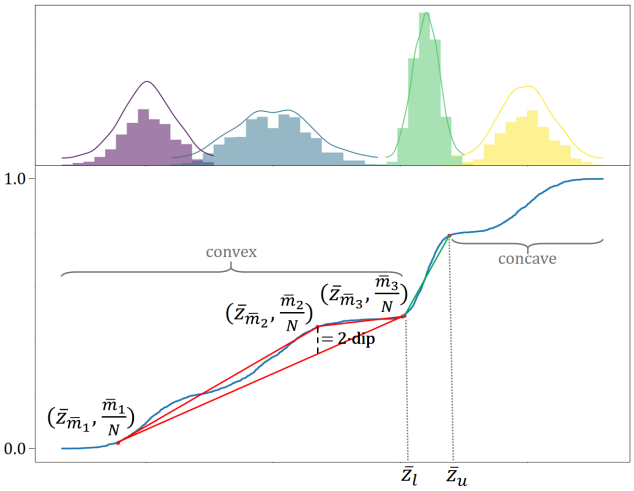


Figure 2: The figure shows the calculation of the Dip-value using an exemplary data set with four modes. [Top] Histogram of the data set. The colors indicate the underlying creation process. [Bottom] The blue line shows the ECDF of the samples. Between $\bar{z}_l$ and $\bar{z}_u$, the green line indicates the region of the maximum slope within the ECDF and thus the main mode(s) of the data set. There must be a convex distribution to the left of $\bar{z}_l$ and a concave distribution to the right of $\bar{z}_u$. The vertical height of the modal triangle (in red), formed by $(\bar{z}_{\bar{m}_1}, \frac{\bar{m}_1}{N})$, $(\bar{z}_{\bar{m}_2}, \frac{\bar{m}_2}{N})$ and $(\bar{z}_{\bar{m}_3}, \frac{\bar{m}_3}{N})$, shows the maximum deviation between the ECDF and a convex distribution and ultimately specifies the Dip-value.

We want to avoid such architecture-based extensions to show that our good results are based only on our core idea. However, extensions like convolutional layers could still be added to later evolutions of our technique.

3 THE DIPENCODER

The DipEncoder is an extension of an ordinary AE, which is able to process the gradients of the Dip-value. Therefore, we would like to discuss the mathematical foundation of those gradients first. An overview of the used symbols can be found in appendix A.

Typically, the objects within a data set are fixed, which is why so far, only the gradient with respect to the projection axis was used (e.g., in [22, 28, 33]). Since we are working within the embedding of an AE, we can also process the gradient regarding the data. To the best of our knowledge, we are the first who use the gradient of the Dip-value in a Deep Learning environment and generally the first who use the gradient with respect to the data.

For the computation of the Dip-value, we need to transform the embedded data set $Z = \text{enc}(X)$ into a one-dimensional space. Therefore, we assign distinct projection axes to each pair of clusters. These axes are stored in a separate shallow NN we call DipModule. It consists of $k(k-1)/2 \times m$ neurons, where k is the number of clusters and m the dimensionality of the embedding. We can now project the embedded samples of clusters a, denoted by $X_a \subseteq X$, and b, denoted by $X_b \subseteq X$, onto their corresponding axis $p_{a,b}$. We

define $X_{a,b} = X_a \cup X_b$ as the set of all objects assigned to either cluster a or cluster b and consequently $Z_{a,b} = \text{enc}(X_a \cup X_b)$. Then the projection is performed as $p_{a,b}^T \cdot z$, where $z \in Z_{a,b}$. Afterwards, we sort our projected embedded data to receive $\overline{Z}_{a,b} \subseteq \mathbb{R}^1$.

To calculate the Dip-value, as described in [14], we interpret $\overline{Z}_{a,b}$ as a probability distribution function and generate the corresponding empirical cumulative representation (ECDF). Next, we start to iterate over the ECDF to find the modal interval $[\overline{z}_l, \overline{z}_u], \overline{z}_l, \overline{z}_u \in \overline{Z}_{a,b}$, as well as the modal triangle $\Delta = ((\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N}), (\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N}), (\overline{z}_{\overline{m}_3}, \frac{\overline{m}_3}{N}))$, where $\overline{m}_1 \leq \overline{m}_2 \leq \overline{m}_3$ are the indices of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3} \in \overline{Z}_{a,b}$ respectively, i.e. the indices when considering the sorted projected data. The modal interval is the area within which the maximum slope of the ECDF lies. This corresponds to the most significant mode(s) of the underlying distribution. While the modal interval iteratively shrinks, the modal triangle introduced by [22] is the triangle formed by $(\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N})$, $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$ and $(\overline{z}_{\overline{m}_3}, \frac{\overline{m}_3}{N})$ that, given the modal interval, has the largest distance between the ECDF and a piecewise linear function that satisfies the conditions of unimodality (convex until $\overline{z}_l$ and concave after $\overline{z}_u$). A visualization of this process is given in Figure 2. The modal triangle fulfills $\text{height}(\Delta) = 2 \cdot \text{dip}$ (the details can be found in [22]). Therefore, the Dip-value can be calculated as follows:

$$\text{dip}(\overline{Z}_{a,b}) = \frac{1}{2N}\left(\left| \overbrace{\frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2}^{=:A} \right| + 1\right)$$

Using this formulation we can deduce the gradient regarding each feature i of the projection axis [22].

$$\frac{\partial \text{dip}(\overline{Z}_{a,b})}{\partial p_{a,b}[i]} = c\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \frac{(\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2})(z_{m_3}[i] - z_{m_1}[i])}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right), \tag{2}$$

where $c = \frac{\overline{m}_3 - \overline{m}_1}{2N}$ if $A > 0$, else $c = -(\frac{\overline{m}_3 - \overline{m}_1}{2N})$. Here, z_{m_j} is the (unprojected) m-dimensional embedded data point corresponding to $\overline{z}_{\overline{m}_j}$, i.e. $p_{a,b}^T \cdot z_{m_j} = \overline{z}_{\overline{m}_j}$, and $[i]$ indicates its i-th coordinate. Furthermore, we can calculate the gradient regarding each feature i of the samples.

$$\frac{\partial \text{dip}(\overline{Z}_{a,b})}{\partial z[i]} = \begin{cases} p_{a,b}[i]\, c \frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_1}, \\ p_{a,b}[i]\, c \frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} & \text{if } z = z_{m_2}, \\ p_{a,b}[i]\, c \frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_3}, \\ 0 & \text{else,} \end{cases} \tag{3}$$

where the definitions are the same as above. The derivations of these equations can be found in appendix D.

In order to clearly separate the clusters, we need to maximize the modality with respect to these samples. In mathematical terms this equals $\max(\text{dip}(\overline{Z}_{a,b}))$ or, since we use gradient descent, we minimize the negative value, i.e. $\min(-\text{dip}(\overline{Z}_{a,b}))$.

However, we must be careful that high modality is not achieved by tearing a cluster apart. Therefore, we need another term supporting a unimodal structure within the clusters on this projection axis. This is given by $\min(\text{dip}(\overline{Z}_{a,\not b}) + \text{dip}(\overline{Z}_{\not a,b}))$, where $\overline{Z}_{a,\not b} \subseteq \overline{Z}_{a,b}$ only contains the samples of cluster a. $\overline{Z}_{\not a,b}$ is defined analogously. We consider these two optimizations to be equally important, which is why we multiply the unimodal constraint by $\frac{1}{2}$.

Applying these terms to the batch-wise optimization of the DipEncoder yields the loss terms $\mathcal{L}_{uni}$ and $\mathcal{L}_{multi}$.

$$\mathcal{L}_{uni}(\mathcal{B}, a, b) = \frac{1}{2}\left(\text{dip}(\overline{Z}^{\mathcal{B}}_{a,\not b}) + \text{dip}(\overline{Z}^{\mathcal{B}}_{\not a,b})\right),$$
$$\mathcal{L}_{multi}(\mathcal{B}, a, b) = -\text{dip}(\overline{Z}^{\mathcal{B}}_{a,b}),$$

where $\overline{Z}^{\mathcal{B}}_{a,b} = \text{sort}\{p_{a,b}^T \cdot z | z \in \text{enc}(X_{a,b} \cap \mathcal{B})\}$. $\overline{Z}^{\mathcal{B}}_{a,\not b}$ and $\overline{Z}^{\mathcal{B}}_{\not a,b}$ are defined analogously. Combined we get the dip loss $\mathcal{L}_{dip}$.

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

With the previously defined gradients (Eq. 2 and 3), we can now optimize the DipEncoder by backpropagation based on these Dip-values. Here the gradient from the projection axes feeds into the DipModule, and the gradient with respect to the data is used to update the neurons of the AE.

One problem with the current loss function is that it does not generalize very well since the embedding is optimized based only on the modal triangle. That is, we only get a non-zero gradient for three samples per execution of the Dip-test. For this reason, we also include the reconstruction loss (Eq. 1) into our final loss term.

$$\mathcal{L}_{final}(\mathcal{B}) = \mathcal{L}_{dip}(\mathcal{B}) + \lambda \mathcal{L}_{rec}(\mathcal{B}) \tag{4}$$

We would like the reconstruction loss to be weighted similarly to $\mathcal{L}_{uni}$ and $\mathcal{L}_{multi}$. Due to the nature of the Dip-value these are limited to $(0, 0.25]$ and $[-0.25, 0)$ respectively. To constrain $\mathcal{L}_{rec}$, we need to define a hypothetical maximum value. For this we assume that the AE initially, i.e. before the backpropagation is executed for the first time, is in its worst state. We therefore set $\lambda = \frac{1}{4\mathcal{L}_{rec}(\mathcal{B}_{init})}$, where $\mathcal{B}_{init}$ is the first batch of data. This makes $0 \leq \lambda \mathcal{L}_{rec}(\mathcal{B}) \leq 0.25$ approximately valid for all $\mathcal{B}$.

To speed up the separation of the individual clusters, we multiply the gradients originating from the Dip-tests within $\mathcal{L}_{uni}$ by dip, where dip equals the corresponding Dip-value, and the gradient originating from the Dip-test within $\mathcal{L}_{multi}$ by $(0.25 - dip)$. This strategy reduces, for example, the weighting of $\mathcal{L}_{uni}$ concerning clusters that already indicate a unimodal structure. Thus, we shift the focus of the optimization towards structures that do not yet show the desired characteristics.

Note that the Dip-test only gives meaningful values if a certain amount of samples is present. Since our samples are divided into several clusters, we need larger batches with more clusters present. Based on an experimental analysis (appendix B), we recommend a minimum batch size of $25 \cdot k$, where k is the number of clusters.

As already shown in [25], the Dip-test struggles to notice a mode as such if it is significantly smaller than another one. Thus, a cluster containing many points together with a cluster containing significantly fewer points is still considered unimodal. This, in particular, weakens the significance of the modal triangle in $\mathcal{L}_{multi}$ and thus the usability of the gradient. To avoid this, [25] suggests to use only the $S|X_b|$ samples from cluster a that are closest to cluster b if $|X_a| > S|X_b|$. Since we want to preserve the general structure of

both clusters as much as possible, we utilize a variation of this idea by randomly sampling $3|\overline{Z}^{\mathcal{B}}_{\not\phi,b}|$ objects from $\overline{Z}^{\mathcal{B}}_{a,\not\phi}$ if $|\overline{Z}^{\mathcal{B}}_{a,\not\phi}| > 3|\overline{Z}^{\mathcal{B}}_{\not\phi,b}|$.

Figure 3 shows an execution of the DipEncoder (3(a) - 3(d)) on the Optdigits data set using the ground truth labels. We see that the clusters within the embedding of the DipEncoder adopt a unimodal structure that is clearly separated from other clusters. Compared to a regular AE (3(e) - 3(h)), the structures stand out much better.

One advantage of the DipEncoder is that we do not have to assume a distribution in advance. We only want to achieve a unimodal structure within a cluster and a multimodal structure between clusters. This is a perfect starting point for unsupervised learning algorithms since the inter- and intra-cluster dependencies are often unknown a priori.

3.1 Update the cluster labels

We use the components of the DipEncoder to develop a parameter-free method to update the cluster labels. For this, we utilize the property that the Dip-test does not only return the Dip-value and the modal triangle but also the modal interval $[\overline{z}_l, \overline{z}_u]$. This interval shows the area of the steepest slope in the ECDF and thus describes the most significant mode(s) within our samples. We use the intervals returned by $\mathcal{L}_{uni}$ to determine the areas of influence of the two clusters on their corresponding projection axis. This information can then be applied to define a threshold that indicates whether an object should be assigned to the left or the right cluster. The threshold T is simply the center point between the upper limit $\overline{z}_{u,L}$ of the left cluster L and the lower limit $\overline{z}_{l,R}$ of the right cluster R.

$$T = (\overline{z}_{u,L} + \overline{z}_{l,R})/2 \tag{5}$$

An example of this process can be seen in Figure 4. Initially, some objects to the right of T still belong to the yellow cluster. However, our procedure indicates that these should rather match the purple cluster. The same applies analogously to objects from the purple cluster that lie to the left of T.

If we want to update the cluster labels, we project each sample onto each projection axis $p_{a,b}$ and determine whether it belongs rather to the left or the right cluster. In the end, each object is assigned to the cluster with which it has most frequently matched. Usually, one cluster always matches, making the assignment unambiguous. If a tie does occur, the sample is assigned to the cluster with the lower ID.

The described method has one major shortcoming. Since the modal interval specifies the area of the steepest slope in the ECDF while satisfying the unimodality constraints (first convex, then concave), this range turns out to be very small in the case of an already unimodal distribution. This behavior is reasonable by definition, but it does not fully reflect our human understanding of the most significant mode. Therefore, we apply a strategy introduced in the implementation of [28]. By mirroring the data set, we can assume quite reliably that we obtain a multimodal structure. It follows that the resulting modal interval has a higher significance with respect to our application. Since we do not change the structure of the samples, we can transfer the mirrored interval to the non-mirrored case. A visual representation of this strategy is presented in Figure 5. We can see that the modal interval of the mirrored samples captures our natural perception of the mode much better.

Algorithm 1: Pseudocode of the DipEncoder

Input: data set X, number of clusters k, number of epochs E
Output: *labels*

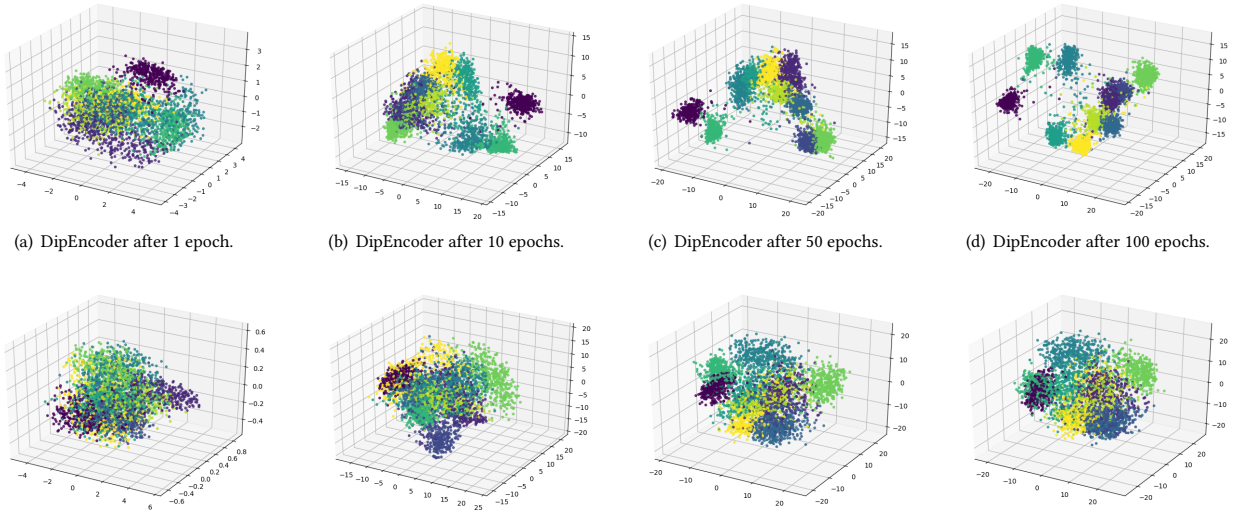
```
// Pretrain AE; save the reconstruction loss of B_init as λ
(AE, λ) = pretrain autoencoder on X using L_rec (Eq. 1)
// Get initial labels and projection axes
labels = k-means(AE.encode(X), k)
DM = DipModule(X, AE, labels) (Eq. 6)
for epoch = 0; epoch ≤ E; epoch += 1 do
    // Update labels as described in Section 3.1
    for x ∈ AE.encode(X) do
        clusterMatches = [0, ..., 0] ∈ ℝ^k
        for a = 1; a ≤ k − 1; a += 1 do
            for b = a + 1; b ≤ k; b += 1 do
                p_{a,b} = DM.getProjectionAxis(a, b)
                Z̄_{a,∄} = sort{p^T_{a,b} · z | z ∈ AE.encode(X_a)}
                Z̄_{∄,b} = sort{p^T_{a,b} · z | z ∈ AE.encode(X_b)}
                [z̄_{l,a}, z̄_{u,a}], [z̄_{l,b}, z̄_{u,b}] = dip(Z̄_{a,∄}), dip(Z̄_{∄,b})
                c_L, c_R = ids of left and right cluster on p_{a,b}
                T = center between the clusters (Eq. 5)
                if (p^T_{a,b} · x) < T then
                    clusterMatches[c_L] += 1
                else
                    clusterMatches[c_R] += 1
        set label of x to argmax(clusterMatches)
    if epoch == E then
        break
    // Train the DipEncoder
    for B in X do
        L_final = L_dip(B) + λ L_rec(B) (Eq. 4)
        optimize AE and DM using L_final
return labels
```

3.2 Clustering algorithm

Next, we utilize the concepts described above to develop a novel clustering algorithm. A pseudocode version of our approach is given in Algorithm 1.

Since the DipEncoder calculates the Dip-values based on known labels, initial clusters are required first. For this, we pretrain an AE using the reconstruction loss and then run an arbitrary clustering algorithm in the resulting embedding. For simplicity, we use k-means for this. We want to emphasize that other algorithms are easily applicable since we do not make any assumptions about distributions and only analyze modalities. The requirement of an initial clustering is often found in DC (for example in [7, 13, 17, 25, 38, 39]). Therefore, these methods can also be seen as AE-based cluster refinement methods.

Once we have initial cluster labels, we can create the DipModule. To obtain initial projection axes $p_{a,b}$, we calculate the distances between each combination of cluster centers within the embedding



(a) DipEncoder after 1 epoch. (b) DipEncoder after 10 epochs. (c) DipEncoder after 50 epochs. (d) DipEncoder after 100 epochs.

(e) AE after 1 epoch. (f) AE after 10 epochs. (g) AE after 50 epochs. (h) AE after 100 epochs.

Figure 3: The images show the embeddings ($m = 10$) of the DipEncoder (3(a) - 3(d)) and a regular AE (3(e) - 3(h)) for the Optdigits data set after 1, 10, 50 and 100 epochs. The colors indicate the clusters of the samples. To create a three-dimensional plot, we use the first three components of a PCA.

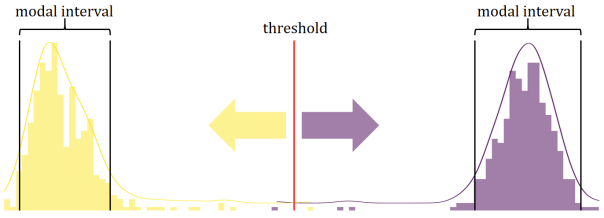


Figure 4: Illustration of our label update method using two clusters projected onto their corresponding projection axis. A threshold T is determined by calculating the center point between $\bar{z}_u$ of the yellow cluster (left) and $\bar{z}_l$ of the purple cluster (right). For each object, the relative position with respect to T is used to check which cluster fits better.

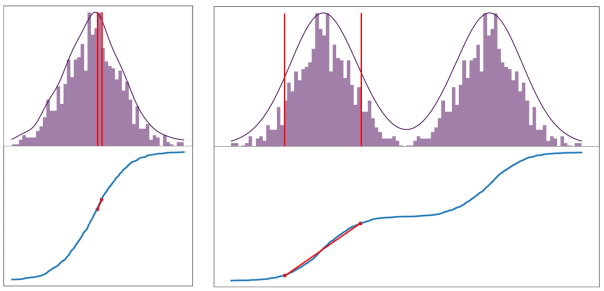
of the pretrained AE.

$$p_{a,b} = \frac{\sum_{x \in X_a} \text{enc}(x)}{|X_a|} - \frac{\sum_{x \in X_b} \text{enc}(x)}{|X_b|} \tag{6}$$

Now we start the iterative optimization of our clustering. First, for each cluster combination we compute the Dip-value with respect to $p_{a,b}$ and update the labels as described in Section 3.1. Next, we use the updated labels to calculate the loss function $\mathcal{L}_{final}$ (Eq. 4) in a batch-wise fashion and thus optimize the AE and DipModule. This procedure repeats for a predefined number of iterations.

4 EXPERIMENTS

To verify the quality of our approach, we conduct experiments on a variety of real-world data sets. Therefore, we compare our results to those of competitor algorithms.



(a) Non-mirrored samples. (b) Mirrored samples.

Figure 5: The images show the modal interval of an example distribution. [Top] Histogram of the data set. [Bottom] The corresponding ECDF. The modal interval is represented by a red line. 5(a) shows the original samples, while 5(b) shows the samples mirrored to the left.

Data sets: First, we would like to briefly describe the used data sets. The general information (N, d, k) can be found in the header of Table 1. To present the wide application field of our method, we use data sets from different domains. Optdigits [8], USPS [16] and MNIST [24] are image data sets containing the digits 0-9. The image data set Fashion-MNIST (F-MNIST) [37] shows items from the *zalando* online store and Kuzushiji-MNIST (K-MNIST) [3] consists of various Kanji characters. Human Activity Recognition (HAR) [8], Pendigits [8] and Letterrecognition (Letters) [8] are numerical data sets representing sensor records of human activity, coordinates of handwritten digits and letter stimuli, respectively. Furthermore, we

Table 1: NMI results of the DipEncoder against the comparison algorithms on various data sets. Each experiment is repeated ten times, and the average result ± standard deviation as well as the maximum result (in brackets) are stated in %. The best average and maximum result per data set are underlined, the runner-up is dotted.

Method	**Optdigits** ($k = 10$) ($N = 5620, d = 64$)	**USPS** ($k = 10$) ($N = 9298, d = 256$)	**HAR** ($k = 6$) ($N = 10299, d = 561$)	**Pendigits** ($k = 10$) ($N = 10992, d = 16$)	**Reuters10k** ($k = 4$) ($N = 10000, d = 2000$)
DipEncoder	88.6 ± 3.0 (92.0)	81.9 ± 0.8 (83.6)	73.5 ± 6.9 (82.4)	75.2 ± 2.1 (78.2)	36.8 ± 4.2 (41.9)
AE+k-means	80.1 ± 2.4 (83.8)	69.6 ± 0.9 (71.2)	67.5 ± 4.2 (73.2)	70.0 ± 0.8 (72.2)	37.2 ± 6.3 (47.3)
DEC	88.5 ± 2.5 (91.9)	80.7 ± 0.6 (81.4)	66.3 ± 4.8 (76.8)	76.9 ± 1.1 (77.9)	37.9 ± 7.1 (51.6)
IDEC	80.4 ± 2.4 (84.0)	69.3 ± 1.0 (70.9)	69.9 ± 2.9 (74.1)	69.8 ± 1.3 (72.2)	39.1 ± 6.7 (51.9)
DCN	84.8 ± 2.3 (84.8)	74.6 ± 1.3 (76.1)	73.4 ± 4.8 (80.9)	73.5 ± 0.5 (74.4)	35.1 ± 7.1 (45.4)
DipDECK	83.5 ± 2.3 (86.7)	68.5 ± 1.3 (70.5)	70.8 ± 1.3 (72.1)	72.8 ± 1.2 (74.7)	15.6 ± 18.1 (45.8)

Method	**20Newsgroups** ($k = 20$) ($N = 18846, d = 2000$)	**Letters** ($k = 26$) ($N = 20000, d = 16$)	**MNIST** ($k = 10$) ($N = 70000, d = 784$)	**F-MNIST** ($k = 10$) ($N = 70000, d = 784$)	**K-MNIST** ($k = 10$) ($N = 70000, d = 784$)
DipEncoder	30.8 ± 0.6 (31.6)	47.1 ± 0.9 (48.2)	85.8 ± 1.6 (87.8)	60.6 ± 2.2 (63.5)	52.2 ± 3.2 (57.0)
AE+k-means	31.2 ± 0.8 (32.4)	42.3 ± 0.9 (43.4)	74.4 ± 1.5 (77.0)	54.2 ± 0.5 (55.1)	46.3 ± 2.4 (49.7)
DEC	15.8 ± 1.0 (17.1)	46.0 ± 2.0 (48.0)	85.2 ± 1.2 (86.6)	59.7 ± 1.4 (62.5)	54.2 ± 2.1 (58.0)
IDEC	28.1 ± 1.3 (30.2)	45.1 ± 1.4 (47.6)	75.3 ± 1.2 (77.8)	54.3 ± 0.7 (55.3)	46.7 ± 1.8 (49.1)
DCN	28.6 ± 1.5 (30.7)	43.7 ± 2.4 (48.4)	82.0 ± 1.7 (84.2)	56.0 ± 0.9 (58.3)	48.2 ± 2.0 (51.7)
DipDECK	00.1 ± 0.0 (00.1)	34.5 ± 3.6 (38.2)	75.8 ± 2.0 (79.4)	53.9 ± 2.9 (57.2)	38.7 ± 4.1 (43.0)

consider the document data sets Reuters [26] and 20Newsgroups[1]. For 20Newsgroups we convert the documents to vectors with TF-IDF, using only the most common 2000 features. We preprocess Reuters as described in [38], using a subset of 10000 objects, called Reuters10k.

Apart from TF-IDF on the document data sets, we do not use other preprocessing steps. Since all DC algorithms are based on an AE, they should be able to learn relevant transformations on their own.

AE setup: We choose the same structure of the AE as presented in [38]. This corresponds to the AE dimensions $d - 500 - 500 - 2000 - m - 2000 - 500 - 500 - d$, where $m = 10$, which have already been used by other DC algorithms, like [13] or [17]. We would like to have a starting situation as similar as possible in our experiments. Therefore, ten AEs are pretrained for each data set, forming the initial setting for all DC algorithms. These AEs are trained for 100 epochs using the ADAM [20] optimizer with a constant learning rate of 0.001 and a batch size of 256. The actual clustering optimization then follows.

During clustering, the DipEncoder uses a learning rate of 0.0001 with a batch size of $25 \cdot k$ (as described in Section 3). The other DC algorithms use a learning rate of 0.0001 with a batch size of 256. All optimizations were run for another 100 epochs.

The DipEncoder was implemented using PyTorch[2] and can be downloaded at: https://dmm.dbs.ifi.lmu.de/downloads.

Comparison algorithms: As comparison algorithms, we choose those comparable by architecture since only these can load the pretrained AEs naturally. These algorithms are DEC, IDEC, DCN and DipDECK. In addition, we execute k-means on the pretrained AEs (AE+k-means). All parameters are set as described in the respective papers. The only exception is DipDECK, where we set $m = 10$ instead of 5. By comparing to the AE+k-means results, one can see how much the results have improved since all the procedures start from the same initial situation. An exception is DipDECk, since there the initial k-means execution is run with a highly overestimated number of clusters ($k_{init} = 35$).

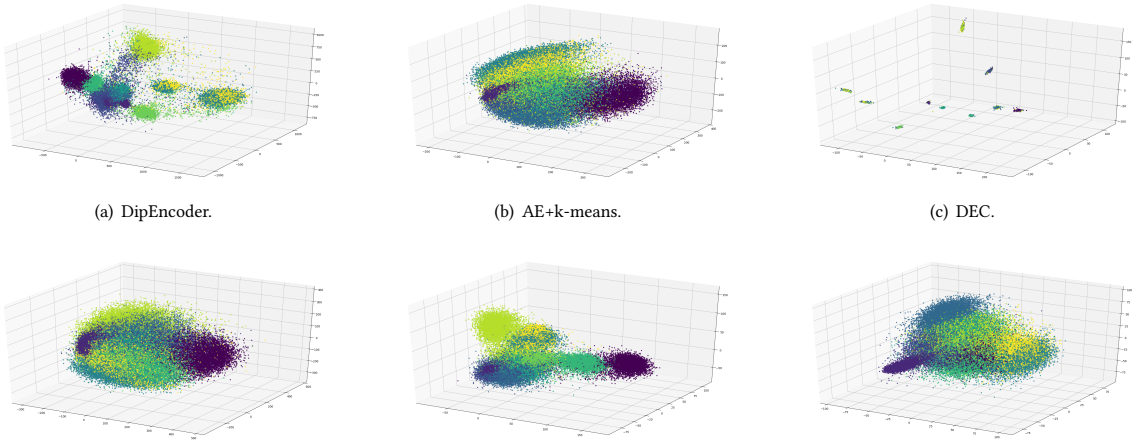
Metrics: We evaluate our approaches using the *normalized mutual information* (NMI) and the *adjusted rand index* (ARI). Both are established metrics in clustering that can take values between 0 and 1, where 1 indicates a perfect result. We report all results in %.

Evaluation: The NMI results of the experiments can be found in Table 1 (ARI results in appendix C). We can see that our method performs best in 12 out of 20 experiments (mean and max). In addition, we take second place six times. We generally show good results on both image and numerical data sets. Only concerning text data do the comparison algorithms perform slightly better. In this case, the other Dip-based clustering method, DipDECK, also seems to have significant problems. For example, it identifies only 1 or 2 clusters for 20Newsgroups, which explains the inferior results. From this, it can be concluded that text data sets, possibly due to the sparse structure, show less multimodal features.

Figure 6 illustrates the results of the compared DC algorithms on MNIST. It is easy to see that the procedures operate with different degrees of rigor as far as the shifting of data is concerned. DEC offers the greatest flexibility since it does not apply any regularization term such as the reconstruction loss. Therefore, one can clearly notice the compressed clusters as identified by DEC. The DipEncoder seems to be the second most flexible algorithm in this regard. AE+k-means, on the other hand, is by definition not able to separate clusters from each other. IDEC and DipDECK limit the ability to push clusters apart by integrating the reconstruction loss into the cost function using a fixed weighting factor. This factor equals 10 compared to the cluster loss for IDEC and 1 for DipDECK.

We want to elaborate on these differences a bit more. When we compare our results from Table 1 with the results from other publications, some comparison methods seem to have problems

[1] http://qwone.com/~jason/20Newsgroups/
[2] https://pytorch.org/



(a) DipEncoder. (b) AE+k-means. (c) DEC.

(d) IDEC. (e) DCN. (f) DipDECK.

Figure 6: Images of the final embeddings ($m = 10$) of MNIST as created by different Deep Clustering algorithms. The colors depict the ground truth labels of the clusters. Features correspond to the first three components of a PCA.

Table 2: DC results on a standardized version of MNIST (zero mean and unit variance). Each experiment is again repeated ten times. Representation corresponds to Table 1.

Method	**MNIST-Standardized** ($k = 10$) ($N = 70000, d = 784$)	
	NMI	ARI
DipEncoder	85.3 ± 2.3 (88.2)	78.7 ± 4.8 (85.0)
AE+k-means	70.9 ± 2.7 (74.8)	63.4 ± 4.6 (70.1)
DEC	82.1 ± 2.6 (85.5)	75.7 ± 5.4 (82.4)
IDEC	85.2 ± 2.0 (87.4)	78.6 ± 4.4 (84.2)
DCN	81.8 ± 2.4 (84.8)	73.9 ± 5.0 (81.2)
DipDECK	81.6 ± 2.3 (86.0)	67.8 ± 6.2 (80.7)

if the data sets have not been standardized (zero mean and unit variance) as a preprocessing step. This concerns, in particular, those methods that use a constant factor for weighting the reconstruction loss without considering the scaling of the original space. To illustrate this, we conduct another experiment where we rerun all algorithms on a standardized version of MNIST. The experiments were performed as described above. The results are displayed in Table 2. We can see that while the results from the DipEncoder are almost unchanged, DipDECK and, in particular, IDEC perform significantly better. This shows the benefit of weighting the reconstruction loss depending on the scaling of the data set to achieve a value similar to the clustering loss.

An additional experiment is performed to verify that our method is able to generalize. Therefore, we apply the DipEncoder in a supervised manner. For this, we use the provided test-training partitions of MNIST and use the known training labels to optimize the DipEncoder. Here, we can skip the pretraining of the AE and the initial k-means execution. In this supervised version of the DipEncoder, the update of the labels is executed only for the test data, after the optimization of the training data is finished after 100 epochs. We compare our results with those of a support vector machine (SVM) [4] in combination with different dimension reduction techniques. SVM is a machine learning algorithm that tries to place planes in the feature space to separate the classes as well as possible. This procedure compares well with our approach as described in Section 3.1. We run SVM in combination with PCA, as the probably most used dimensionality reduction technique, LDA, as a frequently used supervised method, and an AE, as it is the basis of our approach. All methods reduce the number of features to 10 (using the implementations from scikit-learn[3]). Furthermore, we also perform SVM in the embedding created by the DipEncoder. This gives us conclusions about the quality of our label update method. To evaluate these supervised experiments, we use the test data not used for the prior optimization. We use the accuracy (ACC) as a common metric in supervised learning to quantify the results. It ranges from 0 (poor) to 1 (perfect). Additionally, the NMI values are given, so we are able to compare with the previous clustering results. The supervised results are shown in Table 3.

The analysis indicates that the DipEncoder generalizes very well, assigning even previously unknown samples into the correct groups in most cases. Since the supervised DipEncoder also produces better results than SVM in combination with the embedding of the DipEncoder, we can assume that our procedure to update the cluster labels defines reasonable bounds for the clusters.

5 CONCLUSION

In this paper, we have shown for the first time how to calculate the gradient of the Dip-value with respect to the data. To demonstrate how this can practically be used, we developed the DipEncoder. It is an extension of an autoencoder that uses the Dip-test to separate samples from different clusters in an embedding. The only condition is that samples of one cluster show a unimodal structure while the

[3]https://scikit-learn.org/stable/index.html

Table 3: Results of a supervised version of the DipEncoder against SVM in combination with different dimensionality reduction techniques. Each experiment is again repeated ten times. Representation corresponds to Table 1.

Method	MNIST ($k = 10$) ($\frac{N_{\text{train}} = 60000}{N_{\text{test}} = 10000}$, $d = 784$)	
	ACC	NMI
DipEncoder$_{supervised}$	94.2 ± 3.9 (97.2)	90.5 ± 2.3 (92.7)
DipEncoder+SVM	92.9 ± 4.3 (97.1)	88.1 ± 3.1 (92.5)
SVM	86.6 ± 1.1 (87.5)	74.5 ± 1.1 (75.5)
PCA+SVM	58.5 ± 4.9 (67.7)	45.9 ± 3.4 (53.9)
LDA+SVM	87.7 ± 0.0 (87.7)	74.7 ± 0.0 (74.7)
AE+SVM	89.1 ± 1.7 (91.2)	79.4 ± 2.0 (81.8)

combined samples of two clusters show a multimodal structure. A previously defined distribution or other cluster properties are not necessary.

Further, we developed a Dip-based method that can update the labels within each iteration of the DipEncoder. The resulting deep clustering algorithm shows state-of-the-art results on various real-world data sets.

We think that the gradient of the Dip-value with respect to the data allows for a series of new research opportunities. In particular, we expect further research to integrate established Dip-based clustering methods to determine the number of clusters automatically.

ACKNOWLEDGMENTS

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

The present contribution is supported by the Helmholtz Association under the joint research school 'Munich School for Data Science - MUDS'.

REFERENCES

[1] Dana H Ballard. 1987. Modular learning in neural networks.. In *AAAI*, Vol. 647. 279–284.
[2] Theofilos Chamalis and Aristidis Likas. 2018. The Projected Dip-means Clustering Algorithm. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN*. ACM, 14:1–14:7.
[3] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. 2018. Deep Learning for Classical Japanese Literature. *CoRR* (2018).
[4] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (1995), 273–297.
[5] Ying Cui, Xiaoli Z. Fern, and Jennifer G. Dy. 2007. Non-redundant Multi-view Clustering via Orthogonalization. In *ICDM*. IEEE Computer Society, 133–142.
[6] Chris H. Q. Ding and Tao Li. 2007. Adaptive dimension reduction using discriminant analysis and K-means clustering. In *ICML (ACM International Conference Proceeding Series, Vol. 227)*. ACM, 521–528.
[7] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *ICCV*. IEEE Computer Society, 5747–5756.
[8] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[9] R. A. FISHER. 1938. THE STATISTICAL UTILIZATION OF MULTIPLE MEASUREMENTS. *Annals of Eugenics* 8, 4 (1938), 376–386.
[10] Karl Pearson F.R.S. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.
[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. 2672–2680.
[12] Wengang Guo, Kaiyan Lin, and Wei Ye. 2021. Deep Embedded K-Means Clustering. In *ICDM*. IEEE, 686–694.
[13] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved Deep Embedded Clustering with Local Structure Preservation. In *IJCAI*. ijcai.org, 1753–1759.
[14] John A Hartigan and Pamela M Hartigan. 1985. The dip test of unimodality. *The annals of Statistics* (1985), 70–84.
[15] PM Hartigan. 1985. Computation of the dip statistic to test for unimodality: Algorithm as 217. *Applied Statistics* 34, 3 (1985), 320–5.
[16] Jonathan J. Hull. 1994. A Database for Handwritten Text Recognition Research. *IEEE Trans. Pattern Anal. Mach. Intell.* 16, 5 (1994), 550–554.
[17] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *IJCAI*. ijcai.org, 1965–1972.
[18] Christian Jutten and Jeanny Hérault. 1991. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Process.* 24, 1 (1991), 1–10.
[19] Argyris Kalogeratos and Aristidis Likas. 2012. Dip-means: an incremental clustering method for estimating the number of clusters. In *NIPS*. 2402–2410.
[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[21] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.
[22] Andreas Krause and Volkmar Liebscher. 2005. Multimodal projection pursuit using the dip statistic. (2005).
[23] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[25] Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. 2021. Dip-based Deep Embedded Clustering with k-Estimation. In *ACM SIGKDD*. ACM, 903–913.
[26] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.* 5 (2004), 361–397.
[27] Yonghong Luo, Ying Zhang, Xiaoke Ding, Xiangrui Cai, Chunyao Song, and Xiaojie Yuan. 2018. StrDip: A Fast Data Stream Clustering Algorithm Using the Dip Test of Unimodality. In *WISE (Lecture Notes in Computer Science, Vol. 11234)*. Springer, 193–208.
[28] Samuel Maurus and Claudia Plant. 2016. Skinny-dip: Clustering in a Sea of Noise. In *ACM SIGKDD*. ACM, 1055–1064.
[29] Dominik Mautz, Claudia Plant, and Christian Böhm. 2019. Deep Embedded Cluster Tree. In *ICDM*. IEEE, 1258–1263.
[30] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. 2020. Non-Redundant Subspace Clusterings with Nr-Kmeans and Nr-DipMeans. *ACM Trans. Knowl. Discov. Data* 14, 5 (2020), 55:1–55:24.
[31] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.
[32] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. 2019. ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. In *AAAI*. AAAI Press, 4610–4617.
[33] Benjamin Schelling, Lena Greta Marie Bauer, Sahar Behzadi, and Claudia Plant. 2020. Utilizing Structure-Rich Features to Improve Clustering. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 12457)*. Springer, 91–107.
[34] Benjamin Schelling and Claudia Plant. 2018. DipTransformation: Enhancing the Structure of a Dataset and Thereby Improving Clustering. In *ICDM*. IEEE Computer Society, 407–416.
[35] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.
[36] Herman Wold. 1966. Estimation of principal components and related models by iterative least squares. *Multivariate analysis* (1966), 391–420.
[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* (2017).
[38] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In *ICML (JMLR Workshop and Conference Proceedings, Vol. 48)*. JMLR.org, 478–487.
[39] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 3861–3870.
[40] Jianwei Yang, Devi Parikh, and Dhruv Batra. 2016. Joint Unsupervised Learning of Deep Representations and Image Clusters. In *CVPR*. IEEE Computer Society, 5147–5156.

APPENDIX

In the appendix, we give an overview of the used symbols, add ARI results for our quantitative experiments, justify our chosen batch size, and present the derivations of the Dip-value and its gradients.

A SYMBOLS

All symbols used in the paper are explained in Table 4.

Table 4: Overview of the used symbols.

Symbol	Description
$N \in \mathbb{N}$	Size of the data set
$d \in \mathbb{N}$	Dimensionality of the original feature space
$m \in \mathbb{N}$	Dimensionality of the embedded space
$k \in \mathbb{N}$	Number of clusters
$T \in \mathbb{R}$	Threshold of the label update method
$X \subseteq \mathbb{R}^d$	Set of all objects
$\mathcal{B} \subseteq X$	One batch of data
$X_a \subseteq X$	Objects assigned to cluster a
$Z \subseteq \mathbb{R}^m$	Embedded set of all objects
	$\Rightarrow Z = \text{enc}(X)$
$Z_{a,b} \subseteq \mathbb{R}^m$	Embedded objects assigned to cluster a or b
	$\Rightarrow Z_{a,b} = \text{enc}(X_a \cup X_b)$
$p_{a,b} \in \mathbb{R}^m$	Projection axis of cluster a and b
$\overline{Z}_{a,b} \subseteq \mathbb{R}^1$	Sorted version of $Z_{a,b}$ projected to $p_{a,b}$
	$\Rightarrow \overline{Z}_{a,b} = \text{sort}\{p_{a,b}^T \cdot z \mid z \in Z_{a,b}\}$
$\overline{Z}_{a,b}^{\not b} \subseteq \overline{Z}_{a,b}$	Subset of $\overline{Z}_{a,b}$, ignoring samples from cluster b
$dip \in (0, 0.25]$	Dip-value (result of the Dip-test)
Δ	The modal triangle
$\overline{z}_l, \overline{z}_u \in \overline{Z}_{a,b}$	Lower and upper bound of the modal interval
$\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3} \in \overline{Z}_{a,b}$	x-coordinates of the modal triangle in the ECDF
$\overline{m}_1, \overline{m}_2, \overline{m}_3 \in \mathbb{N}$	Indices of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3}$
$z_{m_1}, z_{m_2}, z_{m_3} \in Z_{a,b}$	Non-projected representations of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3}$
	$\Rightarrow \overline{z}_{\overline{m}_j} = p_{a,b}^T \cdot z_{m_j}$

B INFLUENCE OF THE BATCH SIZE

We would like to demonstrate by an experiment why an increasing batch size is necessary for the DipEncoder when the number of clusters increases. For this purpose, we execute the DipEncoder with batch sizes between $5 \cdot k$ and $50 \cdot k$ on different data sets. One should keep in mind that the number of clusters in the data sets ranges from 4 (Reuters10k) to 26 (Letters). The results can be seen in Figure 7.

Here we can observe that most records reach a plateau at around $15 \cdot k$. Above $25 \cdot k$, almost no improvements take place, which is why we select this factor as the default value.

C ARI RESULTS

In addition to the NMI results as shown in Table 1, we evaluate the experiments using the *adjusted rand index* (ARI). Those values are shown in Table 5.

D DERIVATIONS

To consolidate the understanding of our methods, we would like to present some derivations. First, we will deal with the calculation of the Dip-value before we take a closer look at the gradients with respect to the projection axis and the modal triangle. All symbols are defined as described above.

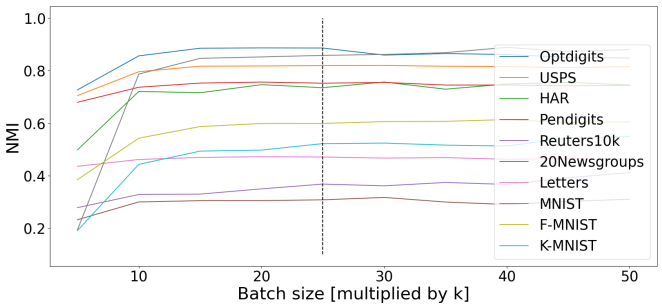


Figure 7: Results of the DipEncoder using different batch sizes on multiple data sets. Each entry corresponds to the average of 10 executions. The black vertical line illustrates the selected default value for the DipEncoder.

D.1 Derivation of the Dip-value

It has already been described in Section 3 that the Dip-value is determined based on the height of the modal triangle Δ on the ECDF. Following holds:

$$\text{height}(\Delta) = 2 \cdot \text{dip}(\overline{Z})$$

To solve this equation we determine the intersection between the vertical line towards $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$ and the line between $(\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N})$ and $(\overline{z}_{\overline{m}_3}, \frac{\overline{m}3}{N})$. Note that the intersection can be above or below $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$.

$$\begin{pmatrix} \overline{z}_{\overline{m}_2} \\ \frac{\overline{m}_2}{N} \end{pmatrix} \pm \alpha \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \overline{z}_{\overline{m}_1} \\ \frac{\overline{m}_1}{N} \end{pmatrix} + \beta \begin{pmatrix} \overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1} \\ \frac{\overline{m}_3 - \overline{m}_1}{N} \end{pmatrix},$$

where $\alpha = \text{height}(\Delta)$.

$$\Rightarrow \beta = \frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}$$

$$\Rightarrow \alpha = \pm \frac{1}{N}(\overline{m}_1 - \overline{m}_2 + \frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}})$$

This yields:

$$\text{dip'}(\overline{Z}) = \frac{1}{2N}\left|\frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2\right|$$

Krause et al. [22] add another $\frac{1}{2N}$ to this formula. This value was also used in the implementation outline from [15]. We assume a regularization purpose in the case of a degenerate triangle resulting in a Dip-value of 0, which is out of the bounds for the Dip-value. Note that this additional value has only a small influence on the final Dip-value and, as a constant, no impact on the gradients.

$$\text{dip}(\overline{Z}) = \frac{1}{2N}\left(\left|\overbrace{\frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2}^{=:A}\right| + 1\right)$$

Table 5: ARI results of the DipEncoder against the comparison algorithms on various data sets. Each experiment is repeated ten times. Representation corresponds to Table 1 (showing the NMI results).

Method	**Optdigits** ($k = 10$) ($N = 5620, d = 64$)	**USPS** ($k = 10$) ($N = 9298, d = 256$)	**HAR** ($k = 6$) ($N = 10299, d = 561$)	**Pendigits** ($k = 10$) ($N = 10992, d = 16$)	**Reuters10k** ($k = 4$) ($N = 10000, d = 2000$)
DipEncoder	85.6 ± 5.8 (91.4)	74.2 ± 1.1 (76.2)	63.4 ± 7.4 (73.5)	64.7 ± 4.2 (70.2)	35.4 ± 4.5 (43.4)
AE+k-means	76.7 ± 4.5 (83.0)	59.7 ± 1.4 (61.6)	58.7 ± 5.4 (65.1)	60.5 ± 1.9 (63.9)	39.9 ± 10.7 (56.7)
DEC	84.9 ± 5.2 (91.1)	72.7 ± 0.9 (74.1)	53.4 ± 7.7 (71.8)	66.7 ± 2.6 (68.6)	35.3 ± 10.6 (57.6)
IDEC	77.1 ± 4.6 (83.1)	59.3 ± 1.3 (60.9)	58.1 ± 2.8 (62.5)	60.2 ± 2.9 (63.9)	36.4 ± 10.1 (56.4)
DCN	81.1 ± 5.0 (86.2)	64.4 ± 2.3 (67.0)	62.7 ± 5.8 (71.9)	62.6 ± 2.0 (64.7)	29.6 ± 10.1 (49.8)
DipDECK	79.6 ± 5.5 (85.7)	58.7 ± 2.8 (63.1)	49.9 ± 1.1 (50.9)	61.7 ± 2.6 (65.4)	12.1 ± 14.3 (36.9)

Method	**20Newsgroups** ($k = 20$) ($N = 18846, d = 2000$)	**Letters** ($k = 26$) ($N = 20000, d = 16$)	**MNIST** ($k = 10$) ($N = 70000, d = 784$)	**F-MNIST** ($k = 10$) ($N = 70000, d = 784$)	**K-MNIST** ($k = 10$) ($N = 70000, d = 784$)
DipEncoder	18.0 ± 0.9 (19.2)	22.8 ± 0.9 (24.0)	81.0 ± 3.0 (84.5)	44.8 ± 2.8 (47.4)	37.7 ± 3.7 (43.9)
AE+k-means	16.9 ± 0.7 (17.9)	18.8 ± 0.6 (19.9)	69.1 ± 2.3 (73.2)	38.3 ± 1.1 (39.9)	32.4 ± 3.1 (37.9)
DEC	5.4 ± 0.6 (6.1)	20.5 ± 2.3 (23.4)	81.6 ± 2.1 (83.5)	44.0 ± 2.8 (48.9)	39.0 ± 2.3 (42.3)
IDEC	12.8 ± 1.1 (14.8)	21.3 ± 1.5 (23.6)	70.3 ± 2.0 (74.2)	38.1 ± 1.1 (39.9)	32.5 ± 2.2 (36.5)
DCN	15.1 ± 1.1 (17.4)	18.9 ± 2.5 (23.9)	77.1 ± 3.5 (80.8)	38.5 ± 1.2 (41.3)	31.5 ± 2.9 (37.5)
DipDECK	00.0 ± 0.0 (00.0)	7.2 ± 1.8 (9.1)	70.7 ± 2.5 (74.4)	32.8 ± 3.3 (37.6)	22.1 ± 4.0 (29.0)

D.2 Derivation of the gradients

Let us assume that $A > 0$ and consolidate factors that are irrelevant for the calculation of the gradients (if $A < 0$, all gradients must be multiplied by -1.). Following holds:

$$\begin{aligned}\mathrm{dip}(\overline{Z}) &= \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_2}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} - \frac{\overline{z}_{\overline{m}_1}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}\right) + \text{const} \\ &= \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2}}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_1}}{p^T z_{m_3} - p^T z_{m_1}}\right) + \text{const}\end{aligned}$$

D.2.1 Gradient regarding the projection axis.

$$\begin{aligned}\frac{\partial}{\partial p[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N} \\ &\left(\left(\frac{z_{m_2}[i]}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_2}(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) - \right. \\ &\left.\left(\frac{z_{m_1}[i]}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_1}(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right)\right) \\ =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{p^T z_{m_3} - p^T z_{m_1}} + \right. \\ &\left.\frac{(p^T z_{m_1} - p^T z_{m_2})(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ = \frac{(\overline{m}_3 - \overline{m}_1)}{2N}&\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \frac{(\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2})(z_{m_3}[i] - z_{m_1}[i])}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

D.2.2 Gradient regarding the modal triangle.

Considering z_{m_1}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_1}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{(-p^T z_{m_2})(-p[i])}{(p^T z_{m_3} - p^T z_{m_1})^2} - \right. \\ &\left.\left(\frac{p[i]}{p^T z_{m_3} - p^T z_{m_1}} + \frac{(-p^T z_{m_1})(-p[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right)\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2} - p^T z_{m_1}}{(p^T z_{m_3} - p^T z_{m_1})^2} - \frac{1}{p^T z_{m_3} - p^T z_{m_1}}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2} - p^T z_{m_1} - p^T z_{m_3} + p^T z_{m_1}}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

Considering z_{m_2}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_2}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p[i]}{p^T z_{m_3} - p^T z_{m_1}}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}\right)\end{aligned}$$

Considering z_{m_3}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_3}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{(-p^T z_{m_2})p[i]}{(p^T z_{m_3} - p^T z_{m_1})^2} - \frac{(-p^T z_{m_1})p[i]}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_1} - p^T z_{m_2}}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

In summary:

$$\frac{\partial}{\partial z[i]}\mathrm{dip}(\overline{Z}) = \begin{cases} p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_1}, \\ p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} & \text{if } z = z_{m_2}, \\ p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_3}, \\ 0 & \text{else} \end{cases}$$

C.5 Benchmarking Deep Clustering Algorithms With ClustPy

Authors:
Leiber, Collin* and Miklautz, Lukas* and Plant, Claudia and Böhm, Christian
** First authors with equal contribution*

Abstract:
Deep clustering algorithms have gained popularity as they are able to cluster complex large-scale data, like images. Yet these powerful algorithms require many decisions w.r.t. architecture, learning rate and other hyperparameters, making it difficult to compare different methods. A comprehensive empirical evaluation of novel clustering methods, however, plays an important role in both scientific and practical applications, as it reveals their individual strengths and weaknesses. Therefore, we introduce ClustPy, a unified framework for benchmarking deep clustering algorithms, and perform a comparison of several fundamental deep clustering methods and some recently introduced ones. We compare these methods on multiple well known image data sets using different evaluation metrics, perform a sensitivity analysis w.r.t. important hyperparameters and perform ablation studies, e.g., for different autoencoder architectures and image augmentation. To our knowledge this is the first in depth benchmarking of deep clustering algorithms in a unified setting.

Venue:
23rd IEEE International Conference on Data Mining Workshops (ICDMW), Shanghai, China, December 1-4, 2023, IEEE, 2023.
CORE ranking: Not applicable to workshop papers
Acceptance rate: 50.3%

DOI:
`https://doi.org/10.1109/ICDMW60847.2023.00087`

Thesis-Reference:
[LMPB23b]

Division of Work:

Joint work of the first authors
Discussions on the benchmarking procedure; Implementing the algorithms using pair-programming; Designing the experiments; Co-writing large parts of the paper; Creating the visualizations

Collin Leiber
Implementing the benchmarking pipeline; Executing the experiments regarding the small feedforward autoencoder, convolutional autoencoder and generalization performance; Writing major parts of the introduction, background and benchmarking section; Creating the presentation slides; Presenting at the conference

Lukas Miklautz
Implementing the augmentation experiments; Executing the experiments regarding the large feedforward autoencoder, the impact of various hyperparameters and the augmentation analysis; Writing major parts of the experiments and discussions section

Claudia Plant
Discussions on relevant algorithms and experiments; Refinement of the final draft; Mentoring and supervision

Christian Böhm
Discussions on relevant algorithms and experiments; Refinement of the final draft; Mentoring and supervision

This paper was removed from the published version of this dissertation due to copyright reasons.