universität wien

Dissertation / Doctoral Thesis

Titel der Dissertation / Title of the Doctoral Thesis

Support Methods for Clustering with a Focus on k-means and Dataset-Transformations

verfasst von / submitted by

Benjamin Schelling

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 786 880
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dr. Claudia Plant

Abstract

In recent decades, humanity has taken Galileo Galilei's quote - "Measure what is measurable, and make measurable what is not so" - to heart and gathered vast amounts of data on all conceivable processes. These amounts of data have reached such a scale that it is no longer possible to evaluate them manually and computer support has become necessary. Therefore, the use of algorithms for automatic data analysis has become more and more relevant and made Data Mining an important field of research.

In this thesis, the focus is on clustering: dividing the data set into groups of data points so that objects in the same group are more similar to each other than to those in other groups. Since clustering is completely "unsupervised" – i.e. does not require the participation of a human user - it is ideal for a fully autonomous data analysis, but the choice of the algorithm used is crucial for the final result. If the algorithm is unsuitable, the analysis of the algorithm may be useless. The goal during my PhD was to develop methods to extend the range of data sets suitable for one of the most commonly used algorithms - k-means. Data sets that k-means could previously not or only insufficiently cluster are adapted to fit into the assumptions of k-means, thus enabling a correct analysis. For this purpose, I developed a mathematical framework that contains the foundations of a theory for such "Dataset-Transformations". Methods that transform data sets from a form that is unsuitable for k-means into a suitable form without changing the basic properties and characteristics of the data set. We present the methods that were developed as such Dataset-Transformations, as well as other support methods for clustering, and analyze how they fit into this framework and meet the required properties.

Zusammenfassung

In den letzten Jahrzehnten hat sich die Menschheit das Postulat von Galileo Galilei - "Alles messen, was messbar ist - und messbar machen, was noch nicht messbar ist" - zu Herzen genommen und riesige Datenmengen zu allen denkbaren Prozessen gesammelt. Diese Datenmengen haben ein solches Ausmaß erreicht, dass es nicht mehr möglich ist, sie manuell auszuwerten und deshalb Computer zur Auswertung herangezogen werden müssen. Der Einsatz von Algorithmen zur automatischen Datenanalyse hat daher immer mehr an Bedeutung gewonnen und Data Mining zu einem wichtigen Forschungsgebiet gemacht.

In dieser Arbeit liegt der Schwerpunkt auf Clustering: der Aufteilung des Datensatzes in Gruppen von Datenpunkten, so dass Datenpunkte in derselben Gruppe einander ähnlicher als zu Datenpunkten in anderen Gruppen sind. Da Clustering völlig "unsupervised" ist - d.h. keine Beteiligung eines menschlichen Benutzers erfordert - ist es ideal für eine völlig autonome Datenanalyse; allerdings macht dies auch die Wahl des verwendeten Algorithmus entscheidend für das Endergebnis. Wenn der Algorithmus ungeeignet ist, kann die Analyse des Algorithmus komplett nutzlos sein. Das Ziel meiner Doktorarbeit ist es, Methoden zu entwickeln, um Datensätze besser kompatibel zu einem der am häufigsten verwendeten Algorithmen – k-means – zu machen. Datensätze, die k-means bisher nicht oder nur unzureichend clustern konnte, werden so adaptiert, dass sie in die Anforderungen von k-means erfüllen, womit eine korrekte Analyse möglich wird. Zu diesem Zweck habe ich ein mathematisches Framework entwickelt, das die Grundlagen einer Theorie solcher "Datensatz-Transformationen" enthält. Methoden, die Datensätze von einer für k-means ungeeigneten Form in eine geeignete Form transformieren, ohne die grundlegenden Eigenschaften und Merkmale des Datensatzes zu verändern. In dieser Arbeit werden die Methoden vorgestellt, die als solche "Datensatz-Transformation" entwickelt wurden und analysiert, inwiefern sie in dieses Framework passen, das heißt die erforderlichen Eigenschaften erfüllen.

Danksagung

Als erstes möchte ich Claudia Plant dafür danken, dass sie mir damals die Chance gegeben hat ein Doktorat zu machen, sowie die Unterstützung währenddessen.

Desweiteren danke ich meinen Gutachtern, Prof. Sedlmair von der Universität Stuttgart sowie Prof. Roth von der Universität Wien.

Danke auch an Martin und Sahar, die zur gleichen Zeit wie ich begonnen haben und nun auch zur gleichen Zeit fertig werden. Ich kann nichts Schlechtes über Sahar sagen und über Martin auch nur, dass er vier Jahre lang die Heizung im Winter immer runtergedreht hat. Mein Dank gilt auch Prof. Lukas, bei dem ich mich eines Tages habilitieren werde, Lena, die die mit Abstand besten Korrekturen macht, Theresa, die es dann sein lies, Katharina, die mich manchmal verwirrt, Ewald, der sich als einziger mit Computern wirklich auskennt, sowie Ylli, Can, Robert und Max.

Meinen Eltern, die sich die Dissertation anschauen und stolz sind.

Ein besonderer Dank gilt meiner Physikerin für all die Unterstützung, die sie mir in nicht immer einfachen Jahren gewährt hat.

This too has passed.

Publications

Most of the results of this thesis were already published in conference proceedings and journal articles. Therefore, the chapters of this thesis are based on the following publications:

- **Paper A:** Benjamin Schelling, Claudia Plant, "KMN - Removing Noise from K-Means Clustering Results", International Conference on Big Data Analytics and Knowledge Discovery (DaWaK), 2018.
- **Paper B:** Benjamin Schelling, Claudia Plant, "DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering", IEEE International Conference on Data Mining (ICDM), 2018. Best Paper Canditate.
- **Paper C:** Benjamin Schelling, Claudia Plant, "Dataset-Transformation: Improving Clustering by enhancing the structure with DipScaling and DipTransformation", Knowledge and Information Systems (KAIS), 2019.
- **Paper D:** Benjamin Schelling, Claudia Plant, "DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering (Extended Abstract)", GI-Jahrestagung INFORMATIK, The best of Data Mining in D/A/CH, 2019.
- **Paper E:** Benjamin Schelling, Lukas Miklautz, Claudia Plant, "Nonlinear Cluster Enhancement: Forcing Clusters into a compact shape", European Conference on Artificial Intelligence (ECAI), 2020.
- **Paper F:** Benjamin Schelling, Lena Greta Marie Bauer, Sahar Behzadi and Claudia Plant, "Utilizing Structure-rich Features to improve Clustering", European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), 2020. Accepted.

Papers written during the time of my PhD, which do not fit the scope of this Dissertation:

- **Paper G:** Benjamin Schelling, Claudia Plant, "Clustering with the Levy Walk: "Hunting" for Clusters", IEEE International Conference on Data Mining (ICDM), 2016. PhD Forum.
- **Paper H:** Benjamin Schelling, Gert Sluiter, Claudia Plant, "RandomLink - Avoiding Linkage-Effects by employing Random Effects for Clustering", International Conference on Database and Expert Systems Applications (DEXA), 2020. Accepted.

- **Paper I:** Sahar Behzadi, Benjamin Schelling, Claudia Plant, "ITGH: Information-theoretic Granger Causal Inference on Heterogeneous Data", Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2020.
- **Paper J:** Sahar Behzadi, Benjamin Schelling, and Claudia Plant, "Information-theoretic Granger Causal Inference on Heterogeneous Data: Problem specication and algorithm", International Journal of Data Science and Analytics (JDSA), 2020. Submitted for Publishing.

Contents

Chapter 1

Data Mining and Clustering

1.1 Data Mining

Data Mining is a comparatively new field of study that emerged as an independent area of research in the 1990s [26]. Data Mining can be broadly described as "the process of discovering interesting and useful patterns and relationships in large volumes of data" [12]. It is often considered as part of the "Knowledge Discovery in Databases" (KDD) process, which takes a broader approach to data analysis and also includes the steps that come before Data Mining, i.e. data cleaning and pre-processing, as well as the steps afterwards, i.e. the interpretation of the results [21]. The goal, as stated, is to find interesting aspects of the data. Depending on the type of aspect sought, a distinction is often made between the following sub-fields of Data Mining:

- Outlier Detection - finding anomalies in the data
- Association Rules - finding relations between variables
- Classification - generalizing known information to new data
- Regression - finding a function that describes the data
- Clustering - finding groups of similar data points

Other sub-fields like Summarisation (finding a compact description of the data) or Dependency Modelling (finding a description for the dependency of the variables) are sometimes also explicitly listed as research topics for Data Mining [22]. Researchers do not always agree on what constitutes a sub-field and how they relate to each other or even under what name they should be subsumed. Classification, as an example, will sometimes be considered as a part of Machine Learning [43] and sometimes part of Data Mining [58]. The boundaries between Machine Learning and Data Mining are also not perfectly well defined. Data Mining-scientists sometimes consider Machine Learning as a subset of Data Mining and vice versa.

The focus in this thesis is on clustering and, more specifically, on supporting clustering algorithms by making data easier to cluster and reducing the impact of the assumptions made by these algorithms.

1.2 Clustering

Clustering, sometimes also called unsupervised classification (or Exploratory Data Analysis) [79], is one of the core research areas of Data Mining. The goal is to analyse a data set without the need for additional information. It is, therefore, an unsupervised problem, i.e. no ground truth is known and no human supervision is required. Data is fed into an algorithm and an analysis of the data is created by the algorithm without the need for any interaction by the user (excluding parameters). Since the algorithm (here) functions essentially as a black box, the risk of failure, i.e. a faulty analysis, is usually higher compared to a (semi-)supervised setting, where at least part of the ground truth is known. The advantage, on the other hand, is obvious: A completely automatic analysis of data. In many areas where data is generated continuously, experts to analyse the data are hard to come by or time is of the essence, an automatic preparation and processing of the data is necessary. For this, clustering can play an essential role.

Clustering focuses on grouping data points in a data set, without needing to know which data points actually belong together. There is, however, "no universally agreed upon and precise definition of the term cluster[ing]" [79]. Most definitions of clustering are variations of the following "definition": "similarities between objects in the same group are high while the similarities between objects in different groups are low" [36]. Essentially, the goal is that "data objects in the same cluster should be similar to each other, while data objects in different clusters should be dissimilar from one another" [79]. The difficulty here is that it is not possible to define "similarity" in a way that holds for all data sets. On some data sets, the Euclidean distance might be enough to estimate similarity, while for others density is the deciding factor or the neighbourhood of a data point is essential. The decision on what constitutes similarity is made when deciding on a clustering algorithm. Depending on whichever algorithm has been chosen the data set might be clustered differently. Only on very simple data sets or by chance clustering algorithms might reach the same final clustering result. Usually, one can expect clustering algorithms to find (at least slightly) different clusters.

In [72], the authors argue that there are two main reasons to use Clustering: For Data Processing and for Exploratory Data Analysis. For Data Processing, Clustering is simply a step in a pipeline, which needs to reach a certain goal, for example, a music recommender system. Clustering might be used so that only a subset of a huge database needs to be checked. In this context, clustering is simply a tool to be tuned for its performance in this pipeline and evaluated in this context. It is not important if different clustering algorithms produce differing results, as long as they serve their purpose. For Exploratory Data Analysis, Clustering is "used to discover aspects of the data which are either completely new, or which are already suspected to exist, or which are hoped not to exist" [72]. The hope is to obtain some insight into the process which generated the

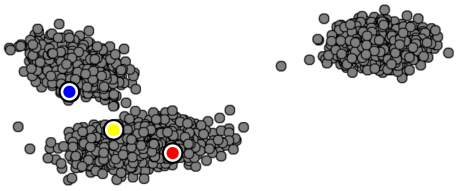
data and, thus, foster understanding. The hope is to learn something, which was not known before. Here, the choice of an algorithm can have a higher impact, as differing clustering results, of course, tell a different story about the data set. The possible insight is impacted, as the clustering results might cause a different interpretation of the data. This, however, does not mean that one is more valid than the other. They could simply have learned a different insight about the data. Data Processing and Exploratory Data Analysis are two different ways of looking at clustering, but they complement each other. The difference is merely whether one wants to use the found knowledge or whether one is interested in the knowledge itself. However, both need a meaningful grouping of the data points.

There are various well-known standard clustering approaches, which have become standard tools in many, if not most, software packages. From Weka [32], Scikit learn [55] or ELKI [68] one can always expect to find algorithms like EM [13], DBSCAN [19], SingleLink [69] or k-means [46, 47] and their various extensions like k-means++ [1]. In the (by now) decades of their existence, these algorithms have proven that they often perform well with good results and can be applied to a variety of problems. Their clustering results are often a meaningful grouping of the data points or at least useful. These algorithms are widely implemented and often highly optimised to reduce their runtime. Of course, for as many data sets where they perform well, one can find as many data sets, where other standard or state-of-the-art methods would be the better choice. The choice of the algorithm to be used is difficult, as it is not possible to confirm the result of the algorithm in an unsupervised setting, except by a thorough analysis of the final clustering result. However, this requires the use of a domain expert, which is contrary to the term “unsupervised” analysis.

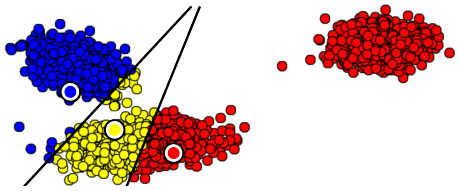
There exist several ways the clustering algorithms can be categorised - centroid-based, distribution-based, hierarchical, hard and soft assignments, etc - which can help to find an algorithm suited for the need of the user. Nevertheless, there is often an immense spectrum of methods to choose from. [20] claims that there are so many different clustering methods, “because the notion of ’cluster’ cannot be precisely defined” and what type of clusters needs to be found, depends on the specific application. It is unlikely that one finds an algorithm fitting perfectly for a specific data set and application, but one might find some which perform satisfactorily. This choice of the “correct” algorithm cannot be taken from the user, and this requires a certain amount of experience and/or knowledge of the data.

1.3 k-means

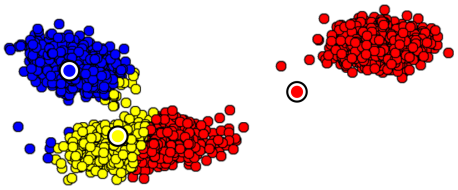
K-means is the best known and most often used clustering algorithm there is. It is studied intensively in scientific circles, with hundreds of papers analysing every aspect of it. Alone for the last 10 years, Google Scholar reports more than 400.000 papers with k-means in their title, demonstrating the impact of this algorithm. It can be found in every field of data-driven research, from speech recognition [11] to mineralogy [2]. It is



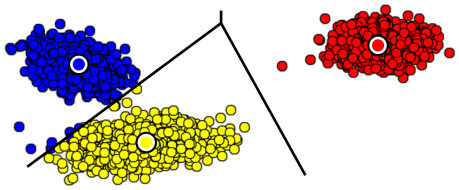
(a) Three data points are randomly selected as centres from the data set.



(b) All data points are assigned to their closest centre.



(c) The centres are updated as the mean of the data points assigned to them.



(d) The algorithm converged to the final clustering result.

Figure 1.1: The k-means clustering process.

highly influential and is considered one of the most important algorithms in all of Data Mining [77].

K-means often stands for a class of clustering approaches, of which the Lloyd/Forgy version [25, 46, 47] stands out as the most frequently used. This is the algorithm which we mean, if we do not clarify it any further and not, for example, the one by MacQueen [49], which is also often referred to as k-means. K-means is one of the cornerstones of clustering approaches. If there is a data mining problem, one can assume that there is a possible solution built on k-means. It has been extended and modified to handle outliers (e.g. KMN [65]), overcome hard assignments (e.g. Fuzzy k-means [16]), create a hierarchical clustering (e.g. bisecting K-means [41]), choose k automatically (e.g. Dip-Means), find arbitrary cluster shapes (e.g. Kernel K-means [67]), to be highly optimised regarding runtime (e.g. Multi-core k-means [4]), and so on. It has become a framework, a starting point for various problems, for which k-means can be adapted to suit various needs and demands depending on the specific problem. See [37] for a detailed historical analysis of k-means with descriptions of several variations.

K-means partitions the n data points into k clusters by assigning every data point to the closest of k cluster centers. The algorithm proceeds iteratively, alternating between updates of cluster centres and assignments of data points, until the algorithm converged according to the objective function. The algorithm starts with k initial cluster centres (Fig. 1.1a). These cluster centres are either chosen randomly from the n data points or with an initialisation strategy like k-means++ [1]. The data points are then assigned to the closest centres. Thus, the space forms now Voronoi cells [14] (Fig. 1.1b). Voronoi cells are defined in the following way:

Definition 1. The Voronoi cell of a centre μ is defined as

$$\{x \in \mathbb{R}^d \mid \ ||x - \mu|| \leq ||x - \hat{\mu}|| \text{ with } \hat{\mu} \text{ centres of k-means}\}$$

with ||.|| the Euclidean distance.

The definition here is restricted to the standard Euclidean space $\mathbb{R}^d$ with the associated distance but is easy to generalise. Essentially, all data points are assigned to the closest centre μ of k-means, which causes a partitioning of the space.

The centres are then updated by computing the mean of the assigned data points which becomes the new centre (Fig. 1.1c). In the MacQueen-version of k-means the update-process is different. Data points are assigned to clusters one by one and the centres are updated (as the mean of the assigned data points) after every data point is assigned to a cluster. For Lloyd/Forgy centres are only updated, when every data point is assigned to a cluster. MacQueen would, in the original version, stop after a single pass through the data set, but can also be continues iteratively like Lloyd/Forgy until the algorithm converges (Fig. 1.1d).

It is easy to see that k-means always converges due to the data sets being discrete (a proof can already be found in [49]). Thus, a (local) minimum of the objective function

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} ||x_i - \mu_j||^2 \tag{1.1}$$

of k-means is found. C_j is the cluster of centre μ_j, i.e. all data points x_i which are assigned to it. The objective function is satisfied, i.e. every data point x is assigned to one of the k centres, so as to minimise the within-cluster sum of squares. Finding a local optimum of the objective is straightforward, simply running k-means until convergence, but there is no guarantee that it is also the global optimum. In fact, finding the global optimum is a problem known to be np-hard [17]. K-means has a linear runtime in the number of data points n, the dimensionality d, the number of clusters k and the number of iterations [34].

1.4 No Free Lunch Theorem - The Assumptions of Algorithms

We have established that no clustering algorithm is a perfect choice for all data sets. This is due to all clustering algorithms having assumptions about the data set. It does not matter if someone uses DBSCAN, EM, SingleLink or any other clustering algorithm, all possible algorithms have some - implicit or explicit - assumptions about the data. These assumptions determine how the algorithm defines similarity which in turn determines what constitutes a cluster. If the assumptions are fulfilled, the algorithm is a viable choice, otherwise, other algorithms would be better suited. Wolpert and Macready formalised this in [74] and [75], showing that "there is no single algorithm that is best suited for all possible scenarios and data sets" [24]. They effectively prove that "if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems" [74]. In an unsupervised setting, where nothing is known about a data set except the values of the data points,

there is no way of knowing whether an algorithm is a good, acceptable, or downright terrible choice.

Wolpert himself links this to the famous quote of Hume: "Even after the observation of the frequent conjunction of objects, we have no reason to draw any inference concerning any object beyond those of which we have had experience" [35]. This is part of Humes famous discussion about the impossibility of induction of which Popper said that Hume "was perfectly right in pointing out that induction cannot be logically justified" [57]. This problem is highly linked to clustering, as there is no reason to draw any inference why a clustering algorithm is an appropriate choice, only because it worked well on the first few data sets. Assume we have a sequence which begins with the numbers, a_1=1, a_2=2, a_3=3, a_4=4, a_5=5, a_6=6, a_7=7, a_8=8, a_9=9. What is the next number in the sequence? The next number a_{10} need not be 10. It could be 11 if the underlying generating process was for palindromic numbers and not natural numbers as one might assume. This is essentially the same problem as for clustering. We have a limited amount of data, i.e. {1,2,3,4,5,6,7,8,9}, which we use to guess the next value. To get there, we have to add an assumption about the data, that is, how these numbers are generated. The problem is that in principle, any number could be the next one as the construction rule could be any arbitrary rule one can think of. Theoretically, all numbers are possible and none should be favoured, but if one had to choose between $a_{10} = 10$ and, say, $a_{10} = 31415$, most would agree that $a_{10} = 10$ is more reasonable. This argument follows Occam's Razor, that "the simplest solution is most likely the right one". Occam's Razor is "a principle that is both intuitively appealing and informally applied throughout all the sciences" [30]. $a_{10} = 10$ is not more probable in the mathematical sense, but the one that corresponds most to all our experiences in the real world[1]. To give a more obvious example: In 2011 scientists at the Large Hadron Collider measured a neutrino that moved faster than light. In theory, this could have been correct, but what happened was that scientists found a damaged aspect of the experiment. It would be possible to deduce from the experiment that all of physics had to be revised, but following Occam's razor, it was simply more plausible that an error had been made.

Clustering follows essentially the same lines. Of course, the data could be generated by an extremely complicated procedure, following a never-before-observed distribution, which, due to random fluctuations, could still not be observed. But, it might also simply be a few Gaussian processes for which k-means is a good choice to analyse the data. The assumptions of k-means about the data are comparatively low, based on a more straightforward generative process. Due to this simplicity, it performs very well on a large range of data sets and its analysis of the data often makes sense, even if the clusters somewhat deviate from the Gaussian-distribution assumption of k-means. With additional knowledge and experience of the user, more complicated algorithms, which entail more detailed assumptions about the data, can be better fitting tools, which allow for a more precise analysis of the data and a more accurate partitioning of the data points. In the absence of this additional information, k-means is a great choice as a first exploratory step.

[1]Tying this into the discussion between a Frequentist and Bayesian interpretation of probability.

1.5 Transforming the Problem

Data sets created by an unusual generating process will be difficult to cluster. It is not trivial to find an algorithm suited for them, and if the generation process is rare enough, it may even be the case that no algorithm was yet created for this type of data set. Therefore, it may not be possible to cluster it properly. While data sets created by a simpler process can be clustered, a suitable algorithm must still be found. Finding it, however, is not straightforward. If the user has good domain knowledge, choosing a suitable algorithm is more likely, but even then, it is not guaranteed that the algorithm is fitting. Whichever way an algorithm is chosen, if the choice is made, all assumptions of the algorithm are accepted. The objective here is to make this decision the "right" one.

Instead of finding an algorithm perfectly suited to the data set - or even creating a new one - the intention in this thesis is to make the data set better suited to the algorithm. By transforming the data set such that the clusters become more pronounced and, thus, easier to find, the risk in choosing an unfitting algorithm is reduced. The easier a data set is to cluster, the more likely a "correct" clustering can be found.

The algorithm, for which we mainly focus on transforming the data set to match its assumptions, is k-means. The idea is to reshape the data set, such that the assumptions of k-means are more likely to be fulfilled. While it is not possible to remove the assumptions of k-means (or any other clustering method), it is possible to try to widen the scope of data sets suited to these assumptions. The idea is to accept the assumptions of k-means, but soften the impact they have by making the data set inherently better suited to these assumptions. Thus, the assumptions of k-means become less severe and data sets, which were before only partially suited to k-means and could only be partly analysed, are now more fitting and the analysis more correct.

K-means assumes Gaussian-shaped, well separated, non-overlapping clusters. If it is possible to transform a data set into such a form, it has taken one of the easiest forms to cluster. Most clustering algorithms are (at least in theory) capable of handling such a setting, thus, they might also have become suitable choices for such a data set.

We focus mainly on k-means, because it is a) widely known and valued with a wide range of implementations, b) it performs adequately on quite a variety of data set, which means that its assumptions are not too far off for many real-world applications, c) improvements for k-means are likely to transfer to other methods as well, d) its behaviour is well understood, which makes it easier to understand its needs to perform even better.

1.6 Supporting the Problem

By transforming the data, the goal is to make clusters easier to find and have them fit better into the assumptions of the algorithm. Transformation of the data, though, focuses only on the type of assumptions that affect the generating process. Re-shaping the clusters and having them, in the case of k-means, fit into Voronoi cells, is an effort to influence the position and form of a cluster, i.e. how they were created. The princi-

ple of the clustering algorithm, nevertheless, is still the same and the the assumptions concerning it still stand.

As an example, k-means can neither estimate the number of clusters by itself nor can it extract outliers from a data set. To solve these types of problems, merely transforming the data is not enough, as k-means has simply no functionality to this effect. However, there exist various possible adaptions of the basic algorithm, which have the intention to extend the framework of k-means to this. K-means, as stated, is the most often used clustering algorithm. It has received wide-spread attention and many extensions of it have been proposed, which solve some of its problems. So has its inability to find the number of clusters been the reason to create methods like X-means [56], DipMeans [40] or PG-means [23]; all of these methods try, in a k-means like fashion, to estimate the number of clusters k. These types of k-means-like algorithms are one of the most active branches of "number of clusters"-estimating algorithms as estimating k is one of the fundamental problems of clustering. Finding outliers in a data set is also a difficulty which cannot be handled by many clustering algorithms. For this, it is necessary to apply a technique like LOF [6] to "clean" the data set from outliers or use an adaption of k-means like k-means-- [9], which has been extended to such situations (at the cost of an additional parameter).

These types of problems exceed the capabilities a transformation of the data has. A transformation is restricted to supporting the algorithm with respect to the assumptions of the generating process. For problems like estimating the number of clusters, specific algorithms or extensions of existing algorithms are used, which target a different type of assumption. The goal for both types of methods is the same: to lessen the impact of the assumptions on which the algorithms are built upon, but while one focuses on general assumptions, one focuses explicitly on the assumptions of the generating process. We refer to these as **"Dataset-Transformations"**. They explicitly try to transform the data set, make the clusters easier to find, but they do not influence the essentials of the clustering method. The more general methods, which focus on the assumptions not necessarily related to the generating process, we refer to as **"Support-methods"**: Methods, which support the clustering process, by e.g. estimating the number of clusters. Both deal with the assumptions of the clustering methods and intend to bypass the involved difficulties. Dataset-Transformations can be seen as a special type of Support-methods.

While there exists a relatively large class of Support-methods and many representatives are well-known, there are relatively few algorithms focusing on the assumptions related to the generating process. Such Dataset-Transformations are therefore on the forefront of the efforts undertaken here.

We will now look at some of the properties that such Dataset-Transformations should have, before we move on to methods that have these properties or meet the intention described here.

Chapter 2

A mathematical Framework for Dataset-Transformations

In the last chapter we have seen the difficulties involved in choosing a clustering algorithm and, in particular, the assumptions they add to the problem. If, as an example, k-means is chosen, one of the implicit assumptions is that the generating process of the clusters is based on a Gaussian distribution. If the data does not satisfy these assumptions, there is a high chance that the clustering result will be inadequate. The assumption of the clusters following a Gaussian distribution is not extraordinary, especially compared to the various assumptions of other clustering algorithms, but it is impossible to know whether the assumption is satisfied. If it were possible to test whether the assumptions of an algorithm are satisfied, the setting would no longer be an unsupervised one, hence, one would have left the area of clustering. Consequently, there is no way of knowing if the assumptions of a method hold and if the algorithm is a "correct" choice or not. With deciding on a clustering approach, one has implicitly accepted all assumptions of the algorithm.

While we cannot solve this problem and have to accept the inherent risk of failure involved with employing clustering algorithms, we can try to lessen the risk. As stated before, every clustering algorithm can be an erroneous choice according to the No-Free-Lunch Theorem. Nevertheless, some methods are preferred by the users, as they perform better in their experience. This is certainly due to their assumptions about the data following Occam's Razor by being less complicated and, thus, corresponding more often to the real world.

The goal, as stated, is to transform data sets into a shape that is more suitable to these preferred methods or, more precisely, to k-means. If k-means is chosen for a data set for which it is not suitable, it might still be possible to change the data set so that it becomes suitable for k-means. This means, that while the generating process of the data set deviates from the implicit assumptions of k-means, it may be feasible to re-shape the data to suit them. Transforming data, though, means that we have to transfer the original data D into a new shape D', with D' having the same properties as D, but an easier shape to cluster. For this, the map from D to D' must have some properties,

which we wish to motivate now.

2.1 Continuity

The first aspect is that a data set transformation should not introduce rips in the data, which did not exist before. If two data points are close in D, they should also be close in D'. Thus, we essentially ask the transformation to be continuous.

Definition 2. A function $f : X \to Y$ is continuous in $x_0 \in X$, if $\forall \epsilon > 0\ \exists \delta > 0$, such that for all $x \in X$, if $|x - x_0| < \delta$, follows $|f(x) - f(x_0)| < \epsilon$. If f is continuous in all $x_0 \in X$, then f is **continuous**.

Continuity ascertains that the neighbourhood of a data point stays remains similar. If two data points are somewhat close to each other, they will also be close to each other after the transformation.

Imagine a non-continuous transformation, which moves the data points along the cluster assignments of k-means. Such an example is shown in Figure 2.1. The information gained by k-means is taken here absolute and the cluster separated according to this information by simply moving them further from each other. While the actual shape of the true clusters is still visible from the Figure - two Gaussian, slightly stretched clusters - this is far more difficult to cluster after the transformation. It is not clear anymore how the data points should be separated and how a correct clustering would look like. The additional structure, which has been created through the separation of the k-means cells, overwhelms the information of the Gaussian shape of the clusters. Clustering this new, transformed data set has become far more difficult through this added structure. Thus, to ensure that the essentials of the data stay the same, a data set transformation needs to be continuous.

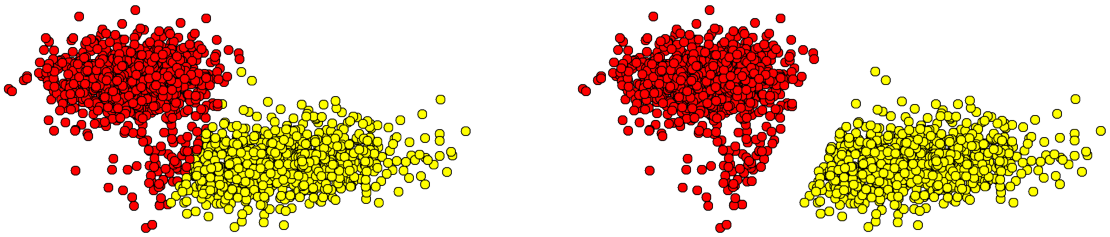


Figure 2.1: A simple data set with the colours showing how k-means would cluster it (left). The effect of moving the clusters with their k-means assignments non-continuously (right).

Definition 2 defines continuity for a function f from a space X to Y with $|.|$ a meaningful distance function. Clearly, $D \subset X$ has to hold as well as $D' \subset Y$. For the sake of simplicity we assume from now on $X = \mathbb{R}^d$ and $Y = \mathbb{R}^{d'}$ with $|.|$ the Euclidean distance. The following remarks, though, hold for a far wider range of spaces and distances and generalising these considerations can be easily done.

We stated, that we assume X to be equal to $\mathbb{R}^d$ with D a subset of $\mathbb{R}^d$. A function is continuous, if it is continuous in all points $x_0 \in \mathbb{R}^d$. One possible consideration is if it might be enough if the function is only continuous on all $x_0 \in D$, essentially reducing the continuity to the continuity of a discrete set. We discussed this difficulty and various other aspects already in [62], but need to recall it here. Let us assume that f is continuous only on all $x_0 \in D$. Since D is a discrete set, there is a minimal distance δ_{min} between all pairs of data points. Thus, if we set δ smaller than this minimal distance, there can be a point $\hat{x}$ for any data point $x_0 \in D$ with $|\hat{x} - x_0| > \delta_{min}$, for which holds: for all $x \in \mathbb{R}^d$, if $|x - \hat{x}| < \delta_{min}$ but $|f(x) - f(\hat{x})| > \epsilon$. Essentially, this allows for a non-continuous effect to happen for this function, if it is only the smallest bit away from a data point. It would be possible to create non-continuous spheres around each data point, such that the transformation is continuous inside the sphere, but any arbitrary re-arranging of the data points is possible outside this sphere. This, effectively, allows transforming any data set into any possible shape, with no regards to its earlier shape. Thus, we need the transformation to be continuous for more than the discrete set D. It is possible to restrict the continuity requirement, but for now, we assume that all points $x_0 \in \mathbb{R}^d$ are continuous transformed.

2.2 Bijection

Continuity ensures that a data set is not ripped apart. With the transformation being a bijection we try to ensure that the decisions of a transformation do not determine the result of a clustering algorithm.

Definition 3. A function $f : X \to Y$ is bijective, if $\forall y \in Y$, there exists exactly one $x \in X$, such that $f(x) = y$.

If a function is bijective, then there exists an inverse function f^* such that $f^*(f(x)) = x$. Thus, all decisions of the transformation can be undone. For every transformation, there exists an inverse transformation which turns the data set back into its original form.

Consider the Figure shown in 2.2. The same data set as before has been mapped onto very few data points. Hence, the condition for bijectivity is no longer fulfilled, as

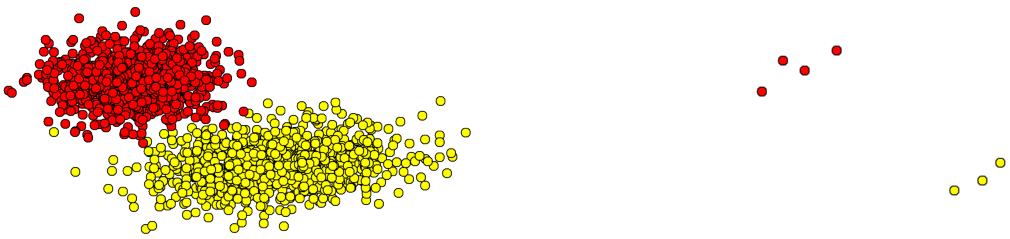


Figure 2.2: The simple data set from before (left), but a transformation has now mapped various data points together, such that they cannot be separated (right). They now have to be clustered together.

$f(x_1) = y = f(x_2)$ with $x_1 \neq x_2$. As multiple data points have been projected onto the same point, they have become indistinguishable from each other. The data points now have exactly the same properties, which makes them identical to a clustering algorithm and forces the algorithm to cluster them together. Therefore, the clustering algorithm cannot make this decision itself. The goal of a transformation is to highlight the structure in the data set and simplify the clustering process, but not to cluster the data. If such a decision is forced on an algorithm, it is equivalent to already having clustered part of the data set. An advantage of having various clustering methods is that different clusterings can tell different stories about a data set. This advantage is negated if the clustering is (partly) pre-determined. Thus, a transformation needs to be bijective, i.e. invertible.

2.3 Homeomorphism

Bijectivity ensures that the effect of a transformation can be undone, i.e. decisions are not forced on a clustering algorithm. Continuity ensures that the neighbourhood of a data point stays the same, i.e. if two data points are close before the transformation they are also close afterwards. This, nevertheless, allows for an extensive reshaping of a data set. In fact, this potential reshaping is still unrestricted enough that something like shown in Figure 2.3 can happen.

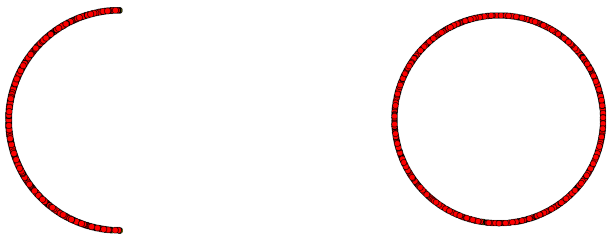


Figure 2.3: By moving each data point by a dependent rotation angle from the centre of the circle it is possible to "close" the halfcircle (left) and transform the data into a circle (right), while being continuous and invertible.

Continuity ensures that no additional rips are created, but does not prevent that rips are closed which were previously present. To prevent this, the inverse function also needs to be continuous. Thus, we have arrived at the definition of a homeomorphic map.

Definition 4. A function $f : X \to Y$ between two topological spaces is a **homeomorphism** if is has the following properties

- f is a bijection
- f is continuous
- the inverse function f^{-1} is continuous

$\mathbb{R}^d$ with the Euclidean distance is a topological space, thus, we need only concentrate on the three requirements of the map being bijective, continuous and the inverse function being continuous. The effect of a function being homeomorphic is, that the basic structure of the space is kept the same. It neither opens nor closes gaps in the data space and keeps the neighbourhoods intact. A homeomorphic map is often described as a continuous stretching and bending of an object into a new shape. Thus, the object can be pulled and indented any way. As long as no tears occur, the transformation behaves as it should. A cluster can be reshaped in a rather extensive fashion, e.g. a half-moon into a circular disc, while still keeping the neighbourhood of a point intact, but it is not possible to create or close holes and, thus, to introduce new structure which was not present before. A data set under a homeomorphic mapping keeps the characteristics of its shape, but can still be transformed so that it becomes far easier to cluster.

Isotopy

We have argued why various requirements of a mapping function are necessary and have finally arrived at a homeomorphic mapping. The next requirement that is commonly added is the one that makes a Homeomorphism into an Isotopy. In a Homeomorphism, a data point x is moved to a place $f(x)$ with the given requirements, but there is no further requirement of the path from x to $f(x)$. For an Isotopy, there needs to be a continuous path between them and the paths for two data points x_1 to $f(x_1)$ and x_2 to $f(x_2)$ are not allowed to cross. Isotopy is a further limitation, i.e. every Isotopy is a Homeomorphism, but not every Homeomorphism an Isotopy. One of the simplest examples for such a Homeomorphism which is not a Isotopy is a trefoil knot which is not isotopic to a loop. Considering that a trefoil knot is a rather complicated structure, it is not likely that one will stumble upon such a structure in a real-world data set. Thus, we would argue that the distinction between Homeomorphism and Isotopy is not something required in almost all real-world cases. Furthermore, most algorithms proceed iteratively or in a single step. Thus, a data point x is moved to its position $f(x)$ stepwise and it is rarely possible to determine whether there is a continuous path. Thus, it would not be possible to verify if the Isotopy-requirement is fulfilled or not. Hence, we argue that Homeomorphism is the meaningful level of requirements of such a transformation.

2.4 Already existing Methods

There exist some methods, which can be considered as Dataset-Transformations. We will consider these methods first and analyse in how far they suit the framework. Potentially fitting are the following methods:

- Normalisation approaches like Z-Transformation
- Feature Extraction/Selection methods like PCA [54]
- General Dimensionality reduction methods like t-SNE [48]

- Clustering approaches like SYNC [5]
- Distance-changing methods like kernel-k-means [67]
- Neural-Network methods like IDEC [31]

We discuss these methods insofar in-depth as to decide whether they are Dataset-Transformations and the discussion can be applied to the methods presented in Chapter 3.

Normalisation approaches

Normalisation approaches like min-max-normalisation are often used as a pre-processing step for Clustering. However, their use is barely taken note of. In various papers, one can find some variations of the phrase "the data sets were normalised before clustering" without any further elaborations. Normalising a data set has a clear and direct influence on the data set as it often drastically changes the final clustering result. The two most common normalisation methods are min-max-normalisation which re-scales all features into a range of $[0, 1]$ and Z-Transformation which ensures that all features get a variance of 1 and mean of 0. The goal of all normalisation approaches is that the features are in a comparable range and, thus, have a similar influence on the clustering result. The features are supposed to contribute equally to the clustering result, as in an unsupervised setting it is not clear which features are important for clustering. Ignoring any change in position that affects all data points equally, e.g. a move by a fixed value, normalisation can be described as a rescaling of the values in a feature. Thus, for a data point x in the feature i the equation

$$x'_i = x_i \cdot \delta_i \tag{2.1}$$

describes the rescaling of said feature with a value δ_i, with δ_i determined by the type of normalisation. Thus, the normalisation for the d-dimensional data point x is given by the diagonal matrix

$$A = \begin{pmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_d \end{pmatrix} \tag{2.2}$$

such that $x' = A \cdot x$ is the rescaled data point. It is easy to see that such a transformation fulfils all requirements listed in Chapter 2 for Dataset-Transformations.

Theorem 1. The Transformation given through Eq. (2.2) is continuous, invertible and its inverse function is again continuous.

Proof. 1) $x' = A \cdot x$ is continuous:
Let $y = x + \epsilon$. From this follows that $|Ay - Ax| = |A(x + \epsilon) - Ax| = |A\epsilon|$ and $|A\epsilon| \to 0$ for $\epsilon \to 0$ and Definition 2 is fulfilled.
2) $x' = A \cdot x$ is invertible.
$Det(A) = \delta_1 \cdots \delta_d > 0$, as $\delta_i > 0$. Thus, there exists an inverse Matrix A^{-1}.

3) The inverse of $x' = A \cdot x$ is continuous:
It is easy to see that the diagonalmatrix $A^{-1} = diag(\delta_1^{-1}, \ldots, \delta_d^{-1})$ is the inverse to A. The rest is anloguous to 1). □

Thus, all requirements for Dataset-Transformations are fulfilled by Normalisation-methods. This proof holds for all normalisation methods, as they only differ in how $\delta_i(D)$ are computed, which, of course, also depends on the specific data set D. These types of pre-processing methods fitting our definition is not surprising as their goal is comparable to a Dataset-Transformation. The intention is to reshape the data such that all features have a similar influence on the clustering result, thus, no single feature has an excessive influence on it and the clustering algorithm can, therefore, create a more balanced result. This follows, to a certain degree, the intention of Dataset-Transformations, which intend to make clusters easier accessible. Therfore, one can count them as Dataset-Transformations.

Feature Extraction and Selection

Feature selection methods choose a certain subset of the axes-parallel features and remove the rest. Feature extraction does the same but is not restricted to axes-parallel features. These methods can be described with simple matrices. Feature selection is simply a diagonal matrix, with 1s at the positions of features which should be kept and 0s elsewhere. Without loss of generality, this matrix is given by $\text{diag}(1, \ldots, 1, 0, \ldots, 0)$ or

$$A = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{2.3}$$

respectively. This matrix changes a data point x from

$$\begin{pmatrix} x_1 \\ \vdots \\ x_i \\ x_{i+1} \\ \vdots \\ x_d \end{pmatrix} \text{ to } \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Feature extraction can be described in a similar way. The main difference is that non-axes parallel features might be extracted. Thus, there needs to be an invertible matrix applied to the data points before the diagonal matrix is applied. Both Feature extraction

and Feature Selection, though, extract some of the features and remove others, whether they are axis-parallel or not. Hence, both of them reduce the dimensionality of a data set and, thus, for both of them holds the following paragraph.

Dimensionality reduction

A map from a space $f : X \to Y$, with X being of higher dimension than Y, e.g. $X = \mathbb{R}^3$ and $Y = \mathbb{R}^2$, can never be bijective making it impossible to find an inverse function. Thus, neither bijectivity nor homeomorphism can be obtained for such mappings. A map from a higher-dimensional space into a lower-dimensional one always loses some information and cannot be undone. Thus, all dimensionality reduction methods (and Feature extraction and Feature Selection-methods), like t-SNE, PCA or ICA, can, by default, never fulfil all requirements listed in Chapter 2.

One possibility to circumvent this and make dimensionality reduction methods potentially fulfilling these requirements is to restrict these requirements such that they need only be valid onto the data points and not the whole space. Thus, a map f is only bijective on $f : D \to D'$. This, though, has the consequence that the proof whether a method is a Dataset-Transformation depends on the specific data set. The same transformation could be a Dataset-Transformation on one data set but not on another. This property is not desirable and, thus, this possibility should not be considered.

Other approaches

We will now go into these categories in a little more detail.

Changed Distances Kernel-approaches change the distances of data points, without actually moving data points. Thus, while they technically do not change the data set, they change how the data set is clustered by an algorithm. Their goal is to make clusters, which are not separable by a hyperplane in lower dimensions, separable by a hyperplane in higher dimensions. Thus, their approach is somewhat comparable, as they also intend to make data sets easier to cluster. Their approach, though, is difficult to subsume under our criteria. Following [38], kernel functions do not adhere to the triangle inequality, thus their "distortion" of the space is not necessarily a metric. Considering all these differences, it seems forced to try to subsume the concept of a kernel method under Dataset-Transformations. That is not to say that approaches with "distorted" distances cannot fall under this category. Even some kernel functions can fall into it. As a trivial example, the identity kernel fulfils all requirements of a kernel function and also the requirements listed in Chapter 2. A slightly more interesting example would be EWKM [39]. EWKM is a Feature Weighting method, which determines the importance a feature should have for clustering. It is functionally identical to a Normalisation-method. Consequently, while such "distorted distance" methods can, in individual cases, be considered a Dataset-Transformation, they are in general too different from our approach. It is even questionable if they can be counted as a pre-processing step for clustering, as

there are clustering algorithms, which are not directly working on the distances between data points, thus, "distorted distances" could have unforeseen consequences.

Neural Networks For Neural Networks, the question of whether one can consider them as a Dataset-Transformation in our sense has to be decided on a case-by-case decision. DEC [78], as an example, can project two data points onto the same position, thus, it is not a Dataset-Transformation. On the other hand, the "trivial" Neural Network with the corresponding matrix the identity, does not change the dimensionality and fulfils all requirements. It is admittedly extremely unlikely that this case happens, but it is technically possible. Thus, we have a similar situation like for kernel approaches, meaning that some Neural Networks can fall into our category and some do not. Just as with kernel methods, it is not always possible to use an arbitrary clustering algorithm after applying a Neural Network. Thus, while some representatives could be counted, it makes no sense to consider these types of methods as generally belonging into our framework and a connection should only be drawn for special cases.

We covered the motivation why the methods we developed were developed and added the framework for Dataset-Transformations. We will now talk about the methods created during the PhD and how they fit into these categories.

Chapter 3

Support Methods

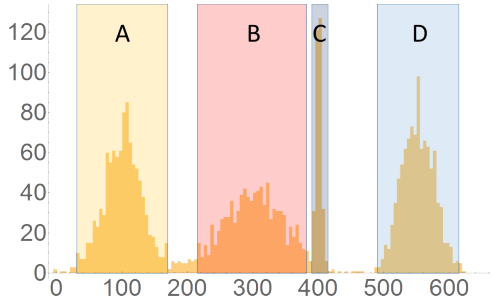
The Dip Test

Since the dip test plays an important role in various methods that we have developed and presented, a short introduction is given.

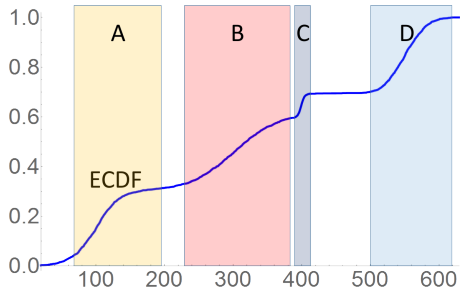
The dip test is a statistical test developed by Hartigan and Hartigan in the 1980s. It calculates the probability of a one-dimensional sample being unimodal or containing several modes. Assuming that a cluster is unimodal, it is possible to estimate whether a sample contains several clusters.

The dip test sorts the input sample and creates an Empirical Cumulative Distribution Function (ECDF) from it. In Fig. 3.1a and b the sample, as well as its ECDF, can be seen. In the histogram of the sample 4 clusters (A, B, C and D) can be easily identified. The same 4 clusters can also be seen in the ECDF. For the dip test, only the ECDF is required. The dip test proceeds by measuring how much the ECDF deviates from a unimodal distribution. It does this by measuring how much the ECDF has to be "offset", such that the closest unimodal distribution fits into this offset. This can be seen in Fig. 3.1c, showing the ECDF $\pm$ the offset. This offset is called the dip value. The dip value is large enough, such that a line can be spanned in between ECDF + dip value and ECDF - dip value. This line corresponds to the ECDF of a unimodal distribution - it is first convex and then concave - and it is not possible to span a unimodal distribution with a smaller offset, as the first convex/then concave-property would not be possible. Thus, the dip-value shows the distance of a sample from a unimodal distribution.

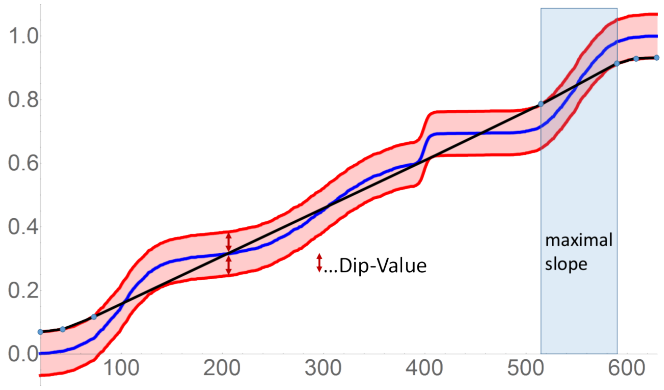
This dip value (or "dip") is then used to compute the probability of the sample being unimodal. This is obtained by simulating how the ECDF of a unimodal distribution would fare under this test, i.e. what its dip would be. Since running all these simulations would require a rather large runtime for a single test, the original paper by Hartigan and Hartigan (as well as most implementations) contains a look-up table which gives probabilities for the dip values. The Dip-Test has a runtime of $O(n)$, with n the size of the sample. Though, since the input must be sorted, the runtime effectively increases to $O(n \log n)$.



(a) A one-dimensional sample consisting of 4 clusters. The clusters are non-overlapping, but of different size and spread.



(b) The corresponding ECDF (Empirical Cumulative Distribution Function) of the sample.



(c) The ECDF is shifted up and down by the dip value until a first-convex-and-then-concave line (a unimodal distribution) can be fitted inside.

Figure 3.1: A histogram and its ECDF (Empirical Cumulative Distribution Function). The dip test measures via the ECDF how much the sample deviates from a unimodal distribution. This distance to the closest unimodal distribution is the dip value.

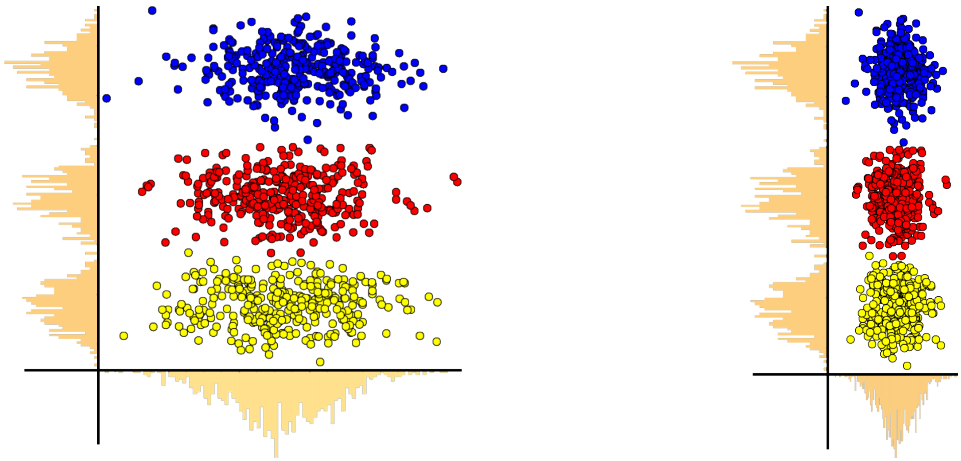
We make use of the dip test as it is very capable in determining whether or not a feature contains multiple peaks (as in Fig. 3.1a). If there are several peaks, it follows that there are several concentrations of data points. Based on the assumption that concentrations of data points that are separated from each other correspond to separate clusters, the dip test can be used to determine whether a feature contains multiple clusters. If a feature contains only a single cluster (i.e. it is unimodal), then this feature is of no particular interest for a clustering method. If, on the other hand, multiple clusters are contained, then this feature is highly important for clustering, as the boundaries between the clusters are easier to find here.

With this reasoning, we consider the dip test an extremely useful tool in the context of clustering. It is parameter-free, thus, functions in an unsupervised setting and it has a good runtime. Despite its advantages, it has until now barely been used in the field

of Data Mining. The only methods which make use of it and were published before the methods presented in the following chapters, were DipMeans [40] and SkinnyDip [51]. DipMeans is a method to estimate the number of clusters, SkinnyDip a method to cluster in the presence of extreme noise. Recently, other researchers have also found the dip test of interest, and methods based on it have been published. Most of them focus on generalising the dip test to higher dimensionality [10, 70] and to estimate the number of clusters [8, 80]. For our approaches, we focus on the original version of the dip test and restrict ourselves to measuring the likelihood of multiple peaks in a sample.

3.1 DipScaling

DipScaling [62] (Appendix C) is the basis for the DipTransformation [63] and contains the underlying concepts for many of the following methods. Its primary intention is the one mentioned in the introduction of the dip test: A feature which contains multiple concentrations of data points, i.e. multiple separate clusters, is more important for clustering than a feature where clusters are indistinguishable. As an example, consider Fig. 3.2a.



(a) A simple data set and its projections onto the axes.

(b) The same data set after it has been rescaled with its dip values.

Figure 3.2: Applying DipScaling to a simple data set.

The features shown as projections onto the axes in Fig. 3.2a both contain the same clusters. They are both projections of the same data set in a different direction but contain very different levels of information relevant for clustering. One feature contains (under the stated assumption of a unimodal cluster) the boundaries between the clusters, which can easily be found and used for clustering. In the other feature, clusters simply blend into each other and create one giant mass of data points. The borders between clusters are essentially wiped out and they blend into each other so that they cannot

be separated. Thus, one feature contains the relevant information for clustering (the boundaries of the clusters), while the other one does not. It is, of course, possible, that the combination of features reveals information about the clusters, which a single feature cannot, but it is a reasonable conclusion, that the feature with multiple peaks should play a more important role for the clustering process. As stated, we focus mostly on k-means, thus, we ask the question of how k-means handles a data set and how a single feature influences this result. K-means computes the centres of the clusters it finds as the mean of the data points. All data points in a cluster are summed up and the average of them is the centre. Thus, the centre is dependant on the numerical value of the data points. If these numerical values are in a certain range, then the centre will also be in this range. If this range is influenced, then the centre will be influenced similarly. We want the clusters, i.e. the centres, to be more influenced by the features with a high dip value, than by a feature with a low one. If the range of a feature with a low dip value is smaller, it has less influence on determining the centre. Hence, if features with small dip values are re-scaled into a smaller range, they are less important for clustering. DipScaling does exactly this. It measures the dip value d_i of feature i and re-scales this feature into the range $[0, d_i]$. The effect of this can be seen in Fig. 3.2b.

In Fig. 3.2a the data set is difficult to cluster. K-means, as our go-to-algorithm, would separate the clusters incorrectly. DipScaling measures the dip values of the axes - 0.07 and 0.02 respectively - and then rescales the axes in this range - [0, 0.07] and [0, 0.02]. This causes a drastically shifted data set, which k-means can cluster perfectly. The effect is as we desired: the features with high levels of information have now a higher influence in determining the clusters. The feature with high dip value is scaled in the range [0, 0.07], thus the centres also move in this range. Changes in the data points in this feature have more influence than in the other feature, which, due to its low dip value, is scaled in the range [0, 0.02]. With such a low range, the centre can only move by - at most - 0.02; a very small distance, compared to the other feature with a range of 0.07. The feature with a high dip value is now the feature which influences the final clustering results more, while the feature with small dip value has barely any impact. If, in the extreme, a feature were to be re-scaled into the range [0,0] it would not affect the clustering result at all. All values would be the same, thus, having no impact for the computed distances.

The effect of this re-scaling is surprisingly large. In Table 3.1 the effect of it in combination with k-means can be seen. A wide range of data sets can be better clustered after re-scaling them with DipScaling. Since the features where the clusters are better pronounced (i.e. the peaks are better separated) have a higher impact on the clustering, the boundaries can be easier found by k-means. This, of course, does not hold for all data sets. In various data sets, the shape of the clusters differs too much from the assumptions of k-means and, thus, it is not possible to properly cluster the data set with k-means, independent of the scaling. These data sets, nevertheless, can profit from DipScaling by applying a clustering algorithm with different assumptions than k-means. In [62], it is shown that many standard and state-of-the-art algorithms benefit from this re-scaling.

Table 3.1: K-means with and without DipScaling. Average value of 100 runs is given. Correct k is given. Better result in bold. Results measured in NMI [71]. The data sets are Fish, Iris, Prestige, User Knowledge, Seeds, BreastTissue, Wine, Leaf, Forrest Type, ProximaPhalangeTW, Plane, Swedish Leaf, Olive Oil, WDBC, Mice Protein, Leaf Texture.

Data set	FISH	IRIS	PRES	USER	SEED	BREA	WINE	LEAF
DipScaling	**0.35**	**0.82**	**0.68**	**0.45**	**0.73**	**0.53**	**0.73**	**0.69**
k-means	0.29	0.69	0.51	0.28	0.70	0.33	0.42	0.66
Data set	FORR	PROX	PLAN	SWED	OLIV	WDBC	MICE	LETE
DipScaling	**0.73**	**0.52**	**0.80**	**0.54**	**0.65**	**0.56**	**0.41**	**0.74**
k-means	0.68	0.46	0.78	0.51	0.56	0.42	0.31	0.70

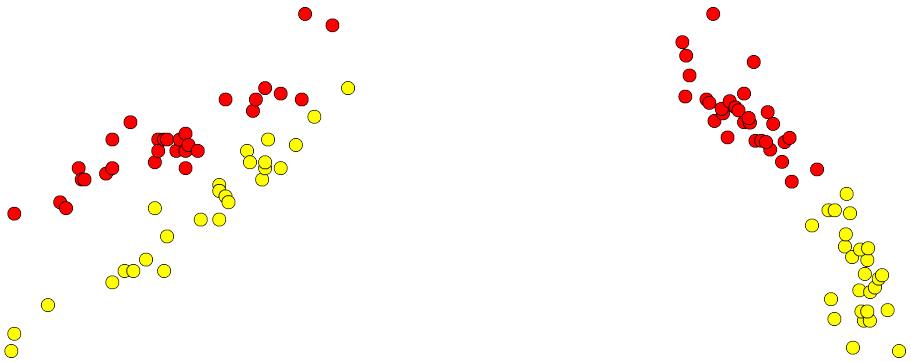
The same proof as for other normalisation methods shown in Chapter 2.4 also holds for DipScaling. It is continuous, invertible and its inverse is again invertible, i.e. a Dataset-Transformation. It reshapes the data set but does so in a rather harmless way. The clusters are stretched to make them more accessible to clustering algorithms, but their basic shape is left as is. The assumption is also that the clusters are axes-parallel. If a cluster is not axes-parallel, DipScaling is might not significantly improve the clustering result. This raised the question, how we could handle such a case.

3.2 DipTransformation

Clusters will not always be axes-parallel. In various data sets, e.g. the Whiteside-data set (see Fig. 3.3a), the clusters are visibly oriented in another direction than the axes. In such a case, DipScaling will have almost no effect at all, as the dip values of the axes are almost identical. Thus, we make DipScaling applicable, by taking a little detour.

The argument we follow is still the same as before. The clusters are stretched such that the most influential feature is not the one with the highest level of information relevant for clustering. The feature with the highest dip value is here at $\approx 45°$. Thus, ideally, the algorithm would re-scale the data set at this angle.

DipTransformation [63] (Appendix B) enables this generalisation of DipScaling by combining it with a rotation operator, based on the dip values of the axes. Technical details can be found in paper [63] or Appendix B. In short, the algorithm rotates the data set by a small angle when it finds interesting dip values and by a large angle, when the dip values are very similar. As an example, the Whiteside data set has its maximal dip value at an angle of roughly 45°. Thus, if the algorithm is close to this angle it rotates the data set only a little, thus, getting closer to the optimal angle. In contrast, the dip values of the axes in the original data set are very similar and of little interest. The DipTransformation applies a larger angle of rotation to quickly find more interesting dip values. Applying this combination of rotation and scaling operators leads to the transformation shown in Fig. 3.3b.



(a) The Whiteside-data set. Colours visualise the ground truth.

(b) The Whiteside-data set after applying DipTransformation.

Figure 3.3: Applying DipTransformation to the Whiteside-data set. The clusters, as can be seen from the ground truth visualised as colours, are now perfectly separated from each other.

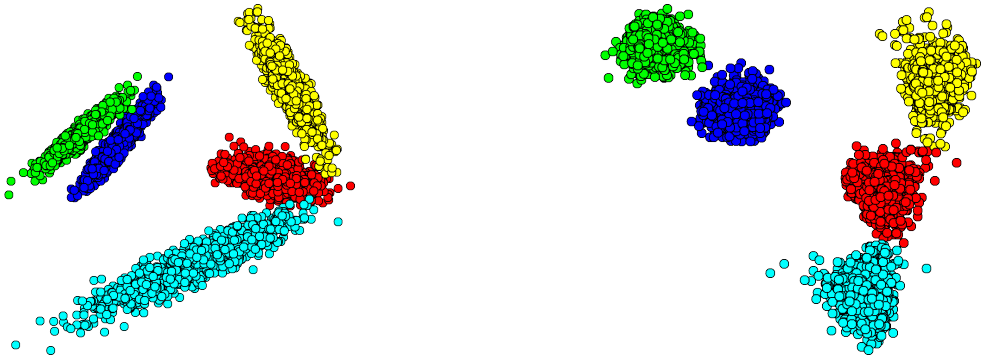
The data set is now very easy to cluster. The feature with the highest dip value is the one with the highest influence in clustering, as it is the one with the largest range.

In [62] we show that the operator is continuous as well as invertible. Beyond that, we proof that DipTransformation is a basis-transformation. Thus, it can be described by a non-singular matrix A. Since A is invertible, A^{-1} exists. Since the determinant of an inverse matrix is the inverse of the determinant of the matrix (i.e. $det(A) \cdot det(A^{-1}) = 1$) the determinant of A^{-1} is non-zero (else the determinant of A would be infinite). As having a non-zero determinant is equivalent to being non-singular, it follows that A^{-1} is a basis-transformation. Furthermore, it is also continuous, as every basis-transformation is a linear operator and every linear operator on a finite-dimensional space is continuous. Thus, we showed the following:

Theorem 2. The DipTransformation fulfils all requirements from Chapter 2 and is thus a **Dataset-Tranformation**.

The DipTransformation successfully generalises DipScaling to non-axes parallel clusters. Data sets that are only partially clusterable or not at all (e.g. Whiteside) can now be perfectly clustered. This holds for a wide range of data sets. DipTransformation in combination with k-means performs better than quite a few comparable methods and standalone clustering approaches. Furthermore, we also showed that clustering approaches besides k-means are compatible with DipTransformation and benefit from it. Details for these claims can be found in [62] and [63].

With DipTransformation, we see how much various clustering methods benefit from a transformed data set and how much a re-scaling of features influences the clustering result. This lead to two separate continuations. First, PCE, which extends this principle to arbitrarily oriented clusters and, second, DipExt, which extracts features with high dip values.



(a) A synthetic data set consisting of arbitrarily oriented clusters.

(b) The clusters are transformed into a Gaussian shape.

Figure 3.4: Applying PCE to a data set.

3.3 Principal Cluster Enhancement (PCE)

DipTransformation can re-scale features in any arbitrary direction, but it always re-scales the features as a whole. Consider the data set shown in Fig. 3.4. DipTransformation would not be capable of re-scaling the clusters such that they fit into the assumptions of k-means since there is simply no linear transformation that could properly separate the clusters from each other. All possible directions of projection would have overlapping clusters and clusters facing in different directions, such that re-scaling this direction would not force clusters into a simpler shape. Ideally, they will have the shape of a Gaussian bubble.

PCE [60] (Appendix E) tries to generalise the principle of DipTransformation, such that clusters can have arbitrary orientations. In the example shown in Fig. 3.4, the clusters do not fit into the Voronoi cell structure of k-means. The clusters are too elongated to fit into them as k-means assumes a somewhat "round" cluster. The goal of PCE is to re-shape the clusters, such that they fit into the assumptions of k-means. DipTransformation had a similar goal - by re-scaling the features the clusters become more equally scaled in all directions and thus more compact. PCE now tries to make this explicit and re-shapes the clusters into a Gaussian bubble. The algorithm achieves this by analysing the k-means clustering result and taking the temporary information reached by it as a starting point. It takes the found Voronoi cells, applies PCA to find the directions of maximal variance and projects the data points in the Voronoi cells into these directions to analyze them. A combination of the dip value and the variance of such a projection creates a value that tells us how much the Voronoi cell must be stretched or compressed in this direction. By applying this stretching and compression to the Voronoi cells, the algorithm can transform the clusters in an iterative process into the Gaussian bubble shape we are aiming for. Technical details and experimental evaluation can be found in paper [60] and Appendix E.

Important for us is how the data points are shifted in the Voronoi cells. In the paper,

we described the movement of a data point x by

$$PCE_i(x) = \sum_{j=1}^{k} \sum_{l=1}^{d} \left(x + \frac{1}{2} \Delta'(d(\mu_j, x)_l, \bar{\sigma}_{lj}) \right) \tag{3.1}$$

with k the number of clusters, d the dimensionality, μ_l the centre as found by k-means of Voronoi cell l, $d(\mu_j, x)_l$ is the distance of the projection of x onto the lth direction of PCA and $\bar{\sigma}_{lj}$ is the adapted variance in this direction. Δ' is the decay-factor of the re-scaling, which determines how the effect of the stretching/compressing behaves with distance to the centre of the cell μ. We proved in the paper that 3.1 is continuous, as long as Δ' is continuous. We discovered, though, that PCE is not invertible, because in cases of very small variances in the cluster, data points can be pushed into overlapping each other.

Assume two data points x and y in a single dimension, with $0 < x < y$. In the paper we chose a Δ' of

$$\Delta'(\mu, x) = \frac{d(\mu, x)}{e^{\frac{d(\mu,x)}{(10\bar{\sigma})}}} \left(\frac{1}{\bar{\sigma}} - 1 \right) \tag{3.2}$$

Thus, $PCE_i(x)$ is given by $x + \frac{1}{2} \frac{d(\mu,x)}{e^{d(\mu,x)/(10\bar{\sigma})}} \left(\frac{1}{\bar{\sigma}} - 1\right)$. We simplify this term by assuming, without loss of generality, that μ is the origin of the coordinate system. This changes the formula to $PCE_i(x) = x + \frac{1}{2} \frac{x}{e^{x/(10\bar{\sigma})}} \left(\frac{1}{\bar{\sigma}} - 1\right) = x + \frac{1}{2e^{x/10\sigma}} \left(\frac{1}{\bar{\sigma}} - 1\right)$. $PCE_i(x) < PCE_i(y)$ holds as long as the function PCE_i is strictly monotonous growing. The simplest way to prove the monotonicity of a function is via its derivative. The derivative, a slightly unwieldy equation, showed that its sign is not always positive or negative for certain values of $\bar{\sigma}$ (e.g. $\bar{\sigma} = 0.3$), proofing that the function is not monotonous. Since the function is not monotonous, there exist $\bar{x}, \bar{y}$ with $\bar{x} \neq \bar{y}$, but $PCE_i(\bar{x}) = PCE_i(\bar{y})$. Thus, the function is not invertible.

PCE in the version as published is not a Dataset-Transformation, as it is not invertible. The effect of this, as argued, is that it forces some decisions on the algorithm. We do not wish for this to happen, thus, we tried to find a version of PCE, which fulfils the requirements of a Dataset-Transformation and performs similarly on data sets. Various versions of Δ have been tried, among others one based on the logarithm. In all of them, we have found that at least one set of measure 0 is possibly mapped onto the same data points. This means, that an infinitely small area violates the invertibility. The chance of a data point being in this area converges to 0. Thus, while the requirements are technically violated, the reason for this specific requirement - forcing decisions on the clustering algorithm - is not given. The same also holds for the version presented here. There is a possibility that a clustering decision is enforced (via mapping two data points onto the same position), but the probability of this happening is infinitesimally small.

3.4 DipExt

DipScaling rescales every feature into the range $[0, d_i]$, where d_i is the dip value of the feature. DipTransformation ensures that this is not restricted to axis-parallel features. DipExt [66] (Appendix F) extends this principle into a Feature Extraction method by removing all features with a dip value below a certain threshold. A feature with a very small dip value will have barely any influence in clustering, thus, it might as well be removed. Only features with high dip values are kept and form a concentrated version of the data set.

One of the main differences to DipTransformation is how the features with high dip values are found. In DipTransformation, the search was based on a rotation-approach. DipExt changes this to a Gradient descent strategy, based on the differentiability of the dip test (proven in [44]). The advantage of this change in the search strategy is, for one, the runtime (changes from quadratic to quasilinear) and the precision of the search. As can be seen in the paper, DipExt is exceedingly competent at finding the maximal possible dip value in the data set.

DipExt reduces the dimensionality, thus, there is some loss of information connected to it. Although it is possible that some clusters can be better separated in the original data set, this does not seem to be the case according to the experimental evaluation. The advantages of the reduced dimensionality, i.e. the smaller effect of the curse of dimensionality, as well as only removing features with small levels of information, counter effects the loss in information enough, such that data sets can now be clustered better than before (see the paper for experimental results). The difference in clustering quality is due to the cluster-relevant information being now presented in a very compact and easily accessible form, which leads to massive improvements in the performance of k-means, but also various other clustering algorithms.

The drawback in this reduced version of the data set, is, of course, the drawbacks outlined in Chapter 2. A dimensionality reduction cannot be undone as it is not invertible. Thus, there are some decisions which are forced upon the algorithm. This, for one, means that the algorithm is not a Dataset-Transformation, as no dimensionality reduction is. Furthermore, as there is some information missing for the algorithm applied to the concentrated data set, some possible clustering results might no longer be available. Taking some of the freedom of decision making from the algorithm, though, seems to be advantageous in this context. Features with a very small dip value are removed from the data set and only those with high levels of structure remain. This leads to a data set, which does not have all information contained in the original data set, but the information it does have is far easier to access. As can be seen in the paper, a wide variety of clustering algorithms perform better on this concentrated version of the data set. The advantage of all decisions being possible is that a very wide range of data sets can be processed with a certain method. Fewer possible decisions mean fewer data sets will be suitable for this method, but the data sets that are suitable for this method will be better processed by it, as fewer decisions also mean fewer wrong decisions. DipExt removes features with very small amounts of information as unimportant.

A decision which proved correct for many data sets and algorithms as can be seen in the experiments.

3.5 DipInit

With DipExt we left the framework of Dataset-Transformations. Although data points change their position, it does not fulfil the requirements as listed in Chapter 2. With DipInit [66] (Appendix F) we move towards Support methods for clustering, like X-means or k-means++. DipInit does not change the position of data points, as a Dataset-Transformation does, but supports one of the already existing algorithms, in this case k-means, with one of its assumptions. K-means continues deterministic after the data points for its initialisation are found. Finding these initial data points, though, is not straightforward and various strategies have been proposed. Random initialisation and the aforementioned k-means++ are the best known and most often used. We have seen how useful features with higher dip values can be and how much their preferential treatment can improve clustering. With DipInit, we apply this insight to an initialisation strategy for k-means.

The feature with the highest dip value is clustered first by splitting it into k parts with equally many data points. Features with lower dip values are added over time, according to the descending dip value, and the clustering process continues. Each time a feature is added, k-means resumes with the "old" centres. By clustering the features with high dip values first, they are more influential in determining the clustering result. They determine the rough shape of the clusters and features which are added later, only change smaller details of the clustering. This principle ensures DipInit putting higher importance on the features with high dip values, while those with smaller values have a smaller influence on the final clustering result. DipInit is highly compatible with the already presented methods: both DipExt and DipTransformation improve in combination with DipInit. The principle of DipInit, furthermore, ensures that k-means is completely deterministic. Most initialisation strategies contain at least some arbitrary aspects, thus, DipInit is one of the few exceptions to this.

The details and experimental evaluation of this support method can be found in Appendix F.

3.6 KMN

KMN [65] (Appendix A) falls also into the category of support methods specifically targeted at k-means. K-means creates Voronoi cells by splitting the data set into partitions, as each data point is assigned to the nearest centre. This often works very well but fails in the case of outliers, since k-means does not have a concept of outliers. Every data point, including outliers, is simply assigned to the closest centre and thus, becomes part of a cluster.

Extending the k-means framework such that outliers are separated from the clusters, has, until recently, been a somewhat neglected problem. The methods, that have been

proposed, like Neo-k-Means [73] and KMOR [28], all ask for additional parameters. K-means-- [9] even asks for the explicit number of outliers. While we cannot state, that KMN comes completely without additional assumptions (all methods have assumptions), it was at the time of the publication the first k-means extension to deal with outliers, without needing additional parameters.

KMN computes, based on the Minimum Description Length (MDL) principle, whether data points belong to a cluster or if they should be classified as outliers. The MDL principle is related to Occam's Razor, as it looks for the simplest model in terms of encoding the model. As Grünwald formulates it: "MDL embodies a form of Occam's Razor" [30]. MDL is a model selection principle, which allows us to find the model which best suits the data set, i.e. the model which allows for the shortest description of the data. By applying the MDL-principle it is possible to avoid adding any additional parameters and to falsify the statement by Chawla and Gionis: "all existing outlier-detection methods use one or more parameters" [9]. KMN is capable of assigning data points as outliers and separating them from the regular data points, while still staying in the k-means framework. It opens additional smaller Voronoi cells and if the MDL principle deemed this re-partitioning meaningful, the data points in them are labelled as outliers. Technical details and experimental evaluation can be found in Appendix A.

Here we come full circle with KMN. The intention, in the beginning, was to minimise the effects of assumptions by using Occam's razor and replace complicated assumptions with simple ones. On this basis, we created various methods for which experimental validation showed us that these replacements could be justified. With KMN we applied the MDL-principle, a formalised version of Occam's Razor, to replace the assumptions of an outlier-free data set. Instead of simply assuming that the data contains no outliers at all, the MDL-principle is used to test and locate the data points which are outliers.

Chapter 4

Contributions, Conclusion and Outlook

Contributions

We presented a framework for Dataset-Transformations and argued which properties such a transformation should have. This framework might pose as a possible orientation for future work and guide further transformation-based algorithms. While directly applying a clustering method onto a data set can succeed, it is often advantageous to apply a pre-processing step before it. In [21] pre-processing is stated as an explicit step in the KDD process to help with data mining. Currently, though, most such pre-processing steps are either with minimal impact or at least seen to be of minimal impact. In many publications, it is either not even mentioned or hidden at the end of a paragraph, if the tested data sets are, for example, normalised or pre-processed in any other way, despite the significant impact such methods can have.

The intention here is to go a step further and pre-process the data, such that clusters become more visible. Many pre-processing steps follow a rather narrow goal of removing errors from the data set, but not improving on it. Some methods remove outliers so that the regular clustering process can proceed (e.g. LOF [6]), some try to identify duplicate data points (e.g. [18]), some try to balance the influence of features for clustering (e.g. Z-transformation), but they try to compensate for blemishes, not enhance the data set. Our concept of data set transformations pro-actively aims to improve the data set from its current state to a clearer one. We use the large influence such pre-processing methods can have and which is often overlooked to explicitly make the analysis of the data easier.

We started this work with DipScaling. Although it is still a regular normalisation method by itself, it does not aim to give all features the same relevance for clustering as Z-transformation and most others do. It intends to guide clustering algorithms by giving priority to structure-rich features such that they have more influence in the clustering process. It fulfils all Dataset-Transformation requirements. So does its extension, Dip-Transformation, which generalises this re-scaling concept to arbitrarily oriented features. Following this approach, subsequent developments split in two directions. PCE extended

the re-scaling of features to re-shaping arbitrarily oriented clusters into Gaussian bubbles. PCE still fulfils some of the requirements of a Dataset-Transformation, but changes in the specifics of the algorithm are necessary to ensure that all requirements were met. DipExt extends DipTransformation into a Feature Extraction method and changes the search strategy to a Gradient descent-approach. In addition to re-scaling features, it also removes the irrelevant features. It is not a Dataset-Transformation, as no dimensionality method can be. Based on the same hypothesis as the mentioned methods, we also presented DipInit, which proposes an initialisation strategy for k-means, which puts higher importance on structure-rich features. It is, just like KMN, which finds and removes outliers during the k-means clustering process, not a Dataset-Transformation. Both these methods fall into the broader category of general Support methods for clustering. They do not change the position of data points but overcome part of the disadvantages that k-means has. Their focus is not on the assumptions of the generating process, but on the assumptions of its clustering process.

Conclusion

All these methods lessen the effect of various assumptions that are made by k-means. K-means is, just like all other Data Mining-algorithms, based on various implicit and explicit assumptions about the data. These methods intend to either reduce or negate the impact of these assumptions. The drawback of these methods, though, is that they are also based on assumptions, for example, the applicability of the dip test to verify whether a feature contains information relevant for clustering.

We have argued in Chapter 1.4 that some assumptions are more reasonable because they are less complicated and fit better into our understanding of the world. According to Occam's Razor, these simple assumptions are more likely to hold compared to complicated ones. Support methods now replace some of the complicated, difficult-to-verify assumptions with simpler ones that are better compatible with Occam's Razor. As an example, KMN removes the assumption of an outlier-free data set and replaces it with the applicability of the MDL-principle; PCE removes the assumption that the clusters are convex/Gaussian and substitutes it with the assumption that the dip test can be applied. Assumptions that cannot be verified in any way or are excessive are exchanged with simple assumptions that reflect our experience of the world. The presented algorithms all apply this switch from challenging to simple assumptions. Data sets that previously could not be clustered properly have a better chance of fitting into the framework of the algorithm and being clustered properly through this exchange of assumptions.

We can, of course, not prove that this is the case, especially not for all types of data sets. Following the No-Free-Lunch theorem, all clustering methods fit some data sets, but not all. This also holds for all Data Mining methods and, naturally, also those presented here. Following the experimental evaluations in the various papers, though, we can state that a surprisingly wide range of data sets are easier to cluster by k-means and other clustering approaches after the methods presented here are applied to them.

This allows us to conclude - again using Occam's razor - that the data sets are improved and our argument holds. We weakened the effect of the already existing assumptions contained with k-means and created a pre-processing step which enhances the data set and thus simplifies the clustering step in the KDD process.

Outlook

We have seen in the earlier chapters, that quite a few methods are listed, which were created during and due to this PhD. These methods cover a wide range of possible applications and lessen the assumptions, which are implicitly or explicitly made by k-means. There are, of course, a few loose threads still left and some of them are still pursued as research. PCE extended DipTransformation, such that clusters can have an arbitrary orientation, but they still are expected to follow a roughly convex shape. We are currently working at relaxing this requirement, by combining the intent of such a Dataset-Transformation with Neural Networks. Since a Neural Network can, in theory, approximate all possible functions, it could also approximate one which reshapes a specific cluster into a Gaussian bubble, irrelevant of how convoluted the shape of the cluster is.

At first, having a method that is capable of transforming all types of data sets into perfect Gaussian bubbles, sounds like the epitome of pre-processing for clustering, but there is a drawback involved with such comprehensive methods. The wider the suitable data sets are - in this case potentially all of them - the more the algorithm has to decide what kind of data set it is facing. The wider the range of possible data sets, the more extensive and encompassing the range of decisions it has to make. For example, if a method assumes the user provides the number of clusters, the algorithm does not have to decide on this by itself. Thus, it cannot make an error in deciding the number of clusters, and a possible source of errors is eliminated. An algorithm which can deal with all types of data sets will have to decide how to deal with the specific data set it is handling. If it knew exactly what the properties of this data set are, it would not need to decide and, thus, could not make errors. Since all data sets are possible, it needs to decide everything and can make almost all errors. Fewer data sets suitable for a method means fewer decisions and thus potentially fewer wrong decisions.

A pre-processing/support technique will always, like all Data Mining algorithms, face this conflict. It is either suited to a wide range of data sets and thus risks being more prone to errors, or suited to a smaller range of data sets, which it is more likely to deal with properly. This conflict is reinforced as it is often not clear what type of data set a specific data set is. With the presented methods, we tried to walk a middle ground in this conflict of suitable-to-all-data-sets and working-well-on-fewer-data-sets. We have seen in the experimental evaluation of the presented techniques, that their performance is impressive on a wide range of data sets. Thus, while there is no guarantee on these methods working properly on all data sets, we are quite satisfied with our success in walking this narrow line between applicability and theoretical considerations.

We have seen and argued - and considering the No-Free-Lunch-Theorem - it is not possible to create a technique suitable for all data sets. All progress here is always

preliminary with methods being applicable to some data sets, but not others. General methods such as k-means, which can be applied to a wide range of problems, are constantly analysed and seldom improved. They are the first steps in the exploration of a new problem that has been ignored or has only recently emerged. If this new type of data set becomes relevant enough, special types of algorithms will be created for it, focusing on the characteristics of this type and replacing the more general algorithms. The advancement of data mining is in constant flux, with new algorithms constantly being created and older ones improved and expanded. No algorithm that is perfectly suitable for all data sets, or even just one type of data set, can ever be created. A compromise always has to be made, which means it is impossible to "complete" this field of study. Nevertheless, or precisely because of this, we accept the impossibility of our endeavour to solve clustering and feel confident to draw the line here.

Bibliography

[1] Arthur, D., Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA, 2007.

[2] Andrews, J.T., Klein, A.J., Jenner, K.A., Jennings, A.E., Campbell, C., *The variability of Baffin Bay seafloor sediment mineralogy: the identification of discrete glacial sediment sources and application to Late Quaternary downcore analysis*, Canadian Journal of Earth Sciences, 2018.

[3] Behzadi S., Schelling, B., Plant, C., *ITGH: Information-theoretic Granger Causal Inference on Heterogeneous Data*, PAKDD, 2020.

[4] Böhm, C., Perdacher, M., Plant, C., *Multi-core K-means*, SDM, 2017.

[5] Böhm, C., Plant, C., Shao, J., Yang, Q., *Clustering by synchronization*, KDD, 2010.

[6] Breunig, M., Kriegel, H., Ng, R., Sander, J., *LOF: Identifying Density-based Local Outliers*, SIGMOD, 2000.

[7] Celebi, M., Kingravi, H., Vela, P., *A Comparative Study of Efficient initialisation Methods for the K-Means Clustering Algorithm*, Expert Syst. Appl., 2013.

[8] Chamalis, T., Likas, A., *The projected dip-means clustering algorithm*, Hellenic Conference on Artificial Intelligence, 2018.

[9] Chawla, S., Gionis, A., *k-means--: A unified approach to clustering and outlier detection*, ICDM, 2013.

[10] Chronis, P., Athanasiou, S., Skiadopoulos, S., *Automatic Clustering by Detecting Significant Density Dips in Multiple Dimensions*, ICDM, 2019.

[11] Chung, Y., Weng, W., Tong, S., Glass, J., *Unsupervised cross-modal alignment of speech and text embedding spaces*, NIPS, 2018.

[12] Clifton, C., *Data Mining*, Encyclopædia Britannica, 2010. Retrieved 2010-12-09.

[13] Dempster, A. P., Laird, N. M., Rubin, D. B., *Maximum-Likelihood from incomplete data via the EM algorithm*, Journal of the Royal Statistical Society, 1977.

[14] Devadoss, S., O'Rourke, J., *Discrete and computational geometry*, Princeton University Press, 2011.

[15] Dua, D., Karra Taniskidou, E., *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2018.

[16] Dunn, J., *A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters*, 1973.

[17] Drineas, P., Frieze, A., Kannan, R., Vempala, S., Vinay, V., *Clustering large graphs via the singular value decomposition*, 2004.

[18] Elmagarmid, A., Ipeirotis, P., Verykios, V., *Duplicate Record Detection: A Survey*, TKDE, 2007.

[19] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *A density-based algorithm for discovering clusters in large spatial databases with noise*, KDD, 1996.

[20] Estivill-Castro, V., *Why so many clustering algorithms: a position paper*, SIGKDD Explorations, 2002.

[21] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., *From data mining to knowledge discovery in databases*, AI magazine, 1996.

[22] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., *Knowledge Discovery and Data Mining: Towards a Unifying Framework*, KDD, 1996.

[23] Feng, Y., Hamerly, G., *PG-means: learning the number of clusters in data*, NIPS, 2006.

[24] Firmin, Sydney, *There is No Free Lunch in Data Science*, KDnuggets, September 2019. `https://www.kdnuggets.com/2019/09/no-free-lunch-data-science.html`. Retrieved 1.7.2020.

[25] Forgy, E., *Cluster analysis of multivariate data: efficiency versus interpretability of classifications*, Biometrics, 1965.

[26] Fournier-Viger, P., *An Introduction to Data Mining*. `http://data-mining.philippe-fournier-viger.com/introduction-data-mining/`. Retrieved 14.5.2020.

[27] Fränti, P., Sieranoja, S., *How much can k-means be improved by using better initialization and repeats?*, Pattern Recognition, 2019.

[28] Gan, G., Kwok-Po Ng M., *k-means clustering with outlier removal*, Pattern Recognition Letters, 2017.

[29] Goebl, S., He, X., Plant, C., Böhm, C., *Finding the Optimal Subspace for Clustering*, ICDM, 2014.

[30] Grünwald, P., *A Tutorial Introduction to the Minimum Description Length Principle*, In: Grünwald, P., Myung, I., Pitt, M., *Advances in minimum description length: Theory and applications*, MIT press, 2005.

[31] Guo, X., Gao, L., Liu, X., Yin, J., *Improved Deep Embedded Clustering with Local Structure Preservation*, IJCAI, 2017.

[32] Hall, M., et al, *The WEKA Data Mining Software: An Update*, SIGKDD Explorations, 2009.

[33] Hartigan, J. A., Hartigan, P. M., *The Dip Test of Unimodality*, The Annals of Statistics, 1985.

[34] Hartigan, J., Wong, M., *Algorithm AS 136: A k-Means Clustering Algorithm*, Journal of the Royal Statistical Society, 1979.

[35] Hume, D., *A Treatise of Human Nature*, 1739f.

[36] Jain, A., *Data clustering: 50 years beyond K-means*, Pattern recognition letters, 2010.

[37] Jain, A., Dubes, R., *Algorithms for clustering data*, Prentice-Hall, 1988.

[38] Jäkel, F., Schölkopf, B., Wichmann, F., *Similarity, kernels, and the triangle inequality*, Journal of Mathematical Psychology, 2008.

[39] Jing, L., Ng M.K., Huang,J. Z., *An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data*, TKDE, 2007.

[40] Kalogeratos, A.,Likas, A., *Dip-means: an incremental clustering method for estimating the number of clusters*, NIPS, 2012.

[41] Karypis, M., Kumar, V., Steinbach, M., *A comparison of document clustering techniques*, KDD, 2000. Text Mining Workshop.

[42] Kleinberg, J., *An impossibility theorem for clustering*, NIPS, 2002.

[43] Kotsiantis, S., Zaharakis, I., Pintelas, P., *Supervised machine learning: A review of classification techniques*, Emerging artificial intelligence applications in computer engineering, 2007.

[44] Krause, A., Liebscher, V., *Multimodal projection pursuit using the dip statistic*, Preprint-Reihe Mathematik, 2005.

[45] Kriegel, H. P., Kröger, P., Zimek, A., *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering*, TKDD, 2009.

[46] Lloyd, S., *Least square quantization in PCM*, Bell Telephone Laboratories Paper, 1957.

[47] Lloyd, S., *Least squares quantization in PCM*, Transactions on Information Theory, 1982.

[48] Maaten, L., Hinton, G., *Visualizing data using t-SNE.* Journal of machine learning research, 2008.

[49] MacQueen, J. B., *Some methods for classification and analysis of multivariate observations*, Berkeley Symposium on Math. Stat. and Prob., 1967.

[50] McInnes, L., Healy, J., Melville, J., *Uniform manifold approximation and projection for dimension reduction*, arXiv:1802.03426, 2018.

[51] Maurus, S., Plant, C., *Skinny-dip: Clustering in a Sea of Noise*, KDD, 2016.

[52] Mautz, D., Ye, W., Plant, C., Böhm, C., *Towards an Optimal Subspace for K-means*, KDD, 2017.

[53] Ng, A., Jordan, M., Weiss, Y., *On spectral clustering: Analysis and an algorithm*, NIPS, 2002.

[54] Pearson, K., *On lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 1901.

[55] Pedregosa, F., et al, *Scikit-learn: Machine Learning in Python*, JMLR, 2011.

[56] Pelleg, D., Moore, A., *X-means: Extending K-means with Efficient Estimation of the Number of Clusters*, ICML, 2000.

[57] Popper, K., *Conjectures and Refutations - The growth of scientific knowledge*, Routledge & Kegan Paul, 1963.

[58] Phyu, T.,*Survey of classification techniques in data mining*, International Multi-Conference of Engineers and Computer Scientists, 2009.

[59] Schelling, B., Bauer, L., Plant, C., *Utilizing Structure-rich Features to improve Clustering*, ECML-PKDD, 2020.

[60] Schelling, B., Miklautz, L., Plant, C., *Non-linear Cluster Enhancement: Forcing Clusters into a compact shape*, ECAI, 2020.

[61] Schelling, B., Plant, C., *Clustering with the Levy Walk: "Hunting" for Clusters*, ICDMW, 2016. PhD Forum.

[62] Schelling, B., Plant, C., *Dataset-Transformation: improving clustering by enhancing the structure with DipScaling and DipTransformation*, KAIS, 2019.

[63] Schelling, B., Plant, C., *DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering*, ICDM, 2018.

[64] Schelling, B., Plant, C., *DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering (Extended Abstract)*, GI-Jahrestagung Informatik, 2019.

[65] Schelling, B., Plant, C., *KMN - Removing Noise from K-Means Clustering Results*, DAWAK, 2018.

[66] Schelling, B., Sluiter, G., Plant, C., *RandomLink - Avoiding Linkage-Effects by employing Random Effects for Clustering*, DEXA, 2020.

[67] Schölkopf, B., Smola, A., Müller, K., *Nonlinear component analysis as a kernel eigenvalue problem*, Neural computation, 1998.

[68] Schubert, E., Zimek, A., *ELKI: A large open-source library for data analysis*, 2019.

[69] Sibson, R., *SLINK: an optimally efficient algorithm for the single-link cluster method*, The Computer Journal, 1973.

[70] Siffer, A., Fouque, P. A., Termier, A., Largouet, C., *Are your data gathered?*, KDD, 2018.

[71] Vinh, N. X., Bailey, J., *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance*, JMLR, 2011.

[72] Von Luxburg, U., Williamson, R., Guyon, I., *Clustering: Science or art?*, ICML, 2012. Workshop on Unsupervised and Transfer Learning.

[73] Wangh, J., Dhillon, I., Gleich, D., *Non-exhaustive, Overlapping k-means*, SDM, 2015.

[74] Wolpert, D., *The Lack of A Priori Distinctions Between Learning Algorithms*, Neural Computation, 1996.

[75] Wolpert, D., Macready, W, *No Free Lunch Theorems for Optimization*, Transactions on Evolutionary Computation, 1997.

[76] Wu, H., Gu, X., *Max-Pooling Dropout for Regularization of Convolutional Neural Networks*, ICONIP, 2015.

[77] Wu, X., et al, *Top 10 algorithms in data mining*, KAIS, 2008.

[78] Xie, J., Girshick, R., Farhadi, A., *Unsupervised deep embedding for clustering analysis*, ICML, 2016.

[79] Xu, R., Wunsch, D., *Clustering*, Wiley, 2009.

[80] Xu, S., Bie, R., Li, L., Yang, Y., *DP-Dip: A skinny method for estimating the number and center of clusters*, Procedia Computer Science, 2018.

[81] Yang, B., Fu, X., Sidiropoulos, N., *Learning From Hidden Traits: Joint Factor Analysis and Latent Clustering*, IEEE Trans. Signal Process, 2017.

[82] Yang, B., Fu, X., Sidiropoulos, N., Hong, M., *Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering*, ICML, 2017.

Appendix A

Paper A: KMN - Removing Noise from K-Means Clustering Results

KMN - Removing Noise from K-Means Clustering Results

Benjamin Schelling and Claudia Plant

University of Vienna

Abstract. K-Means is one of the most important data mining techniques for scientists who want to analyze their data. But K-Means has the disadvantage that it is unable to handle noise data points. This paper proposes a technique that can be applied to the k-Means Clustering result to exclude noise data points. We refer to it as KMN (short for K-Means with Noise). This technique is compatible with the different strategies to initialize k-Means and determine the number of clusters. Moreover, it is completely parameter-free. The technique has been tested on artificial and real data sets to demonstrate its performance in comparison with other noise-excluding techniques for k-Means.

1 Introduction

A scientist who has not had much contact with data mining will use the simplest algorithms when he decides to use data mining techniques. The simplest and best known is probably k-Means [12]. More refined techniques, that offer the possibility of achieving better results are likely to be applied at a later stage, once the scientist has become familiar with the automatic labelling of the data and has learned to appreciate the additional information that such techniques might yield. K-Means is something like a gateway to data mining techniques. It has many advantages that predestine it for this purpose: Its simplicity, the comparatively good results and its runtime. But k-Means also has some disadvantages that are not to be neglected: The initialization that determines which (local) optimum the algorithm converges to, the need to set the k parameter and its inability to handle noise. This can be seen in Figure 1. K-Means will add each data point to a cluster, since the possibility, that a data point is a noise point is simply not supported in the algorithm. The first two problems, initialization and setting k, were examined in detail and various (sometimes very capable) strategies have been proposed, the last problem however received less recognition. This may be partly due to the fact that publicly available data sets rarely contain noise, which is why k-Means-based techniques that find and characterize noise did not seem so important at first. Recently, however, there have been more and more techniques that take noise into account when clustering.

1.1 Related Work

Clustering in the presence of noise is a classic field of research in data mining. Many techniques like DBSCAN [6] have been suggested and studied extensively.

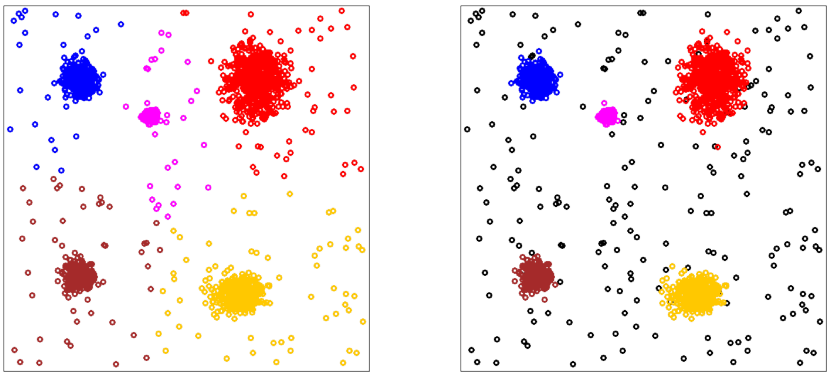


Fig. 1. A typical clustering result of k-Means, if the value of k is chosen correctly, and how the clustering actually should look like.

However, few of them are based on k-Means, hence will not be a likely first choice for a new user of data mining techniques. The focus here is exclusively on techniques which adapt k-Means to noisy data sets and thus offer the possibility to stay within the framework of k-Means while they are still dealing with noisy data sets.

The best known of these clustering-in-the-presence-of-noise techniques based on k-Means is k-Means-- from Chawla et al. [5]. The problem is that it asks for the number of outliers as a parameter for the functionality of the technique. Chawla et al. stated in their paper that "all existing outlier-detection methods use one or more parameters". This seems to be largely correct for the outlier clustering techniques that are based on k-Means. Similarly, the algorithm KMOR proposed by Gan et al. [7] asks for two additional parameters, one of which is the maximum number of outliers. The algorithm ODC [1] has the "difference" between outliers and real data points as a parameter, while Neo-k-Means [16] asks for two parameters α and β, which are also related to the assignment of data points to a cluster, i. e. the number of outliers.

The aim of this work is to find a way to remove noise points from the clusters without additional parameters. Having to set parameters will most likely prevent inexperienced users from using data mining techniques, hence, the ideal clustering technique is one completely without parameters. One could argue that the other problems of k-Means might have already detered possible users, but these problems have many solutions, many of which are implemented in publicly available software. It is easy to add the option of, say, X-Means [14] to estimate the value of k and k-Means++ [2] to achieve a good initialization. The ideal solution would be to add the option to remove noise too, without setting additional parameters.

1.2 Contribution

We present here a k-Means based technique that adapts and expands k-Means to noisy data sets.

- The technique presented in this work, KMN, can efficiently remove noise from k-Means clustering results without additional parameters.
- It does so, while being deterministic. For a k-Means clustering result, the excluded noise points are always the same.
- KMN offers a higher independence from the chosen k for k-Means. This work demonstrates in section 3 how KMN fares for wrongly chosen values of k and shows the resilience it gives in relation to these values.
- It is fully compatible with other k-Means-based techniques that aim to improve k-Means like X-Means and k-Means++.

2 The algorithm

The main intention of this work is to present an approach to exclude noise from the clustering result of k-Means, to which we refer to as KMN. The algorithm starts with a k-Means clustering result and tries to locate the areas where noise is prevalent. K-Means follows a centre-oriented approach, in which data points are assigned to the nearest centre. The centre is then updated as the mean of the assigned data points. This means that the closer a datapoint is to a centre, the more likely it is to be assigned correctly (provided that k-Means is a fitting technique for the dataset and k is correctly selected). Or in other words: the closer a data point is to the centre, the more likely is it correctly assigned. The further away, the more likely is it that it should be assigned to another centre or regarded as noise.

The search for noise should therefore begin at the locations furthest from the cluster centres. The clusters in k-Means are Voronoi cells and between two neighbouring Voronoi cells there is a hyperplane that separates them. At the intersections of these hyperplanes one will find the point farthest from the centres, as shown in Figure 2. Let us call this point m.

In an d-dimensional data space $d+1$ voronoi cells determine such an intersection point. This point m is basically defined as the point in the data space, where $||v_j - m|| = r$, $j \in \{1, 2, \ldots, d+1\}$, holds; v_j are the centres of the voronoi cells, i.e. the centres that k-Means finds, r is the distance of m to the centres of the voronoi cells. If one of the voronoi cell centres had a distance to m unequal r, m would be assigned to the voronoi cell with the smallest distance.

When these intersections are found, the assumption is valid that the area of these intersections should be considered to contain noise data points. Retaining the spirit of k-Means, the algorithm then simply opens a new Voronoi cell at the intersection, but one in which all data points within its boundaries are regarded as noise (see Figure 2.2). It may not always be a good decision to open such a noise voronoi cell, because it is possible that such a noise voronoi cell contains data points and not (only) noise points. It is necessary to have a criterion to

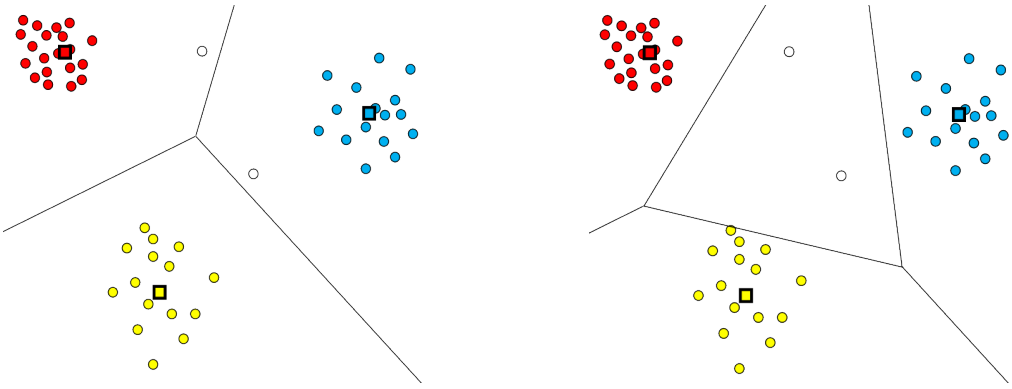


Fig. 2. Three simple clusters and two outliers. The squares are the centres of the clusters as found by k-Means which determine the voronoi cell walls. This shows, if k-Means is a suitable technique, that the data points near the centres are correctly assigned, while the more distant data points are more likely to be wrongly assigned. When a new Voronoi cell is opened at the intersection of Voronoi cells, the noise data points are separated from the clusters.

decide whether or not to open such a noise voronoi cell. KMN uses the principle of Minimum Description Length (MDL).

MDL is a well established principle in the Data Mining community and is used in various technique like X-Means [14]. The basic assumption behind MDL is that the coding cost for a clustering result depends on how good the clustering is. The coding cost is basically an estimate of how much memory is needed to encode the clustering. If the clustering is good, only little memory is needed, but if the clustering is far off, i. e. the model used to encode the data is not fitting, the encoding requires a lot of memory.

If the criterion is applied and the Voronoi cells are either retained or discarded, the next iteration can begin. We see in Figure 2.2 that there are new Voronoi cell intersections and the same steps can be taken as before. Before this happens, the centres of the clusters are updated. The process is iterated until no more intersection improve the clustering, according to the criterion.

This is, very briefly, the procedure of the algorithm. Let us now elaborate on that.

2.1 Finding the Voronoi Intersections

There are several ways to find the intersections of the Voronoi cells. The most obvious way would be to find m using geometric calculations. In this approach one would to work with the equation $||v_j - m|| = r$ directly. This equation is the formula for a sphere with radius r and middlepoint m. The $d+1$ points v_j are given and therefore uniquely determine the centre m. To calculate m, this would entail inverting a $d+1 \times d+1$-matrix, which would mean computations in the order of $O((d+1)^3)$. While this might be acceptable, the difficulty lies with the

choice of the v_j. If there are k centres of k-Means in a d-dimensional data space, without knowing the adjacencies of the centres, one would have to compute all $\binom{k}{d+1}$ possible combinations to find the correct combinations of voronoi cells. This is comparatively expensive and should be avoided.

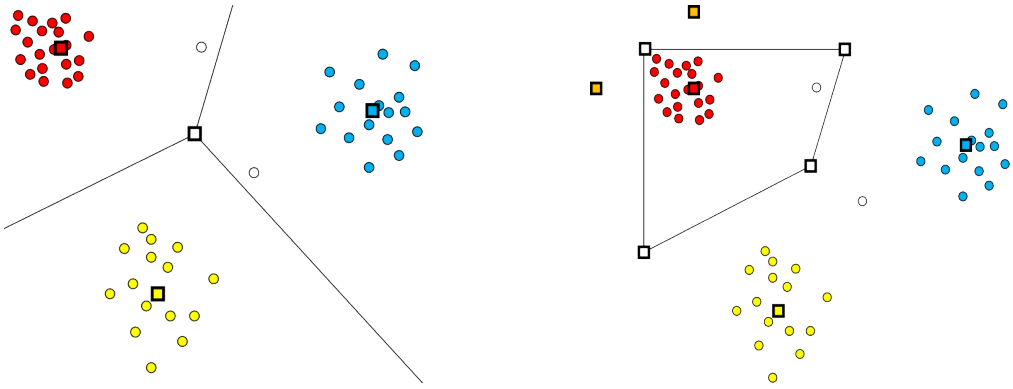


Fig. 3. Convex Hull and intersection. The three clusters determine one voronoi cell intersection, which is represented by the white square. The centres of the other clusters as well as the projections of its own centre determine a bounded polytope, i.e. the voronoi cell.

This technique uses the Avis-Fukuda algorithm [3], which is specialized in finding Voronoi intersections. More precisely, Avis-Fukuda takes the centre of each Voronoi cell, i. e. the centres that k-Means finds, as input and calculates the intersection points of these Voronoi cells. The algorithm is based on a linear optimization approach. It works according to the following principle: it starts with a centre of a Voronoi cell and the other centres that bound it. This forms a convex hull around the centre, a polyhedron, with the corners of this hull being the intersections we are looking for. It calculates the nearest vertex with a linear optimization approach and then follows the beams of the convex hull along to find the other vertices. Avis-Fukuda follows the optimal path and finds all nodes, i. e. all Voronoi intersections. The algorithm uses "Bland's rule" to ensure that the path is the optimal one and that all vertices are found. In this way, all corner points of a convex hull, i. e. all Voronoi intersections, are found.

A very simple example of this is illustrated in Figure 3. The point of intersection between the three Voronoi cells is easy to find, but the Voronoi cell is unbounded. Therefore, we project the centre for all Voronoi cells beyond the data set boundaries, so that all clusters have a limited convex hull defined by the centres found by k-Means and the added projections. The Avis Fukuda algorithm can then be used to find the vertices (represented in Figure 3.2 as white squares). The algorithm continues by finding such an vertex and then moving on to the others. In a two-dimensional environment like this, it is rather straightforward. After the first step, there is always only one direction in which the algorithm can

advance. It follows the rays of the convex hull until it reconnects with the first vertex. The Avis-Fukuda algorithm has then found all nodes, i. e. all Voronoi intersections for the cluster.

The Avis-Fukuda algorithm is estimated to have a runtime of $\mathcal{O}(k \cdot d \cdot v)$, where v is the number of vertices. Since we apply the algorithm for each of the k voronoi cells, we have a runtime of $k \cdot \mathcal{O}(k \cdot d \cdot v)$.

2.2 The MDL-Criterion

The algorithm has found the possible intersections of the Voronoi cells. However, the question now arises as to whether the algorithm should open new Voronoi noise cells or not. MDL assumes that lower coding costs imply a better clustering. Thus, if the new Voronoi noise cell reduces the total coding costs, the algorithm keeps the new Voronoi noise cell. The coding costs consist of two parts: The coding costs for the model $L(M)$ and the coding costs for the data $L(D|M)$. The total coding cost is therefore $L(M, D) = L(M) + L(D|M)$. K is the number of clusters C_i; N is the number of data points x; p_i is the number of parameters. Hence, coding cost is given by the following equation:

$$\begin{aligned} L(M,D) =& L(M) && + L(D|M) \\ =& \sum_{i=1}^{K} \sum_{j=1}^{|C_i|} log_2(\frac{N}{|C_i|}) + \sum_{i=1}^{K} \frac{p_i}{2} log_2(|C_i|) && - \sum_{i=1}^{K} \sum_{x \in C_i} log_2(pdf(x)) \end{aligned}$$

The model coding costs $L(M)$ is not difficult to calculate. Basically, one only needs to know the cluster sizes. The data encoding cost $L(D|M)$ is the more cumbersome part, since one needs to know the distribution of the data points. For algorithms like DBSCAN [6] that are not based on a probability distribution, this can be difficult, but k-Means is based on the assumption of a Gaussian distributed cluster, where the variance is the same in all directions. Therefore we assume the data points to be Gaussian distributed in a cluster. Since the variance is the same in all directions, the distance of a data point from the centre is sufficient to determine its probability. We need two parameters to calculate the normal distribution of a cluster, the expected value and the variance/standard deviation of the cluster. The expected value is easy to calculate, it is simply the centre of the cluster. The variance is a slightly more difficult one.

Let x be a random point in the Gaussian cluster. We assume the cluster to be centred at 0. Every axis of the cluster is $N(0, \alpha)$ distributed, i.e. normal distributed with variance α, and we want to find α as it is the variance we are looking for. We calculate the distance of x to the centre $dist(x, 0) = \sqrt{\sum_{i=1}^{d} (\alpha X_i)^2}$, with X_i being a $N(0, 1)$ distributed random variable, so αX_i is $N(0, \alpha)$ distributed. We have:

$$dist(x, 0) = \sqrt{\sum_{i=1}^{d} (\alpha X_i)^2} = \sqrt{\alpha^2 \sum_{i=1}^{d} (X_i)^2} = \alpha \sqrt{\sum_{i=1}^{d} X_i^2}$$

The term $Y = \sqrt{\sum_{i=1}^{d} X_i^2}$ is known in the literature (e.g. [9]) as being Chi-distributed and with that we have its probability distribution, which we label as $pdf(x)$. The difference is that we have $\alpha\sqrt{\sum_{i=1}^{d} X_i^2}$, but probability theory tells us, if $\sqrt{\sum_{i=1}^{d} X_i^2} \sim pdf(x)$, then $\alpha\sqrt{\sum_{i=1}^{d} X_i^2} \sim \frac{1}{|\alpha|} pdf(\frac{x}{|\alpha|})$. We now know the form of the probability distribution.

The formula for the variance is $Var[Y] = E[Y^2] - E[Y]^2$. We can rewrite $E[Y^2]$ to $E[Y^2] = E[\left(\sqrt{\sum_{i=1}^{d}(\alpha X_i)^2}\right)^2] = \alpha^2 E[\sum_{i=1}^{d} X_i^2]$. The literature tells us $Y^2 = \sum_{i=1}^{d} X_i^2$ is Chi-squared distributed and has a mean of d. Hence:

$$Var[Y] = \alpha^2 d - E[Y]^2$$
$$\alpha = \sqrt{\frac{Var[Y] + E[Y]^2}{d}}$$

The variance and mean values are the variance and mean of the distances of all data points to the centre, therefore we can compute them directly and get α.

$$pdf(x) = \frac{x^{d-1} e^{-\frac{x^2}{2\alpha^2}}}{2^{\frac{d}{2}-1} \alpha^d \Gamma(\frac{d}{2})}$$

is therefore the probability density we were looking for. Γ is the standard Gamma-function.

One could also estimate the variance differently, e. g. like X-Means [14], but this approach seems to be more compatible with the heuristics of our approach. Both have the same mathematical validity, but the use of the chi distribution for the *pdf* seems to take better account of the distortion of Gaussian spheres by outliers in the dataset. With the probability density function found, the coding costs of the data can be calculated and the MDL criterion is set up to test the Voronoi intersections.

The question that remains is which PDF should be used to model the noise. The obvious notion is to assume uniformly distributed noise, but this has the disadvantage of being massively distorted by outliers. Assume that the data set is completely within the $[0.1]^d$-cube. A single data point (2,2, ..., 2) would change the volume of the data set from 1 to 2^d and thus also change the probability of a noise data point by a factor of 2^d. To make it more resistant to such extreme outliers, the algorithm assumes that the noise is also Gaussian distributed. The parameters for the noise distribution are calculated as before, whereby the mean value is the average value of all data points and the variance of the noise is the variance of all data points.

Let us go through the steps of the algorithm so far with Figure 4. We have the old Voronoi noise cells in white that are given by the centres of k-Means and the projections of the cluster centres. They now determine the Voronoi cell intersections that are shown in gray. The MDL criterion is used to check whether the new Voronoi noise cells are to be opened at these intersections. Three of the

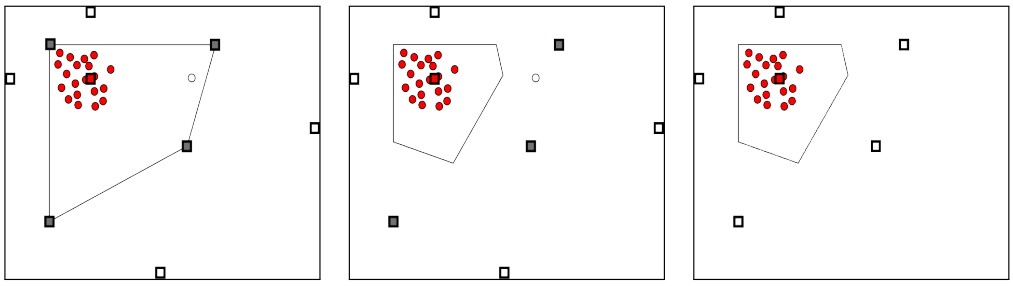


Fig. 4. The convex hull of a cluster in the course of one iteration. First, all potentially neighbouring Voronoi cells are determined (here only represented by their centres) and the intersection points of the cells are calculated; then those intersection points that open "good" noise cells, according to the MDL principle, are retained. Finally, those noise cells that are no longer adjacent are removed.

four are accepted because they either do not change or reduce the coding cost. The fourth option would, however, massively increase the coding cost as it would move many data points from the cluster to a noise cell. Therefore, only three of these intersections are kept and the new convex hull of the cluster is described by the Voronoi cells, which centres can be seen in Figure 4.2. One can see that not all these centres are necessary any more, since two of the old centres are no longer adjacent to the Voronoi cell. When eliminated, the situation would be as shown in Figure 4.3.

If one were to keep these unnecessary noise cells, the next iteration would take slightly longer and the next one somewhat longer. Depending on the number of iterations the runtime would increase massively, so the question arises how to eliminate the unnecessary centres.

2.3 Finding the Voronoi Adjacencies

The Avis-Fukuda algorithm offers the possibility to calculate the Voronoi cell adjacencies, but is not focused on this by default. Mendez et al [13] created an algorithm based on linear optimization, which specializes in this.

The algorithm translates the Voronoi adjacency problem into the linear optimization version of it and then continues the dual problem as it is more practical to solve. It begins with the equivalent of two randomly selected centres of Voronoi cells and tests whether the point in the middle is located in one of the two Voronoi cells. If this is the case, the Voronoi cells are adjacent to each other. If this is not the case, the dual problem forms the basis for the linear optimization problem, which could contain information about the neighbourhood of several other Voronoi cells. Therefore, the algorithm is faster than the Avis Fukuda algorithm, which calculates the adjacency of all pairs of Voronoi cells (see [13] for details). Mendez et al. estimate the runtime of their algorithm as $\mathcal{O}(k^2 \cdot f(k,d))$, where f is unknown, but not worse than polynomial.

By using the Mendez algorithm, our technology can now eliminate the unnecessary centres for our problem. The next iteration would now start with the centres shown in Figure 4.3 and repeat the same steps as before. The algorithm updates the centres of the clusters in the same way as k-Means, resulting in a small change of the Voronoi cell structure. If the algorithm rejects all new possible Voronoi noise cells of an iteration, then it is converged and the cluster is marked as "fully cut out".

Algorithm 1 KMN

Require: k-Means clustering result D
1: **procedure** KMN(D)
2: **Initialize:** Compute initial voronoi cell adjacencies ▷ $\mathcal{O}(k^2 \cdot f(k,d))$
3: **while** Coding cost decreases **do** ▷ l times
4: **Find v voronoi cell intersections** ▷ $k \cdot \mathcal{O}(k \cdot d \cdot v)$
5: **Check intersections v with MDL** ▷ $\mathcal{O}(v \cdot n)$
6: **Compute adjacencies** ▷ $\mathcal{O}(k^2 \cdot f(k,d))$
7: **Update cluster centres** ▷ $\mathcal{O}(k \cdot n)$
8: **Compute coding cost** ▷ $\mathcal{O}(n)$
9: **end while**
10: **return** Cluster $C_1, \ldots, C_k$ and Noise N
11: **end procedure**

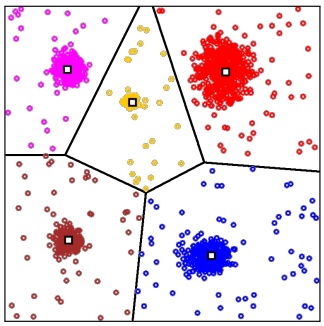
2.4 Pseudo Code

Following the pseudocode shown in Algorithm 1 we can assume that the runtime of this approach is in the order of $\mathcal{O}(l \cdot k^2 \cdot d \cdot v^2 \cdot n \cdot f(k,d))$. This shows us that this approach is relatively stable in terms of the data size n, but is somewhat more influenced by the dimensionality d and the number of clusters k. This is logical because the algorithm has to calculate the Voronoi cell intersections and adjacencies. This is independent of the size of the data set itself and could also be seen as a constant; the algorithm itself would then have a purely linear dependency on n, like k-Means itself has.

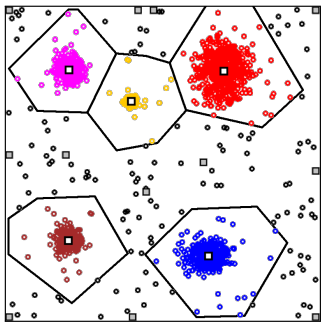
2.5 Performance on Running Example

The theory behind this technique has been presented, now let us see how it performs on our running example. In Figure 5a we have the result of k-Means on a simple dataset, consisting of 5 clusters and 10% noise. The clusters are Gaussian distributed, as k-Means assumes, and differ quite strongly in density and size. The data set is well suited for k-Means and all clusters are well separated by it. The main error lies in the wrongly assigned noise points.

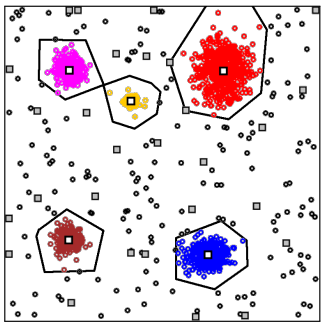
The algorithm iteratively computes the intersections, tests them with the MDL criterion, excludes noise points and updates the centres. The convex hulls



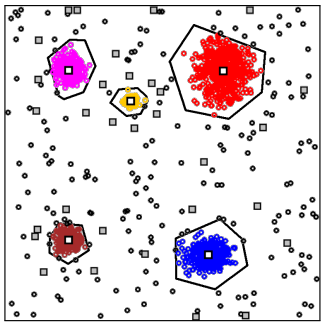
(a) k-Means with k=5. NMI is 0.82.



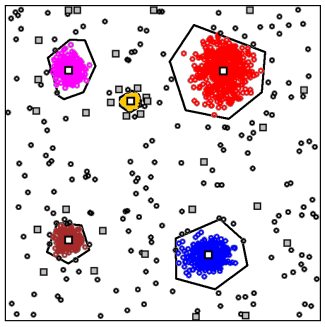
(b) The first iteration



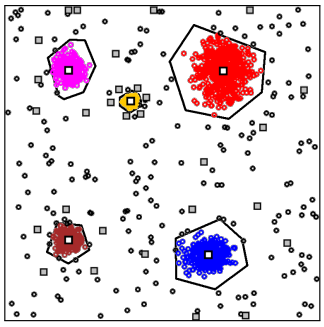
(c) Most noise is found after the second iteration.



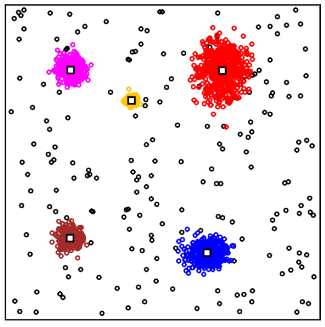
(d) The convex hulls have clearly shrunk.



(e) Some of the convex hulls do not change any more.



(f) The final result. The clusters are separated from the noise. NMI is 0.94.



(g) Ground truth.

Fig. 5. The stages of KMN from the result of k-Means to the final clustering. Cluster are displayed in different colours. Cluster-centre as white squares, centre of adjacent Voronoi cells with grey squares. The last Figure shows the ground truth. Figures best viewed in colour.

shrink until no more Voronoi noise cells are kept. For some of the clusters, this happens earlier than for others due to their spread. At the end almost all noise points are found and the clusters are cut out near ideally. The improvement can

also be measured with the help of the "normalized mutual information"-measure (NMI) [15]. The NMI value increases from 0.82 for the k-Means result to 0.94 for the result of our approach. Some of the outliers are located in the middle of a cluster and therefore cannot be recognized as such. Nevertheless, they are considered to be outliers. Therefore, a perfect NMI of 1.0 is impossible to obtain and 0.94 is almost the best result one can get.

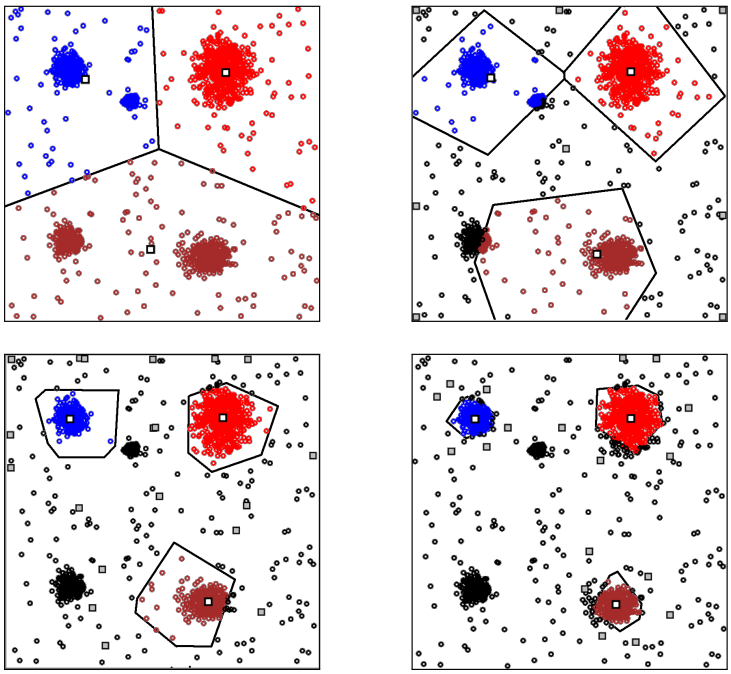


Fig. 6. The behaviour of the algorithm for a too small value of $k = 3$. Depicted is the Result of k-Means, the state after the first iteration, the second iteration and the final result.

3 Resilience in Regards to *k*

This technique has another advantage, which we would like to present now. For regular k-Means, setting k is one of the most important decisions and k is often difficult to estimate. Tools like X-Means [14] help the user with this decision, but are not necessarily correct. KMN now gives some leeway for this decision. We see this in our current example with a wrong k. If we have chosen a value of k that is too small, e.g. $k = 3$, k-Means can by default not separate all clusters. The clustering result will necessarily look similar to the one in Figure 6.1, where two clusters are merged into one. KMN now has the advantage that it eliminates data points from a cluster if they do not fit. We see the result in the first iteration. The

sub-clusters were (mostly) excluded and added to the noise points. The centre is updated (we see that it moves from Figure 6.2 to Figure 6.3) and moves to one of the correct clusters in the following iterations. At the end (Figure 6.4) three of the five clusters are found and separated from the noise, while two of them are simply added to the noise.

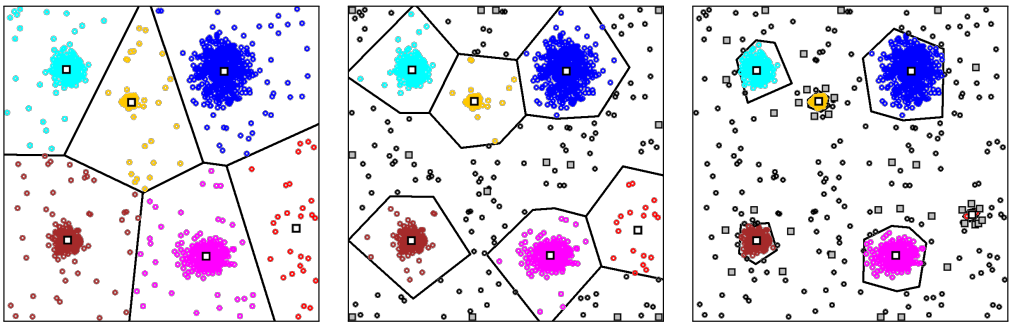


Fig. 7. K is here set to 6. Depicted is the result of k-Means, the state after the first and the second iteration, as well as the final result.

This resilience is also supported in the opposite direction. In Figure 7.1 we have k set to 6. Therefore, k-Means assigns some of the noise points to a separate cluster consisting only of noise points. We see that this "wrong" cluster loses some of its data points in the following iteration (Figure 7.2). It decreases from iteration to iteration until it is practically emptied (Figure 7.4). KMN has reduced the value of k from 6 to 5 by declaring a cluster as completely empty. We see that this is no coincidence if we choose an even bigger k of 10. The result is shown in Figure 8.1 to 8.3. K-Means has found three of the correct clusters (with some extra noise), three clusters consisting of pure noise and divided two clusters into half. When KMN is now applied to this result, the three correct clusters iteratively lose noise and converge to their correct shape. The clusters, which are divided into two halves, also lose their noise points, but remain divided into two halves. KMN does not currently have a function to merge clusters, but we hope to extend KMN to do so in the future. The three "false" clusters that consist only of noise, get most of their data points reassigned to noise. In the end, they consist of no or almost no data points. They effectively disappear.

We see that in the end, KMN may not be a technique for correctly estimating k, but it gives quite some leeway for the correct estimation. This does take some pressure from the estimation of k and techniques like X-Means are given a wider range of correct values.

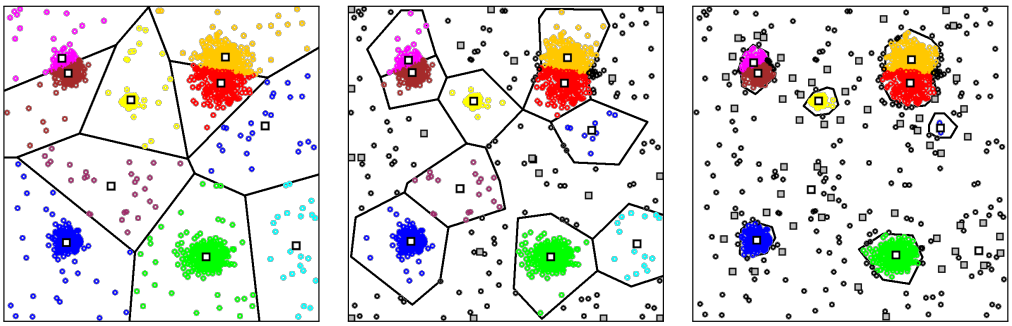


Fig. 8. K is here set to 10. Depicted is the result of k-Means, the state after the first iteration and the final result.

4 Experiments

Synthetic data We have compared the algorithms on the running example and the clustering results are shown in Table 1. The reason for using NMI as a measure for clustering lies in the opinion to see KMN as a clustering-in-the-presence-of-noise-technique. K-Means-- has been given the correct value for the outliers, for Neo-k-Means we used the internal estimator for the parameters. If a data point was assigned to more than one cluster by Neo-k-Means, it was assigned to the nearest centre. Each experiment was repeated 50 times and the average value is displayed in Table 1. We can see that KMN clearly outperforms k-Means-- and Neo-k-Means if the correct value of k is specified, but even if k is off.

Table 1. NMI Values for our running example for varying values of k.

	k=5	k=2	k=3	k=4	k=6	k=10
KMN	**0.874**	**0.412**	**0.675**	**0.813**	**0.850**	**0.780**
k-Means--	0.824	0.410	0.595	0.749	0.816	0.712
Neo-k-Means	0.760	0.354	0.546	0.713	0.808	0.722
k-Means	0.765	0.352	0.554	0.708	0.793	0.755

Real world data The difficulty of testing outlier detecting cluster techniques lies in the lack of suitable datasets, i. e. datasets containing noise data points. Campos et al. have compiled a list of possible datasets that can be used for such techniques [4]. Most of these are UCI datasets for which some of the classes have been declared noise. Most of these are also somewhat unsuitable for k-Means-based techniques, since the cluster results of k-Means have a very low quality (i.e. very low NMI). So there are very few datasets that we can use. Due to

this, we have also included another UCI dataset showing the behaviour of the algorithm on an outlier-free datasets.

Table 2. NMI Values for real word data sets.

	Hayes-Roth	Glass	WBC
KMN	**0.10**	**0.34**	0.56
k-Means--	0.08	0.33	0.34 (l=120) 0.78(l=241) 0.45 (l=361)
Neo-k-Means	0.08	**0.34**	0.00

The first dataset in Table 2, Hayes-Roth, is without outliers. The outlier parameter of k-Means-- was therefore set to 0, which means that k-Means-- gives the same results as k-Means. Therefore, we see that KMN improves on the result of k-Means. This is because k-Means assigns some data points to the wrong cluster, which KMN finds and identifies as noise. KMN notices that they do not belong in the current cluster. Clustering noise-free datasets often yields a small improvement in clustering quality compared to k-Means. Not enough to use KMN on a data set, where noise is known not to be present, but enough to warrant mention.

The data sets Glass and WBC both contain outliers. One has to keep in mind that k-Means-- needs to know the number of outliers, which is often very difficult to estimate, while KMN is parameter-free. On the WBC data set k-Means-- fares better when given the correct number of outlier. The NMI-values become identical when the outlier-number is off by roughly 30% and when the value is off by more than that, KMN delivers the better results.

The data sets were each clustered 50 times per algorithm. The runtime of KMN for WBC proved to be quite high and hence only one iteration was performed. All cluster-algorithms were given the correct values of k on the data sets.

5 Outlook and Conclusion

We wanted to create an algorithm that would be able to remove noise data points from a k-Means clustering and could achieve this without any additional parameters. In Figure 1 we see how much a k-Means clustering result can deviate from the correct clustering, even though the data set is well suited for k-Means, simply due to the noise data points that k-Means cannot account for. Through KMN we have now developed an additional algorithm for k-Means, which can be used to remove noise data points. Due to its modular design it can be used after k-Means has run its course, which means that it is completely compatible with other extensions of k-Means, such as k-Means++. Moreover, we were also able to show that KMN makes k-Means more robust in terms of too small or big k value.

For future work we have the goal to extend KMN towards a general noise extracting algorithm that can be applied as an addition for any clustering algorithm. For this goal it is necessary to abolish the voronoi cell structure of this algorithm, since that is inherent for k-Means, but not necessarily for other algorithms. Removing the voronoi cell structure might also lead to a more dynamic approach that would give us a greater flexibility.

References

1. Ahmed, M., Naser, A., *A novel approach for outlier detection and clustering improvement*, ICIEA, 2013.
2. Arthur, D., Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA, 2007.
3. Avis, D., Fukuda, K., *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, Discrete & Computational Geometry, 1992.
4. Campos, G., et al, *On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study*, Data Min Knowl Disc, 2016.
5. Chawla S., Gionis A., *k-means–: A unified approach to clustering and outlier detection*, ICDM, 2013.
6. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *A density-based algorithm for discovering clusters in large spatial databases with noise*, KDD, 1996.
7. Gan G., Kwok-Po Ng M., *k-means clustering with outlier removal*, Pattern Recognition Letters 90, 2017.
8. Hawkins D., *Identification of Outliers*, Chapman and Hall, 1980.
9. Johnson, N., Kotz, S., Balakrishnan, N. *Continuous Univariate Distributions*, Houghton Mifflin, 1994.
10. Lichman, M, *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2013.
11. Mangasarian, O., Wolberg, W., *Cancer diagnosis via linear programming*, SIAM News, 1990.
12. MacQueen, J. B., *Some methods for classification and analysis of multivariate observations*, Berkeley Symposium on Math. Stat. and Prob., 1967. Computing Voronoi Adjacencies in High Dimensional
13. Mendez J., Lorenzo J., *Computing Voronoi Adjacencies in High Dimensional Spaces by Using Linear Programming*, Mathematical Methodologies in Pattern Recognition and Machine Learning, 2013.
14. Pelleg, D., Moore A. W., *X-means: Extending K-means with Efficient Estimation of the Number of Clusters*, ICML, 2000.
15. Vinh, N. X., Bailey, J., *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance*, JMLR, 2011.
16. Wangh, J.J., Dhillon, I., Gleich, D., *Non-exhaustive, Overlapping k-means*, SDM, 2015.

Appendix B

Paper B: DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering

DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering

Benjamin Schelling
Faculty of Computer Science
University of Vienna
Vienna, Austria

Claudia Plant
Faculty of Computer Science and
ds:UniVie
University of Vienna
Vienna, Austria

***Abstract*—A data set might have a well-defined structure, but this does not necessarily lead to good clustering results. If the structure is hidden in an unfavourable scaling, clustering will usually fail. The aim of this work is to present a technique which enhances the data set by re-scaling and transforming its features and thus emphasizing and accentuating its structure. If the structure is sufficiently clear, clustering algorithms will perform far better. To show that our algorithm works well, we have conducted extensive experiments on several real-world data sets, where we improve clustering not only for k-means, which is our main focus, but also for other standard clustering algorithms.**

***Index Terms*—Clustering, Dip-Test, Dataset-Transformation**

I. Introduction

The clustering of a data set is strongly dependent on the structure it contains. If there is hardly any structure or if the structure is well hidden, clustering will most likely fail because the boundaries between the clusters are hard to determine. A strong and clearly defined structure usually leads to significantly better clustering results. Accentuating the structure would therefore be useful for clustering, but to the best of our knowledge there are currently no methods that are capable of doing so. The most one can try is normalizing the data set in the hope that this defines the structure more clearly.

We present here DipTransformation[1], which is capable of accentuating structure and turning the data set into a more clusterable form.

Consider the data shown in Fig. 1 as a 3D scatterplot as running example. It is actually not a complicated data set, consisting of three stretched Gaussian distributed clusters, with different rotations and a third dimension of uniform distributed noise, which has about the same range as the clusters. The problem here is twofold: 1) The third dimension, which does not contain any structure, is given the same weight as the dimensions that contain the entire cluster structure. 2) The clusters, while not overlapping and with clear borders, are most unfavourably scaled.

The standard clustering algorithms are surprisingly bad on this data set. K-means scores here merely 0.01 in NMI[2], DBSCAN [8], Spectral Clustering [17] and SingleLink [19] also perform disappointingly. The best choice would be EM [6] with scores 0.43 in NMI.

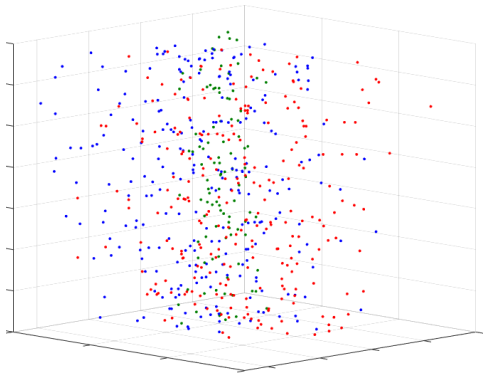


Fig. 1. Our running Example through this paper shown as a 3D Scatterplot.

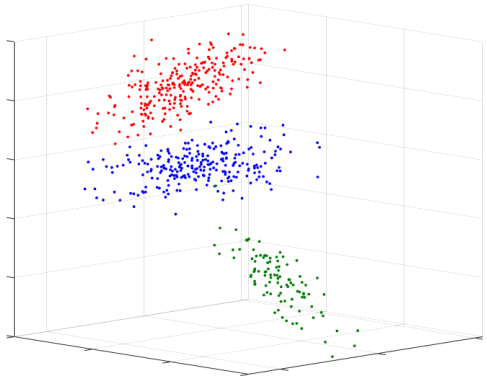


Fig. 2. Our running example after the DipTransformation in a 3D Scatterplot. It is now far easier to cluster.

Since the data set consists of a superfluous third dimension, we try dimensionality reducing techniques in the hope of adapting the dataset into a more clusterable form. The combination of clustering and dimensionality reduction is well established and might yield results here (see [13] for more details on this). However, neither PCA (0.03 in NMI) nor ICA (0.01 in NMI) lead to a data set that can be clustered

[1]Source code is found here: https://dm.cs.univie.ac.at/research/downloads/

[2]Measured in Normalized Mutual Information (NMI) [21]. NMI is scaled between 0.0 and 1.0, with 0.0 the worst possible score and 1.0 the best.

with k-means. The best choice would be t-SNE which scores ≈0.78, but has highly varying results. (All these techniques in combination with k-means with correct k.) The clusters are purely in the first two dimensions - so techniques like PROCLUS [1] and CLIQUE [2] which search for a clusters in axis-parallel subspaces could be successful, but they are not (0.21 and 0.71 in NMI, correct k for PROCLUS).

DipTransformation makes it possible to compensate for the unfortunate scaling of the features. We stated that the problem lies therein, that uniform/unimodal features (i.e. essentially structure-free features) receive the same degree of attention as such features that deviate from it. The basic assumption is that multimodal features are more interesting in regards to clustering since they contain more cluster structure. For k-means, this implicates that features with more structure should be larger scaled compared to features with barely any structure. A larger scaling would lead to a higher impact in k-means clustering due to the greater effect they have in computing the centres of the clusters and thus the way the clusters are determined. This requires a measure that evaluates the "interestingness" of a feature and therefore its scaling. We find this in the Dip-test [10] explained in Section II-A. The Dip-Test gives a appropriate measurement of the structure a feature has and thus the scaling it "deserves".

DipTransformation is capable of re-scaling and transforming our running example into a form that is almost perfectly clusterable with k-means. The clusters are better separated from each other and the structure of the data is more pronounced (see Figure 2). K-means now reaches an NMI of 0.97.

A. Contributions

This work presents an almost parameter-free method - the DipTransformation - that is able to improve the structure of a data set and thus allows k-means to cluster data sets better. The algorithm does not assume a special distribution for the clusters or data. It simply enhances structure and thereby improves clustering. Thus, it is not only a preparatory technique for k-means, it can also be used to improve clustering for various clustering techniques. DipTransformation is deterministic and requires no distance calculations. We extensively tested on real world data sets for a wide range of algorithms.

B. Related Work

The most common approach when a data set cannot be clustered well by any cluster algorithm is to create a new algorithm that can handle that data set. The reverse approach of adapting the data set to the algorithm is the much more unorthodox approach. It is usually only done in the simplest way, i.e. by normalizing a data set. In addition, there is the Z-transformation (sometimes referred to as Z-normalization), which is also relatively conventional, but is already applied far less often. Apart from these two methods, however, we are not aware of any approaches that attempt to adapt a data set with the aim of enhancing structure for improved clustering. Of course there are techniques that try to improve clustering, for instance k-means++ [3], which provides an initialization strategy for k-means that is often very successful, but transforming a data set is unusual. One might consider SynC [4] as a transformation technique, because it collapses clusters into single points using the principle of Synchronization.

Subspace clustering techniques such as the aforementioned PROCLUS and CLIQUE can be considered related work, since they intend to reduce dimensionality, i.e. adapt the data set by removing "unnecessary" information. The DipTransformation does not remove any information, but - as the analysis of the Running Example will show - it is very capable in dealing with such noise information. Of particular interest are FOSSCLU [9] and SubKMeans [16] which intend to reduce dimensionality with the goal of finding a subspace compatible with k-means.

We are also aware of progress in the field of Deep Learning, where techniques such as DEC [24] and DCN [22] are being developed, aimed at finding good subspaces using neural networks.

Spectral clustering takes a data set and transforms it into a distance matrix, computes its eigenvectors and applies (mostly) k-means to the data set. It is not necessary to use k-means, other partitioning algorithms can also very well be used. In this regard are spectral clustering techniques similar to the DipTransformation. They take the data set and try to transform it into a more clusterable form. One of the most well known is the fundamental technique by Ng, Jordan and Weiss [17]. We also use the popular Self-Tuning Spectral Clustering [25] as well as FUSE [23] as comparison methods due to them being state-of-the-art techniques.

DipTransformation uses the Dip-test for measuring structure and therefore one can consider all data mining-techniques that use the Dip-test as related. It was first used in data mining by DipMeans [11] with the goal of estimating the number of clusters for k-means. After that, we only found SkinnyDip [15] using the Dip-test. We conclude, that it is still a rather unknown tool, that has not yet found full recognition.

One must bear in mind while reading this, that DipTransformation is not a rival for all the mentioned techniques in the classical sense, but that it can be used as a supporting technique that eases the difficulty in the task they attempt. We will show in the experimental section (see Section IV) that they can all benefit from DipTransformation.

II. The Algorithm

A. The Dip-Test

To understand how the algorithm works, we must first go into detail about the Dip-test.

The Dip-Test was created by Hartigan & Hartigan in the 1980s as a measure of how much a sample deviates from unimodality. Unimodality is defined here as a distribution that is convex until it reaches its maximum and concave thereafter.

The test starts with sorting the sample and then creating the Empirical Cumulative Distribution Function (ECDF). This can be seen in Fig. 3. The histogram shows 4 clusters (A, B, C and D), which can be clearly identified in the ECDF to its right. The Dip-Test only requires the ECDF; the histogram is

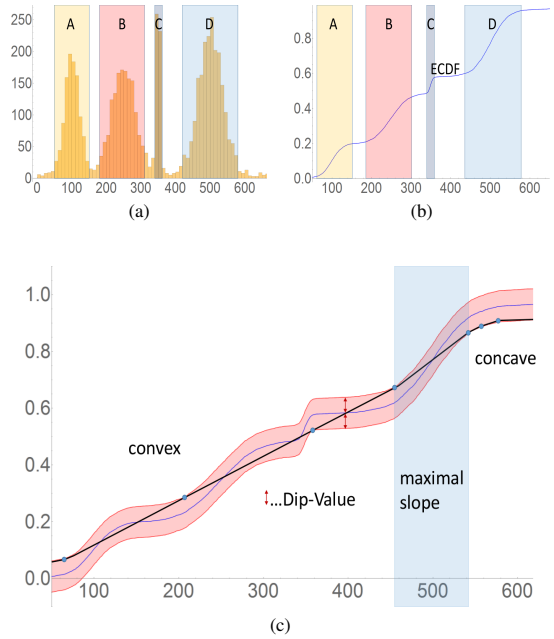


Fig. 3. A histogram and the resulting ECDF (empirical cumulative distribution function). The dip-test uses the ECDF to find out how much it deviates from a unimodal distribution, i.e. how big the offset is to fit a unimodal distribution. The offset is the dip-value.

only for visual clarity. It therefore has no bin-width parameter. In fact, it has no parameter at all.

The Dip-test measures the extent to which the ECDF deviates from unimodality. It computes how much the ECDF has to be offset, so that it can fit a unimodal distribution. This can be seen in Fig. 3(c). The ECDF has been shifted vertically by a certain value (the dip value), and ECDF+dip and ECDF-dip is plotted there. This offset is large enough so that a line can be drawn in between ECDF+dip and ECDF-dip, which is first convex and then concave. This line is representative of the closest possible unimodal distribution. The dip-value (or "dip") shows how much the ECDF is off from such a unimodal distribution.

The Dip-Test also gives a second value, a probability of how likely a sample is unimodal, as well as the interval of the highest slope, but we only need the offset/dip-value. The dip-value is always in the interval $(0, 0.25]$, hence it is always positive.

The Dip-Test has a runtime of $\mathcal{O}(n)$, but since its input must be sorted to create the ECDF, the effective runtime for this part of the technique is $\mathcal{O}(n \log n)$. Further details about the Dip-Test can be found in [10].

B. Applying the Dip-Test

By means of the Dip-Test, we obtain a value, the dip statistic, which provides a measure of the structure of a dimension and thus, as explained in the introduction, a measure of the "relevance" of the dimension. The more relevant a dimension is, the larger will it be scaled (in relation to the other dimensions) and the greater its influence on the clustering result from k-means.

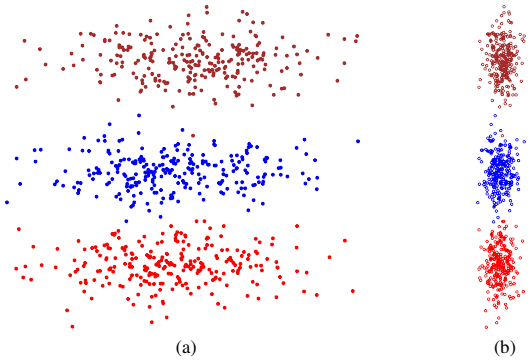


Fig. 4. A simple data set before (a) and after (b) scaling with the Dip-Values.

Let us consider this approach with Fig. 4. The dip values of the individual dimensions are 0.009 for the projection into the horizontal dimension and 0.063 for the projection in the vertical one. We rescale the horizontal axis in the interval [0,0.009] and the vertical one in [0,0.063] and get the dataset represented in Fig. 4b.

The changed distances make this data set now easily clusterable by k-means. The dimension containing the structure is now much more pronounced and accordingly more influential for k-means. The improvement of the clustering result is best described using the NMI value, which increases from an average value of 0.55 for the unscaled data set to 0.98 for the rescaled data set (100 random initialisations each). The only error and the reason why an NMI-value of 1.0 is not reached is due to some edge-data points that have been falsely assigned, but could not reasonably be expected to be correctly clustered.

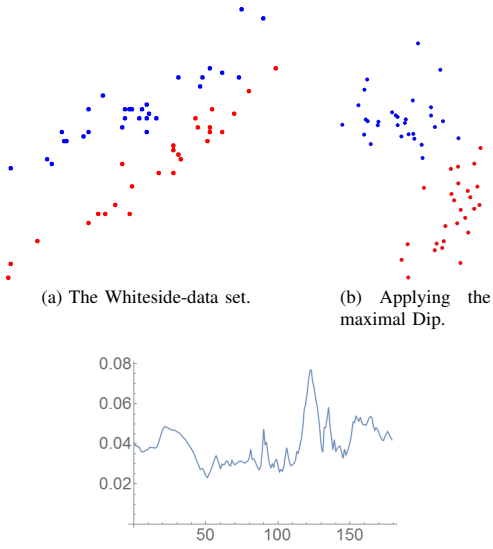
This (somewhat trivial) example shows how important it is to enhance the structure of a data set. The horizontal axis in which the data set has barely, if any, structure is reduced to a very small range and the vertical axis, where the clusters and structure are located, is now the relevant dimension that determines the result of k-means.

C. Lopsided Cluster

We do not generally assume axis-parallel stretched cluster. Let us therefore look at the Whiteside data set depicted in Fig. 5a. The Whiteside data set is a real-world data set.

The clusters are obviously not axis-parallel. We have the same situation here as in the simple data set shown in Fig. 4, as in that k-means fares rather badly. To be precise the NMI-value is 0.006. Scaling the axes with the Dip-Test values is ineffective; it improves clustering only minimally.

We see in Fig. 5c that the dip value changes greatly depending on the angle at which the dip value is measured. If we rotate the Whiteside data set by the angle at which we find the maximal dip value, the clusters are almost axis-parallel. Now scaling the axes with their dip value leads to



(a) The Whiteside-data set. (b) Applying the maximal Dip.

(c) The Dip-value of the Whiteside data set when rotated.

Fig. 5. The Whiteside-data set and the change of the Dip-value of an axis when the data set is rotated (0 to 180 degrees) as well as the Whiteside-data set, when it is rescaled at the maximal dip.

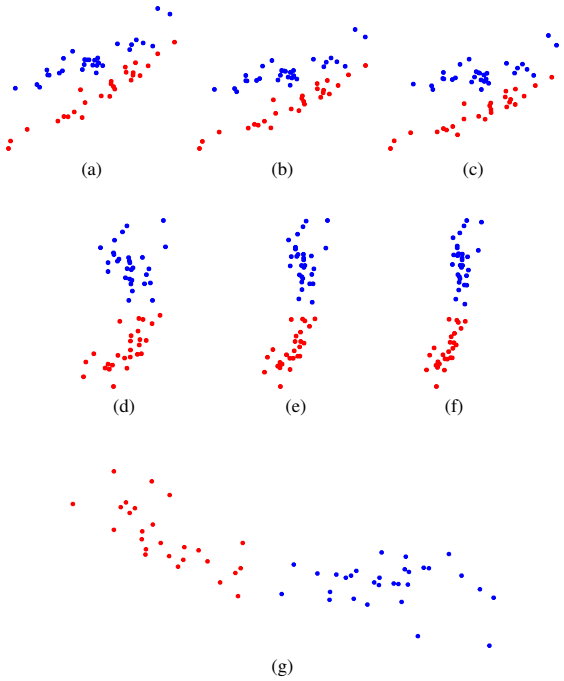


Fig. 6. Steps in the DipTransformation of the Whiteside data set and the final data set. There seem to be big leaps in the transformation; this is due to the iterations, when the maximal new dip value is smaller compared to the $MaxDip$, and thus no scaling takes place. Until a new $MaxDip$ is found the data set is rotated quite a lot and the changes seem extensive.

the transformed data set shown in Fig. 5b. This data set can be clustered considerably better by k-means. In fact, we get an average NMI of 0.92. This is a massive improvement over the previous NMI of 0.006, but it is possible to improve even further, as we will see.

Determining the angle of the maximal Dip-value is not straightforward. Of course, one could use the brute-force approach of simply testing as many angles as possible, but this will prove impractical at the latest when the data set is of higher dimensionality. A too simple search algorithm will also not lead to a satisfactory result, since the data, as we have seen in Fig. 5c, has more than a few local optima. Hence, we do not try to find only a single maximal dip-angle, but scale the data set along along several high dip angles. Basically, the algorithm does not restrict itself to only re-scaling the data set along the maximal dip-value, but rescales the data set along multiple such high dip values. This converts the data set into a more clusterable form. The algorithm scales the data set in various instances, so that the structure of the clusters become more clearly defined.

We start with two dip-values, D_1 and D_2, the two dip values along the axes. We calculate the ratio between those values $r = Max(\frac{D_1}{D_2}, \frac{D_2}{D_1})$. If r is high, then there is a good chance that we have hit a good dip-value. Then we rotate the data set in clockwise direction by an angle of $\frac{1}{r} * c$, with c as the rotation speed parameter. This ensures that we rotate the data set only by a small angle if chances are good that we are close to a high dip value and by a bigger angle if the dip values are similar, i.e. chances are that no high dip value is close. The rotation speed parameter c can be freely selected. The larger c is selected, the shorter the runtime, but a smaller c of course makes it more likely to find high dip-values. (Throughout the paper, the rotation speed parameter c is set to 5. Its effects are further explored in Section II-E.) The algorithm remembers the maximal overall dip value (we refer to it as $MaxDip$) and every time it finds a new maximal overall dip value, the axes are scaled with their respective dip values. The total degrees the data set has been rotated is remembered and after 360° the algorithm stops. For the Whiteside-data set with $c = 5$ leads this to the transformation displayed in Fig.6.

At the beginning of the transformation, the changes are comparatively small. This is because the dip values of the axes are not very different and therefore scaling the axes only leads to limited changes. $MaxDip$ is updated, whenever a new Maximum is found. The following iterations is the maximal dip value of the axes not greater than $MaxDip$ and hence no scaling takes place. However, after several rotations, the maximal dip value of the axes is greater than $MaxDip$ and the data set is scaled again. This happens between Fig. 6(c) and Fig. 6(d). Because the data set is rotated over several iterations, it has been rotated by a rather large angle and the following scaling makes the data set look quite different. The algorithm remembers the new $MaxDip$. The data set continues to rotate, but due to the way the algorithm selects the rotation angle, it is only rotated by a small angle, which is advantageous because finds high dip values. In the following iterations, the data set

is again scaled several times. In the second row of Fig. 6, it can be seen that the data set does not change drastically, but becomes more compact and the clusters are defined more clearly with each iteration. Fig. 6(g) shows the final state of the data set. Between Fig. 6(f) and Fig. 6(g) is again a stretch where the data set is rotated but no new $MaxDip$ is found.

This transformation of the Whiteside-data is very easy to cluster for k-means. We get an average NMI-value of 1.0 (in 100 iterations), which means that the data set is perfectly clustered. This result is not specific to a value of $c = 5$, but can also be reached by e.g. $c = 9, 8, 7, 6, ...$ However, the transformed data set may look slightly different for a different value of c.

This transformation is easier to cluster, when comparing to the original Whiteside-data set shown in Fig. 5, but also when comparing to the transformation along the maximal Dip-value angle shown in Fig. 5b. One could have expected these transformations to be more similar, if not identical, but that is not the case. The transformation here is not along an orthogonal basis.

Scaling along axes leads to a basis transformation that stretches the basis vectors, but leaves orthogonality intact. Applying the transformation method sketched above leads also to a change in basis vectors, which no longer implies that two previously normal (i.e. perpendicular) vectors are normal to each other afterwards.

Theorem 1: The DipTransformation DT is a linear operator. More precisely, it is a basis-transformation.

Proof. Every rotation in $\mathbb{R}^n$ can be expressed as a matrix R. Scaling a data set in $\mathbb{R}^n$ simply means applying a diagonal matrix S with the scaling-parameters in the main diagonal. Hence, applying the DipTransformation on a data set is equivalent with applying the Rotation- and Diagonal-matrices R_1, S_1, R_2, S_2, R_3, S_3,.... Thus, the DipTransformation DT is the product of matrices, which is again a matrix. A matrix is a linear operator, hence the DipTransformation is a linear Operator.
A rotation is an orthogonal matrix with determinant 1, a scaling matrix has the determinant $c_1 \cdots c_n$, with c_i the entries in the diagonal. Since the Dip-Test values c_i can never be zero, the determinant of the scaling matrix is non-zero. The determinant has the property $Det(A \cdot B) = Det(A) \cdot Det(B)$, hence the determinant of the DipTransformation is $Det(DT) = Det(R_1) \cdot Det(S_1) \cdots Det(R_l) \cdot Det(S_l) = 1 \cdot (c_{11} \cdots c_{n1}) \cdots 1 \cdot (c_{1l} \cdots c_{nl}) \neq 0$. Thus is DT a matrix with non-zero determinant, i.e. a basis-transformation. ■

The focus of DipTransformation is on k-means, but we see from Fig. 6 that other techniques might also benefit from this approach. We will explore in Section IV how other techniques are influenced by this (and other) transformed data set(s).

D. More than 2 Dimensions

The algorithm for a 2-dimensional data set was explained in detail, because it forms the basis for data sets with more than two dimensions. There are several ways to adapt this approach; the one that seems to work best is now explained:

The main difference is that there are more than two directions, the data set can be rotated in. It would seem logical to rotate the data set in all directions at once following the angle-computation as before, but there is a problem involved with that: Rotations are not commutative. That means, it makes a difference in which order the rotations are executed. Finding only one non-axes parallel angle in which the data set is rotated, is anything but straightforward, since all we have are the dip values of the axes, that we can use to compute axes-parallel rotation angles. Nevertheless, the algorithm simply executes one rotation after another. However, since every rotation changes the data set (slightly), it is better to recalculate the dip values. This could be omitted to save runtime, but the recalculated dip values are more precise and this in turn improves the transformation, especially with larger rotation speed parameters c. One might expect that changing the order in which the rotations are executed might improve the transformation, but this is not the case, according to the experiments we conducted. We also tried only executing the rotation with the highest/lowest dip value, but this even seems to impair the transformation. Through all rotations the algorithms remembers the maximal dip value found as $MaxDip$, just as before. Whenever the rotated data set has a dip value larger as $MaxDip$, the data set is scaled and $MaxDip$ is updated. The rotation angles are calculated in the same way as for a 2-dimensional data set. Furthermore, the executed rotations are rotations in the plane given by two axes-vectors.

One has to keep in mind that in higher dimensional data set, the algorithm has a larger area to search for high dip values. It is only a "half-circle" or 180° that needs to be traversed for a 2-dimensional data set. (In the interest of precision, however, the algorithm looks over the full 360°.) For a $d-$dimensional data set, it would be half of a $d-$dimensional sphere. To compensate for this, the algorithm assumes that $d{\cdot}360°$ have to be traversed. This range ascertains that all maxima can (theoretically) be found. Furthermore, it is not necessary to find the maximal dip values exactly; being close enough is sufficient to assure a good transformation.

E. The Rotationspeed Parameter c

The rotation speed parameter c has been explained in Section II-C and we now want to analyse its effect on the DipTransformation. Fig. 7 shows how NMI (for k-means) changes with different values of c and the effect is not very pronounced. The data sets depicted were chosen, because their values do not overlap, but the effect is rather similar for all data sets examined in Section IV.

There is a slight tendency for the clustering results to lose quality at higher values of c (best visible for the Whiteside data set), but it is not very pronounced. For an unknown data

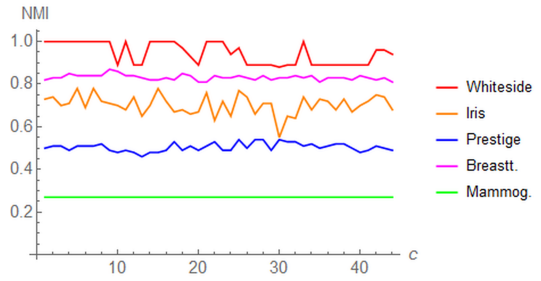


Fig. 7. The NMI-value for k-means with correct values for k vs the rotation speed parameter c for five real world-data sets.

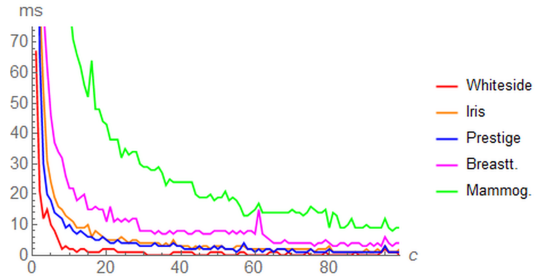


Fig. 8. The time DipTransformation takes in milliseconds depending on the value of the rotation speed parameter c for five real world-data sets.

set is a smaller value for c nevertheless more recommended. DipTransformation is dependent on finding the high dip-values and that is simply more likely for smaller rotation speed. If runtime is the primary factor (for example, for very large data sets), then a larger value for c might be more recommendable. Fig. 8 shows how the runtime is linked to the parameter c and how it decreases for smaller c. There is a (small) trade off between clustering quality and runtime, but as a general rule a high value of c seems not too detrimental. For this work, however, we stick to the fixed value $c = 5$.

III. Runtime and Pseudocode

Following the structure of the DipTransformation algorithm (outlined in Algorithm 1) we can make a runtime estimation: Scaling of a data set as well as rotating a data set has a runtime of $\mathcal{O}(n)$; Computing the Dip-Values is in the order of $\mathcal{O}(n \log n)$, since the values have to be sorted. There are two For-loops over d, where d stands for the number of dimensions. If the number of iterations in the while-loop is l, then the runtime can be estimated as:

$$\mathcal{O}(n) + \mathcal{O}(n \log n) + l \cdot d \cdot d \cdot \big(\mathcal{O}(n) + \mathcal{O}(n \log(n)) + \mathcal{O}(n)\big) \approx \mathcal{O}\big(l \cdot d^2 \cdot n \log(n)\big)$$

IV. Experiments

Persuading someone that a data set is easy to cluster, if the data set is more than two-dimensional, is difficult. The goal of the DipTransformation though is to ensure that a data set becomes easier to cluster. This work will of course

Algorithm 1 DipTransformation

Require: Data D, Rotationspeed c

procedure DipTransf(D, c)
  $Degree \leftarrow 0$
  Compute $DipValues$
  Scale(D,DipValues)
  $MaxDip \leftarrow Max(DipValues)$
  while $Degree < dim * 180°$ **do**
    for i = 1,...,dim **do**
      for j = i+1,...,dim **do**
        $a \leftarrow Max(Dip(i)/Dip(j), Dip(j)/Dip(i))$
        Turn $D(i, j)$ by angle c/a
        $Degree \leftarrow Degree + c/a$
        Compute $DipValues$
        if $Max(DipValues) > MaxDip$ **then**
          Scale(D,DipValues)
          $MaxDip \leftarrow Max(DipValues)$
        end if
      end for
    end for
  end while
  return D
end procedure

show with NMI values of experiments on real-world data set that DipTransformation is capable of doing that, but we would also like to show that with plots. Fig. 9 shows pairwise plots of the "Banknote Authentication" data set from the UCI-Repository [7]. This data set was chosen because it has a small dimensionality of four, so that all pairwise plots can be shown. Fig. 9 illustrates the difficulty involved with clustering this data set. The clusters are not clearly separated and often overlap, so it is not suited for k-means. In numerical values can this be expressed as an NMI-value 0.03 for k-means with the correct value for k. After the DipTransformation (shown in Fig. 10) the data set is much better structured and the clusters are well separated. K-means can now identify the clusters rather well. In fact, the NMI-value for k-means with the correct value for k is now 0.68.

A. Synthetic Data

The first analysed data set is the synthetic data set given in Fig. 1. This is a data set that - as we have seen - is quite difficult to cluster; k-means fares extremely bad and scores no higher than 0.01 in NMI. Other algorithms are often only marginally better. Table I shows the NMI-results. Most of them are not impressive, with 11 of the 20 algorithms scoring below 0.10. Applying the DipTransformation onto the data set leads to a massively enhanced data set with clearly stronger defined structure. The three clusters that were before stretched and scaled quite unfavourable for clustering are now well separated and compact. Clustering of this data set is far easier and the results shown in Table I demonstrate this. All of the used algorithms improve due to the DipTransformation - on average

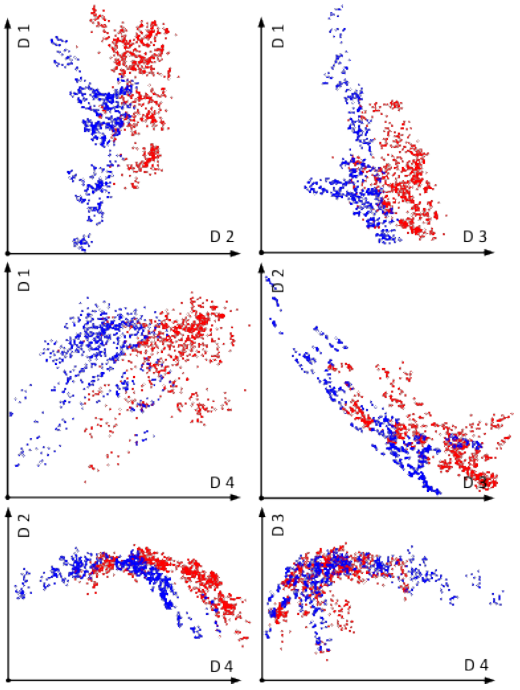


Fig. 9. Pairwise Plot of the Banknote-Authentication data set before DipTransformation. The dimensions are given as axes-label.

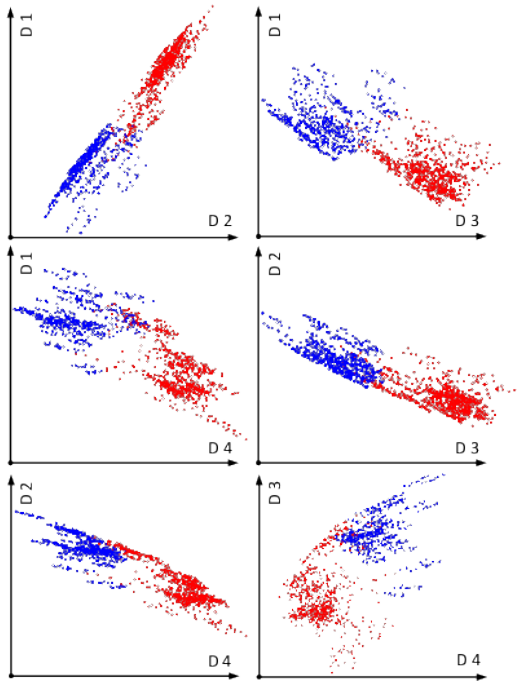


Fig. 10. Pairwise Plot of the Banknote-Authentication data set after the DipTransformation. The clusters are visibly better separated from each other.

TABLE I
CLUSTERING OF THE RUNNING EXAMPLE BEFORE AND AFTER DIPTRANSFORMATION. THE AVERAGE IMPROVEMENT IN NMI IS 0.636.

Running Example	before	after
k-means	0.01	0.97
k-means++	0.01	0.98
DipMeans	0.00	0.98
SkinnyDip	0.00	0.98
Spectral Clust.	0.36	0.97
STSC	0.00	0.99
FUSE	0.06	0.75
DBSCAN	0.23	0.95
SingleLink	0.01	0.96
EM	0.43	0.50
SubKMeans	0.08	0.63
FossClu	0.60	0.78
SynC	0.05	0.95
PCA	0.03	0.63
ICA	0.01	0.98
t-SNE	0.78	0.80
PROCLUS	0.23	0.92
CLIQUE	0.71	0.98
DEC	0.38	0.53
DCN	0.12	0.58

0.636 in NMI. After the DipTransformation are 12 of the 20 algorithms better than 0.90.

FossClu [9] and SubKMeans [16] try to find an optimal subspace for clustering while transforming the data set themselves. These transformation attempts are also more successful after DipTransformation has transformed the data set. Even other transformations profit from the DipTransformation.

B. Real-World Data sets

We have tested the DipTransformation extensively on 10 real world data sets, which differ greatly in dimensionality, number of data points and number of clusters. The ultimate goal of the DipTransformation is to enhance the structure of a data set and thus to improve clustering. To show that this goal can be achieved, we have transformed these data sets and applied the basic k-means algorithm on them. The results can be seen in Table II. The difference in clustering quality is obvious. While k-means on the original data sets usually fares somewhat lacking and other algorithms like Spectral Clustering are often the better choices, it does perform extremely well on the transformed data sets. Transforming the data set has enhanced its structure so that k-means can cluster the data sets better than the compared methods. In 9 of the 10 cases k-means clusters better (or no worse) than the other methods. Only in one case does it take second place with a deficit of 0.02.

We chose the algorithms we found to be most relevant as comparison methods here. This included the data set-transformation techniques of normalizing and Z-Transformation, the standard data mining algorithms DBSCAN [8], EM [6] and SingleLink [19], DipMeans [11] and SkinnyDip [15] as techniques based on the Dip-Test, SubKMeans [16] and FossClu [9] as the most similar Subspace-clustering-techniques, the aforementioned Spectral Clustering-methods, SynC [4], as well as PCA and t-SNE in combination

TABLE II
EXPERIMENTAL RESULTS. ALL NON-DETERMINISTIC RESULTS HAVE BEEN REPEATED 100 TIMES AND THE AVERAGE IS GIVEN. THE CORRECT VALUE FOR NUMBER OF CLUSTERS IS ALWAYS GIVEN.

Data set	Whiteside	Skinsegmen.	Banknote	Iris	Prestige	Userknow.	Mammographic	Seeds	Breast Tissue	Leaf
# of data points	56	245057	1375	150	98	258	830	210	106	340
# of dimensions	2	3	4	4	5	5	5	7	9	14
# of clusters	2	2	2	3	3	4	2	3	6	30
DipTransformation	**1.00**	**0.32**	**0.68**	**0.84**	**0.68**	**0.53**	0.27	**0.78**	**0.51**	**0.69**
k-means	0.01	0.02	0.03	0.70	0.51	0.26	0.11	0.70	0.32	0.65
Normalized	0.01	0.02	0.02	0.68	0.50	0.28	0.27	0.67	0.49	**0.69**
Z-Transformation	0.01	0.02	0.02	0.68	0.51	0.28	0.28	0.67	0.49	**0.69**
k-means++	0.01	**0.32**	0.03	0.75	0.56	0.22	0.11	0.71	0.18	0.57
DipMeans	0.00	—	0.25	0.55	0.45	0.00	0.00	0.00	0.00	0.45
SkinnyDip	**1.00**	—	0.34	0.55	0.55	0.30	0.00	0.53	0.26	0.00
Spectral Clust.	0.06	—	0.03	0.60	0.60	0.29	0.09	0.34	0.45	**0.69**
STSC	0.35	—	0.26	0.39	0.53	0.03	—	0.66	0.31	0.09
FUSE	0.09	—	0.03	0.46	0.06	0.02	0.06	0.15	0.11	0.31
DBSCAN	0.27	—	0.46	0.62	0.54	0.27	0.14	0.50	0.41	0.59
SingleLink	0.11	—	0.03	0.61	0.08	0.05	0.00	0.05	0.27	0.35
EM	**1.00**	0.23	0.01	0.58	0.28	0.44	0.01	0.63	0.37	0.25
SubKMeans	0.01	0.01	0.01	0.66	0.56	0.22	**0.29**	0.73	0.45	0.66
FossClu	—	0.27	0.44	0.75	0.48	0.50	0.08	0.50	0.32	0.34
SynC	0.12	0.13	0.14	0.58	0.52	0.13	0.24	0.48	0.29	0.27
t-SNE + k-means	0.02	—	0.64	0.31	0.02	0.06	0.11	0.16	0.08	0.35
PCA + k-means	0.01	0.01	0.01	0.64	0.56	0.21	0.26	0.74	0.49	**0.69**

TABLE III
K-MEANS++ BEFORE AND AFTER THE DIPTRANSFORMATION AS WELL AS K-MEANS AFTER THE DIPTRANSFORMATION. NUMBER OF CLUSTERS IS GIVEN.

Data set	Whiteside	Skinsegmen.	Banknote	Iris	Prestige	Userknow.	Mammographic	Seeds	Breast Tissue	Leaf
k-means before	0.01	0.02	0.03	0.70	0.51	0.26	0.11	0.70	0.32	0.65
k-means++ before	0.01	0.32	0.03	0.75	0.56	0.22	0.11	0.71	0.18	0.57
k-means after	**1.00**	0.32	0.68	0.84	**0.68**	0.53	**0.27**	**0.78**	**0.51**	0.69
k-means++ after	**1.00**	**0.44**	**0.69**	**0.86**	**0.68**	**0.64**	**0.27**	0.76	0.49	**0.70**

with k-means. For PCA and t-SNE we decided not to reduce the dimensionality, because there is no straightforward answer on how far one should reduce the dimensionality and because DipTransformation also does not reduce dimensionality.

a) Parameters and Determinism: Algorithms like DBSCAN always raise the question of how to set the parameters. To compare the DBSCAN results fairly, we decided to make the parameters dependent on the average pairwise Euclidean distance of data points. Let us call it e. We tested all combinations of distances in $\{0.05 \cdot e, 0.1 \cdot e, 0.2 \cdot e, 0.3 \cdot e, 0.4 \cdot e, 0.6 \cdot e, 0.8 \cdot e, e\}$ and MinPts in $\{1, 2, 3, 5, 10, 50\}$. Only the best NMI result is reported.

All techniques that require the number of clusters as a parameter have been given the correct number of clusters k. The only exception is SingleLink where all values in the interval $[k, 2k]$ have been tested and only the best result is reported. This decision is due to the characteristic of SingleLink to declare single data points or small subsets of a cluster as clusters.

Non-deterministic algorithms such as k-means have been iterated 100 times to reduce random effects and provide robust results.

b) Skinsegmentation: The Skinsegmentation-data set is a somewhat difficult data set simply due to its size of roughly a quarter of a million data points. For many of the provided implementations was the size too large and the execution failed. This also applies to some of the standard methods like SingleLink, DBSCAN and Spectral Clustering. These were tested on more than one implementation on different platforms, but would not run through anyway.

c) Spectral Clustering: If this paper refers to Spectral Clustering as an algorithm and not the class of algorithms, then the classical algorithm by Ng, Jordan and Weiss [17] is meant.

Besides these considerations it is most noticeable that k-means++ leads to the same increase in NMI as the DipTransformation. However, K-means++ and DipTransformation are by no means mutually exclusive and can be used together. This in fact leads to an even better performance on the Skinsegmentation data set. While they separately reach a level of 0.32 in NMI, they manage 0.44 in combined form.

C. K-means++ and DipTransformation

As mentioned, k-means++ and DipTransformation are not mutually exclusive. We tested on all the data sets used in the experiments whether k-means++ fared better before or after the DipTransformation. The results are shown in Table III. Following these, we can say that k-means++ is a bit of a

double-edged sword on the original data sets. On some of them (Skinsegmentation, Iris) k-means++ is clearly better than k-means; on some of them (Breast Tissue, Leaf) it is the other way round. After the DipTransformation, the situation is far more beneficial for k-means++. Usually, there is only a small difference between k-means and k-means++ (≤ 0.02), indicating that there might be fewer local optima, compared to the original data set. The only times when k-means and k-means++ do differ (Skinsegmentation, Userknowledge) is when k-means++ performs quite a bit better than k-means alone.

DipTransformation can be used together with all types of support techniques for k-means (or other clustering algorithms). For example, X-means [18] can be used to find the number of clusters, k-means-- [5] to find outliers, k-means++ to find an initialization, SubKMeans to find a subspace and all this in combination with the DipTransformation.

D. DipTransformation and Clustering Algorithms besides K-means

DipTransformation was developed with a focus on k-means, but as we have stated throughout our work, DipTransformation only enhances the structure; it does not adapt the data set so that it only fits k-means and therefore other algorithms can also benefit from it. We have seen the transformation of the Whiteside as well as the Banknote-Authentication data set in Fig. 6 respectively Fig. 10 and both of these do seem easier to cluster for various algorithms. We have taken 5 of the data sets used in the real world data sets experiments and clustered their DipTransformations with 4 standard data mining algorithms, i.e EM, DBSCAN, SingleLink and Spectral Clustering. The results can be seen in Table IV. We chose the standard algorithms because they are well-established in the community, which makes the results all the more credible. For the sake of completeness is k-means also included here. In two cases do we see a tiny decrease in clustering quality of 0.01 in NMI. In two more cases does the quality not change at all. In the other 21 cases does the quality increase - in some cases substantially. On average, counting all cases, the quality increases by 0.223 in NMI.

Combined with Fig. 6 and 10, this should be a very convincing argument that DipTransformation can play an important role in Clustering as a support technique applied to the data set before clustering.

E. Runtime Comparisons

The runtime was estimated to have a $\mathcal{O}(n \log n)$-dependency on the number of data points n. Synthetic data sets ranging from 1.000 to 15.000 data points were created to test for this dependency and to compare with other algorithms. The runtime is plotted in Fig. 11. It is not immediately apparent, but $\mathcal{O}(n \log n)$ is a very good estimate for the runtime. We also see that DipTransformation performs quite well compared to the other algorithms tested there. DipTransformation is faster for all data sets. To be fair is in this test also the runtime of k-means included in the measured time for DipTransformation,

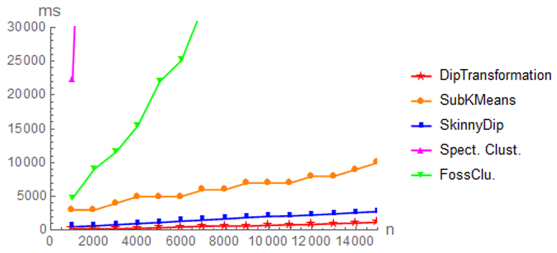


Fig. 11. Runtime relative to the data set size n.

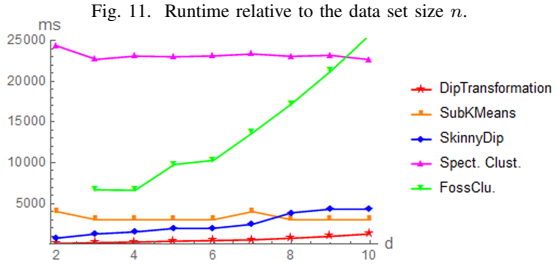


Fig. 12. Runtime relative to the dimensionality of the data set d.

since the other methods cluster data, which DipTransformation does not do by itself.

Besides the size of the data set is also the influence of the dimensionality of the data set on the runtime essential. This is shown in Fig. 12. We created 9 data sets ranging in dimensionality from 2 to 10 with 1.000 data points each. Here is DipTransformation (+k-means) again faster than all other methods, but we do see in the behaviour of the measured time, that other methods like Spectral Clustering are less affected by the dimensionality. The estimation of an $\mathcal{O}(d^2)$ dependency on dimensionality is again a very good one, so the conjecture that at some point DipTransformation will need more time than e.g. Spectral Clustering is a likely one. However, if one extrapolates from the curves, it seems as if that would happen at a rather high dimensionality.

The algorithms we found to be most similar to DipTransformation were chosen here. They were tested in Java (DipTransformation and FossClu), Scala (SubKMeans) and R (Spectral Clutering and SkinnyDip) on an Intel Xeon E5 with 16Gb RAM.

V. Limitations and Outlook

DipTransformation is a very powerful technique for improving the structure and emphasizing clusters, but there are certain limits. For example, if two clusters overlap or interlock, DipTransformation would by design not be able to separate them. DipTransformation stretches and scales the data, therefore all cluster which do not have a hyperplane in-between them cannot be separated. The same applies to clusters that contain more than one mode. The dip test is designed to work with unimodal distributions. Multimodal clusters can prevent DipTransformation from working properly.

TABLE IV
VARIOUS CLUSTERING ALGORITHMS BEFORE AND AFTER DIPTRANSFORMATION. THE CORRECT VALUE FOR k IS ALWAYS GIVEN. ON AVERAGE THE CLUSTERING IMPROVES BY 0.223 (MEASURED IN NMI).

Data set		Whiteside	Iris	Prestige	Mammographic	Breast Tissue
EM	before	**1.00**	0.58	0.28	**0.01**	0.37
	after	**1.00**	**0.90**	**0.63**	0.00	**0.45**
DBSCAN	before	0.27	**0.62**	0.54	0.14	0.41
	after	**0.72**	0.61	**0.60**	**0.15**	**0.45**
SingleLink	before	0.11	**0.61**	0.08	0.00	0.27
	after	**0.82**	**0.61**	**0.54**	**0.16**	**0.35**
Spectral Clustering	before	0.06	0.60	0.60	0.09	0.45
	after	**1.00**	**0.65**	**0.65**	**0.26**	**0.50**
k-means	before	0.01	0.70	0.51	0.11	0.32
	after	**1.00**	**0.84**	**0.68**	**0.27**	**0.51**

In terms of runtime, the main constraint we encountered was the $\mathcal{O}(d^2)$-dependency on the dimensionality of a data set. For very high dimensional data sets it might be useful to combine dip transform with a dimensionality reduction algorithm at the current state.

We do intend to implement a dimensionality-reducing feature into DipTransformation. The dip-test provides a probability estimate of the unimodality of a feature. For the running example, the dip test gave a probability of 100% for the third dimension to be unimodal. This is a correct estimate as the third dimension was constructed as uniformly distributed noise. If we assume that a unimodally distributed characteristic is essentially not of great interest, then we could eliminate this dimension and reduce the running example to a two-dimensional data set. This two-dimensional data set would then be treated as explained in this paper. At some point, however, it might happen that the dip test finds another unimodal feature to be and the data set can be further reduced.

According to this roadmap, the DipTransformation could be converted into a technique that improves the structure of a data set while reducing dimensionality. We intend to do this in the near future.

VI. CONCLUSION

In conclusion, we can say that we have achieved our goal of creating a technique that can improve the structure of a data set and thus its clustering. We have shown that this statement is true by testing it extensively on various data sets.

For k-means, which was the main focus, this is now particularly clear. On the tested data sets, k-means was usually a sub-ideal choice and other algorithms were clearly better. After the DipTransformation was k-means the best-performing algorithm on all but one data set.

We have also shown that DipTransformation is compatible with other algorithms and also improves their clustering results. DipTransformation can therefore be used as a pre-clustering step, that enhances the data set, and the clustering algorithm can be chosen as the user likes best.

REFERENCES

[1] Aggarwal, C. C., Procopiuc, C., *Fast Algorithms for Projected Clustering*, SIGMOD, 1999.

[2] Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P., *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*, SIGMOD, 1999.

[3] Arthur, D., Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA, 2007.

[4] Böhm, C., Plant, C., Shao, J., Yang, Q., *Clustering by synchronization*, KDD, 2010.

[5] Chawla, S., Gionis, A., *k-means--: A unified approach to clustering and outlier detection*, ICDM, 2013.

[6] Dempster, A. P., Laird, N. M., Rubin, D. B., *Maximum-Likelihood from incomplete data via the EM algorithm*, Journal of the Royal Statistical Society, 1977.

[7] Dua, D., Karra Taniskidou, E., *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2018.

[8] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *A density-based algorithm for discovering clusters in large spatial databases with noise*, KDD, 1996.

[9] Goebl, S., He, X., Plant, C., Böhm, C., *Finding the Optimal Subspace for Clustering*, ICDM, 2014.

[10] Hartigan, J. A., Hartigan, P. M., *The Dip Test of Unimodality*, The Annals of Statistics, 1985.

[11] Kalogeratos, A.,Likas, A., *Dip-means: an incremental clustering method for estimating the number of clusters*, NIPS, 2012.

[12] Krause, A., Liebscher, V., *Multimodal projection pursuit using the dip statistic*, Preprint-Reihe Mathematik, 2005.

[13] Kriegel, H. P., Kröger, P., Zimek, A., *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering*, TKDD, 2009.

[14] MacQueen, J. B., *Some methods for classification and analysis of multivariate observations*, Berkeley Symposium on Math. Stat. and Prob., 1967.

[15] Maurus, S., Plant, C., *Skinny-dip: Clustering in a Sea of Noise*, KDD, 2016.

[16] Mautz, D., Ye, W., Plant, C., Böhm, C., *Towards an Optimal Subspace for K-means*, KDD, 2017.

[17] Ng, A., Jordan, M., Weiss, Y., *On spectral clustering: Analysis and an algorithm*, NIPS, 2002.

[18] Pelleg, D., Moore A. W., *X-means: Extending K-means with Efficient Estimation of the Number of Clusters*, ICML, 2000.

[19] Sibson, R., *SLINK: an optimally efficient algorithm for the single-link cluster method*, The Computer Journal, 1973.

[20] Silva, P., Marcal, A., Almeida da Silva, R., *Evaluation of Features for Leaf Discrimination*, Springer Lecture Notes in Computer Science, 2013.

[21] Vinh, N. X., Bailey, J., *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance*, JMLR, 2011.

[22] Yang, B., Fu, X., Sidiropoulos, N., Hong, M., *Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering*, ICML, 2017.

[23] Ye, W., Goebl, S., Plant, C., Böhm, C., *FUSE: Full Spectral Clustering*, KDD, 2016.

[24] Xie, J., Girshick, R., Farhadi, A., *Unsupervised Deep Embedding for Clustering Analysis*, ICML, 2016.

[25] Zelnik-Manor, L., Perona, P., *Self-Tuning Spectral Clustering*, NIPS, 2004.

Appendix C

Paper C: Dataset-Transformation: Improving Clustering by enhancing the structure with DipScaling and DipTransformation

Dataset-Transformation: Improving Clustering by enhancing the structure with DipScaling and DipTransformation

Benjamin Schelling[1] and Claudia Plant[12]

[1] Faculty of Computer Science, Research Group Data Mining,
University of Vienna, Vienna, Austria
[2] ds:UniVie, Vienna, Austria

Abstract. A data set might have a well-defined structure, but this does not necessarily lead to good clustering results. If the structure is hidden in an unfavourable scaling, clustering will usually fail. The aim of this work is to present techniques - DipScaling and DipTransformation - which enhance the data set by re-scaling and transforming its features and thus emphasizing and accentuating its structure. If the structure is sufficiently clear, clustering algorithms will perform far better. We refer to such techniques as "Dataset-Transformations" and try to provide a mathematical framework for them. To show that our algorithms work well, we have conducted extensive experiments on several real-world data sets, where we improve clustering not only for k-means, which is our main focus but also for other standard clustering approaches.

1 Introduction

The clustering of a data set is strongly dependent on the structure it contains. If there is hardly any structure or if the structure is well hidden, clustering will most likely fail because the boundaries between the clusters are hard to determine. A strong and clearly defined structure usually leads to significantly better clustering results. Accentuating the structure would, therefore, be useful for clustering, but to the best of our knowledge, there are currently no methods that are capable of doing so.

Confronted with a data set one cannot quite cluster, one would usually proceed by creating a new clustering method which is capable of dealing with the new and problematic type of data set, but this is not the approach we chose here. Instead, we wish to create methods which transform the data set - by enhancing the already existing structure - such that established clustering methods can deal with it. We present here DipScaling and DipTransformation[3], which are capable of accentuating structure and bringing the data set into a more cluster-able form.

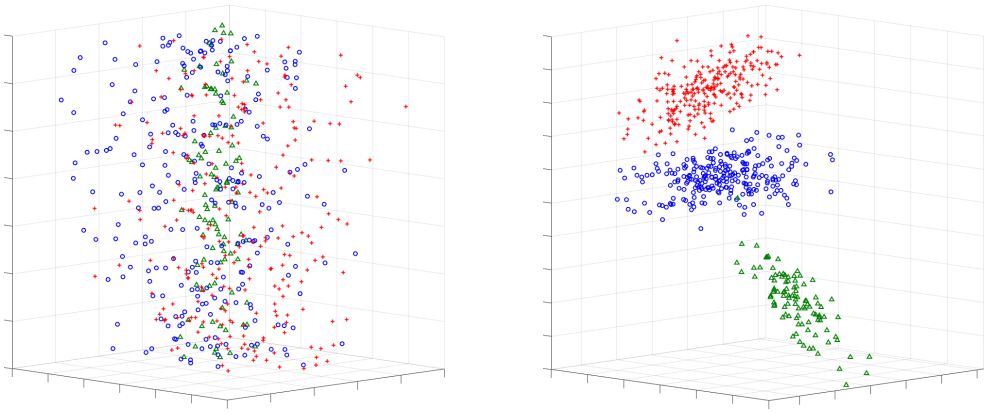
Consider the data shown in Fig. 1a as a 3D scatter-plot of our running example. It is actually not a complicated data set, consisting of three stretched Gaussian distributed clusters, with different rotations and a third dimension of uniformly distributed noise, which has about the same range as the clusters. The problem here is twofold: 1) The third dimension, which does not contain any structure, is given the same weight as the dimensions that contain the entire cluster structure. 2) The clusters, while not overlapping and with clear borders, are most unfavourably scaled.

The standard clustering algorithms yield surprisingly poor results on this data set. K-means scores merely 0.01 in NMI (Normalized Mutual Information)[4], DBSCAN (Ester et al., 1996), Spectral Clustering (Ng et al., 2002) and SingleLink (Sibson, 1973) also perform disappointingly. The best choice appears to be EM (Dempster et al., 1977) with an NMI score of 0.43.

Since the data set consists of a superfluous third dimension, we try dimensionality reduction techniques in the hope of adapting the dataset into a more clusterable form. The combination of clustering and dimensionality reduction is well established and might yield better results here (see (Kriegel et al.,

[3] Source code and data sets: https://dm.cs.univie.ac.at/research/downloads/

[4] Normalized Mutual Information (NMI) (Vinh et al., 2010) is one of the most often used evaluation measures for clustering. NMI is scaled between 0.0 and 1.0, with 0.0 the worst possible score and 1.0 the best.



(a) Our running example through this paper shown as a 3D scatter-plot.

(b) Our running example after DipTransformation in a 3D scatter-plot. It is now far easier to cluster.

Fig. 1: The running example before and after the DipTransformation.

2009) for more details on this). However, neither PCA (0.03 in NMI) nor ICA (0.01 in NMI) lead to a data set that can be clustered with k-means. The best choice seems to be t-SNE (Van der Maaten and Hinton, 2008) which scores approximately 0.78, but has highly varying results. (All these techniques were used in combination with k-means with correct k. The average of 100 runs is given.) The clusters are purely in the first two dimensions - so techniques like PROCLUS (Aggarwal et al., 1999) and CLIQUE (Agrawal et al., 1998) which search for clusters in axis-parallel subspaces could be successful, but our experiments show otherwise (0.21 and 0.71 in NMI, correct k for PROCLUS).

DipScaling makes it possible to compensate for the unfortunate scaling of the features. We briefly stated that the problem lies partly therein, that uniform/unimodal features (i.e. essentially structure-free features) receive the same degree of attention as such features that deviate from it. Uniform/unimodal features have no visible cluster-structure. Clusters are not distinguishable there and, thus, these features are difficult to cluster. Multimodal features, on the other hand, have clearly separated clusters. They are very important for clustering as the clusters can be found far easier. For k-means, which is our main focus here, this implicates that features with more structure should be larger scaled compared to features with barely any structure. If a feature is scaled to a very small range, the data points are almost the same in regard to this feature and, thus, this feature will have a very small impact. A very large scaling of a feature, on the other hand, means a high impact; the k-means clustering will be heavily influenced by those structure-rich, multimodal features in computing the centres of the clusters and the way the clusters are determined.

This requires a measure that evaluates the amount of structure of a feature and, therefore, its scaling. We find this in the Dip-test (Hartigan and Hartigan, 1985) explained in Section 2.1. The Dip-test gives an appropriate measurement of the structure a feature has and, thus, the scaling it "deserves". In Section 2.2 it is explained in more detail how this measurement is used.

DipScaling rescales the axes-parallel features. Restricting oneself to only the axes, however, is basically the same as assuming independence of the axes-parallel features, which is not necessarily the case. **DipTransformation** expands on DipScaling and generalizes it, so that this assumption is no longer needed. It is capable of handling data sets where the clusters are not necessarily axes-parallel, such as our running example. DipTransformation searches for multimodal features that are non-axes parallel and then, when found, apply the strategy of DipScaling. DipTransformation is capable of transforming our running example into a form that is almost perfectly cluster-able with k-means. The

clusters are better separated from each other and the structure of the data is more pronounced (see Figure 1b). K-means now reaches an NMI of 0.97.

1.1 Contributions

This work presents a parameter-free method as well as an almost parameter-free method - DipScaling and DipTransformation - which are capable of improving the structure of a data set and thus allowing for better clustering. Our main focus lies with k-means, but this also holds for other methods as we will show for the standard clustering approaches. DipScaling and DipTransformation do not assume a specific distribution for the clusters or data. They simply enhance structure and thereby improve clustering. They are both deterministic and require no distance calculations. We extensively tested on real world data sets for a wide range of algorithms.

1.2 Related Work

The most common approach when a data set cannot be clustered well by any clustering algorithm is to create a new algorithm that can handle that data set. The reverse approach of adapting the data set to the algorithm is the much more unorthodox approach. It is usually only done in the simplest way, i.e. by normalizing a data set, such as rescaling it into the [0,1]-interval. In addition, there is the Z-transformation (sometimes referred to as Z-normalization), which rescales the axes-parallel features to a mean of 0 and a variance of 1. Z-transformation is also relatively conventional but is already applied far less often. Apart from these two methods, however, we are not aware of any approaches that attempt to adapt a data set with the aim of enhancing structure for improved clustering. Of course, there are techniques that try to improve clustering, for instance, k-means++ (Arthur and Vassilvitskii, 2007), which provides an initialization strategy for k-means that is often very successful, but transforming a data set is unusual. One might consider SynC (Böhm et al., 2010) as a transformation technique because it collapses clusters into single points using the principle of synchronization.

Subspace clustering techniques such as the aforementioned PROCLUS and CLIQUE can be considered related work since they intend to reduce dimensionality, i.e. adapt the data set by removing "unnecessary" information. The DipTransformation does not remove any information, but - as the analysis of the running example will show - it is very capable of dealing with such noise information. Of particular interest are FOSSCLU (Goebl et al., 2014) and SubKMeans (Mautz et al., 2017) which intend to reduce dimensionality with the goal of finding a subspace compatible with k-means.

We are also aware of progress in the field of Deep Learning, where techniques such as DEC (Xie et al., 2016) and DCN (Yang et al., 2017) are being developed, with the aim of finding good subspaces using neural networks.

Spectral clustering takes a data set and transforms it into a distance matrix, computes its eigenvectors and applies (mostly) k-means to the data set. It is not necessary to use k-means, other partitioning algorithms can also very well be used. In this regard, spectral clustering techniques are similar to DipScaling and DipTransformation. They try to transform the data set into a more clusterable form. One of the most well known is the fundamental technique by Ng, Jordan and Weiss (Ng et al., 2002). We also use the popular Self-Tuning Spectral Clustering (Yang et al., 2008) and FUSE (Ye et al., 2016) as state-of-the-art comparison methods.

DipScaling and DipTransformation use the Dip-test for measuring structure and, therefore, one can consider all data mining-techniques that use the Dip-test as related. It was first used in data mining by DipMeans (Chamalis and Likas, 2018) with the goal of estimating the number of clusters for k-means. After that, there is SkinnyDip (Maurus and Plant, 2016), a technique to cluster in the presence of noise, using the Dip-test. Very recently has (Siffer et al., 2018) also been published, a generalization of the Dip-test to higher dimensions. It is used as a tool to find out if there are multiple clusters in a data set or not to decide if clustering makes sense. Hence, it is a clustering support technique like ours. The Dip-test is still a rather unknown tool, that has not yet found full recognition.

One must bear in mind while reading this, that DipScaling and DipTransformation are not rivals for the mentioned techniques in the classical sense, but that they can be used as supporting techniques that ease the difficulty in the task they attempt. We will show in the experimental section (see Section 5) that they can benefit from DipScaling and DipTransformation.

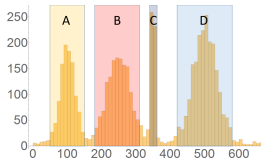
2 The Algorithm

To understand how the algorithm works, we must first go into detail about the Dip-test.

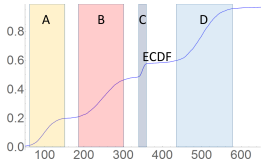
2.1 The Dip-Test

The Dip-test was created by Hartigan & Hartigan in the 1980s as a measure of how much a one-dimensional sample deviates from unimodality. Unimodality is defined here as a distribution that is convex until it reaches its maximum and concave thereafter.

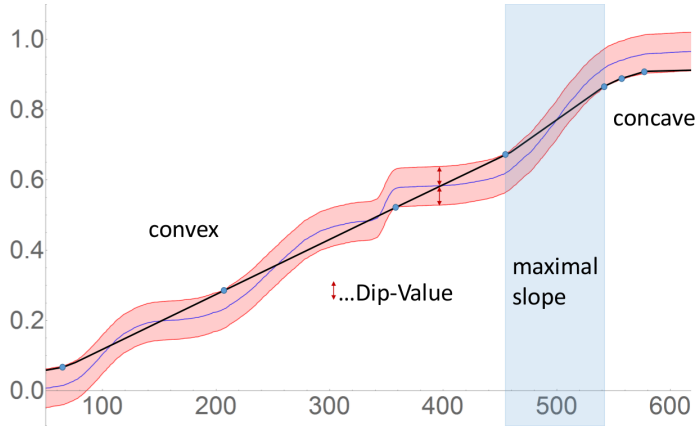
The test starts with sorting the sample and then creating the Empirical Cumulative Distribution Function (ECDF). This can be seen in Fig. 2. The histogram shows 4 clusters (A, B, C and D), which can be clearly identified in the ECDF to its right. The Dip-test only requires the ECDF; the histogram is only for visual clarity. It, therefore, has no bin-width parameter. In fact, it has no parameter at all.



(a) A one-dimensional sample from a data set with 4 obvious clusters.



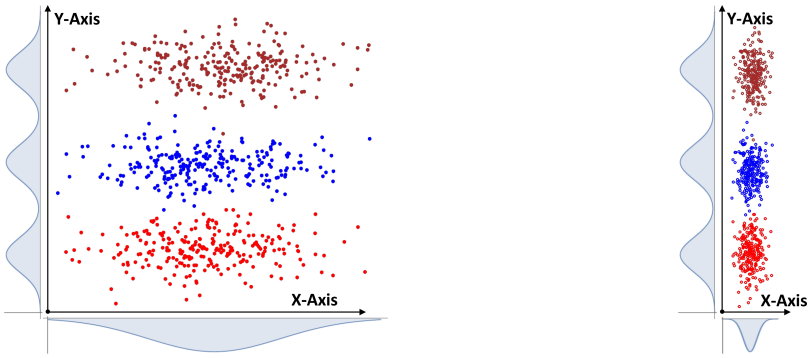
(b) The ECDF of the data set from Fig. 2a.



(c) The Dip-test measures how far away a ECDF is from a unimodal distribution.

Fig. 2: A histogram and the resulting ECDF (empirical cumulative distribution function). The Dip-test uses the ECDF to find out how much it deviates from a unimodal distribution, i.e. how big the offset is to fit a unimodal distribution. The offset is the dip-value.

The Dip-test measures the extent to which the ECDF deviates from unimodality. It computes how much the ECDF has to be offset, so that it can fit a unimodal distribution. This can be seen in Fig. 2c. The ECDF has been shifted vertically by a certain value (the dip value), and ECDF+dip and ECDF-dip is plotted there. This offset is large enough so that a line can be drawn in between ECDF+dip and ECDF-dip, which is first convex and then concave. This line is representating the closest possible unimodal distribution. The dip statistic, (we refer to it as the dip-value or "dip") shows how far away the ECDF is from such a unimodal distribution. It can be understood as the distance of the sample to a unimodal distribution.



(a) The data set and the projections onto its axes to show the structure found there.

(b) The axes are now scaled according to the found structure.

Fig. 3: A simple data set before (a) and after (b) applying DipScaling.

The Dip-test also returns another value, the probability of how likely a sample is unimodal, as well as the interval of the highest slope, but we only need the dip-value. The dip-value is always in the interval $(0, 0.25]$, hence it is always positive.

The Dip-test itself has a runtime of $\mathcal{O}(n)$, but since its input must be sorted to create the ECDF, the effective runtime for this part of the technique is $\mathcal{O}(n \log n)$. Further details about the Dip-test can be found in (Hartigan and Hartigan, 1985).

2.2 Applying the Dip-Test

The Dip-test gives a value, the dip value, which estimates how much structure can be found in a feature. We stated in the introduction that we use this estimation to re-scale the features and thus improve clustering. For us, the Dip-test measures the influence the feature should have. The more influence it is supposed to have, the larger it will be scaled (in relation to the other features) and the greater is its importance in determining the result of k-means.

Let us explain the approach with Fig. 3. The figure shows a very simple data set consisting of three Gaussian distributed clusters. Even though the data set is very simple and should be easy to cluster for k-means (non-overlapping, Gaussian clusters), k-means performs rather poorly. The problem is obvious: the scaling of the clusters is very unfavourable. Thus, we rescale the features to make the structure of the data set more accessible. We restrict ourselves to re-scaling the axes-parallel features, i.e. we stay in the framework of DipScaling.

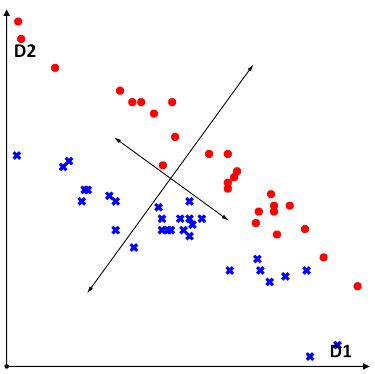
The first step is to analyse the features. For that, we project the data onto the axes. These projections can be seen in Fig. 3. The Y-axis has a clear cluster-structure, i.e. three different clusters can be identified, while the X-axis is completely without structure, basically the clusters are completely undistinguishable. Measuring the amount of structure gives us a value of 0.009 for the X-axis and 0.063 for Y-axis. We rescale the X-axis in the interval [0,0.009] and the Y-axis in [0,0.063] and get the dataset represented in Fig. 3b. The feature with the structure is now scaled comparatively large and has a high impact in clustering, just as we wanted.

This change makes this data set now easily cluster-able by k-means. The dimension containing the structure is now much more pronounced and accordingly more influential for k-means. The improvement of the clustering result is best described using the Normalized Mutual Information (NMI) (Vinh et al., 2010) value, which increases from an average value of 0.55 for the unscaled data set to 0.98 for the rescaled data set (100 random initialisations each). The only error and the reason why an NMI-value of 1.0 is not reached is due to some edge-data points that have been falsely assigned, but could not reasonably be expected to be correctly clustered.

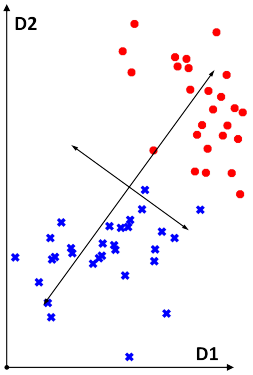
This (rather trivial) example shows how important it is to enhance the structure of a data set. The horizontal axis in which the data set has barely, if any, structure is reduced to a very small range and the vertical axis, where the clusters and structure are located, is now the relevant dimension that determines the result of k-means.

2.3 DipScaling

We have seen in Fig. 3 the effect that re-scaling the features of a data set can have. The Dip-test is used to obtain an estimation of the amount of structure contained in a feature and this estimation - the dip value - is used for rescaling the feature. We refer to this procedure as DipScaling. For this, it is necessary that the original features can be handled separately, i.e. they are independent of each other. If they were in some way dependent, i.e. correlated, on each other the clusters would not be parallel to the axes. This is of course a very significant assumption, which is not always justified and which is not true for every data set. DipTransformation does not share this restriction. DipScaling is the foundation on which DipTransformation builds upon; it can be used on its own, as a very fast, first re-shaping of the data set, very much like Normalisation and Z-Transformation. We will see in Section 5 that it can improve clustering, but most of the time DipTransformation is the better choice.



(a) The Whiteside-data set.



(b) Whiteside-data after scaling in the direction of the maximal dip value with the respective dip value.

Fig. 4: Shown is also the direction of the feature with the maximal dip-value, as well as its orthonormal direction. These directions are scaled with their dip values to show how much structure is found in these directions.

2.4 DipTransformation

If features are not independent, clusters are not stretched parallel to the axes. Let us look at the Whiteside-data set (Hand et al., 1993) (depicted in Fig. 4a) as such a case. The Whiteside-data set is a real-world data set.

The clusters are obviously not axes-parallel. We have the same situation here as in the simple data set shown in Fig. 3, as in that k-means fares rather badly. To be precise the NMI-value is 0.006. DipScaling is ineffective here; it improves clustering only minimally. The situation requires an approach that can find features, which are not axes-parallel, but are interesting in regard to their dip value. In the data set shown in Fig. 3, this is not a relevant aspect, as the feature with a high dip value is axes-parallel, but here this is not the case. The axes-parallel features of the Whiteside data are very

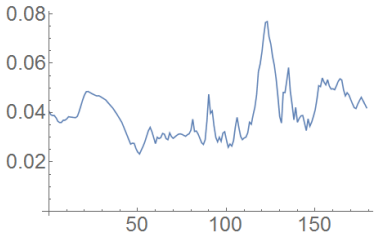


Fig. 5: The brute force approach to finding the maximal dip value is to measure the dip value at as many angles as possible. This graph shows the dip values of the Whiteside-data set and how they change depending on the angle at which they are measured. Basically, the data set is rotated by, say, 10°, the data set projected onto a axis, and the dip value of the now one-dimensional sample computed.

uninteresting, i.e. they have a low dip value. But, if we project the data onto a straight line at an angle of roughly 123°, we find a feature with a very high dip value. Following our earlier arguments, we are very interested in these feature, as they most likely contain the structure that determines the quality of a clustering.

In Fig. 5 we see that the dip value changes notably depending on the angle at which the dip value is measured. While the dip value changes continuously, it is full of local minima and maxima. Let us assume, that we found the angle of the maximal dip value and scaled the Whiteside data set with this dip value in the direction of the angle. This leads to the transformed data set shown in Fig. 4b. A data set that can be clustered considerably better by k-means, due to the clearly more accessible structure of the data. In fact, we get an average NMI of 0.92. This is a massive improvement to the previous NMI of 0.006, but it is possible to improve even further, as we will see.

Determining the angle of the maximal Dip-value is not straightforward. Of course, one could use the brute-force approach of simply testing as many angles as possible, but this will prove impractical at the latest when the data set is of higher dimensionality. A too simple search algorithm for the angle with the highest dip value will also not lead to a satisfactory result, since the data, as we have seen in Fig. 5, has more than a few local optima. Our solution to this predicament is to not only search for a single angle with maximal dip-angle, but to find multiple angles with a high dip value and scale the data set along them. Basically, DipTransformation does not restrict itself to a single re-scaling like DipScaling, but finds multiple interesting features. The algorithm scales the data set in various instances, so that the structure of the clusters becomes more clearly defined and more cluster-able.

A search strategy to find interesting angles, i.e. with high dip values, is needed, but instead of changing the angle and analysing the projection, we rotate the data set and analyse the features that are now the axes. This is basically the same approach, but simplifies some computations. The procedure starts by computing the dip values of the axes, or more precisely the projection of the data set onto the axes. We then have some information about the "landscape" of the dip values depending on the angle. For the Whiteside-data set the landscape is shown in 5. The difficulty is that the absolute maximal dip value depends on the specific data set. For Whiteside it is around 0.08, but other data sets may have a very different maximal dip value. Hence, we cannot transfer any expectation from one data set to another. Nevertheless, we have the information from the axes and this gives us a starting point. For example, for the Whiteside data set we know the two dip-values, D_1 and D_2, along the axes. The ratio between those values $r = Max(\frac{D_1}{D_2}, \frac{D_2}{D_1})$ tells us something of about the "quality" of these features. If r is high, then there is a good chance that we have hit a good dip-value. After all, high r means that either D_1 or D_2 is quite larger than the other, i.e. it is a feature with a high dip value, which we are looking for. We continue by rotating the data set in clockwise direction by an angle of $\frac{1}{r} * c$, with c as a rotation speed parameter. This ensures that we rotate the data set only by a small angle if one of the axes has a high dip value. Since the dip value changes continuously, it is quite possible that the axis is close to an angle with even higher dip value. On the other hand, if r is small, then both axes

have similar dip value, thus, they are not interesting for our search of high dip values and the direct neighbourhood is probably not very interesting as well. Rotating by $\frac{1}{r} * c$ will lead to a larger leap and the uninteresting area is skipped.

The rotation speed parameter c can be freely selected. The larger c is selected, the shorter is the runtime, but a smaller c of course makes it more likely to find high dip-values. (Throughout the paper, the rotation speed parameter c is set to 5. Its effects are further explored in Section 2.6.)

The algorithm remembers the maximal found dip value (we refer to it as $MaxDip$) and every time it finds a new maximal dip value, the axes are scaled with their respective dip values. The total degrees the data set has been rotated by is stored and after 360° the algorithm stops. For the Whiteside-data set with $c = 5$ this leads to the transformation displayed in Fig. 6.

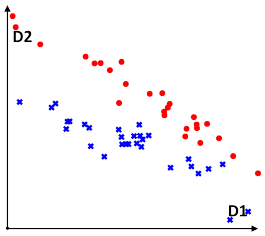
At the beginning of the transformation, the changes are comparatively small. This is because the dip values of the axes are not very different and therefore scaling the axes only leads to limited changes. $MaxDip$ is updated, whenever a new maximal dip value is found. In the following iterations the maximal dip value of the current axes is not greater than $MaxDip$ and hence no scaling takes place. However, after several rotations, the maximal dip value of the axes is greater than $MaxDip$ and the data set is scaled again. This happens in each step of Fig. 6. The step from Fig. 6c to Fig. 6d seems to be enormous, but all that happens here is that we come very close to the overall maximal dip value, when rotating. The dip values of the axes are now very different and the scaling leads to a seemingly very much changed data set, but if closely examined, one sees that that all that happens is that the data set is stretched. The dip value of the axis with the clusters is rather large (the cluster structure is located here), while the other axis has a small dip value. Scaling the axes with the dip value causes what we wanted: A data set where the cluster structure is far more obvious. We are now rather close to the data set-transformation from Fig. 4b.

The algorithm is not yet finished. It remembers the new $MaxDip$. The data set continues to rotate, but because we are very close to a high value the algorithm selects only a small rotation angle. This is advantageous because the next dip value of an axis is even higher. Scaling with these dip values leads to Fig. 6e, with an even more pronounced structure. The next iteration(s) change only very little. The data set does not change drastically, but becomes more compact and the clusters are defined more clearly with each iteration. Fig. 6g shows the final state of the data set after the DipTransformation.

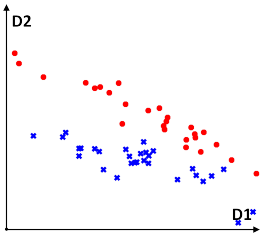
All in all, we can say, that the proposed search strategy works very well, and we are very good at finding interesting angles with high dip values. Thus, the resulting transformation of the Whiteside-data is very easy to cluster for k-means. We get an average NMI-value of 1.0 (in 100 iterations), which means that the data set is perfectly clustered. This result is not specific to a value of $c = 5$, but can also be reached by e.g. $c = 9, 8, 7, 6, ...$ However, the transformed data set may look slightly different for a different value of c.

This transformation is easier to cluster in comparison to the original Whiteside-data set shown in Fig. 4a, but also when comparing it to the transformation along the maximal Dip-value angle shown in Fig. 4b. One could have expected these transformations to be more similar, if not identical, but that is not the case. The transformation here is not along an orthogonal basis, as the one there. Scaling along axes leads to a basis transformation that stretches the basis vectors, but leaves orthogonality intact. Applying the transformation method sketched above also leads to a change in basis vectors, which no longer implies that two previously normal (i.e. perpendicular) vectors are normal to each other afterwards.

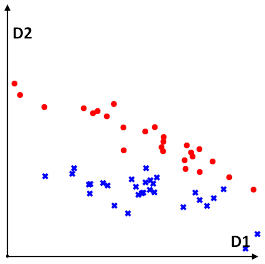
The foci of DipScaling and DipTransformation are on k-means as we have stated, but we see from Fig. 6 that other techniques might also benefit from this approach. We will explore in Section 5 how the performance of other techniques is influenced by this (and other) transformed data set(s).



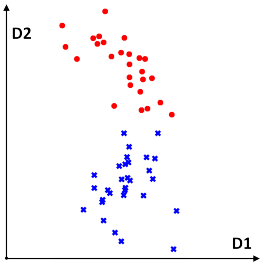
(a) The Whiteside data after the first rotation.



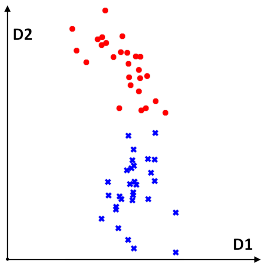
(b) Small changes become visible.



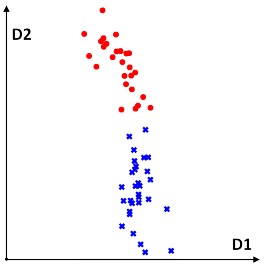
(c) The horizontal axis is now very close to a very large dip value.



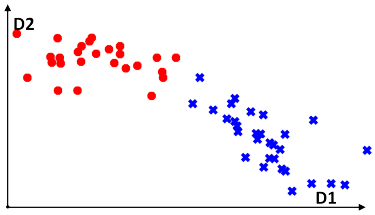
(d) The maximal dip value is now almost found. Rescaling the data is very influential.



(e) The maximal dip value comes even closer. The data set is only rotated by a very small angle.



(f) The maximal dip value is found.



(g) The final transformation.

Fig. 6: Steps in the DipTransformation of the Whiteside data set (a) - (f) and the final data set (g). There seem to be big leaps in the transformation; this is due to those cases, when the dip values of the axes become very different and scaling the data set seems to lead to enormous changes.

2.5 More than 2 Dimensions

DipTransformation for a 2-dimensional data set was explained in detail, because it forms the basis for data sets with more than two dimensions. There are several ways to adapt this approach; the one we found to work best is explained in the following:

The main difference is that there are more than two directions, the data set can be rotated in. It seems logical to rotate the data set in all directions at once following the angle-computation as before, but there is a problem involved with that: rotations are not commutative. That means, it makes a difference in which order the rotations are executed. Finding only one non-axes parallel angle in which the data set is rotated, is anything but straightforward, since all we have are the dip values of the axes, that we can use to compute axes-parallel rotation angles. Due to the difficulties here, we simply avoid them by rotating only the 2D-subspace spanned by two axes-vectors. This has the advantage that we can simply take the earlier strategy and apply it to the 2D-subspace. The only remaining question is, which of the axes (respectively their spanned subspace) one should rotate first, but this seems to be a decision without much influence, according to the experiments we conducted. The order seems to have limited influence, as long as all directions are covered. Executing only the rotation with the highest/lowest dip value, for example, seems to impair the transformation. Thus, a strategy that alternately rotates in all direction seems to be the best choice.

Through all rotations the algorithm remembers the maximal found dip value as $MaxDip$, just as before. Whenever the rotated data set has a dip value larger than $MaxDip$, the data set is scaled and $MaxDip$ is updated. The rotation angles are calculated in the same way as for a 2-dimensional data set.

One has to keep in mind that in higher dimensional data sets, the algorithm has a larger area to search for high dip values. It is only a "half-circle" or 180° that needs to be traversed for a 2-dimensional data set. (In the interest of precision, however, the algorithm looks over the full 360°.) For a $d-$dimensional data set, it would be half of a $d-$dimensional sphere. To compensate for this, the algorithm assumes that $d{\cdot}360°$ have to be traversed. This range ascertains that (theoretically) all maxima can be found. Furthermore, it is not necessary to find the maximal dip values exactly; being close enough is sufficient to assure a good transformation.

2.6 The Rotationspeed Parameter c

The rotation speed parameter c has been introduced in Section 2.4 and we now want to analyse its effect on the DipTransformation. Fig. 7 shows how NMI (for k-means) changes with different values of c and the effect is (mostly) not very pronounced. The data sets depicted were chosen, because their values do not overlap, but the effect is rather similar for all data sets examined in Section 5.

There is a slight tendency for the clustering results to lose quality at higher values of c (best visible for the Whiteside data set), but it is not very pronounced. However, for an unknown data set a smaller value for c is recommended. DipTransformation is dependent on finding the high dip-values and that is simply more likely for smaller rotation speed. If runtime is the primary factor (for example, for very large data sets), then a larger value for c might be more recommendable. Fig. 8 shows how the runtime is linked to the parameter c and how it decreases for larger c. There is a (small) trade off between clustering quality and runtime, but as a general rule a high value of c seems not too detrimental. For this work, however, we stick to the fixed value $c = 5$.

3 An Attempt at a Mathematical Framework for Dataset-Transformations

We have already mentioned that there are basically no dataset-transformation-based techniques, although we have never clearly stated what we consider to be such a transformation. This could be due to the apparent obviousness of what it means; we would nevertheless like to try give a more formal definition.

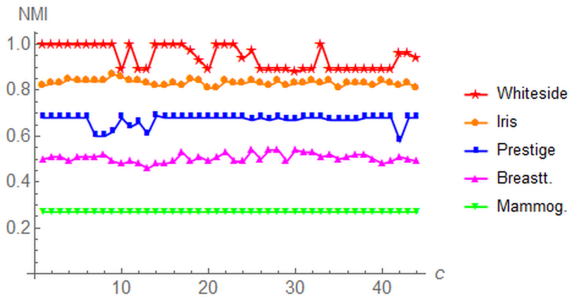


Fig. 7: The NMI-value for k-means with correct values for k vs the rotation speed parameter c for five real world-data sets.

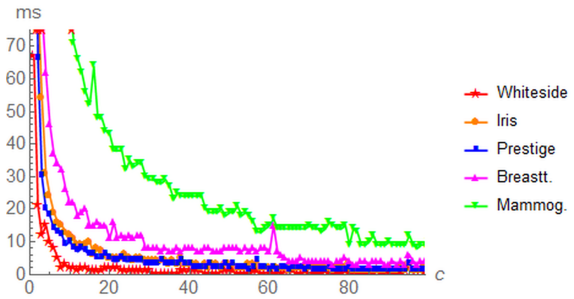


Fig. 8: The runtime of DipTransformation in milliseconds depending on the value of the rotation speed parameter c for five real world-data sets.

The most obvious attempt is to just restrict Dataset-Transformation to continuous Transformations, i.e.:

Definition 1. *Let D be a data set, consisting of n $d-$dimensional data points $x_1, \ldots, x_n$. The operator T is a Dataset-Transformation, if $\forall \epsilon > 0 \quad \exists \delta > 0$, such that, $\forall i, j$ if $||x_i - x_j|| < \delta$, then $||T(x_i) - T(x_j)|| < \epsilon$.*

This is basically the straightforward application of the definition of continuity to our problem, but the problems are obvious. The data set is necessarily a finite one, so the restriction of $\forall \epsilon > 0 \quad \exists \delta > 0$ cannot apply here, as one can always make δ so small that the $||x_i - x_j|| < \delta$-requirement holds for no data point and is thus trivially fulfilled. Following this line of thought, every operator on a finite set can be considered continuous and this is too broad. One can extend and vary this approach, but the main problem of D being finite remains and foils all approaches that restrict themselves to D. Eventually, the only approach left is to lift the restriction onto D and consider a potential Dataset-Transformation T as an operator on $\mathbb{R}^d$, with $D \subset \mathbb{R}^d$. This is of course a restriction as D can now be merely numerical in nature, but one has to consider that the Dip-test needs numerical values any way. It might also be possible to not restrict oneself to euclidean spaces, as is done here, but e.g. Banach-spaces instead. An advantage won by that though is not discernible. If such a generalization would become necessary at a future point in time it can be easily obtained on this foundation, but for the moment we restrict ourselves to euclidean spaces and accept that basically all available numerical data sets would be considered subsets of $\mathbb{R}^d$ anyway.

With the decision made that a Dataset-Transformation warps the space itself and with it the data points, we can formulate the definitions we wanted to obtain. The most obvious attempt would be a

continuous transformation of the space, hence:

Definition 2 (Dataset-Transformation). *Let D be a data set, consisting of n $d-$dimensional data points $x_1, \ldots, x_n$. The operator $T : \mathbb{R}^d \mapsto \mathbb{R}^m$, with $m \leq d$ and $D \subset \mathbb{R}^d$ is a* ***Dataset-Transformation****, if it is continuous.*

This by itself is in excess of what is actually needed. If, for example, all data points would contain only positive values a restriction to $\mathbb{R}^d_+$ would be enough. Nevertheless, we stick with this definition for now. This raises the question if DipScaling and DipTransformation can be considered as Dataset-Transformations.

Theorem 1. *The DipTransformation DT is a linear operator. More precisely, it is a basis-transformation.*

Proof. Every rotation in $\mathbb{R}^n$ can be expressed as a matrix R. Scaling a data set in $\mathbb{R}^n$ simply means applying a diagonal matrix S with the scaling-parameters in the main diagonal. Hence, applying the DipTransformation on a data set is equivalent with applying the rotation- and diagonal-matrices R_1, S_1, R_2, S_2, R_3, S_3,.... Thus, the DipTransformation DT is the product of matrices, which is again a matrix. A matrix is a linear operator, hence the DipTransformation is a linear Operator.
A rotation is an orthogonal matrix with determinant 1, a scaling matrix has the determinant $c_1 \cdots c_n$, with c_i being the entries in the diagonal. Since the Dip-test values c_i can never be zero, the determinant of the scaling matrix is non-zero. The determinant has the property $Det(A \cdot B) = Det(A) \cdot Det(B)$, hence the determinant of the DipTransformation is $Det(DT) = Det(R_1) \cdot Det(S_1) \cdots Det(R_l) \cdot Det(S_l) = 1 \cdot (c_{11} \cdots c_{n1}) \cdots 1 \cdot (c_{1l} \cdots c_{nl}) \neq 0$. Thus DT is a matrix with non-zero determinant, i.e. a basis-transformation.

This shows that the DipTransformation (and therefore DipScaling, for which the proof holds as well) is continuous, as every basis-transformation is continuous, hence:

Theorem 2. *The DipTransformation DT is a* ***Dataset-Transformation*** *as defined in Def. 2.*

Defining a Dataset-Transformation as a continuous change of the data set seems to be the most straightforward and meaningful approach. Let us consider for example t-SNE (Van der Maaten and Hinton, 2008) as a technique that "transforms" a data set: It starts with the higher-dimensional data points, computes similarities and places the data points into a lower-dimensional space. While the technique intends to keep close data points close to each other, this is not guaranteed, and they can be put far away from each other. The data space is basically ripped apart and sometimes data points end up in a completely new neighbourhoods. A continuous transformation like DipTransformation guarantees that this cannot happen, which is why we considered it as the "most obvious attempt". The data set is distorted and distances changed, but the basic shape of the data set is kept.

This holds especially, if the dimensionality is kept the same. The definition with $T : \mathbb{R}^d \mapsto \mathbb{R}^m$, with $m \leq d$ does not rule out dimensionality reducing techniques like PCA. PCA can be understood as finding the vector with the highest variance in the data and projecting it onto the first axis, the vector with the second-highest variance onto the second axis, and so on. If the dimensionality is kept the same then PCA is basically a basis-transformation itself and, therefore, a Dataset-Transformation according to our definition. If the dimensionality is not kept the same, then PCA is nevertheless continuous (though not a basis-transformation any more) and thus a Dataset-Transformation. The same argument can be made for ICA. Normalization and Z-Transformation are obviously also covered by our definition, contrary to e.g. t-SNE, as mentioned earlier, and various methods like SynC.

We stated that the basic shape of the data set is kept, if the dimensionality is kept the same. If the dimensionality is kept, the transformation is also **bijective**; it is possible to reconstruct the original data set, if required. With a dimensionality reduction technique reconstructing the original data set is basically impossible as some information (however slight) is lost. This argument is based on DipScaling/DipTransformation being basis-transformations, thus it is not applicable to our definition of Dataset-Transformation generally. Including the requirement of bijectivity into the definition of a Dataset-Transformation is possible, as that guarantees that the effect could be undone, but continuity seems to be enough, as that means that the local neighbourhood of data point is (roughly) kept the same. This means that the structure is (roughly) the same, but - ideally - more pronounced.

Most clustering methods cannot be subsumed under our definition of a Dataset-Transformation, as they are not interested in transforming a data set. They simply want to find the clusters hidden in it, and are, therefore, not covered here. Our approach is different because we want to enhance and simplify a data set and we hope to give a first theoretic basis with this definition onto which future works can be placed. This basis is not complex, but it is a start and we have high hopes that it will prove itself as a stable foundation for an - hopefully - emerging new field in Data Mining.

4 Runtime and Pseudocode

The runtime of DipScaling is easy to estimate. First, one needs to compute the dip-values for every axis, which is $d \cdot \mathcal{O}(n \log(n))$, due to the necessity to order the input for the Dip-test, and the subsequent scaling of the axes with the values which is $d \cdot \mathcal{O}(n)$. Hence, as a total for the runtime for DipScaling we get

$$\mathcal{O}(d \cdot n \log(n)).$$

The DipTransformation algorithm (outlined in Algorithm 2) is slightly more complicated to estimate. Scaling of a data set as well as rotating a data set has a runtime of $\mathcal{O}(n)$; Computing the Dip-values is in the order of $\mathcal{O}(n \log n)$, since the values have to be sorted. There are two For-loops over d, where d stands for the number of dimensions. If the number of iterations in the while-loop is l, then the runtime can be estimated as:

$$\begin{aligned} &\mathcal{O}(n) + \mathcal{O}(n \log n) + l \cdot d \cdot d \cdot \big(\mathcal{O}(n) + \mathcal{O}(n \log(n)) + \mathcal{O}(n)\big) \\ &\approx \mathcal{O}\big(l \cdot d^2 \cdot n \log(n)\big) \end{aligned}$$

Experiments on the runtime can be found in Section 5.5.

Algorithm 1 DipScaling

Require: Data D
procedure DipScale(D)
  for i = 1,...,dim **do**
    Compute $DipValue$ for axis i
  end for
  for i = 1,...,dim **do**
    Rescale values of axis i to [0,DipValue(i)]
  end for
  return D
end procedure

Algorithm 2 DipTransformation

Require: Data D, Rotationspeed c

procedure DipTransformation(D, c)
  $Degree \leftarrow 0$
  Compute $DipValues$
  DipScale(D,$DipValues$)
  $MaxDip \leftarrow Max(DipValues)$
  while $Degree < dim * 180°$ **do**
    for i = 1,...,dim **do**
      for j = i+1,...,dim **do**
        $a \leftarrow Max(Dip(i)/Dip(j), Dip(j)/Dip(i))$
        Turn $D(i, j)$ by angle c/a
        $Degree \leftarrow Degree + c/a$
        Compute $DipValues$
        if $Max(DipValues) > MaxDip$ **then**
          DipScale(D,$DipValues$)
          $MaxDip \leftarrow Max(DipValues)$
        end if
      end for
    end for
  end while
  return D
end procedure

5 Experiments

Convincing someone that a data set is easy to cluster, if the data set is more than two-dimensional, is difficult. The goal of DipScaling and DipTransformation though is to ensure that a data set becomes easier to cluster. This work will of course show with NMI values of experiments on real-world data sets that DipTransformation is capable of doing that, but we would also like to show that with plots. Fig. 5.1 shows pairwise plots of the "Banknote Authentication" data set from the UCI-Repository (Dua and Graff, 2019). This data set was chosen because it has a small dimensionality of four, so that all pairwise plots can be shown. Fig. 5.1 illustrates the difficulty involved with clustering this data set. The clusters are not clearly separated and often overlap, so it is not suited for k-means. In numerical values this can be expressed with an NMI-value of 0.03 for k-means with the correct value for k. After the DipTransformation (shown in Fig. 5.1) the data set is much better structured and the clusters are well separated. K-means can now identify the clusters rather well. In fact, the NMI-value for k-means with the correct value for k is now 0.68. DipScaling has a far less impressive effect and improves clustering to a NMI-value of 0.18, which is not surprising as the clusters are obviously not axes-parallel, i.e. independent.

5.1 Synthetic Data

The first analysed data set is the synthetic data set given in Fig. 1a. This is a data set that - as we have seen - is quite difficult to cluster; k-means fares extremely bad and scores no higher than 0.01 in NMI. Other algorithms are often only marginally better. Table 1 shows the NMI-results. Most of them are not impressive, with 11 of the 20 algorithms scoring below 0.10. DipScaling leads to a somewhat better picture, but as the features are not independent, it is not a good choice. The DipTransformation on the other hands leads to a massively enhanced data set with clearly stronger defined structure. (Pairwise plot is shown in Fig. 1b.) The three clusters that were stretched and scaled quite unfavourable for clustering before are now well separated and compact. Clustering of this data set is far easier and the results shown in Table 1 demonstrate this. All of the used algorithms improve

Table 1: Clustering of the Running Example before and after DipTransformation. The average improvement in NMI is 0.636.

Running Example	before	after
k-means	0.01	0.97
k-means++	0.01	0.98
DipMeans	0.00	0.98
SkinnyDip	0.00	0.98
Spectral Clust.	0.36	0.97
STSC	0.00	0.99
FUSE	0.06	0.75
DBSCAN	0.23	0.95
SingleLink	0.01	0.96
EM	0.43	0.50
SubKMeans	0.08	0.63
FossClu	0.60	0.78
SynC	0.05	0.95
PCA + k-means	0.03	0.63
ICA + k-means	0.01	0.98
t-SNE + k-means	0.78	0.80
PROCLUS	0.23	0.92
CLIQUE	0.71	0.98
DEC	0.38	0.53
DCN	0.12	0.58

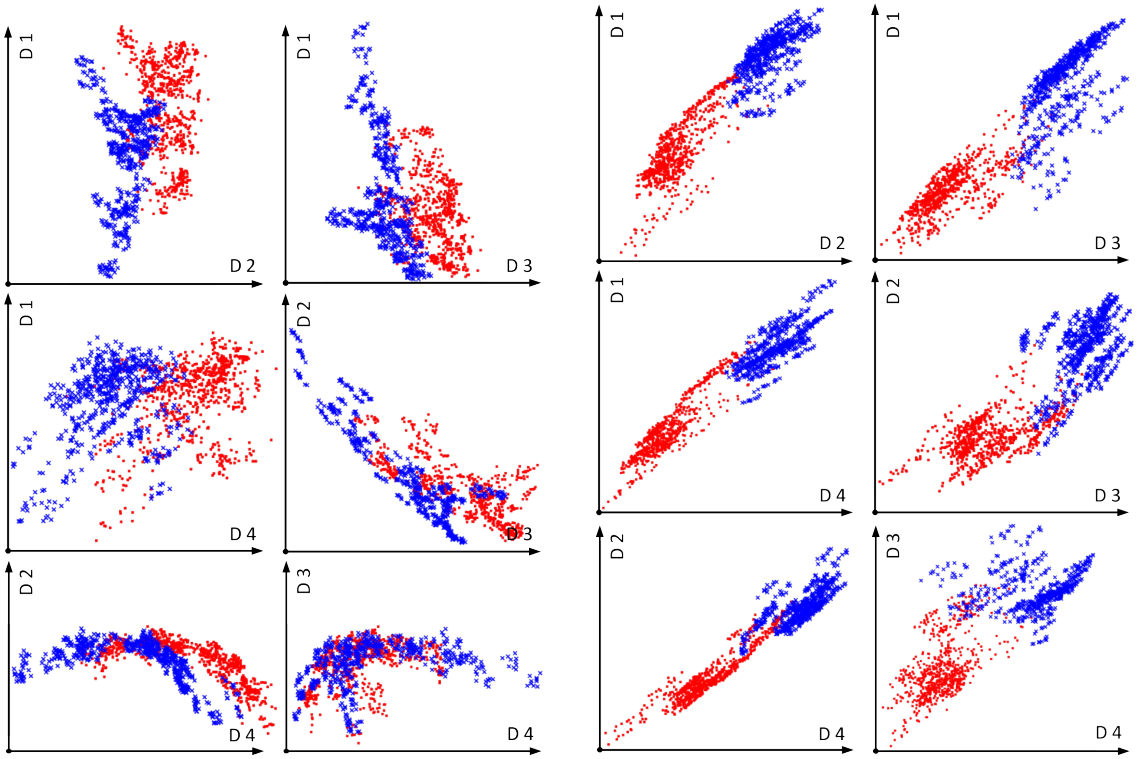


Fig. 9: Pairwise Plot of the Banknote-Authentication data set before and after DipTransformation. The dimensions are given as axes-label. The clusters are visibly better separated from each other after the DipTransformation, an effect which cannot be obtained with DipScaling alone.

Table 2: Experimental results. All non-deterministic results have been repeated 100 times and the average is given. Parameters as described.

Data set	Whiteside	Skinsegmen.	Banknote	Iris	Prestige	Userknow.	Mammographic	Seeds	Breast Tissue	Leaf
# of data points	56	245057	1375	150	98	258	830	210	106	340
# of dimensions	2	3	4	4	5	5	5	7	9	14
# of clusters	2	2	2	3	3	4	2	3	6	30
DipTransformation	**1.00**	**0.32**	**0.68**	**0.84**	**0.68**	**0.53**	0.27	**0.78**	0.51	**0.69**
DipScaling	0.01	0.16	0.18	0.81	**0.68**	0.45	0.27	0.73	**0.52**	0.68
k-means	0.01	0.02	0.03	0.70	0.51	0.26	0.11	0.70	0.32	0.65
Normalized	0.01	0.02	0.02	0.68	0.50	0.28	0.27	0.67	0.49	**0.69**
Z-Transformation	0.01	0.02	0.02	0.68	0.51	0.28	0.28	0.67	0.49	**0.69**
k-means++	0.01	**0.32**	0.03	0.75	0.56	0.22	0.11	0.71	0.18	0.57
DipMeans	0.00	—	0.25	0.55	0.45	0.00	0.00	0.00	0.00	0.45
SkinnyDip	**1.00**	—	0.34	0.55	0.55	0.30	0.00	0.53	0.26	0.00
Spectral Clust.	0.06	—	0.03	0.60	0.60	0.29	0.09	0.34	0.45	**0.69**
STSC	0.35	—	0.26	0.39	0.53	0.03	—	0.66	0.31	0.09
FUSE	0.09	—	0.03	0.46	0.06	0.02	0.06	0.15	0.11	0.31
DBSCAN	0.27	—	0.46	0.62	0.54	0.27	0.14	0.50	0.41	0.59
SingleLink	0.11	—	0.03	0.61	0.08	0.05	0.00	0.05	0.27	0.35
EM	0.72	0.23	0.11	0.78	0.43	0.34	0.14	0.64	0.38	0.25
SubKMeans	0.01	0.01	0.01	0.66	0.56	0.22	**0.29**	0.73	0.45	0.66
FossClu	—	0.27	0.44	0.75	0.48	0.50	0.08	0.50	0.32	0.34
SynC	0.12	0.13	0.14	0.58	0.52	0.13	0.24	0.48	0.29	0.27
t-SNE + k-means	0.02	—	0.64	0.31	0.02	0.06	0.11	0.16	0.08	0.35
PCA + k-means	0.01	0.01	0.01	0.64	0.56	0.21	0.26	0.74	0.49	**0.69**
ICA + k-means	0.64	—	0.08	0.57	0.46	0.19	0.12	0.61	0.42	**0.69**

Table 3: K-means++ as well as k-means before and after the DipTransformation. Parameters as described.

Data set	Whiteside	Skinsegmen.	Banknote	Iris	Prestige	Userknow.	Mammographic	Seeds	Breast Tissue	Leaf
k-means before	0.01	0.02	0.03	0.70	0.51	0.26	0.11	0.70	0.32	0.65
k-means++ before	0.01	0.32	0.03	0.75	0.56	0.22	0.11	0.71	0.18	0.57
k-means after	**1.00**	0.32	0.68	0.84	**0.68**	0.53	**0.27**	**0.78**	**0.51**	0.69
k-means++ after	**1.00**	**0.44**	**0.69**	**0.86**	**0.68**	**0.64**	**0.27**	0.76	0.49	**0.70**

due to the DipTransformation - on average 0.636 in NMI. After the DipTransformation 12 of the 20 algorithms are better than 0.90.

FossClu (Goebl et al., 2014) and SubKMeans (Mautz et al., 2017) try to find an optimal subspace for clustering while transforming the data set themselves. These transformation attempts are also more successful after DipTransformation has transformed the data set.

5.2 Real-World Data sets

We have tested DipScaling and DipTransformation extensively on 10 real world data sets, which differ greatly in dimensionality, number of data points and number of clusters. The ultimate goal of the Transformations is to enhance the structure of a data set and thus to improve clustering. To show that this goal can be achieved, we have transformed these data sets and applied the basic k-means algorithm on them. The results can be seen in Table 2. The difference in clustering quality is obvious. While k-means usually fares somewhat lacking on the original data sets and other algorithms like Spectral Clustering are often the better choices, it does perform extremely well on the transformed data sets. Transforming the data set has enhanced the structure so that k-means can cluster the data sets better than the compared methods, especially combined with DipTransformation. In 8 of the 10 cases the DipTransformation + k-means is now the best choice (sometimes together with other methods), in one case is DipScaling + k-means even the best choice (with the smallest possible lead of 0.01 over DipTransformation) and in the last case DipScaling and DipTransformation loose with a small deficit of 0.02. DipTransformation clearly improves the structure of the data sets, so that k-means is now a very good choice. DipScaling is often also a good choice and improves k-means, but it is also obvious that it does not always perform on the same level as DipTransformation. We assume that this is due to the level of independence of the features in the data sets. The Whiteside-data set has features far from being independent, and hence, DipScaling is not a good choice. Other data sets like Prestige and Breast Tissue seem to fit more into the assumption of independence.

We chose the algorithms we found to be most relevant as comparison methods here. This included the data set-transformation techniques of normalizing and Z-Transformation, the standard data mining algorithms DBSCAN (Ester et al., 1996), EM (Dempster et al., 1977) and SingleLink (Sibson, 1973), DipMeans (Chamalis and Likas, 2018) and SkinnyDip (Maurus and Plant, 2016) as techniques based on the Dip-test, SubKMeans (Mautz et al., 2017) and FossClu (Goebl et al., 2014) as the most similar Subspace-clustering-techniques, the aforementioned Spectral Clustering-methods, SynC (Böhm et al., 2010), as well as PCA, ICA and t-SNE in combination with k-means. For PCA, ICA and t-SNE we decided not to reduce the dimensionality, because there is no completely straightforward answer on how far one should reduce the dimensionality and because DipTransformation also does not reduce dimensionality. For DipScaling alone Normalization and Z-Transformation would be the most comparable methods.

We used the NMI-score (Vinh et al., 2010) as a measure of comparison. We are aware of another frequently used, state-of-the-art metric for evaluating clustering results, called Adjusted Mutual Information (AMI). We have found that the results do not vary to the extent that the "take home message" changes, so we omit AMI results to reduce clutter. The same also holds for other variants of NMI. There are multiple ways on normalizing NMI, but the difference is usually also not very large.

Parameters and Determinism Algorithms like DBSCAN always raise the question of how to set the parameters. To compare the DBSCAN results fairly, we decided to make the parameters dependent on the average pairwise Euclidean distance of data points. Let us call it e. We tested all combinations of distances in $\{0.05 \cdot e, 0.1 \cdot e, 0.2 \cdot e, 0.3 \cdot e, 0.4 \cdot e, 0.6 \cdot e, 0.8 \cdot e, e\}$ and MinPts in $\{1, 2, 3, 5, 10, 50\}$. Only the best NMI result is reported.

All techniques that require the number of clusters as a parameter have been given the correct number of clusters k. The only exception is SingleLink where all values in the interval $[k, 2k]$ have been tested and only the best result is reported. This decision is due to the characteristic of SingleLink to declare single data points or small subsets of a cluster as clusters.

Non-deterministic algorithms such as k-means have been iterated 100 times to reduce random effects and provide robust results.

Mammographic Mass This UCI data set contains data points with missing entries. These fragmented data points have been removed from the data set. All methods were tested on the cleaned data set.

Table 4: Various Clustering algorithms before and after DipTransformation. Parameters as before. On average the clustering improves by 0.223 (measured in NMI).

Data set		Whiteside	Iris	Prestige	Mammographic	Breast Tissue
EM	before	**1.00**	0.58	0.28	**0.01**	0.37
	after	**1.00**	**0.90**	**0.63**	0.00	**0.45**
DBSCAN	before	0.27	**0.62**	0.54	0.14	0.41
	after	**0.72**	0.61	**0.60**	**0.15**	**0.45**
SingleLink	before	0.11	**0.61**	0.08	0.00	0.27
	after	**0.82**	**0.61**	**0.54**	**0.16**	**0.35**
Spectral Clustering	before	0.06	0.60	0.60	0.09	0.45
	after	**1.00**	**0.65**	**0.65**	**0.26**	**0.50**
k-means	before	0.01	0.70	0.51	0.11	0.32
	after	**1.00**	**0.84**	**0.68**	**0.27**	**0.51**

Table 5: Various Clustering algorithms before and after DipScaling. Parameters as before. On average the clustering improves by 0.096 (measured in NMI).

Data set		Whiteside	Iris	Prestige	Mammographic	Breast Tissue
EM	before	**1.00**	0.58	0.28	0.01	0.37
	after	**1.00**	**0.90**	**0.63**	**0.03**	**0.53**
DBSCAN	before	**0.27**	**0.62**	0.54	0.14	0.41
	after	**0.27**	**0.62**	**0.61**	**0.15**	**0.44**
SingleLink	before	**0.11**	**0.61**	0.08	0.00	**0.27**
	after	**0.11**	0.60	**0.54**	**0.23**	0.12
Spectral Clustering	before	**0.06**	0.60	0.60	0.09	0.45
	after	**0.06**	**0.65**	**0.64**	**0.28**	**0.51**
k-means	before	**0.01**	0.70	0.51	0.11	0.32
	after	**0.01**	**0.81**	**0.68**	**0.27**	**0.52**

Skinsegmentation The Skinsegmentation-data set is a somewhat difficult data set simply due to its size of roughly a quarter of a million data points. For many of the provided implementations the size was too large and the execution failed. This also applies to some of the standard methods like SingleLink, DBSCAN and Spectral Clustering. These were tested on more than one implementation on different platforms, but would not run through anyway.

Spectral Clustering If this paper refers to Spectral Clustering as an algorithm and not the class of algorithms, then the classical algorithm by Ng, Jordan and Weiss (Ng et al., 2002) is meant.

Besides these considerations it is most noticeable that k-means++ leads to the same increase in NMI as the DipTransformation on the Skinsegmentation-data set. However, K-means++ and Dip-Transformation are by no means mutually exclusive and can be used together. This in fact leads to an

Table 6: Average Improvement for the 5 standard clustering algorithms on the 5 data sets in combination with DipTransformation and DipScaling.

Method	k-means	EM	Spectral C.	SingleLink	DBSCAN
DipTransformation	0.33	0.14	0.25	0.28	0.11
DipScaling	0.13	0.15	0.07	0.11	0.02

even better performance on the Skinsegmentation data set. While they separately reach a level of 0.32 in NMI, they manage 0.44 in combined form.

5.3 K-means++ and DipTransformation

As mentioned, k-means++ and DipTransformation are not mutually exclusive. We tested on all the data sets used in the experiments whether k-means++ fared better before or after the DipTransformation. The results are shown in Table 3. Following these, we can say that k-means++ is a bit of a double-edged sword on the original data sets. On some of them (Skinsegmentation, Iris) k-means++ is clearly better than k-means; on some of them (Breast Tissue, Leaf) it is the other way round. After the DipTransformation, the situation is far more beneficial for k-means++. Usually, there is only a small difference between k-means and k-means++ (≤ 0.02), indicating that there might be fewer local optima, compared to the original data set. The only times when k-means and k-means++ do differ (Skinsegmentation, Userknowledge) is when k-means++ performs quite a bit better than k-means alone.

DipTransformation (also DipScaling) can be used together with all types of support techniques for k-means (or other clustering algorithms). For example, X-means (Pelleg and Moore, 2000) can be used to find the number of clusters, k-means-- (Chawla and Gionis, 2013) to find outliers, k-means++ to find an initialization, SubKMeans to find a subspace and all this in combination with the DipTransformation.

5.4 DipScaling/DipTransformation and Clustering Algorithms besides K-means

DipScaling and DipTransformation were developed with a focus on k-means, but as we have stated throughout our work, they enhance the structure; they do not adapt the data set so that it only fits k-means and therefore other algorithms can also benefit from them. We have seen the transformation of the Whiteside- as well as the Banknote-Authentication-data set in Fig. 6 respectively Fig. 5.1. Both of these do seem easier to cluster for various algorithms. We have taken 5 of the data sets used in the real world data sets experiments and clustered their transformations with DipScaling and DipTransformation with 4 standard data mining algorithms, i.e EM, DBSCAN, SingleLink and Spectral Clustering. The results can be seen in Table 4. We chose the standard algorithms because they are well-established in the community, which makes the results all the more credible. For the sake of completeness k-means is also included here. For DipTransformation we see a tiny decrease in clustering quality of 0.01 in NMI in two cases. In two more cases does the quality not change at all. In the other 21 cases does the quality increase - in some cases substantially. On average, counting all cases, the quality increases by 0.223 in NMI. For DipScaling the situation is somewhat similar. In two cases the clustering quality decreases (both in combination with SingleLink), in 6 it stays the same and in 17 it increases. It improves on average by 0.096 in NMI.

To find out how compatible DipScaling and DipTransformation are with the other standard clustering approaches in comparison to k-means we made Table 6. It shows by how much the standard clustering techniques improve in combination with our Dataset-Transformations on the 5 data sets. While k-means is the one that profits the most, so is e.g. Spectral Clustering not far off. SingleLink is

even closer, but when looking at Table 4 and 5 it becomes clear that SingleLink is more volatile, while Spectral Clustering profits more constantly.

This should, in combination with Fig. 6 and Fig. 5.1, be a very convincing argument that DipTransformation as well as DipScaling can play an important role in Clustering as support techniques applied to the data set before clustering to enhance structure.

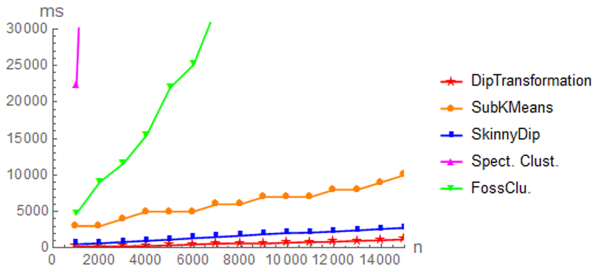


Fig. 10: Runtime relative to the data set size n. Dimensionality is 5.

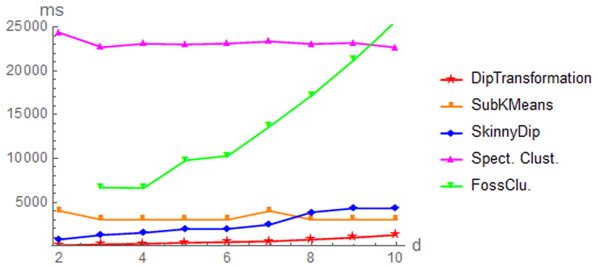


Fig. 11: Runtime relative to the dimensionality of the data set d. Data set size is ≈ 1.500 data points.

5.5 Runtime Comparisons

For the DipTransformation the runtime was estimated to have a $\mathcal{O}(n \log n)$-dependency on the number of data points n. Synthetic data sets ranging from 1.000 to 15.000 data points were created to test for this dependency and to compare with other algorithms. The runtime is plotted in Fig. 10. It is not immediately apparent, but $\mathcal{O}(n \log n)$ is a very good estimate for the runtime. We also see that DipTransformation performs quite well compared to the other tested algorithms. DipTransformation is faster for all data sets. To ensure comparability in this test is also the runtime of k-means included in the measured time for DipTransformation, since the other methods cluster data, which DipTransformation does not do by itself.

Besides the size of the data set also the influence of the dimensionality of the data set on the runtime is essential. This is shown in Fig. 11. We created 9 data sets ranging in dimensionality from 2 to 10 with 1.000 data points each. Here DipTransformation (+k-means) is again faster than all other methods, but we do see in the behaviour of the measured time, that other methods like Spectral Clustering are less affected by the dimensionality. The estimation of an $\mathcal{O}(d^2)$ dependency on dimensionality is again very good, so the conjecture that at some point DipTransformation will need more time than e.g. Spectral Clustering is likely. However, if one extrapolates from the curves, it seems as if that would happen at a rather high dimensionality.

We compared DipTransformation with the algorithms which we found most similar to it. DipScaling is a different, more basic type of method, hence, we also compare it with different approaches. The closest related approaches are doubtlessly Normalization and Z-Transformation, PCA and ICA are also included as basic techniques. Contrary to before the runtime for k-means is not included (also

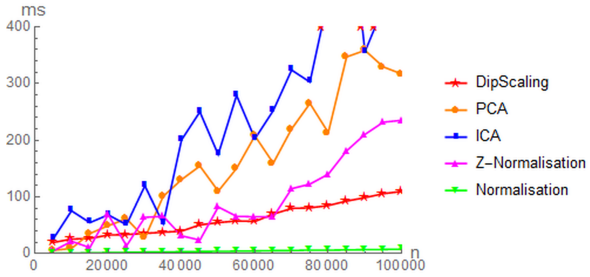


Fig. 12: Runtime relative to the data set size n. Dimensionality is 5.

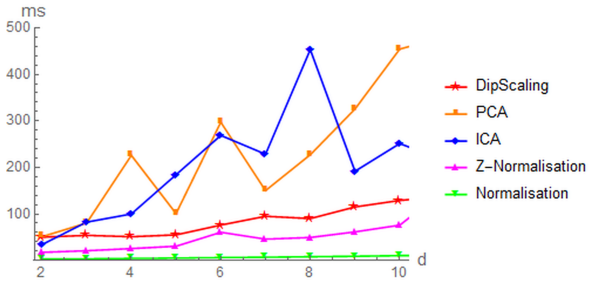


Fig. 13: Runtime relative to the dimensionality of the data set d. Data set size is $\approx$ 50.000 data points.

not for the compared methods). The most obvious difference to DipTransformation is the overall runtime. DipScaling is far faster than DipTransformation, which is no surprise, considering DipScalings simplicity and that it is used as part of DipTransformation.

The estimation for the dependency on the number of data points and the dependency on the dimensionality made in Section 4 for DipScaling was $\mathcal{O}(n \log n)$ respectively $\mathcal{O}(d)$. These are both correct according to our runtime experiments depicted in Fig. 12 and Fig. 13. It is surprising that Z-Transformation respectively PCA/ICA is slightly erratic, when testing the dependencies, but this is probably due to implementation details. For all of them the standard R-Implementations were used, contrary to Normalization, which was implemented by us. R tends to call on C and/or Fortran-code, so the assumption is close that internal methods/libraries are called, which might cause the fluctuations. Our implementation of Normalization has no (relevant) fluctuations, so this seems probable. The fluctuations are small enough, so that the trend can be seen and DipScaling is at the very least in the same range of runtime-needs as the compared methods, which means that it is very fast.

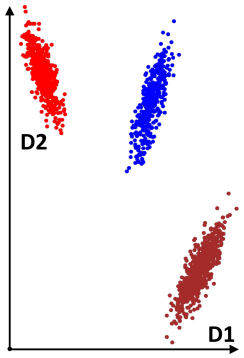
Algorithms are implemented in Java (DipTransformation and FossClu), Scala (SubKMeans) and R (Spectral Clustering, SkinnyDip, PCA, ICA, Normalization and Z-Transformation) and executed on an Intel Xeon E5 with 16Gb RAM.

5.6 Structure

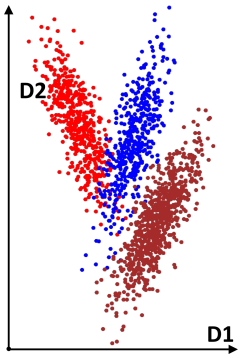
We have been confronted with the question of how the DipTransformation fares in regard to the amount of structure contained in the data set. The difficulty here is of course that "structure" is not a perfectly well-defined term, and hence, the "amount of structure" is difficult to measure. We decided on the following: We start with a data set, where the clusters are clearly separated (see Fig. 14a). The data set is very simple (k-means scores above 0.90 in NMI) and the common consensus would most likely be that the data set is well structured. We created more data sets in almost the same style, but the variance in the clusters is linearly increased. The clusters become wider and start to overlap (Fig. 14b). While the clusters are still somewhat well-separated, they are harder to distinguish and some of the

data points could belong to any cluster. K-means fares worse here (≈ 0.50 in NMI). The structure of the data set becomes less obvious, as the clusters start to overlap. Increasing the variance even further leads to basically overlapping clusters (Fig. 14c), which are now mostly inseparable. Many data points could belong to any cluster. At this point we feel justified to state that the data set contains almost no structure and the data set is not much more than a mush of data points. The performance of k-means drops below 0.10 in NMI.

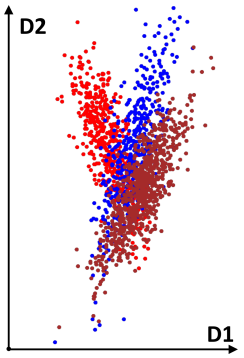
We have changed the variance of the clusters in a range so that k-means performs from 0.90 (high structure, well separated clusters) down to 0.10 (barely any structure, overlapping clusters) in NMI on the data sets and tested how DipTransformation and DipScaling influence the clustering results. If k-means performs very well, then the Dataset-Transformations have limited influence and improve clustering only slightly. As the data sets become more difficult to cluster, the influence of DipTransformation and DipScaling becomes more obvious. DipTransformation is the better choice here compared to DipScaling and can improve clustering immensely. A more difficult data set also makes it more difficult for the Dataset-Transformation. When the data set becomes not much more than a mush of data points, they also have too little information to have a strong effect; there is not enough structure left, as that it could be enhanced any more. DipScaling is in this case roughly as effective as DipTransformation.



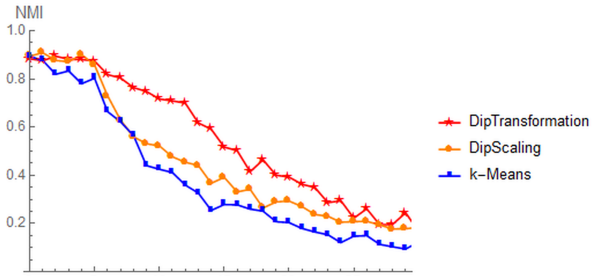
(a) Clearly separated clusters - high structure.



(b) Clusters start to overlap - structure is slowly disappearing.



(c) Overlapping clusters - no structure visible.



(d) Change in NMI depending on the amount of structure found in the original data set.

Fig. 14: Behaviour of DipTransformation and DipScaling relative to the "amount of structure" present in the data set.

6 Limitations and Outlook

DipTransformation (and partly DipScaling) is a very powerful technique for enhancing the structure and emphasizing clusters, but there are certain limitations. For example, if two clusters overlap or interlock, DipTransformation would by design not be able to separate them. It stretches and scales the data. Therefore, all clusters which do not have a hyperplane in-between them cannot be separated. The same applies to clusters that contain more than one mode. The Dip-test is designed to work with unimodal distributions. Multimodal clusters can prevent the DipTransformation from working properly. DipScaling has the same restrictions; furthermore, it assumes independent features, which is very often not the case.

In terms of runtime, the main constraint we encountered was the $\mathcal{O}(d^2)$-dependency of DipTransformation on the dimensionality of a data set. For very high dimensional data sets it might be useful to combine DipTransformation with a dimensionality reduction algorithm or stick to DipScaling. DipScaling might not improve clustering quite as much as DipTransformation, but it is very fast.

We do intend to implement a dimensionality-reducing feature into the DipTransformation. The Dip-test provides a probability estimation of the unimodality of a feature. For the running example, the Dip-test gave a probability of 100% for the third dimension to be unimodal. This is a correct estimate as the third dimension was constructed as uniformly distributed noise. If we assume that a unimodally distributed characteristic is essentially not of great interest, then we could eliminate this dimension and reduce the running example to a two-dimensional data set. This two-dimensional data set would then be treated as explained in this paper. At some point, however, it might happen that the Dip-test finds another unimodal feature and the data set can be further reduced.

According to this roadmap, the DipTransformation could be converted into a technique that improves the structure of a data set while reducing dimensionality. We intend to do this in the near future.

7 Conclusion

In conclusion, we can say that we have achieved our goal of creating a technique that can improve the structure of a data set and thus its clustering. We have shown that this statement is true by testing our Dataset-Transformations extensively on various data sets.

For k-means, which was the main focus, this is now particularly clear. On the tested data sets, k-means was usually a sub-ideal choice and other algorithms were clearly better. After applying DipTransformation/DipScaling k-means was the best-performing algorithm on all but one data set.

We have also shown that DipTransformation and DipScaling are compatible with other algorithms and also improves their clustering results. They can therefore be used as a pre-clustering step, that enhances the data set, and the clustering algorithm can be chosen according to the users preferences.

Bibliography

Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C. and Park, J. S. (1999), Fast algorithms for projected clustering, *in* 'Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data', SIGMOD '99, ACM, New York, NY, USA, pp. 61–72.

Agrawal, R., Gehrke, J., Gunopulos, D. and Raghavan, P. (1998), Automatic subspace clustering of high dimensional data for data mining applications, *in* 'Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data', SIGMOD '98, ACM, New York, NY, USA, pp. 94–105.

Arthur, D. and Vassilvitskii, S. (2007), K-means++: The advantages of careful seeding, *in* 'Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms', SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1027–1035.

Böhm, C., Plant, C., Shao, J. and Yang, Q. (2010), Clustering by synchronization, *in* 'Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '10, ACM, New York, NY, USA, pp. 583–592.

Chamalis, T. and Likas, A. (2018), The projected dip-means clustering algorithm, *in* 'Proceedings of the 10th Hellenic Conference on Artificial Intelligence', SETN '18, pp. 14:1–14:7.

Chawla, S. and Gionis, A. (2013), 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.', pp. 189–197.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977), 'Maximum likelihood from incomplete data via the em algorithm', *Journal of the Royal Statistical Socoety, Series B* **39**(1), 1–38.

Dua, D. and Graff, C. (2019), 'UCI machine learning repository'.
URL: *http://archive.ics.uci.edu/ml*

Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. (1996), A density-based algorithm for discovering clusters 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, AAAI Press, pp. 226–231.

Goebl, S., He, X., Plant, C. and Böhm, C. (2014), Finding the optimal subspace for clustering, *in* 'Proceedings of the 2014 IEEE International Conference on Data Mining', ICDM '14, pp. 130–139.

Hand, D. J., Daly, F., McConway, K., Lunn, D. and Ostrowski, E. (1993), *A handbook of small data sets*, Chapman and Hall, London, U.K.

Hartigan, J. A. and Hartigan, P. M. (1985), 'The dip test of unimodality', *Ann. Statist.* **13**(1), 70–84.

Kriegel, H.-P., Kröger, P. and Zimek, A. (2009), 'Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering', *ACM Trans. Knowl. Discov. Data* **3**(1), 1:1–1:58.

Maurus, S. and Plant, C. (2016), Skinny-dip: Clustering in a sea of noise, *in* 'Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '16, pp. 1055–1064.

Mautz, D., Ye, W., Plant, C. and Böhm, C. (2017), Towards an optimal subspace for k-means, *in* 'Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '17, pp. 365–373.

Ng, A. Y., Jordan, M. I. and Weiss, Y. (2002), On spectral clustering: Analysis and an algorithm, *in* T. G. Dietterich, S. Becker and Z. Ghahramani, eds, 'Advances in Neural Information Processing Systems 14', MIT Press, pp. 849–856.

Pelleg, D. and Moore, A. (2000), X-means: Extending k-means with efficient estimation of the number of clusters, *in* 'In Proceedings of the 17th International Conf. on Machine Learning', Morgan Kaufmann, pp. 727–734.

Sibson, R. (1973), 'Slink: An optimally efficient algorithm for the single-link cluster method.', *Comput. J.* **16**(1), 30–34.

Siffer, A., Fouque, P.-A., Termier, A. and Largouët, C. (2018), Are your data gathered?, *in* 'Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '18, pp. 2210–2218.

Van der Maaten, L. and Hinton, G. (2008), 'Visualizing data using t-SNE', *Journal of Machine Learning Research* **9**, 2579–2605.

Vinh, N. X., Epps, J. and Bailey, J. (2010), 'Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance', *J. Mach. Learn. Res.* **11**, 2837–2854.

Xie, J., Girshick, R. and Farhadi, A. (2016), Unsupervised deep embedding for clustering analysis, *in* M. F. Balcan and K. Q. Weinberger, eds, 'Proceedings of The 33rd International Conference on Machine Learning', Vol. 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp. 478–487.

Yang, B., Fu, X., Sidiropoulos, N. D. and Hong, M. (2017), 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', pp. 3861–3870.

Yang, C., Zhang, X., Jiao, L. and Wang, G. (2008), Self-tuning semi-supervised spectral clustering, *in* '2008 International Conference on Computational Intelligence and Security, CIS 2008, 13-17 December 2008, Suzhou, China, Volume 1 - Conference Papers', pp. 1–5.

Ye, W., Goebl, S., Plant, C. and Böhm, C. (2016), Fuse: Full spectral clustering, *in* 'Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '16, ACM, New York, NY, USA, pp. 1985–1994.

Author Biographies

Benjamin Schelling received a B.A. of Philosophy as well as a BSc and MSc of Mathematics at the University of Vienna, Austria. He is currently enrolled as a PhD-student at the Faculty of Computer Science at the Data Mining research group at the University of Vienna. His main research interest are support methods for clustering, such as dataset-transformations.

Claudia Plant is full professor of computer science and leader of the Data Mining research group at University of Vienna, Austria. Her research focuses on new methods for exploratory data mining, e.g., clustering, anomaly detection, graph mining and matrix factorization. Many approaches relate unsupervised learning to data compression, i.e. the better the found patterns compress the data the more information we have learned. Other methods rely on finding statistically independent patterns or multiple non-redundant solutions, on ensemble learning or on nature-inspired concepts such as synchronization. Indexing techniques and methods for parallel hardware support exploring massive data. Claudia Plant has co-authored over 100 peer-reviewed publications, among them many contributions to the top-level data mining conferences KDD and ICDM and 3 Best Paper Awards. Papers on scalability aspects appeared at SIGMOD, ICDE, and the results of interdisciplinary projects in leading application-related journals such as Bioinformatics, Cerebral Cortex and Water Research.

Appendix D

Paper D: DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering (Extended Abstract)

(Hrsg.): ,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 1

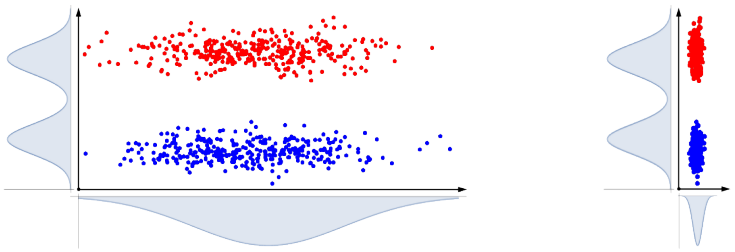
DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering (Extended Abstract)

Presentation of work originally published in the Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM 2018).

Benjamin Schelling[1] Claudia Plant[1 2]

Keywords: Clustering, Dip-test, Dataset-Transformation

The clustering of a data set depends strongly on the structure it contains. A data set might have a well-defined structure, but this does not necessitate good clustering results. If the structure is hidden in an unfavourable scaling, clustering usually fails. Confronted with a data set one cannot quite cluster, usually this would lead to a new clustering method which is capable of dealing with the new and problematic type of data set, but this is not always necessary. The aim of the DipTransformation is to enhance the data set by re-scaling and transforming its features and thus emphasizing and accentuating its structure. If the structure is sufficiently clear, clustering algorithms - even well-established ones - will perform far better. To the best of our knowledge, there are currently no methods besides DipTransformation that have the goal of enhancing structure.

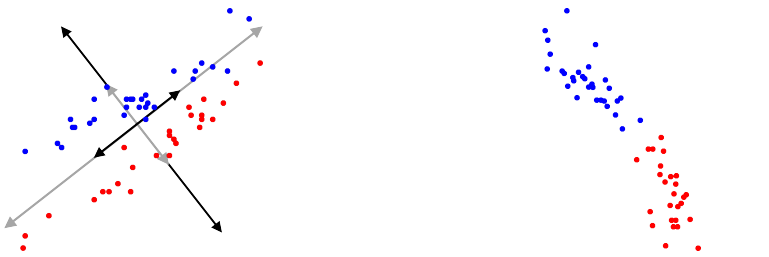


(a) (b)

Fig. 1: A simple, synthetic data set before (a) and after (b) scaling it with its dip values. The dip value are used to tell us how much structure a feature contains and how relevant it is for clustering.

DipTransformation makes it possible to compensate for the unfortunate scaling of the features with the help of the Dip test [1]. The Dip test measures the amount of structure in a feature. Take a look at Fig. 1. It is a very simple data set, consisting of two Gaussian

[1] Faculty of Computer Science, University of Vienna, Vienna, Austria
[2] ds:UniVie, University of Vienna, Vienna, Austria



(a) The Whiteside-data set and the main direction of interest. (b) After the DipTransformation.

Fig. 2: We find the direction with the most structure (black: how much structure is found, grey: how large it is originally scaled) and scale the features according to it. It is now very easy to cluster.

distributed clusters. It should be very easy to cluster, but many algorithms (e.g. k-means) have massive difficulties with it, due to the scaling. Measuring the amount of structure of the features with the dip test and re-scaling the features leads to Fig. 1.b, a very easy to cluster data set. The heuristic here is that a feature is scaled relative to how much structure is found. The horizontal axis has barely any structure, it is uni-modal, and thus is scaled such that it has only a very small extension, which means that it has no great influence on clustering, as the values are all very similar. The feature with high structure – it is clearly multi-modal, i.e. has clusters one can distinguish from each other – is scaled such that this feature has a high influence on clustering.

The DipTransformation is not limited to axes-parallel re-scaling. Using a cleverly devised search strategy, it can automatically find non-axis-parallel features with high dip values, which it rescales as explained. One such example is shown in Fig. 2. The Whiteside-data set is a real-world data set that is difficult to handle for many clustering approaches, due to its clusters which are tricky to differentiate. Most approaches fail completely, but after the transformation, it is almost trivial.

In conclusion, we developed a technique that can improve the structure of a data set and thus its clustering. We show in [2] that this is true by testing it extensively on various data sets, all of which become far easier to cluster for various standard and state-of-the-art clustering approaches. DipTransformation assumes no data distribution, is deterministic, basically parameter-free and quite fast compared to various clustering approaches. It can thus be used as a pre-clustering step, that enhances the data set, and the clustering algorithm can be selected according to user preferences.

Bibliography

[1] Hartigan, J. A., Hartigan, P. M., *The Dip Test of Unimodality*, The Annals of Statistics, 1985.

[2] Schelling, B., Plant, C., *DipTransformation: Enhancing the Structure of a Dataset and Thereby Improving Clustering*, ICDM, 2018.

Appendix E

Paper E: Non-linear Cluster Enhancement: Forcing Clusters into a compact shape

Non-linear Cluster Enhancement: Forcing Clusters into a compact shape

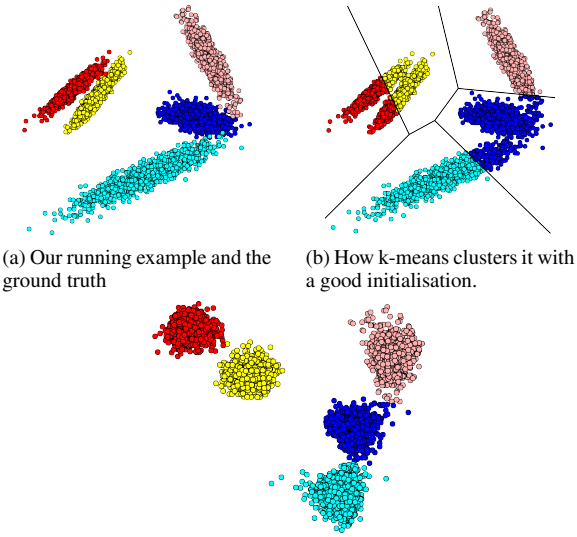
Benjamin Schelling[1] and **Lukas Miklautz**[1] and **Claudia Plant**[1,2]

Abstract. K-means is one of the most widely used clustering algorithms there is and applied for a wide range of settings, but how do we know that a data set is suited for it? K-means's assumptions about the data are relatively strict: clusters should be Gaussian distributed with uniform variance in all directions. These assumptions are rarely satisfied in a data set. While clusters that do not deviate from these assumptions too far, can be cut out with sufficient precision, the farther the data is from these assumptions, the more likely k-means is to fail. Instead of testing whether the assumptions are met and k-means can be applied, we make it so. Our goal is to improve the suitability of data sets for k-means and widen the range of possible data sets it can be applied to. Our algorithm changes the position of data points so that the clusters become more compact and, thus, fit better into the requirements of k-means. Based on cluster-wise PCA and local Z-transformation we estimate the form of the correct clusters and move the data points so that the correct clusters become more compact with each iteration and – in the end – have uniform variance, as well as increase the distance between clusters. We explain the theory behind our approach and validate it with extensive experiments on various real world data sets.

1 INTRODUCTION

K-means [18] is one if not the most widely used clustering algorithm there is. It is being studied intensively in scientific circles, with dozens of articles about every aspect of it. It is used from speech recognition [5, 17] to autonomous driving [28] and is frequently an important building block for a more extensive system as k-means is simple, fast and often gives good results. It is used in combination with classical Clustering methods, e.g. as an initialisation for EM [7], as well as for new neural networks-based methods, e.g. DEC [29].

In principle, k-means requires clusters that have a Gaussian bubble shape with equal variance in all directions (we shorten this to "uniform variance"), but it can handle many forms of clusters as long as they are mostly convex, non-overlapping and somewhat well separated. K-means partitions the data into Voronoi cells (see [9] for a short introduction; an example can be seen in Fig. 1b). The closer the clusters match the assumptions of Gaussian shape with uniform variance while being well separated, the better they fit into this Voronoi cell structure of k-means. Compact clusters are more suited for these cells, while non-convex clusters might not fit at all. Thus, data sets are often preprocessed by normalizing them. Normalizing in the [0,1]-range or Z-transformation prevents clusters from being extremely stretched in one direction and very contracted in another. The clusters are therefore in a more compact shape, which makes them easier identifiable as they fit better into the Voronoi cells. Normalization is often recommended for k-means [21] and other clustering approaches [14].



(a) Our running example and the ground truth

(b) How k-means clusters it with a good initialisation.

(c) How we manage to change the data set. Clusters are now almost perfectly Gaussian.

Figure 1: The running example, how k-means clusters it and how it looks after PCE (our method).

The running example shown in Fig. 1a is simple and would be well suited for k-means, as the clusters follow a Gaussian distribution and are well-separated. The spread of the clusters, though, makes it difficult for k-means to partition it well. A typical clustering result of k-means, if the initialisation goes well, is shown in Fig. 1b. Clusters are cut into multiple parts and incorrectly merged. Even if the initialization strategy succeeds, i.e. returns one data point from each cluster as the start initialization for the centers, k-means performs poorly. The clusters do not fit into the Voronoi cell structure of k-means. Merely normalizing the data is not enough to make it suited for k-means. Besides the normalization of data, the most commonly used approaches that can be considered transformations are PCA [10] and ICA [6]. However, both are linear and as can be seen in Fig. 1a the clusters cannot be separated linearly (see Table 1 that they have almost no effect on the running example). Non-linear transformations are rare. Most, like DEC and IDEC [12], are based in Neural Networks. These could in principle separate the clusters, but often

[1] Faculty of Computer Science, University of Vienna, Vienna, Austria
[2] ds:Univie, University of Vienna, Vienna, Austria

proceed rather rough and tear the data set apart (see Section 4), making the clusters unlikely to fit into the Voronoi cell structure.

The method that we present takes the (preliminary) clusters found by k-means (i.e. the Voronoi cells) and shifts the data points depending on the shape and spread of them. It forces very elongated clusters into a more compact form with uniform variance while separating the clusters better from each other. It does so while still keeping the basic shape and form of the data set intact (see Section 4). A successful application for the running example is shown in Fig. 1c. The clusters fit now seamlessly into the Voronoi cell structure of k-means, which can perfectly cluster the data. Thus, we extend the spectrum of data sets suited for k-means. We intend to demonstrate that this increase in suitability is substantial enough, that the combination of our transformation with k-means can now outperform a wide range of various clustering algorithms. With this, we intend to forestall the retort "Why do you not simply use another algorithm besides k-means?" which also ignores how wide-spread k-means has become in various fields less familiar with various clustering approaches.

1.1 Contributions

The method presented in this paper, PCE (Principal Cluster Enhancement) is a **non-linear** transformation, making the clusters better suited for k-means.

- The focus is on k-means, as we explicitly try to make the clusters more suited to it, but many methods might benefit from more compact clusters. We include experiments for different standard clustering methods and demonstrate that they are also compatible with our method.
- We demonstrate that PCE keeps the basic shape intact, transforms the data non-linear while making clusters more compact.
- PCE and the reasons for its decisions are easy to understand. Some methods, e.g. Neural Network-based ones, are often hard to understand and resemble black boxes in this regard.
- The procedure is deterministic, excluding the initialization of k-means. With a deterministic initialisation for k-means it would be completely deterministic.
- Our method is light in runtime-requirement. The major part is executing PCA, a technique heavily researched and optimized.

1.2 Related Work

PCE is an algorithm that tries to lessen the assumptions of k-means and broaden the range of data sets suited for it. In regards to being a support-method for k-means, it falls into a long line of algorithms, from which X-means [22], to estimate the number of clusters, or k-means++ [1], to find a good initialisation, are most likely the best known. K-means has, over time, become a sort of framework which is freely adapted to suit the needs of various approaches. Examples are e.g. FOSSCLU [11] or SubKMeans [20], that look for subspaces compatible with k-means.

Closely related to us in terms of the pursued goal is DipTransformation [23] that has the explicit goal of changing the structure of the data set to improve clustering. It also transforms the data, but it is restricted to linear transformations like PCA and ICA. A basic version of DipTransformation is DipScaling [24], which scales the axes according to the information given by the Dip-Test [15]. It falls into the category of very simple normalization-methods like Z-transformation (also referred to as Z-normalization) and the normalization into the $[0, 1]$-range, which are the two fundamental transformation-methods which are often used for pre-processing. Transformation methods per se are rare. Apart from those mentioned, one of the closer members of the clustering community is SynC [2], which collapses multiple data points onto a single data points. One could further include Spectral Clustering-methods, that create a distance matrix from the data, which is after some steps clustered with partitioning methods like k-means. These could be considered as a pre-processing step for k-means. Kernel-based approaches [25] do not change the data set, but interpret distances differently, which is a similar concept. Feature Weighting-method like EWKM [16] also do not change the data points but put a different relevance on each feature for clustering. Closer are Neural Network-based methods like DEC, IDEC and DCN [30] that combine k-means and Deep Learning, which actually change the data set.

Our approach makes use of the Dip-Test [15], a statistical test for multi-modality, which has lately garnered some attention in the Data Mining-Community (see [19] for an introduction). The first method was DipMeans [4], a method from the k-means framework, that estimates the number of clusters in a data set. There is also SkinnyDip, a method to cluster in the presence of high noise, and the aforementioned DipTransformation and DipScaling. The Dip-Test has also recently been generalized from one dimension to multiple dimensions [26] and employed as a test, if clustering makes sense, i.e. if multiple clusters are present in the data set.

2 THE ALGORITHM

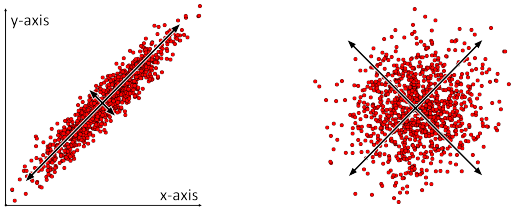
2.1 The principle: PCA and Z-transformation

We aim to change the data set so that clusters become more compact and thus easier to cluster. The basis, which we make heavy use of, is Z-transformation on one hand and PCA on the other. Z-transformation is used to obtain clusters with uniform variance. Z-transformation divides a feature by its standard deviation-value, which brings this feature to a variance of 1. It can be expressed as

$$x' = \frac{x - \mu}{\sigma} \tag{1}$$

with μ as the mean of the feature, x a data point and the variance σ^2, which means the standard deviation is σ.

The data set shown in Fig. 2a has the same variance in the x- and y-axis. Only applying Z-transformation will not change the shape of the cluster, i.e. it will not have uniform variance. For this we need PCA. PCA gives the directions with the highest variance. If we apply the Z-transformation along these directions we get Fig. 2b. The cluster is now of uniform variance and would be perfectly suited for k-means.



(a) A simple stretched cluster and its main components.

(b) The cluster is forced into a uniformly gaussian shape.

Figure 2: For a simple cluster, the main components of PCA and their variance is computed (shown with the length of the main components). Performing Z-Transformation along these directions leads to a shape with uniform variance.

2.2 Multiple clusters caught in a Voronoi cell

More than one cluster complicates the matter, as several clusters can be caught in a Voronoi cell. Consider the simple data set shown in Fig. 3a with two stretched clusters very close to each other, similar to two of the clusters from the running example. To the human eye, the correct clustering is easy to see, but k-means will converge to the sub-par clustering shown there with two clusters in a cell. The application of the same PCA/Z-transformation combination as before does not change the data set sufficiently to escape this state. We wish to change the data set so that the clusters are clearly separated and k-means can easily cluster it. For this, we need to adapt our approach by adjusting how we use the standard deviation.

The projections of the data points in the Voronoi cell to the main components of PCA is shown in Fig. 3b. These projections contain very different levels of cluster-information. We can see that in one of the projections we actually "caught" two clusters instead of one. Shown there is also the "strength" of the directions, i.e. the standard deviation/variance, of the main components as the length of the arrows. The cluster has roughly the same σ in both directions and thus applying the former approach does not change the data set enough for k-means to escape the local optima shown here. The goal is to push the clusters so far apart so that k-means will move out of this optima and distinguish between the clusters correctly. This can be done by "changing" σ. If σ becomes smaller, if more than one cluster is found in a direction, it will push the clusters farther away from each other and k-means can escape the local optima it is in and converge to a better state. Effectively, we change the probability landscape of k-means. We make some optima more likely while deterring k-means from others.

To test whether there is more than one cluster in such a direction, we use the Dip-test. The Dip-test shows us how likely such a projection is uni-modal, i.e. if more than one cluster has been caught. The probability found by the Dip-Test, p_d, is used to adapt the standard deviation σ. The formula we use is given by Eq. (2).

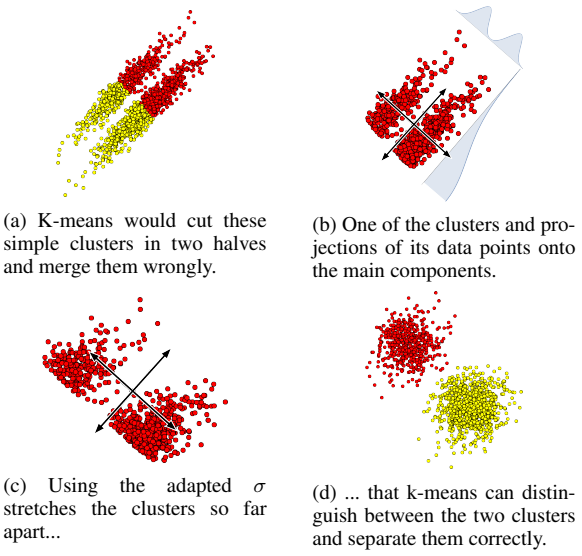
$$\bar{\sigma} = \sigma + p_d \cdot \sigma \tag{2}$$

This doubles the standard deviation for a uni-modal direction, while a multi-modal direction is kept the same. The effect is shown in Fig. 3. In Fig. 3b the standard deviations of the projections are in an equilibrium. Applying the earlier transformation does not change the data set. Using $\bar{\sigma}$ interprets the need to change the data set very differently. The uni-modal direction is contracted much more compared to before, as $\bar{\sigma}$ is twice as large as σ. This leads to Fig. 3c. The clusters are now so far apart, that k-means can distinguish between them and the clusters are correctly clustered. Using $\bar{\sigma}$ allows for clusters to become better separated from each other in the data. For a single cluster like the one shown in Fig. 2 nothing much changes, but for a multi-cluster Voronoi cell like the one in Fig. 3 our transformation starts pushing clusters away from each other, reducing the difficulty for k-means to cluster the data correctly.

We use the Dip-test as a measure for the possibility of multiple clusters, as it is rather precise and has a good runtime of $\mathcal{O}(n \log(n))$ (n the number of data points tested). We increase $\bar{\sigma}$ linearly in Eq. (2) from σ to 2σ with the likelihood of uni-modality, because, following Occam's Razor, it is the least complex approach.

2.3 Interaction of Voronoi cell-Transformations

There are two things we need to consider: 1) The preliminary clusters found by k-means are most likely flawed. We have shown how we intend to deal with this. 2) We cannot analyse a cell by itself. We focus on this now. Consider two data points very close to each other but in different Voronoi cells. If we apply the combination of Z-transformation/PCA only on the data points in a single Voronoi cell, the two data points might end up far apart from each other. This would lead to large gaps in the data and might even rip apart a correct cluster because it is part of two different Voronoi cells. Hence, we cannot analyse a cell alone and need a way how a transformation of one Voronoi cell affects another cell.



(a) K-means would cut these simple clusters in two halves and merge them wrongly.

(b) One of the clusters and projections of its data points onto the main components.

(c) Using the adapted σ stretches the clusters so far apart...

(d) ... that k-means can distinguish between the two clusters and separate them correctly.

Figure 3: How the adapted σ helps with distinguishing between clusters.

Applying the transformation of one cell to all data points in all cells is possible. However, this would lead to clusters no longer being able to obtain uniform variance, since the direction in which cells contract/expand can conflict with each other. Thus, we need to lessen the impact of the transformation with distance to the Voronoi cell. This way clusters can be re-shaped into uniform variance, but not deter the transformation of a different cell. If the impact is lessened continuously, close data points with different cell-assignments would stay close and transformation would not cause rips in the data.

We now need to cover some basics. The formula for the Z-transformation is given by Eq. (1). We want to keep the cells roughly where they are, so we need to move a data point back to where it came from; thus, we add μ, the mean of the feature, to the data point.

$$x' = \frac{x - \mu}{\bar{\sigma}} + \mu \tag{3}$$

This essentially just expands/contracts the data points in a specific direction, but leaves the center of the data points where it is. In our setting, μ is equivalent to the projection of the cluster-center as found by k-means onto the principal component found by PCA, because we wish to scale the clusters in the specific directions of PCA, but not to move the position of the clusters. The intention is to force the clusters into a shape with uniform variance, but not necessarily to change their position.

Our main interest is the effect of the transformation depending on the distance to the center of the cluster. Thus, we compute how much a data point is moved, depending on the distance to the center.

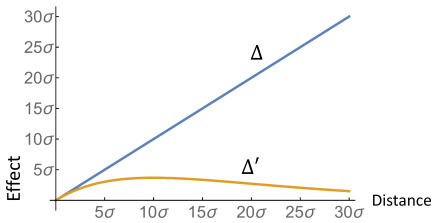


Figure 4: The effect of Δ and Δ' depending on the distance to the center of the cluster. The effect of Δ' is very similar to Δ up to a distance of $\approx 2\ \bar{\sigma}$ from μ, but the farther a data point is from the center of the Voronoi cell, the smaller the effect of the cell becomes.

$$\Delta = x' - x = \frac{x-\mu}{\bar{\sigma}} + \mu - x = \\ = (x-\mu)\left(\frac{1}{\bar{\sigma}} - 1\right) = d(x,\mu)\left(\frac{1}{\bar{\sigma}} - 1\right) \quad (4)$$

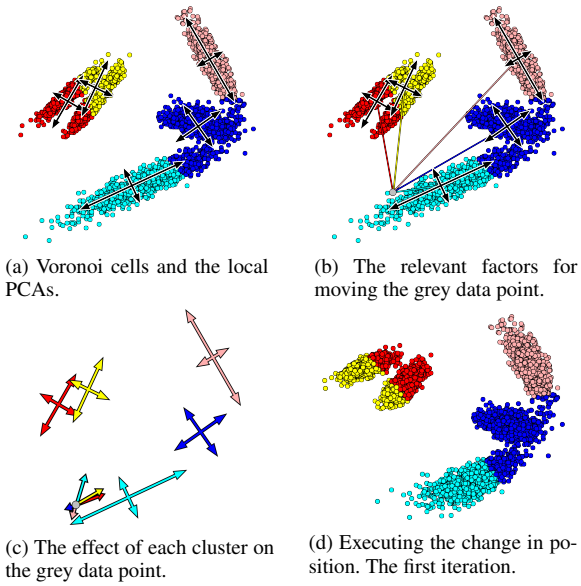
Δ is the difference between the old and the new position of the data point. $d(x,\mu)$ is the distance of x from μ. We see that the change of the data point is linearly dependent on the distance of the data point to the center of the transformation. We want to weaken this effect with distance. Instead of a linear, the effect should increase continuously and sub-linear, eventually declining to 0 with distance to the center. Discontinuity would mean rips in the data set. A straightforward solution, which worked well in the experiments, is to add a factor to Eq. (4) that ensures that the effect declines with distance, i.e. we add an exponential decay factor to it.

$$\Delta' = \frac{d(\mu,x)}{e^{\frac{d(\mu,x)}{(10\bar{\sigma})}}}\left(\frac{1}{\bar{\sigma}} - 1\right) \quad (5)$$

With the added decay factor, the influence of the transformation slows continuously with distance to the center of the Voronoi cell. The explicit form can be seen in Fig. 4. Close to the center of the cell, roughly in an area of $2\bar{\sigma}$, which is about 95% in a Gaussian distribution, the transformation behaves just like a normal Z-transformation. This effect diminishes with distance to the center and starting from a distance of $10\bar{\sigma}$ the effect becomes smaller and smaller in contrast to the effect given by Eq. (4). Clusters with different orientations can now obtain uniform variance.

We noticed that sometimes if multiple clusters are in a Voronoi cell the algorithm produces a sort of oscillating effect in the transformation. This is most likely due to the adapted standard deviation in Eq. (2). To avoid this we slow the transformation down. We doubled the speed in Eq. (2), so now we halve it and only apply $\frac{\Delta'}{2}$.

The factor $(\frac{1}{\bar{\sigma}} - 1)$, which has not been altered from Eq. (4) to Eq. (5), is basically the "sign" of the transformation. It determines if the data is stretched ($\bar{\sigma} < 1$), contracted ($\bar{\sigma} > 1$) or is left as is ($\bar{\sigma} = 1$), i.e. the data is already uniform in this direction. The ultimate goal is to bring all cluster in all directions to a uniform variance. This can be described as $\sum_{j=1}^{k}\sum_{l=1}^{d}|\bar{\sigma}_{lj} - c| = 0$, with $\bar{\sigma}_{lj}$ being the adapted variance in direction l for cell j and c a constant. This formula is the intrinsic objective function of PCE. It formalizes the intuition that all clusters should be brought to uniform variance. If the objective function is 0, PCE would stop, as no more change in the position of the data points is necessary. The uniform variance does not need to be 1. Any constant value c would be suitable, as we merely try to obtain the same variance in all directions, not necessarily $\bar{\sigma}_{lj} = 1$.



(a) Voronoi cells and the local PCAs.

(b) The relevant factors for moving the grey data point.

(c) The effect of each cluster on the grey data point.

(d) Executing the change in position. The first iteration.

Figure 5: Applying the algorithm - the first iteration.

Algorithm 1 PCE

Require: Data D, number of clusters k
1: **procedure** PCE(D,k)
2: **while** Cluster assignments change **do**
3: Execute k-means on D
4: **for** j = 1,...,k **do**
5: Compute local PCA for Cluster j
6: Compute variance for main components of PCA
7: **end for**
8: **for** i = 1,...,n **do**
9: **for** j = 1,...,k **do**
10: Compute effect of Z-transformation along the direction of principal components of PCA from Cluster j on data point i with Eq. (5).
11: **end for**
12: **end for**
13: **for** i = 1,...,n **do**
14: Sum up the effects of Z-transformation/PCA from all Clusters on data point i as computed in Line 10 and execute it.
15: **end for**
16: **end while**
17: **return** D
18: **end procedure**

We illustrate the approach of PCE by going through the pseudo code with the running example. After executing k-means (Line 3 in the Pseudo-code in Algorithm 1) it is easy to see that clusters and Voronoi cells do not match (Fig. 5a). We use random initialisation (see Section 4). PCA is computed in all Voronoi cells (Line 5) and the effects of these local PCAs on the data points are computed (Line 10). We take the grey data point as an example. All relevant information for it is shown in Fig. 5b. To compute the effects of the clusters on it, we first project it onto the main components found by PCA and compute the distance to the centers. These distances give us with Eq.

5 the effect of the cells on the grey data point. In Fig. 5c we can see how the cells influence the grey data point, i.e. how they move it. Adding these changes up results in the overall effect on the grey data point. Computing these effects for all data points (Line 9-12) and executing them (Line 13-15) leads to Fig. 5d. The first iteration of PCE. The clusters are now closer to the Gaussian bubble shape with uniform variance they ideally have. The algorithm continues as one iteration might not make the clusters perfectly uniform in variance. K-means is updated, i.e. executed with the same centers, which have also been moved. Every iteration brings the clusters closer to uniform variance, and, eventually, the cluster assignments do not change any more. Identical cluster assignments in two successive iterations are interpreted as a stable configuration (Line 2/16), i.e. the clusters have become of such uniform variance, that another iteration would not improve the data any more. In the final result (Fig. 1c) the clusters are perfectly compact and suited for k-means.

2.4 Runtime

Following the pseudo code given in Alg. 1 we can estimate the runtime of the algorithm to be:
$\mathcal{O}(i \cdot (n \cdot k \cdot d + k(\mathcal{O}(PCA) + n \cdot log(n) \cdot d) + n \cdot k \cdot d + n))$
with i the number of iterations, d the dimensionality of the data set, k the number of clusters and n the number of data points. We left PCA as its own factor as it is by far the largest part in regards of runtime. PCA is $\mathcal{O}(d^3 + d^2 \cdot n)$, thus summed up the runtime is

$$\mathcal{O}(i \cdot k \cdot d^3 \cdot n \cdot log(n))$$

The cubic runtime in d seems excessive, but it is a surprisingly small issue as this is caused solely by PCA and the runtime would be linear without it. PCA is a standard data mining algorithm for which highly optimized and parallelized implementations exist. High-level APIs and distributed computing frameworks perform PCA with impressive speed and even a matrix with millions of entries can be decomposed in mere seconds [3]. There exist also approximative algorithms for PCA [13] reducing the runtime to $O(d^2\ log(d))$. The implementation of PCA in the commonly used scikit-learn package uses this implementation. So, while PCA is definitely the bottleneck for our algorithm, the cubic estimation is essentially a worst-case scenario, which is only encountered for the naïve implementation of PCA and, thus, easily avoided.

3 EXPERIMENTAL EVALUATION

Our goal is to make data sets more suited for k-means by making the clusters more compact. The direct result of this should be a notable increase in clustering quality when comparing k-means before and after PCE. Furthermore, this increase should be large enough for k-means to beat a wide range of various clustering algorithms. Thus, we tested our method on various publicly available real world data sets from the UCI repository [8] and compared with classical clustering method, like DBSCAN and SingleLink, as well as state-of-the-art approaches. The results can be seen in Table 1.

We measured clustering quality in Normalized Mutual Information (NMI) [27], which is currently widely used to estimate the success of a clustering method. NMI scales between 1.0 (perfect clustering result) and 0.0 (purely random cluster assignments).

On average, k-means improves by 0.11 in NMI. This is the notable increase we were looking for and it made k-means the best-performing method on the data sets. It is now outperforming a wide spectrum of clustering approaches. PCE managed to make the clusters better suited to k-means and to re-shape the clusters so that they fit into the Voronoi cell structure of k-means. In Fig. 1c we saw how well the clusters became suited for k-means and in the experiments we see that the effect of improving k-means is not limited to synthetic data.

Following the argumentation in [23] that a method that improves k-means will most likely also improve other methods, we also tested our approach with other standard clustering methods. In Table 2 we choose 4 of the data sets used in Table 1 and 4 of the standard clustering approaches, i.e. EM, SingleLink, DBSCAN, Spectral Clustering as well as k-means++ and tested by how much PCE could improve their clustering results. We also included other transformations and tested how much they could improve the clustering results. PCE is used here as a pre-processing step, which explains the choice of the other methods. We used here k-means++ instead of k-means, but their results barely differ (see Section 4).

From the 20 comparisons in Table 2 testing data sets and methods, PCE was the best choice in 18. An overview of these results for all of the 7 real-world data sets for these methods can be seen in Table 3. This is basically an abridged version of Table 2. The conclusion is the same: The quality of clustering could be notably improved. PCE could enhance the shape of the clusters, such that the data sets became easier to cluster. It is the best pre-processing step here and highly compatible with all of the 4 classical methods. EM can, to a degree, counteract stretched cluster, thus PCE is more effective on some data sets for it and has less influence on others. Interestingly, k-means performs better than EM and has on average a lead of 0.04 over EM. We wish to point out, that Spectral Clustering and SingleLink improve by 0.12 and 0.13 respectively (more than k-means).

In consideration of all these experiments, we can state that we succeeded in making these data sets easier accessible to clustering methods by moving data points so that clusters become more compact and easier to find.

3.1 Parameters

Data sets were normalized in the [0,1]-range for all methods. We tried to be as fair as possible to all comparison methods. Transformation methods like PCA are all shown in combination with k-means. Methods like k-means++, which entail random aspects, have been executed 100 times and the average NMI is given. The correct number of clusters, which e.g. EM needs, is always given to the algorithm. EM is initialized with k-means as that lead to better results. DBSCAN is difficult to parametrize and thus, we follow the lead from [23]. We computed the average distance between data points, a, and tested every combination of $minPts \in \{1, 2, 3, 5, 10, 50\}$ and $\epsilon \in \{0.05 \cdot a, 0.1 \cdot a, 0.2 \cdot a, 0.4 \cdot a, 0.6 \cdot a, 0.8 \cdot a, a\}$. Only the best NMI is reported. For PCA and ICA we followed the lead in [20]. Thus, for PCA we use the often-applied setting that 90% of the variance is kept and for ICA the number of dimensions is the number of clusters. We also tried keeping all dimensions, but this did not change the message of the reported results (PCE performed still better on all data sets). These results are not included due to space-restrictions. For SingleLink we found that CompleteLinkage lead to clear better results compare to SingleLinkage and is thus used here as the cluster creation criterion. EWKM has a parameter λ which should be chosen in the range $[1, 3]$. We tried for each run $\lambda \in \{1, 2, 3\}$ and took the best result. For kernel k-means we tried Gaussian and Polynomial kernels, as they are two of the more popular choices. DEC and IDEC are difficult to parametrize. We pretrained ten autoencoders for each

Table 1: Comparison between methods measured in NMI. For non-deterministic methods is the average of 100 runs given. The correct number of clusters is given for every methods that can use it. Parameters and technical details in Section 3.1. Best result shown in bold.

Data set	Iris	Vertebrae C.	Seeds	Wifi-local.	Breast Cancer	Breast Tissue	Wine	Run. Ex.
PCE	0.84	**0.52**	**0.82**	**0.91**	**0.81**	**0.55**	**0.88**	**0.93**
k-means	0.71	0.26	0.67	0.82	0.74	0.50	0.84	0.76
DipTransformation	0.84	0.29	0.78	0.65	0.72	0.51	0.71	0.67
DipScaling	0.81	0.27	0.73	0.69	0.73	0.52	0.72	0.61
Z-Transformation	0.64	0.30	0.74	0.80	0.73	0.49	0.86	0.76
PCA	0.74	0.27	0.66	0.83	0.74	0.46	0.85	0.75
ICA	0.57	0.27	0.63	0.61	0.65	0.43	0.76	0.75
k-means++	0.72	0.26	0.67	0.81	0.74	0.49	0.84	0.77
Spectral	0.59	0.27	0.60	0.77	0.79	0.49	0.87	0.89
STSC	0.58	0.27	0.53	0.84	0.36	0.30	0.86	0.42
IDEC	0.25	0.11	0.29	0.40	0.17	0.30	0.22	0.46
DEC	0.24	0.11	0.28	0.43	0.17	0.30	0.23	0.47
DipMeans	0.58	0.00	0.44	0.84	0.29	0.00	0.00	0.76
SynC	0.58	0.13	0.48	0.80	0.64	0.29	0.59	0.48
DBSCAN	0.61	0.21	0.42	0.49	0.76	0.44	0.42	0.88
SingleLink	0.74	0.17	0.61	0.49	0.48	0.39	0.78	0.67
EM	**0.87**	0.49	0.64	0.90	0.54	0.48	0.86	0.89
FossClu	0.73	0.07	0.57	0.87	0.43	0.36	0.61	—
SubKMeans	0.66	0.30	0.73	0.80	0.73	0.45	0.87	0.77
kernel k-m. (Gauss.)	0.66	0.27	0.67	0.80	0.40	0.44	0.83	0.78
kernel k-m. (Polyn.)	0.69	0.25	0.63	0.69	0.75	0.49	0.74	0.69
EWKM	0.67	0.25	0.60	0.71	0.14	0.49	0.58	0.73

Table 2: 4 of the real-world data sets from Table 1 and we show how PCE fares on them in combination with clustering methods besides k-means in comparison with the most closely related methods.

EM	Vertebrae.	Wifi-local.	BreastTissue	Wine
PCE	**0.53**	0.89	**0.52**	**0.88**
orig	0.49	0.90	0.48	0.87
PCA	0.27	0.90	0.49	**0.88**
ICA	0.29	0.82	0.41	0.84
DipTrans.	0.15	0.90	0.48	0.83
Z-trans	0.49	**0.92**	0.47	0.85
SingleLink				
PCE	**0.45**	**0.87**	**0.50**	**0.83**
orig	0.17	0.49	0.39	0.78
PCA	0.17	0.64	0.41	0.47
ICA	0.03	0.13	0.23	0.02
DipTrans.	0.16	0.71	0.36	0.39
Z-trans	0.02	0.40	0.38	0.61
DBSCAN				
PCE	**0.33**	0.44	**0.52**	**0.55**
orig	0.21	**0.49**	0.44	0.42
PCA	0.18	0.40	0.45	0.50
ICA	0.18	0.36	0.40	0.46
DipTrans.	0.18	0.47	0.46	0.52
Z-trans	0.26	0.46	0.44	0.43
Spectral				
PCE	**0.49**	**0.85**	**0.52**	**0.88**
orig	0.27	0.76	0.50	0.87
PCA	0.26	0.75	0.50	0.75
ICA	0.28	0.63	0.45	0.84
DipTrans.	0.21	0.76	0.49	0.83
Z-trans	0.32	0.74	0.50	**0.88**
k-means++				
PCE	**0.50**	**0.91**	**0.53**	**0.87**
orig	0.27	0.84	0.50	0.84
PCA	0.27	0.83	0.46	0.84
ICA	0.28	0.59	0.43	0.85
DipTrans.	0.24	0.78	0.51	0.75
Z-trans	0.30	0.80	0.46	**0.87**

Table 3: Average NMI of standard clustering methods on the 7 real world data sets from Table 1.

	EM	SingleL.	Spectral	DBSCAN	k-means++
PCE	**0.71**	**0.65**	**0.75**	**0.51**	**0.75**
orig.	0.68	0.52	0.63	0.48	0.66
PCA	0.61	0.50	0.52	0.48	0.65
ICA	0.59	0.12	0.60	0.43	0.59
DipTrans.	0.62	0.50	0.64	0.50	0.65
Z-trans.	0.67	0.45	0.63	0.47	0.65

data set as described in [29] and setting the latent dimension equal to the number of clusters. DEC and IDEC were each run on the ten pretrained autoencoders for 200 epochs, which ensured convergence.

4 DISCUSSION AND CONCLUSION

In the following, we discuss point by point some of the aspects of PCE and its relation to other methods.

Source code: Source code, data sets and labels can be found here: `https://dm.cs.univie.ac.at/research/downloads/`

Continuity: The effect of a single direction in a cell on a data point is given with Eq. 5. Summarizing these effects for all k Voronoi cells and d directions gives:

$$PCE_i(x) = \sum_{j=1}^{k} \sum_{l=1}^{d} \left(x + \frac{1}{2} \Delta'(d(\mu_j, x)_l, \bar{\sigma}_{lj}) \right)$$

where $d(\mu_j, x)_l$ is the distance of the projection of x onto the lth direction of PCA and $\bar{\sigma}_{lj}$ the adapted variance in this direction. This is the formula for a single iteration of PCE. It is easy to see from it that PCE is continuous, i.e. data points that are close before will be close afterwards.

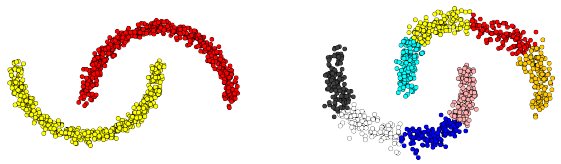


Figure 6: An unfitting data set with a wrong number of clusters for PCE. Orig. data on the left, PCE-result on the right.

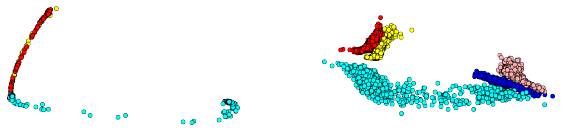


Figure 7: The running example with DEC and IDEC. Rotated by 90°. Plotted with groundtruth-label to help recognition.

Theorem 1 *PCE is continuous.*

Proof: Let $y = x + \epsilon$ with $x, y, \epsilon \in \mathbb{R}^d$.
$|PCE_i(y) - PCE_i(x)| = |PCE_i(x + \epsilon) - PCE_i(x)|$
$\sum_{j=1}^{k} \sum_{l=1}^{d} \left(\epsilon + \frac{1}{2}\Delta'(d(\mu_j, x + \epsilon)_l, \bar{\sigma}_{lj}) - \frac{1}{2}\Delta'(d(\mu_j, x)_l, \bar{\sigma}_{lj})\right)$
Since Δ' is continuous itself, $|PCE_i(y) - PCE_i(x)| \to 0$ for $\epsilon \to 0$ and thus a single iteration is continuous. As the concatenation of continuous functions is again continuous, PCE is continuous. ■

This proof is valid for a wide range of possible Δ'-functions. Our Δ' is the standard Z-transformation with an adapted variance and an exponential decay factor. In the proof, we only used that our Δ' is continuous. Thus, this proof also holds e.g. for a logarithmic or polynomial decay factor or a completely different definition of Δ', as long as it is continuous.

Continuity is a crucial feature for a transformation as it ensures that the local neighbourhood is kept the same in at least an approximative fashion. Distances between data points change, but if data points are close, they will still be close after the transformation. As a practical example, consider the data shown in Fig. 6. It is unsuited to PCE as its clusters are too far from a convex shape. PCE is not capable of separating the clusters, but it will not cause any "damage" (we used a wrong number of clusters, but similar behaviour could also be observed if the number of clusters is given as 2, 3, . . .). The continuity and that PCE merely stretches and contracts data points causes a very careful transformation, which 1) does not produce rips in the data, 2) keeps the basic shape intact and 3) is unlikely to transform the shape if not enough information is available, i.e. if the measured variance/dip-values are not deviating relevantly from each other. If this is the case, PCE is unsuited for the data set; resulting in PCE refraining from doing anything relevant. Fig. 6 depicts such a case. PCE is unsuited for the data set, but it does not change much and the data set becomes no more difficult to cluster. PCE is "careful", if not enough information is present for it.

Contrary to PCE, Deep Learning-based methods are far more extreme in their transformation approach. In Fig. 7 is shown how DEC/IDEC transform the running example. The running example is a rather simple data set, but DEC/IDEC restructure it completely. Neural Networks often seem like a black box, where many decisions are barely comprehensible. This is such a case. The basic shape is heavily distorted and the structure of the clusters now unrecognisable. The advantage of PCE over Deep Learning-based methods, which are almost the only other non-linear transformations there are, is 1) its approach and decision-making is easy to understand 2) it is far more conservative and refrains from destroying structure.

Non-convex clusters: We have seen in Fig. 6 that PCE cannot handle all types of data sets. If the clusters are too far from a convex shape or are massively overlapping, PCE will have problems. Though, it will most likely refrain from acting in such a case, which does not deterriorate things. It might be possible to adapt PCE towards these cases by employing other ways of estimating the local shape of data points instead of PCA. We intend to analyze this in future works.

The effect of Initialisation: PCE, as well as k-means or EM, is deterministic after the initialisation has been decided. This raises the question of the effect the initialisation has. The two main initialisation methods for k-means are random initialisation (RI), where the centers are assigned a random data point, which is used in this paper, and k-means++. K-means++ is often the gold-standard and improves over RI, but this is not the case here. The average difference between RI to k-means++ on these data sets is for k-means merely 0.005 in NMI, with k-means++ being slightly **worse**. The same effect can be seen with EM (0.006 in NMI) and PCE (0.006 in NMI). That is not to say that initialisation is not a relevant factor. Taking the best of 10 runs of k-means (according to the objective function of k-means) leads to an average improvement of 0.011 in NMI on the real world data sets for k-means evenly distributed on all data sets. The same strategy improves EM by about 0.005 and PCE even by 0.019 in NMI.

On the tested data sets, PCE could notably improve the results. This means, that it is rather likely that there are stretched clusters in the data, comparable to the running example. K-means++ chooses new centers based on distances, which means that it might choose two data points from such a stretched cluster, or give two close clusters like the yellow/red one in the running example only one starting center. RI does not take distances into account, which means that the stretch of the clusters makes no difference, whether a cluster gets a starting center. Thus, on these types of data sets k-means++ is not per se the best choice of initialisation strategy.

Regarding EM and objective functions: One variation of the retort mentioned in the introduction is "Why not use EM instead?" EM can, to a degree, take care of stretched clusters and thus, overcome the Voronoi cell structure of k-means. This ignores, that EM is a clustering method, while PCE is a transformation approach, that can be used as a pre-processing step for clustering methods. It gives EMs ability to handle stretched clusters to methods like k-means or SingleLink. Also, PCE improves k-means so far, that it is on average better by 0.10 in NMI compared to EM.

Furthermore, PCE can also improve EM. PCE changes the position of data points, which changes the optima towards which an algorithm can converge to. Thus, the loss landscape of the objective function itself is changed. This opens stable configurations which the algorithm could not reach before. The Seeds-data set, for example, improves for EM from 0.64 to 0.78 in NMI. We intend to analyse this change in local optima more thoroughly in future works.

Table 4: The NMI-values for the running example for k-means before and after PCE for wrong values of k. Better result bold.

	k=2	k=3	k=4	k=6	k=7	k=8	k=9	k=10
k-m.	0.35	0.62	0.71	0.76	0.73	0.70	0.68	0.66
PCE	**0.36**	**0.63**	**0.81**	**0.89**	**0.84**	**0.79**	**0.75**	**0.71**

Wrong k: What happens if k is wrong? We tested the running example for wrong values of k and could still observe an improvement (Table 4). The same holds for the real-world data sets. We set $k \pm 1$ its real value. In both cases, we still got an increase in average NMI.

PCE optima: Different optima of k-means can lead to different final transformations, and similar k-means optima can lead PCE to similar final transformations. Sometimes, though, also different optima will lead with PCE to similar final transformations. Our working hypothesis is that PCE has – similar to k-means – various stable states to which it can converge to, satisfying the objective function, i.e. having uniform variance in all directions. This is a question which we will analyze in more detail in the future.

Conclusion: The usual approach in Data Mining is to find a clustering method for a data set and, if none fits, to create a new approach. PCE is the other way around; if the data set does not fit, we make it fit into the assumptions of the clustering method. We devised a method that iteratively re-shapes the clusters, moves them further apart from each other and makes them more compact by forcing them into a shape with uniform variance. We tested PCE with extensive experiments and showed that it also holds up under real-world conditions, where clusters are usually messier than in synthetic examples. It improved not only k-means but also the standard clustering methods. Since they approach clustering in very different ways, we assume that a wide range of algorithms could benefit from PCE.

REFERENCES

[1] 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 '07, pp. 1027–1035, Philadelphia, PA, USA, (2007). Society for Industrial and Applied Mathematics.

[2] Christian Böhm, Claudia Plant, Junming Shao, and Qinli Yang, 'Clustering by synchronization', in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pp. 583–592, New York, NY, USA, (2010). ACM.

[3] Reza Bosagh Zadeh, Xiangrui Meng, Alexander Ulanov, Burak Yavuz, Li Pu, Shivaram Venkataraman, Evan Sparks, Aaron Staple, and Matei Zaharia, 'Matrix computations and optimization in apache spark', in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 31–38. ACM, (2016).

[4] Theofilos Chamalis and Aristidis Likas, 'The projected dip-means clustering algorithm', in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, SETN '18, pp. 14:1–14:7, New York, NY, USA, (2018). ACM.

[5] Yu-An Chung, Wei-Hung Weng, Schrasing Tong, and James Glass, 'Unsupervised cross-modal alignment of speech and text embedding spaces', in *Advances in Neural Information Processing Systems 31*, eds., S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 7354–7364. Curran Associates, Inc., (2018).

[6] Pierre Comon, 'Independent component analysis, a new concept?', *Signal Processing*, **36**(3), 287 – 314, (1994). Higher Order Statistics.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, 'Maximum likelihood from incomplete data via the em algorithm', *Journal of the Royal Statistical Socoety, Series B*, **39**(1), 1–38, (1977).

[8] Dheeru Dua and Casey Graff. UCI machine learning repository, 2019.

[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani, *The elements of statistical learning*, Springer series in statistics New York, 2001.

[10] Karl Pearson F.R.S., '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).

[11] Sebastian Goebl, Xiao He, Claudia Plant, and Christian Böhm, 'Finding the optimal subspace for clustering', in *Proceedings of the 2014 IEEE International Conference on Data Mining*, ICDM '14, pp. 130–139, Washington, DC, USA, (2014). IEEE Computer Society.

[12] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin, 'Improved deep embedded clustering with local structure preservation', in *IJCAI*, pp. 1753–1759, (2017).

[13] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp, 'Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions', *SIAM review*, **53**(2), 217–288, (2011).

[14] Jiawei Han, Jian Pei, and Micheline Kamber, *Data mining: concepts and techniques*, Elsevier, 2011.

[15] J. A. Hartigan and P. M. Hartigan, 'The dip test of unimodality', *Ann. Statist.*, **13**(1), 70–84, (03 1985).

[16] L. Jing, M. K. Ng, and J. Z. Huang, 'An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data', *IEEE Transactions on Knowledge and Data Engineering*, **19**(8), 1026–1041, (Aug 2007).

[17] Duc Le, Zakaria Aldeneh, and Emily Mower Provost, 'Discretized continuous speech emotion recognition with multi-task deep recurrent neural network.', in *INTERSPEECH*, pp. 1108–1112, (2017).

[18] J. B. MacQueen, 'Some methods for classification and analysis of multivariate observations', in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, eds., L. M. Le Cam and J. Neyman, volume 1, pp. 281–297. University of California Press, (1967).

[19] 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*, KDD '16, pp. 1055–1064, New York, NY, USA, (2016). ACM.

[20] 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*, KDD '17, pp. 365–373, New York, NY, USA, (2017). ACM.

[21] Ismail Bin Mohamad and Dauda Usman, 'Standardization and its effects on k-means clustering algorithm', *Research Journal of Applied Sciences, Engineering and Technology*, **6**(17), 3299–3303, (2013).

[22] Dau Pelleg and Andrew Moore, 'X-means: Extending k-means with efficient estimation of the number of clusters', in *In Proceedings of the 17th International Conf. on Machine Learning*, pp. 727–734. Morgan Kaufmann, (2000).

[23] Benjamin Schelling and Claudia Plant, 'Diptransformation: Enhancing the structure of a dataset and thereby improving clustering', in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 407–416, (Nov 2018).

[24] Benjamin Schelling and Claudia Plant, 'Dataset-transformation: improving clustering by enhancing the structure with dipscaling and diptransformation', *Knowledge and Information Systems*, (Aug 2019).

[25] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller, 'Nonlinear component analysis as a kernel eigenvalue problem', *Neural computation*, **10**(5), 1299–1319, (1998).

[26] 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 and Data Mining*, KDD '18, pp. 2210–2218, New York, NY, USA, (2018). ACM.

[27] Nguyen Xuan Vinh, Julien Epps, and James Bailey, 'Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance', *J. Mach. Learn. Res.*, **11**, 2837–2854, (December 2010).

[28] F. Wang, N. Zheng, D. Cao, C. M. Martinez, L. Li, and T. Liu, 'Parallel driving in cpss: a unified approach for transport automation and vehicle intelligence', *IEEE/CAA Journal of Automatica Sinica*, **4**(4), 577–587, (2017).

[29] Junyuan Xie, Ross Girshick, and Ali Farhadi, 'Unsupervised deep embedding for clustering analysis', in *Proceedings of The 33rd International Conference on Machine Learning*, eds., Maria Florina Balcan and Kilian Q. Weinberger, volume 48 of *Proceedings of Machine Learning Research*, pp. 478–487, New York, New York, USA, (20–22 Jun 2016). PMLR.

[30] 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*, pp. 3861–3870, (2017).

Appendix F

Paper F: Utilizing Structure-rich Features to improve Clustering

Utilizing Structure-rich Features to improve Clustering

Benjamin Schelling[1,2,3] ✉, Lena Greta Marie Bauer[4], Sahar Behzadi[3], and Claudia Plant[3,4]

[1] MCML, Munich, Germany
[2] Ludwig-Maximilians-Universität München, Munich, Germany
[3] Faculty of Computer Science, University of Vienna, Vienna, Austria
[4] ds:UniVie, Vienna, Austria
firstname.lastname@univie.ac.at

Abstract. For successful clustering, an algorithm needs to find the boundaries between clusters. While this is comparatively easy if the clusters are compact and non-overlapping and thus the boundaries clearly defined, features where the clusters blend into each other hinder clustering methods to correctly estimate these boundaries. Therefore, we aim to extract features showing clear cluster boundaries and thus enhance the cluster structure in the data. Our novel technique creates a condensed version of the data set containing the structure important for clustering, but without the noise-information. We demonstrate that this transformation of the data set is much easier to cluster for k-means, but also various other algorithms. Furthermore, we introduce a deterministic initialisation strategy for k-means based on these structure-rich features.

1 Introduction

Clustering is the task of grouping data points and dividing a data set into the correct partitioning. For this, it is most important that the clusters can be distinguished from each other and the boundaries between them are easily found. Useless information can make this difficult for many if not all clustering approaches, causing a faulty analysis of the data, as the clusters can not be separated. To battle this difficulty we propose an approach that extracts the features with the highest amount of information relevant for clustering and constructs a condensed data set, which is easier to cluster.

Features where the clusters do not blend into each other, are those important for clustering. If the clusters are well separated, it is possible to determine boundaries between them, enabling a good clustering result. Concentrations of data points show that there are clusters present and a meaningful partitioning can be found. They also reveal where to draw the boundaries.

Let us demonstrate this using the running example shown in Fig. 1, consisting of three non-overlapping Gaussian clusters and a third feature made up of uniform noise. The first axis of the running example is shown in Fig. 2. It contains hardly any information useful for clustering as it is unimodal, meaning that

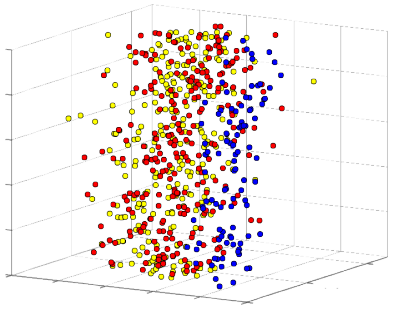


Fig. 1. The running example in a 3D scatterplot.

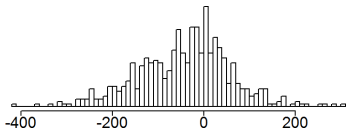


Fig. 2. Histogram of the first axis of the running example.

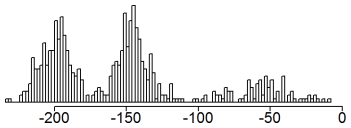


Fig. 3. Histogram of the feature with the highest dip value.

the clusters overlap and have no clear boundaries. In contrast, the projection of the data set on a feature that has clear boundaries between clusters is shown in Fig. 3. It is far more relevant for clustering, as it shows where the data set should be partitioned. This feature contains all the information necessary for clustering, as the clusters are almost perfectly separated in this projection. K-means, for instance, finds the correct partitioning based solely on this feature. However, this projection is not an arbitrary one, it is the projection with the maximal possible dip value in this specific data set. The dip test [8] is a parameter-free statistical test developed in the 1980s which measures how much a one-dimensional sample deviates from uni-modality. Multi-modality indicates clear cluster boundaries, making the dip test a useful tool to estimate how much information relevant for clustering is contained in a feature.

Our approach DipExt (dip test based extraction) searches for and extracts structure-rich features. DipExt creates a lower-dimensional representation containing the structure of the data set in a condensed form. The 2 features with the highest dip values, as found by our approach, can be seen in Fig. 4. The second feature is not needed regarding the clustering goal, but we include it for visualisation-purposes. Also shown are the 2D representations of the data set

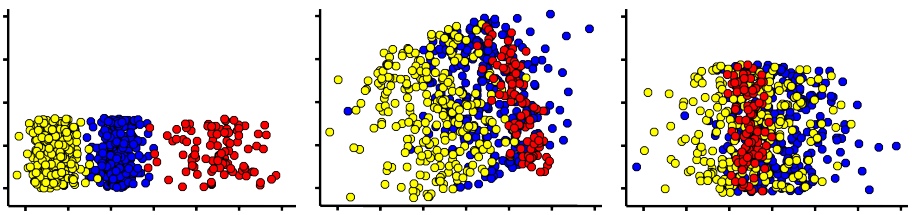


Fig. 4. The running example in 2D with our method (left), PCA (middle) and ICA (right).

obtained by PCA and ICA. The difference is clear: While our approach produces a lower-dimensional representation which is trivial to cluster, PCA and ICA do not. In contrast to them, our method explicitly searches for *structure*, i.e. information relevant for clustering, while PCA searches for variance and ICA for statistical independence. Neither of the latter two guarantees that the found features are important for clustering. Features with high dip values, on the other hand, contain separated concentrations of data points, i.e. densely grouped data points. Therefore, they most likely carry the information relevant for clustering.

Our goal is to find a lower-dimensional transformation of the data which contains the cluster-information. DipExt extracts the features with structure and combines them, creating a condensed form of the data, which contains only the information relevant for clustering. After these features are found we re-scale them according to their relevance for clustering, i.e., with their dip value. This scaled subset of features comprises the information from the data processed for clustering, so that, for example, k-means can find the correct clustering more easily. Additionally, we also present an initialisation strategy for k-means called DipInit (dip test based Initialisation), to ensure that k-means converges to a suitable optimum. By clustering features with high dip values first, DipInit ensures that these features have a higher impact on the clustering result. Thus DipInit makes full use of structure-rich features. It is deterministic and based on our assumption that the dip value can determine the importance of a feature for clustering. It is highly compatible with DipExt and makes full use of its characteristics, but it is also very competent on its own.

1.1 Related Work

DipExt finds a subset of the features, extracting those relevant for clustering. Thus, it could be counted as a Subspace Clustering-method (see [12] for an introduction to subspace clustering). The difference between DipExt and many Subspace Clustering methods is that they generally look for a separate subspace for every cluster, while DipExt looks for a common subspace for all clusters. The goal to find one optimal subspace, valid for all clusters, is a more recent trend in subspace clustering (and, therefore, not mentioned in [12]). This trend is sometimes referred to as "cluster-aware" [26] or **"cluster-friendly"** [27] **subspace clustering**/dimensionality reduction. Examples are SubKMeans [17] or FossClu [6]. SubKMeans separates the features into a "cluster-friendly" subspace and one which does not contain features important for clustering [17]. FossClu proceeds similarly and removes unimodal features based on its objective to create an optimal subspace. Autoencoder-based approaches like DCN [27] are at the forefront of this subspace clustering trend. DCN explicitly looks for a "k-means friendly space" via its objective function. DCN, along with DEC [25] and IDEC [7], was one of the first deep learning-based methods combining dimensionality reduction with clustering. They transform the data non-linearly, making the data difficult to interpret and sometimes lead to extremely distorted representations.

The best known "general" dimensionality reduction methods are PCA and ICA. The main difference between this "general" dimensionality reduction and "cluster-friendly" dimensionality reduction is the consideration of cluster structure in the found subspace. While "cluster-friendly" subspace clustering methods try to find a subspace suitable explicitly for clustering, PCA and ICA are general methods used in many areas of Data Mining. Further examples are, t-SNE [13], which tries to preserve the neighbourhoods of data points while projecting to smaller dimensionalities, or UMAP [15], both of which are non-linear.

The dip test [8] was first used in data mining in DipMeans [10], to estimate the number of clusters. In SkinnyDip [16] it is used to cluster noisy data. Recently, it has been generalised to higher dimensions in [22] and [3]. However, in [22] it is not used as a clustering algorithm, but as a criterion of whether clustering makes sense. Both these generalisations effectively apply the one-dimensional dip test with a criterion to select the next data point in a multi-dimensional data set. DipTransformation [19] reshapes the data set to an easier clusterable form making use - as will we - of DipScaling [20]. It transforms the data by changing the position of the data points to make clusters more compact. It does not, however, change the dimensionality of the data. DipScaling is closly related to normalisation methods like Z-transformation or min-max-normalisation. It can be also subsumed under feature weighting methods, like EWKM [9], which assign a "weight" for the importance for clustering to a feature.

As stated, part of our approach is an initialisation strategy for k-means - DipInit. K-means is very sensitive to its initialisation as this determines to which local optimum k-means converges. The most common strategies are k-means++ [1] and random initialisation. They are, like most other strategies (see [2] for an overview), based on random effects. Contrary to that, DipInit is deterministic. It employs the dip test to start clustering on features, where the clusters are well separated, allowing k-means to find better optima.

1.2 Contributions

- We present a deterministic method, DipExt, which extracts the features with the highest level of structure in a data set and rescales them. It creates a condensed version of the data set, containing the information needed for clustering. This version is far easier to cluster for k-means, on which we focus, but also for other methods, as we will show.
- This condensed version has often a considerably smaller dimensionality. It is possible to get an arbitrary dimensionality as specified by the user, or to find it **automatically**. We demonstrate that our approach is stable in regard to choosing the dimensionality.
- Initialisation-strategies for k-means mostly include random components. **DipInit** is an initialisation strategy which is **deterministic** and makes use of the specific properties of the subspace found by DipExt. It puts the main focus on features with high dip-values, helping to converge to a better local optimum.

2 The Algorithm(s)

2.1 The Dip Test - How much Structure is in a Feature?

As stated, the goal is to find the features with the highest dip value. First, we cover a few basics about the dip test.

The dip test is a statistical tool developed in the 1980s by Hartigan & Hartigan to measure multi-modality. It is a well established and **parameter-free** test to measure the probability of a univariate sample to be multi-modal. Essentially, it tells the user whether a sample has multiple peaks/modes in it, like the one shown in Fig. 3 or if there is only one peak like in Fig. 2. It does this by estimating how much the sample deviates from a uni-modal/uniform distribution. This deviation value - we refer to it as dip value - lies in the range $(0, 0.25]$. A dip value close two 0 indicates that the sample is uni-modal, while a value close to 0.25 suggests the presence of multiple modes. The dip test itself has a runtime of $\mathcal{O}(n)$, but needs sorted input and thus the total runtime is $\mathcal{O}(n \cdot \log(n))$. Due to the space restrictions, it is not possible to describe the dip test in detail. An introduction can be found in [16].

In our case, we search for the feature with the highest dip value in a multi-dimensional data set. Thus, we measure the dip value for a projection vector v. More specific, we project the data D onto the vector v via the scalar product $\cdot$, i.e, $f = D \cdot v$, to get the feature f in the direction of v. This projection f is now univariate and we can compute its dip value. The dip test is scaling invariant, thus, it is enough to compute the dip value for the vectors in the unit sphere with $||v|| = 1$. The question is: How do we find the projection vector in the unit sphere with the maximal dip value?

2.2 DipExt - Extracting Features with Structure

The dip landscape on the unit sphere is highly complicated for most data sets. For both the running example and the Skinsegmentation data set, we can plot this dip landscape as both data sets are 3-dimensional, making their unit sphere 2-dimensional. The dip landscapes are shown in Fig. 5 as a heatmap of the dip values of the polar coordinates of the unit vectors. These landscapes are clearly

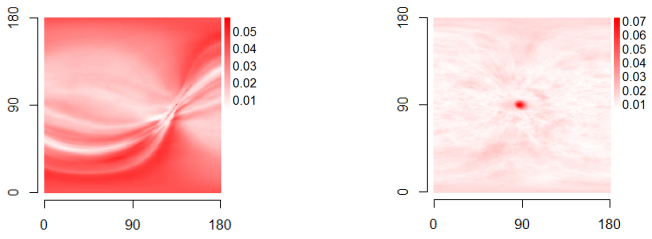


Fig. 5. The dip landscape for the Skinsegmentation data set (left) and the running example (right). The dip values were computed for every vector on the 2D unit sphere.

not trivial, thus, finding the maximal dip value dip_{max} of the projection vector v_{max} is not straightforward.

Krause et al. showed in [11] that the dip is continuous almost everywhere and, more importantly, that a gradient for a projection vector can be computed. In a d-dimensional data set D, the dip value of a changing projection vector $v \in \mathbb{R}^d$ changes smoothly and, thus, the partial derivative of the dip value for the projection vector v is given by:[5]

$$\frac{\partial\ dip(v)}{\partial v_i} = \begin{cases} -\frac{i_3-i_1}{n} \cdot \frac{v\cdot(\beta_i\gamma-\gamma_i\beta)}{(v\cdot\gamma)^2}, & \eta > 0 \\ \frac{i_3-i_1}{n} \cdot \frac{v\cdot(\beta_i\gamma-\gamma_i\beta)}{(v\cdot\gamma)^2}, & \eta \leq 0 \end{cases}, \tag{1}$$

with $\cdot$ the scalar product, $\beta = (\beta_1, \ldots, \beta_d) = x_{i_2} - x_{i_1}$, $\gamma = (\gamma_1, \ldots, \gamma_d) = x_{i_3} - x_{i_1}$. The indices i_1, i_2, i_3 give the *modal triangle* and $x_{i_1}, x_{i_2}, x_{i_3} \in D$ are corresponding to the indices. The height h of the modal triangle fulfils $h = 2 \cdot dip$. The value $\eta = i_2 - i_1 - (i_3 - i_1)(v \cdot \beta)/(v \cdot \gamma)$ merely ensures that the gradient points in the correct direction. Please refer to [11] for a thorough explanation of the technical details.

Since the gradient can be computed, we can make use of **Gradient Descent**-approaches to search for v_{max}. Naive Gradient Descent is unsuited for our needs, as it quickly converges to a local optimum, which the dip landscape is full of, and finding v_{max} becomes unlikely. We need to ensure that our search strategy does not stop too soon and keeps on looking, even if the dip value slightly decreases after a step. This is obtained with a momentum term, which keeps the search from changing direction too fast:

$$\begin{aligned} w_t &= m \cdot w_{t-1} + s \cdot \nabla dip(v) \\ v_{t+1} &= v_t + w_t \end{aligned} \tag{2}$$

The momentum m is set to 0.95 and step-size s to 0.1 for all experiments in this paper. Both of these are common values for these parameters (they are, e.g, used in [24]) and worked well for various data sets. $\nabla dip(v)$ is simply the gradient of the dip, i.e., the direction in which the dip value for the projection vector increases, as computed in Eq. (1). As a starting point, we chose the axis with the highest dip value. This momentum-based search strategy is very capable of finding v_{max}. We also tried other Gradient Descent-Strategies like ADAM, NAG or AMSGrad, but they did not improve the search effectively. The area that should be searched to find v_{max}, of course, increases with the dimensionality of the data set. To ensure that our search strategy keeps up with this increased area, we found that it is useful to start the search not only from the axis with the highest dip value but also from other axes with high dip values. Starting from all axes, however, is unnecessary and only increases runtime. Experiments demonstrated that it is sufficient to start from the $\log(d)$ axes with the highest dip values to ensure that a wider area is searched without being to quickly satisfied with a local optimum.

[5] We follow the argument given in [16] in regard to the explicit form of the derivative.

Table 1. The maximal dip values as found by us compared to overall maximal values.

Data set	SKIN	BANK	IRIS	USER	BRST	FRST	MICE	AIBO	PROX	MOTE	DIAT
Brute	0.048	0.089	0.124	0.070	0.065	0.116	0.146	0.096	0.195	0.067	0.165
DipExt	0.046	0.085	0.124	0.065	0.043	0.107	0.076	0.090	0.192	0.064	0.162

Now, when v_{max} is found, we can extract the projection $f_{max} = D \cdot v_{max}$ of the data onto v_{max} from the data D and continue the search on the orthogonal complement of D with respect to v_{max}. Thus, f_{max} is stored as the projection with the highest dip value, i.e., the feature most important for clustering. The Gradient Descent strategy is repeated on the orthogonal complement of v_{max}, which has now a dimensionality of $d - 1$. After enough features are found (we cover in Section 2.4 what "enough" entails), they are combined to a new condensed data set.

To show that our strategy is very capable in finding these high dip values, we also searched for them via brute force. We computed the highest dip value for various real world data sets and compared them to the highest dip value as found by us. The results can be seen in Table 1. We got extremely close to the optimal values with our strategy on almost all data sets. So, even if other parameter values or Gradient Descent-strategies are chosen, one could not realistically hope to find better dip values. The two data sets, where our strategy was sub-optimal will be covered later on.

2.3 DipScaling - Scaling Features according to their Relevance

DipExt has extracted the features with the highest dip values. As an example, the condensed form of the Banknote-Authentication data set is shown in Fig. 6.

The clusters (the data points are coloured according to the ground truth) are perfectly separated. This is impressive, as it is not an easy-to-cluster-data set (as can be seen in the experiments). A scatter plot, as well as the 2D-representation

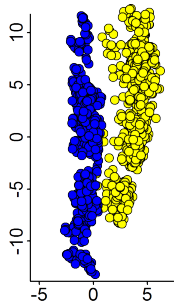


Fig. 6. The 2D extraction of the Banknote-data set.

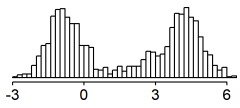


Fig. 7. The first feature.

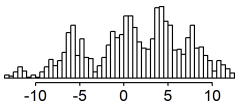


Fig. 8. The second feature.

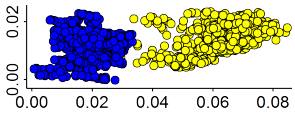


Fig. 9. The extraction after rescaling it with DipScaling.

of the data set with various methods, can be found in the Download, showing the difficulty involved with understanding this data set. Separating the clusters is challenging, but DipExt found a representation, where a correct partitioning is possible. However, even with a perfect initialisation, k-means will not find a good clustering. The axes, i.e., the two features with the highest dip values as found by DipExt (shown as histograms in Fig. 7 and 8) are scaled as they were in the original data set, causing a terrible result for k-means. We can bypass this predicament, by rescaling the axes according to their relevance for clustering. For this, DipScaling [20] is used. The result can be seen in Fig. 9. The data set is now very easy to cluster for k-means.

DipScaling computes the dip values of the axes and rescales the axes with them. Essentially, it executes min-max-normalisation and the new maximum of the axis is its dip value. The effect of this transformation is that the importance of the axes for k-means changes. Axes with very low dip values have a small range, i.e., the values are somewhat similar, thus, they do not influence the computation of the centres in the k-means update step exceedingly. An axis with a high dip value, on the other hand, is scaled rather large causing it to have more influence in determining the centres. This is in line with our assumption of a high dip value signalling a high importance for clustering.

2.4 How many Features?

A question we left open before was when to stop extracting features. After the feature with the maximal dip value dip_{max} has been extracted, DipExt continues on the orthogonal complement of this feature to find the feature with the next highest dip value. It is possible to continue until DipExt has extracted all available features, but this is clearly unsatisfactory. In Fig. 10, the dip values of the extracted features for various data sets with a dimensionality larger than 25 are plotted. The behaviour is roughly the same for all of them. They start with a somewhat high dip value (the absolute value depends on the data set, thus, dip_{max} can differ greatly), a few features with higher dip values might still be found, before the values start to drop to a somewhat constant base level. One possibility would be to apply the Elbow-method (sometimes called knee-method) to find the point where the dip values start to change, but this would mean extracting features only to get their dip value, without using the features later

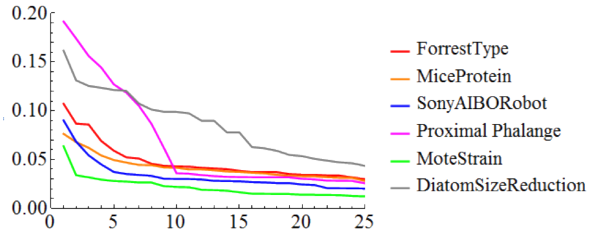


Fig. 10. The DipValues of data sets with dimensionality > 25.

on. This is an overhead, which we are not willing to accept. Instead, a heuristic which we found to work well is to simply search for features until a feature has a dip value smaller than $dip_{max}/2$ and stop after that. Consider the following explanation for this strategy: The features found by DipExt are scaled by DipScaling, thus, a feature with only half the range compared to others will have a small impact on clustering. It will change the position of the cluster centres minimally, if at all. Since its effect is negligible, it might as well be left out. This heuristic manages to keep the dimensionality of the condensed data set small and, furthermore, tailors DipExt to the specific dip landscape of a data set. We discuss it in more detail in Section 3.1.

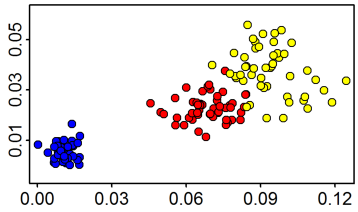
2.5 DipInit - Ensuring the correct Optimum

In the "Banknote-Authentication"-data set it is enough to apply DipScaling to ensure that k-means converges to a good optimum. In general, however, it is not clear to which optimum k-means converges as it is highly sensitive to the initialisation. The main assumption for this paper is, that a high dip value determines the relevance for clustering. Based on this assumption, we create a new initialisation strategy that is tailored to DipExt.

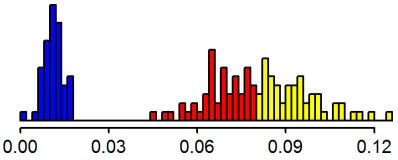
We have seen before how much cluster information can be contained in a single feature and how much a feature with a high dip value can reveal about the correct boundaries of a cluster (compare Fig. 2 to 3). The obvious conclusion is to use this for the initialisation of k-means by clustering the axes with high dip values first. The basic shape of the clustering is thus determined by the feature with the highest dip value, which "knows" the most about the boundaries of the clusters. Features with a smaller dip value are brought in afterwards when the basic shape is fixed.

We show the effect of this initialisation on the DipExt-version of the Iris-data set (shown in Fig. 11a). Since DipExt reduced the dimensionality to 2, we can plot it directly. Fig. 11b shows a histogram of the feature with the highest dip value. This one-dimensional feature is now clustered first. It is sorted, split into equally large parts all containing $\frac{n}{k}$ many data points (equal-frequency binning) and then k-means is executed. To this 1D-data set, the axis with the second-highest dip value is now added. We now have a 2D-data set with the labels from the clustering of the 1D-data set (Fig. 11c). K-means resumes and we get the result as shown in Fig. 11d. If there were more axes, they would be iteratively added and, each time, k-means would resume.

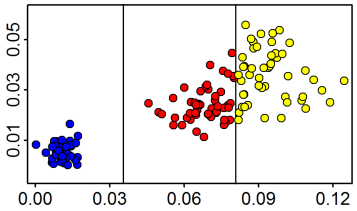
The basic shape of the clustering is fixed, when the first axis is clustered (Fig. 11c). Adding the second axis leads to a few changes at the border of the red and yellow cluster, as the Voronoi cells of k-means slightly change. The borders of the Voronoi cells are included in Fig. 11c and Fig. 11d. The final result gets us very close to the ground truth shown in Fig. 11a. As we wanted, the features with the highest dip value determine the principal form of the clustering, while less structured features, which are added to the data consecutively, have a smaller influence on the clustering and mainly improve on details. An advantage of DipInit is that it is **deterministic**. Contrary to most initialisation strategies



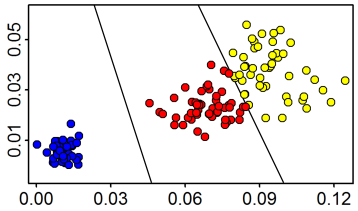
(a) The 2D reduction of the Iris data set. Colours correspond to the ground truth.



(b) The feature with the highest dip value is split in k parts with equally many data points, and then k-means is executed.



(c) Colours show the results if only the first axis as shown in (b) is clustered.



(d) Running k-means on (c) with the preliminary clusters leads to the final clustering.

Fig. 11. The DipInit initialisation strategy on the reduced Iris-data set.

for k-means, which contain some random elements (see [2] for an overview), every step of DipInit is performed without any decision left to chance (Pseudo code for DipInit can be found in the Download). A further advantage of this initialisation is that it makes full use of the characteristics of DipExt. DipInit needs the features with the highest dip values, which DipExt extracted. It can be used as a stand-alone initialisation for k-means, where it is more than competitive (see Download for details), but it truly shines in combination with DipExt.

3 Experiments

Sourcecode: The source code, datasets, labels, parameters for our and all compared methods can be found under the following links:
`https://doi.org/10.6084/m9.figshare.12063252.v1`
`https://dm.cs.univie.ac.at/research/downloads`

With DipExt we extract the subset of the features with the highest dip values. DipScaling rescales these features into a more fitting range. DipInit ensures that k-means converges to an optimum, which makes sense from the point of our main assumption. The question is how well this approach fares in comparison to other methods. To answer this, we compared our approach on 11 data sets to a wide range of algorithms. We chose the compared methods from cluster-aware subspace clustering (SubKmeans, FossClu, DEC, IDEC), transformation approaches

Table 2. Experimental results given in Adjusted Mutual Information (AMI) [23]. DipExt in combination with DipInit. For random methods the average of 100 runs is shown. Correct k is always given. Cases which were aborted after 1h or failed due to non-trivial implementation bugs are marked with —. Best result bold.

Data set	SKIN	BANK	IRIS	USER	BRST	FRST	MICE	AIBO	PROX	MOTE	DIAT
Orig. dim.	3	4	4	5	10	27	70	77	80	84	345
Red. dim.	2	2	2	4	9	5	7	5	8	2	14
DipExt	**0.48**	**0.81**	**0.88**	**0.61**	**0.50**	**0.83**	**0.54**	**0.60**	**0.49**	**0.45**	**0.80**
k-means	0.02	0.03	0.68	0.27	0.27	0.68	0.29	0.35	0.45	0.30	0.76
SubKMeans	0.01	0.01	0.65	0.21	0.41	0.49	0.34	0.00	**0.49**	0.40	0.76
FossClu	0.27	0.44	0.74	0.49	0.29	0.55	0.26	0.35	—	—	—
DEC	0.11	0.08	0.61	0.20	0.34	0.54	0.22	0.25	0.42	0.12	0.44
IDEC	0.01	0.11	0.58	0.22	0.33	0.50	0.22	0.20	0.34	0.12	0.58
DipTrans.	0.39	0.68	0.83	0.50	0.46	0.68	0.46	0.16	0.48	0.26	0.67
EWKM	0.03	0.08	0.78	0.24	0.28	0.08	0.30	0.01	0.45	0.12	0.70
Z-trans.	0.01	0.01	0.63	0.20	0.44	0.51	0.33	0.23	0.47	0.35	0.74
min-max n.	0.02	0.02	0.69	0.27	0.45	0.70	0.35	0.35	0.48	0.35	0.71
SkinnyDip	—	0.34	0.55	0.29	0.24	0.48	0.19	—	0.45	0.00	—
DipMeans	—	0.25	0.55	0.00	0.00	0.00	0.00	0.00	0.45	0.00	0.75
PCA	0.01	0.01	0.63	0.21	0.44	0.50	0.32	0.21	0.46	0.35	—
ICA	0.24	0.07	0.55	0.16	0.37	0.63	0.48	0.32	0.43	0.37	0.79
t-SNE	—	0.73	0.34	0.05	0.01	0.00	0.04	0.00	0.00	0.24	0.05
UMAP	0.00	0.40	0.75	0.37	0.29	0.75	0.49	0.13	0.46	0.37	0.78
DBSCAN	—	0.20	0.30	0.01	0.27	0.06	0.17	0.00	0.00	0.08	0.25
EM	0.19	0.00	0.84	0.45	0.27	0.71	0.29	0.34	0.45	0.06	0.75
Spec. Clust.	—	0.03	0.59	0.28	0.41	0.74	0.43	0.04	0.47	0.00	0.71
SingleLink	—	0.12	0.70	0.26	0.14	0.41	0.18	0.04	0.03	0.03	0.02
k-means++	0.02	0.03	0.69	0.27	0.16	0.66	0.30	0.39	0.45	0.30	0.74

(DipTransformation, EWKM, Z-transformation, min-max-normalisation), dip-based methods (SkinnyDip, DipMeans) and general dimensionality reduction methods (PCA, ICA, t-SNE, UMAP) and standard methods (DBSCAN, EM, Spectral Clustering, SingleLink, k-means++). We compare to a wide range of methods, to show that the results we obtain are highly competitive. This is also the reason why standard methods like DBSCAN are included. Most of the approaches (DEC, SubKmeans, t-SNE, ...) are combined with k-means, but for some data sets, k-means might simply be the wrong framework and, e.g., DBSCAN a far better choice. To show that our results are competitive, these very different methods are also included.

The data sets - SkinSegmentation (SKIN), Banknote-Authentication (BANK), Iris (IRIS), Userknowledge (USER), BreastTissue (BRST), Forresttype (FRST), Mice Protein (MICE), SonyAIBORobot (AIBO), ProximalPhalanxOutlineAgeGroup (PROX), MoteStrain (MOTE) and DiatomSizeReduction (DIAT) - are all publicly available. They range greatly in size (BRST n=106, SKIN n=245057) and dimensionality (SKIN d=3, DIAT d=345), which shows that our approach is not per se restricted in these regards. Links to the data sets can be found with the Source Code. The results of these experiments can be seen in Table 2.

As an evaluation metric to measure the quality of clustering we used Adjusted Mutual Information (AMI) [23] with the max-normalisation. AMI ranges from 1.0 (perfect clustering result) to 0.0 (purely random cluster assignments). Please keep in mind, that different normalisations can lead to (slightly) different results. Apart from AMI, Normalized Mutual Information (NMI) is also frequently used to evaluate clustering results. However, using NMI instead of AMI has little influence. The values are very similar, with DipExt again being the best method on all data sets.

The results in Table 2 paint a very clear picture. Our approach is the method with the best results on all data sets. It is particularly noteworthy for us that we succeeded in improving the quality of clustering for k-means by more than 0.26 in AMI on average. BANK, as an example, was improved from 0.03 to 0.81 in AMI. We managed to extract a subspace where clusters can be far easier found. The condensed data set makes the information relevant for clustering more easily accessible than SubKMeans, PCA, DipTransformation, ... were able to do so. Furthermore, the condensed version of the data set has now a considerably smaller dimensionality: MOTE has now 2 instead of 84 dimensions, DIAT 14 instead of 345.

Parameters: Due to the large number of compared methods, listing all parameters would be rather expansive. They can instead be found in the Download.

As we have seen, our approach reaches very impressive results. Now, there are a few topics which we wish to discuss in regard to the stability of our dimensionality-criterion, runtime and the applicability of DipExt to methods besides k-means.

3.1 How much Structure is enough Structure?

As stated, we extract features until one is found with a dip value smaller than $dip_{max}/2$. We changed this threshold to $dip_{max}/3$ and $dip_{max}/1.5$ to see what the effect of this threshold is and found that the results were barely influenced by it (see Table 3). Only three data sets had a change ≥ 0.02 in AMI, and they would essentially still be among the best clustering results. We can see from Fig. 12 that the main difference in these thresholds was the number of features DipExt extracts. Here comes another advantage of DipScaling into effect, which shows how well it fits DipExt and DipInit. DipScaling rescales the axes depending on

Table 3. Comparing stopping-criterion $dip_{max}/2$ to /3 and /1.5.

	SKIN	BANK	IRIS	USER	BRST	FRST	MICE	AIBO	PROX	MOTE	DIAT
/1.5	0.48	0.81	0.88	0.61	0.54	0.83	0.46	0.54	0.48	0.45	0.81
/2	0.48	0.81	0.88	0.61	0.50	0.83	0.54	0.60	0.49	0.45	0.80
/3	0.48	0.81	0.86	0.61	0.50	0.81	0.54	0.60	0.49	0.45	0.79

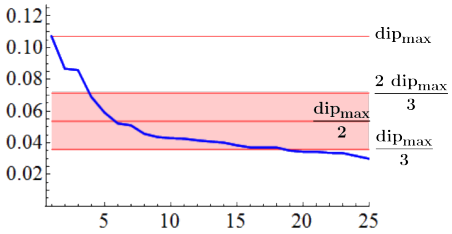


Fig. 12. The dip values for the 25 features with maximal dip values for FRST.

their dip values, thus, features with low dip values have a smaller impact on clustering, making DipExt more stable in regards to the number of features extracted. Furthermore, DipInit ensures that the features with high dip values are clustered first, fixing the principal shape of clustering, such that features with low dip values have a smaller impact on clustering. The combination of these effects makes our approach extremely stable in regard to the number of features extracted. It makes barely a difference if there are 5 or 15 features extracted by DipExt. The result will be almost the same.

3.2 Runtime

A more detailed calculation of the runtime as well as the pseudocode can be found in the Download. The results show an $\mathcal{O}(n \cdot \log(n))$-dependency on the number of data points n, as well as an $\mathcal{O}(d \cdot \log(d))$-dependency on the dimensionality of the data set d for our approach. This behaviour can also be seen in Fig. 13, showing that these estimates are correct. We also find that DipExt and DipInit are competitive to many state-of-the-art methods, even though the code is currently not optimised in regard to runtime. Please see the Download for details.

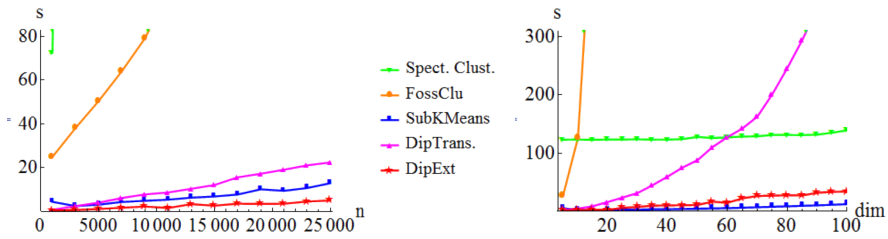


Fig. 13. Runtime relative to the dimensionality of the data set d (right). Dimensionality ranging from 5 to 100 with a data set size of $\approx$ 1.500 data points. Runtime relative to the data set size n (left). Data size ranging from 1.000 to 25.000 data points with a constant dimensionality of 5.

Table 4. Methods besides k-means on the original data (-) and in combination with DipExt (+).

Data set	SKIN	BANK	IRIS	USER	BRST	FRST	MICE	AIBO	PROX	MOTE	DIAT
DBSCAN	—	0.20	0.30	0.01	0.27	0.06	0.17	0.00	0.00	0.08	0.25
DBSCAN+	—	**0.78**	**0.55**	**0.10**	**0.43**	**0.40**	**0.22**	**0.15**	**0.45**	**0.28**	**0.76**
EM	0.19	0.00	0.84	0.45	0.27	0.71	0.29	0.34	**0.45**	0.06	0.75
EM+	**0.50**	**0.79**	**0.85**	**0.52**	**0.47**	**0.82**	**0.51**	**0.64**	**0.45**	**0.34**	**0.76**
Spec. Clust.	—	0.03	0.59	0.28	0.41	0.74	0.43	0.04	0.47	0.00	**0.71**
Spec. Clust.+	—	**0.89**	**0.68**	**0.53**	**0.52**	**0.81**	**0.52**	**0.56**	**0.48**	**0.12**	0.70
SingleLink	—	0.12	0.70	0.26	0.14	0.41	0.18	0.04	0.03	0.03	**0.02**
SingleLink+	—	**0.90**	**0.78**	**0.49**	**0.37**	**0.48**	**0.35**	**0.09**	**0.48**	**0.39**	0.01

3.3 Methods besides k-means

DipExt has been created with k-means in mind. However, there is no reason why it should not also be used in combination with other methods. To test whether it is compatible, we applied 4 of the most often used clustering algorithms to the subset of features extracted by DipExt. The results (shown in Table 4) are clear. All methods profit massively from DipExt: 38 of 41 results improved showing an average increase in clustering quality of 0.23 in AMI. We want to emphasise that some of these results (e.g. Spectral Clustering on BANK with 0.89 in AMI) are now better than all other listed in Table 2. This shows that DipExt is not bound to k-means. Furthermore, it emphasises our claim that DipExt creates a condensed version of the data set containing all the information relevant for clustering. It removes distracting features containing mostly noise information which are detrimental to clustering, i.e., features with uni-modal distributions. The extracted features contain clearer borders between clusters making the separation of them easier. Thus, for a wide range of methods, the subset of features found by DipExt is easier to cluster than the original data set.

4 Discussion and Conclusion

DipExt and DipInit improved clustering for k-means by more than 0.26 in AMI on average. Compared to the original data sets, k-means in combination with DipInit can cluster the subspace found by DipExt far better.

For two data sets, however – MICE and BRST – DipExt could not find the maximal dip values of the data sets compared to brute force. The result of brute force improved MICE from 0.55 to 0.62 in AMI and did not affect BRST. Using brute force on the other data sets improved some of their clustering results as well, if only by a small value. These observations are further indications of the strong connection between dip value and success of clustering. The higher the dip values are, the better the clustering results will be. The clusters can now be better separated, corroborating our assumption. However, the improvement of brute force compared to DipExt is only very small, as DipExt is already highly capable of finding the features with the maximal dip values as we have seen in

Section 2.2/Table 1. At best, only a slightly better dip value can be found. This shows that DipExt is very close to the optimal subspace, which can be extracted in our framework on these data sets.

DipExt can be considered as a "cluster-friendly subspace clustering"-method. Many of these methods, though, focus only on one method, mostly k-means, and try to find a subspace suited for it. While DipExt is also highly compatible with k-means, especially with the DipInit initialisation, it is not limited to it, contrary to, e.g., SubKMeans. DipExt extracts features and transforms them into a low-dimensional representation of the data, which is far easier to cluster for a wide range of methods. We demonstrated this for various standard methods in section 3.3/Table 4, where we showed that their results could be greatly improved. These results are now so good that they often outperform other "cluster-friendly subspace clustering"-methods. Considering the difference in the approach of the standard methods, we conclude that this improvement also applies to a wide range of other clustering approaches. This shows that DipExt creates a "clustering-friendly subspace" that is not limited to one method.

There are, of course, limitations to our method. Multimodal or interlocking clusters pose a problem. It is also possible to create pathological cases where the clusters are positioned such that no univariate projection reveals anything relevant about the cluster structure. We have further found that very small datasets (< 100 data points) are not suitable for DipExt. It seems that this is not enough data points to make the dip test statistically significant. For many data sets, though, DipExt and DipInit are interesting tools for clustering. DipInit is one of the few deterministic initialisation strategies for k-means. K-means is probably the most often used clustering algorithm, widely applied in all areas of data-driven research. However, since the results of k-means depend on the initialisation, one either runs it once and might have a mediocre result, or runs it multiple times and has to choose one of them. With DipInit, k-means finds one optimum which is in accord with our assumptions and which most likely, following our experiments, is well above average. DipExt is very useful for exploratory data analysis. It extracts only those features where the cluster boundaries are well-defined, creating a condensed data set which is far easier to understand (see IRIS in Fig. 11a or BANK in Fig. 9). This transformation can now be far better clustered, not only by k-means but by various other methods as well. DipExt may not be limited to clustering. As DipExt allows for a better separation of clusters, combining it with methods for estimating the number of clusters is a possible application. One could also consider the combination of DipExt with classification-methods. We intend to explore possible applications for DipExt and potential improvements further in the future.

References

1. Arthur, D., Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA, 2007.
2. Celebi, M., Kingravi, H., Vela, P., *A Comparative Study of Efficient initialisation Methods for the K-Means Clustering Algorithm*, Expert Syst. Appl., 2013.

3. Chronis, P., Athanasiou, S., Skiadopoulos, S., *Automatic Clustering by Detecting Significant Density Dips in Multiple Dimensions*, ICDM, 2019.
4. Dempster, A. P., Laird, N. M., Rubin, D. B., *Maximum-Likelihood from incomplete data via the EM algorithm*, Journal of the Royal Statistical Society, 1977.
5. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *A density-based algorithm for discovering clusters in large spatial databases with noise*, KDD, 1996.
6. Goebl, S., He, X., Plant, C., Böhm, C., *Finding the Optimal Subspace for Clustering*, ICDM, 2014.
7. Guo, X., Gao, L., Liu, X., Yin, J., *Improved Deep Embedded Clustering with Local Structure Preservation*, IJCAI, 2017.
8. Hartigan, J. A., Hartigan, P. M., *The Dip Test of Unimodality*, The Annals of Statistics, 1985.
9. Jing, L., Ng M.K., Huang,J. Z., *An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data*, TKDE, 2007.
10. Kalogeratos, A.,Likas, A., *Dip-means: an incremental clustering method for estimating the number of clusters*, NIPS, 2012.
11. Krause, A., Liebscher, V., *Multimodal projection pursuit using the dip statistic*, Preprint-Reihe Mathematik, 2005.
12. Kriegel, H. P., Kröger, P., Zimek, A., *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering*, TKDD, 2009.
13. Maaten, L., Hinton, G., *Visualizing data using t-SNE*. Journal of machine learning research, 2008.
14. MacQueen, J. B., *Some methods for classification and analysis of multivariate observations*, Berkeley Symposium on Math. Stat. and Prob., 1967.
15. McInnes, L., Healy, J., Melville, J., *Uniform manifold approximation and projection for dimension reduction*, arXiv:1802.03426, 2018.
16. Maurus, S., Plant, C., *Skinny-dip: Clustering in a Sea of Noise*, KDD, 2016.
17. Mautz, D., Ye, W., Plant, C., Böhm, C., *Towards an Optimal Subspace for K-means*, KDD, 2017.
18. Ng, A., Jordan, M., Weiss, Y., *On spectral clustering: Analysis and an algorithm*, NIPS, 2002.
19. Schelling, B., Plant, C., *DipTransformation: Enhancing the Structure of a Dataset and thereby improving Clustering*, ICDM, 2018.
20. Schelling, B., Plant, C., *Dataset-Transformation: improving clustering by enhancing the structure with DipScaling and DipTransformation*, KAIS, 2019.
21. Sibson, R., *SLINK: an optimally efficient algorithm for the single-link cluster method*, The Computer Journal, 1973.
22. Siffer, A., Fouque, P. A., Termier, A., Largouet, C., *Are your data gathered?*, KDD, 2018.
23. Vinh, N. X., Bailey, J., *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance*, JMLR, 2011.
24. Wu, H., Gu, X., *Max-Pooling Dropout for Regularization of Convolutional Neural Networks*, ICONIP, 2015.
25. Xie, J., Girshick, R., Farhadi, A., *Unsupervised deep embedding for clustering analysis*, ICML, 2016.
26. Yang, B., Fu, X., Sidiropoulos, N., *Learning From Hidden Traits: Joint Factor Analysis and Latent Clustering*, IEEE Trans. Signal Process, 2017.
27. Yang, B., Fu, X., Sidiropoulos, N., Hong, M., *Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering*, ICML, 2017.

Supplement to 'Utilizing Structure-rich Features to improve Clustering'

Benjamin Schelling[1,2,3], Lena Greta Marie Bauer[4], Sahar Behzadi[3], and Claudia Plant[3,4]

[1] MCML, Munich, Germany
[2] Ludwig-Maximilians-Universität München, Munich, Germany
[3] Faculty of Computer Science, University of Vienna, Vienna, Austria
[4] ds:UniVie, Vienna, Austria
firstname.lastname@univie.ac.at

1 The Banknote-Authentication data set

The Banknote-Authentication-data set (BANK-data set) is quite difficult to cluster. K-means, for example, will produce a clustering result which is essentially not better than a random assignment of data points. To show how capable DipExt is at extracting the structure relevant for clustering, we compared the 2D-reduction of DipExt with all other methods from Table 1 in the main paper, which are capable of creating a 2D-reduction of a data set. Some methods such as FossClu, reduce dimensionality, but one cannot choose the final dimensionality. Thus, we could not include them in the plots.

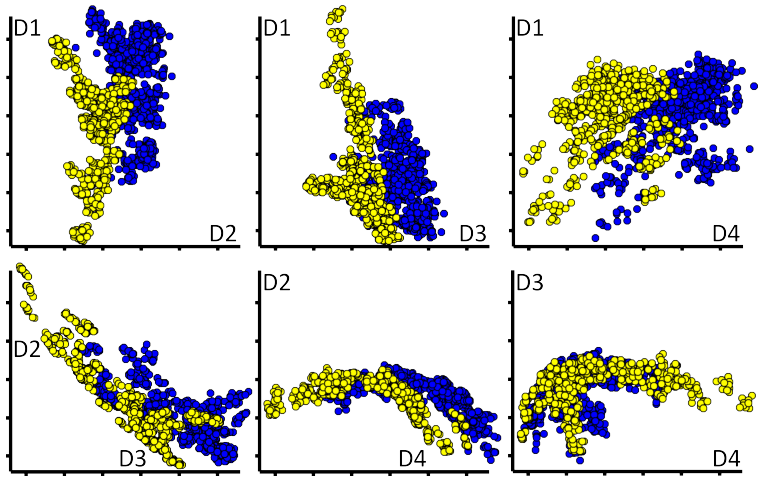


Fig. 1. Scatterplot of the Banknote-Authentication-data set

We chose the BANK-data set, as it is 4-dimensional, which allows us to show all pairwise feature scatter plots with a reasonable amount of plots. They can

be seen in Fig. 1. The clusters are heavily overlapping in each plot and a good separation seems almost impossible. It is no surprise that k-means manages to cluster the data no better than random assignment. Nevertheless, there is structure important for clustering in this data set, one simply needs to find it.

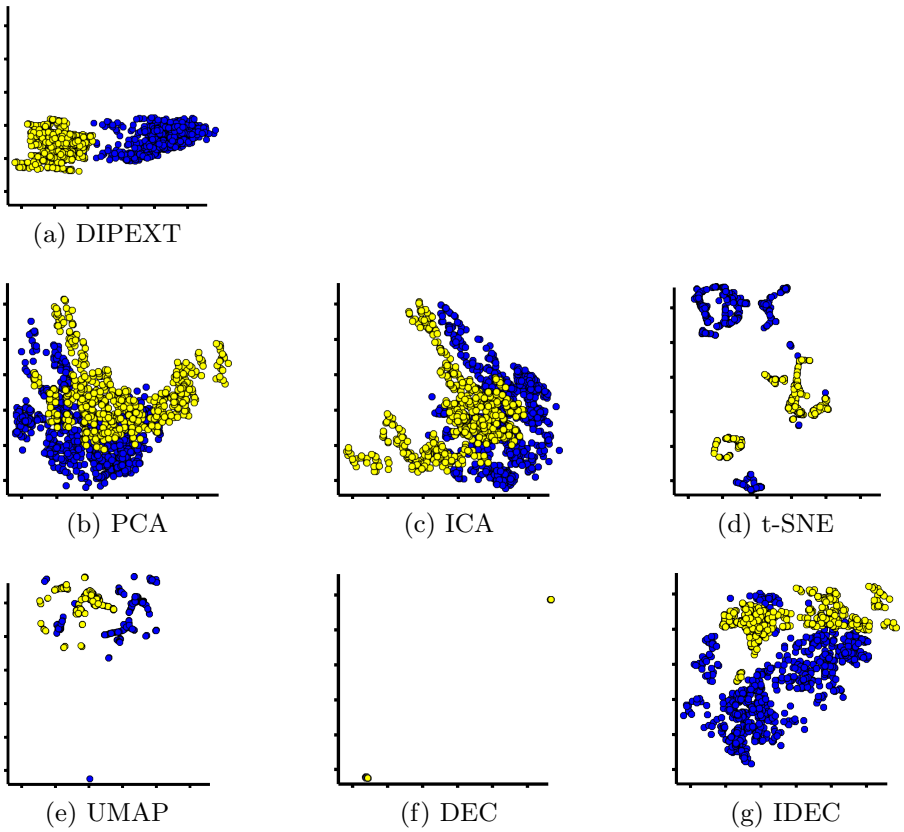


Fig. 2. The results of various algorithms to reduce the Banknote-Authentication-data set to 2 dimensions. Colours correspond to the ground truth.

In Fig. 2 the various 2D-representations of the BANK-data set produced by various methods can be seen. DipExt (Fig. 2a) creates an almost perfect separation of the clusters. Thus we can see that it is possible to create a transformation of the data, which is easy to cluster. The other methods, however, are less successful with that. PCA and ICA (Fig. 2b and 2c) cannot separate the clusters, but at least their representations are somewhat similar to the data. t-SNE and UMAP (Fig. 2d and 2e) create 2D-representations, where the original shape of the data can only be guessed. This is often the disadvantage of non-linear transformations. The data is sometimes so heavily distorted that the origin of the data seems almost unrelated to the transformed data. t-SNE and UMAP, furthermore, reconstruct the data dependent on probability, which means that the

transformation is not even continuous. This can create various rips in the data, destroying the neighbourhood of data points. The Deep Learning approaches DEC and IDEC are continuous, but as we see for DEC (Fig. 2f), this alone is not always enough. DEC is especially brutal and projects the data onto only two single points. IDEC (Fig. 2g) is less extreme, but its representation is also rather distorted and clustering is difficult. Non-linear methods can, in theory, transform more types of data sets into a form easier to cluster, but, as we see, the risk of producing a completely useless transformation shape is also far higher. DipExt is here the only algorithm which manages to represent the data, such that it resembles the original data and can be easily clustered.

Algorithm 1 DipInit

Require: Data D, Number of Clusters k
procedure DipInit(D, k)
    Compute dip values of axes
    Sort axes in regard to dip values
    Sort first axis
    Split the first axis into k parts with equally many data points
    Apply k-means to the first axis with the k parts as initialisation
    for j = 2,...,dim **do**
        Add j-th axis to data points
        Resume k-means on the data
    end for
    return label as found by k-means
end procedure

2 DipInit

In the main paper, we did not have the space available to look into the performance of DipInit as an initialisation method for k-means in more detail. We will do this now. To this end, we compared DipInit with a few of the more common initialisation strategies for k-means. We included Forgy [3] and Hartigan-Wong [5], as two of the better-known strategies, but most often users usually apply either of the two following strategies: random initialisation or k-means++ [1]. See [2] for a survey on this topic.

Initialisation strategies for k-means are usually evaluated in two regards: the objective function of k-means - Sum of Squared Error (SSE) - and clustering quality. For clustering quality, we again used AMI as a measurement tool. The overview of our experiments can be seen in Table 1. We opted here to show only the average results of all 11 data sets, as it would otherwise become an extremely expansive table. For SSE the value of our method DipInit is taken as 100% and the other methods are evaluated on how much larger/smaller their result is.

The results paint a very clear picture. DipInit is on these data sets - before and after applying DipExt - a very capable choice, outperforming the other

Table 1. Comparing various initialisation strategies for k-means in regard to clustering quality (AMI) and the objective funtion of k-means (SSE) on the original data sets, as well as after applying PCA and DipExt respectively. The average for all data sets from the main paper is given. Best result in bold.

	DipExt		Original Data		PCA	
	AMI	SSE	AMI	SSE	AMI	SSE
DipInit	**0.639**	**100.0%**	**0.408**	**100.0%**	**0.362**	**100.0%**
Rand. Init.	0.594	120.7%	0.372	106.4%	0.314	106.6%
k-means++	0.603	118.4%	0.364	106.5%	0.305	103.2%
Forgy	0.590	120.6%	0.366	108.6%	0.314	106.4%
Hartigan-Wong	0.599	122.1%	0.382	104.8%	0.316	105.0%

methods according to the objective function of k-means, SSE, as well as clustering quality in AMI. The same can also be said if PCA is applied to the data sets; DipInit again improves over the other initialisation methods. The advantage, however, is most prominent after DipExt has been applied to the data sets. On average, DipInit finds here a 20% better optimum in SSE and yields a 0.04 higher AMI score. To put this into perspective, on the DipExt-version of the data sets, all of the random-based methods have a standard deviation of SSE of roughly 10% on average. Thus, DipInit finds optima better than two times the standard deviation. This again shows the high compatibility of DipInit and DipExt.

Randomisation based initialisation strategies can be arbitrarily often restarted which can lead to finding different optima for k-means. Thus, we tested the effects of restarting these strategies and how that influenced the clustering quality. We chose to restart them 100 times. We found this decision to be justified, as all these randomisation based strategies - Random Initialisation, k-means++, Forgy, Hartigan-Wong - gave on average an extremely similar value (difference < 0.005) for clustering quality. It seems that these are enough restarts in order to see that they find essentially the same results. Comparing the average on these datasets if every method had 100 restarts, to DipExt on the original data sets showed that there was barely a difference - 0.411 vs 0.408 - meaning that DipInit finds results of the same quality as these other methods if they are rerun 100 times. Interestingly, however, this is not the case after PCA and DipExt have been applied to the data sets. Here, DipInit is **better** than the other strategies. With PCA it has a lead of 0.362 to 0.345 and with DipExt of 0.639 to 0.615. All of this shows that DipInit is a very capable initialisation strategy for k-means, which should be considered when deciding on the "correct" initialisation.

There is a second aspect, which can be extracted from Table 1. One can estimate the importance of DipExt compared to DipInit in improving the average clustering quality. DipInit gives an improvement over the other k-means initialisations of roughly 0.04 in AMI. The effect of DipExt, however, is far more significant. It improves AMI from below 0.40 to over 0.60. Thus, it has an effect of boosting AMI by more than 0.20, which is an extreme increase in clustering

quality. DipInit chooses one specific optimum for k-means, but it stays in the framework of k-means, i.e. it chooses one of the optima k-means could converge to. DipExt, on the other hand, changes the position of the data points, which changes the optima of k-means themselves. It allows to find clusterings which were not stable with k-means before. Thus, it has of course a higher influence in regard to improving clustering.

3 Pseudo Code

Pseudocode for DipExt (Algorithm 2) as well as DipInit (Algorithm 1) can be found here. Pseudocode often allows for additional insight into the workings of an algorithm, as it might clear up possible unanswered questions. It is also useful for estimating runtime.

Algorithm 2 DipExt

Require: Data D
procedure DipExt(D)
  $dip_{max} \leftarrow 0$
  $D_{new} \leftarrow 0$
  for i = 1,...,dim **do**
    Find log(dim) axes $a_1, \ldots, a_{\log(dim)}$ with the highest dip values.
    $f_i \leftarrow 0$
    $v \leftarrow 0$
    for j = $a_1, \ldots, a_{\log(dim)}$ **do**
      while Gradient Descent **do**
        Apply Gradient Descent to a_j until convergence.
        $f_j \leftarrow$ projection found by Gradient Descent with maximal dip value
        $v_j \leftarrow$ the projection vector of projection f_j
      end while
      if dip(f_j)>dip(f_i) **then**
        $f_i \leftarrow f_j$
        $v \leftarrow v_j$
      end if
    end for
    $dip_{max} \leftarrow Max(dip_{max}, dip(f_i))$
    Add f_i to D_{new}
    if $dip(f_i) < dip_{max}/2$ **then**
      stop for-loop
    end if
    Compute orthogonal complement $v^{\perp}$ of v
    $D \leftarrow D \cdot v^{\perp}$
  end for
  $D_{new} \leftarrow$ Apply DipScaling to D_{new}
  return D_{new}
end procedure

Estimating the runtime of DipExt is rather straightforward. Algorithm 2 shows that we have two for-loops, one over all dimensions d and the second over $\log(d)$ many elements. This gives as a $\mathcal{O}(d \cdot \log(d))$-factor for the dependency on the dimensionality d. Inside the loop merely the Gradient Descent is of relevance for the runtime. Following the arguments in [6] the gradient has an estimation of $\mathcal{O}(n)$, but due to the need for sorted input the total runtime in respect to the number of data points n is $\mathcal{O}(n \cdot \log(n))$. The Gradient Descent seems to be constant in regards to the number of steps it has to take until convergence. Thus, in total, we find DipExt to have a runtime of

$$\mathcal{O}(n \cdot d \cdot \log(n) \cdot \log(d)) \tag{1}$$

As stated in the main paper, the runtime experiments corroborate these results.

The source code is currently not optimised regarding runtime. There are various possibilities, the most obvious one, parallelizing the Gradient Descent searches. We start from $\log(d)$ many different axes and run completely independent calculations. These could all be outsourced onto different processors.

4 Experiments

As stated, we present here in Table 2 the clustering results measured in NMI [8] instead of AMI. The results are, in the end, very similar. The difference between AMI and NMI is, that AMI corrects for chance. It takes into account that the size of clusters and the number of clusters can differ and includes that in the computation of its score. Many of these methods, however, are used in combination with k-means or use k-means as part of their algorithm. Thus, their clusters are mostly the same size and the number of clusters is the same. AMI, therefore, does not need to correct much and the results are comparable to NMI. We used the max-normalisation for AMI.

5 Parameters

In regard to parameters, we tried to be as fair as possible to comparison methods. Non-deterministic methods like k-means with random initialisation have been iterated 100 times (except the Deep Learning methods DEC/IDEC) and the average AMI/NMI-value is given. To all methods that needed the number of clusters, like k-means and EM, the correct k was given. For DBSCAN the average pairwise distance between data points a was computed and every combination out of $minPts \in \{1, 2, 5, 10, 25, 50, 100\}$ and $\epsilon \in \{0.01 \cdot a, 0.1 \cdot a, a\}$ was tested. Only the best AMI/NMI-value is reported. For PCA and ICA we followed the lead given in [7]. For PCA this meant keeping 90% of the variance. For ICA the reduced dimensionality is set to k, the number of clusters. EWKM has a parameter λ which should be chosen in the range $[1, 3]$. For each run, we tried

Table 2. Experimental results given in Normalized Mutual Information [8]. DipExt in combination with DipScaling and DipInit. For random methods the average of 100 runs is shown. Correct k is always given. Cases which were aborted after 1h or failed due to non-trivial implementation bugs are marked with —. Best result bold.

Data set	SKIN	BANK	IRIS	USER	BRST	FRST	MICE	AIBO	PROX	MOTE	DIAT
Orig. dim.	3	4	4	5	10	27	70	77	80	84	345
Red. dim.	2	2	2	4	9	5	7	5	8	2	14
DipExt	**0.48**	**0.81**	**0.88**	**0.62**	**0.54**	**0.83**	**0.55**	**0.60**	**0.49**	**0.45**	**0.80**
k-means	0.02	0.03	0.68	0.28	0.32	0.68	0.30	0.35	0.45	0.30	0.76
SubKMeans	0.01	0.01	0.66	0.22	0.45	0.50	0.35	0.00	**0.49**	0.41	0.77
FossClu	0.27	0.44	0.75	0.50	0.32	0.56	0.27	0.35	—	—	—
DEC	0.11	0.08	0.61	0.21	0.38	0.54	0.24	0.25	0.42	0.12	0.45
IDEC	0.01	0.11	0.58	0.23	0.37	0.51	0.23	0.20	0.34	0.12	0.59
DipTrans.	0.39	0.68	0.83	0.51	0.50	0.68	0.47	0.16	**0.49**	0.26	0.67
EWKM	0.03	0.08	0.78	0.25	0.33	0.10	0.31	0.01	0.45	0.12	0.70
Z-trans.	0.01	0.01	0.64	0.21	0.48	0.51	0.34	0.23	0.47	0.35	0.74
min-max n.	0.02	0.02	0.69	0.28	0.49	0.70	0.37	0.35	**0.49**	0.36	0.72
SkinnyDip	—	0.34	0.55	0.30	0.26	0.49	0.19	—	0.45	0.00	—
DipMeans	—	0.25	0.55	0.00	0.00	0.00	0.00	0.00	0.45	0.00	0.75
PCA	0.01	0.01	0.63	0.22	0.48	0.51	0.34	0.21	0.46	0.35	—
ICA	0.24	0.07	0.56	0.17	0.41	0.63	0.49	0.32	0.43	0.37	0.79
t-SNE	—	0.73	0.35	0.06	0.08	0.02	0.06	0.00	0.00	0.24	0.05
UMAP	0.00	0.40	0.75	0.38	0.34	0.75	0.50	0.13	0.46	0.37	0.78
DBSCAN	—	0.20	0.31	0.23	0.40	0.26	0.35	0.11	0.16	0.10	0.30
EM	0.19	0.01	0.84	0.45	0.32	0.71	0.31	0.34	0.45	0.06	0.76
Spec. Clust.	—	0.03	0.59	0.29	0.45	0.74	0.44	0.04	0.47	0.00	0.72
SingleLink	—	0.12	0.70	0.27	0.20	0.42	0.20	0.04	0.03	0.03	0.03
k-means++	0.02	0.03	0.69	0.28	0.22	0.67	0.31	0.39	0.45	0.30	0.74

$\lambda \in \{1, 2, 3\}$ and took the best result. For DipTransformation we left the rotation speed at $c = 5$, as proposed in the paper. t-SNE and UMAP have a wide range of parameters, which are difficult to set optimally, and they are furthermore not deterministic. We left the t-SNE parameters at their default values as set in the R-implementation of the "tsne"-package, which implies, amongst others, a perplexity value of 30. To balance the randomness, we calculated t-SNE 10 times and ran k-means 10 times on each of the transformations and reported the average of the 100 runs. UMAP has an impressive range of parameters on can set, from learning rate to number of epochs used. We left most the parameters at their default values of the "UMAP"-package of R, but set the initialisation to k-means, as most other methods here also make use of it and set the number of neighbours to 10. Neural Network-based method are difficult to parametrise, but in the end, we followed the guidelines given in [9]. We pretrained a feed forward autoencoder with layers $d - 500 - 500 - 2000 - h - 2000 - 500 - 500 - d$, where d is the number of input dimensions of the data set and h is the hidden layer dimension in which the clustering is performed. h was set to $\min\{d, k\}$, where k is the number of clusters. The training was done with the Adam optimiser with default momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For pretraining

we trained for 50.000 mini-batch iterations with a learning rate of $1e-4$. For the joint clustering we trained for another 50.000 mini-batch iterations, where the learning rate was reduced every 10.000 iterations by 10 as proposed in [9].

References

1. Arthur, D., Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA, 2007.
2. Celebi, M., Kingrav, H., Vela, P., *A comparative study of efficient initialization methods for the k-means clustering algorithm*, Expert Systems with Applications, 2013
3. Forgy, E.W., *Cluster analysis of multivariate data: efficiency versus interpretability of classifications*, Biometrics, 1965.
4. Guo, X., Gao, L., Liu, X., Yin, J., *Improved Deep Embedded Clustering with Local Structure Preservation*, IJCAI, 2017.
5. Hartigan, J. A., Wong, M. A., *Algorithm AS 136: A K-means clustering algorithm*, Applied Statistics, 1979.
6. Krause, A., Liebscher, V., *Multimodal projection pursuit using the dip statistic*, Preprint-Reihe Mathematik, 2005.
7. Mautz, D., Ye, W., Plant, C., Böhm, C., *Towards an Optimal Subspace for K-means*, KDD, 2017.
8. Vinh, N. X., Bailey, J., *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance*, JMLR, 2011.
9. Xie, J., Girshick, R., Farhadi, A., *Unsupervised deep embedding for clustering analysis*, ICML, 2016.