Prototype Based Clustering in High-Dimensional Feature Spaces

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.–Ing.)

angenommen durch die Fakultät für Informatik

der Otto–von–Guericke–Universität Magdeburg

von Dipl. Inform. Roland Winkler

geb. am 19.07.1982 in Bernburg

Gutachter

Prof. Dr. habil. Rudolf Kruse

Prof. Dr. habil. Frank Klawonn

P.D. Dr. habil. Alexander Hinneburg

Promotionskolloquium:

Magdeburg, den 18.06.2015

PROTOTYPE BASED CLUSTERING IN HIGH-DIMENSIONAL FEATURE SPACES

ROLAND WINKLER

DISSERTATION

Fakultät für Informatik
Otto–von–Guericke–Universität Magdeburg

Magdeburg, 11.12.2015

ABSTRACT

Clustering algorithms have many applications in high-dimensional feature spaces: group detection in astronomical spectral data, remote sensing for city planing, aircraft movement patterns analysis, text mining and group detection in social networks among many others. The mathematical background and practical use of clustering algorithms is currently not well understood if the feature space has more than 20 dimensions.

In my thesis, I study the 'curse of dimensionality' in terms of distance concentration. I show that this effect can be described using the pairwise covariances of the marginal distribution functions that model the data. Furthermore, I compare 10 prototype based clustering algorithms using 800,000 clustering results of artificial data sets. I investigate how and why clustering algorithms are effected by the number of features. Using the clustering results, I also review how well 5 of the most commonly used cluster quality indices estimate the true cluster quality.

ZUSAMMENFASSUNG

Clusteralgorithmen in hochdimensionalen Merkmalsräumen können vielseitig eingesetzt werden: zum Beispiel zur Gruppenerkennung in astronomischen Spektren, Analyse von Bodenbeschaffenheiten, für die Erkennung von Flugzeugbewegungsmustern, zur Textanalyse und zur Gruppenerkennung in sozialen Netzwerken. Wenn die Anzahl der Dimensionen 20 überschreitet, sind jedoch die mathematischen Hintergründe und praktische Anwendung von Clusteralgorithmen nur sehr wenig erforscht.

In dieser Arbeit untersuche ich den "Fluch der Dimensionen" mittels dem Begriff der Distanzkonzentration. Ich zeige, dass dieser Effekt im Datenmodell mittels der paarweisen Kovarianzkoeffizienten der Randverteilungen beschrieben werden kann. Zusätzlich vergleiche ich 10 prototypbasierte Clusteralgorithmen mittels 800.000 Clusterergebnissen von künstlich erzeugten Datensätzen. Ich erforsche, wie und warum Clusteralgorithmen von der Anzahl der Merkmale beeinflusst werden. Mit den Clusterergebnissen untersuche ich außerdem, wie gut 5 der populärsten Clusterqualitätsmaße die tatsächliche Clusterqualität schätzen.

ACKNOWLEDGMENTS

I want to thank my two supervisors, Prof. Rudolf Kruse and Prof. Frank Klawonn for their continuous support. Thanks especially to Prof. Frank Klawonn for his great advice, help and ideas and Prof. Rudolf Kruse for pushing me where necessary. Big thanks goes to Lars Stockmann, Lars Rösler and Annette Temme for sanity-checking my thesis and overall improving its quality. Thanks to Thorsten Mühlhausen and the DLR for their hospitality where I spend half of my thesis time and the AIP for hosting the data. Thanks goes to Christian Borgelt for posing the problem: 'Many prototype-based algorithms are not able to analyse high-dimensional data. But that problem is largely ignored by the community.' which got me interested in the topic in the first place.

I cannot express enough gratitude for all the countless people that unknowingly contributed to this work. That includes people in scientific history, the countless software engineers that created LaTeX and its packages, Java for programming, GIT for revision control as well as all the Organizations that created the open source software that I used.

I want to thank my parents and friends for their long lasting understanding that my priorities were often not with them. Last but not least, many thanks to Doreen Dittrich, my partner in life and love for her unending patience, occasional kicks in the backside and for her support when I spend my weekends with this thesis rather than with her.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

In the last 50 years, data processing has become more and more important. Computer systems have become more powerful while data gathering tools like sensors have become cheaper and more abundant. There is no reason to believe this trend could stop any time soon which makes it very likely that future data sets contain many observations with many data sources. Data is accumulated and analysed throughout our entire culture, in scientific experiments, in finance and economy, in social networks and countless other areas, a few examples are presented in Section 1.1.

Throughout this work, I will share some experiences and topic related thoughts in side notes like this one.

Data mining is the process of transforming data into useful information. Clustering, as part of data mining, is the search for patterns in previously unsorted collections of data. It is not well understood how the number of features/dimensions of a data set effects the process of clustering. This work is dedicated to fill this gap, the focus hereby lies on prototype based clustering algorithms.

A more detailed explanation of the title and a short introduction into the topic is presented in Section 1.2. More subtopics that are relevant for this work are approached using 4 thesis questions, stated in Section 1.3, along with a motivation for each question.

1.1 EXAMPLES

To understand the significance of this thesis, first consider the examples presented in this section. The examples range from physical applications in spectroscopy over text clustering to clustering in graphs. Examples from economic systems like stock markets or customer characterization are not directly addressed but the example of clustering aircraft movements on an airport can be regarded as a representative. All these examples have one thing in common: the data objects of these data sets have many attributes and the data mining task is to find groups of data objects that belong together. Even the limited number of examples in this section show a large variety of applications, hence give a glimpse in the diversity of the topic. There are possibly hundreds of other applications for clustering high-di-

mensional data sets, many might not even be similar to the presented examples.

1.1.1 *Astronomical Spectral Data*

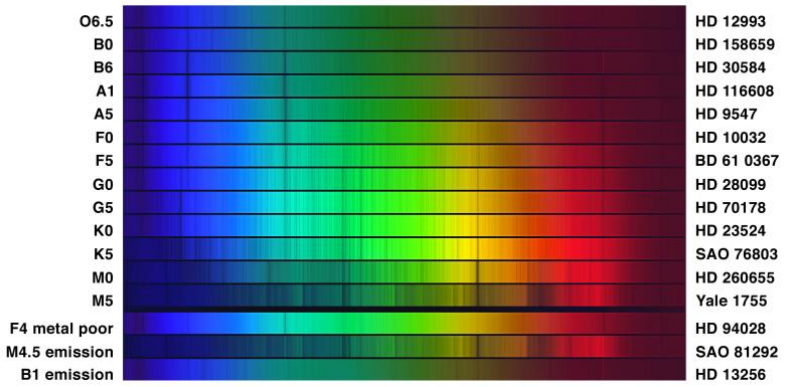


Figure 1.1: Spectra of different types of stars[1]

In astronomy, spectra of stars or distant galaxies contain a lot of information [Ostlie and Carroll, 2006]. By observing the spectrum of a star, it is possible to estimate its chemical composition, its lateral velocity and its rotational speed. The stars photosphere radiates light with a distinct range of wavelength depending on its temperature. The stars chromosphere above the photosphere does not radiate as much light as the photosphere and is almost not visible. In contrary, depending on the chemical composition of the chromosphere, parts of the light is filtered out and becomes visible as dark lines in the stars spectrum. In Figure 1.1, several different spectra of various stars are presented.

It is expected that most stars are born in groups within large molecular clouds [Ostlie and Carroll, 2006]. Each molecular cloud has a unique fingerprint of chemical elements that are roughly equally distributed through the entire cloud. Consequently, the stars, which are born from a single molecular cloud, should have roughly the same composition. In our galaxy, these star clusters dissolve (e.g. the stars loose their gravitationally bound to the host cluster) over time and the group structure of the stars is not apparent any more today. But the stars, which are born together, should have similar chemical composition and some other galactic orbital parameters, even though their location might diverge over time. Scientists hope this can indicate which stars originally belonged together in order to reconstruct the history of the milky way galaxy. This problem is essentially a cluster-

1 Source: `http://en.wikipedia.org/wiki/File:Obafgkm_noao_big.jpg`, *Date:* 18.04.2004, *NASA*

ing task in a feature space with many parameters, hence an example for a high-dimensional clustering problem.

1.1.2 *Hyper-spectral Remote Sensing*

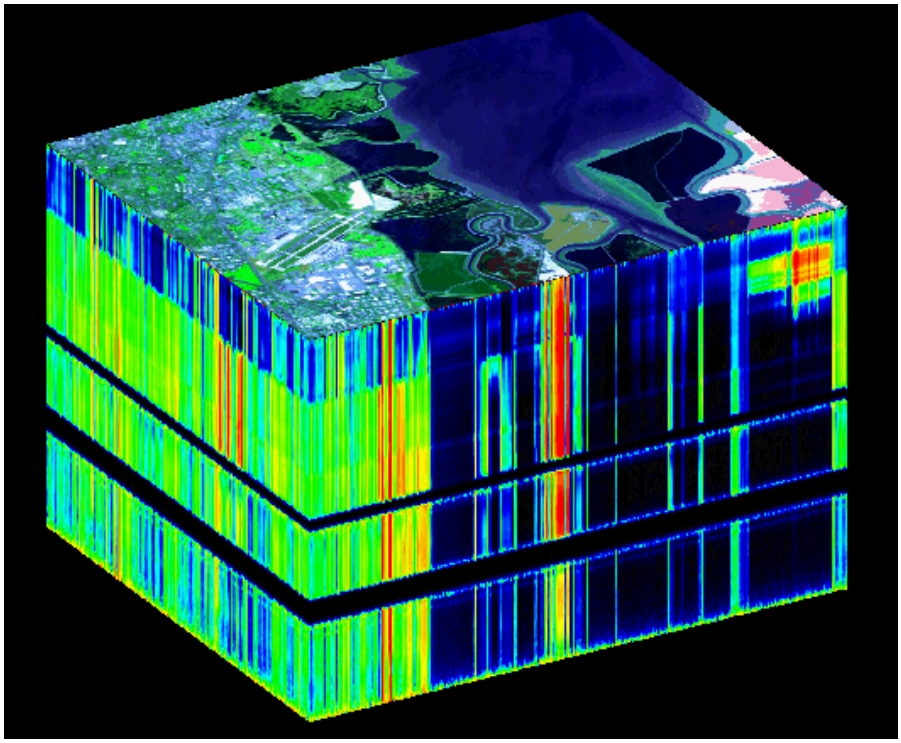


Figure 1.2: Aviris data example[2]

Hyper-spectral remote sensing is a process of taking pictures of the earth surface from a satellite or airplane [Lee et al., 1999; Melgani and Bruzzone, 2004] and observing a broad range of frequencies in the electromagnetic spectrum. Analysing the data gives information about the chemical composition of the surface and can be used for example in mineralogy, biology, agriculture, or city planning. Applications are for example identifying the health status of a forest or the type of crops in agricultural areas or its status of fertilisation. The data is usually a 2-dimensional image with each pixel containing the spectrum (instead of the usual RGB colors). Due to the spectrum, the data is essentially a 3-dimensional cube with each voxel (3D-pixel or volume pixel) holding a single luminosity value. For example the AVIRIS project of NASA can produce images like shown in figure 1.2. The third dimension in this figure represents the spectral data in each pixel.

2 Courtesy NASA/JPL-Caltech, source: *http://aviris.jpl.nasa.gov/data/image_cube.html*, *Date: 13.02.2013*

This type of 3-dimensional representation is usually not well suited for data mining because the interpretation of the spectral lines is missing. Instead, the spectra are reduced to the relative strength of emission and absorption lines. The number of interesting lines can easily reach 100, resulting in a high-dimensional clustering problem with each pixel of the 2-dimensional image and its related spectral lines interpreted as a data object. The additional area information of the location of pixels creates a challenging complication to the distance measurement between two data objects and increases the structural complexity of the data set.

1.1.3 *Aircraft Movement Patterns*

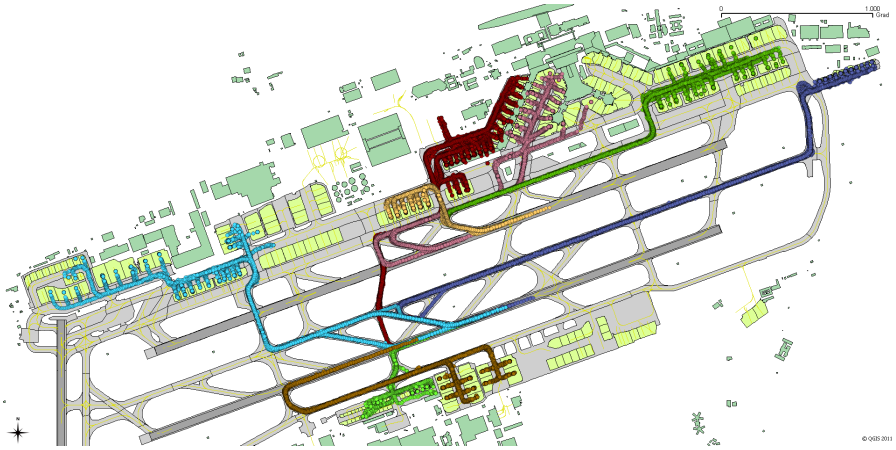


Figure 1.3: 8 Clusters out of a total of 62 clusters in the S.O.D.A. dataset. The clusters are colour coded and each data object is represented by a series of dots.

A closely related example to this work is the clustering of aircraft movements at Frankfurt airport, published in [Winkler et al., 2013]. During my time at DLR, I worked on this data set in order to improve the data analysis process in cooperation with Fraport AG. They develop a tool called S.O.D.A. (Surveillance Data Analysis Tool) to analyse the movement patterns of aircraft on the airfield of the Frankfurt airport. At the time of analysis, the database contained approximately 700,000 aircraft tracks and the goal was to find groups of aircraft that move on similar routes.

Due to the large number of tracks in the database and the complexity of comparing two tracks directly, we decided to simplify the task by transforming the data. A set of 457 reference points are defined on the airport traffic ways. For each track, the closest distance to each reference point is computed. To simplify the data further, the distance values are transformed using a simple, trapezoid fuzzy rule [Zadeh, 1965]. Each fuzzified distance value corresponds to one feature in the final data set, the resulting dataset is therefore 457-dimensional.

Using a special variation of the fuzzy c-means clustering algorithm [Winkler et al., 2013] (not presented in this work), a total of 62 clusters were found. In Figure 1.3, 8 out of the 62 clusters are presented. To reduce the overlapping effect of fuzzy clustering for this figure, only tracks with a membership value of at least 0.85 to their respective cluster are shown. Without taking special care for the high number of features, this result would not have been possible.

The algorithm works well, but it has significant shortcomings in terms of mathematical rigidity. This algorithm fails my current quality requirements and is therefore not presented in this work.

The cluster structure of this data set provides an additional insight in movement patterns on the airport. This data can be used for example to verify simulations of the airport. It might also be used to estimate typical taxi times for aircraft which in turn reduces the uncertainty of travel time predictions for customers.

1.1.4 *Text Mining*

Text clustering is used to automatically grouping documents, sorting emails, finding and assigning keywords, automated text searches for similar documents, and many more [Aggarwal and Zhai, 2012; Hotho et al., 2001; Shafiei et al., 2007]. The most common approach is, to assign a score for each word or phrase in a document. The vector of scores of a document is then interpreted as a data object, which lies within a very high-dimensional feature space. When limiting a set of documents to a list of terms, a fixed-dimensional feature space is generated, which can be used to cluster the documents according to their location in the feature space. The dimensionality of such text clustering feature spaces can be very high. Thousands of dimensions are common with the additional problem that most values in a data object are 0 [Katz, 1987]. Most approaches use dimension reduction techniques to approach the problem in a meaningful manner, but even the reduced dimensionality can be high. Examples for such data sets can be found in the UCI machine learning database, for example the *bag of words*[3] or the *Reuter 50 50*[4] data set.

1.1.5 *Network Clustering*

Clustering a graph structure to find groups of connected subgraphs is not a typical application for clustering algorithms, as they mostly operate on vector spaces. There is an ever increasing need of finding groups in graphs. Typical algorithms to find cliques in graphs are too restricted for many applications, for example in social networks [Ma et al., 2010; Saha et al., 2011]. Also in biology and medical research, groups are searched in graphs, for example for protein–protein interactions [Asur et al., 2007]. These applications have in common that

3 http://archive.ics.uci.edu/ml/datasets/Bag+of+Words
4 http://archive.ics.uci.edu/ml/datasets/Reuter_50_50

the data sets are mapped to a very high-dimensional real vector space prior to the data mining process. The dimension reduction techniques generally increase the complexity of the group finding process, because it cannot be guaranteed that the group structure before and after the data reduction process is identical. A clustering algorithm, which can be applied directly on the high-dimensional projection of the graph data, can avoid this problem provided the algorithm is able to find clusters in that high-dimensional space.

1.1.6 *Microarray Data*

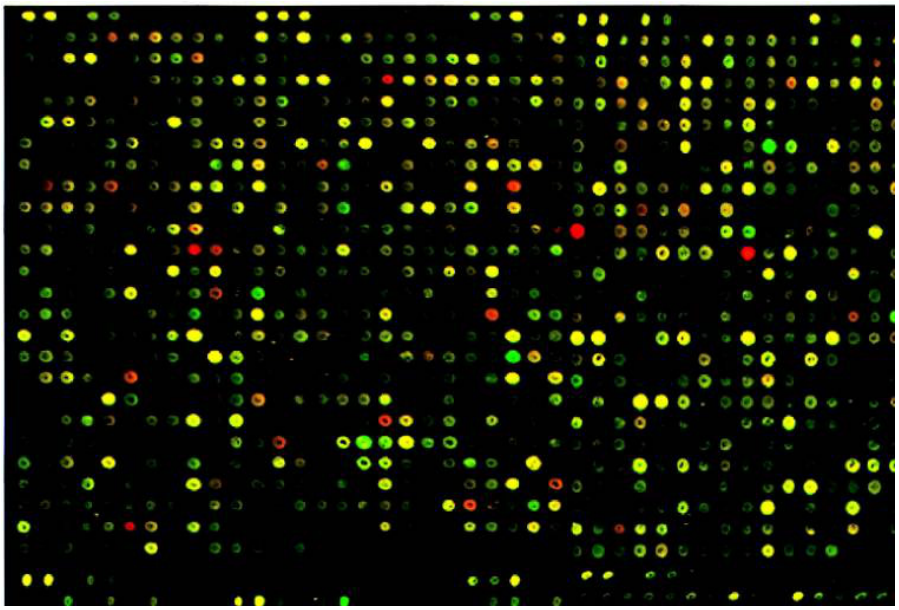


Figure 1.4: A small part of a micro-array.

The huge successes of microarrays in biology [Ness, 2006] resulted in a large afford to analyse data, which has more dimensions than data objects. An example of a microarray is presented in Figure 1.4, see also [Lashkari et al., 1997]. Each spot in the figure corresponds to one type of molecule (e.g. a gene) and is interpreted as a feature of the data set. Since it is easier to prepare molecules of the same origin than to acquire samples (or patients), the number of features often exceeds the number of data objects in microarray data sets. In this thesis however, it is assumed that a data set has much more (ideally several magnitudes more) data objects than features, even though the number of features is high. Therefore, the contents of this work are generally of limited value to microarray data mining.

1.2 DATA MINING

The science of data mining is concerned with the question: how to find knowledge within data, see [Frawley et al., 1992]. But more data does not automatically lead to more knowledge. Counter-intuitively, extracting knowledge from data can become more complex and difficult if data is more abundant because the data complexity may rise with the available data. There are basically two ways the amount of data can become large: by a huge number of observations and by storing more values per observation. Each observation is represented by a data object, which mathematically is a tuple or vector $\vec{x}$ of m attributes, which are usually called features or dimensions in this work. The number of dimensions m specifies the number of features of a data set or likewise the number of attributes/parameters of each data object. The mathematical environment for the data is referred to as feature space. A data set X is in this work limited to a finite subset of an m-dimensional real vector space: $X \subset \mathbb{R}^m$. The problems connected to data sets with many features/dimensions are investigated within this work.

1.2.1 *Prototype based Clustering*

It is assumed that a data set consists of multiple disjoint subsets (classes) of data objects. Likewise, it is assumed that the classes are generated from independent processes and it is not known which data object belongs to which class. Clustering is the process of partitioning data objects into disjoint subsets, called clusters, in such a way, that the clusters can be associated with the classes of the data set [Jain et al., 1999]. In other words, the data set has an inherent and hidden structure, determined by the classes. Clustering is the task of finding that structure. Please note, that the term *class* refers to a property of the data set while the term *cluster* refers to the result of a clustering algorithm, but both are partitions of the same data set. Ideally, the clusters and classes of a data set are identical. As will be seen later however, this is often not the case.

During writing this theses, in many occasions I needed a way to distinguish classes from clusters. This specific distinction is not shared within the clustering/classification community. Many authors use the two terms synonymously or make no distinction.

There are many approaches to clustering [Fayyad et al., 1996], prototype based clustering (see Chapter 3) is one of them. A prototype of a cluster can be seen as a representative of all data objects within this cluster. It can also be associated with the centre of the cluster. An example for a prototype based clustering algorithm, applied to an artificial data set is presented on the left-hand side in Figure 1.5. In this example, a 2-dimensional data set with 3 classes is clustered by the fuzzy c-means clustering algorithm (see Section 3.3). The prototypes are represented as black outlined circles and the association of data objects to their clusters are colour coded. The 'tails' of the prototype represent the trajectory the prototypes took from their (ran-

dom) starting position until their final position, they visualise the progress of the clustering process. The concentric height-lines represent the strength of cluster association, which are at levels of 0.9, 0.8, 0.7, 0.6 and 0.5. Please note, that the 0.5 cluster association height-lines between two neighbouring clusters do not touch due to the residual influence of the third cluster. In this example, the classes (groups of data objects) are well represented by the clusters, visualised by the colour assignments.

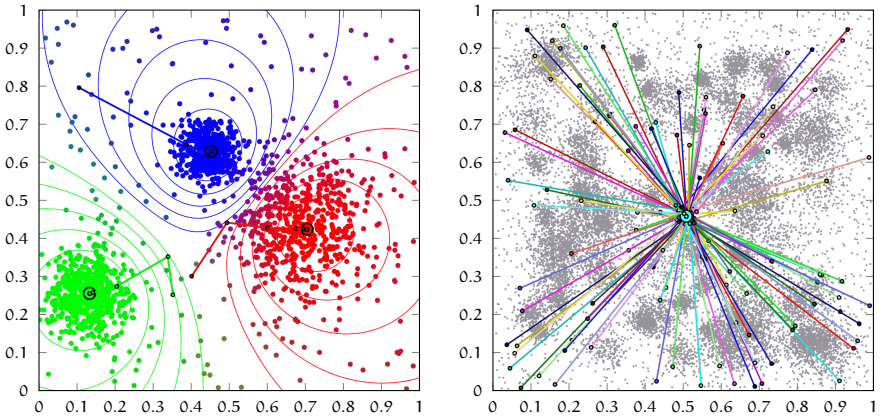


Figure 1.5: A 2-dimensional example data set with 3 classes/clusters (left) and a 2-dimensional projection of a 50-dimensional data set with 100 classes/clusters (right), both clustered with the standard fuzzy c-means clustering algorithm, see Section 3.3.

1.2.2 *The influence of Dimensions*

This problem is similar to: How many trees make a forest? How many grains of sand are required to form a heap? It is not possible to draw a crisp line here.

The data set in the example on the left-hand side of Figure 1.5 has only 2 features/dimensions. In reality, the number of dimensions m can be high, 10 or more are very common, see for example the data sets in the USI mashine learning repository [Bache and Lichman, 2013]. What exactly *high* means depends on the application (examples are presented in the next section) or on the data mining (i.e. clustering) algorithm. From personal experience, data sets with 5 or less features are usually not regarded as high-dimensional, while data sets with 100 and more features certainly can be regarded as high-dimensional. The notion is somewhat fuzzy in between, for example 20 dimensions is in the grey area and for some algorithms 20 dimensions is already a problem, for others it is not as indicated by the results in Chapter 5. Generally, the number of dimensions m can be regarded as high (and the vector space $\mathbb{R}^m$ as high-dimensional feature space) if the number of features m has a significant influence on the data mining procedure.

A high number of features can cause unexpected problems, loosely called the 'curse of dimensionality' [Bellman, 2003]. On the right-

hand side in Figure 1.5 a 2-dimensional projection of a data set with 50 dimensions and 100 classes is presented. The data set is generated using similar rules as in the left-hand side example and it is again clustered with the fuzzy c-means clustering algorithm. When extrapolating from the 2-dimensional example, the clustering algorithm should work, but apparently, something is wrong. All prototypes run from their randomly defined starting position into the centre of mass of the data set. As a consequence, all clusters become identical and share the entire data set equally. None of the clusters in this example can be associated with any of the classes in the data set, the clustering result is useless. This work is dedicated to explain this and similar effects and to analyse the influence of the number of dimensions m on the process of clustering, specifically for prototype based clustering algorithms.

When I observed this effect for the first time, I was convinced there is a bug in the software. But that was not the case, this effect is real.

1.2.3 *Dimension Reduction*

When dealing with high-dimensional data, the first idea is often to reduce the number of dimensions. In this work however, dimension reduction is not further considered. The explanation for this decision is presented in this subsection.

It is possible that the data objects of an m-dimensional data set are located in a lower $< m$-dimensional manifold. If all data objects of the data set are located in this lower dimensional manifold, the true (lower) dimensionality of the data is called intrinsic dimensionality [Silva and Tenenbaum, 2002]. For example, the data could lie on the surface of a m-dimensional hyper-sphere, which is $m-1$ dimensional, one dimension lower than the embedding m-dimensional feature space. The general approach of dealing with lower intrinsic dimensionality is to use dimension reduction approaches in order to reduce the complexity of the data set.

Dimension reduction can be done in parallel to clustering, see for example [Belkin and Niyogi, 2001; Chakrabarti and Mehrotra, 2000]. These procedures however are very complex and out of scope of this work.

Alternatively, the number of dimensions can be reduced prior to performing the clustering algorithm, hence simplifying the clustering process. Feature selection is (as the name states) a collection of methods to select features (i.e. dimensions or attributes), which seem to be relevant for later analysis, see for example [Guyon, 2006]. Feature selection is typically based on metrics to specify the amount of (marginal) information of the individual features, the most common might be the Kullback-Leibler information gain [Kullback and Leibler, 1951]. Alternatively feature subset selection can be used, see [Hall, 1999], where not individual features but subsets of features are rated and the best subset is selected. Feature selection is used for example if

there are many more features than data objects, see the microarray data example in Section 1.1.6.

Linear feature extraction is more general than feature selection because it finds arbitrarily oriented linear subspaces of the original feature space and projects the data to these linear subspaces. The most popular algorithm for linear feature extraction is principle component analysis (PCA) [Jolliffe, 2002; Pearson, 1901]. Another approach is independent component analysis (ICA) [Comon, 1994], where it is assumed that the data is a mixture of several, statistically independent sources.

Finally, non-linear feature extraction generalises linear feature extraction further by projecting the data into general lower-dimensional manifolds. The problem however is, that general manifolds can be much more complex than linear subspaces. Some of the algorithms for non-linear feature extraction are based on other data mining ideas like self organizing maps [Kohonen, 1982, 2001]. The kernel trick can also be used to transform the PCA method into a non linear transformation: kernel PCA [Schölkopf et al., 1997]. Isomap [Tenenbaum et al., 2000] is jet another algorithm capable of mapping the data into a non linear manifold. A starting point for more research on the topic of feature extraction can be found in [Guyon, 2003, 2006].

Another algorithms of dimension reduction is based on non-linear point mapping, proposed by [Sammon, 1969]. In this case, the data is directly transformed into a lower dimensional representation by minimizing the difference in their pairwise distance before and after the projection.

There are many arguments pro- and contra dimension reduction. If it is known in advance, that the data is located in a lower dimensional manifold and the shape of the manifold is known in advance, it is usually a good idea to use an appropriate dimension reduction algorithm. If this is not the case, dimension reduction is not recommended. It is generally not possible to guarantee that the cluster structure of the data is identical in the lower dimensional representation of the data as it is in the original data set. Of course, this is the goal of all dimension reduction algorithms, but it is usually not known how well this goal is achieved and there is currently no good way to test it. Therefore, it is not possible to know that dimension reduction does more good in terms of complexity reduction than bad in terms of obscuring the cluster structure within the data set. Also, it might happen that the dimensionally reduced data is still high-dimensional. Due to these arguments, dimension reduction is not further considered for this work.

1.3 MOTIVATION, THESIS QUESTIONS AND OUTLINE

As demonstrated in Section 1.1, the search for patterns (clustering) in a data set has a wide variety of applications. It is foreseeable that many more of this type of applications become relevant in the near future. This makes the topic of this thesis ever more significant.

Prototype based clustering is one of the (if not the) most widely used strategy for finding clusters in data sets and the process is very well understood if the number of dimensions is small. As already presented in Figure 1.5 however, standard prototype clustering algorithms do not always work as intended in high-dimensional feature spaces. Even worse, the problems connected to the dimensionality are poorly understood on the algorithmic level. There is some mathematical ground-work on this subject, most notably in [Beyer et al., 1999; Durrant and Kabán, 2008; Kabán, 2011], but that is not sufficient to understand the effects properly. It is also not enough to engineer a well working clustering algorithm.

This topic is summarised by thesis question Q_1:

Every scientific journey starts with a question. Or in this case, four of them. Going on this journey without a question or goal of some kind is like pushing a well cooked noodle through space.

Q_1 What is the 'curse of dimensionality' in the framework of clustering and under which circumstances does it occur?

Due to its importance for the remaining part of this thesis, Q_1 is approached first, in Chapter 2.

In low-dimensional feature spaces, it is often possible to just plot the data set in a suitable format and guestimate the class structure. This is possible because a projection on two dimensions often holds enough information to make a visual inspection of the data. In high-dimensional feature spaces, this is only possible in very simple cases, as presented in Figure 1.5 for example. In contrast, consider Figure 1.6 which is the 2-dimensional projection of a data set with 15 dimensions and 10 classes but with a more complex shape of the classes. As a human, it is impossible to eyeball the result. Even when colour coded, it is impossible to see the class structure well because there are no 2-dimensional projections that would clearly separate the classes.

In such a circumstance, it is very difficult to select a clustering algorithm, which would likely work on the data set. The second and third thesis questions are dedicated to that problem:

Q_2 Which prototype based clustering algorithms can be used for high-dimensional data sets?

Q_3 How is the clustering quality influenced by the number of dimensions and the number of classes/clusters?

Thesis question Q_2 demands a comparison of clustering algorithms and an expectation of their performance. The other thesis question, Q_3 goes deeper and demands an explanation of why some algorithms perform well and others do not. Such questions are best approached using a benchmark. In many papers, individual algorithms

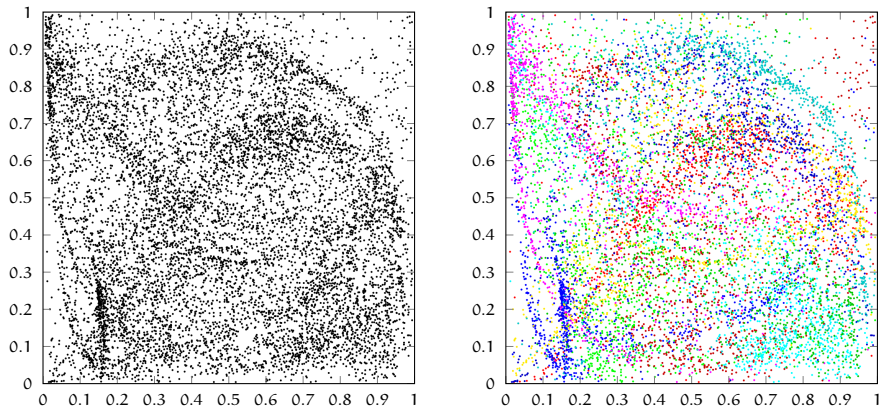


Figure 1.6: A 2-dimensional projection of a 15-dimensional data set with 10 classes. The pure data set is presented on the left-hand side and the same data set with colour-coded class information is presented on the right-hand side. This is an example of a data set from data set family D_3, see Section 4.1.3.

are applied on a few, simple artificial data sets or on well known data sets such as the Iris data set [Anderson, 1935], the Wine data set or the Blood type data set [Bache and Lichman, 2013]. None of these is enough to answer thesis questions Q_2 and Q_3 because they are too selective. A large survey, ranging over many numbers of dimensions and several different types of data sets with different properties are necessary, which is exactly what is done within this thesis.

A range of prototype based clustering algorithms are defined and their application is discussed in Chapter 3. The algorithms are tested and compared by using a Benchmark, which is presented in Chapter 4. The benchmark is fairly complex and is designed to answer not only Q_2 and Q_3 but also Q_4 (see below). Four different families of procedurally generated data sets (described in detail in Appendix A) with a variety of dimensions and classes are used. Also, Chapter 4 contains a description of how to measure the quality of a clustering result, hence allowing a quantitative comparison of algorithms based on the knowledge of the correct result. In total, 800'000 clustering processes are performed, which gives a large enough foundation to compare the algorithms on a statistically significant level.

To answer the third thesis question, Q_3, a deeper understanding about the behaviour of clustering algorithms in high-dimensional feature spaces is required. It is especially interesting to understand the transition between a successful clustering in low-dimensional feature spaces and an unsuccessful clustering in high-dimensional feature spaces. The hope is that this knowledge leads to the design of clustering algorithms that are more resistant to the number of dimensions (and classes/clusters), especially for algorithms that are not discussed in this work. The number of classes within a data set is similarly rel-

evant as the number of dimensions (see [Winkler et al., 2011a]), which is why the number of classes is part of this question.

There is one more topic that needs to be addressed for clustering in high-dimensional feature spaces. It is expressed by asking the last thesis question:

Q_4 Are internal cluster quality index measures useful to assess the quality of clustering results in high-dimensional feature spaces?

Q_4 demands an answer to the problem: How to determine the quality of a clustering result if the true class structure is unknown? Besides finding a good clustering algorithm, this is maybe the most pressing problem for clustering high-dimensional feature spaces. It is not intuitively clear, how the quality of a clustering result can be estimated without the knowledge of the correct result. As already demonstrated in Figure 1.6, a visual inspection might not always give a good indication.

Usually, internal cluster quality indices are used to estimate the quality of a clustering result. In this work, 5 of the most commonly used indices are reviewed to answer thesis question Q_4, they are presented in Chapter 4. The large number of clustering results, which are generated to answer thesis questions Q_2 and Q_3, provide the necessary data to review internal cluster quality indices on a statistical level. Therefore, Q_4 is approached simultaneously with Q_2 and Q_3 by using the same benchmark. Since the true class structure of the data sets is always known, it is possible to judge the performance of the internal cluster quality indices.

In Chapter 5, the results of the benchmarks are presented and the answers to the three practical thesis questions Q_2, Q_3 and Q_4 are finally given. The huge amount of data, which is gathered throughout the benchmarks cannot be completely presented in Chapter 5. In Appendices D, E and F a complete overview of all relevant benchmark results are provided in graphical form.

All relevant software that was developed for this work is available online as open source project EDMOAL, a brief overview is available in Appendix B. It contains all algorithms which were required to generate the results within this work. Likewise, all data sets, results and other information to reproduce the results within this work are available online, see Appendix C.

The answers to all thesis questions are summarized in Chapter 6. This last chapter also contains a critical discussion about the decisions, which had a significant influence on this work. Furthermore, open questions are addressed and points for further research are presented.

2

THE CURSE OF DIMENSIONALITY

The term *curse of dimensionality* was coined by Richard E. Bellman in 1957 on his topic of dynamic programming [Bellman, 2003]. The computational problems at that time (1957) were significantly different to the ones we face now, but the problem of gaining knowledge from high-dimensional data sets is still as significant as in 1957. The curse of dimensionality has many interpretations but might be summed up with: 'Some algorithms that work well on low-dimensional data do not produce useful results when applied on high-dimensional data'. That description however does not give any useful insight into the problem, clearly a mathematical formulation is required. Within this chapter, such a description is presented.

For practical reasons, the reference is pointing to the reprinted version of 2003.

For computational problems that involve the distances to reference points (e.g. least squares methods), prototypes (e.g. clustering or classification) or query points (e.g. k-nearest neighbours), the curse of dimensionality is best expressed in terms of distance concentration, which is defined in the first section in this chapter. In Section 2.2, some necessary and sufficient conditions for distance concentration under the assumption of using a p-norm induced distance function are presented. The thesis question Q_1 is also discussed in that section. Following the restriction on the p-norm induced family of distance functions, a short discussion on the choice of distance functions is presented in Section 2.3. For a given data set, it is not immediately clear how strong distance concentration might effect any algorithm if the distribution function of the data is unknown. The gradient descend/ascend method can be used to get estimates on distance concentration, which is presented in Section 2.4.

In the last three sections in this chapter, aspects that are more generally connected to high-dimensional feature spaces are considered. One of the major problems of high-dimensional feature spaces is, that it is hard to test if an assumed distribution of the data is plausible, which is presented in more detail in Section 2.5. Subsequently, the influence of noise and outliers as well as missing values in high-dimensional feature spaces is discussed in Section 2.6.

Most topics discussed in this chapter have a wide range of application, much wider than clustering. To note the importance of dis-

tance concentration for prototype based clustering in particular, this chapter closes with a short discussion on this topic in Section 2.7.

2.1 DISTANCE CONCENTRATION

The first to describe the curse of dimensionality by a model of *distance concentration* were [Beyer et al., 1999]. In the paper, they analyse problems for a k-NN (k-nearest neighbour) search in high-dimensional data. Although the mathematical description is for k-NN searches, it is also valid for describing problems of clustering algorithms because the relevant factor are the distances to a query point (prototype). Most data mining algorithms require a finite dimensional feature space, which is why the theory presented below is restricted to m-dimensional vector spaces $\mathbb{R}^m$.

Actually, [Beyer et al., 1999] did not use the term 'distance concentration', instead they used 'stability' or 'meaningfulness' of distances. I prefer the term 'distance concentration' as it describes the effect more precisely.

2.1.1 *Definition of Distance Concentration*

Let $m \in \mathbb{N}$ denote the dimensionality or the number of features of a data set, a random variable or any other dimension dependent object. Let $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ be a distance function on the vector space $\mathbb{R}^m$, that satisfies the properties $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^m$:

$$\begin{aligned} d(\vec{x},\vec{y}) &= d(\vec{y},\vec{x}) && (\textit{symmetry}) \\ d(\vec{x},\vec{y}) &= 0 \Leftrightarrow \vec{x} = \vec{y} && (\textit{identity of indiscernibles}) \\ d(\vec{x},\vec{y}) &\geqslant 0 && (\textit{non-negativity}) \\ d(\vec{x},\vec{y}) &= d(\vec{x}+\vec{z},\vec{y}+\vec{z}) && (\textit{translation invariance}) \end{aligned} \tag{2.1}$$

If d additionally satisfies

$$d(\vec{x},\vec{z}) \leqslant d(\vec{x},\vec{y}) + d(\vec{y},\vec{z}) \quad (\textit{triangle inequality}) \tag{2.2}$$

it is called a metric and can be induced by norms $\|\cdot\|$: $d(\vec{x},\vec{y}) = \|\vec{y}-\vec{x}\|$. Distance functions induced by 'fractional norms', as they are often discussed in conjunctions with distance concentration and the curse of dimensionality (see Section 2.3.2), are valid candidates for d that do not satisfy the triangle inequality hence are not a metric. The first to describe the following definition of distance concentration were again Beyer et al. [Beyer et al., 1999], with a slight difference: they introduced a parameter $p \in \mathbb{R}_{>0}$ as an exponent to d. Due to construction of Properties (2.1): if d is a distance function, also d^p is a distance function. Therefore, it is not necessary to keep noting the exponent p in all equations as Beyer et al. did in their paper.

Annoyingly, 'fractional norms' are not norms because they do not satisfy the triangle inequality. I still go with the generally accepted term here.

The symbol $\mathbb{R}_{>0}$ refers to all positive real values excluding 0, likewise: $\mathbb{R}_{\geqslant 0}$ refers to all positive real values including 0.

Nevertheless, the parameter p can be helpful if d is induced by a p-norm:

$$d(\vec{x},\vec{y}) = \Big(\sum_{k=1}^{m} (|x_k - y_k|^p)\Big)^{\frac{1}{p}} \tag{2.3}$$

because the outer exponent vanishes for d^p, which is used in Section 2.2.2.

The set of distances from a query point $\vec{q} \in \mathbb{R}^m$ to all data objects of a data set $X \subset \mathbb{R}^m$ is defined as

$$D_{\vec{q}}(X) = \{d(\vec{q}, \vec{x}_j) \mid \vec{x}_j \in X\} \tag{2.4}$$

Mathematically, a set cannot hold two identical objects. Since all data objects in a data set have a unique number assigned to them, they are not identical w.r.t. the definition of a set. The ID is however never used in any calculation or equation.

The empirical mean (sample expectation) of distances is then defined as

$$\widehat{E}(D_{\vec{q}}(X)) = \frac{1}{n} \sum_{j=1}^{n} d(\vec{q}, \vec{x}_j)$$

where $n = |X|$ is the number of data objects in data set X. The empirical variance of distances is defined as

$$\widehat{V}(D_{\vec{q}}(X)) = \frac{1}{n-1} \sum_{j=1}^{n} (d(\vec{q}, \vec{x}_j) - \widehat{E}(D_{\vec{q}}(X)))^2$$

From that, the empirical relative variance of distances is defined as:

$$\widehat{RV}\left(D_{\vec{q}}(X)\right) = \frac{\widehat{V}\left(D_{\vec{q}}(X)\right)}{\widehat{E}^2\left(D_{\vec{q}}(X)\right)} \tag{2.5}$$

Analogously, the relative variance (without the term 'empirical') is defined for random variables. Let the distributions G and H be joint cumulative distribution functions in $\mathbb{R}^m$. Furthermore, let $\vec{X}$ and $\vec{q}$ be m-dimensional random variables that are G- and H-distributed respectively: $\vec{X} \sim G$ and $\vec{q} \sim H$; also let $\vec{X}$ and $\vec{q}$ be independent. Then $D_{\vec{q}}(\vec{X}) = d(\vec{X}, \vec{q})$ is a random variable of distances. Note that the notation here is analogously chosen to the case of samples above, but $D_{\vec{q}}(\vec{X})$ is a random variable in $\mathbb{R}$ instead of a set of distances. The mean E as well as the variance V of $D_{\vec{q}}(\vec{X})$ can be directly specified using the distributions G and H. The relative variance is defined analogously to the empirical relative variance:

Note the different font for random variables X and data sets X. The difference is subtle, but since the two mathematical objects stand at the same place in similar equations, the similarity is intentional.

$$RV\left(D_{\vec{q}}(\vec{X})\right) = \frac{V\left(D_{\vec{q}}(\vec{X})\right)}{E^2\left(D_{\vec{q}}(\vec{X})\right)} \tag{2.6}$$

Also, it is assumed that expectation and variance exist for any distribution discussed in this chapter.

Let $G^{(m)}$ and $H^{(m)}$ be sequences of m-dimensional probability distribution functions into $\mathbb{R}^m$ where the notation (m) specifies the sequence element counter (increasing the dimensionality for each element in the sequence by 1). Furthermore, let $\vec{X}^{(m)}$ and $\vec{q}^{(m)}$ be sequences of independent m-dimensional random variables $\vec{X}^{(m)} \sim G^{(m)}$ and $\vec{q}^{(m)} \sim H^{(m)}$ and let $d^{(m)} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_{\geqslant 0}$ be a sequence of distance functions with the in (2.1) stated properties.

Other authors often require $H^{(m)} = G^{(m)}$, but that restriction is not necessary for the below developed theory.

Distance concentration is said to occur, if and only if the relative variance of distances becomes 0 for increasing dimensionality:

$$\lim_{m\to\infty} \mathrm{RV}\left(D^{p}_{\vec{q}^{(m)}}(\vec{X}^{(m)})\right) = \lim_{m\to\infty} \frac{\mathrm{V}\left(D^{p}_{\vec{q}^{(m)}}(\vec{X}^{(m)})\right)}{\mathrm{E}^2\left(D^{p}_{\vec{q}^{(m)}}(\vec{X}^{(m)})\right)} = 0 \tag{2.7}$$

Or maybe it is not that hard for a mathematician with a much higher skill than me at solving this type of problems.

Depending on the sequence of distributions $G^{(m)}$, $H^{(m)}$ and distance functions $d^{(m)}$, it can be very hard to calculate the sequence of relative variances without simplifications or approximations. It might be possible to calculate the relative variance for selected sequences of probability distributions of $\vec{X}^{(m)}$ easily, but the process is in general very difficult.

2.1.2 *Equivalence Theorems*

In the past years, several equivalent expressions are found for equation (2.7). Actually, Beyer at al. described their results in form of Equation (2.8). The relative variance of distances is equivalent to the variance of relative distances:

$$\begin{aligned}
\frac{\mathrm{V}\left(D_{\vec{q}}(\vec{X})\right)}{\mathrm{E}^2\left(D_{\vec{q}}(\vec{X})\right)} &= \frac{\mathrm{E}\left(D_{\vec{q}}(\vec{X})^2\right) - \mathrm{E}^2\left(D_{\vec{q}}(\vec{X})\right)}{\mathrm{E}^2\left(D_{\vec{q}}(\vec{X})\right)} \\
&= \frac{\mathrm{E}\left(D_{\vec{q}}(\vec{X})^2\right)}{\mathrm{E}^2\left(D_{\vec{q}}(\vec{X})\right)} - \left(\frac{\mathrm{E}\left(D_{\vec{q}}(\vec{X})\right)}{\mathrm{E}\left(D_{\vec{q}}(\vec{X})\right)}\right)^2 \\
&= \mathrm{E}\left(\left(\frac{D_{\vec{q}}(\vec{X})}{\mathrm{E}\left(D_{\vec{q}}(\vec{X})\right)}\right)^2\right) - \mathrm{E}^2\left(\frac{D_{\vec{q}}(\vec{X})}{\mathrm{E}\left(D_{\vec{q}}(\vec{X})\right)}\right) \\
&= \mathrm{V}\left(\frac{D_{\vec{q}}(\vec{X})}{\mathrm{E}\left(D_{\vec{q}}(\vec{X})\right)}\right)
\end{aligned} \tag{2.8}$$

Rather than arguing with a sequence of single random variables, let $X^{(m)}$ be a sequence of sets of m-dimensional random variables with n elements each: $X^{(m)} = \{\vec{x}_1^{(m)}, \ldots, \vec{x}_n^{(m)}\}$ on $\mathbb{R}^m$. Let $d_{\min}^{(m)}$ and $d_{\max}^{(m)}$ be random variables that are defined as:

The term 'almost surely' means, that the probability that the event occurs is 1. This is only important to exclude pathological cases of probability 0.

$$\begin{aligned}
d_{\min}^{(m)} &= \min_{\vec{x}^{(m)}\in X^{(m)}} \left(D_{\vec{q}^{(m)}}(\vec{x}^{(m)})\right) \\
d_{\max}^{(m)} &= \max_{\vec{x}^{(m)}\in X^{(m)}} \left(D_{\vec{q}^{(m)}}(\vec{x}^{(m)})\right)
\end{aligned}$$

Distance concentration occurs if and only if the ratio of maximum and minimum distances becomes 1 almost surely:

$$\lim_{m\to\infty} \mathrm{RV}\left(D_{\vec{q}^{(m)}}(X^{(m)})\right) = 0 \quad \Leftrightarrow \quad \frac{d_{\max}^{(m)}}{d_{\min}^{(m)}} \xrightarrow{P} 1 \tag{2.9}$$

with $\frac{d_{max}^{(m)}}{d_{min}^{(m)}} \xrightarrow{P} 1$ being the short notation of the convergence in probability: for all $\varepsilon > 0$, $\lim_{m\to\infty} P\left(\left|\frac{d_{max}^{(m)}}{d_{min}^{(m)}} - 1\right| > \varepsilon\right) = 0$. In other words, distance concentration occurs if and only if the ratio of distances between the furthest and nearest data object becomes 1. Note that this statement is *not* equal to stating that the distances become equal! It might happen, that the difference between d_{max} and d_{min} is constant or raises much slower than both distances which approach infinity. However, if there is no variation in relative distances, the relative difference between the largest and smallest distance to $\vec{q}^{(m)}$ becomes arbitrarily small. Of course, the argumentation works in the other direction as well, if the ratio between minimal and maximal distance to $\vec{q}^{(m)}$ becomes arbitrarily close to 1, the variation of relative distances must be arbitrarily small. In their original work [Beyer et al., 1999] described Equation (2.9) only in the forward direction. Later [Durrant and Kabán, 2008] published a converse theorem, reversing the implication hence creating an equivalence theorem.

Independently of Durrant et al., [Hsu and Chen, 2006, 2009] came to the same converse theorem, even though they used a different approach for the proof. They also defined other equivalent theorems which are now briefly addressed. With the continuous mapping theorem, the ratio of d_{max} and d_{min} can be reversed:

$$\frac{d_{max}^{(m)}}{d_{min}^{(m)}} \xrightarrow{P} 1 \quad \Leftrightarrow \quad \frac{d_{min}^{(m)}}{d_{max}^{(m)}} \xrightarrow{P} 1 \tag{2.10}$$

Also for any pair of random variables $\vec{x}_i^{(m)}$ and $\vec{x}_j^{(m)}$, $\forall i, j \in \mathbb{N}$, $1 \leqslant i \neq j \leqslant n$, it holds for all m:

$$\frac{d_{max}^{(m)}}{d_{min}^{(m)}} \geqslant \frac{d^{(m)}(\vec{x}_i^{(m)}, \vec{q}^{(m)})}{d^{(m)}(\vec{x}_j^{(m)}, \vec{q}^{(m)})} \geqslant \frac{d_{min}^{(m)}}{d_{max}^{(m)}}$$

Since the fractional between $d_{max}^{(m)}$ and $d_{min}^{(m)}$ is a special case of each pair of random variables, it holds:

$$\frac{d_{max}^{(m)}}{d_{min}^{(m)}} \xrightarrow{P} 1 \quad \Leftrightarrow \quad \forall i, j : \frac{d^{(m)}(\vec{x}_i^{(m)}, \vec{q}^{(m)})}{d^{(m)}(\vec{x}_j^{(m)}, \vec{q}^{(m)})} \xrightarrow{P} 1 \tag{2.11}$$

2.2 CONDITIONS FOR DISTANCE CONCENTRATION

The open question, under which circumstances distance concentration occurs, cannot be finally answered because the necessary and sufficient conditions for distance concentration remain unknown for arbitrary distance functions. But it is possible to evaluate the question if the distance function is induced by a (fractional) p-norm, raised to the power of p.

Let d be induced by a (fractional) p-norm and raised to the power of p: $\|\cdot\|_p^p$, which means it has the form

$$d(\vec{x}, \vec{y}) = \|\vec{y} - \vec{x}\|_p^p = \sum_{k=1}^{m} |\vec{y}_k - \vec{x}_k|^p \tag{2.12}$$

The relevance of this choice becomes apparent in the next chapter (see Chapter 3), as many prototype based clustering algorithms are restricted to the squared euclidean distance, discussed also in Section 2.7. The restriction to $\|\cdot\|_p^p$ induced distance functions is necessary only for technical reasons, it is used to decompose the norm into its dimensional components, which is heavily used within this section.

Let G and H be m-dimensional probability distributions and $\vec{X} \sim G$ and $\vec{q} \sim H$ as well as $\vec{Z} = \vec{q} - \vec{X}$ be m-dimensional random variables. Then $D_{\vec{q}}(\vec{X}) = d(\vec{X}, \vec{q}) = \|\vec{q} - \vec{X}\|_p^p = \|\vec{Z}\|_p^p$ is also a random variable. If $\|\vec{Z}\|_p^p$ is decomposed as defined in Equation (2.12), it holds

$$D_{\vec{q}}(\vec{X}) = \|\vec{Z}\|_p^p = \sum_{k=1}^{m} |\vec{Z}_k|^p \tag{2.13}$$

where $|\vec{Z}_k|^p$ are 1-dimensional random variables in $\mathbb{R}_{\geqslant 0}$. In the following subsections, the here defined symbols are used to show the occurrence of distance concentration, given further characteristics of $\vec{X}$ and $\vec{q}$.

Due to rewriting the distance function, the relative variance is expressed in:

$$\mathrm{RV}\left(D_{\vec{q}}(\vec{X})\right) = \mathrm{RV}\left(\|\vec{Z}\|_p^p\right) = \frac{\mathrm{V}\left(\|\vec{Z}\|_p^p\right)}{\mathrm{E}^2\left(\|\vec{Z}\|_p^p\right)} \tag{2.14}$$

and distance concentration occurs, if and only if:

$$\lim_{m\to\infty} \mathrm{RV}\left(\|\vec{Z}^{(m)}\|_p^p\right) = \lim_{m\to\infty} \frac{\mathrm{V}\left(\|\vec{Z}^{(m)}\|_p^p\right)}{\mathrm{E}^2\left(\|\vec{Z}^{(m)}\|_p^p\right)} = 0 \tag{2.15}$$

where the superscript (m) is again used to express sequences of random variables with the dimensionality m indicating the index in the sequence.

2.2.1 *Independent and Identical Distributed Dimensions*

In this first subsection on analysing the conditions for distance concentration, it is assumed that all dimensions are independent and identically distributed. It is shown that under this conditions, distance concentration occurs whenever the number of dimensions goes to infinity.

Let G and H be defined as above, such that $\vec{X}$ and $\vec{q}$ are independent random vectors with statistically independent and identically distributed (i.i.d.) components: $\vec{X}_k \sim G_1$ and $\vec{q}_k \sim H_1$. Since $\vec{X}$ and $\vec{q}$ are m-dimensional random variables and all individual dimensions are i.i.d. , all components $\vec{Z}_k$ are i.i.d. random variables with identical expectation μ and variance σ^2. Then the expectation of the normed dimensions $\mu_{|\cdot|} = E(|\vec{Z}_k|^p)$ are identical for all components of $\vec{Z}$. Analogously, the variances of the normed dimensions are identical: $\sigma^2_{|\cdot|} = V(|\vec{Z}_k|^p)$

$$E(\|\vec{Z}\|_p^p) = E\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m} E\left(|\vec{Z}_k|^p\right) = m\mu_{|\cdot|} \qquad \text{and}$$

$$V(\|\vec{Z}\|_p^p) = V\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m} V\left(|\vec{Z}_k|^p\right) = m\sigma^2_{|\cdot|}$$

Inserting this into Equation (2.15) yields:

$$\lim_{m\to\infty} \frac{V\left(\|\vec{Z}^{(m)}\|_p^p\right)}{E^2\left(\|\vec{Z}^{(m)}\|_p^p\right)} = \lim_{m\to\infty} \frac{m\sigma^2}{m^2\mu^2} = 0$$

Therefore, whenever the dimensions are i.i.d. and the number of dimensions approaches infinity, the distances concentrate for any $\|\cdot\|_p^p$ (fractional-) norm induced distance function. This result is independent of the data distribution G, the query point distribution H as well as the parameter p. In reality though, it is highly unlikely that data sets consist of samples of i.i.d. dimensions.

Initially I used the symbol F for what is distribution G in the text. Who thought that 'F-distribution' was a good name for a specific probability distribution? Next, I shall define a very specific '$\vec{x}$-vector', then all algebra books are screwed.

2.2.2 *Independent, Normalised Dimensions*

François et al. [François et al., 2007] claim that they found a way to prove that it is enough for the features to be independent, normalised and centred in order to induce the concentration of distances with increasing dimensionality. In their paper, they first prove the i.i.d. case using the strong law of large numbers. Subsequently, they state that a variation of the strong law of large numbers also hold if the data is not identically distributed but normalised and centred (expectation of the data generating random variable is 0 and its variance 1). However, the prove makes use of distinct values for the expectation of all dimensions which does only exist in the i.i.d. case. They do this assumption at the very beginning of their proof which is why I consider it invalid (see Step 1, Section 5.1.1. in their paper). I give here a prove to a similar problem, which uses the same arguments as the i.i.d. case in the last subsection. The difference is, that assumptions about $V(|\vec{Z}_k|^p)$ and $E(|\vec{Z}_k|^p)$ are formulated rather than $V(\vec{Z}_k)$ and $E(\vec{Z}_k)$ as in the paper of François et al.

Let G and H be defined as above, such that $\vec{X}$ and $\vec{q}$ are random vectors with independent (but not necessarily identical distributed) components: $\vec{X}_k \sim G_k$ and $\vec{q}_k \sim H_k$. Furthermore, let the dimensions be scaled such that $V(|\vec{Z}_k|^p) = 1$ and let $E(|\vec{Z}_k|^p) \geqslant K$ for a $K \in \mathbb{R}_{>0}$. The first condition ensures that all features are equally important when applying a data mining algorithm and that the scale at which the values are stored does not dominate the data mining process.

The second condition is rather technical to prevent pathological cases like the following: Let $\|X\|_1^1 = \sum_{k=1}^{m} |\vec{X}_k|$ be a random variable with $P(\vec{X}_k = 0) = 1 - (\sqrt{k^3} \cdot (k^3 - 1))^{-1}$, $P(\vec{X}_k = k^3) = (\sqrt{k^3}(k^3 - 1))^{-1}$. Then $\vec{X}_k = |\vec{X}_k|$, $V(|\vec{X}_k|) = 1$ and $E(|\vec{X}_k|) = \frac{\sqrt{k^3}}{k^3-1}$ and the expectation approaches 0 faster than $\frac{1}{k}$. Therefore, with increasing dimensionality, the relative variance increases (Equation 2.16 would not approach 0). But the distribution is rather pointless from a data mining perspective. In a sample of such distribution, all data objects are either zero or outliers, hence it holds no valuable information that could be mined. Therefore the property that $\exists K \in \mathbb{R}_{>0}$ so that $\forall k \leqslant m \in \mathbb{N} : E(\|\vec{Z}_k\|_d) \geqslant K$ is no constraint for practical applications.

Following the same argument as before and decomposing the d-norm into its components yields for all m:

$$E(\|\vec{Z}\|_p^p) = E\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m} E\left(|\vec{Z}_k|^p\right) \geqslant m \cdot K \qquad \text{and}$$

$$V(\|\vec{Z}\|_p^p) = V\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m} V\left(|\vec{Z}_k|^p\right) = m$$

Which, when inserted into Equation (2.15) yields:

$$\lim_{m\to\infty} \frac{V\left(\|\vec{Z}^{(m)}\|_p^p\right)}{E^2\left(\|\vec{Z}^{(m)}\|_p^p\right)} \leqslant \lim_{m\to\infty} \frac{m}{m^2 \cdot K^2} = 0 \tag{2.16}$$

The result above shows, that the effect of distance concentration is not induced by having identical distributions in all dimensions. Also, even if the ratio between the variance and expectation in all dimensions is optimal (the expectation is equal to K), the relative distance approaches 0 with the number of dimensions increasing arbitrarily.

2.2.3 *Dependent, Normalised Dimensions*

For clustering applications, having independent components in the data distribution holds no real value since data sets like this cannot hold any interesting group structure. Relaxing the independence requirement makes the statement much more useful, but it also complicates the process of reasoning about the concentration of distances. Let $\vec{X}$ and $\vec{q}$ be again m-dimensional random variables with features

$\vec{X}_k \sim G_k$. Likewise, let $\vec{q}_k \sim H_k$ be random variables that are independent to all $\vec{X}_k$. The difference vector between $\vec{X}$ and $\vec{q}$ is again defined as $\vec{Z} = \vec{q} - \vec{X}$ and is also a random variable. Again, let the individual components of $\vec{Z}$ be scaled in such a way, that $V(|\vec{Z}_k|^p) = 1$ for all $k \leqslant m$ and let there be a $K \in \mathbb{R}_{>0}$ so that $E(|\vec{Z}_k|^p) \geqslant K$.

Only trivial clustering problems can arise if all dimensions of the feature space are samples of independent random distributions. From the practical point of view, this subsection might contain the first useful (mathematical) statement in this chapter.

This time, the random variables modelling the individual dimensions are not independent and therefore, the variance $V(\|\vec{Z}\|_p^p)$ cannot be as easily decomposed as in the last subsections. Instead, the covariances of the individual dimensions have to be taken into account.

As before, the task is to analyse whether or not the relative variance approaches 0 if the dimensionality approaches infinity, see Equation (2.15). First, the variance term in the nominator is expressed in its covariance form, using again the decomposition of the p-norm from Equation (2.12).

$$V\left(\|\vec{Z}\|_p^p\right) = V\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{Cov}(|\vec{Z}_k|^p, |\vec{Z}_l|^p)$$

The number of elements in the double sum increases quadratically with the number of dimensions, so the average covariance $\overline{\sigma}^2_{|\cdot|}$ can be defined as:

$$\overline{\sigma}^2_{|\cdot|} = \frac{1}{m^2}\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{Cov}(|\vec{Z}_k|^p, |\vec{Z}_l|^p) \tag{2.17}$$

The interpretation of the average covariance requires a little thought because negative covariance values can be present. Due to the normalization $V(|\vec{Z}_k|^p) = 1$, the covariance values are equal to the pairwise correlation coefficients:

$$\mathrm{Cor}\left(|\vec{Z}_k|^p, |\vec{Z}_l|^p\right) = \frac{\mathrm{Cov}\left(|\vec{Z}_k|^p, |\vec{Z}_l|^p\right)}{\sqrt{\underbrace{V\left(|\vec{Z}_k|^p\right)}_{=1}} \cdot \sqrt{\underbrace{V\left(|\vec{Z}_l|^p\right)}_{=1}}}$$

$$= \mathrm{Cov}\left(|\vec{Z}_k|^p, |\vec{Z}_l|^p\right)$$

And therefore, the values of the individual covariances are within the interval $[-1, 1]$. Also the average covariance $\overline{\sigma}^2_{|\cdot|}$ is equal to the average correlation of the dimensions. The question is, how can negative covariances (correlations) be interpreted? First it is important to realise, that $|\vec{Z}_k|^p$ are distance values. For the sake of the argument, let the dimensions k and l have a covariance value / correlation coefficient of -1: $\mathrm{Cov}(|\vec{Z}_k|^p, |\vec{Z}_l|^p) = -1$. Then a large distance in dimension k corresponds to a small distance in dimension l and vice versa. This means the sum of both dimensions is constant: $|\vec{Z}_k|^p + |\vec{Z}_l|^p = \mathrm{const} \in \mathbb{R}_{>0}$ and they cannot contribute to the contrast in distance values. As

a consequence, the relative variance of all dimensions reduces the more negative covariances are present. In this sense, the presence of negative covariances *reduces* the relative variance of distances.

Similar to the average covariance, let the average expectation of all dimensions be defined as:

$$\overline{\mu}_{|\cdot|} = \frac{1}{m} \sum_{k=1}^{m} E\left(|\vec{Z}_k|^p\right)$$

The distributive property of the expectation over addition originates from $\int f(x) + g(x)dx = \int f(x)dx + \int g(x)dx$ *and should be written in any basic textbook about probability theory. It is not necessary to consider the German text book [Irle, 2005].*

and since it is possible to exchange the sum and expectation even for dependent random variables [Irle, 2005], it holds:

$$E\left(\|\vec{Z}\|_p^p\right) = E\left(\sum_{k=1}^{m} |\vec{Z}_k|^p\right) = \sum_{k=1}^{m} E\left(|\vec{Z}_k|^p\right) = m \cdot \overline{\mu}_{|\cdot|} \tag{2.18}$$

Due to construction, $\exists K > 0$ such that $E(|\vec{Z}_k|^p) \geqslant K$ and therefore $m \cdot \overline{\mu}_{|\cdot|} \geqslant m \cdot K$.

Inserting Equation (2.18) and (2.17) into Equation (2.14) yields:

$$RV\left(\|\vec{Z}\|_p^p\right) = \frac{V\left(\|\vec{Z}\|_p^p\right)}{E^2\left(\|\vec{Z}\|_p^p\right)} = \frac{m^2\overline{\sigma}_{|\cdot|}^2}{m^2\overline{\mu}_{|\cdot|}^2} = \frac{\overline{\sigma}_{|\cdot|}^2}{\overline{\mu}_{|\cdot|}^2} \leqslant \frac{\overline{\sigma}_{|\cdot|}^2}{K^2} \tag{2.19}$$

And finally, the occurrence of distance concentration can be determined by inserting the last equation into (2.15):

$$\lim_{m\to\infty} RV\left(\|\vec{Z}^{(m)}\|_p^p\right) = \lim_{m\to\infty} \frac{(\overline{\sigma}_{|\cdot|}^{(m)})^2}{(\overline{\mu}_{|\cdot|}^{(m)})^2} \leqslant \lim_{m\to\infty} \frac{(\overline{\sigma}_{|\cdot|}^{(m)})^2}{K^2} \tag{2.20}$$

This is a nice result. The reformulation of the equation provides insight into the structure of the distance concentration that was hidden before. Loosely speaking, the relative variance of distances depends on the amount of pairwise covariance (correlation) of distances and is scaled by the squared average expected distance of the individual features.

Since the first part of Equation (2.20) is achieved with equivalent transformation, it states a sufficient and necessary condition for distance concentration. So it would also fit into the list of equivalent theorems in Section 2.1.2, however is limited to distance functions, induced by (fractional) p-norms, raised to the power of p. Since K is a constant, even a sufficient condition for distance concentration can be stated:

$$\lim_{m\to\infty} (\overline{\sigma}_{|\cdot|}^{(m)})^2 = 0 \quad \Longrightarrow \quad \lim_{m\to\infty} \frac{V(\|\vec{Z}^{(m)}\|_d)}{E^2(\|\vec{Z}^{(m)}\|_d)} = 0 \tag{2.21}$$

Which simply states: if the average pairwise correlation approaches 0, also the relative variance approaches 0 and distance concentration occurs.

This result is consistent with many publications on the subject, see [Aggarwal et al., 2001; François et al., 2007; Hinneburg et al., 2000; Kabán, 2013, 2012]. In all these publications, the respective authors agree that distance concentration becomes an issue if the number of irrelevant features is high. An irrelevant feature implies that its values are independent to the underlying structure of the data set: it holds no relevant information. That is a different way of stating, that the pairwise covariance between the relevant and irrelevant features is low or even 0. In particular, it means that irrelevant features reduce the average pairwise covariance. By expressing irrelevant features as a special case of low pairwise covariance of features (or dimensions), the theory presented above can be regarded as generalization of the notion of irrelevant features.

Hsu and Chen propose in [Hsu and Chen, 2006, 2009], to call a sequence of distance functions $d^{(m)}$ stable, if it does not lead to distance concentration, independently of the sequence of distributions $G^{(m)}$. It might well be, that there is no non-trivial sequence of distance functions that satisfies Hsu and Chen's definition. Hsu and Chens definition is also very different to the approach in this work since the occurrence of distance concentration depends on the correlation within the data distribution. It is therefore not clear how the two different ways of arguing about distance concentration relate to each other.

In my opinion, this approach of combined data distribution and distance function is the real step forwards.

The approach of combining distance function and data distribution is an important difference to the publications, mentioned in the last paragraphs. Within these papers, the data distributions and distance functions are treated as separate entities. This is not the case for the description in this section, as the preconditions are stated for the distributions of distances of data to a query point. In other words, the preconditions are stated for a combination of distance function and data/query distribution. As a result, the theory is more versatile and can be easier expressed than stating preconditions for distance functions and data distributions separately. Considering both entities separately, does not yield much progress because there are always corner cases where the choice of the norm voids general conclusions on the data distribution and vice versa.

This result states that distance concentration of dependent features only is subject to the pairwise interactions of distances of features. It is enough to know the pairwise covariance in order to draw conclusions about the entire high-dimensional distribution. As a consequence, it is possible to limit distance concentration with the easy to calculate pairwise correlation of distances. The drawback is that the result is limited to distance functions that are induced by the family of $\|\cdot\|_p^p$ norms.

Since the restriction to $\|\cdot\|_p^p$ induced distance functions is due to technical reasons, I am sure that a similar result is true for arbitrary distance functions. I was not able to prove that statement though.

2.3 DISTANCE FUNCTIONS

So far, the influence of the choice of the distance function d on the distance concentration effect was not considered. In the last section, the theory is restricted to p-norm induced distance functions because it is possible to decompose these norms, but the influence of the p parameter is not analysed. This is done in the first subsection of this section. The choice of the distance function is in general application dependent and special distance functions to counter the influence of distance concentration might not always be an option. Therefore, it is hard to solve the distance concentration issue on the level of distance functions. It is still useful to consider the choice of the distance function as a way to mitigate the influence of distance concentration. In the next three subsections, strategies to mitigate the influence of distance concentration are discussed.

2.3.1 *The* p *in* $\|\cdot\|_p^p$

The choice of using $\|\cdot\|_p^p$ induced distance functions is well justified since the family of p-norms is one of the most popular in literature and practice. It is discussed in almost all papers on the topic of high-dimensional feature spaces, see for example in [Aggarwal et al., 2001; Beyer et al., 1999; Donoho, 2000; Durrant and Kabán, 2008; Hsu and Chen, 2009; Kabán, 2011, 2012].

The parameter p controls the sensitivity of the distance function to small differences within the individual dimensions. If the parameter p is large (larger than 2) in $\|\vec{x}-\vec{y}\|_p^p$, the total distance between $\vec{x}$ and $\vec{y}$ is dominated by the largest difference in the individual dimensions: $\arg\max_k |\vec{x}_k - \vec{y}_k|$. In the other extreme, $p = 1$ however, all dimensions are equally important. Therefore, low values for p create a stronger contrast in distance values than large values of p.

2.3.2 *Fractional Distance Functions*

For some applications, maybe even most, not all metric properties might be necessary or even desirable for a distance function. Relaxing one or more of the metric properties might open up some freedom to increase the resistance of the distance function w.r.t. the curse of dimensionality. Fractional distance functions, that are distance functions induced by fractional p-norms with $0 < p < 1$ have been studied extensively, for example in [Aggarwal et al., 2001; François et al., 2007; Kabán, 2013]. The advantage of fractional distances is, that they value small differences in many dimensions higher than large differences in just a few dimensions. The effect is a better distance contrast in high-dimensional feature spaces.

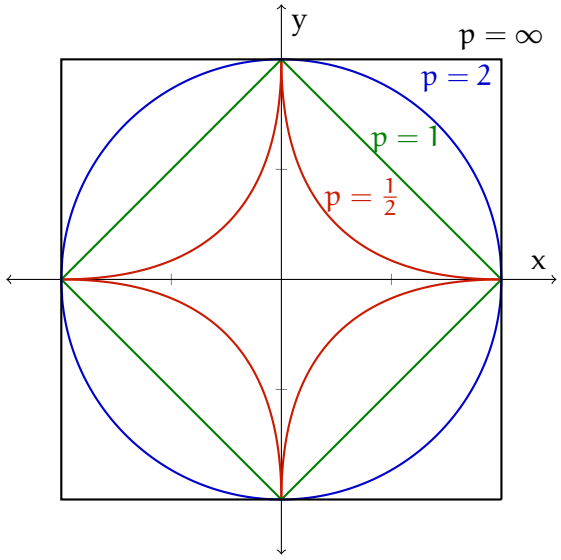


Figure 2.1: Several circles, as defined by (fractional) p-norm induced distance functions. All points on one circle are equally far away from the centre, as defined by the respective distance function.

A circle is defined by the set of all points, that are equally distant to a reference point or centre. The choice of the distance function thereby influence the shape of the circle. In Figure 2.1, several circles of a variety of (fractional) p-norm induced distance functions are presented. Each line represents the set of points in a 2-dimensional plane that are equally far away from the centre. The concave curvature of the fractional distance line suggests, that small points that differ only a little bit in two dimensions are considered equally far away as points that differ only in one dimension. Hence, the contrast in distances is increased for fractional norms if the number of dimensions is high.

Sometimes, shapes that do not look like a circle still are circles. Please consider the definition of a circle.

Fractional distance functions do not satisfy the triangle inequality (see Equation (2.2)). Let $\vec{x} = (0.5, 0)$, $\vec{y} = (0, 0)$ and $\vec{z} = (0, 0.5)$ and d be a fractional norm with $p < 1$. Then the triangle inequality $d(\vec{x}, \vec{z}) \leqslant d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z})$ is violated because

$$d(\vec{x}, \vec{z}) = (\underbrace{0.5^p}_{>0.5} + \underbrace{0.5^p}_{>0.5})^{\frac{1}{p}} > 1$$
$$d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) = (0.5^p)^{\frac{1}{p}} + (0.5^p)^{\frac{1}{p}} = 1$$

So if an algorithm does not rely on that property, fractional norms can be used to reduce the influence of distance concentration. Aggerwal et al. argument in [Aggarwal et al., 2001], that fractional distance functions should behave better than other p-norm induced distance functions in high-dimensional applications. However, François et al. showed in [François et al., 2007] some counter examples where this is not always the case. Also the theory, presented in Section 2.2.3 is valid for any value of p, including fractional distances. Therefore, frac-

tional distances cannot be regarded as a general solution to distance concentration, they can however mitigate the effect.

2.3.3 *Rescaling to Increase Distance Value Contrasts*

Not only fractional p-norm distance functions are a way of countering the effect of distance concentration. François et al. discussed in [François et al., 2007] alternative distance functions that are specifically designed to decrease the influence of distance concentration. Also Jayaram et al. [Jayaram and Klawonn, 2012] discussed if it is possible to prevent distance concentration by design of a distance function. They argue only on the m-dimensional unit cube $[0,1]^m$, and conclude that all distance measures that do not produce infinite distance values on that restricted space are subject to distance concentration. However, they do not provide a rigorous proof for that claim. They also discuss which properties of a distance function can be relaxed to prevent distance concentration.

2.4 TESTS FOR DISTANCE CONCENTRATION

In the last sections, distance concentration is described for a series of dimension dependent probability distributions $G^{(m)}$, $H^{(m)}$ and distance functions $d^{(m)}$. This is of course not a realistic scenario because the distribution of data sets is usually unknown and even worse, the class of distributions might also be unknown. So the question arises, how to characterise distance concentration, given a data set (sample) of unknown distribution and a distance function d.

If the distributions were known, it would not be necessary to analyse the data.

Let $X = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset \mathbb{R}^m$ be a data set and its data objects are independently drawn samples from an unknown data distribution G: $\vec{x}_i \sim G$, $i = 1 \ldots n$. Furthermore, let $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_{\geqslant 0}$ be a distance function, as defined at the beginning of this chapter, Equation (2.1). The question is, how to test the data set X and the distance function d for distance concentration, given that d and X are no sequences?

The choice of the distance function is usually based upon the application and data analysing algorithm that is to be performed. After such a choice is made, a test for distance concentration can provide knowledge if the distance function suffers from distance concentration on this specific data set.

For a sequence of random variables with dimensionality approaching infinity, the Equations (2.7), (2.9), (2.10) and (2.11) are equivalent, but not equivalently well suited for testing a real, given data set of finite dimensionality for a given data mining problem. Equations (2.9) and (2.10) are very sensitive to local noise and Equation (2.11) requires to compute the fraction of distances to each pair of data objects, which is computationally expensive and can only be performed

for small data sets. The relative variance in Equation (2.7) and also (2.15) on the other hand can be computed in $\mathcal{O}(n \cdot m)$ and are only sensitive to extreme outliers (local noise is no problem) which can be detected in advance. In this case, Equation (2.7) is best suited in order to describe the effects of distance concentration on a given pair of data set and distance function.

The value of $\widehat{\mathrm{RV}}\left(D^p_{\vec{q}}(X)\right)$ is larger than 0 for any data set X. A data mining algorithm that is used to perform an analysis on X by using d might require a certain minimal relative variance (contrast in distances) in order to function properly. Let $0 < \varepsilon_{\mathcal{A}} \in \mathbb{R}$ be that value for algorithm $\mathcal{A}$. Some algorithms like Fuzzy c-Means might need a much higher contrast in distances than others (see the following chapters).

It is useful to know if the maximal relative variance in distances ε_{max} is below the specified threshold $\varepsilon_{\mathcal{A}}$, with

$$\varepsilon_{max} = \max_{\vec{q} \in \mathbb{R}^m} \widehat{\mathrm{RV}}\left(D^p_{\vec{q}}(X)\right) \tag{2.22}$$

Or in words: if $\varepsilon_{max} < \varepsilon_{\mathcal{A}}$, the distance function d can be safely regarded as not useful for analysing data set X with algorithm $\mathcal{A}$ and vice versa.

In case the idea of creating a clustering algorithm using the sample relative variance comes up: these images show that this is not a very promising approach. The highest sample relative variance values are usually not achieved near the centre of the classes. The situation might change if weighted mean and variance are used.

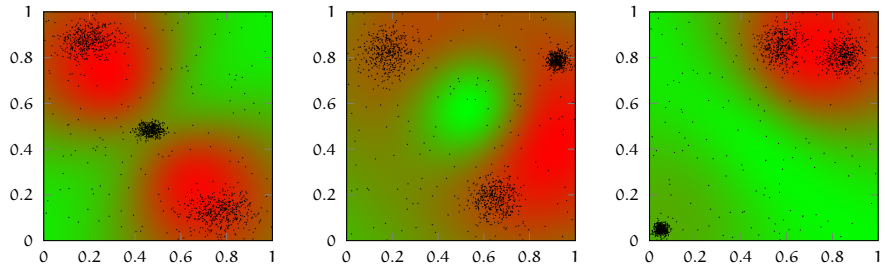


Figure 2.2: Heat map of the sample relative variance of three 2-dimensional data sets, probed by $\vec{q}$. Red represents a high relative variance while green shows a low relative variance.

To prove that distance concentration is not an issue for a given algorithm is a little bit more useful, but also a little more complicated. In this case, a lower bound on the minimal relative variance in distances is necessary. But there is immediately a problem. The value of $\widehat{\mathrm{RV}}\left(D^p_{\vec{q}}(X)\right)$ can become arbitrarily small by moving $\vec{q}$ very far away from all data objects. To get a useful result from a test on the minimal relative variance, it is necessary to specify the set of valid values for $\vec{q}$: $Q \subset \mathbb{R}^m$. That means, if the data analysis algorithm needs a guaranteed minimal variance in distances on Q:

$$\varepsilon_{min} = \min_{\vec{q} \in Q} \widehat{\mathrm{RV}}\left(D^p_{\vec{q}}(X)\right) \tag{2.23}$$

Again in words: if $\varepsilon_{min} > \varepsilon_{\mathcal{A}}$, the combination of distance function d and algorithm $\mathcal{A}$ can be regarded as stable on X.

Calculating either values of the last two equations might be very expensive (in computer hardware or computation time) because the geometrical structure of the data set X determines the result just as much as the distance function d and the data set might be very large. See Figure 2.2 for three different, 2-dimensional examples. The sample relative variance is colour coded for any location $\vec{q}$ in the feature space $[0,1]^2$ with red showing a high empirical relative variance and green a low empirical relative variance respectively. The structure of the data set determines the structure of the relative variance in a non-trivial way. Non-linear optimisation algorithms are feasible here, for example gradient ascending methods. To effectively find the local maxima of relative variance, particle swarm optimization algorithms [Kennedy and Eberhart, 1995] might be a good option because of the high number of local minima and maxima. As they mainly require the gradient as input, it might be helpful to give the derivative of the gradient w.r.t. the query point $\vec{q}$.

$$\widehat{RV}\left(D_{\vec{q}}(X)\right) = \frac{\widehat{V}\left(D_{\vec{q}}(X)\right)}{\widehat{E}^2\left(D_{\vec{q}}(X)\right)} = \frac{\widehat{E}\left(D_{\vec{q}}(X)^2\right)}{\widehat{E}^2\left(D_{\vec{q}}(X)\right)} - 1$$

Deriving this equation w.r.t. the query point $\vec{q}$ yields

$$\nabla_{\vec{q}}\widehat{RV}\left(D_{\vec{q}}(X)\right) = \nabla_{\vec{q}}\frac{\widehat{E}\left(D_{\vec{q}}(X)^2\right)}{\widehat{E}^2\left(D_{\vec{q}}(X)\right)} = n \cdot \nabla_{\vec{q}}\frac{\sum_{j=1}^{n}\left(d(\vec{q},\vec{x}_j)\right)^2}{\left(\sum_{j=1}^{n} d(\vec{q},\vec{x}_j)\right)^2}$$

$$= n\frac{\sum_{j=1}^{n}\nabla_{\vec{q}}\left(d(\vec{q},\vec{x}_j)\right)^2}{S_2^2} - 2n\frac{S_1 \cdot \left(\nabla_{\vec{q}}\sum_{j=1}^{n} d(\vec{q},\vec{x}_j)\right)}{S_2^3}$$

$$= \frac{2n}{S_2^2}\left(\sum_{j=1}^{n} d(\vec{q},\vec{x}_j)\nabla_{\vec{q}}d(\vec{q},\vec{x}_j) - \frac{S_1}{S_2}\cdot\sum_{j=1}^{n}\nabla_{\vec{q}}d(\vec{q},\vec{x}_j)\right)$$

Within my experiments, presented in Chapters 4 and 5, I actually used the gradient optimization method to calculate minimal and maximal levels of distance concentration. The computation process was however very slow because the objective function was too flat and the algorithm did not converge fast enough to be useful.

with $S_1 = \sum_{j=1}^{n}\left(d(\vec{q},\vec{x}_j)\right)^2$ and $S_2 = \sum_{j=1}^{n} d(\vec{q},\vec{x}_j)$. It should be noted, that gradient methods are difficult to apply to evaluate relative variance in high-dimensional feature spaces because the objective function is very flat and the gradient is very weak throughout almost the entire feature space. This implies that the parameter optimization step size must be chosen wisely, which is not easy. Maybe, more complex non-linear optimization algorithms need to be considered to solve this problem but a comprehensive study in this direction is out of scope of this work.

2.5 DISTRIBUTION TESTING

Let again, X be a sample (data set) of unknown distribution and let G be a guess (null-hypothesis), that X is G distributed, called null-distribution. To estimate how plausible this hypothesis is, a statistical test has to be performed. The typical approach [Irle, 2005] is, to compute a test statistic from X and the probability that the test statistic is observed. The null-hypothesis is accepted if the computed probability is higher than a pre-defined significance value. In this case, it is assumed that the data is a sample of G and that the null-hypothesis is indeed correct. That is however no prove that X is indeed G-distributed, it just states that it is very likely to be the case.

Again, any basic book on statistics will do.

A standard approach for calculating a test statistic (see [Irle, 2005]) is to calculate an empirical density function of the data. This approach works well if the number of dimensions is low, even if the class of distributions is unknown. The class of distribution refers to the (parametrised) equation that describes the distribution.

The empirical density function can be visualised using a histogram.

Hinneburg and Keim [Hinneburg and Keim, 1999] argue, it is generally infeasible to generate an empirical density function in high dimensional feature spaces and therefore the standard hypothesis test cannot be performed. To generate an empirical density function, the feature space has to be split into an m-dimensional grid and the data objects populate the cells of this grid. Such a grid necessarily requires at least one split in each dimension and with m dimensions, there are at least 2^m cells. The number of cells increases exponentially with m and to provide a sufficient population of the cells, the number of data objects n needs to increase exponentially w.r.t. m as well. This is usually not the case, hence the statement of Hinneburg and Keim.

My impression of distribution testing is, that it requires a lot of experience as well as try and error. Due to the lack of a direct visualization of high-dimensional data, that is not easy.

The situation changes somewhat, if the class of distributions is already known and sufficiently simple. In this case, only the parameters of the distribution have to be estimated. For example the parameters of a single normal distribution with arbitrary covariance matrix can be simple enough. However, if the data is sampled from a mixture of normal distributions, the situation is already much more complex and the test might not be possible, even though the class of distributions is known.

It might also be possible to do hypothesis testing that is not based on the empirical density function. For example by using easy to calculate statistical properties of the data set like the expectation and the variance. If more external information about the hypothetical distribution function is available, it might be possible to test for pieces of this information. However, that is not a general approach and depends very much on the available additional information.

The inability to test if an assumed distribution actually represents the data in a high-dimensional feature space can have quite severe implications for clustering. Imagine a clustering algorithm is applied

on a data set and the result suggests some partitioning of the data set into smaller subsets. Each of the subsets can be represented by a simple probability distribution and the distribution of the original data set can be constructed by a combination of these individual distributions. If there are no obvious errors in the clustering result and the recombined distribution function cannot be tested, there is not much that can be done to verify the result. Simply trusting the clustering result requires a great level of trust in the algorithm which be ill advised as presented in Section 5.3.2.

2.6 NOISE, OVERLAPPING CLASSES AND OUTLIERS

I use a very generic definition for the term 'noise' that is not shared among all authors in data analysis. Since, there is no general agreed definition of noise, I use one that fits the contents of this work.

Noise is a severe problem for data mining in general and also in clustering. The term 'noise' refers to a wide variety of effects that mask the interesting properties of data. Some aspects of noise are discussed in the framework of high-dimensional feature spaces in the subsections below.

Since the topic of this work is clustering, the most common types of noise in clustering applications and the implications in high-dimensional data sets are discussed in this section. A comprehensive discussion on the topic of noise is out of scope of this work and only a short discussion is presented.

2.6.1 *Imprecise Data*

Please note, that the clustering community does not agree on the definition of the terms below. Depending on the author, various definitions for the same term as well as various terms for the same definition can be found.

Imprecise data, as for example defined in [Borgelt, 2000] in Chapter 2.2 means, that the value of a measurement is not precisely known. This can be expressed for example by stating a validity range or more generally, the knowledge that the true value of a measurement must be within a set of possible values. For example all kinds of physical properties like length, energy levels, brightness, velocities, temperature and many others which are only accurately measured within some errors that cannot be prevented.

Let the true value of some multidimensional property be $\vec{y} \in \mathbb{R}^m$ and the imprecise data object $\vec{X}$ be subject to some random error $\vec{Z}_I$ with $\vec{X} = \vec{y} + \vec{Z}_I$. $\vec{X}$ and $\vec{Z}_I$ are modelled as m-dimensional random variables while $\vec{y}$ is defined as a real vector. The dimensions of the random error vector $\vec{Z}_I$ are modelled to be independent because the measurement error of one feature does not influence the measurement error of another feature. Let d be a distance function for the m-dimensional feature space, than a sample of $\vec{X}$ contains an imprecision of magnitude $z = d(\vec{y}, \vec{X})$. With increasing dimensionality, the magnitude of the error z is also increasing depending on the distance function d.

Let there be several different data generating processes that define the classes of a data set (which is the typical case for clustering). Let furthermore the data of one class have a typical value $\vec{y}_i \in \mathbb{R}^m$ but the measurements of this value are subject to random errors due to imprecise measurements. To ensure a meaningful analysis (clustering), the pairwise distances between the classes $\vec{y}_i$ must increase faster than the magnitude of the error due to imprecision.

In some applications like gene expression data (see Example 1.1.6), this is not the case which causes great trouble in the data mining procedure. If the number of dimensions m is very high and only a subset of the features are relevant for the data mining process, the irrelevant features increase the imprecision within the data while they do not contribute to the pairwise distinction of the classes. Since the individual dimensions of $\vec{Z}_I$ are uncorrelated, this problem is covered by the theoretical discussion in Section 2.2.2.

2.6.2 *Class Distribution*

The natural distribution of a class within a data set can generate similar effects to imprecise data. The uncertainty does not originate from the measuring method, but from the source that is measured. For example in the famous Iris data set [Anderson, 1935; Bache and Lichman, 2013], the sepal length and width does not (only) vary due to an imprecise measuring method, but due to different properties of the plants. The distribution of data is not a 'noise' in the ordinary sense, but it has similar (damaging) effects during data analysis, because it spreads out the data objects and causes classes to overlap. The difference to imprecise measurements is, that the class distribution might have correlated features, while the error vector of imprecision is considered to have independent features which implies that they are uncorrelated.

This is also the reason why data analysis in general and clustering in particular is interesting in the first place. Without (sufficient) variation in the data, clustering would be next to trivial and boring.

Let the idealised representation of a class within a data set be $\vec{y} \in \mathbb{R}^m$ and the observed data object be modelled by an m-dimensional random variable $\vec{X}$ in $\mathbb{R}^m$ be subjective to some random offset from $\vec{y}$: $\vec{Z}_L$ with $\vec{X} = \vec{y} + \vec{Z}_L$. Again, $\vec{X}$ and $\vec{Z}_L$ are modelled as random variables while $\vec{y}$ is defined as a real vector, but this time, the individual features of $\vec{Z}_L$ might be correlated.

The class distribution is no problem if the distances between data objects of the same class are much smaller than the distances of data objects that belong to different classes. This is in particular important for clustering and classification applications. If the classes mix too strongly, it is hard to distinguish between them, hence it is not possible to decide which data object belongs to which class. Due to the lack of independence between the dimensions of $\vec{Z}_L$, this particular source of variation is covered by the theory presented in Section 2.2.3.

Fuzzy clustering methods were developed to model the fact that it might not be possible to recover the assignment of data objects in the overlapping regions, see the first publications on clustering: [Bezdek, 1981; Dunn, 1973; Ruspini, 1969]. The fuzzy model is used to assign data objects partly to multiple clusters. Since overlapping classes are very common, all but one algorithm that is presented in the next chapter utilise fuzzy assignment of data objects.

2.6.3 *Background Noise*

In a data set, background noise manifests itself as data objects, that are present in the data set but are unrelated to the given problem [Dave, 1991]. For example, if the task is to cluster papers by their scientific field and within the set of documents, also non-scientific articles are present. Another example would be to cluster aircraft tracks on an airfield to find the main travel routes while the data base contains tracks that are unusual and cannot be assigned to any of the main routes.

Let the data set X be G distributed. The cumulative distribution function has the form of a sum of two functions $G = (1-\alpha)G_D + \alpha G_B$ where G_D is the cumulative distribution function of the data of interest while G_B is the cumulative distribution function of the background noise and is typically unknown. α specifies the amount of noise that is present in the data and if a sample of G is drawn, a data object can either be sampled by G_D or G_B. Background noise is considered to be damaging and if its proportion is high enough (large value for α), it has to be dealt with explicitly in the analysis algorithm.

Without precise knowledge of background noise, it is not a good idea to remove background noise data objects from the data set because it is usually not known a-priori whether a data object belongs to the background or to the effect that should actually be analysed. This is a huge problem in high-dimensional data analyses, because as stated in 2.5, it is almost impossible to verify the data distribution. This lack of knowledge prevents the development of a comprehensive model of the noise which in turn leads to model noise, see Section 2.6.4.

Background noise is a very common problem for clustering applications. It is explicitly considered within prototype based clustering algorithms by using the noise cluster, first introduced in [Dave, 1991]. Also this type of noise is explicitly considered for 4 of the 10 algorithms, presented in the next chapter.

2.6.4 *Model Noise*

Model noise is related to the two last discussed kinds of noise and is mainly driven by the inability to model the data accurately. Model noise means, that even though the data set is a sample of distribution class G, the model assumes a distribution class $H \neq G$. Again, 'distribution class' refers to the equation describing the distribution function and H and G differ by more than just some parameters. Of course, using the wrong model is a bad idea, but this practice is very common in clustering, if not omnipresent. Many data mining models are based on a mixture of Normal distributions (or ellipsoidal classes in case of clustering) while the data is something completely different. The reason is simply that the true class of data distribution G is unknown and the 'educated best guess' is often a mixture of Normal distributions for H, that need to be parametrised.

The argument from Section 2.5 holds again: it is computationally very difficult to test for the class of distributions with sufficient accuracy because the number of data objects that are required for that rises exponentially with the number of dimensions. Additionally, since it is often known a-priori that the model does not reflect the data, any statistical test might produce a negative result. The question is then, how well a flawed model (of clusters) can reproduce the data structure (of classes).

An approach for clustering is to use adaptive models, that are capable of adjusting the shape of its clusters to complex data distributions. The number of parameters of such an adaptive model can rise very fast with the number of dimensions. For an m-dimensional normal distribution with arbitrary covariance matrix, the number of parameters is $m + m^2$ which is the number of variables for the expectation plus the number of variables in the covariance matrix. If the data consists of c classes, each modelled in the same way, the number of parameters is $c + c \cdot (m + m^2)$ which is the class probability plus the number of classes times the parameters in the distribution function. Such a huge parameter space can lead to unstable results because the number of local optima for the model rises with the number of parameters.

More generally, a clustering algorithm might produce many different sensible results when applied on a high-dimensional data set, but it is intrinsically not possible to decide which of them represents the data best. This problem is known for a long time and triggered the invention of cluster quality indices [Davies and Bouldin, 1979; Dunn, 1973], see also Section 4.3 However, the benchmark presented in this thesis shows that these indices are very unreliable, especially in high-dimensional feature spaces, see Section 5.3.

2.6.5 *Outliers and Missing Values*

Outliers are one of the easiest to detect noise characteristics. However, they pose a rather delicate problem in high-dimensional data sets. Outliers are data objects that usually contain some errors are rarely considered useful data. Where and why the corrupted data values occur is not important, however it is important how often outliers occur. Let the data set $X \subset \mathbb{R}^m$ be a sample of the m-dimensional distribution function G. Let there be $m_{out} < m$ dimensions in which outliers can occur with a probability $p > 0$. The probability that at least one value in a randomly sampled data object contains an outlier value increases with the number of dimensions that can produce outliers m_{out}. For simplicity, let the probability that an outlier occurs in dimension i be at least p_{min}, independently of the feature in which it occurs. If the outlier producing process is independent in the features, the probability p_{out} that at least one dimension is corrupted with an outlier is $p_{out} = 1 - (1 - p_{min})^{m_{out}}$. Similarly, $(1 - p_{min})^{m_{out}}$ is the probability that none of the dimensions that potentially can produce outliers actually produce them. Negating that results in the probability that at least one outlier occurs. For $m_{out} \to \infty$ and a fixed p_{min}, the probability that an outlier occurs approaches 1.

If missing values occur independently of the data generation process, they are similar to outliers because the probabilistic mechanisms work in a similar way. In principle, they are equivalent because any value that is either an outlier or is simply missing does not hold useful information. They are also equivalent if outlier values are deleted and marked as missing or, if missing values are filled with arbitrary values that are unreasonable for the given data. If a constant fraction of the features are likely to contain missing values, the frequency of data objects with missing values increases with the number of dimensions in the same way as outliers. Therefore, both problems can be approached in the same way if outliers can be detected with certainty.

Deleting data objects or features is called feature- or data object marginalization [Cooke et al., 2001]. In a high-dimensional feature space however, filtering out all data objects which contain outliers or missing values can reduce the number of samples in the data set to an unbearable degree. How exactly missing values and outliers are handled is subject to the data mining problem and cannot be discussed in general.

Even if the number of missing values does not reduce the amount of data significantly, it is not always a good option to actually remove them. If the missing values are biased in some way, removing data objects or features might change the cluster structure and vital information might be lost.

A different way of dealing with a (limited) amount of missing values is, to estimate the values that are missing prior to the analysis

step. This strategy is called imputation [Cooke et al., 2001] and has similar drawbacks as marginalization. Instead of loosing information about the cluster structure, it is possible to artificially insert (parts of) a cluster structure that was previously not present in the data. Also it is possible to merge clusters due to the imputation process and to introduce previously not present biases within the data. So this is generally a bad idea, but might be the only available option sometimes.

Both approaches, deleting data objects or features respectively and guessing missing values are only viable if the number of missing values is very small compared to the number of features and data objects. If the number of missing values is too large, the influence of negative effects cannot be neglected which means other strategies have to be used. Sometimes, an existing data analysis algorithm is adapted so it can handle missing values naturally, see for example [Green et al., 2001; Sarkar and Leong, 2001; Wagstaff, 2004], however that might not always be possible.

Clustering high-dimensional data is not very well understood and because missing values is a topic of its own, it is out of scope of this thesis.

2.7 CLUSTERING IN HIGH-DIMENSIONAL SPACES

There are several good points regarding the notion of distance concentration that are important for prototype based clustering. The prototype of a cluster can be regarded as a natural query point $\vec{q}$ because the distribution of distances of the data objects w.r.t. a prototype is relevant for clustering algorithms.

Alternatively, $\vec{q}$ can be regarded as a data object and the distances to all prototypes can be analysed for distance concentration. This last approach is especially relevant for sampling random initial prototype locations which is discussed in Section 3.7.4. This view is crucial for understanding the problems of dimensionality induced clustering problems, see Section 5.1.7.

The most famous distance function discussed in this work is the euclidean distance, which is one of the p-norms. Most prototype based algorithms require the first derivative of the distance function w.r.t. the prototype, in order to find a direction to optimise its parameters. If d has the form of $d(\vec{q}, \vec{x}) = \|\vec{q} - \vec{x}\|_p^p$, its derivative w.r.t. $\vec{q}$ is

$$\begin{aligned}\nabla_{\vec{q}}\, d^p(\vec{q}, \vec{x}) &= \nabla_{\vec{q}} \sum_{k=0}^{m} |\vec{q}_k - \vec{x}_k|^p \\ &= p \cdot (\vec{q} - \vec{x}) \circ |\vec{q} - \vec{x}|^{p-2}\end{aligned}$$

I chose to use a quite unusual notation $\circ$ here because I find vector notation simpler to read and to understand. Since I made vectors obvious with the arrow above, it should be clear that $|\vec{x}|$ cannot be the one-dimensional absolute value of $\vec{x}$.

with $\mathbb{R}^m \circ \mathbb{R}^m \longrightarrow \mathbb{R}^m$ is an element-wise vector multiplication and $|\vec{q} - \vec{x}|^{p-2}$ is the element-wise absolute value:

$$|\vec{q} - \vec{x}|^{p-2} = \begin{pmatrix} |\vec{q}_1 - \vec{x}_1|^{p-2} \\ \vdots \\ |\vec{q}_m - \vec{x}_m|^{p-2} \end{pmatrix}$$

It is very advantageous for the clustering process to set the derivative to be $\vec{0}$ and solve for $\vec{q}$, which is not possible for a general p. The euclidean norm ($p = 2$) is the only p-norm where the absolute term vanishes:

$$\vec{0} = \nabla_{\vec{q}}\, d(\vec{q}, \vec{x}) = \nabla_{\vec{q}} \sum_{k=0}^{m} (\vec{q}_k - \vec{x}_k)^2 = 2 \cdot (\vec{q} - \vec{x})$$

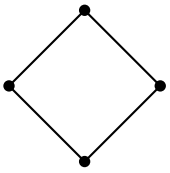
This equation can be solved for $\vec{q}$. As presented in Section 2.2 and concluded from other authors [Aggarwal et al., 2001; Kabán, 2012], the Euclidean distance is especially subject to distance concentration. Not using the Euclidean distance for clustering would mean to abandon the alternating optimization strategy (discussed in the next chapter) for clustering and use a gradient descent algorithm instead. A gradient descend method however is much less stable than alternating optimization, see for example Section 5.1.5. in [Borgelt, 2005]. Using a distance concentration resistant distance function instead of the Euclidean distance is usually not advantageous because of the inherent instability of gradient descend methods.

So the question arises: is prototype based clustering even a viable data mining method in a high-dimensional feature space? Let $\vec{Z}_k = \vec{X} - \vec{q}$ be the difference in the k'th feature (dimension) of the distribution of a data set $\vec{X}$ and a prototype $\vec{q}$ in an m-dimensional feature space. Therefore, $\|\vec{Z}\|_2^2$ is the distribution of distances from the prototype query point $\vec{q}$. If the data set contains a clear cluster structure (i.e. the above discussed noise terms are weak compared to the separation of the classes), and the query vector $\vec{q}$ is close to the centre of one of these classes (i.e. it is a prototype, located within a class), $\|\vec{Z}\|_2^2$ should produce a clear, positive correlation between a relevant portion of pairs of dimensions. That is because data objects of the same class must be close to the query point in multiple dimensions while data objects of a different class are further away. However, it is still a problem to find the location of the classes in the first place. A detailed analysis of this problem is necessary for answering thesis question Q_2 and is partly discussed in Section 5.1.7.

Even though I was trying to find such a model for quite some time, I did not succeed.

A different and interesting approach in this context would be to develop a mathematical test, that takes clustering algorithms as input and provides a minimal value for the concentration of distances that is required for the algorithm to function properly. With the result from Section 2.2, and such a model, it might be possible to estimate

for a given data set and clustering algorithm, if the algorithm can provide a meaningful result on that data set. Something not quite as sophisticated as such a model is provided in Section 5.1.7, which might be regarded as an indication into the right direction.



3

PROTOTYPE BASED CLUSTERING ALGORITHMS

In this chapter, the clustering algorithms relevant for this work and their applications are discussed. All of the algorithms are prototype based (fuzzy) partitional clustering algorithms [Höppner et al., 1999], see Sections 3.1.1 and 3.1.2 for details of these terms. The algorithms were invented with a variety of motivations in mind but are not specifically designed to handle high-dimensional data sets. Hence, it cannot be expected that they produce good results. Surprisingly though, some algorithms perform remarkably well (see Chapter 5). The here presented specific selection of algorithms becomes more clear when considering the arguments presented in Section 5.1.7. In simple terms, they are selected because of the way membership values (see Section 3.1) are calculated and because they have a prototype which lines up nicely with the theory presented in the last chapter.

There are many clustering algorithms, based on a variety of concepts which are too broad for this work. The selection of prototype based (fuzzy) clustering algorithms excludes the algorithms which are mentioned in the following. In high-dimensional feature spaces, hierarchical clustering algorithms (e.g. [Zhang et al., 1996]) have different problems than partitional algorithms. A combined discussion would exceed the scope of this work. Similarly, some partitional clustering concepts are not discussed either. For example, the density based DBScan [Ester et al., 1996] is excluded because the lack of a prototype makes it hard to compare the algorithm with prototype based algorithms. Grid based clustering approaches like DENCLUE [Hinneburg and Keim, 2003] and OptiGrid [Hinneburg and Keim, 1999] are not discussed for the same reason.

I tested DBScan regardless. It produced almost perfect clustering results on D_1 and D_2 but it has terrible performance on D_3 and D_4, see Chapter 4 for explanation on data set families $D_1, \ldots, D_4$. In context of the concentration of distances, I might take a closer look at DBScan in future work.

Many prototype based clustering algorithms cannot be discussed in this work either. Kernel based clustering algorithms [Girolami, 2002; Muller et al., 2001; Schölkopf et al., 1998, 2001; Zhang and Chen, 2003, 2002] are excluded because they are highly adjustable due to their kernel function. Depending on the kernel function, these algorithms might be very good or very bad at clustering high-dimensional data, and the difficulty would be to find a well suited kernel function for a given problem. Considering kernel-based clustering methods would open up many possibilities and might fill an entire thesis on its own, which is why they cannot be considered in this

work. The Gustafson and Kessel clustering algorithm [Gustafson and Kessel, 1978] is not discussed, because the algorithm considers ellipsoidal clusters of arbitrary orientation. It utilises a covariance matrix with m^2 entries. With so many parameters for each cluster, the algorithm can become very unstable in high-dimensional feature spaces. For the same reason, the expectation maximization algorithm, discussed in Section 3.6, is restricted to spherical covariance matrices. Similarly, clustering algorithms that increase the complexity of the FCM algorithm significantly are not discussed, for example the Gath and Geva algorithm [Gath and Geva, 1989] or competitive agglomeration clustering [Frigui and Krishnapuram, 1997].

Experienced readers might skip over the remaining part of this chapter as it is fairly standard. The only non-standard approach is a cool-down on the noise distance, described in Section 3.7.2.

The notation and style of presenting the algorithms is inspired by the habilitation thesis of Christian Borgelt [Borgelt, 2005]. Especially the HCM and FCM algorithms in Sections 3.2 and 3.3, the expectation maximization algorithm in Section 3.6 as well as the evaluation indices in in the next chapter in Sections 4.2 and 4.3 are presented in a similar way as in [Borgelt, 2005].

3.1 MATHEMATICAL FRAMEWORK

All algorithms presented below fall into the same mathematical framework. This framework is presented in the next subsections.

3.1.1 *(Fuzzy) Partitional Clustering Algorithms*

Partitional clustering algorithms decompose a data set into subgroups in 'one go' by either directly producing a final (fuzzy) partitioning or iteratively improving an initial (fuzzy) partitioning. Let X be a data set of n data objects in m dimensions, $X = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset \mathbb{R}^m$. A decomposition of X into c disjoint subsets $X = \bigcup_{i=1}^{c} C_i$, $C_i \cap C_j = \emptyset$ is called a crisp partitioning of X and the subsets $C_1, \ldots, C_c$ are called classes of X. In the fuzzy case, each class is associated with a fuzzy set on X, which means, each data object $\vec{x}_j \in X$ has a value of membership $u_{ij} \in [0, 1]$ that describes its level of assignment to class C_i. The set of classes form a fuzzy partitioning of X if and only if for all data objects $\vec{x}_j \in X$ holds:

$$\sum_{i=1}^{c} u_{ij} = 1 \tag{3.1}$$

The union of fuzzy sets $C_{i \cup k} = C_i \cup C_k$ is then realised by summing the membership values of all data objects: $u_{i \cup k, j} = u_{ij} + u_{kj}$.

For the purpose of clustering, the clustering algorithms divide the data set X into c (fuzzy) clusters (as opposed to classes) which also form a (fuzzy) partition of X. As already stated in the introduction in Section 1.2.1, the term *class* indicates a property of the data set and

the term *cluster* indicates a property of the algorithm, applied on the data set. These two partitionings of X, (classes and clusters) do not need to be identical and the difference between the two partitionings indicate a degree of clustering quality, which is discussed in Sections 4.2 and 4.3.

Please note, that the classes are described to form a *crisp* partitioning of X, it is possible that they form a natural fuzzy partitioning as well. Fuzzy classes can exist in reality, if an object is in a transition phase between two distinct states. For example, in a data set containing weather phenomena, a crisp distinction between the class of 'hot' conditions and 'cold' conditions might not be possible. This notion of reality in terms of fuzzy sets (i.e. fuzzy classes and fuzzy clusters) goes back to the fuzzy set modelling of Zadeh [Zadeh, 1965]. For the clustering algorithm, it is not important whether the data set contains fuzzy classes or crisp classes since the class information is unavailable to the clustering algorithm. Only the location of a data object within the feature space is known. Even if the data set contains natural crisp classes, fuzzy clustering algorithms can be applied because they might have computational advantages over crisp clustering algorithms. After a fuzzy clustering algorithm is applied, the fuzzy partitioning of the clusters can be transformed into a crisp partitioning to match the crisp classes.

The clustering algorithms, discussed in this chapter live all in the same mathematical framework. They are based on an optimization strategy, discussed in the next two subsections. In Subsection 3.1.5, a common visualization scheme is presented, that is applied to all algorithms.

3.1.2 *Prototype-based Clustering Algorithms*

Prototype based clustering implies that each cluster is represented by a prototype. A prototype is similar to a data object, an element of the feature space $\mathbb{R}^m$ and can be seen as a reference point for the cluster or even its centre. This fact is particularly relevant for this work because the prototype forms a natural query point for distance calculations which ties these algorithms to the concentration of distances, see Section 2.1.1.

Since each algorithm utilises c clusters, the set of all prototypes $Y\{\vec{y}_1, \ldots, \vec{y}_c\} \subset \mathbb{R}^m$ of a clustering algorithm contains $c = |Y|$ elements. Mathematically, a prototype $\vec{y}_i \in Y$ is an element of the feature space: $\vec{y}_i \in \mathbb{R}^m$. In many equations in this chapter, the distance between a prototype and a data object $\vec{x}_j \in X$ is addressed. If not

otherwise stated, the value d_{ij} is defined as the Euclidean distance between prototype $\vec{y}_i$ and data object $\vec{x}_j$:

$$d_{ij} = \|\vec{y}_i - \vec{x}_j\| = \sqrt{\sum_{k=1}^{m} (\vec{y}_{i,k} - \vec{x}_{j,k})^2} \tag{3.2}$$

where $\|\cdot\|$ refers to the Euclidean norm. The use of the Euclidean distance a standard distance function is explained two Subsections below, in Subsection 3.1.4. The algorithms, discussed here differ in the interpretation of these distances.

3.1.3 *Objective Functions and Alternating Optimization*

Prototype based clustering algorithms can be understood as an optimization problem. The clustering algorithm is defined by an objective function, symbolised by J, which has the data set X and some variable parameters Θ as input and a real value as ouput: $J : X \times \Theta \to \mathbb{R}$. The parameter set Θ corresponds to a particular (fuzzy) partitioning. A (locally) optimal (either minimal or maximal) value of $J(X, \Theta)$ corresponds to a solution of the clustering problem. In general, it is impractical to compute a globally optimal combination of parameters Θ w.r.t. the objective function J. Finding a global optimum is even for the most simple clustering algorithms NP-hard (see [Aloise et al., 2009; Mahajan et al., 2009]). As the best alternative, the clustering algorithm is supposed to deliver a locally optimal solution of the clustering problem as that is considered to be good enough.

A direct computation of the local minimum is only possible in very simple cases like if the data contains only one cluster. This is usually not the case, instead an iterative process is used to find a local optimum for J. A specific iteration step is indicated by the counter variable $t \in \mathbb{N}$ with $t = 0$ being the initialization. In most equations, t is left out to simplify the notation, only when dealing with variables of different iterations (usually when calculating values for iteration $(t+1)$, t is used as an upper index in round braces. Starting for some initial condition $\Theta^{(0)}$, the next generation of parameters $\Theta^{(t+1)}$ is computed using $\Theta^{(t)}$ until a (locally) optimal solution is found. Initialization is its own topic of research if done thoroughly. A short description and further literature can be found at the end of this chapter in Section 3.7.4. Generally, this iterative optimization process can be done using a gradient optimization approach [Cauchy, 1847], see for example [Snyman, 2005]. To apply the gradient optimization approach, the objective function J must only be differentiable w.r.t. all members of Θ, which makes it very versatile. However, gradient optimization is rather unstable, see for example Section 5.1.5 in [Borgelt, 2005]. A much more robust approach is *alternating optimization* (see

for example [Bezdek and Hathaway, 2002]), but it also imposes more restrictions on J.

Like the gradient optimization approach, alternating optimization is based on finding local extreme values of J. A necessary condition for local extreme values of J are zero values of its first derivative. The additional condition, that the second derivative is not zero is usually implicitly fulfilled and is not explicitly considered. The idea behind alternating optimization is, that Θ is decomposed into two (or more) disjoint subsets: $\Theta = \Theta_1 \cup \Theta_2$, such that the following equations can be solved for any $\theta_1 \in \Theta_1$, and $\theta_2 \in \Theta_2$ respectively:

$$0 = \frac{\partial}{\partial \theta_1} J(X, \Theta) = (\frac{\partial}{\partial \theta_1} J)(X, \theta_1, \Theta_2)$$
$$0 = \frac{\partial}{\partial \theta_2} J(X, \Theta) = (\frac{\partial}{\partial \theta_2} J)(X, \Theta_1, \theta_2)$$

Note, that the differentiation parameter θ_1 is contained in Θ_1, but that the differentiated objective function looses all other members of Θ_1. The equation is then solved for θ_1, which means that θ_1 can be expressed solely as a function of X and Θ_2 where for the purpose of calculating θ_1, the members of Θ_2 are considered to be constants. That way, all values for $\Theta_1^{(t+1)}$ can be computed by using the values of $\Theta_2^{(t)}$. After that, the same procedure is used to calculate the next iteration values for $\Theta_2^{(t+1)}$ depending on the values of $\Theta_1^{(t)}$ (or alternatively $\Theta_1^{(t+1)}$ as presented below in this work). Of course, this way of expressing and solving equations is only possible if the original objective function J permits such calculations. This induces tight restrictions on the structure of J, but at the same time leaves enough freedom to construct different clustering algorithms. The here presented general 2-step procedure can be easily extended to 3 or more steps, see for example the expectation maximization algorithm, presented in Section 3.6.

To my regret, my publications [Winkler et al., 2010b] and indirectly [Winkler et al., 2010c] contain algorithms ignoring these restrictions. From my current perspective, I would rejected these papers as a referee. Nevertheless, the algorithms work as intended on their examples.

3.1.4 *Objective Function based Clustering Algorithms*

As stated above, the first differentiation of the objective function must be solvable for the prototype location, which is only possible for the algorithms below if the distance is measured with the Euclidean norm (as discussed in Section 2.7) or a similar norm like the Mahalanobis distance [De Maesschalck et al., 2000].

Prototype based clustering algorithms essentially optimise the objective function J by optimizing the positions of a set of c prototypes $Y = \{\vec{y}_1, \ldots, \vec{y}_c\} \subset \mathbb{R}^m$. The set of prototypes Y are part of the parameters Θ and can be characterised as data object representatives or similarly as centres of clusters.

A second set of parameters U is used to assign data objects to clusters. For a fuzzy clustering algorithm, the assignment is done in

a fuzzy way (hence the name) which means that a data object $\vec{x}_j$ is assigned to a cluster i with a degree of membership (or membership value) $u_{ij} \in [0,1]$. Equation (3.1) must hold for u_{ij}, meaning that all membership values of a data object must add up to 1. A fuzzy partition matrix $U \in [0,1]^{c \times n}$ is used to cover the assignments of all data objects to all clusters. A set of Lagrange multipliers $\Lambda \in \mathbb{R}^n$ [Bellman, 1956] are used to ensure that Equation (3.1) holds. The membership matrix U and the Lagrange multipliers Λ are part of the parameters Θ of the objective function J.

The membership degrees of different data objects are often independent. Therefore, it is usually not necessary to hold the entire membership matrix in memory during the execution of a (fuzzy) clustering algorithm. See for example the implementation of EDMOAL described in Appendix B.

All algorithms presented in this chapter are optimization problems and the alternating optimization strategy is utilised to find a solution. Θ is divided into suitable subsets, including the membership matrix $U \in [0,1]^{c \cdot n}$, the set of prototype positions $Y \subset \mathbb{R}^m$ and other parameters. For fuzzy clustering methods, usually two subsets of parameters are enough, details are described when presenting the individual algorithms, see Sections 3.2 to 3.6. The general approach is similar for each algorithm. First the membership values U are considered to be fix and the objective function J is optimised by recomputing the prototype locations Y as a function of the non-variable membership values U. In a second step, the prototype locations Y are considered to be non-variable and J is optimised by recomputing the membership vales U as a function of the prototypes.

3.1.5 *Visualization*

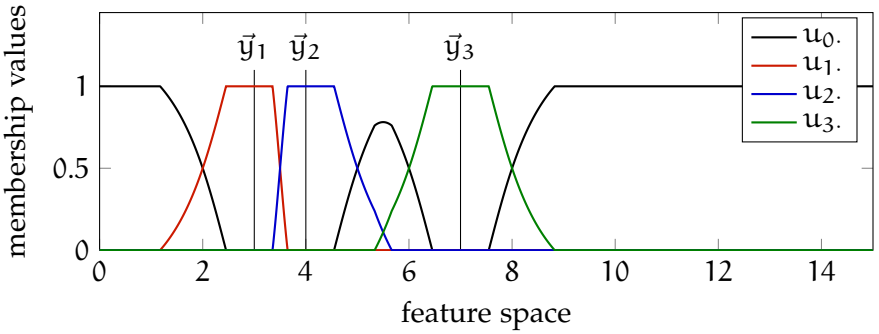


Figure 3.1: Membership value levels of 3 normal clusters $u_{1\cdot}$, $u_{2\cdot}$ and $u_{3\cdot}$ as well as the the noise cluster membership levels $u_{0\cdot}$ in a 1-dimensional feature space with 3 prototypes at $\vec{y}_1 = 3$, $\vec{y}_2 = 4$ and $\vec{y}_3 = 7$.

Understanding the mathematical description is important, but I find it equally important to develop an intuitive understanding of the algorithms which I hope to provide with the visualizations.

Throughout this chapter, the process of clustering is illustrated in two ways. First, the membership values $u_{i\cdot}$ of three clusters $i = 1,2,3$ (and $i = 0$ for the noise cluster, see below) are plotted in a 1-dimensional feature space, see Figure 3.1. These membership value plots give a visual impression of how the relative distances of data objects to the prototypes are translated into membership values. The

coloured lines represent the membership values of the true clusters, the black line (if available) represents the noise cluster. The prototypes are located at $\vec{y}_1 = 3$, $\vec{y}_2 = 4$ and $\vec{y}_3 = 7$ respectively, indicated by vertical lines in the plot. The plot can also be interpreted as a classical representation of the fuzzy sets that cover the entire 1-dimensional feature space.

The second way the algorithms are illustrated is by applying them on a small, 2-dimensional example data set, presented in the left-hand side in Figure 3.2. The data consists of 3, normal distributed classes with varying variance, 1500 data objects each and 500 uniformly distributed noise data objects on the unit square $[0, 1]^2$. The data set is used only for visualization and is not useful to assess the quality of a clustering algorithms. In the right-hand side of Figure 3.2, the result of the PNFCM clustering algorithm (see Section 3.4) is shown. The (mixed) colours of the data objects represent the values of the membership matrix, with black representing the noise cluster. Coloured circles with black borders represent prototypes. The 'tail' of the prototypes show the way they took from their initial position to their final position, each iteration is represented by a small coloured circle with black border. Areas that are filled with colour represent membership values with $u_{..} = 1$ for the corresponding cluster. Height-lines are printed at membership value levels of $u_{..} = 1$, $u_{..} = 0.9$, $u_{..} = 0.8$, $u_{..} = 0.7$, $u_{..} = 0.6$ and $u_{..} = 0.5$.

I hand-picked the initial prototype locations to ensure that all clusters are found and to have consistent colours throughout this chapter.

Technically, the shaded area is printed if the membership values $u_{..} > 0.9999$ instead of $u_{..} = 1$. Otherwise, algorithms that produce crisp membership values asymptotically, like EMGMM (see Section 3.6), would produce undesired floating point error effects.

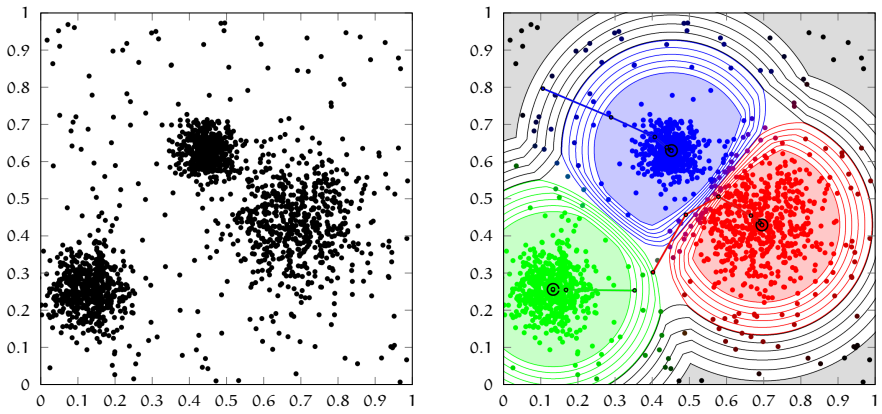


Figure 3.2: A 2-dimensional data set (left) and the same data set, showing the prototypes and membership values (right).

In the last chapter, several noise sources are discussed. Clustering it self is, due to the partitioning aspect, relatively robust w.r.t. imprecise data (see Section 2.6.1) and distribution noise (see 2.6.2) in a quite natural way. Model noise (see Section 2.6.4) is a serious problem because the model that is assumed for the shape of clusters is generally not adopted precisely to the data and the true shape of the classes within the data is approximated with very simple cluster shapes. There are some algorithms that try to mitigate this problem by allowing ellips-

oidal clusters, like Gustafson-Kessel fuzzy clustering [Gustafson and Kessel, 1978] or expectation maximization with arbitrarily shaped Gaussian models. These approaches however are not well suited for high-dimensional feature spaces because they have too many degrees of freedom to work properly. Background noise (see Section 2.6.3) poses another serious problem for clustering algorithms because data objects that do not contribute to the cluster structure of the data might obscure the important structure of the data set. Background noise is approached by introducing an additional noise cluster, which means that data objects are specifically recognised (clustered) as being noise which is represented in black colour in the example images above.

3.2 HARD C-MEANS

Historically, HCM is simply called 'k-means', but I chose to use c *instead of* k *because I use* c *for the number of clusters and* k *as a counter in other occasions.*

The hard c-means algorithm (HCM), better known as 'hard k-means' or simply 'k-means' is the oldest prototype clustering algorithm, first published in its modern form by [MacQueen, 1967]. HCM is a crisp clustering algorithm, which means it assigns data objects uniquely to one cluster. Hence, the membership matrix is restricted to $U \in \{0,1\}^{c\times n}$. The objective function for HCM is defined as:

$$J_{HCM}(X,U,Y) = \sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij} d_{ij}^2 \tag{3.3}$$

The locally optimal parameters for J_{HCM} are found if J_{HCM} is (locally) minimised, therefore the task is to find a combination of U and Y such that no small change in any parameter results in a reduction of $J_{HCM}(X,U,Y)$. With the restriction of Equation (3.1) and limitation to crisp membership values, it follows that only one of $u_{1j},\ldots,u_{cj}$ can be 1, and the others must be 0. It follows with introducing the iteration variable t, that

$$u_{ij}^{(t+1)} = \begin{cases} 1 & \text{if } d_{ij}^{(t)} < d_{kj}^{(t)},\ k = 1\ldots c,\ k \neq j \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

As discussed in Section 3.1.3, it is necessary for minimizing J_{HCM} that its derivative w.r.t. the prototype locations must be 0:

$$\begin{aligned} \vec{0} &\stackrel{!}{=} \nabla_{\vec{y}_i} J_{HCM}(X,U,Y) = \nabla_{\vec{y}_i} \sum_{k=1}^{c}\sum_{j=1}^{n} u_{kj} d_{kj}^2 \\ &= \sum_{j=1}^{n} u_{ij} \nabla_{\vec{y}_i} \|\vec{y}_i - \vec{x}_j\|^2 = 2 \cdot \sum_{j=1}^{n} u_{ij}(\vec{y}_i - \vec{x}_j) \end{aligned}$$

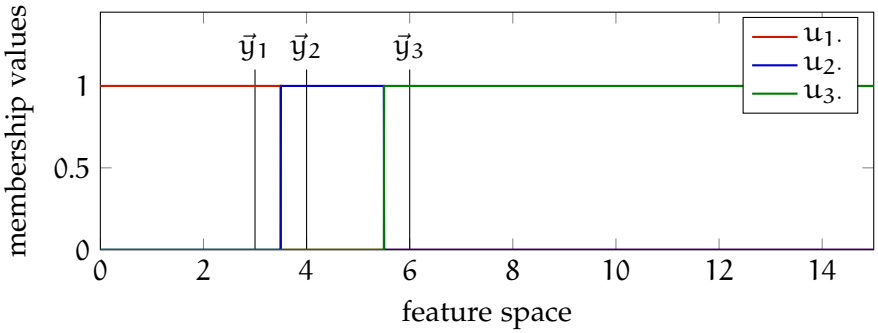


Figure 3.3: Membership value levels for HCM of 3 normal clusters $u_{1\cdot}$, $u_{2\cdot}$ and $u_{3\cdot}$ in a 1-dimensional feature space with 3 prototypes at $\vec{y}_1 = 3$, $\vec{y}_2 = 4$ and $\vec{y}_3 = 7$.

With rearranging, it follows:

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{(t+1)}} \tag{3.5}$$

Starting from an initialization, equation (3.5) is applied until all prototype positions stabilise: $y_i^{(t)} \approx y_i^{(t+1)}$, $\forall i = 1, \ldots, c$. The runtime complexity for each iteration of HCM is $\mathcal{O}(c \cdot n \cdot m)$.

Testing floating point variables for equality for terminating the iteration process is possible but error prone and usually a bad idea. The termination threshold can be implemented by comparing the membership value assignments, which in case of HCM can be implemented as as a list of integer values without significant additional cost, see the implementation within EDMOAL.

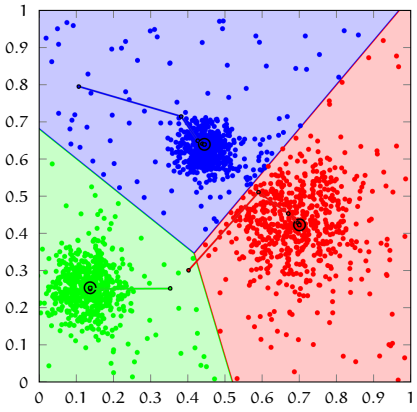


Figure 3.4: The 2-dimensional example data set, clustered with HCM.

The assignment to the closest prototype is visualised in the membership values plot in Figure 3.3. Applied on the example data set produces the result presented in Figure 3.4. Since there are no fuzzy assignments for data objects, the areas of crisply assigned data objects touch each other, which form the encircled areas.

Interestingly, the cluster areas form Voronoi cells.

3.3 FUZZY C-MEANS

The fuzzy c-means algorithm (FCM) is the first fuzzy clustering algorithm out of many discussed here. It was first published in [Dunn, 1973; Ruspini, 1969] and later generalised by Bezdek in [Bezdek, 1981] to its current, final version. In contrast to HCM, FCM is a fuzzy partitional clustering algorithm, hence the partition matrix U is not restricted as for HCM. Also, FCM requires an additional parameter called fuzzifier: $\omega \in \mathbb{R}$, $\omega > 1$ which is discussed in a moment. The objective function of FCM is defined as:

$$J_{FCM}(X, U, Y) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{\omega} d_{ij}^2 \tag{3.6}$$

J_{FCM} differs from J_{HCM} just in the loosened restriction of membership values and the exponential fuzzifier ω. ω is called fuzzifier because it specifies how smooth the membership values change w.r.t. the relative distance among prototypes. A low value of ω means a more crisp clustering algorithm, and indeed for $\omega = 1$ FCM is equivalent to HCM. In contrast, a very high value of ω would make the clustering very fuzzy, and for $\omega \to \infty$, all data objects are shared equally among all clusters.

As J_{HCM}, J_{FCM} has to be minimised in order to find a (locally) optimal solution. This time however, the value for u_{ij} has to be calculated by differentiating the objective function. In order to ensure the restrictions of Equation (3.1), a Lagrange multiplier is added to the objective function.

$$J_{FCM}(X, U, Y, \Lambda) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{\omega} d_{ij}^2 + \sum_{j=1}^{n} \lambda_j \left(1 - \sum_{i=1}^{c} u_{ij}\right) \tag{3.7}$$

with n real parameters $\Lambda = \{\lambda_1, \ldots, \lambda_n\} \subset \mathbb{R}$. J_{FCM} is first derived w.r.t. u_{ij}:

$$\begin{aligned} 0 &\stackrel{!}{=} \frac{\partial}{\partial u_{ij}} J_{FCM}(X, U, Y, \Lambda) \\ &= \frac{\partial}{\partial u_{ij}} \sum_{k=1}^{c} \sum_{l=1}^{n} u_{kl}^{\omega} d_{kl}^2 + \frac{\partial}{\partial u_{ij}} \sum_{l=1}^{n} \lambda_l \left(1 - \sum_{k=1}^{c} u_{kl}\right) \\ &= \omega \cdot u_{ij}^{\omega-1} d_{ij}^2 - \lambda_j \end{aligned}$$

With rearranging, it follows

$$u_{ij} = \left(\frac{\lambda_j}{\omega \cdot d_{ij}^2}\right)^{\frac{1}{\omega-1}} = \left(\frac{\lambda_j}{\omega}\right)^{\frac{1}{\omega-1}} \cdot \left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{\omega-1}} \tag{3.8}$$

With regard to Equation (3.1), it follows

$$1 = \sum_{k=1}^{c} u_{kj} = \sum_{k=1}^{c} \left(\frac{\lambda_j}{\omega}\right)^{\frac{1}{\omega-1}} \cdot \left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}$$
$$= \left(\frac{\lambda_j}{\omega}\right)^{\frac{1}{\omega-1}} \cdot \sum_{k=1}^{c} \left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}$$

and therefore

$$\left(\frac{\lambda_j}{\omega}\right)^{1-\omega} = \frac{1}{\sum_{k=1}^{c} \left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}}$$

Inserting this equation into (3.8) and applying the iteration variable yields

$$u_{ij}^{(t+1)} = \frac{\left(\frac{1}{\left(d_{ij}^{(t)}\right)^2}\right)^{\frac{1}{\omega-1}}}{\sum_{k=1}^{c} \left(\frac{1}{\left(d_{kj}^{(t)}\right)^2}\right)^{\frac{1}{\omega-1}}} \tag{3.9}$$

Equations like this can be simplified by hiding the fractional distances in their respective exponents. However, they are easier to understand the way they are written here, even though it takes more space on paper.

In Figure 3.5, the membership values plot for FCM is presented (see

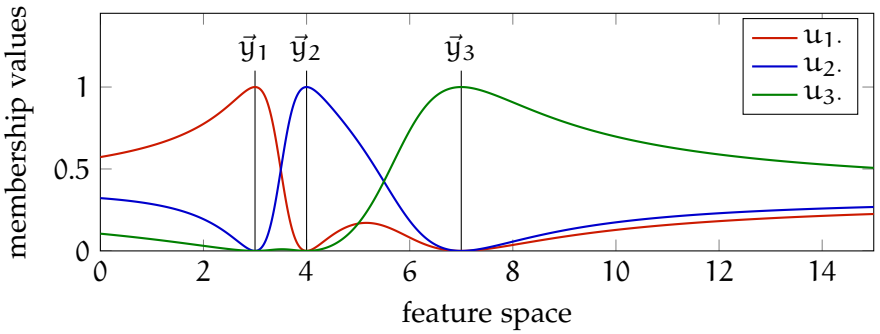


Figure 3.5: Membership value levels u for FCM with fuzzifier $\omega = 2$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

Section 3.1.5 for the description). Data objects near just one of the prototypes y_1, y_2 and y_3 are assigned with higher membership to their respective prototype than data objects that have similar distances to multiple prototypes. Updating the prototypes for FCM is very similar to HCM:

$$\vec{0} \stackrel{!}{=} \nabla_{\vec{y}_i} J_{FCM}(X, U, Y, \Lambda) = \nabla_{\vec{y}_i} \sum_{k=1}^{c} \sum_{j=1}^{n} u_{kj}^{\omega} d_{kj}^2$$
$$= \sum_{j=1}^{n} u_{ij}^{\omega} \nabla_{\vec{y}_i} \|\vec{y}_i - \vec{x}_j\|^2 = 2 \cdot \sum_{j=1}^{n} u_{ij}^{\omega} (\vec{y}_i - \vec{x}_j)$$

With rearranging and iteration variable, the prototype update equation is derived as:

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^{\omega} \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^{\omega}} \tag{3.10}$$

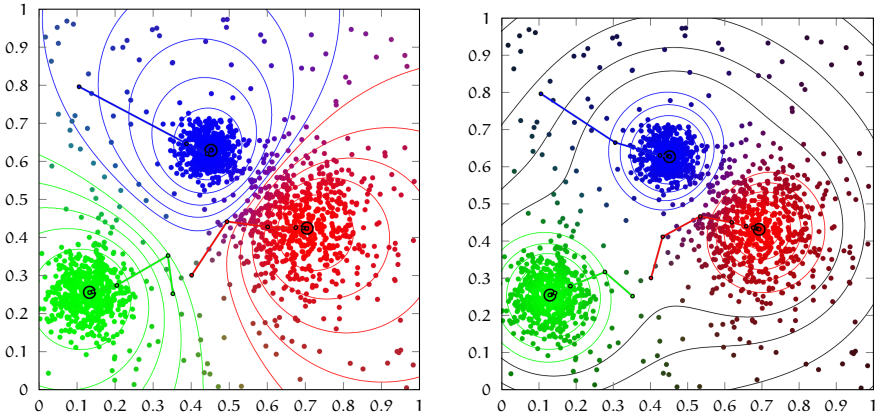


Figure 3.6: The 2-dimensional example data set, clustered with FCM (left) and NFCM (right) with parameters $\omega = 2$ and $d_{noise} = 0.2$.

The iteration process is stopped if a maximum number of iterations have been reached or if the maximal movement distance of prototypes drops below a predefined threshold $0 < \varepsilon \in \mathbb{R}$, see Section 3.7.5 for details. The runtime complexity for each iteration of FCM is the same as for HCM: $\mathcal{O}(c \cdot n \cdot m)$. The fuzzy assignment of data objects in between two prototypes is visualised on the left-hand side in Figure 3.6.

When implementing FCM, NFCM and alike, I found it best to treat the noise cluster as a different entity. When programming in java, c or similar, it is incredibly confusing to count sometimes from 1 to n and sometimes from 0 to n (or from 0 to $n-1$ vs. 0 to n). I did it this way in the beginning because it represented the mathematical description but changed the code later because the corner cases became too much of a problem.

3.3.1 *FCM with Additional Noise Cluster*

FCM can be very sensitive to noise (outliers) because the model of prototypes does not apply well to single data objects far away from 'good' data. The effect can be observed in Figure 3.5. On the outer edges of the plot, the membership levels of all prototypes approach $\frac{1}{c}$, which means that all prototypes are pulled towards data objects, that are far away. Noise FCM (NFCM) [Dave, 1991] adds one additional cluster (the noise cluster) to FCM which has a constant distance to all data objects. The noise cluster is not represented as a prototype, but it still adds factors to the fuzzy partitioning of a data object. If a data object is very far away from all prototypes, it gets a high membership value to the noise cluster and low membership values to regular clusters.

Formally, the equations for FCM changes to:

$$J_{NFCM}(X, U, Y) = \sum_{i=0}^{c} \sum_{j=1}^{n} u_{ij}^{\omega} d_{ij}^2 \tag{3.11}$$

with cluster index 0 denoting the noise cluster and $0 \leqslant d_{0j} = d_{noise} \in \mathbb{R}$ being the constant distance to the noise cluster. Deriving the update equations for NFCM is analogous to FCM, only the indices are different. For $i = 0, \ldots, c$ and $j = 1, \ldots, n$:

$$u_{ij}^{(t+1)} = \frac{\left(\frac{1}{\left(d_{ij}^{(t)} \right)^2} \right)^{\frac{1}{\omega - 1}}}{\sum_{k=0}^{c} \left(\frac{1}{\left(d_{kj}^{(t)} \right)^2} \right)^{\frac{1}{\omega - 1}}} \tag{3.12}$$

and for $i = 1, \ldots, c$:

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)} \right)^{\omega} \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)} \right)^{\omega}} \tag{3.13}$$

The effect of the additional noise cluster is presented in Figure 3.7. Contrary to FCM without a noise cluster, the membership values of data objects far away from all prototypes are almost exclusively assigned to the noise cluster, hence they do not influence the prototype locations significantly.

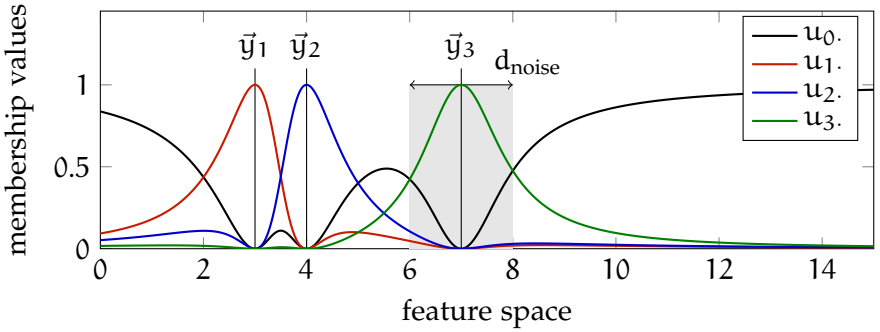


Figure 3.7: Membership value levels u for NFCM with fuzzifier $\omega = 2$ and noise distance $d_{noise} = 1$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

A visualization of NFCM is presented on the right-hand side in Figure 3.6. The noise cluster clearly restricts the outreach of the membership levels of the other clusters such that they are confined to the

spread of the relevant data. This not only prevents outliers of gaining too much influence on the prototypes, it also creates more localised clusters, that fit very well for normal distributed data. The down side is, that it can be hard to find a suitable noise distance d_{noise} for a given problem. If the noise distance is set very low compared to the distances of normal prototypes to data objects, formally $d_{noise} << d_{ij}$, an effective clustering is prevented. Also, for $d_{noise} \to \infty$, the noise cluster becomes insignificant and thus, NFCM becomes identical to FCM. For a more detailed discussion on the choice of the noise distance, see Section 3.7.2.

3.3.2 *(N)FCM with dimension dependent fuzzifier*

As presented in [Winkler et al., 2011a] and in Section 5.1.2, neither FCM nor NFCM are particularly well suited for high-dimensional feature spaces. We have shown in the mentioned paper that this problem with FCM can be mitigated using the fuzzifier ω. The algorithms FCM_m and $NFCM_m$ are formally identical to FCM and NFCM respectively, but their fuzzifier is not free to choose for the user but it is set to $\omega = 1 + \frac{1}{m}$. This means, the algorithm becomes more crisp for higher dimensional feature spaces, approaching HCM for a very high dimensionality. The reason to use the fuzzifier in this way is explained in Section 5.1.2.

In our original paper, we proposed to use $\omega = 1 + \frac{2}{m}$ because it works well and it includes the special case of standard FCM for 2 dimensions. However, during the extensive testing for this work, I found that $\omega = 1 + \frac{1}{m}$ is slightly more effective.

For clarity, the FCM version with the fuzzifier $\omega = 2$ is referred to as FCM_2 in future. Please note, that FCM_m utilizes a more crisp fuzzifier for $m = 2$: $\omega = 1 + \frac{1}{2} = 1.5$ for FCM_m and $\omega = 2$ for FCM_2.

3.4 FUZZY C-MEANS WITH POLYNOMIAL FUZZIFIER FUNCTION

Fuzzy c-means with polynomial fuzzifier function (PFCM) [Klawonn and Höppner, 2003a], replaces the exponential fuzzifier function in FCM with a polynomial function of grade 2. This clustering algorithm is motivated by addressing some of the weaknesses of FCM. Observe again the left-hand side of Figure 3.6. The small space between the 0.5 height-lines of each pair of clusters is caused by the respective third cluster which sits further away. The same effect can be also observed in Figure 3.5 as the red clusters membership levels create a local maximum in-between the blue and green cluster. Logically, there is no reason that the third, further away cluster should have a local maximum of membership values at this location. To avoid this, PFCM is designed to create crisp membership levels whenever data objects can be clearly assigned to a certain cluster or if data objects can be clearly *not* assigned to a cluster.

Actually, the name for this algorithm was invented by me because it was just named 'new approach' in the original publication.

The objective function J_{PFCM} is defined as

$$J_{PFCM}(X, U, Y) = \sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij}) d_{ij}^2 \tag{3.14}$$

with $h(u) = \alpha u^2 + (1-\alpha)u$ and $\alpha \in [0,1]$. The parameter α functions as a linear factor to mix FCM with $\omega = 2$ and HCM. When applying PFCM, α is not very well suited as parameter because it is very unintuitive w.r.t. the behaviour of the algorithm. $h(u)$ can be reformulated with $\alpha = \frac{1-\beta}{1+\beta}$, $\beta \in [0,1]$ or $\beta = \frac{1-\alpha}{1+\alpha}$ which produces the fuzzifier function

$$h(u) = \left(\frac{1-\beta}{1+\beta} u^2 + \frac{2\beta}{1+\beta} u \right) \tag{3.15}$$

The effect of the membership function and its parameter β can be best understood with a very simple example. Consider the 2 prototypes $\vec{y}_2$ and $\vec{y}_3$ in a 1-dimensional environment and an imaginary data object $\vec{x}_0$ in between $\vec{y}_2$ and $\vec{y}_3$ that has the distances d_{20} and d_{30} to the prototypes, illustrated in Figure 3.8. The parameter β specifies the location of $\vec{x}_0$ in the feature space at which the membership values change from crisp to fuzzy or vice versa. This statement is only true if there is no other prototype nearby that produces non-zero membership values. The membership value is crisp with $u_{20} = 1 \Leftrightarrow \frac{d_{20}^2}{d_{30}^2} \leqslant \beta$ or $u_{20} = 0 \Leftrightarrow \frac{d_{20}^2}{d_{30}^2} \geqslant \frac{1}{\beta}$ or fuzzy with $u_{20} \in (0,1) \Leftrightarrow \beta < \frac{d_{20}^2}{d_{30}^2} < \frac{1}{\beta}$.

PFCM produces a (fuzzy) partition of the data set and therefore, Equation (3.1) must hold which is again enforced by using the Lagrange extension. The full membership function for optimizing its parameters is:

$$\begin{aligned} J_{PFCM}(X, U, Y, \Lambda) = &\sum_{i=1}^{c} \sum_{j=1}^{n} \left(\frac{1-\beta}{1+\beta} u_{ij}^2 + \frac{2\beta}{1+\beta} u \right) d_{ij}^2 \\ &+ \sum_{j=1}^{n} \lambda_j \left(1 - \sum_{i=1}^{c} u_{ij} \right) \end{aligned} \tag{3.16}$$

which has to be minimised. As for FCM, deriving J_{PFCM} w.r.t. the membership values yields:

$$0 \stackrel{!}{=} \frac{\partial}{\partial u_{ij}} J_{PFCM}(X, U, Y, \Lambda) = \left(2\frac{1-\beta}{1+\beta} u_{ij} + \frac{2\beta}{1+\beta} \right) d_{ij}^2 - \lambda_j$$

and with rearranging follows

$$u_{ij} = \frac{1}{1-\beta} \left(\frac{1}{2} \frac{(1+\beta)\lambda_j}{d_{ij}^2} - \beta \right) \tag{3.17}$$

Due to the nature of the membership update equation of FCM, it is not necessary to create a rule for $u_{ij} \in [0,1]$. That changes with

PFCM where it must be ensured explicitly. It is enough however, to ensure $u_{ij} \geqslant 0$ since together with the Lagrange extension implies $u_{ij} \leqslant 1$. Formally, this could be done using the Karush-Kuhn-Tucker conditions [Karush, 1939; Kuhn and Tucker, 1951], which are extensions of the Lagrange multipliers for constrains that are formulated as inequalities instead of equalities.

The resulting objective function is more difficult to process with alternating optimization because the inequality of $u_{ij} \geqslant 0$ must be ensured explicitly during updating the membership values. A trick is used to achieve that. Suppose it is known for which clusters Equation (3.17) produces positive values w.r.t. x_j and that the number of such membership values is $\check{c}_j \in \mathbb{N}$, $\check{c}_j \leqslant c$. Explicitly set all membership values to 0 for which Equation (3.17) would produce negative values, then

$$\begin{aligned} 1 = \sum_{\substack{k=1 \\ u_{kj} \geqslant 0}}^{c} u_{kj} &= \sum_{\substack{k=1 \\ u_{kj} \geqslant 0}}^{c} \frac{1}{1-\beta} \left(\frac{1}{2} \frac{(1+\beta)\lambda_j}{d_{kj}^2} - \beta \right) \\ &= \frac{1+\beta}{1-\beta} \frac{1}{2} \lambda_j \sum_{\substack{k=1 \\ u_{kj} \geqslant 0}}^{c} \frac{1}{d_{kj}^2} - \frac{\check{c}_j \cdot \beta}{1-\beta} \\ \Rightarrow \lambda_j &= 2 \cdot \frac{1}{1+\beta} \cdot \frac{1 + (\check{c}_j - 1)\beta}{\sum_{\substack{k=1 \\ u_{kj} \geqslant 0}}^{c} \frac{1}{d_{kj}^2}} \end{aligned}$$

Inserted into equation (3.17) leads to the update equation for membership values provided it is known which membership values are positive:

$$u_{ij} = \frac{1}{1-\beta} \left(\frac{1 + (\check{c}_j - 1)\beta}{\sum_{\substack{k=1 \\ u_{kj} \geqslant 0}}^{c} \frac{d_{ij}^2}{d_{kj}^2}} - \beta \right) \tag{3.18}$$

The equation seems to contain circular reasoning because the knowledge of the value of u_{ij} requires the knowledge of its sign. It is however possible to determine whether or not a membership value will be positive without calculating its value. For that, it is necessary to sort the distances $d_{1j}, \ldots, d_{cj}$ regarding one data object x_j in ascending order. Let φ be a permutation so that $\forall 1 \leqslant i < k \leqslant c$: $d_{\varphi(i)j} \leqslant d_{\varphi(k)j}$. From Equation (3.18) can also be deduced that for any two distances $d_{ij} \leqslant d_{kj} \Rightarrow u_{ij} \geqslant u_{kj}$ and therefore holds $\forall 1 \leqslant i < k \leqslant c$

$u_{\varphi(i)j} \geqslant u_{\varphi(k)j}$. Let $\tilde{c} \in \mathbb{N}$, $\tilde{c} \leqslant c$, then it is sufficient to know the smallest index $\varphi(\tilde{c})$ for witch equation (3.18) yields a negative value.

$$0 \leqslant \frac{1}{1-\beta}\left(\frac{1+(\tilde{c}_j-1)\beta}{\sum_{k=1}^{\tilde{c}} \frac{d^2_{\varphi(\tilde{c})j}}{d^2_{\varphi(k)j}}} - \beta\right) \quad \Leftrightarrow \quad \sum_{k=1}^{\tilde{c}} \frac{d^2_{\varphi(\tilde{c})j}}{d^2_{\varphi(k)j}} - \tilde{c} \leqslant \frac{1}{\beta} - 1$$

The number $\check{c}_j$ is calculated by gradually increasing $\tilde{c}$ from 1 until the test fails the first time. Formally:

$$\check{c}_j = \max\left\{\tilde{c} \in \mathbb{N} \;\middle|\; \tilde{c} \leqslant c, \sum_{k=1}^{\tilde{c}} \frac{d^2_{\varphi(\tilde{c})j}}{d^2_{\varphi(k)j}} \leqslant \frac{1}{\beta} + \tilde{c} - 1\right\} \tag{3.19}$$

Calculating $\check{c}_j$ does not come with a significant increase in computation cost. See the implementation in EDMOAL.

Calculating $\check{c}_j$ requires to sort the distances of the prototypes w.r.t. each data object which increases the runtime complexity for each iteration by a factor of $\ln(c)$.

Finally, the update equation for the membership values can be declared:

$$u_{ij}^{(t+1)} = \begin{cases} \dfrac{1}{1-\beta}\left(\dfrac{1+(\check{c}_j^{(t)}-1)\beta}{\sum_{k=1}^{\check{c}_j^{(t)}} \frac{\left(d_{ij}^{(t)}\right)^2}{\left(d_{\varphi(k)j}^{(t)}\right)^2}} - \beta\right) & \text{if } \varphi(i) \leqslant \check{c}_j^{(t)} \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

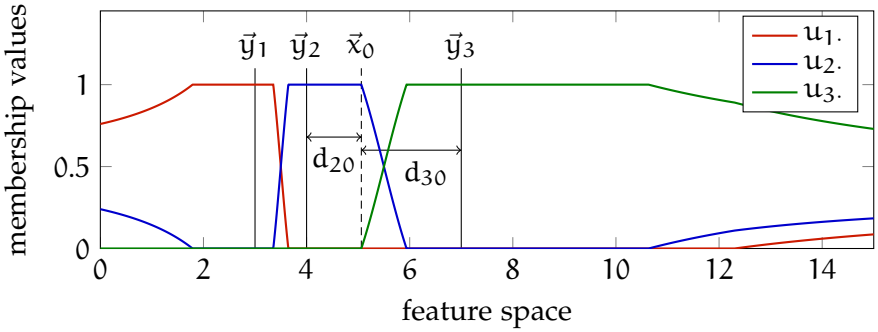


Figure 3.8: Membership value levels u for PFCM with $\beta = 0.3$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

In Figure 3.8, the result of the effect of the membership function on the membership values can be observed. In the vicinity of prototypes, the membership values become crisp, which can have advantages on the clustering process because it hides the data objects near one prototype from all other prototypes. The result is a better separation and

spread as well as a reduced chance that two prototypes share the same class and divide it into two clusters.

Updating the prototype positions for PFCM is done analogously to FCM.

$$\vec{0} \stackrel{!}{=} \nabla_{\vec{y}_i} J_{\text{PFCM}}(X, U, Y, \Lambda) = \nabla_{\vec{y}_i} \sum_{k=1}^{c} \sum_{j=1}^{n} h(u_{kj}) d_{kj}^2$$
$$= \sum_{j=1}^{n} h(u_{ij}) \nabla_{\vec{y}_i} \|\vec{y}_i - \vec{x}_j\|^2 = 2 \cdot \sum_{j=1}^{n} h(u_{ij})(\vec{y}_i - \vec{x}_j)$$

which results into the update equation

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \vec{x}_j}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)} \tag{3.21}$$

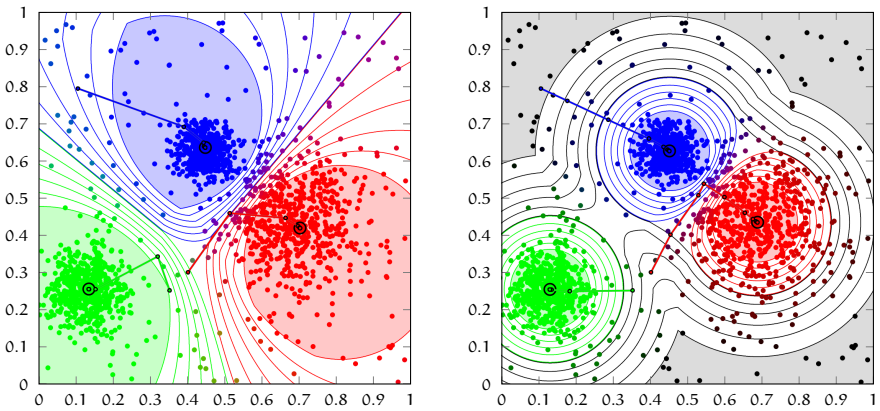


Figure 3.9: The 2-dimensional example data set, clustered with PFCM (left) and PNFCM (right) with parameters $\beta = 0.3$ and noise distance $d_{\text{noise}} = 0.2$.

Analogously to FCM, the iteration of PFCM is stopped if a maximum number of iterations have been reached or if the maximal movement distance of prototypes drops below a predefined threshold $0 < \varepsilon \in \mathbb{R}$. The runtime complexity for each iteration of PFCM is not as good as FCM because it requires a sorting of prototypes according to the distances for each data object. Therefore, it is in $\mathcal{O}(c \ln(c) \cdot n \cdot m)$.

A visual representation of the applied algorithm is presented in Figure 3.9. The areas with membership level 1 are represented by coloured areas. Contrary to FCM, the data objects near a prototype are crisply assigned which hides these data objects from the other clusters. This 'hiding' mechanism works also if a data object is not crisply assigned to one cluster, but is shared by at least two. It still may have membership levels of 0 to any cluster where the prototype

is further away, as for example in between the red and blue cluster in Figure 3.9. Unlike for FCM, the height-lines for membership levels of 0.5 of the red and blue cluster touch each other, which indicates that at no place in between, membership levels above 0 are assigned to the green cluster.

A similar algorithm to PFCM was proposed in [Klawonn, 2004; Klawonn and Höppner, 2003b] and utilises exponential fuzzifier functions, I call it therefore FCM with exponential fuzzifier functions (EFCM):

$$h(u) = \frac{1}{e^{\alpha} - 1}(e^{\alpha u} - 1)$$

EFCM has almost the same properties as PFCM, it also requires a sorting of prototypes w.r.t. their distance to a data object and it produces areas of crisp membership values similar to PFCM. Due to its similarity, it is not further discussed in this work.

3.4.1 *PFCM with Additional Noise Cluster*

Similar to FCM, PFCM can be equipped with a noise cluster, transforming it into PNFCM. The reason to do that is the same for FCM: data objects that are located far away from all prototypes receive membership degrees of roughly $\frac{1}{c}$ for each cluster, which may have a negative effect on the clustering quality. The noise distance d_{noise} is treated like a normal distance to a prototype when sorting the clusters w.r.t. their distance to a data object. The noise cluster is also included in calculating $\check{c}_j$ which means, that a data object $\vec{x}_j$ can have a crisp assignment to the noise cluster if $d_{\text{noise}} < d_{ij}$ and $\check{c}_j = 1$. Again, the equations for PNFCM change only in their indices, with cluster index 0 referring to the noise cluster.

$$J_{\text{PNFCM}}(X, U, Y) = \sum_{i=0}^{c} \sum_{j=1}^{n} h(u_{ij}) d_{ij}^2 \tag{3.22}$$

For $i = 0, \ldots, c$ and $j = 1, \ldots, n$:

$$u_{ij}^{(t+1)} = \begin{cases} \dfrac{1}{1-\beta} \left(\dfrac{1 + (\check{c}_j^{(t)} - 1)\beta}{\sum\limits_{k=0}^{\check{c}_j^{(t)}} \dfrac{\left(d_{ij}^{(t)}\right)^2}{\left(d_{\varphi(k)j}^{(t)}\right)^2}} - \beta \right) & \text{if } \varphi(i) \leqslant \check{c}_j^{(t)} \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

and for $i = 1, \ldots, c$

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \vec{x}_j}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)} \tag{3.24}$$

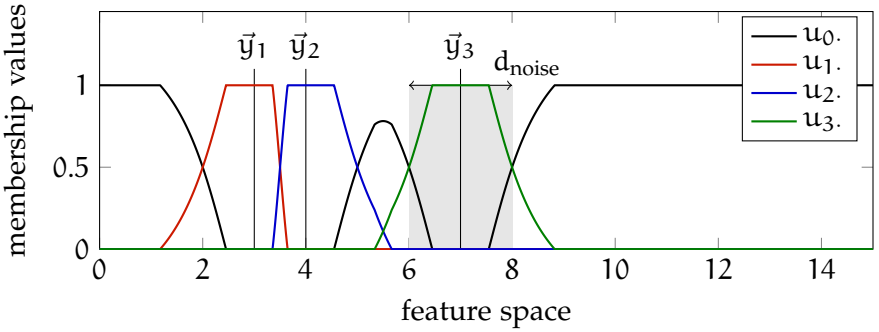


Figure 3.10: Membership value levels u for PNFCM with $\beta = 0.3$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

The effect of the noise cluster on the membership values can be observed in Figure 3.10 as a contrast to Figure 3.8. The prototypes have a reduced range of influence and the noise cluster effectively hides outliers from the prototypes. The reduced range also changes the shape of the crisply assigned area to be more spherical, as can be observed at the right-hand side of Figure 3.9. Some data objects might be crisply assigned to the noise cluster, indicated by the grey coloured area. The clusters form relatively separated islands without influencing one another significantly if they are sufficiently far apart. Also the way the prototypes take from their initial position to their final position are different w.r.t. PFCM. The green and blue prototypes for example are hardly influenced, if at all, by data objects other than the data objects of the clusters they are covering at the end of the clustering process.

3.5 REWARDING CRISP MEMBERSHIPS FUZZY C-MEANS

The rewarding crisp memberships fuzzy c-means clustering algorithm (RCFCM), published by [Höppner and Klawonn, 2001; Höppner and Klawonn, 2003] provides a different approach to acquire a clustering algorithm similar to HCM. The difference is again modulated by altering the objective function:

$$J_{RCFCM}(X, U, Y) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^2 d_{ij}^2 - \sum_{j=1}^{n} a_j \sum_{i=1}^{c} (u_{ij} - \frac{1}{2})^2 \tag{3.25}$$

The parameters $\{a_1, \ldots, a_n\}$ are not directly chosen by the user, like for example ω for FCM, they will be discussed in a moment. Due to the second term, membership values close to 1 or 0 are more favourable than membership values close to $\frac{1}{2}$. Again, the objective function is minimised using the alternating optimization approach. The Lagrange multiplier is applied to satisfy the partitioning condition in equation (3.1) which yields the objective function:

$$J_{RCFCM}(X, U, Y, \Lambda) = \sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^2 d_{ij}^2 - \sum_{j=1}^{n} a_j \sum_{i=1}^{c} (u_{ij} - \frac{1}{2})^2 + \sum_{j=1}^{n} \lambda_j \left(1 - \sum_{i=1}^{c} u_{ij}\right) \tag{3.26}$$

Calculating the update equations for RCFCM is similar to FCM. The necessary condition for a minimum of J_{RCFCM} must be satisfied, which is again that the derivative must be 0.

$$\begin{aligned} 0 &\stackrel{!}{=} \frac{\partial}{\partial u_{ij}} J_{RCFCM}(X, U, Y, \Lambda) = 2u_{ij} d_{ij}^2 - 2a_j \left(u_{ij} - \frac{1}{2}\right) - \lambda_j \\ &= 2u_{ij}\left(d_{ij}^2 - a_j\right) + a_j - \lambda_j \\ \Rightarrow u_{ij} &= \frac{\lambda_j - a_j}{2\left(d_{ij}^2 - a_j\right)} \end{aligned}$$

Again, with Equation (3.1) follows

$$\begin{aligned} 1 &= \sum_{k=1}^{c} u_{kj} = \sum_{k=1}^{c} \frac{\lambda_j - a_j}{2\left(d_{kj}^2 - a_j\right)} \\ \Rightarrow \lambda_j - a_j &= 2\frac{1}{\sum_{k=1}^{c} \frac{1}{2\left(d_{kj}^2 - a_j\right)}} \end{aligned}$$

With inserting this into the equation above and applying the iteration constant yields

$$u_{ij}^{(t+1)} = \frac{\frac{1}{\left(d_{ij}^{(t)}\right)^2 - a_j^{(t)}}}{\sum_{k=1}^{c} \frac{1}{\left(d_{kj}^{(t)}\right)^2 - a_j^{(t)}}} \tag{3.27}$$

From this equation it is intuitively clear, that a_j should not be larger than d_{kj}^2, $k = 1 \ldots c$ because that would yield negative membership values. Also, the argument was to encourage crisp values and negative values of a_j would have the opposite effect. Therefore, $a_j \in \left[0, d_{*j}^2\right]$

with $d^2_{*j} = \min\left\{d^2_{1j}, \ldots, d^2_{cj}\right\}$. If α_j is chosen to be 0, RCFCM is equivalent to FCM. If α_j is set to d^2_{*j}, RCFCM is equivalent to HCM because for each data object exists at least one prototype with a modified distance of 0 which yields a membership value of 1 and consequently a membership value of 0 for all others. In their original papers, Höppner and Klawonn suggest to set $\alpha_j = d^2_{*j} - \eta$ with η being a constant factor $0 \leqslant \eta \in \mathbb{R}$ chosen by the user. However, such an approach has two drawbacks. First, it is possible that $d^2_{*j} < \eta$ which would yield again a negative value for α_j. That can be prevented with an additional rule like $\alpha_j = d^2_{*j} - \eta$ if $d^2_{*j} > \eta$ and $\alpha_j = 0$ otherwise. Second and much more important, a constant additional value is not invariant for linear coordinate transformation. That means, that η would have to be chosen depending on the numerical size of values in X, which makes it difficult to chose a useful value for η. Therefore, I suggest a multiplicative parameter, setting $\alpha_j = \eta \cdot d^2_{*j}$ with $\eta \in [0, 1]$ which solves both issues. That way, η fulfils for RCFCM a similar role as β for PFCM.

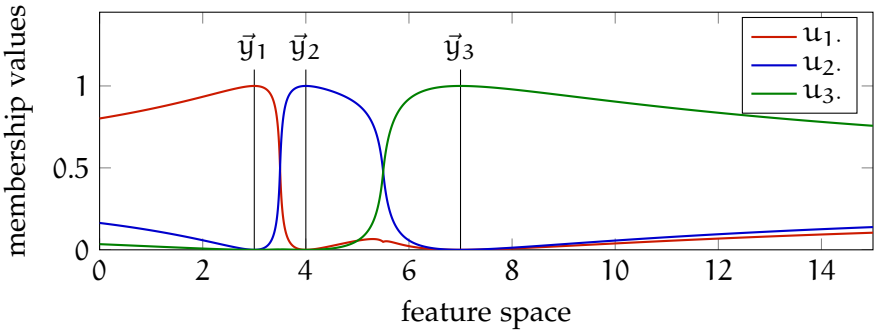


Figure 3.11: Membership value levels u for RCFCM with $\eta = 0.8$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

The membership function values of the 1-dimensional example are presented in Figure 3.11. The membership values of RCFCM are noticeably less fuzzy than in FCM, but they do not form crisp plateaus like in PFCM.

There is always the option to use the gradient descend method. Due to its instability, I would strongly advise against it though.

For updating the prototype positions $\vec{y}_i^{(t)}$, there are two strategies that can be applied. The first (lets name it RCFCM$_A$) one is to treat the $\alpha_j = \eta \cdot d^2_{*j}$ values as 'adaptable constants' as it is done by Höppner and Klawonn in [Höppner and Klawonn, 2001; Höppner and Klawonn, 2003]. They are treated as constants because they are not involved in the updating process, but they are not constant in their values because their value is recalculated in each iteration. The second option (RCFCM$_B$) is to treat $\alpha_j(Y) = \eta \cdot \min\left\{d^2_{1j}, \ldots, d^2_{cj}\right\}$ as a function of the prototypes Y. The second option implies that when deriving the objective function for the prototype vectors, the equation for α_j as a function of the prototypes is taken into account. Both options have advantages and disadvantages.

Starting with RCFCM_A, the second term in RCFCM is added to change the way how membership values are calculated (i.e. to force a more crisp clustering). It is not intended however that it influences how prototypes are updated. For updating the prototypes, α_j are considered to be constant, and thus, the update equation is derived analogously to FCM with $\omega = 2$:

I consider the update scheme of RCFCM_A as a major flaw in the algorithm and I almost discarded RCFCM all-together because of that. However, RCFCM_A works very well, see Chapter 5

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2 \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2} \tag{3.28}$$

Treating the values of α_j as constants has one drawback. They are not involved in the optimization process, but are still changed every iteration, so the function that is supposed to be optimised (J_{RCFCM}) changes every iteration. In this case, it is hard to speak of an optimization algorithm for J_{RCFCM} because in each iteration a different equation is optimised.

The alternative is to treat the α_j as functions of prototypes and adapt the prototype update process accordingly. The derivative of the update equation for RCFCM_B is similar to FCM and results in an update equation:

This option of treating additional parameters in FCM as 'adaptable constants' is also used in Competitive Agglomeration Clustering [Frigui and Krishnapuram, 1997] and its derivatives which is equally dirty from the mathematical point of view.

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2 \vec{x}_j \; - \eta \sum_{\substack{j=1 \\ d_{ij}^{(t)} = d_{*j}^{(t)}}}^{n} \vec{x}_j \cdot U_{\frac{1}{2},j}^{(t+1)}}{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2 - \eta \sum_{\substack{j=1 \\ d_{ij}^{(t)} = d_{*j}^{(t)}}}^{n} U_{\frac{1}{2},j}^{(t+1)}} \tag{3.29}$$

with $U_{\frac{1}{2},j}^{(t+1)} = \sum_{i=1}^{c} \left(u_{ij}^{(t+1)} - \frac{1}{2}\right)^2$ being the additional term of RCFCM. With this more elaborate version of the update equation, RCFCM is a proper optimization algorithm. The drawback is, that it imposes some unwanted and undesired effects during updating the prototypes. As the minus sign indicates, there are circumstances that the update vector w.r.t. one data object points away from the data object. In extreme cases, this can cause the prototype to leave the convex hull of the data objects, which should never happen. The update vector of y_i w.r.t. x_j points away from x_j, if the inequality $d_{ij}^2 \cdot u_{ij}^2 < \eta \cdot (u_{ij} - \frac{1}{2})^2 \cdot d_{*j}^2$ holds. This can be the case for all prototypes if the highest membership (associated with the closest prototype) value is significantly smaller than $\frac{1}{4}$ and the parameter η is chosen to be close to 1. This means that the objective function would be best minimised if all prototypes are further away from this data object, which is obviously not

good if this data object is part of a cluster. In practice, this version of the update equation does not work quite as well as RCFCM_A. Often, prototypes get pushed away from the majority of the data objects and are unable to converge on a cloud of data objects.

Despite the fact that RCFCM_B is superior to RCFCM_A in terms of mathematical correctness, RCFCM_A works better in practice and is therefore the update equation of choice in the reminder of this work. A noise cluster does not change that (if at all, it makes it worse) and therefore, in the next subsection, RCFCM_B is not further discussed.

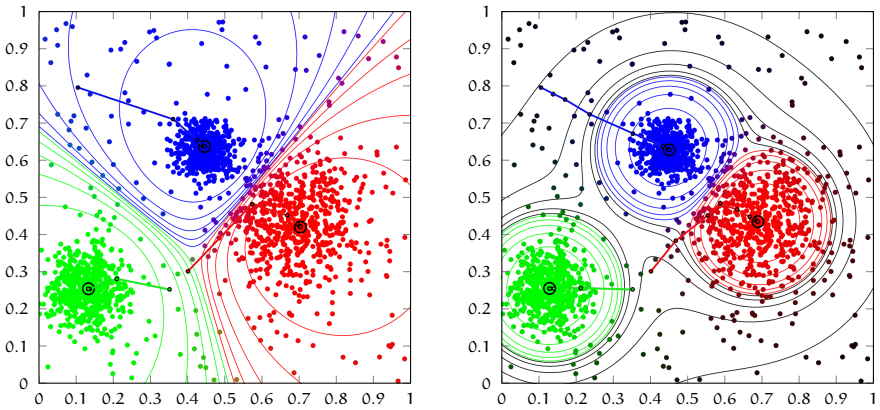


Figure 3.12: The 2-dimensional example data set, clustered with RCFCM (left) and RCNFCM (right) with parameters $\eta = 0.8$ and noise distance $d_{\text{noise}} = 0.2$.

As usual, the iteration process is stopped as soon as the maximal change in prototype location is below a predefined threshold. The runtime complexity is identical to FCM: each iteration of RCFCM is in $\mathcal{O}(c \cdot n \cdot m)$. The computation time for the parameters a_j is linear in the number of prototypes for each data object, which does not effect the class of computational complexity.

The membership height-lines of RCFCM in Figure 3.12 are smaller and membership levels are more crisp than for FCM in Figure 3.6, which means the algorithm works as intended. The effect can easily be controlled by changing the parameter η, which is similar to adjusting the fuzzifier ω in the interval $[1, 2]$.

3.5.1 *RCFCM with Additional Noise Cluster*

Similar to FCM and PFCM, RCFCM can be equipped with a noise cluster, changing the abbreviation to RCNFCM. The equations for RCNFCM change similar as before only in its indices.

$$J_{\text{RCNFCM}}(X, U, Y) = \sum_{i=0}^{c} \sum_{j=1}^{n} u_{ij}^2 d_{ij}^2 - \sum_{j=1}^{n} a_j \sum_{i=1}^{c} (u_{ij} - \frac{1}{2})^2 \qquad (3.30)$$

For $i = 0, \ldots, c$ and $j = 1, \ldots, n$:

$$u_{ij}^{(t+1)} = \frac{\frac{1}{\left(d_{ij}^{(t)}\right)^2 - a_j^{(t)}}}{\sum_{k=0}^{c} \frac{1}{\left(d_{kj}^{(t)}\right)^2 - a_j^{(t)}}} \tag{3.31}$$

and for $i = 1, \ldots, c$

$$\vec{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2 \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t+1)}\right)^2} \tag{3.32}$$

As for PNFCM, the noise distance d_{noise} is treated as a normal distance value and is therefore also included in calculating the parameters a_j. Since d_{noise} is a constant, the a_j parameters become bounded to $a_j \in [0, \eta \cdot d_{noise}]$.

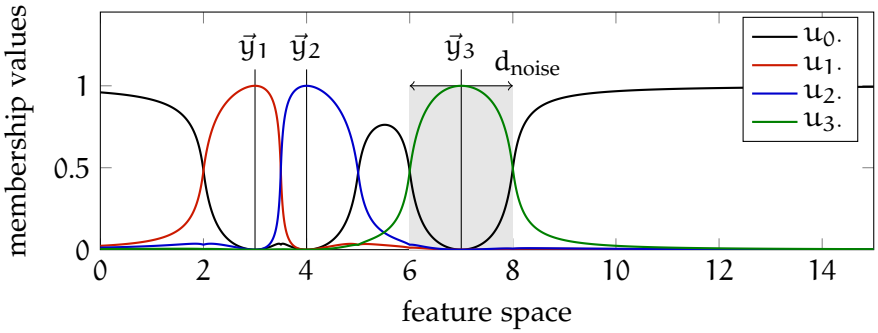


Figure 3.13: Membership value levels u for RCNFCM with $\eta = 0.8$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

The change in membership values due to the noise cluster can again be observed by comparing Figures 3.11 and 3.13. The crispness of the noise clustering can be observed by comparing the right-hand side of Figures 3.6 and 3.12. Not surprisingly, the effect is similar as in NFCM and PNFCM, stronger than for NFCM and less crisp than for PNFCM.

3.6 EXPECTATION MAXIMIZATION

The expectation maximization algorithm (EM) [Bilmes, 1997; Borgelt, 2005; Dempster et al., 1977; Wu, 1983] is a general algorithm to fit a statistical model to a data set. It is an iterative algorithm that adjusts the model parameters to maximise the likelihood of observing

the given data set [Fisher, 1925]. In contrast to the last algorithms discussed in this chapter, it is based on probability theory rather than fuzzy sets and also the involved variables are interpreted differently even though they fulfil similar roles. EM leaves the choice of the statistical model open, hence it defines a large family of algorithms. In this work, only a special type of model, the Gaussian mixture model (GMM), is discussed because of its similarity to previous algorithms. For the GMM model, it is assumed that the data is generated by a mixture of multidimensional normal distributions. Each normal distribution is parametrised by an expectation vector and a covariance matrix Σ of size $m \times m$. In a high-dimensional environment, a general covariance matrix can pose problems because the high number of involved parameters create too many possibilities to adapt the model to the data. To avoid problems related to the number of parameters, the covariance matrix is limited to be a scaled identity matrix: $\Sigma = \sigma^2 \cdot E$ with $\sigma^2 \in \mathbb{R}$ being the classical variance and E the identity matrix. In other words, only hyperspherical normal distributions are considered for EMGMM within this thesis.

The explanations below are following [Borgelt, 2005], however the notations are incompatible. I am sorry for that, but given the content above, I could not use the same notation as C. Borgelt. Also, I omitted the tailing C (or in this case Θ) in the function bodies behind the ';' in order to increase readability.

In various publications, for example in [Bilmes, 1997; Borgelt, 2005], it is written in detail how to arrive at the equation for the expected log-likelihood (objective) function stated below. The derivation of the objective function (Equation (3.41)) from first principles is quite complex which is why some of the tedious equations are omitted in this work. The idea and general approach however is presented because the interpretation of the mathematical symbols are different than for fuzzy clustering algorithms.

The Gaussian mixture model consists of c m-dimensional normal distributions $\mathcal{N}_i = \mathcal{N}(\vec{y}_i, \sigma_i^2)$, $i = 1...c$, one for each class/cluster. The set of expectation vectors $Y = \{\vec{y}_1, \ldots, \vec{y}_c\}$ for the normal distributions is analogous to the prototypes from previous algorithms, which is why the same symbols are used for them. The set of variances $S = \{\sigma_1^2, \ldots, \sigma_c^2\}$ state the spread of the normal distribution in the feature space. Each distribution is defined by its probability density function $f_{\mathcal{N}_i} : \mathbb{R}^m \to \mathbb{R}_{\geqslant 0}$ and to complete the mixture model the probability that a (any) data object is sampled from (generated by) $\mathcal{N}_i$ is denoted by $r_i = P(v = i)$ with $v \in \{1, \ldots, c\}$ being a random variable that models the relative weight of the clusters. The set of probabilities/weights $R = \{r_1, \ldots, r_c\}$, are a-priori unknown and are therefore part of the optimization process of EMGMM. The values in R cover all possible values for the random variable v, therefore the probabilities must sum up to 1:

$$1 = \sum_{i=1}^{c} r_i \tag{3.33}$$

and can also be interpreted as the discrete probability density function of v. The EMGMM model is defined by the probability density function of GMM:

$$f_{GMM}(\vec{x}) = \sum_{i=1}^{c} r_i \cdot f_{\mathcal{N}_i}(\vec{x}) \tag{3.34}$$

Since f_{GMM} is a continuous probability density function, the probability to find a data object at any specific point $\vec{x}$ in the feature space is 0. But the probability density at $\vec{x}$ might be larger than 0, which is expressed by stating that the *likelihood* to find a data object at $\vec{x}$ is equal to the probability density at $f_{GMM}(\vec{x})$. Formally, let $\vec{X}$ be a GMM-distributed random variable that lives in the feature space $\mathbb{R}^m$. $P(\vec{X} = \vec{x}) = 0$ holds for all $\vec{x} \in \mathbb{R}^m$, but the likelihood $L(\vec{X} = \vec{x}) = f_{GMM}(\vec{x})$ is not constantly 0. The likelihood to observe an entire data set $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ is the combined likelihood to observe $\vec{x}_1$ *and* $\vec{x}_2$ *and* ... *and* $\vec{x}_n$ simultaneously. Within the frame of probability theory, this can be expressed by multiplying the individual likelihood functions, which results in a likelihood function for the data set X:

$$L(X) = \prod_{j=1}^{n} f_{GMM}(\vec{x}_j) = \prod_{j=1}^{n} \sum_{i=1}^{c} r_i \cdot f_{\mathcal{N}_i}(\vec{x}_j) \tag{3.35}$$

or equivalently by the log-likelihood function

$$\ln(L(X)) = \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{c} r_i \cdot f_{\mathcal{N}_i}(\vec{x}_j) \right) \tag{3.36}$$

The task of EMGMM is to optimise the parameters of Y, S and R so that the likelihood of observing the data set $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ becomes maximal. The problem with this equation is the sum inside the logarithm, which prevents an effective optimization of the parameters in Y, S and R using the alternating optimization strategy.

To make the EM algorithm mathematically feasible, it is necessary to introduce the additional (unknown) class property of the data objects. It is assumed that for each data object $\vec{x}_j \in X$, a hidden information $v_j \in \{1, \ldots, c\}$ exists that describes to which class (normal distribution) $\vec{x}_j$ belongs. The above defined random variable $v \in \{1, \ldots, n\}$ models the same information and $P(\vec{X} = \vec{x}_j, v = v_j)$ describes the joint probability to observe data object $\vec{x}_j$ and that it belongs to class v_j. The extended data set (X, V) with $V = \{v_1, \ldots, v_n\}$ is defined as a tuple of the data set and the assignment values.

In a sense, it is the set of assignments V that any clustering algorithm is supposed to detect.

$f_{\vec{X},v}(\vec{x}, i)$ denotes the joined probability density of observing the extended data object $(\vec{x}, i) \in (X, V)$:

$$f_{\vec{X},v}(\vec{x}, i) = P(v = i) \frac{1}{(2\pi\sigma_i^2)^{\frac{m}{2}}} e^{-\frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}}$$
$$= r_i \frac{1}{(2\pi\sigma_i^2)^{\frac{m}{2}}} e^{-\frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}}$$

Note again, the different font of the set of variables V and the vector of random variables $\vec{V}$. I admit, the font difference is subtle, but since the arrow indicates the vector notation, the symbols should be identifiable.

Let the likelihood to observe the extended data set (X, V) be expressed by the extended (log-) likelihood function:

$$L(X, V) = \prod_{j=1}^{n} f_{\vec{X},v}(\vec{x}_j, v_j)$$
$$\ln(L(X, V)) = \sum_{j=1}^{n} \ln\left(f_{\vec{X},v}(\vec{x}_j, v_j)\right)$$

Note that the sum inside the logarithm (see Equation (3.36)) vanished at the expense of using the unknown values v_j. The trick to solve this problem is to model the class parameters v_j as random variables.

Let $\vec{V} = (v_1, \ldots, v_n)$ be a vector of random variables with $v_j \in \{1, \ldots, c\}$ that model the unknown class properties of the data objects. Stating the (log-) likelihood with this set of random variables instead of the unknown class property converts the result of the (log-) likelihood function into a random variable.

$$L(X, \vec{V}) = \prod_{j=1}^{n} f_{\vec{X},v}(\vec{x}_j, v_j) \tag{3.37}$$

$$\ln(L(X, \vec{V})) = \sum_{j=1}^{n} \ln\left(f_{\vec{X},v}(\vec{x}_j, v_j)\right) \tag{3.38}$$

The probability that data object j belongs to distribution i, given the observation of $\vec{x}_j$ is expressed using:

$$u_{ij} = P(v = v_j \mid v_j = i, \vec{X} = \vec{x}_j)$$
$$= P(v = i \mid \vec{X} = \vec{x}_j) \tag{3.39}$$

The value is analogous to the fuzzy values of previous algorithms, which justifies the identical notation.

It is assumed that the expectation of the extended randomised (log-) likelihood functions $E(L(X, \vec{V}))$ and $E(\ln(L(X, \vec{V})))$ exist. The conditional expectation of the extended randomised log-likelihood function, given the observation of the data set X is treated as objective function $J_{EMGMM} = E(\ln(L(X, \vec{V}))|X)$. The task is to maximise the (conditional) expectation by adjusting the parameters in Y, S, R and U, hence the name *Expectation Maximization*. The objective function is reformulated to apply the alternating optimization strategy, see

[Borgelt, 2005], pages 142–144 for the derivation of the step between the first and second line:

$$\begin{aligned}
&J_{EMGMM}(X, U, Y, S, R) = E(\ln(L(X, \vec{V}))|X) \\
&= \sum_{i=1}^{c} \sum_{j=1}^{n} P(v = i \mid \vec{X} = \vec{x}_j) \ln\left(f_{\vec{X},v}(\vec{x}_j, i)\right) \\
&= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} \ln\left(f_{\vec{X},v}(\vec{x}_j, i)\right) \qquad (3.40) \\
&= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} \ln\left(r_i \cdot \frac{1}{(2\pi\sigma_i^2)^{\frac{m}{2}}} e^{-\frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}}\right) \\
&= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} \left(\ln(r_i) - \frac{m}{2}\ln(2\pi\sigma_i^2) - \frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}\right) \qquad (3.41)
\end{aligned}$$

The values in R sum up to 1 (see Equation (3.33)) which is achieved in the alternative optimization algorithm by adding Lagrange multipliers to J_{EMGMM}, similar to previous algorithms. The Lagrange-extended objective function is therefore:

$$\begin{aligned}
&J_{EMGMM}(X, U, Y, S, R, \lambda) \\
&\quad = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} \left(\ln(r_i) - \frac{m}{2}\ln(2\pi\sigma_i^2) - \frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}\right) \\
&\qquad\qquad + \lambda\left(1 - \sum_{i=1}^{c} r_i\right) \qquad (3.42)
\end{aligned}$$

Using this objective function, the alternating optimization strategy (see Section 3.1.3) is applied to maximise the parameters in $\Theta = \{Y, S, R, U, \lambda\}$ by using a three step process by combining the parameters into groups U, Y, S and $\{R, \lambda\}$.

Computing $\frac{\partial}{\partial u_{ij}} J_{EMGMM} = 0$ to calculate the value of u_{ij} does not make sense, since the derivative of J_{EMGMM} does not depend on u_{ij}. It is however possible to calculate u_{ij} using its definition (see Equation (3.39)).

$$\begin{aligned}
u_{ij} &= P(v = i \mid \vec{X}_j = \vec{x}_j) = f_{v|\vec{X}}(i \mid \vec{x}_j) = \frac{f_{\vec{X},v}(\vec{x}_j, i)}{f_{\vec{X}}(\vec{x}_j)} \\
&= \frac{f_{\vec{X},v}(\vec{x}_j, i)}{\sum_{k=1}^{c} f_{\vec{X},v}(\vec{x}_j, k)} = \frac{r_i \cdot f_{\vec{X}|v}(\vec{x}_j \mid i)}{\sum_{k=1}^{c} r_k \cdot f_{\vec{X}|v}(\vec{x}_j \mid k)}
\end{aligned}$$

With introducing the iteration constant, the variables for the *expectation step* can be calculated:

$$u_{ij}^{(t+1)} = \frac{r_i^{(t+1)} \cdot f_{\vec{X}|v}(\vec{x}_j \mid i)}{\sum_{k=1}^{c} r_k^{(t+1)} \cdot f_{\vec{X}|v}(\vec{x}_j \mid k)} \qquad (3.43)$$

by using the remaining parameters $Y^{(t)}$, $S^{(t)}$ and $R^{(t)}$. In the maximization step $Y^{(t+1)}$, $S^{(t+1)}$ and $R^{(t+1)}$ are calculated using $U^{(t+1)}$ and J_{EMGMM}. The parameter for the maximization step are computed, as usually, by using the necessary condition of a local maxima of the objective function: setting the derivative 0.

Starting with the new prototype position:

$$\begin{aligned}\vec{0} &\stackrel{!}{=} \nabla_{\vec{y}_i} J_{EMGMM}(X, U, Y, S, R, \lambda)\\ &= \sum_{j=1}^{n} u_{ij} \nabla_{\vec{y}_i} \left(-\frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}\right) = -\frac{1}{2}\frac{1}{\sigma_i^2}\sum_{j=1}^{n} u_{ij} \nabla_{\vec{y}_i} d_{ij}^2\\ &= -\frac{1}{2}\frac{1}{\sigma_i^2}\sum_{j=1}^{n} u_{ij}(\vec{y}_i - \vec{x}_j) = \sum_{j=1}^{n} u_{ij}(\vec{y}_i - \vec{x}_j)\\ \Rightarrow \vec{y}_i^{(t+1)} &= \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{(t+1)}} \end{aligned} \tag{3.44}$$

Deriving for σ_i^{-2} instead if σ_i simplifies the equations significantly. That is also the reason for not simplifying J_{EMGMM} w.r.t. σ in Equation 3.41.

Using the results directly for updating the variances yields:

$$\begin{aligned}0 &\stackrel{!}{=} \frac{\partial}{\partial \sigma_i^{-2}} J_{EMGMM}(X, U, Y, S, R, \lambda)\\ &= \sum_{j=1}^{n} u_{ij} \frac{\partial}{\partial \sigma_i^{-2}} \left(-\frac{m}{2}\ln(2\pi\sigma_i^2) - \frac{1}{2}\frac{d_{ij}^2}{\sigma_i^2}\right)\\ &= \sum_{j=1}^{n} u_{ij} \frac{\partial}{\partial \sigma_i^{-2}} \left(-\frac{m}{2}\ln(2\pi) + \frac{m}{2}\ln(\sigma_i^{-2}) - \frac{1}{2} d_{ij}^2 \sigma_i^{-2}\right)\\ &= \frac{1}{2}\sum_{j=1}^{n} u_{ij} \frac{\partial}{\partial \sigma_i^{-2}} \left(m\ln(\sigma_i^{-2}) - d_{ij}^2 \sigma_i^{-2}\right)\\ &= \frac{1}{2}\sum_{j=1}^{n} u_{ij} \left(m\sigma_i^2 - d_{ij}^2\right) = m\sigma_i^2 \sum_{j=1}^{n} u_{ij} - \sum_{j=1}^{n} u_{ij} d_{ij}^2 \end{aligned}$$

from which follows:

$$\left(\sigma_i^{(t+1)}\right)^2 = \frac{1}{m}\frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \left(d_{ij}^{(t+1)}\right)^2}{\sum_{j=1}^{n} u_{ij}^{(t+1)}} \tag{3.45}$$

And finally, updating the marginal probabilities:

$$\begin{aligned}0 &\stackrel{!}{=} \frac{\partial}{\partial r_i} J_{EMGMM}(X, U, Y, S, R, \lambda)\\ &= \frac{\partial}{\partial r_i}\sum_{j=1}^{n} u_{ij}\ln(r_i) + \frac{\partial}{\partial r_i}\lambda\left(1 - \sum_{k=1}^{c} r_k\right)\\ &= \sum_{j=1}^{n} u_{ij}\frac{1}{r_i} - \lambda\\ \Rightarrow r_i &= \frac{1}{\lambda}\sum_{j=1}^{n} u_{ij} \end{aligned} \tag{3.46}$$

Using the property that all r_i have to sum up to one yields:

$$1 = \sum_{k=1}^{c} r_k = \sum_{k=1}^{c} \frac{1}{\lambda} \sum_{j=1}^{n} u_{kj} = \frac{1}{\lambda} \sum_{k=1}^{c} \sum_{j=1}^{n} u_{kj}$$

$$\Rightarrow \lambda = \sum_{j=1}^{n} \underbrace{\sum_{k=1}^{c} u_{kj}}_{=1} = n \tag{3.47}$$

Replacing λ in equation (3.46) and adding the iteration variable leads to

$$r_i^{(t+1)} = \frac{1}{n} \sum_{j=1}^{n} u_{ij}^{(t+1)} \tag{3.48}$$

Now, all update equations for the maximization step are complete. The iteration process is terminated as before, if the prototype locations in Y, the variances S and marginal probabilities R change less than a threshold $\varepsilon > 0$. Y, S and R have to be stored anyway, which contain $m \cdot c + c + c$ values which is much less, assuming $m << n$.

In case of EMGMM, I did not found a way to implement the algorithm without storing the membership value matrix. The computations of Y, S and R require the values within U and they cannot be computed in parallel to U because S depends on the final values of Y.

The clustering process with EMGMM is quite different from the above discussed fuzzy approaches. The difference comes from the ability of EMGMM to adjust to the size as well as the number of data objects within a cluster. So it is much more adaptive than the fuzzy approaches, at the same time, it is also more sensitive to get stuck in a local minima that is not optimal for clustering purposes. As Borgelt pointed out in [Borgelt, 2005], a globally optimal solution is that all clusters but one collapse on just one data object while the remaining cluster covers almost the entire data set. Therefore, the variance of the clusters must be restricted to a certain interval. Also due to the fact that exponential values are to be calculated, numerical problems are likely to arise which is another reason to consider an interval of valid values for the variances of individual prototypes.

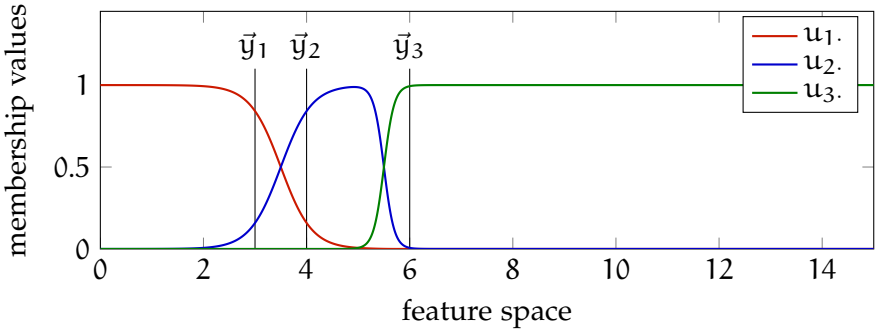


Figure 3.14: Conditional probabilities u for EMGMM with a variance $\sigma^2 = 0.3$ for all prototypes in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

There are further differences regarding the conditional probability values (membership values) w.r.t. FCM and alike. In fuzzy algorithms,

the membership value near a prototype always approaches 1 for that cluster. For EMGMM, this is not necessarily the case as the conditional probability values are calculated in a different way. The effect can be observed in Figure 3.14, the location of the prototypes might be quite different than the location of the maximal conditional probability for one cluster. This effect is partly artificial because with the EMGMM algorithm, it rarely happens that two prototypes come close to each other and have the same variance and cluster probability. So in reality, this situation is less likely to occur.

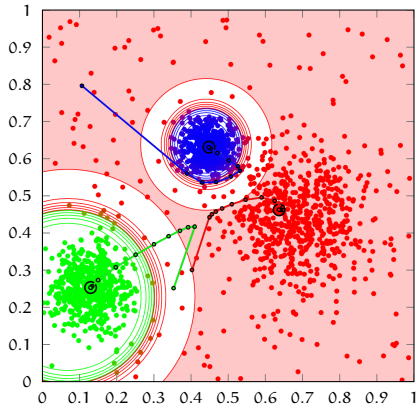


Figure 3.15: The 2-dimensional example data set, clustered with EMGMM.

The initialization for EMGMM is also changing slightly, because all three sets of variables Y, S and R have to be initialised. As before, Y is initialised randomly. However, there is no reason to initialise S and R randomly. It is enough to set all values of S to a reasonable value, for example $\forall \sigma^2 \in S$ set $\sigma^2 = \overline{E}(D^2_{\overline{E}(X)}(X))$. The mean of the data set is used as query point $\vec{q} = \overline{E}(X)$ and the mean of the squared distance to the query point $\vec{q}$ is used for the variance σ^2. Finally, the values of R can be simply set to $\frac{1}{c}$ which induces no preference for any cluster: $\forall r \in R$ set $r = \frac{1}{c}$.

Applied on the example data set, EMGMM produces a quite common effect with expectation maximization and a Gaussian mixture model. 2 of the 3 clusters correctly cover their respective data objects, while one cluster becomes huge and covers not only the data objects of one class, but also the noise data objects of the entire data set. So one of the clusters effectively fulfils the role of the noise cluster, which is not available for EMGMM. Limiting the diameter of the clusters is one way of mitigating the effect, if the size of the clusters is known in advance, alternatively, one additional cluster can be introduced that is initialised with a larger variance to soak up all the noise data objects. Another effect on EMGMM can be observed: the iteration process is not as straight forwards as for the other fuzzy algorithms. It is very likely that the prototypes change directions and run in loops because

To my knowledge, there is no version of the EM algorithm with the Gaussian mixture model, containing a noise cluster. It might be possible to develop one version though.

the variance of the individual normal distributions is changing over time.

3.7 APPLYING PROTOTYPE BASED CLUSTERING ALGORITHMS

Applying clustering algorithms is not trivial. There are several parameters to consider, most importantly the number of clusters but also algorithm specific parameters and a suitable termination threshold for iterative algorithms. In this section, these topics are discussed as well as: initialization methods, termination detection and cascading clustering algorithms.

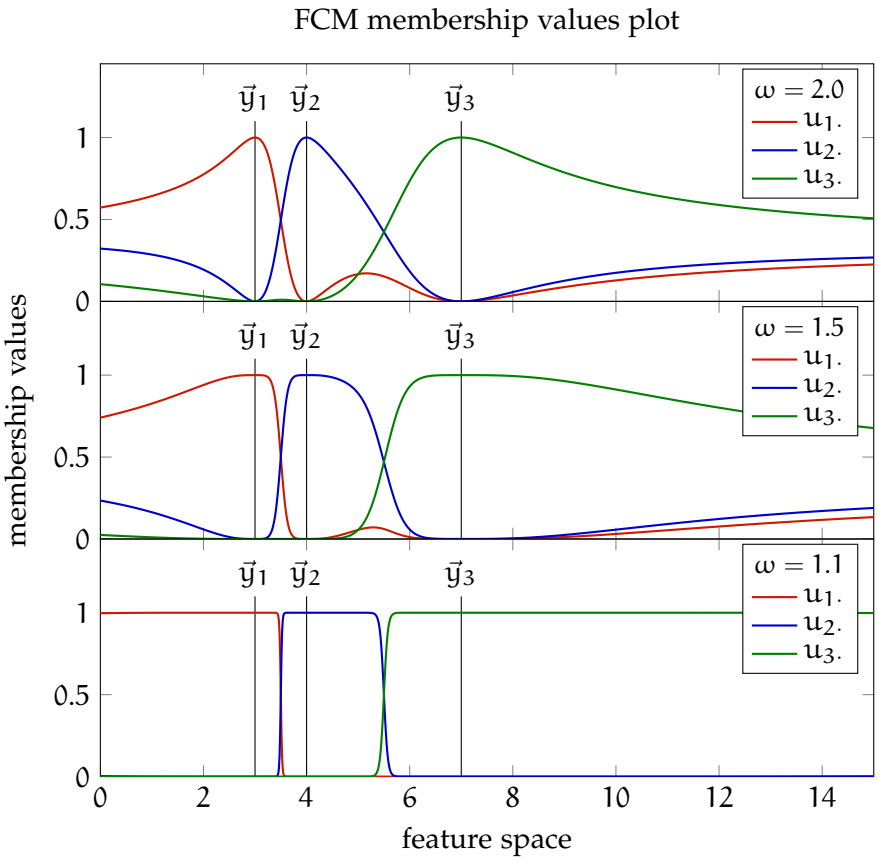


Figure 3.16: Membership value levels u for FCM with $\omega = 1.2$, $\omega = 2$ and $\omega = 5$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

3.7.1 *Fuzzifier Parameters*

The fuzzifier ω in FCM and NFCM (Section 3.3), the β parameter in PFCM and PNFCM (Section 3.4) as well as the distance correction parameter η for RCFCM and RCNFCM (Section 3.5) all serve a

similar purpose. With them, it is possible to adjust the level of fuzziness/crispness. Consider again the 1-dimensional data set to plot the membership values, presented in Section 3.1.5. In Figures 3.16, 3.17 and 3.18 the membership values for these three clusters are plotted w.r.t. the spatial location of the prototypes. In a low dimensional data set, these parameters can be used to adjust the form of clusters, in particular the parameters control how much clusters should overlap. In the above mentioned figures, it is easy to see that the parameters ω, β and η can be used to control the relative size of the clusters.

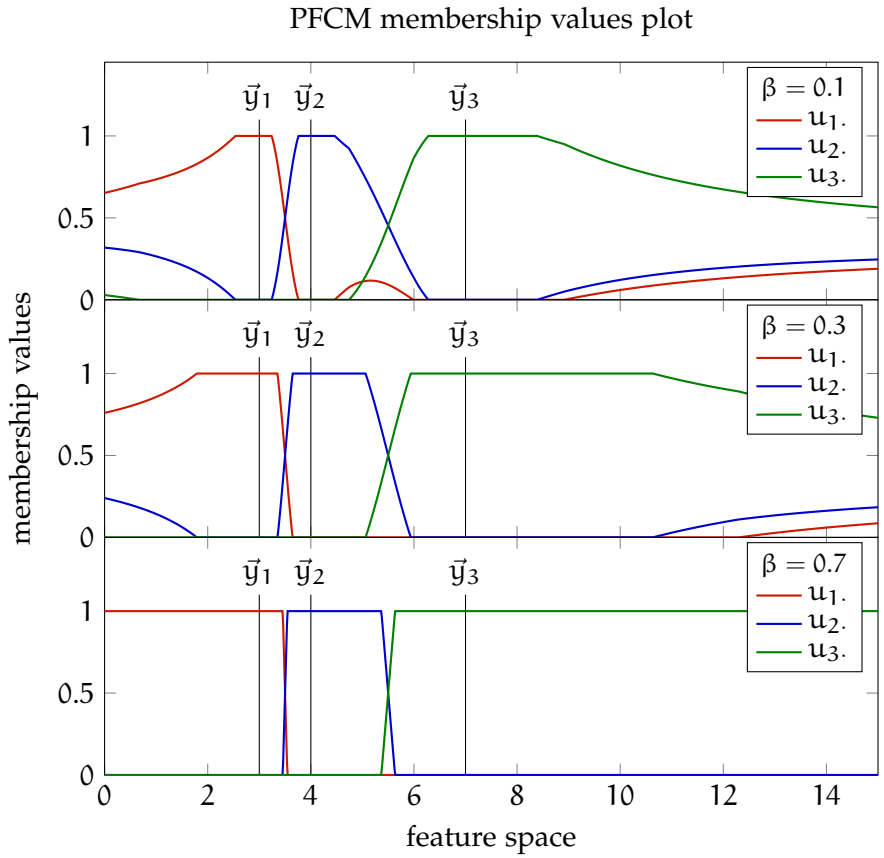


Figure 3.17: Membership value levels u for PFCM with $\beta = 0.2$, $\beta = 0.5$ and $\beta = 0.8$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

In a high-dimensional data set, overlap is not clearly defined because the separation of data objects and the definition of the centre of a cluster is ambiguous. Since overlap is not a viable concept in high dimensions, the fuzzifier parameters can be used to counter the effects of high-dimensional data sets, as it is done for FCM_m and $NFCM_m$, see Section 3.3.2 and Section 5.1.3 for more details.

There is no general best way of selecting of the fuzzifier parameter. Some times, the best strategy is try and error or using a Monte-Carlo simulations.

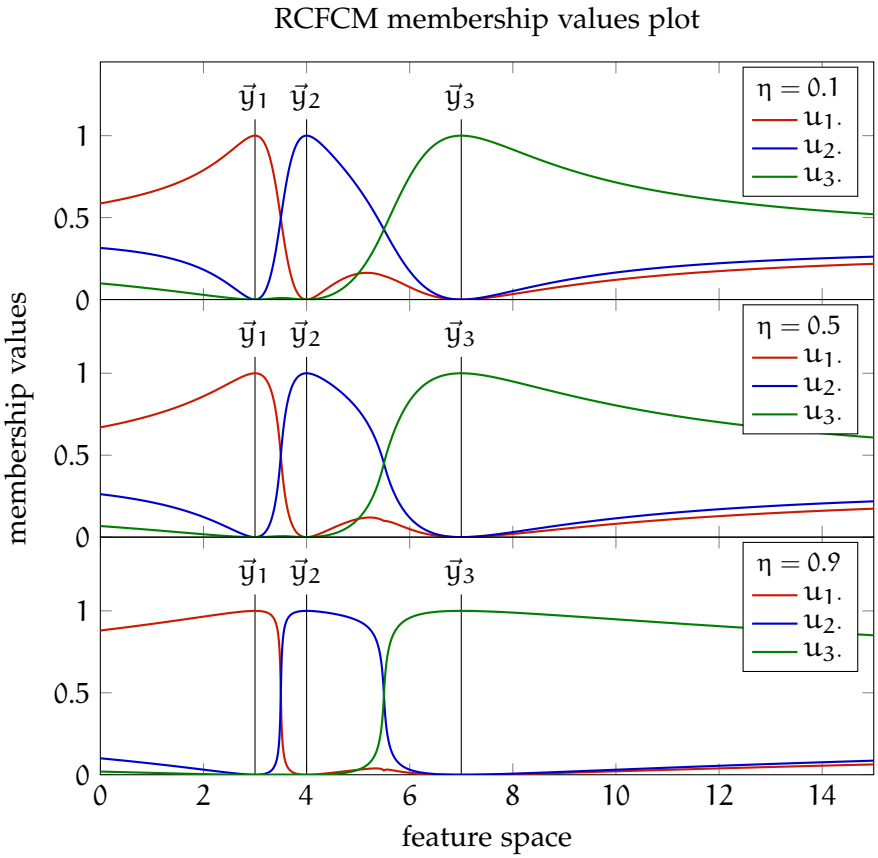


Figure 3.18: Membership value levels u for RCFCM with $\eta = 0.2$, $\eta = 0.5$ and $\eta = 0.8$ in a 1-dimensional scenario with 3 prototypes at $x = 3$, 4 and 7.

3.7.2 *(Variable) Noise Distance*

Another parameter is the noise distance d_{noise}, relevant for NFCM, $NFCM_m$, PNFCM and RCNFCM (Sections 3.3.1, 3.3.2, 3.4.1 and 3.5.1). Originally, the noise distance is used to limit the influence of noise and outliers during the clustering process [Dave, 1991]. This works quite well in low dimensions, if the size of the classes can clearly be specified. However, it is not always possible to determine the size of the classes or the classes might vary greatly in size and shape in which case a best estimate has to be sufficient. Using noise clustering seems in general a good idea because it removes the inconsistent assignment of data objects, that are far away from all clusters. A conservative estimation (i.e. larger than the best value) of the noise distance might often be advisable because underestimating the noise distance can prevent a successful clustering, see Figure 3.19.

Setting the correct noise distance for a given problem is not easy. To ensure a good covering of the classes, the noise distance should be larger than the radius of a class. Otherwise data objects that are far away from the centre of a class might be assigned to the noise

cluster instead of a cluster. Davé proposed in [Dave, 1991] to use a statistical method of pairwise data object distances to estimate the noise distance. Such a statistics is in $\mathcal{O}(n^2)$, which is not practical in many cases. Alternatives have been proposed for example in [Cimino et al., 2005] where they redefine the noise distance during the iteration process of the clustering method. The success of their method greatly depends on how the cluster boundaries are shaped, it only works well on quite crisp cluster boundaries.

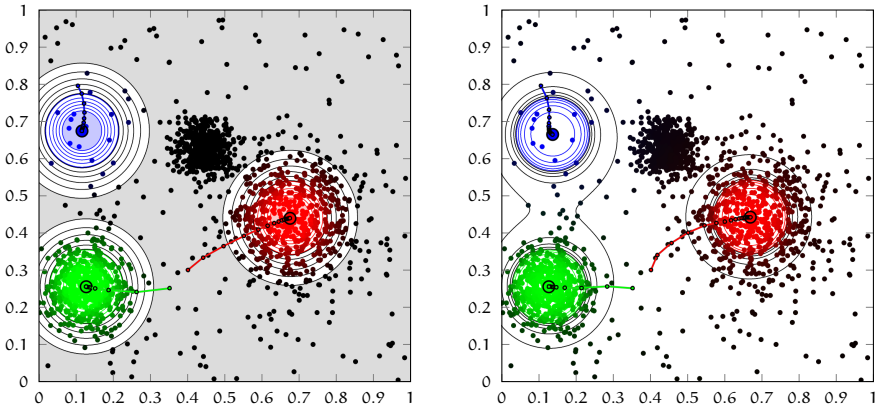


Figure 3.19: The 2-dimensional example data set, clustered with PNFCM (left) and RCNFCM (right) with parameters $\beta = 0.3$, $\eta = 0.8$ and noise distance $d_{noise} = 0.1$.

The noise cluster can be quite troublesome in high-dimensional data sets because the noise distance limits the distance until which data objects have a noticeable influence on the prototypes of the regular clusters. In case of PNFCM, this influence on a data object even drops to 0 if it is far enough away from the prototypes. On the one hand, this is an intended effect, on the other hand, this can pose a serious problem. After initialization, it cannot be guaranteed that all prototypes are located near a class of the data set. It might even be the case that only some random noise data objects are within the influence distance of that prototype which is limited due to the noise cluster. In this case, the prototype is 'lost' for this clustering process because it will never find a class. As an example of this case, see Figure 3.19, where the noise distance is only half of its value in the right-hand sides of Figures 3.9 and 3.12. As a result, in both cases, the blue cluster only represents noise data objects.

To avoid such a situation, the noise distance can be set to a very large value at the beginning of the iteration process, lowering it over time. Any decreasing function with a lower limit d_{noise} can be used. Because the problem has a similar character like finding an optimum for a very epistatic optimization problem, a function similar to simulated cooling [Kirkpatrick et al., 1983] was chosen. Let $d_{noise}^{(t)}$ denote

the noise distance, depending on the number of iterations t and is defined as

$$d_{noise}^{(t)} = d_{noise} + (d_{maxnoise} - d_{noise})e^{-\alpha t} \tag{3.49}$$

which is visualised in Figure 3.20 and used during the experiments in the next chapter. The parameter $d_{maxnoise}$ describes the maximal noise distance in the first iteration while α describes how fast the true noise distance d_{noise} is approached. This way of handling noise may be combined with a minimal number of iterations, to ensure that the algorithm terminates when the variable noise distance $d_{noise}^{(t)}$ is sufficiently low. All these new parameters are quite easy to set for a clustering algorithm.

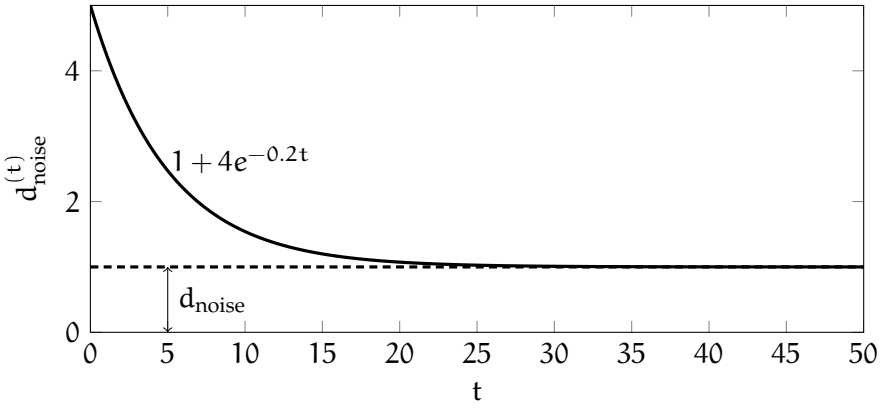


Figure 3.20: Variable noise distance, with $d_{noise} = 1$, $d_{noisemax} = 5$ and $\alpha = 0.2$.

Unfortunately, the variable noise distance imposes a serious mathematical problem. Clustering algorithms as described above are optimization algorithms. A change in parameters like the variable noise distances creates a different optimization problem in each iteration. This is the same problem like in the update equation for RCFCM, see Section 3.5. It may happen that the objective function value increases, due to the change in parameter, even though the optimization procedure is to minimise the objective function. This implies that using this approach, it is not possible any more to prove that the algorithm approaches a local minimum over time.

I have not seen any proof of the principle correctness of clustering algorithms other than HCM and FCM. The clustering community seems to be quite sloppy on this, me included.

The mathematical problem can be avoided if the algorithm is applied in two stages. In the first stage, the noise distance is reduced as described above. In the second stage, the location of the prototypes are used to initialise the same algorithm and perform the clustering process with a fixed noise distance, see also Section 3.7.6.

In retrospect, I should have done that to avoid the mathematical inconsistencies.

3.7.3 *Number of Clusters*

All the clustering algorithms, discussed previously in this chapter require the number of clusters as input. This number should match the

number of classes in the data set, which is often unknown. Determining the number of classes in a data set is very difficult and many papers in clustering science are dedicated to that problem, see [Borgelt and Kruse, 2006; Cuevas et al., 2000; Fraley and Raftery, 1998; Frigui and Krishnapuram, 1996, 1997; Ray and Turi, 1999; Tibshirani et al., 2001; Winkler et al., 2010b] just to name a few. Since even the term 'class/cluster' depends on the problem at hand, finding the number of classes in a data set cannot be described in general. As a result, there are many approaches to identify this number. An exhaustive covering of the topic would be outside the scope of this work, only a short introduction can be given here.

If a good method is available to estimate the quality of clustering results, a try and error approach can be used. The data set is clustered several times with different number of clusters, and the result which yields the best score is used. Such an approach is used by several authors, for example in [Borgelt and Kruse, 2006; Gath and Geva, 1989; Ray and Turi, 1999; Tibshirani et al., 2001]. It should be noted however that most approaches are problematic for high-dimensional data analysis because quality estimation is unstable and unreliable, see Section 5.3 for more information.

Other methods, based on the local density of data objects like DBScan [Ester et al., 1996] and in general hierarchical clustering methods or estimating the number of connected components [Cuevas et al., 2000] are proposed as well. These methods are not well suited for high-dimensional data sets because the density term is ill defined for high-dimensional data due to the concentration of distances. For example, when applied on high-dimensional data sets, DBScan produces either tiny clusters with lots of data objects in the noise cluster or one cluster covers the entire data set.

I made extensive tests with DBScan on high-dimensional data sets. I was unable to find any combination of parameters for DBScan that would work for data sets where the classes are not very concentrated. I did not try OPTICS (see [Ankerst et al., 1999]) though, which is supposedly should work better.

A third way of finding the optimal number of clusters in a data set is presented in [Frigui and Krishnapuram, 1997] as well as in [Winkler et al., 2010a]. The idea is, to start with an overestimation of prototypes which is lowered during the process of an iterative clustering method until the optimal number of clusters is found. Both ideas have a good potential, but it should be noted that it is difficult to find good parameters for the algorithms to run properly and both seem to be not well suited for high-dimensional data.

3.7.4 *Initialization of Prototypes*

If clustering is the optimization of a set of parameters to find a local minimum, initialization is the algorithm to determine the best local minimum.

Once the number of clusters is determined, initializing the prototypes (i.e. finding their initial location) can strongly influence the result of the clustering process. This is especially true for HCM, where prototypes can be stuck without representing a single data object (or representing only noise data objects) due to an unlucky initialization. The quality of other algorithms depend on the initialization as well, a

bad initialization can basically prevent a successful clustering process, even if that is unlikely. There are several approaches to mitigate that risk and to increase the clustering quality and reduce the number of iterations needed for performing the algorithm.

If the data does not contain much noise or outliers, the maximindist method [B.G. Batchelor, 1969] can be used, which samples the initialization positions from the data and maximises the pairwise distances of the samples. As high-dimensional data sets are seldom encountered without noise, this method is usually not a good option.

More sophisticated methods like mountain clustering [Chiu, 1994; Yager and Filev, 1994] are in the region of complexity as a full clustering method, hence can be treated as one. Other methods based on density or hierarchical clustering methods are also possible. But due to the problem that the density is only weakly defined in high-dimensional data sets, they do not provide helpful alternatives in high-dimensional data sets. Another alternative, specialised for high-dimensional data is presented in [Winkler et al., 2013]. The trick in this approach is, to use an alternative distance function which specifically counters the effects of distance concentration. Starting with an overestimation of prototypes that reduces over time can result in a combined estimation for the number of prototypes and a useful initial position for them. The drawback of this approach is, much like the variable noise distance, that it cannot be proven to produce an optimal result. Therefore, it might be used for initialization purposes only.

There are also arguments to not use specialised initialization methods. If a clustering algorithm has to rely on a good initialization to be able to work properly, its result strongly depends on the initialization. In that sense, the initialization algorithm determines the clustering result and therefore, the initialization algorithm is the true clustering algorithm in this case and must face the same problems as regular clustering algorithms. Therefore, sophisticated initialization methods are in truth clustering algorithms, which just shifts the problem from one part of the computation into another but does not solve the problem.

The alternative to data driven initialization is randomly initialised prototypes. The most simple initialization is, sampling the prototype locations from a uniform random distribution on the feature space. This method is rather data independent and fast. Since the random sampling of initial positions influences the clustering result, it is often helpful to do several runs with different samples of initialization positions and use the best result. This procedure however can be slower than using data driven initialization methods.

3.7.5 *Termination Threshold*

The iteration process of the above mentioned clustering algorithms can be terminated using several methods. Usually, the maximum number of iterations t_{max} is limited to avoid arbitrary long execution times of the algorithm. All of these methods require a threshold $\varepsilon \in \mathbb{R}_+$, which is user defined.

From experience, $t_{max} = 50$ iterations is enough to converge to a useful solution.

- It is possible to detect when the membership matrix does not change significantly any more.
  Formally, if $\max_{i=1\dots c} \max_{j=1\dots n} |u_{ij}^{(t)} - u_{ij}^{(t-1)}| < \varepsilon$ holds, the iteration process is terminated. However, this implies that the membership matrix must be stored, which is not necessary in most cases to perform the algorithm. Therefore, especially for data sets with many data objects and clusters, this is a serious overhead in storage complexity.

- If the objective function value does not reduce significantly from one iteration to the next, the iteration process is stopped. Calculating the objective function value J during iteration is possible with little overhead in the above mentioned clustering algorithms. So if $|J^{(t)} - J^{(t-1)}| < \varepsilon$, the algorithm is stopped. There is again a drawback. From a users perspective, it is hard to understand how much change in objective function value is significant. For this reason, this approach is not very practical and seldom used.

- If the change of location of the prototypes is below a certain threshold, the algorithm is terminated. This approach is both practical and easy to understand. The algorithm is stopped if $\max_{i=1\dots c} d(y(t)_i, y(t-1)_i) < \varepsilon$ is true. A drawback of this approach is, that the threshold ε depends on the scale of the data set, which has to be taken into account by the user.

- Using the euclidean distance in this case seems natural, but is misleading in high dimensions. Since the prototypes move in m dimensions, the (euclidean-) distance depends on the number of dimensions. To avoid this dependency, the maximum norm d_{max} instead of the euclidean norm d can be used, taking only the dimension of maximum movement into account:

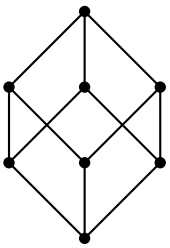
$$d_{max}(y_i^{(t)}, y_i^{(t-1)}) = \max_{k=1\dots m} |y_{i,k}^{(t)} - y_{i,k}^{(t-1)}|$$

Due to its independence w.r.t. the number of dimensions, the last option is preferred in this work.

3.7.6 *Cascading Clustering Algorithms*

There are many situations, where one application of clustering is not sufficient. For example, if the data set is composed of many small classes, which in turn group together to form larger structures. Or if the spatial size of classes as well as the number of data objects per class is very divergent. There are countless more examples where it is useful to apply an algorithm on subsets of the data, based on a previously gained partitioning. This kind of cluster algorithm application is out of scope of this work, but it should be noted that in high-dimensional data sets, a clear grouping of smaller clusters might not be detectable.

In a different situation, it might be helpful to use the prototype locations of one clustering run as initialization for a different algorithm. This way, it is possible to improve the stability of an otherwise unstable clustering algorithm by subjecting to the problems discussed in Section 3.7.4.



4

BENCHMARKS

In the last chapter, in Sections 3.2 to 3.6, 10 clustering algorithms have been presented: HCM, FCM_2, $NFCM_2$, FCM_m, $NFCM_m$, PFCM, PNFCM, RCFCM RCNFCM and EMGMM. Thesis questions Q_2, Q_3 and Q_4 are approached by applying a benchmark on these 10 algorithms. Here, a benchmark is understood as a set of tests that can be used to measure the quality of the mentioned algorithms. Two additional tools are required to achieve that: suitable benchmark data sets and measures to determine the quality of the clustering algorithms results. Both topics are not trivial and are covered within this chapter, beginning with the generation of high-dimensional data sets.

A benchmark as opposed to a real application is useful because it provides a controlled testing environment where the result is known and it is possible to apply the algorithms on a large variety of data sets. By changing the parameters (e.g. number of dimensions, number of classes) of the data generation process, it is possible to analyse the influence of these parameters on the applicability of the algorithms. The biggest disadvantage w.r.t. a real application is however, that a benchmark can only cover a limited variety of data sets and any real data set is likely to be different than any benchmark data set. The relevance of the benchmark is therefore limited by the variety of data sets that are used. In this work, four different families of data sets are utilised, presented in the first section of this chapter, Section 4.1.

The quality of a clustering result can be estimated in at least two ways. In Section 4.2 a method is presented to judge a clustering result by comparing it with the correct result. This is of course only possible if the true clustering result is known like in this benchmark environment. In reality, the true result is not known, so the quality of clustering results have to be determined using a different approach. Measures to determine the cluster quality, based on the abstract notion of a good clustering result and in the absence of the knowledge of the true result, are presented in Section 4.3.

Finally, the execution and how the benchmark results are interpreted is presented in Section 4.4. The results and answers to the thesis-questions Q_2, Q_3 and Q_4, are presented in the next chapter, Chapter 5.

4.1 ARTIFICIALLY GENERATED DATA SETS

Four different families of data sets are used for the benchmark in this work. A data set family consists of a collection of data sets that are generated in a particular way. The first two families are fairly simple mixtures of normal distributions and might be considered as the perfect data sets for prototype based clustering. The third data set is born from the idea to provide a data set where the classes show complex, non linear dependencies among its dimensions that is likely to be found in reality. That is to say, the classes should not mimic reality in the sense that they are simulations of existing data. But the data set is supposed to have similar features, like extending and overlapping non-ellipsoidal shapes with strange and unpredictable geometry. The last family of data sets lives in a boolean environment, where all dimensions are either 0 or 1.

The 4 families of data sets have several things in common. To answer Q_2, Q_3 and Q_4, all data sets are scalable in the number of dimensions m and the number of classes c. The data objects within the individual classes are always generated independently of any other class. This property is utilised by generating the classes before assembling the data sets. So for each number of dimensions m and for each data set family, a set of 250 classes are generated. The data sets within a data set family are then assembled using a random selection of the 250 pre-generated classes. The data objects live in $\mathbb{R}^m$ and are not strictly bound to the unit hypercube $[0,1]^m$, but are generated close to it so that no rescaling prior to applying the clustering algorithms is necessary. Finally, there are no missing values and no extreme outliers or invalid values in the data.

The description of how the data sets are generated is necessarily very technical. Only a short overview of the 4 data set families is presented below, the details are described in Appendix A. Since the data sets are assembled from independently generated classes, the description below focusses on the generation of the classes rather than the data sets. Only in the first two data set families, an assembled data set is shown.

4.1.1 D_1: *Spherical Normal Shaped Classes of Identical Size*

Of course, you know from my introduction in the first chapter, that at least standard FCM_2 fails the 'sanity check' for $m = 50$.

The first data set family D_1 is deliberately designed to be easy for clustering algorithms. It can be used as a sanity check whether or not a clustering algorithm produces a sensible result on high-dimensional data and it can be expected that any clustering algorithm generates almost perfect results on any data set of the D_1 family. The classes for D_1 are sampled from spherical normal distributions with identical variance $\sigma^2 = 0.01$ and 1000 data objects per class. The expectation vectors of the generating normal distributions are sampled

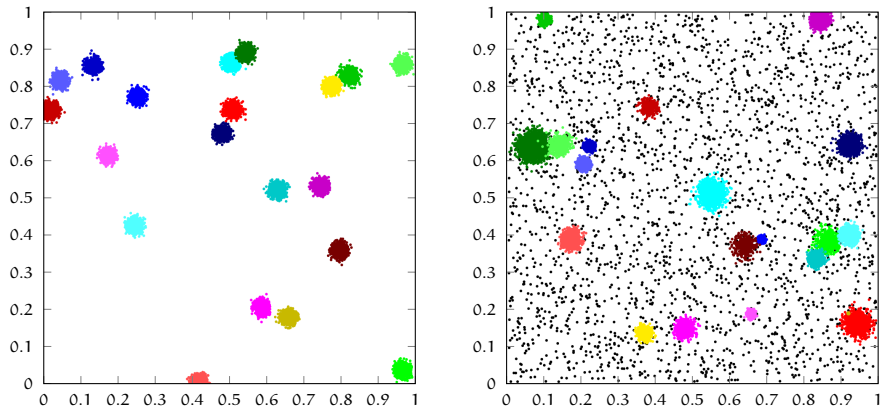


Figure 4.1: Examples of two data sets from families D_1 (left) and D_2 (right). Both with 20 classes and in case of D_2 with 10% noise.

uniformly from the unit hypercube and therefore do not need to be parametrised. By placing the expectation vector inside the unit hypercube, the data objects, sampled from the normal distributions might occasionally be sampled outside the unit hypercube, but they are still close enough such that no rescaling of the data sets is necessary.

In the left-hand side of Figure 4.1, a 2-dimensional example of D_1 is presented where the colour indicates the class information. In 2-dimensions, it is likely, that some classes overlap and it is quite possible that some classes are clustered within the same cluster due to the overlap. However, in dimensions higher than 5 or for a low number of classes, this data set family should be perfect for prototype based clustering and any decent clustering algorithm should be able to find a perfect representation of the classes.

4.1.2 D_2: *Spherical Normal Shaped Classes of Various Size with Noise*

The second family of data sets D_2 is similar to the first, but more challenging. The classes are still sampled from spherical normal distributions, but with variable variance and number of data objects per class. The number of data objects is sampled from a uniform distribution on the interval between 200 and 1800. The variance of the normal distributions (classes) is also sampled from a uniform distribution on the interval between 0 and $\sigma_{max} = 0.02$, independently of the number of data objects. Finally, to each data set in D_2, a fixed fraction of noise data objects are added that are sampled from a uniform distribution on the unit hypercube $[0, 1]^m$. The number of noise data objects is set to be 1/10th of the total number of data object in a data set. Since the number of data objects per class is randomly sampled, the number of noise data objects is determined after the classes for one data set are selected. The right-hand side of Figure 4.1 shows a 2-dimensional example of D_2. As in case of D_1, data objects might be sampled outside

of the unit hypercube, but due to the construction of the classes, the data objects cannot be sampled far away from it.

4.1.3 D_3: *Distorted Classes Data Set*

Data sets of the families D_1 and D_2 are very unrealistic. In reality, data sets do not contain neat and spherical classes, they can have all kinds of complex shapes. The family of data sets D_3 is designed to mimic such complex dependencies by producing very distorted and non-ellipsoid shaped classes.

A data set of the D_3 family is constructed as before, from c, independently generated classes in a m-dimensional real vector space, bound to the unit hypercube. All classes contain a random number of data objects, again sampled from a uniform distribution on the interval $[200, 1800]$. The final data sets of D_3 also contains noise data objects, obtained in the same way as for data sets in D_2.

The process of building the classes however, is much more complicated as in case of the D_2 data set family. The details of the generation process are described in Appendix A.3, only the general strategy without too much detail is described here. Each class is generated individually and the shape of the class is generated iteratively. The iteration process is started with a sample from simple probability distributions, a mixture of an m-dimensional normal distribution and a uniform distribution, bound on the unit hypercube $[0, 1]^m$. After initialization, unary and binary distortion functions are applied on the data objects of the class, forging distortions and pairwise dependencies within each iteration. The functions are randomly selected from a pool of predefined functions. In total, $t_{max} = 2 \cdot m$ iterations are performed, which consist of three individual steps:

1. For all dimensions $k \in \{1, \ldots, m\}$, randomly select one unary function $f \in \{u_1, \ldots, u_6\}$ and apply it on dimension k of each vector in a class $\forall \vec{x}^{(t)} \in C^{(t)}$: $\vec{x}'_k = f(\vec{x}^{(t)}_k)$, creating C' in the process.

2. For all dimensions $k \in \{1, \ldots, m\}$, randomly select one other dimension $r \in \{1, \ldots, m\}$, $r \neq k$ and randomly select one binary function $g \in \{b_1, \ldots, b_5\}$ and apply it on dimension k of each vector in the class $\forall \vec{x}' \in C' : \vec{x}''_k = g(\vec{x}'_k, \vec{x}'_r)$, creating C'' in the process.

3. For all dimensions, $k \in \{1, \ldots, m\}$ normalise the data to keep it well confined in the unit hypercube, creating the final value of $x^{(t+1)}_k$ for this iteration step: $\forall \vec{x}'' \in C'' : \vec{x}^{(t+1)}_k = \text{normalise}(\vec{x}''_k)$, generating $C^{(t+1)}$

Each iteration step is applied independently from the previous until $t_{max} = 2 \cdot m$ iterations are reached and the process is stopped. After

the iteration process is finished, the data objects within the class are pushed in direction of one of the randomly selected corners of the unit hypercube. For extensive detail on the exact process to generate the classes, please see Section A.3 within the appendix.

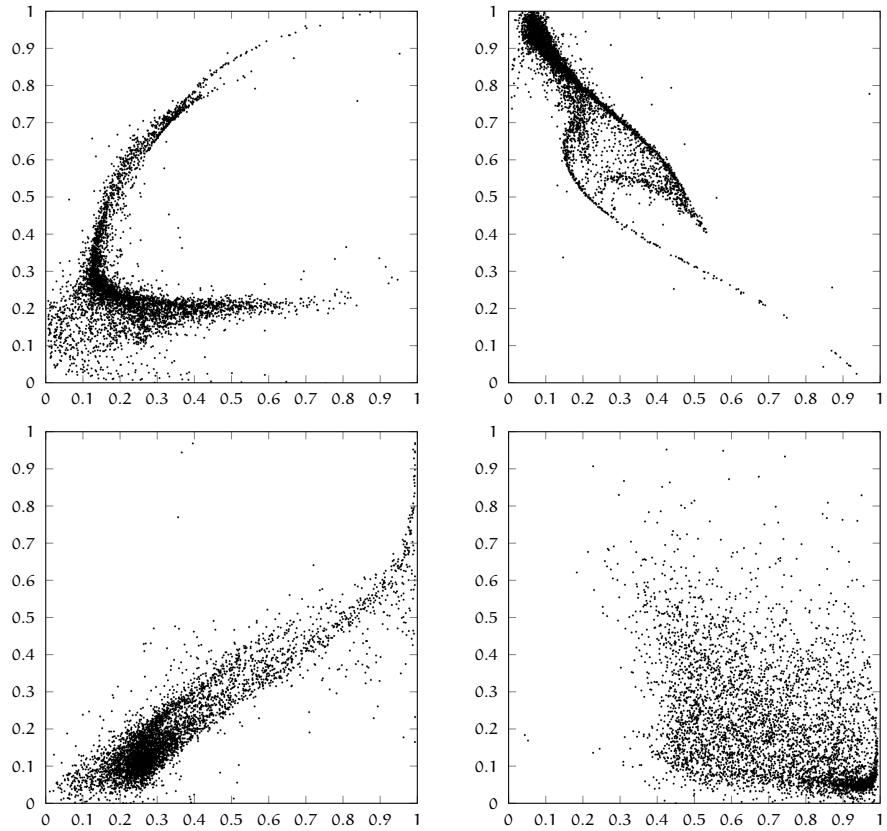


Figure 4.2: Four examples of 10-dimensional classes of the D_3 data set family, projected on 2 dimensions.

I use 5000 *data objects for the figure because the complex structure of the classes can be better visualised with* 5000 *rather than* $\approx$ 1000 *dots.*

As an example, in Figure 4.2, four different classes that are generated in 10 dimensions, with 5000 data objects each are shown. This process of generating classes works quite well until around $m = 1000$ dimensions, at which point the 2-dimensional projection of the data distribution becomes very similar to a projection of a normal distribution. In other words, the distortion becomes so complex, that the sampling of a few thousand data objects is not enough to recognise the structure of the class. To cover even higher dimensions, the number of data objects must increase exponentially with the number of dimensions, as already mentioned in Section 2.5. Within the benchmark for this work, the number of dimensions is limited to 100, which is well within the range of useful parameters for this method of artificial data generation.

4.1.4 D_4: *Corner Classes Data Set*

The fourth and final data set family D_4 for the benchmark consists of binary data objects that live in the corners of the unit hypercube $\{0,1\}^m$. Of course, there are dedicated approaches to analyse binary data like for example Frequent Pattern Mining [Aggarwal and Srikant, 1994]. But these approaches are not always best suited for clustering. In reality, it is very rarely the case that there is a clustering algorithm that is ideal for a given problem. In this context, it is particular interesting how the prototype based clustering algorithms deal with data, that is not really suited for prototype based clustering, such as the family of D_4 data sets. The data sets within the D_4 data set family are designed to be similar to Microarray data (see Section 1.1.6) or the aircraft movement data set, presented in Section 1.1.3.

The generation of classes for data sets within D_4 is done by generating seeds that are then randomly changed in order to generate data objects of the classes. Again, the number of data objects per class is sampled from a uniform distribution between 200 and 1800 data objects. As for D_2 and D_3, the data sets in D_4 have 10% randomly generated noise data objects.

The seed of one class is sampled from one of the corners of the unit hypercube, where a random number with expectation $\sqrt{m}$ of the dimensions are sampled with 1 entries and the remaining with 0. Then, the data objects of that class are generated by randomly altering some entries of the seed, flipping some of the 1 entries to 0 and some of the 0 entries to 1. Details of the process are described in Appendix A.4. As for the data set family D_3, the classes in D_4 are generated independently from one another, hence the generation of classes is done one class at a time and a data set is assembled from previously generated classes.

For example, a 10-dimensional seed may be used to produce the following 6 data objects of one cluster:

dimension:	1	2	3	4	5	6	7	8	9	10
seed:	0	1	0	0	0	0	1	0	1	1
data obj. 1:	0	0	0	0	0	0	1	1	1	1
data obj. 2:	0	1	0	1	0	0	1	0	0	0
data obj. 3:	0	1	1	0	0	0	1	0	1	1
data obj. 4:	0	1	0	0	0	0	1	0	0	1
data obj. 5:	0	1	1	1	0	0	1	0	1	1
data obj. 6:	0	1	0	1	0	0	1	0	0	0

The red entries in the data object list show the places where 0 or 1 entries are switched to be the opposite. In this way, the data objects of one class are reasonably similar to one another and group around the seed. The noise data objects are sampled in the same way as seeds are generated, but without changing any of the entries because no group structure is generated.

4.2 EXTERNAL CLUSTER QUALITY MEASURES

The measurement of the quality of clustering results is based on comparing two (fuzzy) partitions of the data set X. In particular, the (fuzzy) membership matrix U (see 3.1.4) that comes out as the result of a clustering process of one clustering algorithm can be compared with another (fuzzy) membership matrix that comes from another algorithm or is known from the class structure of the data. Within this work, it is not particularly interesting to compute the similarity of the results of two clustering algorithms. It is however interesting to compute the similarity of the partitioning as generated by an algorithm, with the true class structure of the data set as it is known from the generation of the data sets within the benchmark. The term *external* for clustering quality measures means, that *external knowledge* (in this case the class information) is used to measure the quality of the clustering result. The algorithm that compares two (fuzzy) membership matrices is described in the next two subsections, starting with the F_1 measure in a two-class problem, followed by an extension of the F_1 measure to the multi-class problem.

4.2.1 F_1 *Measure for Two-Class Problems*

The F_1 measure [van Rijsbergen, 1979] was originally designed for evaluating the quality of two-classes problems. The two-class problem refers to a partitioning of a data set that consists of one class of interest and one background class. The concept of precision and recall [van Rijsbergen, 1979] is used to quantify the classification quality for a two-class problem. Let $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ be a data set, $C \subset X$ be the class of interest and $B = X \setminus C$ be background data objects that do not belong to the class of interest. An algorithm, that predicts which object belongs to the class and the background respectively might determine the cluster C' and the background set $B' = X \setminus C'$.

Precision is defined as the fraction of correctly classified data objects relative to the number of data objects, identified to be members of the class of interest.

$$p = \frac{|C \cap C'|}{|C'|}$$

Similarly, recall is defined as the fraction of correctly classified data objects relative to the number of data objects, that actually are members of the class of interest.

$$r = \frac{|C \cap C'|}{|C|}$$

In other words, precision describes, how many of the data objects identified as being in the class C', are correct but it does not say

anything about how many are left out. This is covered by recall, which counts the number of correctly identified data objects vs. how many data objects actually are in the class C. Both values take numbers between 0 and 1, with 1 indicating a good quality and 0 indicating bad quality.

Precision and recall can easily be adapted for fuzzy classification values. Let $U' \in [0,1]^{2\times n}$ be the membership matrix as a result from a clustering/classification process, which replaces C' and let $U \in [0,1]^{2\times n}$ be the membership matrix replacing C. The elements are subject to the usual restriction $u_{ij}, u'_{ij} \in [0,1]$, $i \in 1,2$, $j = 1,\dots n$ and $u_{1j} + u_{2j} = 1$, $u'_{1j} + u'_{2j} = 1$. In this context, u_{1j} refers to the membership value of the cluster of interest and u_{2j} to the membership value of the background. Fuzzy versions of precision and recall are defined by the next two equations.

$$p = \frac{\sum_{j=1}^{n} u_{1j} \cdot u'_{1j}}{\sum_{j=1}^{n} u'_{1j}} \qquad r = \frac{\sum_{j=1}^{n} u_{1j} \cdot u'_{1j}}{\sum_{j=1}^{n} u_{1j}}$$

There is no fundamental difference between fuzzy or crisp precision and recall measures. If crisp results instead of fuzzy results are measured, it is enough to restrict one or both fuzzy membership values $u_{ij} \in [0,1]$ ($u'_{ij} \in [0,1]$) to crisp membership values $u_{ij} \in \{0,1\}$ ($u'_{ij} \in \{0,1\}$), so even a mixed comparison is possible.

Precision and recall (whether crisp or fuzzy) taken individually, do not hold sufficient information to judge the quality of the clustering/classification. For example in a crisp problem, precision would be maximal, if there is only one data object classified ($|C'| = 1$ and this one data object is also a member of the class C. Such a classification is clearly of no use if C contains many data objects. This situation is reflected in recall, which has a very low value of $\frac{1}{|C|}$. On the other hand, assume that all data objects are in the clustered/classified set, so that $C' = X$. In this case recall is $r = 1$, but again, such a clustering/classification is not useful, which is reflected in a relatively low value in precision: $\frac{|C|}{|X|}$. As a conclusion, it is always necessary to look at both values together.

A way of combining precision and recall is the F_1 measure, which takes the harmonic mean of both values.

$$F_1(C, C') = 2 \cdot \frac{p \cdot r}{p + r} \tag{4.1}$$

The subscript 1 means, that precision and recall are treated with equal importance. The measure F_β produces a weighted result, giving either precision or recall more importance in the combined value.

$$F_\beta(C, C') = (1 + \beta^2) \cdot \frac{p \cdot r}{\beta^2 \cdot p + r}$$

An adaptive value for β is not utilised within this work, which leaves 1 as only viable value for the parameter β, hence the measure is called F_1 measure.

4.2.2 F_1 *Measure for Multi-Class Problems*

The F_1 measure discussed above is only able to measure the similarity/quality of a two-class clustering result w.r.t. one class of interest. For general clustering problems, it is necessary to extend this approach to a multi-class problem (see [Borgelt, 2006]) which is described in this subsection. The multi-class problem is similar to the two-class problem, the difference is that each partitioning consists of multiple classes (clusters). In this case, it is assumed that there are an equal amount of classes as there are clusters in both partitionings. The extension is achieved by finding the best mapping (permutation) between the two a sets of classes/clusters and averaging the values of the respective two-class $F-1$ measures.

Let there be $c \in \mathbb{N}$ classes in the data set X with n data objects and a (fuzzy) partition $U \in [0,1]^{c\times n}$ of classes as well as a (fuzzy) partitioning $U' \in [0,1]^{c\times n}$ of clusters with the usual restrictions. Then one particular class with index i is treated as the class of interest and it is compared to a cluster with index i'. Let there be a permutation π that maps all classes to the clusters: $i' \equiv \pi(i)$. The mapping π is necessary because the ordering of classes is generally not identical to the ordering of the clusters. Given such a mapping, cluster specific precision and recall values can be computed.

$$p_{ii'} = \frac{\sum_{j=1}^{n} u_{ij} \cdot u'_{i'j}}{\sum_{j=1}^{n} u_{ij}} \qquad r_{ii'} = \frac{\sum_{j=1}^{n} u_{ij} \cdot u'_{i'j}}{\sum_{j=1}^{n} u'_{i'j}}$$

The F_1-measure of a class-cluster pair i, i' is defined as in the two-class problem, Equation (4.1).

$$F_1(i,i') = 2 \cdot \frac{p_{ii'} \cdot r_{ii'}}{p_{ii'} + r_{ii'}}$$

For a particular mapping π, the F_1-measure is then defined as the mean of the individual F_1-measures.

$$F_1(\pi) = \frac{1}{c} \sum_{i=1}^{c} F_1(i, \pi(i)) \tag{4.2}$$

There are also suggestions to use a weighted mean [Sebastiani, 2002], according to the number of data objects, involved in the individual F_1 measures. Using the weighted mean implies that the influence of the classes on the final score depends on the size of the classes. This

procedure puts a higher importance on larger classes and might be advantageous in some applications but the opposite can be true as well. Sometimes the small classes hold the most valuable information because it is rather easy to find the large classes anyway and the problem might be to correctly recognise the data objects in the small classes of a data set. In the context of this benchmark however, all classes are considered equally important.

To gain a meaningful result in this way, the mapping π must be chosen so that the F_1 measure is maximised because only for a maximised value, it can be assumed that the mapping of classes to clusters is optimal. Being a permutation, there are $c!$ different mappings. Of course it is not efficient to test all possible permutations to find the maximal combined F_1 value. The so called *Hungarian Method* [Kuhn, 2005] is an algorithm to find the maximal overall F_1 score. The algorithm requires a two-dimensional matrix $V \in [0,1]^{c \times c}$ which holds all pairwise F_1 scores: $v_{ij} = F_1(i,j)$ with $i = 1,\ldots,c$ and $j = 1,\ldots,c$. Computing V is in runtime complexity $\mathcal{O}(n \cdot c^2)$ while the Hungarian algorithm produces the best mapping π_{max} in $\mathcal{O}(c^3)$, which means an overall runtime complexity in $\mathcal{O}(c^3 + n \cdot c^2)$. With $c << n$, the overall runtime complexity for computing F_1 is in $\mathcal{O}(n \cdot c^2)$. The computation is even faster if at least one of the membership matrices is crisp. In this case the runtime complexity reduces to $\mathcal{O}(n \cdot c + c^3)$ and if both are crisp, it is in $\mathcal{O}(n + c^3)$.

4.2.3 *The Importance of the F_1 Measure and Alternatives*

The F_1 measure plays a central role in this work because it is used to measure the quality of clustering results when comparing them with the true result. It is used to generate the quality plots, presented in Appendices D to G. It is also used to answer thesis questions Q_2, Q_3 and Q_4. Therefore, it is vitally important for this work that the measure is giving useful results. Later in this work, this fact is occasionally emphasized.

A popular alternative to the F_1 approach is the Rand index [Rand, 1971], but it is not well suited for clustering applications with a large number of classes. Without going into too much detail, imagine two partitionings, P_1 and P_2 of the data set X and a pair of clusters/classes $C_1 \in P_1$ and $C_2 \in P_2$, $C_1, C_2 \subset X$. The Rand index counts the number of data objects that are in both clusters/classes $|C_1 \cap C_2|$ plus the data objects that are in neither cluster/class: $|X \setminus (C_1 \cup C_2)|$. The data sets in this benchmark contain many clusters, which means that the likelihood that two data objects are in neither cluster/class is very high so that the second term is always very large for any pair of clusters/classes. Therefore, the algorithm would produce very similar index values for two partitions, independently of the similarity of the partitions. There are ways to correct for these biases, see for example

[Vinh et al., 2009]. I decided to use the F_1 index in this work because it does not suffer from similar difficulties as the rand index and is optimally suited for the investigation, presented in the next chapter.

4.3 INTERNAL CLUSTER QUALITY MEASURES

Internal cluster quality measures are used to estimate the quality of a (fuzzy) partitioning (cluster algorithm results) without knowing the true partitioning (class information). The term 'internal' refers to the fact that no external information (i.e. class information) is used for the quality assessment. This situation is very common in reality because clustering algorithms would not be needed if the result is already known. Besides visual inspection, internal quality measures provide one of very few possibilities to determine whether or not a clustering algorithm was successful in partitioning a data set. The reliability of these measures in high-dimensional data sets is investigated using this benchmark. This investigation then concludes the thesis questions with an answer to Q_4.

The first indication for the quality of a partitioning may be provided by the objective function value of a prototype clustering algorithm. Since the value of the objective function is to be minimised (or maximised in case of EMGMM), the lowest (highest) value for a given algorithm indicates the best clustering quality if comparing results from the same algorithm. However, the value of the objective function is subjective to the algorithm and cannot be used to quantify the quality of a variety of clustering algorithms, or even the same algorithm with different parameters (e.g. the fuzzifier).

To compare the quality of (fuzzy) partitionings, generated by different clustering algorithms, a clustering algorithm independent approach is required. Some of the quality measurements listed in the subsections below are based on the abstract model assumption, that data objects of one cluster shall be as similar as possible while data objects of different clusters should be as different as possible. This abstract concept is quantified by using two values: cluster diameter and cluster separation. A small cluster diameter and a large cluster separation is considered to be good because this indicates well identified clusters with little overlap.

The diameter and separation of clusters can be evaluated on data object level by computing distance values for each pair of data objects. Alternatively, the locations of the prototypes can be used instead, which is of course only possible if a prototype based clustering algorithm is used. Both approaches are presented in the next subsection, even though only the second option is used for benchmarks.

4.3.1 *Cluster Diameter and Cluster Separation*

Let $Y = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset \mathbb{R}^m$ be a set of data objects, $U \in [0,1]^{c \times n}$ be a (fuzzy) membership matrix with the usual restriction of $1 = \sum_{i=1}^{c} u_{ij}$ and let $Y = \{\vec{y}_1, \ldots, \vec{y}_c\} \subset \mathbb{R}^m$ be a set of prototypes. For pairwise data object accessing, the diameter of cluster i is defined by

$$D_i^2 = \frac{1}{\left(\sum_{j=1}^{n} u_{ij}\right)^2} \sum_{j=1}^{n} \sum_{l=1}^{n} u_{ij} \cdot u_{il} \cdot d^2(\vec{x}_j, \vec{x}_l)$$

and for prototype centric diameter estimation, the diameter is defined by

$$D_i^2 = 4 \frac{1}{\sum_{j=1}^{n} u_{ij}} \sum_{j=1}^{n} u_{ij} \cdot d^2(\vec{x}_j, \vec{y}_i) \tag{4.3}$$

The factor 4 originates from the squared distance from the prototype to a data object, which corresponds to the radius instead of the diameter: $4r^2 = (2r)^2 = d^2$. As can be seen from the definition, in case of pairwise data object accessing, the runtime complexity to compute all cluster diameters is in $\mathcal{O}(n^2 \cdot c)$, which is too expensive for the benchmark, discussed below. The alternative, prototype centric diameter estimation (Equation (4.3)), is in $\mathcal{O}(n \cdot c)$, which is much more practical. Note that squared distances are used here because most clustering algorithms minimise the sum of squared distances between prototypes and data objects.

Determining the cluster separation is again possible for pairwise data object accessing and for prototype centric estimation. The separation of cluster i and cluster k is defined for pairwise data object accession by

$$S_{ik}^2 = \frac{1}{\left(\sum_{j=1}^{n} u_{ij}\right) \cdot \left(\sum_{l=1}^{n} u_{kl}\right)} \sum_{j=1}^{n} \sum_{l=1}^{n} u_{ij} \cdot u_{kl} \cdot d^2(\vec{x}_j, \vec{x}_l)$$

For prototype centric separation, the value is simply the distance between prototypes i and k.

$$S_{ik}^2 = d^2(\vec{y}_i, \vec{y}_k)$$

As can be seen from the definition, in case of pairwise data object accession, the runtime complexity is in $\mathcal{O}(n^2 \cdot c^2)$, which is again too expensive for the benchmark. The alternative, prototype centric diameter estimation is in $\mathcal{O}(c^2)$ and is used instead.

4.3.2 *Bezdek Separation Index*

The Bezdek separation index [Bezdek et al., 1997] is a variation of the the separation index, introduced by Dunn [Dunn, 1973]. The separation index by Dunn is only applicable on crisp partitions, which is insufficient for this work. Bezdek extended the index to work with fuzzy membership values, using the cluster diameter and separation. Let X, Y, U, D and S be defined as in the last subsection, then the Bezdek separation index I_{BS} is the ratio of the smallest cluster separation to the largest cluster diameter.

$$I_{BS} = \frac{\min\limits_{i,k=1,\dots,c} S_{ik}}{\max\limits_{i=1,\dots,c} D_i} \tag{4.4}$$

The index is viable whether the membership matrix is crisp or fuzzy, it also is viable for pairwise data object assessing and prototype centric estimation of distance and separation values. The actual value of the index however, depends on these factors.

The Bezdek separation index is a direct conclusion from the abstract clustering goal of minimizing the cluster diameter and maximizing the cluster separation. When comparing partitionings with the Bezdek separation index, larger values for I_{BS} stand for better clustering quality.

4.3.3 *Davies-Bouldin Index*

The Davies-Bouldin index I_{DB} [Davies and Bouldin, 1979] is defined using the same separation and diameter definition as the Bezdek separation index. The equation however does not only take the absolute extremes into account but builds an average over all clusters.

$$I_{BS} = \frac{1}{c} \sum_{i=1}^{c} \max_{\substack{k=1,\dots,c \\ k \neq i}} \frac{D_i + D_k}{2S_{ik}} \tag{4.5}$$

The index puts the radii (i.e. half their diameter) of two clusters (i and k) into relation of their separation. The mean of the largest values of these relationships is the index value. In other words, for each cluster the Davies-Bouldin index selects the other cluster, that has the worst relative separation and builds the mean over all clusters.

Again, this index is derived from the abstract notion of clustering. However in this case, a smaller value indicates a higher clustering quality.

4.3.4 *Xie-Beni Index*

The Xie-Beni index [Xie and Beni, 1991] is different from the two indices above.

$$I_{XB} = \frac{\frac{1}{n}\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{\omega} d^2(\vec{y}_i, \vec{x}_j)}{\min_{i,k=1,\ldots,c} S_{ik}} \tag{4.6}$$

As can be seen from the equation, it puts the relative objective function value of FCM into relation to the minimal separation of clusters. If this index is to be used to compare different prototype based clustering algorithms, a fixed value for ω needs to be selected and the prototype location and (fuzzy) membership values of the original clustering algorithm are used to calculate the value of I_{XB} as described in Equation (4.6). Whether or not this strategy is successful is analysed in Section 5.3.

There is however a good reason to consider the Xie-Beni index. Both, the Bezdek separation index and the Davies-Bouldin index have a fundamental problem because they do not take into account that some of the classes might be located close together or might even overlap. When determining the quality of a clustering process based on the model approach alone, an accurate covering of the classes might result in a lower score than a non-optimal clustering. The Xie-Beni index does not purely rely on the abstract clustering model approach, which might give it an advantage in real applications.

For the Xie-Beni index, smaller values indicate a better clustering.

4.3.5 *Normalised Partition Coefficient Index*

The normalised partition coefficient (NPC) index [Backer and Jain, 1981] is only useful for fuzzy clustering algorithms. For a membership matrix U, the partition coefficient index is defined by the average of the sum of squared membership values of the data objects.

$$I_{PC} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c} u_{ij}^2 \tag{4.7}$$

Due to the definition of membership values $1 = \sum_{i=1}^{c} u_{ij}$, the inner sum of equation (4.7) is bound to $\frac{1}{c} \leqslant \sum_{i=1}^{c} u_{ij}^2 \leqslant 1$. The minimum value is reached for $u_{ij} = \frac{1}{c}$, as $\sum_{i=1}^{c} u_{ij}^2 = \sum_{i=1}^{c} \frac{1}{c^2} = \frac{c}{c^2} = \frac{1}{c}$ for all data objects $j = 1, \ldots, n$. This averages to an overall value of $\frac{1}{n}\sum_{j=1}^{n} \frac{1}{c} = \frac{1}{n} \cdot \frac{n}{c} = \frac{1}{c}$. To gain a value, independent of the number of clusters, the index must be normalised, resulting in the normalised partition coefficient.

$$I_{NPC} = 1 - \frac{c}{c-1}(1 - I_{PC}) \tag{4.8}$$

Higher values indicate a more crisp clustering result for both, the normalised and the not-normalised partition coefficient. A better clustering result supposedly produces more crisp assignments to clusters because the algorithm can clearly identify the cluster to which a data objects belongs. For a perfectly crisp result, the normalised partition coefficient always yields a value of 1 which means that the index cannot be used to assess the quality of a crisp clustering algorithm.

4.3.6 *Normalised Partition Entropy Index*

The normalised partition entropy (NPE) index [Borgelt, 2005] is very similar to the partition coefficient index. It is related to Shannons definition of entropy [Shannon, 1948] in information theory. For a discrete random variable X with strictly positive discrete probability distribution, defined as $\vec{u} = \{u_1, \ldots, u_c\}$, with $1 = \sum_{i=1}^{c} u_i$, the Shannon entropy of $\vec{u}$ is defined as

$$S(u) = -\sum_{i=1}^{c} u_i \cdot \log_2(u_i) \tag{4.9}$$

In information theory, the Shannon entropy defines the amount of information in a message. In this case, the message that is transmitted is the cluster index which has an alphabet of c different characters and the membership values $\vec{u}$ define the probability that they are received over a communication channel. The Shannon entropy specifies how many bits in average have to be transmitted in order to define the state of X.

Please do not confuse this with the wrong statement that fuzzy sets are the same as probability distributions. They are not because the integral of a fuzzy set does not need to be 1.

In case of clustering, consider one data object $\vec{x}$. Just like for the expectation maximization clustering algorithm, the membership values are interpreted as the conditional probabilities of observing the classes given that $\vec{x}$ was observed. The Shannon entropy then gives a measure for the predictability of the clustering result. A good predictability (low Shannon entropy value) is interpreted as a good clustering result.

The partition entropy for clustering is defined as:

$$I_{PE} = -\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c} u_{ij} \cdot \log_2(u_{ij}) \tag{4.10}$$

Similar to the partition coefficient, the partition entropy can be normalized to avoid a dependence on the number of clusters in the data set.

$$I_{NPE} = -\frac{1}{n\log_2(c)}\sum_{j=1}^{n}\sum_{i=1}^{c} u_{ij} \cdot \log_2(u_{ij}) \tag{4.11}$$

The factor $\frac{1}{\log_2(c)}$ is multiplied by Equation (4.10) because the extreme case of all membership values being equal to $\frac{1}{c}$ yields the

maximal value for the partition entropy. The infimum value of the index is 0, but can not be achieved because it would require crisp membership values. The crisp clustering case is not covered by the index directly because the logarithm is not defined for 0. However for small positive values the value of the term inside the sum approaches $0 = \lim_{x \to 0} x \cdot \log_2(x)$ so that it is easy to extend the index can to crisp membership values. Like the partition coefficient, the partition entropy index is not useful for evaluating the quality of crisp clustering algorithms, because its value is always 0.

A small value for normalised partition entropy index indicates more crisp clustering result, which is again usually considered to be better than a low crispness.

4.3.7 *Internal Indices and the Noise Cluster*

The NPC and NPE indices can easily be extended to handle the noise cluster if it is considered to be a normal cluster. The three other indices however (the BS, DB and XB indices), depend on the location and size of clusters and cannot easily handle the noise cluster because it has no location and no natural size. In practice, this can introduce a bias if a noise clustering algorithm is assessed with these indices. To avoid such problems, the noise cluster can be considered to have a constant separation with all clusters: $S^2_{i,\,\text{noise}} = 2 \cdot d_{\text{noise}}$ and a diameter of $D_{\text{noise}} = 2 \cdot d_{\text{noise}}$ as well.

These values are motivated by the notion, that all data objects have a distance of d_{noise} to the noise cluster. Therefore, if the noise cluster had a centre, all data objects would be located on a sphere surface with radius d_{noise} around the centre. With the same argument, the noise cluster can be noted to have a distance of $2 \cdot d_{\text{noise}}$ to all other clusters because any data object that is located at a distance of d_{noise} from one of the true clusters would have identical membership values to the real cluster as to the noise cluster. Therefore, the noise cluster can be considered to be located $2 \cdot d_{\text{noise}}$ away from all other clusters. With this rather artificial extension of the indices, all validation indices can be used to compare the quality of normal and noise clustering algorithms.

4.4 BENCHMARK SETUP

The benchmark is based on a statistical analyses of the quality of (fuzzy) partitionings that are created by the clustering algorithms. A statistical approach is necessary to prevent dependencies such as initialization of prototypes, overlapping of classes and others. This is achieved by repeating a specific set-up configuration (i.e. data set family, number of dimensions, number of classes) multiple times. The

construction of the data sets for the data set families is described in Subsection 4.4.1. The parameters of the algorithms and the initialization strategy are presented in subsection 4.4.2.

The benchmark is performed using EDMOAL[1] which is discussed in Appendix B in more detail. Even though EDMOAL contains all the relevant algorithms (i.e. clustering, data generation, quality indices etc.) for this work, it does not include the routines that are used to organize their application and generate the output files.

4.4.1 *Data Sets, Dimensions and Number of Classes*

For the data set families D_1, D_2 and D_3, data sets for 11 different dimensions $m \in \{2, 3, 5, 7, 10, 15, 20, 30, 504, 70, 100\}$ are generated. The data set family D_4 is not well suited for low dimensions because there are simply not enough different data objects possible. Thus only data sets in $m \in \{10, 15, 20, 30, 50, 70, 100\}$ dimensions are created for D_4. For all data set families and for each dimension number, a pool of 250 classes are generated, using the mechanism described for the individual data set families (see Section 4.1). Within each data set family and for all numbers of dimensions, data sets with 20 different counts of classes $c \in \{2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 25, 30, 40, 50, 60, 80, 100, 120, 150, 200\}$ are used. To allow for a statistical analysis of the results, 20 data sets for each combination of data set family, number of dimensions and number of classes are generated. The quadruple (D_k, m, c, s) points to a single data set, with $D_k \in \{D_1, D_2, D_3, D_4\}$ denoting the data set family, m and c the number of dimensions and classes respectively, subject to the above mentioned lists of values and $s \in \{1, 2, \ldots, 20\}$ being the index of the data set. Each individual data set is constructed using a random selection of the 250 previously generated classes, plus the noise data objects.

The total computation time on a 6-core Intel i7 (12 logical cores) was roughly 80 days. This type of experiment would simply not be possible for algorithms with a runtime complexity in $\mathcal{O}(n^2)$ or worse.

In total, there are $(3 \cdot 11 + 7) \cdot 20 \cdot 20 = 16,000$ different data sets. Since all algorithms depend on the initialization of their prototypes, each data set is clustered 5 times by each algorithm with different initialization values. To each configuration of data sets plus initialization, all 10 algorithms are applied independently, which results in a total number of 800,000 individual experiments for this benchmark.

In fact, 1,000,000 experiments are conducted for a total of 20,000 data sets.

In Section 2.4, a test for distance concentration was proposed. During generation of all data sets, I executed a gradient descend algorithm to find the maximal relative variance of a data set. I decided to not include these results in this work because the limitations I imposed on the calculations to keep the runtime manageable rendered the results useless. The imposed limit was, that only one gradient descending run was used. However, the gradients were too flat to allow for a fast convergence of the algorithm, which lead to long computation times. As a result, the data was not very useful. It is certainly

I generated data sets with 7, 17, 35, 70 and 170 classes but decided later not to present them within this work.

I found the increment between two succeeding cluster counts should not decrease. For example the sequence 12, 15, 17, 20 seems to be inconsistent because the gaps are 3, 2, 3.

1 https://github.com/Roland-Winkler/EDMOAL

possible to find a much better estimate by using a swarm algorithm [Kennedy and Eberhart, 1995] and a clever step-size for the gradient method. This might be an interesting project for future investigations.

4.4.2 *Clustering Algorithms*

All 10 algorithms presented in the last chapter are used in the benchmark, that are HCM, FCM_2, $NFCM_2$, FCM_m, $NFCM_m$, PFCM, PNFCM, RCFCM RCNFCM and EMGMM. Since the number of classes in a data set is known, this parameter does not need to be estimated, always the correct number of clusters is chosen. The noise distance was estimated by using the classes prior to assembly of the data sets. Let $C_1, \ldots, C_{250}$ be the classes of one of the data set ranges $(D_k, m, \cdot, s)$. The noise distance d_{noise} for all algorithms, performed on all data sets of the range $(D_k, m, \cdot, s)$ is defined by

$$d_{noise} = \max_{i=1\ldots250} \max_{\vec{x}\in C_i} d(\vec{x}, \vec{\mu}_i) \tag{4.12}$$

with $\vec{\mu}_i$ being the arithmetic mean of the class: $\vec{\mu}_i = \frac{1}{|C_i|} \sum_{\vec{x}\in C_i} \vec{x}$. For this benchmark, it is better to overestimate the noise distance rather than to underestimate it, so using the largest value of all these distances was reasonable. To avoid prototypes getting 'lost' (see Section 3.7.2), an initial maximal noise distance of $d_{maxnoise} = 10 \cdot d_{noise}$ with a reduction parameter $\alpha = 0.2$ and a minimal number of iterations of 10 for applying an algorithm is used.

Algorithm	Parameters
HCM	
FCM_2	$\omega = 2$
$NFCM_2$	$\omega = 2$
FCM_m	$\omega = 1 + \frac{1}{m}$
$NFCM_m$	$\omega = 1 + \frac{1}{m}$
PFCM	$\beta = \frac{1}{2}$
PNFCM	$\beta = \frac{1}{2}$
RCFCM	$\eta = 1 - \frac{1}{m}$
RCNFCM	$\eta = 1 - \frac{1}{m}$
EMGMM	$\sigma_{min} = 0.001$ and $\sigma_{max} = 100$

Table 4.1: Individual parameters of the algorithms, used in the benchmark.

The number of iterations is bounded to be 30, which is slightly on the low side, but is sufficient for the benchmark in most cases. A test showed that after approximately 25 to 30 iterations, improvements of the clustering quality was negligible, even though the termination threshold was not yet reached. As described in Section 3.7.5, the maximum movement distance is used for terminating the iteration process early. The data objects are mainly located in the unit-hypercube,

$$
\begin{aligned}
D_k &\in \{D_1, D_2, D_3, D_4\} \\
m &\in \begin{cases} \{2,3,5,7,10,15,20,30,50,70,100\} & \text{if } D_k \in \{D_1, D_2, D_3\} \\ \{10,15,20,30,50,70,100\} & \text{if } D_k = D_4 \end{cases} \\
c &\in \{2,3,4,5,6,8,10,12,15,20,25,30,40,50, \\
&\qquad 60,80,100,120,150,200\} \\
s &\in \{1,\ldots,20\} \\
r &\in \{1,\ldots,5\} \\
A &\in \{\mathrm{HCM}, \mathrm{FCM}_2, \mathrm{NFCM}_2, \mathrm{FCM}_m, \mathrm{NFCM}_m, \\
&\qquad \mathrm{PFCM}, \mathrm{PNFCM}, \mathrm{RCFCM}, \mathrm{RCNFCM}, \mathrm{EMGMM}\}
\end{aligned}
$$

Table 4.2: Setup combinations for data sets in the benchmark.

which makes a threshold value of $\varepsilon = 0.01$ reasonable. The other individual parameters for applying the algorithms are listed in table 4.1.

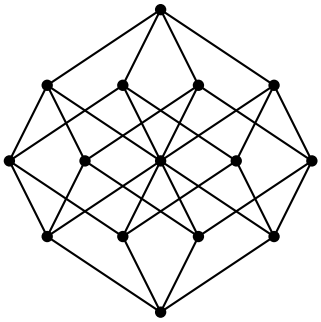
With all algorithms, each individual data set was clustered 5 times, with different randomly sampled initialization positions for the prototypes. To gain comparable results, all 10 algorithms are performed using the same initial positions in one set-up (data set and initialization position pair).

In summary, the tuple (D_k, m, c, s, r, A) describes a unique experiment with D_k being the data set family, m the number of dimensions, c the number of classes, s the data set index, r the index of the initial condition and A the algorithm. The values are subject to the values, presented in Table 4.2.

4.4.3 *Performing the Experiments*

Finally, after an experiment described by the tuple (D_k, m, c, s, r, A) was performed, the cluster quality indices are applied to measure the quality of the clustering result. That includes the F_1 measure, as well as the BS index, BD index, XB index, NPC index and NPE index. Some further information with less relevance is also stored. That includes the number of iterations the algorithm was running, its objective function value after clustering and a variant of the F_1 measure that defuzzifies the membership values first before calculating the index using the winner takes all strategy. I chose to not utilise the second F_1 measure because it did not give more insight.

All diagrams that are presented in the next chapter as well as in appendix chapters D, E and F are based on the benchmark, described in this chapter. The data that is used for generating the diagrams is stored in one large table, see Appendix C.4 for details.



5

EXPERIMENTAL RESULTS

In this chapter, the results of applying the clustering algorithms (see Chapter 3) on the benchmark data sets (see Section 4.1) are presented. The results, composed of the internal and external quality index values are prepared in various different ways to answer thesis questions Q_2, Q_3 and Q_4.

In the first section of this chapter, Section 5.1, the algorithms are analysed individually and their behaviour is described as well as interpreted to approach thesis question Q_3. To give a complete picture of how and why the algorithms behave as they do, some additional thoughts and arguments are necessary which are presented at the beginning of that section.

A direct comparison of the algorithms is presented in Section 5.2, which is required for thesis question Q_2. The diversity of the result does not allow a clear ranking of the algorithms, but the benchmark provides some insightful information on the behaviour of the algorithms in high-dimensional feature spaces. As already indicated in Section 4.3, it is often necessary to know the quality of a clustering result, meaning internal cluster quality indices are applied. The performance of these indices in high-dimensional feature spaces is investigated in Section 5.3, which is done to answer thesis question Q_4.

Only a small subset of all results can be presented and discussed here. A complete graphical representation is presented in Appendices D, E and F.

5.1 NUMBER OF DIMENSIONS AND CLUSTERS

In this section, thesis question Q_3 is approached: How is the clustering quality influenced by the number of dimensions and the number of clusters? Since high dimensional data sets can be very diverse, there cannot be an exhaustive answer to the question. Using the benchmark data sets, a trend for each algorithm is visible, which may help to understand the effects on other data sets. Each algorithm is analysed individually and the influences of dimensions and number of clusters is discussed. Please note, that the number of clusters always matches the number of classes in a data set. So it is possible

for the algorithm to find a near-perfect solution. As a consequence, the number of classes and clusters are both addressed by the symbol c.

The quality of the clustering result is measured using the F_1 quality index, presented in Section 4.2. The F_1 measure is applied, using the prototype centred distance calculations, both for the cluster diameter and the inter-cluster distances. The final prototype positions, as provided by the algorithms, are used for this purpose. Any conclusion, made in this chapter depends on the fact that the F_1 measure is appropriate to measure the quality of a clustering result w.r.t. the true result.

It is often useful to apply a clustering algorithm several times on the same data set and select the best result. This reduces the dependency on the initial conditions and increases the chances of finding the best possible clustering result. This strategy is done with the results of the benchmark data sets as well. Let $F_1(D_k, m, c, s, r, A)$ refer to the F_1 value of a specific clustering result. $D_k \in \{D_1, D_2, D_3, D_4\}$ selects the data set family, m the number of dimensions, c the number of classes, $s \in \{1, \ldots, 20\}$ the data set index, $r \in \{1, \ldots, 5\}$ the initialization and A the algorithm. Each data set (D_k, m, c, s) is clustered 5 times with algorithm A using different initializations. The F_1 index is calculated for all 5 clustering results and the maximal F_1 value is used to estimate the quality of the clustering result.

$$F_1(D_k, m, c, s, A) = \max_{r=1,\ldots,5} F_1(D_k, m, c, s, r, A) \tag{5.1}$$

As described in the last chapter, for each combination of data set family D_k, number of dimensions m and number of classes c, 20 data sets, indicated by $s = 1, \ldots 20$ are generated. This generates 20 maximal F_1 values for each algorithm A which are used to calculate the mean $\overline{F_1}$ and empirical standard deviation $\sqrt{\widehat{V}(F_1)}$ of the 20 values.

I calculated several more performance indicators besides the mean and the empirical standard deviation. But the others like the median did not reveal additional insight, so I chose not to present them.

$$\overline{F_1}(D_k, m, c, A) = \frac{1}{20} \sum_{s=1}^{20} F_1(D_k, m, c, s, A) \tag{5.2}$$

$$\widehat{V}(F_1(D_k, m, c, A)) = \frac{1}{19} \sum_{s=1}^{20} (F_1(D_k, m, c, s, A) - \overline{F_1}(D_k, m, c, A))^2 \tag{5.3}$$

The empirical standard deviation in this case is not a performance indicator as such, but since 20 values is quite small for a statistical analysis of the effect, it is necessary to have an indication how reliable the mean value is. The mean of the maximal F_1 values is used to estimate the clustering quality for a specific setup, which is presented visually in this section (see for example Figure 5.1) and Appendix D.

The upper image of Figure 5.1 shows in each cell the mean score of the 20 different experiments for all combinations of numbers of

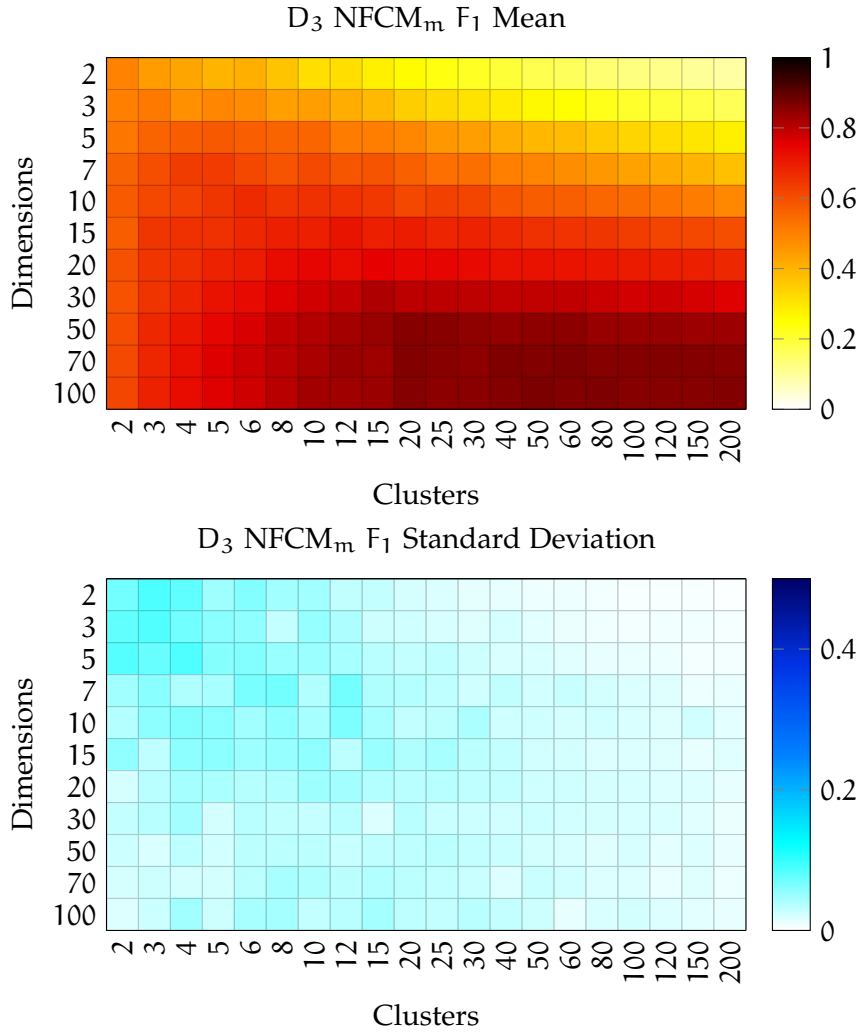


Figure 5.1: Example of the F_1 score plot. It shows the F_1 mean and standard deviation of data set family D_3, clustered with the $NFCM_m$ clustering algorithm.

dimensions $m \in \{2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$ and classes $c \in \{2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 25, 30, 40, 50, 60, 80, 100, 120, 150, 200\}$. The empirical standard deviation shown in the lower image is presented in the same fashion to see how much the score varies for the individual cells. A score of 1 is best for the F_1 measure and a score near 0 corresponds to a very bad clustering result (see Section 4.2.2). Since the standard deviation is less important, it is not discussed in this chapter any further, but it is still presented in Appendix D. The plots hold a lot of information, but most importantly, it presents a way to understand the limits of an algorithm in terms of dimensions and classes/prototypes.

The plot in Figure 5.1 is quite peculiar because it presents some unexpected result. In the upper panel, the clustering quality as measured by the F_1 measure is presented. As indicated by the colour, the

NFCM_m algorithm performs better with more dimensions and more classes, which is very counter intuitive since the clustering problem becomes harder the larger these values are. So in this particular case, the increase in cluster quality w.r.t. the number of dimensions can be explained with the structure of the D_3data set family. The data sets contain very likely overlapping classes, which can be handled better by the algorithm if the number of dimensions is higher- and the overall class overlapping is lower. The effect is much smaller for the D_2 data set family, which produces more compact classes (see Appendix D.2).

The increase in quality in between 5 to 20 classes and dimensions equal and above 50 can be explained with the influence of the noise cluster. It seems that the clustering quality is particularly bad because some classes are completely assigned to the noise cluster. With more classes in the data set, the relative number of classes that are assigned to the noise cluster becomes lower, which increases the F_1 score. When comparing with the non-noise variant of the algorithm (see Appendix D.3), the effect vanishes.

The lower panel of the figure shows the empirical standard deviation of the F_1 scores, which shows that the F_1 score does not vary much around the mean value. This means that the algorithm performs with a quite constant quality, even though the quality is not particularly good.

To understand the behaviour of algorithms better on a more fundamental level, a simple evaluation procedure is used. As already presented in the introduction in Figure 1.5, given a high-enough number of dimensions and a high-enough number of prototypes, all prototypes of FCM_2 run into the centre of mass (CoM) $\frac{1}{n}\sum_{j=1}^{n}\vec{x}_j$ of the data set $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$. So it seems the CoM has some special properties to it. Consider the data set $X_m = \{\vec{x}_1, \ldots, \vec{x}_m\} \subset \mathbb{R}^m$ consisting of the $n = m$ unit vectors of m-dimensional Euclidean vector space.

$$X_m = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \cdots, \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\}$$

The simplex is a geometrical form, not to be confused with the linear optimization simplex algorithm.

The data objects represent classes of equal weight and without any spatial expansion, the centre of mass of X_m is $\vec{x}_{\text{CoM}} = (\frac{1}{m}, \ldots, \frac{1}{m})$. Note that the data set forms a simplex which is embedded in an $m-1$-dimensional linear subspace of the m-dimensional real vector space. Therefore, the clustering problem that is analysed with the simplex data set is related to an $m-1$-dimensional feature space. For

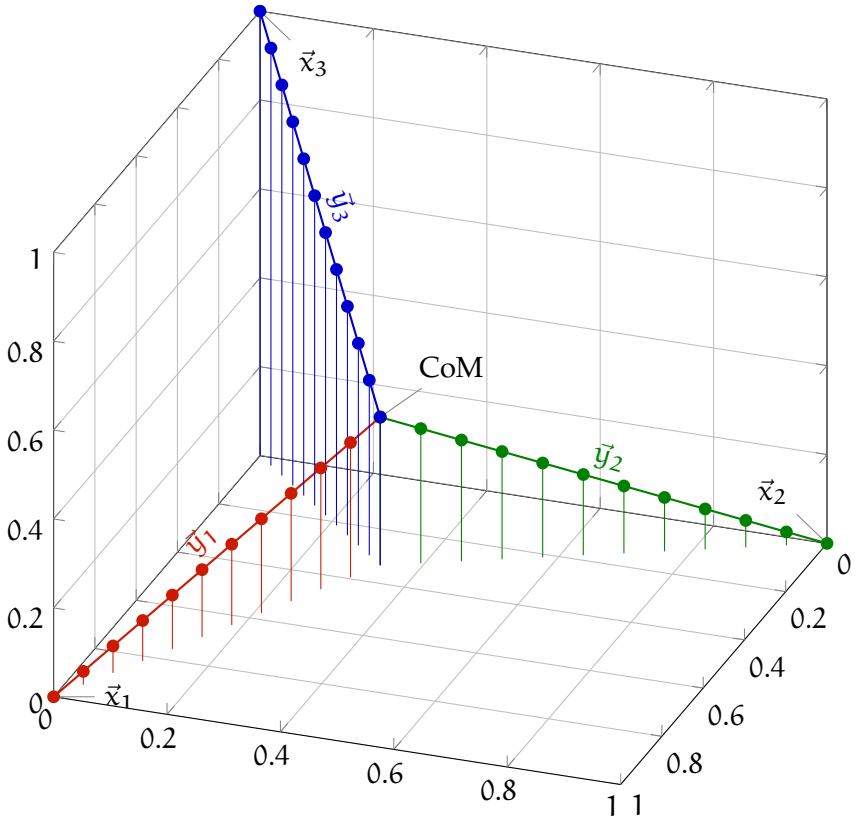


Figure 5.2: Path of the prototypes from the centre of mass to their respective data objects as a 3D plot. The colour indicates the prototype. All indicated vectors lye in a 2-dimensional plane, defined by unit vectors of $\vec{x}_1$, $\vec{x}_2$ and $\vec{x}_3$ which form a simplex.

each data object (class), a prototype is placed in the centre of mass, and than gradually moved to one of the data objects. Let $\alpha \in [0, 1]$ be a variable to control the location of the prototypes $\vec{y}_1, \ldots, \vec{y}_m : [0, 1] \to \mathbb{R}^m$:

$$\vec{y}_i(\alpha) = \alpha \cdot \vec{x}_i + (1 - \alpha) \cdot \vec{x}_{CoM}$$

For a prototype based clustering algorithm, the prototypes are gradually moved from the centre of mass to their respective data objects. The membership values for the data objects are calculated according to the update equations while the objective function values are computed and plotted in a graph. In Figure 5.2, the process of moving the prototypes from the CoM to their respective data objects is visualised for $m = 3$. Since the location of the prototypes is determined by the parameter α, the membership values and the objective function values of the clustering algorithms from chapter 3 become functions of α as well. To compare graphs of data sets with different dimensionality for an objective function J, the graph values are normalised to have value 1 at $\alpha = 0$ in all cases:

$$J(X, \alpha) = \frac{J(X, Y(\alpha), U(\alpha))}{J(X, Y(0), U(0))} \tag{5.4}$$

See Figure 5.5 which shows the use of the simplex evaluation setup.

5.1.1 *Hard C-Means*

The clustering quality of the HCM algorithm depends strongly on the initialization of the prototypes, see for example [Bubeck et al., 2012; Kuncheva and Vetrov, 2006]. After initialization, it is very likely that one prototype is closer to multiple classes than any other prototype. Because a data object is always assigned to the closest prototype, the data objects from these classes are effectively hidden from all other prototypes. Once that happened, it is very unlikely that any other prototype gets near enough to reveal one of the hidden classes. Therefore, the initialization process dominates how the data set is split between the clusters and often several classes are covered by the same cluster. The remaining classes are then covered by too many clusters, which makes it likely that some classes are split into multiple clusters. Also, it is likely that some clusters cover only noise data objects if the data set contains them.

This effect depends on the number of clusters/classes and also slightly on the number of dimensions. Observe Figure 5.3, which shows the clustering quality of HCM on data set family D_1. The quality of the clustering with HCM is only acceptable for $c = 2$ clusters. For more than 2 clusters, the quality is too low to be considered useful. There is also a slight decrease in clustering quality with increasing number of dimensions but this is only a second order effect. Since the quality for a small number of dimensions is already low, HCM cannot be considered to be a good clustering algorithm when using random initialization, especially because data sets in D_1 are supposed to be very easy to cluster.

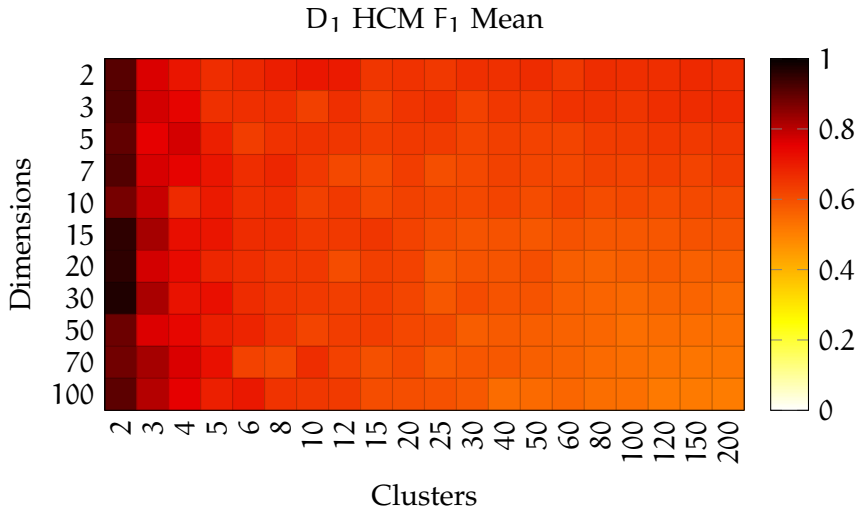


Figure 5.3: HCM cluster quality on data set family D_1.

5.1.2 *Fuzzy c-Means with $\omega = 2$*

Compared to HCM, the situation is very different for FCM_2: as long as a prototypes location is not identical to a data object, the data object is not hidden from any other prototype. As presented in the introduction in Figure 1.5, FCM_2 is almost useless on high-dimensional data. The effect does not only depend on the number of dimensions, but also on the number of classes/clusters, as can be observed in Figure 5.4. The interpretation of this effect however, is quite a bit more difficult than with HCM.

I found it very surprising that the curse of dimensionality effect produces such a sharp transition between FCM_2 being useful to being useless.

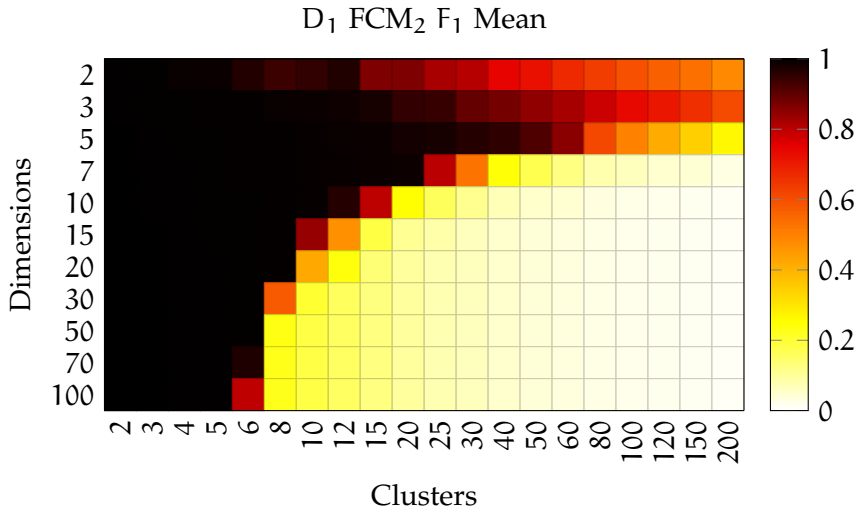


Figure 5.4: FCM_2 cluster quality on data set family D_1.

First, the drop in cluster quality on the top-right corner of Figure 5.4 can be explained with overlapping and in general closely packed classes. Since many classes are located in a very small space, the classes are not well enough separated.

The drop in clustering quality in the lower-right half of the plot however is connected to the curse of dimensionality. The interesting question is, what happens there and why?

As can be seen from Figure 1.5 in the introduction, all prototypes go into the centre of mass and therefore, the clustering quality is next to 0. The obvious explanation is, that there must be a (local) minimum of the objective function J_{FCM} for FCM_2 in the centre of mass. This can be tested by observing the objective function in the simplex data set test, presented above. In Figure 5.5, the simplex procedure objective function value plots for dimensions $m \in \{2, 3, 5, 10, 20, 50, 100, 200, 500, 1000\}$ show clearly, that there is a local minimum in the CoM at $\alpha = 0$. As the number of dimensions and prototypes increases, the local minimum at $\alpha = 0$ expands in spatial direction and becomes stronger. Or in other words, the maximum becomes higher and moves further to the right (sits at larger values for α).

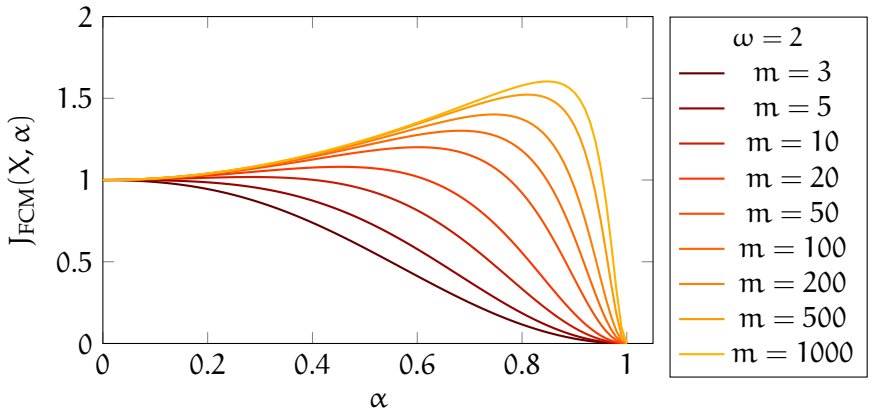


Figure 5.5: The objective function value of FCM_2 J_{FCM}, plotted depending on the location variable α within the simplex test-data set, see Equation (5.4).

Of course in this example, the number of prototypes/clusters is connected to the number of dimensions. The effect is studied in [Winkler et al., 2011a] in more detail, also for independent number of dimensions and clusters/classes. We determined, that the expansion of the minimum depends largely on the number of dimensions while the depth of the minimum increases with the number of prototypes. In other words, with increasing dimensionality, prototypes run into the centre of mass even if they are initialised far away from it.

A geometric explanation can be given for this effect. The hypervolume of the feature space where prototypes would not fall into the CoM becomes smaller with increasing dimensionality. This hypervolume of useful initialization positions is located near the centres of the classes and becomes very small compared to the hypervolume of the entire feature space. This in turn means, that the likelihood of (randomly) initializing prototypes close to the classes becomes tiny, rendering FCM_2 useless in high-dimensional feature spaces. With increasing number of prototypes, the minimum gets deeper with the consequence that it is less likely that local features of the data set can counter the curse of dimensionality.

So the next question is: why is there a local minimum in the centre of mass of high-dimensional data sets for FCM_2? The answer to this question lies in the update equations of FCM_2, combined with the concentration of distances, discussed in Section 2.1. Let the number of dimensions m be large, say $m >> 10$ and the number of classes/clusters c be larger than m, $c > m$, and let the classes be not located in a linear subspace of the feature space $\mathbb{R}^m$. Let $\vec{x}_j$ be a data object and $\vec{y}_1, \ldots, \vec{y}_c$ be randomly located prototypes with independently sampled dimensions. Then from Equation (2.16) follows, that the distance to all prototypes are approximately equal: $d_{1j} \approx \cdots \approx d_{cj} \approx d_{const}$. Or in words, from the point of view of a data object, all prototypes are approximately equally far away. This

implies, that the data object is approximately equally shared between all clusters.

$$u_{ij} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{\omega-1}}}{\sum_{k=1}^{c}\left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}} = \frac{\frac{1}{d_{ij}^2}}{\sum_{k=1}^{c}\frac{1}{d_{kj}^2}} \approx \frac{\frac{1}{d_{const}^2}}{\sum_{k=1}^{c}\frac{1}{d_{const}^2}} = \frac{1}{c} \tag{5.5}$$

As a consequence, the prototype update equation (3.10) yields roughly the same result for all prototypes.

$$\begin{aligned}\vec{y}_i &= \frac{\sum_{j=1}^{n} u_{ij}^{\omega}\vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{\omega}} = \frac{\sum_{j=1}^{n} u_{ij}^{2}\vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{2}} \\ &\approx \frac{\sum_{j=1}^{n}\left(\frac{1}{c}\right)^2\vec{x}_j}{\sum_{j=1}^{n}\left(\frac{1}{c}\right)^2} = \frac{\left(\frac{1}{c}\right)^2\sum_{j=1}^{n}\vec{x}_j}{\left(\frac{1}{c}\right)^2\sum_{j=1}^{n}1} = \frac{1}{n}\sum_{j=1}^{n}\vec{x}_j\end{aligned} \tag{5.6}$$

Which is by definition the centre of mass of the data set. In this way, the concentration of distances causes the collapse of FCM_2 in high-dimensional spaces.

Introducing the noise cluster does not change much of this behaviour. If the noise distance d_{noise} is smaller than d_{const}, all data objects are primarily assigned to the noise cluster, while for $d_{noise} > d_{const}$, the noise cluster has hardly any effect on the clustering process at all. Therefore, the behaviour of $NFCM_2$ is very similar to FCM_2 in high-dimensional spaces.

Please note, that Equations (5.5) and (5.6) hold true for any value of ω. However, for $\omega > 2$, the approximation is valid for larger differences in distances while for $\omega < 2$, the distance approximation must be more accurate for the effect to occur. This is the reason to consider FCM_m, discussed in the next subsection.

5.1.3 *Fuzzy c-Means with* $\omega = 1 + \frac{1}{m}$

In [Winkler et al., 2011a], we determined that the local minimum in the centre of mass of a data set can vanish if the fuzzifier ω of FCM is chosen close to 1, depending on the number of dimensions. The idea led to studying FCM_m and its noise related counterpart $NFCM_m$, with the fuzzifier being defined as $\omega = 1 + \frac{1}{m}$, which is slightly more crisp than suggested in our paper. The effect can be observed in Figure 5.6, where FCM_m is studied in the same way as FCM in Figure

5.5. As it can be seen, the local minimum at the centre of mass vanishes, which means, FCM_m should not be effected in the same way as FCM_2 in high-dimensional feature spaces. The sharp drop-off of objective function values for $m = 200$, $m = 500$ and $m = 1000$ in Figure 5.6 are numerical artefacts that can limit the use of FCM_m. Due to the fuzzifier of $\omega = 1 + \frac{1}{m}$, the update for the membership values becomes

$$u_{ij} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{\omega-1}}}{\sum_{k=1}^{c}\left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{1+\frac{1}{m}-1}}}{\sum_{k=1}^{c}\left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{1+\frac{1}{m}-1}}} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{m}}{\sum_{k=1}^{c}\left(\frac{1}{d_{kj}^2}\right)^{m}} \quad (5.7)$$

With m being very large, the value d_{kj}^{-2m} in Equation (5.7) can become arbitrary close to 0 for distance values below 1. Therefore, FCM_m becomes numerically unstable for a very high number of dimensions. A limit of ω to a minimum of 1.01 for double precision data seems reasonable.

When observing the performance of FCM_m in the benchmark data set D_1, the clustering process is quite good, which indicates that the dimension dependent problem vanished, see Appendix D. Only for dimensions close to 100 and a high number of classes/clusters, close to 200, the clustering quality reduces slightly. Even on the more complex data set D_3, with the distorted clusters, the algorithm performs quite well.

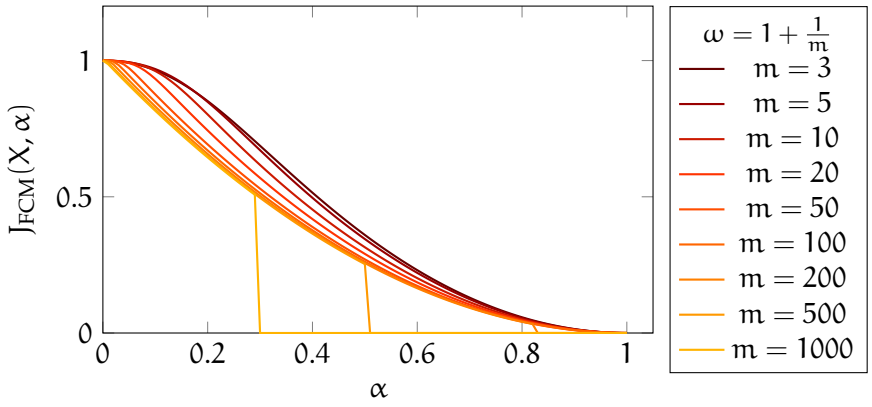


Figure 5.6: The objective function value of FCM_m J_{FCM}, plotted depending on the location variable α and a dimension dependent fuzzifier $\omega = 1 + \frac{1}{2}$ within the simplex test-data set.

The interesting question is, why does a fuzzifier approaching 1 prevents the breakdown of FCM for a high number of classes and dimensions $m >> 5$? The answer again, can be found in the update

equations. This time, let $\omega = 1 + \varepsilon$ be close to 1, with $\varepsilon = \frac{1}{m}$ and starting again from Equation (3.9).

$$u_{ij} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{\omega-1}}}{\sum_{k=1}^{c}\left(\frac{1}{d_{kj}^2}\right)^{\frac{1}{\omega-1}}} = \frac{\frac{1}{d_{ij}^{\frac{2}{\varepsilon}}}}{\sum_{k=1}^{c}\frac{1}{d_{kj}^{\frac{2}{\varepsilon}}}} \tag{5.8}$$

Let again $d_{ij} \approx d_{const}$, this time however the exponent is different, which means the approximation of the entire equation is not possible in the same way as for $\omega = 2$. Let δ_{ij} be the residual of the approximation, with $d_{ij} = d_{const} + \delta_{ij}$, then with the Bernoulli inequality [Bronstein and Semendjajew, 1979] (page 293) $(1+x)^r \geqslant 1 + r \cdot x$ follows:

$$\left(\frac{d_{ij}}{d_{const}}\right)^{\frac{2}{\varepsilon}} = \left(1 + \frac{\delta_{ij}}{d_{const}}\right)^{\frac{2}{\varepsilon}} \geqslant 1 + \frac{2}{\varepsilon} \cdot \frac{\delta_{ij}}{d_{const}} \tag{5.9}$$

The curse of dimensionality for FCM_2 is problematic, if $(\frac{d_{ij}}{d_{const}})^{\frac{2}{\varepsilon}} \to 1$, but if ε is small enough, so that $\frac{2}{\varepsilon} \cdot \frac{\delta_{ij}}{d_{const}} >> 0$, the term $(\frac{d_{ij}}{d_{const}})^{\frac{2}{\varepsilon}}$ cannot approach 1. This way, the approximation of Equation (5.5) is not valid for $\omega = 1 + \frac{1}{m}$. Therefore, the membership values give a sufficient contrast, they do not approach $\frac{1}{c}$ and the prototypes are not pulled into the centre of mass. The same argument can be used to explain, why the prototypes of HCM are not pulled into the centre of mass: in case of HCM, ε would be 0. This analysis also applies to $NFCM_m$ which is why it does not show significant problems either.

5.1.4 *Fuzzy c-Means with Polynomial Fuzzifier Function*

PFCM can be seen as a combination of HCM and FCM_2. As pointed out already in Section 3.4, there is an area of crisp data object assignment, close to the prototypes. This crisp assignment hides data objects that are much closer to the one prototype from all other prototypes, so that far away prototypes are not effected by these data objects. As a consequence, if a prototype is close enough to a class in such a way that all data objects belonging to this class are crisply assigned, no other prototype is pulled into the direction of this class. This effectively reduces the number of classes in the data set, as seen from the other prototypes. Assume for a moment, that there are c classes in the m-dimensional feature space $\mathbb{R}^m$, located at some (random) location in the feature space. If $c <= m$, the class centres are located in a $c-1$-dimensional linear subspace $\mathbb{R}^{c-1}$ of $\mathbb{R}^m$. So if the prototypes in PFCM potentially hide classes from the other prototypes, it has the potential to reduce the intrinsic number of dimensions of the data set.

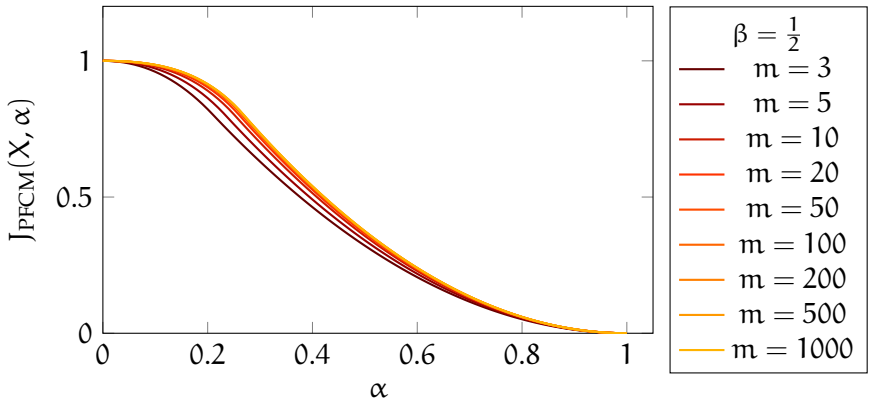


Figure 5.7: The objective function value of PFCM J_{PFCM}, plotted depending on the location variable α within the simplex test-data set.

Given this insight, it is not surprising, that P(N)FCM performs reasonably well on high-dimensional data spaces. It does not suffer from the same difficulties as HCM, because the area of crisp assignment is often small enough to prevent a covering of multiple classes. At the same time, because of the crisp assignment, the prototypes are not drawn into the centre of mass of the data set. As was shown in [Winkler et al., 2012], the algorithm creates only a very weak local minimum in the centre of mass, see Figure 5.7. Therefore, it can be expected to be stable in high-dimensional feature spaces.

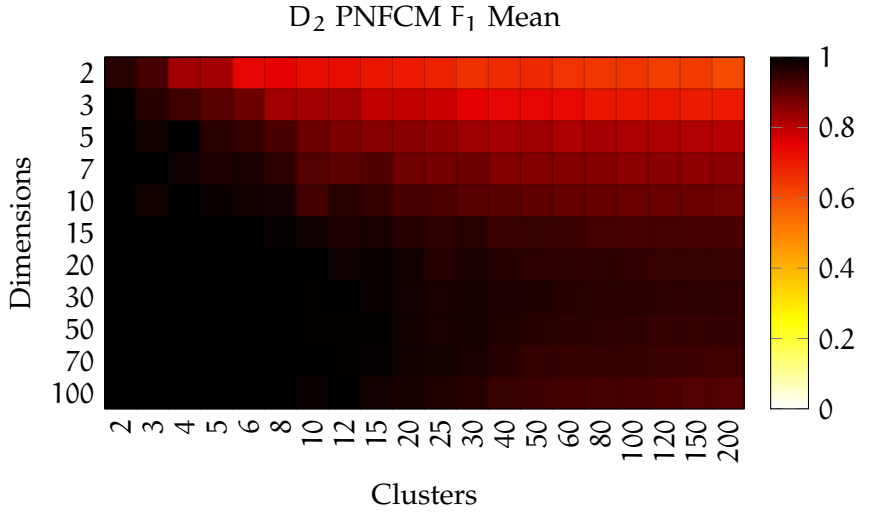


Figure 5.8: PNFCM cluster quality on data set family D_2.

As already mentioned at the end of Chapter 3.4.1, the noise version of this algorithm does not perform well in its canonical form. However, with the procedure of gradually reducing the noise distance, described in Section 3.7.2, this problem can be avoided and lead to a better clustering quality. It also allows the algorithm to be viable in high-dimensional feature spaces.

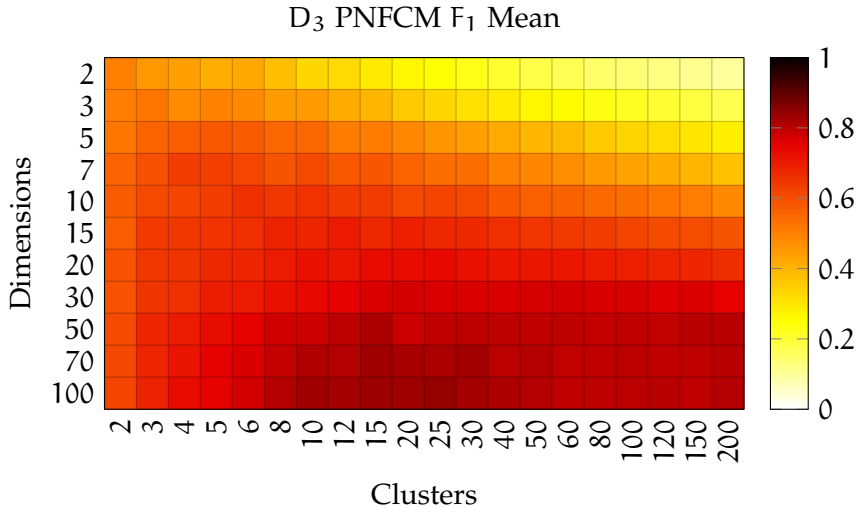


Figure 5.9: PNFCM cluster quality on data set family D_3.

During the benchmark, PFCM as well as PNFCM performed reasonably well, especially on the data set family D_2, see Figure 5.8. PFCM as well as PNFCM did not so well on D_3, for $m \geqslant 50$, see Figure 5.9. The hiding effect of data objects were not sufficient because the classes are extended too far out in the feature space.

5.1.5 *Crisp Rewarding Fuzzy c-Means*

Even though RCFCM was originally not invented for countering the curse of dimensionality of clustering, its update equation (Equation (3.27)) seems like the algorithm was specifically designed to counter just that.

$$u_{ij} = \frac{\frac{1}{d_{ij}^2 - a_j}}{\sum_{k=1}^{c} \frac{1}{d_{kj}^2 - a_j}}$$

By subtracting a constant value from the distances, the contrast among distance values is increased. So even though the distances between one data object and all prototypes might be similar, after subtracting the constant term, the relative similarity vanishes. Therefore, the approximation for FCM_2 in Equation (5.5), is not applicable for RCFCM and the prototypes do not run into the centre of mass.

This is also supported by the CoM analysis with the simplex test-data set, presented in Figure 5.10. No local minimum near the CoM of the objective function is visible and when applied on the benchmark data sets the algorithm does not run into the same problems as FCM_2.

Applied on D_1 and D_2, RCFCM and RCNFCM respectively show almost perfect performance. See for example RCNFCM on D_2, presen-

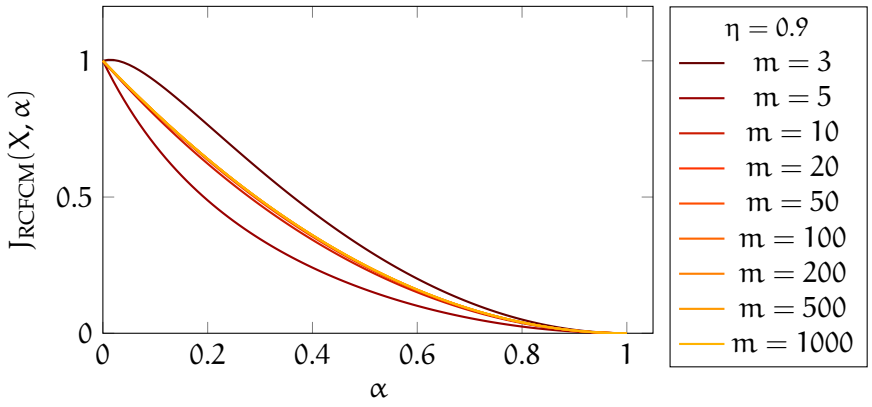


Figure 5.10: The objective function value of RCFCM J_{RCFCM}, plotted depending on the location variable α within the simplex test-data set.

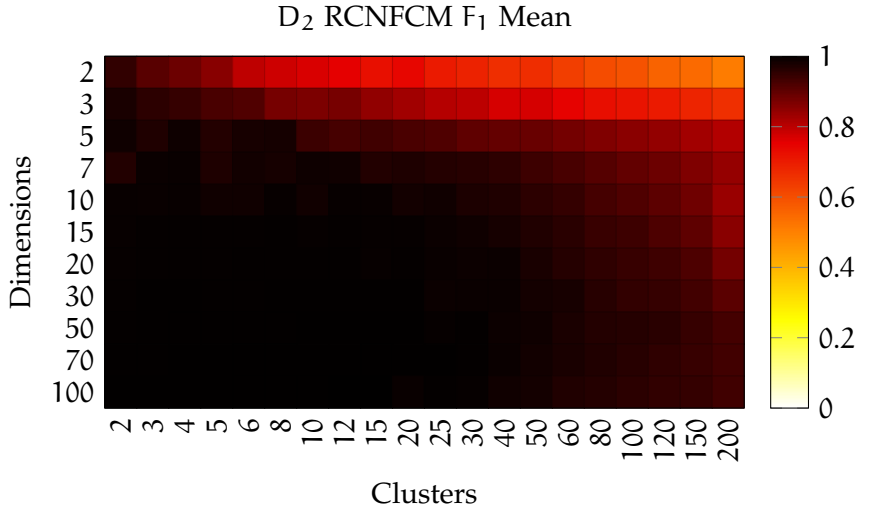


Figure 5.11: RCNFCM cluster quality on data set family D_2.

ted in Figure 5.11. The clustering is, apart from the area of overlapping classes in the top-right, almost perfect and very reliable. If tests were only made with this kind of idealised data sets, one could think that RCFCM and RCNFCM is the solution to the curse of dimensionality for prototype based clustering. However, D_3 and D_4 reveal, that this is not the case. The result on this two last data sets is good, but not as perfect as in case of D_1 and D_2, see the corresponding figures in Sections D.3 and D.4.

5.1.6 *Expectation Maximization with Gaussian Mixture Models*

The EMGMM algorithm is a quite different case than the other fuzzy clustering algorithms. Contrary to the others, it can adjust the size of its clusters in the feature space individually and optimise their spread. At first, that seems to be a good idea, but it also introduces a

new layer of parameters that can corrupt in the process of clustering. The benchmark actually suggests, that EMGMM is capable of clustering high-dimensional data sets. The specific process of optimizing the normal distributions leads to results where at least one cluster is very broad, while the other clusters cover only their respective class nicely. The large cluster actually seems to serve as a noise cluster, but without the benefit of being a uniform distribution. This does not seem to have a negative effect in high dimensions though but a deeper analysis to this effect is necessary to be sure.

No unusual behaviour is quite a remarkable result in itself, even though its quite boring and not very insightful.

Other than that, there was one more case of disturbing behaviour of EMGMM that is not shown in the plots. With increasing number of dimensions, the clusters tend to become very narrow, covering only a very small portion of the data set while one cluster dominated the entire data set, effectively covering all data objects except a few that are very close to the other prototypes. This effect is already discussed in [Borgelt, 2005] and the solution was to give boundaries to the variance (spatial expansion) of the clusters. But it seemed to get worse with higher dimensions and tighter boundaries were necessary. With more degrees of freedom (non-sphercial clusters for example), EMGMM would probably have even larger problems in high-dimensional spaces.

An objective function plot like for the other fuzzy clustering algorithms is not possible for EMGMM. The update process for membership values does not only depend on the location of the prototypes but also on the cluster variance, which cannot be determined using the parameter α.

5.1.7 *Membership Values and Distance Concentration*

There is one more approach to understand the sensitivity of fuzzy clustering algorithms to the effects of distance concentration. Observe Figure 5.12 which is composed of the lower panels of Figures 3.16, 3.17 and 3.18. At $x \approx 5.5$ the distance to $\vec{y}_2$ and $\vec{y}_3$ is approximately equal and the membership value graphs meet.

What an irony! Observed from a well chosen perspective, in one of the most simple examples with just a 1-dimensional feature space lies the key to understand the behaviour of an algorithm in the complex circumstances of a high-dimensional feature space.

This situation is similar to a random initialization at high-dimensional feature spaces, where the distance to all prototypes is approximately equal, see the argumentation in Section 5.1.2. A random initialization of prototypes is usually done so that the dimensions are sampled independently from one another, see Section 3.7.4. In other words, the initial locations of prototypes are a sample of an m-dimensional random distribution with independent dimensions. Any data object $\vec{x}$ can then be regarded as a random query point $\vec{q}$ and as discussed in Section 2.2, the set of prototypes are then subject to distance concentration. As a consequence, they are all roughly equally far away from the data object/query point $\vec{x} = \vec{q}$, just like in the 1-dimensional example in Figure 5.12.

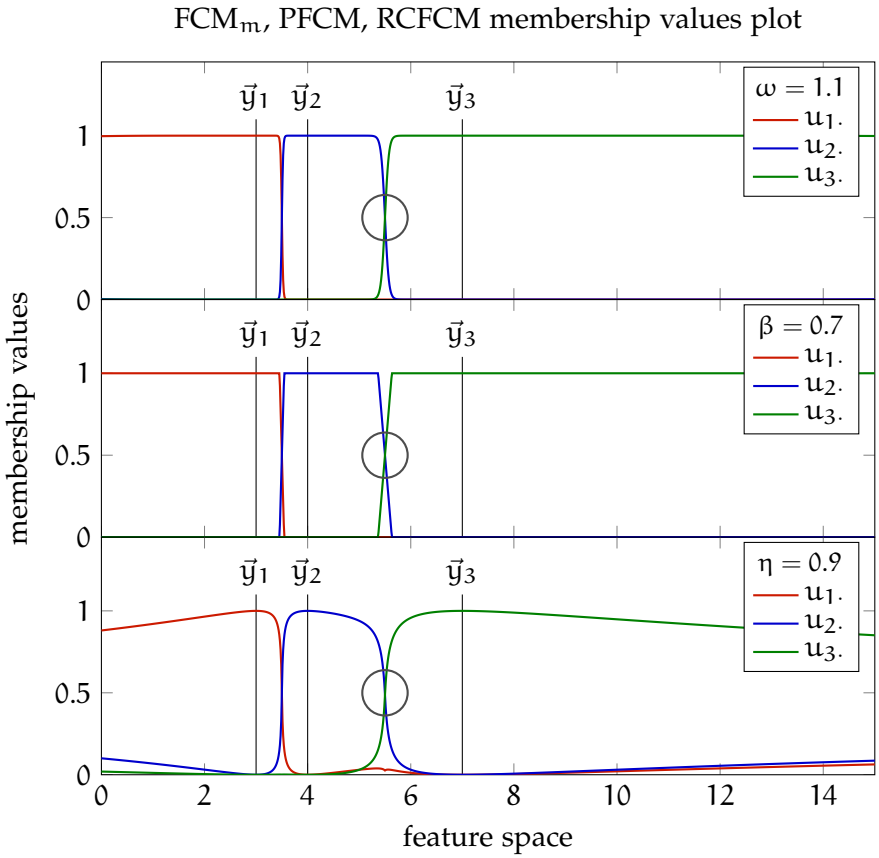


Figure 5.12: Membership value plots of FCM_m, PFCM and RCFCM with additional annotations. The circle indicates the intersection where the gradient is interesting.

If the gradient of membership values is shallow at the crossing point, a small change in distances to the prototypes induces a small change in membership degrees, hence a small overall contrast in membership values. A steep gradient at the crossing point however means, that a small variation of distances induces a large variation of membership values, hence a strong contrast. In other words, these plots indicate how sensitive a clustering algorithm reacts to distance concentration.

As already mentioned at the end of Section 2.7, I tried to create a measure that indicates how sensitive a (any) clustering algorithm is to distance concentration. The gradient of the membership value plots at the intersection indicated by circles in Figure 5.12 was a good candidate for such a measure. However, this approach is not available for EMGMM and HCM as well as competitive agglomeration clustering [Frigui and Krishnapuram, 1997] and it is also not valid for non-prototype clustering algorithms like DBScan [Ester et al., 1996] DENCLUE [Hinneburg and Keim, 2003] and OptiGrid [Hinneburg and Keim, 1999] and hierarchical clustering methods [Zhang et al., 1996].

So the usefulness of such a measure would be rather slim, which defies the intention of giving a general approach.

5.2 COMPARING CLUSTERING ALGORITHMS

In the last section, the algorithms were investigated on an individual level. In this section, their capabilities are compared with each other. When faced with a task to cluster a data set, it is often interesting which algorithm performs best under the given circumstances. Thus, answering thesis question Q_2: which algorithm is the best to use in a high-dimensional clustering task?

Since the success of individual algorithms depends heavily on the data set family D_k, the number of dimensions m and classes/clusters c, the benchmark results are presented in a similar fashion as the individual cluster quality plots (see Appendix D). For a combination of data set family D_k, number of classes/clusters c and number of dimensions m, the algorithms are ranked by their mean maximum F_1 value $\overline{F_1}$, see Equation (5.2). The algorithm A^* with the best (highest) mean maximum F_1 value $A \neq A^* : \overline{F_1}(D_k, m, c, A^*) \geqslant \overline{F_1}(D_k, m, c, A)$ is presented colour coded in the individual cells of the diagrams. In some cases multiple algorithms produced a perfect clustering result in all of the 20 data sets $(D_k, m, c, 1, \ldots, 20)$, which means they all have a $\overline{F_1}$ value of 1. If there is no clear best algorithm, all candidates with the same $\overline{F_1}$ value are grouped in one (D_k, m, c) cell.

5.2.1 *Algorithms, applied on* D_1

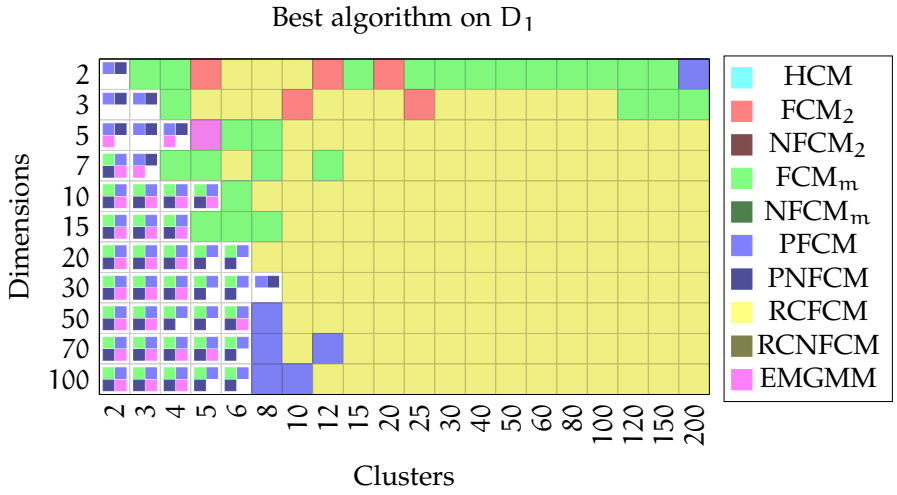


Figure 5.13: Map of algorithms that performed best on data set D_1.

The first data set family D_1, described in detail in Section 4.1.1 is the simplest data set of the four discussed in this work. In the per-

formance plot, presented in Figure 5.13, the best algorithm that performed for a combination of number of dimensions m and number of classes/clusters c is colour coded according to the list on the right-hand side of the figure. Since the data set family does not contain any noise, clustering algorithms that expect noise do not perform very well, compared to the others. The only exception is PNFCM. Since PNFCM can assign crisp membership values, it is possible that all data objects are crisply assigned to one or more clusters, which means that no data object is (partly) assigned to the noise cluster. This can happen especially in data sets with a small number of classes/clusters compared to the number of dimensions, so that the classes are well separated.

It is somewhat surprising to see that FCM_2 performs best in some cases, where the number of dimensions is low and the number of classes/clusters is between 5 and 25. This indicates that FCM_2 is capable of handling overlapping classes reasonably well, which is the motivation to use FCM_2 in the first place. With the exception of $m = 2$ and $c = 200$, PFCM seems to work best for low number of classes/clusters. In most of the low-cluster and low-dimension cases, FCM_m seems to be one of the best algorithms. In the vast majority of cases, RCFCM is the best algorithm. However it seems to lack the ability to produce perfect results in cases with low number of clusters, which is why FCM_m, PFCM, PNFCM, and EMGMM show a better performance in the lower left part of the diagram.

5.2.2 *Algorithms, applied on* D_2

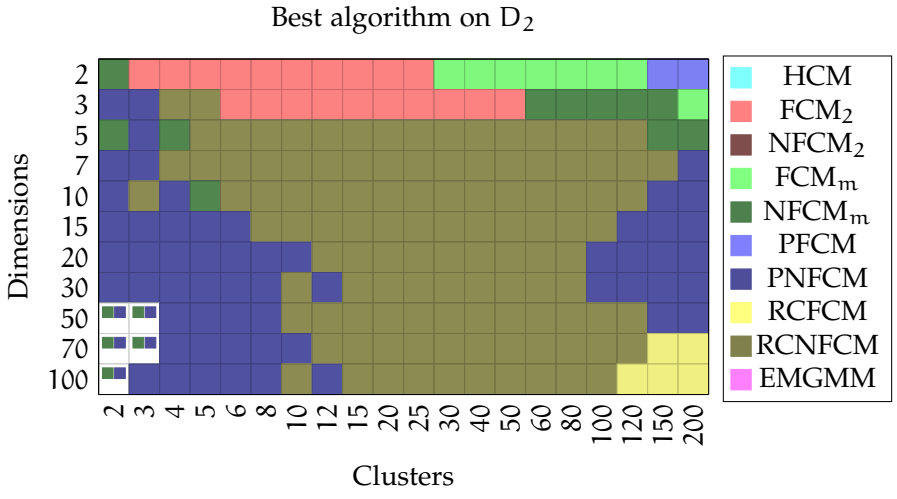


Figure 5.14: Map of algorithms that performed best on data set D_2.

Contrary to D_1, the second data set family D_2 contains noise. It comes to no surprise, that clustering algorithms that expect noise data objects perform better than others. However, in low dimensional data

sets with overlapping classes, non-noise clustering algorithms like FCM_2 and FCM_m seem to be the better choice. For higher dimensional data sets however, PNFCM and RCNFCM dominate the ranking. The pattern however, seems to be quite odd. Both for low number of clusters and for a high number of clusters, PNFCM performs better than RCNFCM. When looking at the individual performance maps, it seems that RCNFCM has difficulties with overlapping classes, at least more than PNFCM. Also PNFCM is able to produce a perfect clustering result in data sets with a low number of classes/clusters. RCNFCM is not able to do that, even though it produces good results.

It is however surprising to see that in cases of high-dimensions and high number of classes/clusters RCFCM performs better than either PNFCM or RCNFCM. There is no obvious explanation. It can be speculated, that for high number of classes/clusters and in a high-dimensional environment, it is very likely that entire classes are clustered as noise, therefore reducing the F_1 value. This seems to out-weight the inability to detect noise for RCFCM.

5.2.3 *Algorithms, applied on* D_3

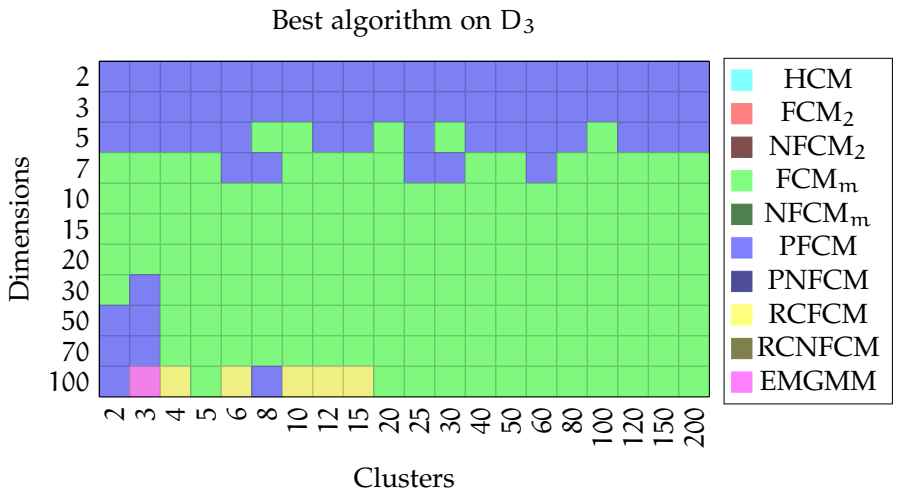


Figure 5.15: Map of algorithms that performed best on data set D_3.

After the first two data set families, D_3 shows some surprising results. In none of the previous two data set families, FCM_m seems to be the algorithm of choice, in D_3 however, it is the best algorithm in most of the configurations. Also PFCM performs best in the low-dimensional area, which was not the case in D_1 and D_2 either.

It is especially noteworthy that not in a single case, an algorithm with a noise cluster created the highest $\overline{F_1}$ value, even though the data set contains as much noise data objects as D_2. That is astonishing because the noise distance is calculated in the same way as for data sets in the D_2 family where the strategy seemed to work very well, see

Section 4.4.2. Maybe the choice of the noise distance was inappropriate for the extended shape of the classes in the D_3 data set family, so that the noise distance was either to small or too large. An alternative explanation would be that the structure of the complex and extended classes prevents a good clustering when considering a noise cluster, independently of the true noise distance. A range of experiments with different noise distances would be necessary to find that out, which is not feasible any more at this point. A deeper investigation might be a topic for further research in the future.

5.2.4 *Algorithms, applied on* D_4

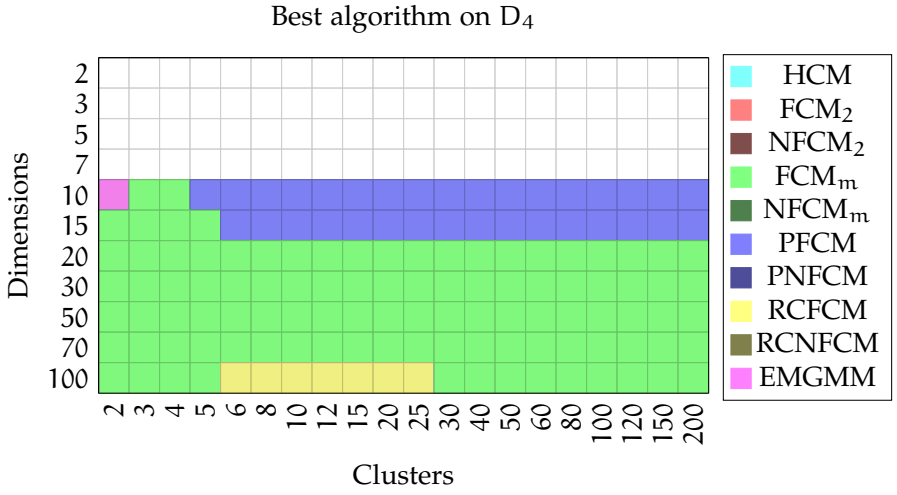


Figure 5.16: Map of algorithms that performed best on data set D_4. The white squares indicate an empty cell because the data set family D_4 does not have data sets below 10 dimensions.

Despite the fact that the D_3 and D_4 families of data sets are very different, the performance rankings are very similar. In relatively low dimensions $m = 10$ and $m = 15$, PFCM performs best, while in higher dimensions FCM_m is clearly dominating. It should be noted though, that the D_4 data set family showed some strange performance characteristics, see Appendix D.4. Between $m = 10$ and $m = 15$ dimensions, all algorithms perform less well than in dimensions $m = 50$, $m = 70$ and $m = 100$. Due to the generation process (see Section 4.1.4 and A.4), the classes in the dimensions 10 and 15 overlap more than on 20 and above, but it is unexpected that this is so apparent in the quality results.

5.2.5 *In summary: which algorithm is best?*

In most papers, data sets, similar to the families of data sets D_1 and D_2 are tested. Based on these data set families alone, it would be easy

to declare that in high-dimensional data sets, RCFCM as well as PNFCM and RCNFCM are the best algorithms to solve related problems in high-dimensional feature spaces. The results from data set families D_3 and D_4 suggest, this would be an ill advised conclusion. After observing the above presented results, it must be concluded that there is no single best algorithm for high-dimensional feature spaces. It is however possible to conclude that some algorithms do not seem to provide a good clustering result overall. These algorithms are HCM, EMGMM as well as FCM_2 and $NFCM_2$. HCM did not perform best in any of the test cases, which indicates that there might always be a better fuzzy algorithm available. EMGMM performed best occasionally, but not consistently enough to consider this algorithm for any difficult clustering problem. FCM_2 and $NFCM_2$ are obviously not a good choice for high-dimensional feature spaces. But this was clear from the beginning since it was the motivation to start this work in the first place, see Figure 1.5 in the introduction. In that sense, the thesis question Q_2 cannot be answered with a satisfying and comprehensive advice.

It is remarkable that in not a single case HCM performed best, not even in low dimensions and for a low number of classes. Despite the fact that HCM has an advantage because all clustering problems are crisp. That is something I did not expect when I started my investigation. HCM is very popular, maybe this is not well deserved.

The results from the presented data sets show, that the conclusion *an algorithm is likely to perform best on one kind of data set because it performs best on an other kind of data set* is not valid and should be avoided. Likewise, it is not always clear why some algorithms perform better than others, especially when comparing (N)FCM$_m$, P(N)FCM and RC(N)FCM. The interpretation that was given here is very situational and might not be valid for other data sets. It is not possible to predict what happens if the algorithms are applied to an unknown data set, which means its not possible to know a-priori which algorithm to use.

Since it is clear that there is no single best algorithm and since the F_1 measure cannot be used for a real world data set because it requires the knowledge of the true result, internal indices are required to detect which algorithm works best in a given situation. This topic is investigated in the next section.

5.3 INTERNAL CLUSTER QUALITY INDEX VERIFICATION

In this section, thesis question Q_4 is approached: are internal index measures useful to assess the quality of clustering results in high-dimensional feature spaces? Internal index measures are primarily used for two tasks: first to rank clustering results to select the best and second to quantify the quality of the clustering result. As was seen from the analysis in the last section, this is really necessary as there does not seem to be a clearly superior algorithm. Both parts can be tested using the clustering results from the benchmark data sets,

presented in Chapter 4. The two different parts of the question require different approaches which are presented in the next two subsections.

For both investigations, the clustering results of HCM are ignored because HCM produces only crisp clustering results. The exclusively crisp membership levels would bias the results of the XB index, the PE index as well as the NPC index (see Section 4.3). As was established already in Section 5.2.5, HCM does not perform particularly well on any of the benchmark data sets anyway. Also, as for the F_1 index (see the second paragraph in Section 5.1), the prototype centric cluster diameter and cluster separation are used to calculate the index values.

5.3.1 *Ranking of Clustering Results by Internal Indices*

As is apparent from the results of the last chapter, it is a-priori not clear which clustering algorithm is best suited for an unknown data set. Usually, the data set is clustered using several different clustering algorithms and the best result is selected. If the true result is unknown, internal quality measures are used to estimate the ranking of the results. Assuming the F_1 measure provides the correct ranking, the quality of the internal cluster quality index induced rankings can be measured using the benchmarks from this thesis.

As before, the tuple (D_k, m, c, s, r, A) indicates a specific clustering result. The data set (D_k, m, c, s) does not depend on the algorithm that is used for the clustering nor on the initialization, which allows a ranking of all clustering results based on that data set. That means for a fix data set family D_k, number of dimensions m, number of classes/clusters c and data set s the ranking is computed by evaluating the clustering results for all initializations $r = 1 \dots 5$ and all 9 fuzzy clustering algorithms $A_f \in \{\text{FCM}_2, \text{NFCM}_2, \text{FCM}_m, \text{NFCM}_m, \text{PFCM}, \text{PNFCM}, \text{RCFCM}, \text{RCNFCM}, \text{EMGMM}\}$. For the clustering results of 45 different experiments, specified by the pair (r, A), the F_1 index values and the internal index values are computed. Without loss of generality, let the pairs be ordered according to the value of the F_1 index, such that $F_1(D_k, m, c, s, r_i, A_i) \geqslant F_1(D_k, m, c, s, r_{i+1}, A_{i+1})$ for $i \in \{1, \dots, 44\}$. For any internal index, $I \in \{I_{BS}, I_{DB}, I_{XB}, I_{NPC}, I_{NPE}\}$, the ranking quality is than defined by the F_1 quality index induced ranking position i of the highest I-ranked clustering result:

$$R_I(D_k, m, c, s) = \begin{cases} \underset{i \in \{1,\dots,45\}}{\arg\max}\ I(D_k, m, c, s, r_i, A_i): & I \in \{I_{BS}, I_{NPC}\} \\ \underset{i \in \{1,\dots,45\}}{\arg\min}\ I(D_k, m, c, s, r_i, A_i): & \text{otherwise} \end{cases} \tag{5.10}$$

In words, the ranking quality $R_I(D_k, m, c, s) \in \{1, \dots, 45\}$ specifies the F_1 ranking position of the best ranked experiment according to index I. A maximal value for I_{BS} and I_{NPC} and a minimal value for I_{DB},

I_{XB} and I_{NPE} indicate the best index-selected clustering result. So the internal index I identifies experiment (r_i, A_i) as the best clustering result and the counting index i indicates the 'true' ranking position of that clustering result as measured by the F_1 index. Therefore, the counting index i in this case is equal to the ranking error of internal cluster quality index I. It is possible that several (r, A) pairs share the same F_1 value, so that the ranking error indicated by i is higher due to the arbitrary ordering of equal F_1 scores. To prevent this bias, the position of the best ranked pair with equal F_1 value is used to calculate the ranking error in such cases.

Finally, the mean of ranking quality values is calculated for all $s = 1, \ldots, 20$ different data sets of one data set family D_k, number of dimensions m, number of classes/clusters c: $\overline{R_I}(D_k, m, c)$.

$$\overline{R_I}(D_k, m, c) = \frac{1}{20} \sum_{s=1}^{20} R_I(D_k, m, c, s) \tag{5.11}$$

The data is presented (see Figure 5.17) in the same fashion as the F_1 score plots (see in Figures 5.1, 5.3, etc.), but this time the colour indicates the ranking error, which is quite different to the score plots. The lower panel shows again the standard deviation of the 20 ranking error values, which is more relevant this time because it shows how consistently the internal index can create the ranking quality. A visual representation of all 5 internal cluster quality indices for the 4 data set families is presented in Appendix E.

It cannot be expected that internal quality measures produce perfect results and some variation is to be expected as well.

In the example in Figure 5.17, the ranking quality of the BS index is presented. The colour indicates the ranking error, with darker colour meaning less quality. Please note, that this can be interpreted as: lighter colour is better than darker colour, which is reversed to the way the F_1 score plots are interpreted. In this case, there is a strike of higher ranking errors from dimension $m = 20$, classes $c = 20$ to the lower right corner. This is accompanied with a increase in the ranking error standard deviation, which shows that for this example, the BS index produces less reliable results.

All together, the structures that appear in the ranking performance plots are quite concerning. Some of the curiosities are:

- As expected from the index definition, the NPC and NPE index show very similar results, except for data set family D_1, where the NPC index performs better.
- For the data set family D_1, the BS-Index seems to predict the the ranking of the best clustering algorithm very well. The DB and NPC index seem to work better for fewer classes/clusters, while the XB index works better for many classes/clusters. The NPE index on the other hand seems to predict the ranking not

I checked the source-code that produced the results several times. It is complex and I cannot rule out completely that the effects are induced by some error or bug in the software. However, I tested every part extensively and did not find any mistake. Therefore, I assume the effects are real. I would like to see them confirmed by an independent analysis.

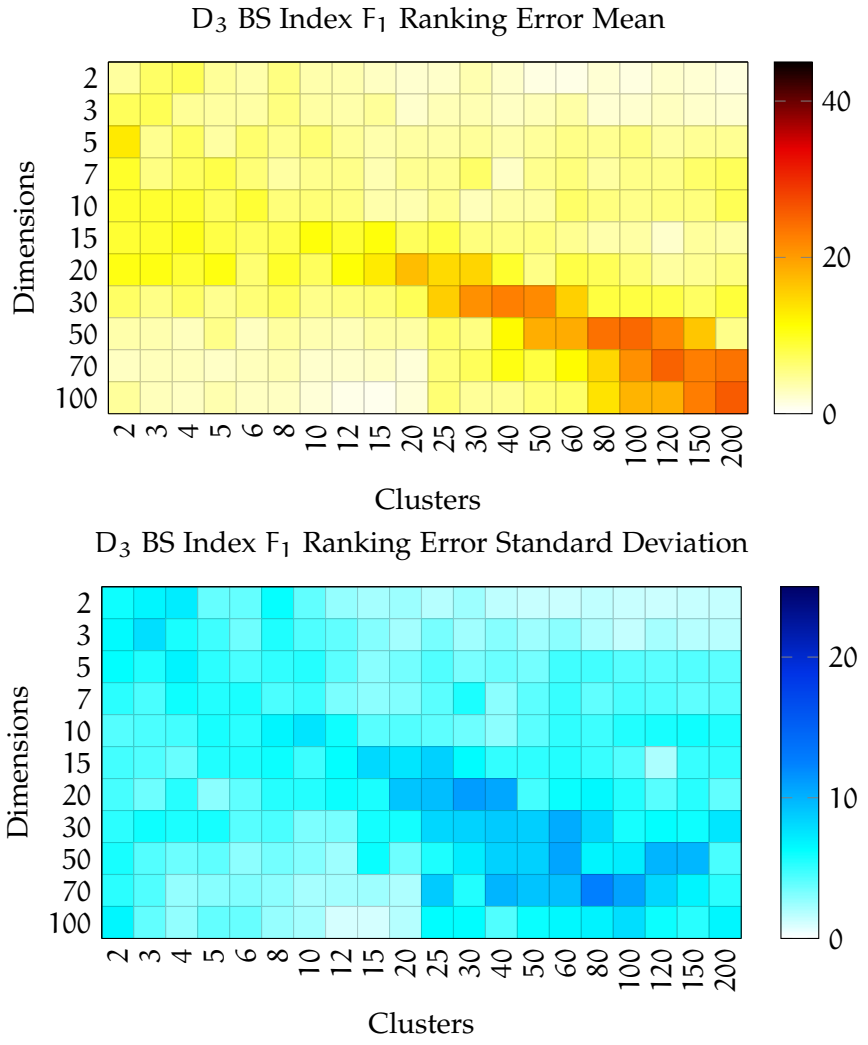


Figure 5.17: Ranking quality of the BS index on data set family D_3.

so well but still provides better cluster quality indication for higher number of dimensions.

- Despite the similarity between D_1 and D_2, the ranking errors on D_2 seem to be completely unrelated to D_1, except for the XB index, which shows a slight decrease of ranking quality. Also the DB index shows lower indication quality for higher dimensions while the BS, NPC and NPE indices show an increase of ranking quality for higher number of dimensions.
- Surprisingly, the ranking quality for D_3 is better than for D_2 for the BS and DB indices, despite the higher complexity of the data set. The other 3 indices show a slightly lower ranking quality.
- The BS index seems to perform much better than the other indices on D_3 and D_4.

Without looking too closely at the curiosities, the overall ranking quality is surprisingly poor throughout all data sets and indices. The best overall performance was presented by the BS index on the D_1 data set family (see Appendix E.1), but that data set family is hardly realistic. On data set families D_3 and D_4 none of the indices is particularly reliable when it comes to the ability to recognise the best clustering result, see Appendix E.

The argumentation in the text can be regarded as a rough interpretation of the situation. It is possible to do a deeper investigation, but the quality of the results does not justify the afford.

Take for example the ranking quality of the BS index on the D_3 data set family, presented in Figure 5.17 which is the best index for data set families D_3 and D_4. For most of the dimensions and cluster counts, the mean ranking error is better than 8 (that is, in 2/3 of the cases). The standard deviation in the corresponding cells shows a value of around 5. Assuming the ranking error values follow roughly a normal distribution, then in 98% of the cases, the clustering result with the best BS index value is among the best $23 = 8 + 3 \cdot 5$ clustering results, according to the F_1 index. This means, the BS index predicts that the clustering result with the best index value is somewhere among the best 50% of the clustering results. Despite the visually good performance, this is not good enough to select the 'best' clustering result consistently.

As already mentioned in Section 4.2.3, this conclusion is subject to the proposition that the F_1 index reflects the true cluster quality and induces the correct ranking of the clustering results.

5.3.2 *Quantifying Cluster Quality with Internal Indices*

From the last subsection, it is obvious that the internal quality measurements cannot reliably reproduce the F_1 ranking to a satisfying degree. Even though the ranking has good practical applications, it is also interesting how well the internal indices reflect the true clustering quality as indicated by the F_1 measure. To answer this question, the indices need to be able to predict the true clustering quality. In other words, the index values need to be correlated with the true cluster quality. Again, it is assumed that the F_1 index represents the true clustering quality and the correlations are computed w.r.t. the F_1 index.

The correlation is measured using Spearmans rank correlation coefficient [Hotelling and Pabst, 1936]. The more popular Pearson correlation coefficient [Irle, 2005] is not a good choice here because it is very sensitive to outliers (which exist quite regularly in the index data) and because the internal indices do not necessarily have a linear dependency to the F_1 measure. As a result, Spearmans rank correlation coefficient is much more robust and provides better insight into the relationship of the internal indices to the external F_1 index.

The rank correlation coefficient is defined over the rank of the data. Let $X = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset \mathbb{R}^m$ of at least two dimensions $m \geqslant 2$ be

a data set. Let X_k denote the values of X in dimension k: $X_k = \{\vec{x}_{1,k}, \ldots, \vec{x}_{n,k}\} \subset \mathbb{R}$ and X_l denote the values of X in dimension l: $X_l = \{\vec{x}_{1,l}, \ldots, \vec{x}_{n,l}\} \subset \mathbb{R}$. The natural ordering of the data objects w.r.t. an attribute is used to compute the rank. To make the equations simpler, let the data objects in X be ordered according to k if the rank for attribute k is computed and that holds $\vec{x}_{1,k} \leqslant \cdots \leqslant \vec{x}_{j,k} \leqslant \cdots \leqslant \vec{x}_{n,k}$. The function $\operatorname{rank}_k(\vec{x}_j) \in \mathbb{R}$ of data object $\vec{x}_j$ according to dimension k defines the position of $\vec{x}_j$ in the ordered list. If two or more data objects have the same value in dimension k, the rank of all of these data objects is equal to the average of the positions:

$$\operatorname*{rank}_k(\vec{x}_j) = \begin{cases} j & \text{if} \quad \vec{x}_{j-1} < \vec{x}_j < \vec{x}_{j+1} \\ \frac{1}{j^+ - j^- + 1} \sum_{t=j^-}^{j^+} t & \text{if} \quad \vec{x}_{j^-} = \cdots = \vec{x}_j = \cdots = \vec{x}_{j^+} \end{cases} \tag{5.12}$$

The list of data objects is sorted two times, one time according to dimension k and one time w.r.t. dimension l, giving two rank values for all data objects. These rank values are than used to compute the Spearman rank correlation coefficient.

$$C_{k,l}(X) = \frac{6}{n^3 - n} \sum_{j=1}^{n} \left(\operatorname*{rank}_k(\vec{x}_j) - \operatorname*{rank}_l(\vec{x}_j) \right)^2 \tag{5.13}$$

The value of the rank correlation coefficient shows, how strong the two dimensions k and l of X are functionally dependent. Any functional dependency can be measured with rank correlation of 1 as long as the function is monotonously increasing.

The rank correlation coefficient is used to estimate how well an internal cluster quality index is able to represent the true cluster quality, measured by the F_1 index. Since the F_1 index has a positive slope with the true cluster quality, some of the internal indices are redefined to also have a positive slope w.r.t. true cluster quality. That means, the DB, XB and NPE index values are multiplied by -1 prior to calculating the rank correlation coefficient.

The complete results are again presented in Appendix F. The internal index values are supposed to be independent on the clustering algorithm. This time, there is no need to calculate the mean over the $s = 1, \ldots, 20$ data sets of identical data set family D_k, number of dimensions m and number of classes/clusters c. So in total, the rank correlation coefficient of index values for 900 clustering results $(D_k, m, c, \cdot, \cdot, \cdot)$ are computed. These are index results of $s = 1, \ldots, 20$ different data sets, $r = 1, \ldots, 5$ different initializations per data set and 9 different clustering algorithms (HCM is left out again because of its pure crisp clustering).

As discussed at the beginning of this section (see the second paragraph in Section 5.3, the prototype centric cluster diameter and intercluster distances are used for some of the internal indices. In some

cases, two prototypes were identical (they have a distance of 0) or one cluster was covering only one or no data object, which resulted in a 0 cluster diameter. The BS, DB and XB indices rely on the inverse of the minimal pairwise distance of two different clusters, see Equations (4.5) and (4.6) or the inverse of the diameter of the clusters, see Equation (4.4). In cases where zero values are present for these distances, a meaningful calculation of the BS, DB and XB indices is not possible. As a result, some index values could not be taken into account for calculating the rank correlation coefficient. The total number of unused index values is 3,909 out of the total number of 2,160,000 calculated index values of the BS, DB and XB indices.

A positive rank correlation coefficient near 1 for an internal index I indicates a good agreement with F_1, which in turn indicates that the internal index I can be used to assess the cluster quality. A small rank correlation value, close to 0 indicates that the value of index I is not connected to the true cluster quality, which means that index I cannot be used to assess the cluster quality. Even worse, a negative rank correlation coefficient means that a good value of index I corresponds to a low true cluster quality and vice versa.

The colour scheme of these plots required some RGB magic and took a lot of time to get right. I find them not only very intuitive but also quite pretty.

The rank correlation coefficient is presented in the same way as the other performance values before, see for example Figure 5.18. The colour scale is different to visualize the presence of negative values in an obvious way. As can be seen in this figure, the rank correlation coefficient can be influenced by the bad performance of FCM_2 and $NFCM_2$ for a high number of dimensions and classes/clusters. The reason for that is, that (N)FCM_2 performs so bad that it acts like an outlier which is easily detected by the rank correlation. Also, as can be seen in Appendix G, the EMGMM algorithm can create a lot of outliers for some indices. EMGMM tends to create very large and broad clusters as well as very narrow, concentrated clusters with almost equal prototype locations. That creates an artificial minimum inter-cluster distance that can get very small and create outliers for the DB and XB index because both indices contain the inverse of the separation of clusters, see equations (4.5) and (4.6). This is a computational artefact from the choice of using prototype distances rather than pairwise data object distances, discussed in Section 4.3.1. It was therefore decided to create two versions of each plot, one time with (N)FCM_2 and EMGMM and one time without them, see Appendix F for more plots. As discussed before, HCM is still missing from all plots because some of the indices can not measure the cluster quality due to the crispness of the result.

Even though the spearman correlation coefficient is quite robust against this, the indices still get confused which limits their usefulness and disturbs the result.

Judging from the previous subsection (Section 5.3.1), it can be expected that in some occasions, the internal indices and the cluster do not correlate very well. Still, the plots show unexpected features, especially the amount of negative correlation is troublesome. See for example Figure 5.19. They show both the correlation coefficients for

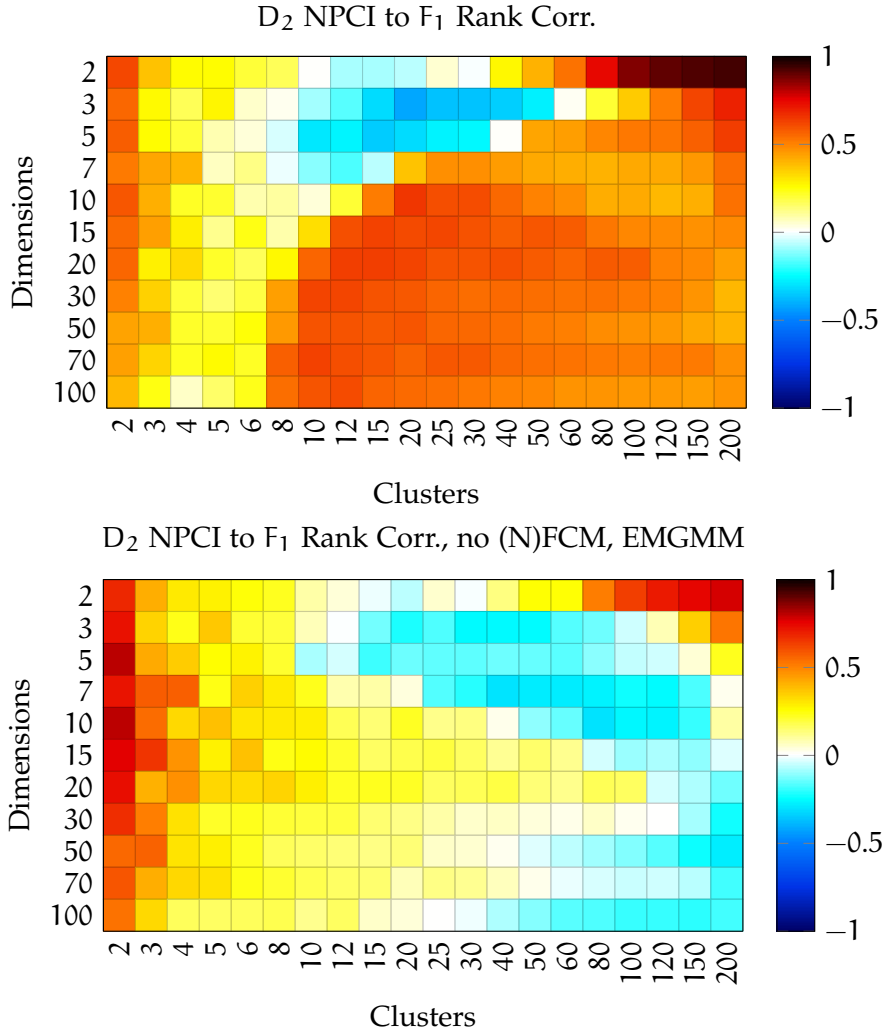


Figure 5.18: Top: Rank correlation of the NPC index with the F_1 index on the D_2 data set family. Bottom: Rank correlation of the NPC index with the F_1 index on the D_2 data set family, without clustering results from the FCM_2 and $NFCM_2$ algorithms.

data set family D_3, in the upper panel between the DB index and F_1 and in the lower panel between the XB index and the F_1 measure. The huge area of the negative correlation coefficients for the XB index is unexpected, especially that it is present for a low number of dimensions but not for a high number of dimensions. Please observe the full list of plots in Appendix F, where the left column of plots show the correlation coefficients including (N)FCM_2 and EMGMM while in the right column, the index values for clustering results from (N)FCM_2 and EMGMM are excluded.

The traditional notion of a good clustering of a data set: *'data objects of the same cluster are closely together while data objects of different clusters are far apart'* breaks down in high-dimensional feature spaces because the distances are not very meaningful due to the concentration effect.

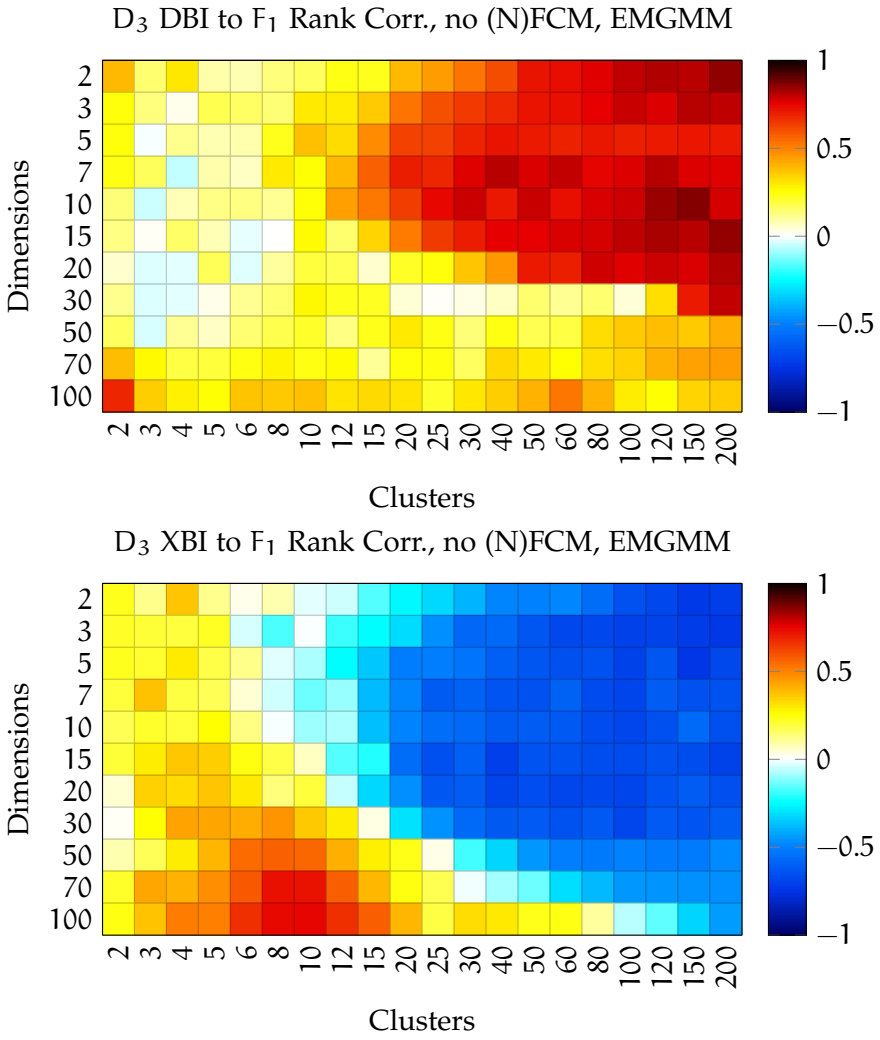


Figure 5.19: Top: Correlation of the DB index with the F_1 index on the D_3 data set family. Bottom: Correlation of the XB index with the F_1 index on the D_3 data set family. Both panels show the correlation coefficient without clustering results from the (N)FCM$_2$ and EMGMM algorithms.

Since the internal indices are invented to measure this abstract notion of a good clustering, it should be expected that the quality of the indices depends on the number of dimensions and classes/clusters in a similar way as the clustering result itself. The effect of the Curse of Dimensionality should be most present for the BS, DB and XB indices. However, that is not what is observed.

The patterns are very interesting and even though they are not the core topic in this thesis, I decided to investigate.

The structure that is visible is very diverse. There is no clear gradient dependency on the number of dimensions, it seems that some other effect dominates the correlation coefficient. The remaining part of this subsection is devoted to understand what is going on.

In Figure 5.20, two example plots of the internal quality index values versus the F_1 index value for some of the extreme negative correl-

ation spots are plotted. The plots show the value of an internal index in relation to the F_1 index for data sets from the D_2 data set family with $m = 3$ dimensions and $c = 40$ classes/clusters. In the upper plot, the BS index is presented, in the lower plot the NPC index. Each plot here is a visualization of the data that is used in one cell in the rank correlation coefficient plots, shown for example in Figure 5.18. To eliminate that the number of dimensions is a significant factor, a case was chosen where the number of dimensions is low, 3 in this case. Each dot in a plot corresponds to one clustering result and the clustering algorithms that were used to generate the clustering results are colour coded, as presented in the legend. It seems that the correlation coefficient is dominated by the selection of the algorithm rather than any other factor in this case. Especially in the lower plot, it seems that the NPC index is positively correlated with the F_1 index for any individual clustering algorithm, but the index value is negatively correlated when taking all clustering algorithms together.

In case additional plots are required, they can be generated by using the provided data tables. The necessary LaTeX and TikZ source code snippets can be found in the download section for this thesis.

In case of a positive correlation of the BS and NPC index for data set family D_3, at dimensionality of $m = 7$ and $c = 120$ classes/clusters a similar effect can be observed, see Figure 5.21. In both images, the now positive correlation coefficient is dominated by the selection of algorithms.

To investigate the influence of the clustering algorithm selection on the internal index measures, a range of plots of the correlation coefficient defining data are presented in Appendix G. These plots show that the correlation coefficient is in many cases dominated by the selection of algorithms. The separation of results with a noise cluster (results from $NFCM_2$, $NFCM_m$, PNFCM and RCNFCM) from the results without a noise cluster (results from FCM_2, FCM_m, PFCM, RCFCM and EMGMM) can be an artefact of the definition of internal indices when a noise cluster is present, see Section 4.3.7. But also within the subgroups of purely non-noise clustering algorithms as well as purely noise clustering algorithms, the selection of algorithms often dominated the correlation coefficient between the internal indices and the F_1 measure. Also a trend is visible when observing the plots in Appendix G. On data sets with higher number of classes/clusters, the separation of clustering algorithms becomes more dominant, as the plots of $c = 50$ and $c = 150$ show, contrary to the other plots of $c = 5$ and $c = 20$.

But not only the selection of algorithms is relevant, also the indices directly contradict each other. Observe again Figure 5.19. The correlation coefficient maps for the D_3 data set family of the DB and XB index are presented. In the upper right half of the images, the correlation coefficients are reversed, meaning that both indices indicate that clustering results that are measured well by one index and bad by the other and vice versa. In this particular case, the XB index evaluated the results of the RC(N)FCM algorithm to be very good compared to

the other results while the DB index and the F_1 index evaluated the RC(N)FCM algorithm as bad compared to the others, hence the observed correlation coefficient, see Appendix-Section G.2.3 for a plot of the situation.

I am quite shocked by this results. It is one thing to have a diffuse knowledge that internal quality indices are not trustworthy and quite another to see the bias that the indices induce. At this point, I might go as far to say that internal indices cannot be trusted in general unless proven trustworthy in a specific situation. Independently of the number of dimensions.

In summary, the relationship of the internal cluster quality indices to the true clustering quality is dominated by the selection of clustering algorithms. This defies the purpose of applying the internal cluster quality indices since they are mainly used to compare different clustering algorithms. This also answers thesis question Q_4 in a rather unsatisfying way. The clustering quality in high-dimensional feature spaces cannot be determined by internal indices BS, DB, XB, NPC and NPE because the predictive power of these index measures is dominated by the selection of clustering algorithms and not by the quality of the clustering algorithm results. On a data set where the true clustering result is unknown, there is no way to know whether an internal index is positively correlated with the true clustering result, not correlated at all, or even negatively correlated. Without this knowledge however, the internal index values are meaningless. Of course this conclusion is again based on the assumption that the F_1 measure reflects the true clustering quality well.

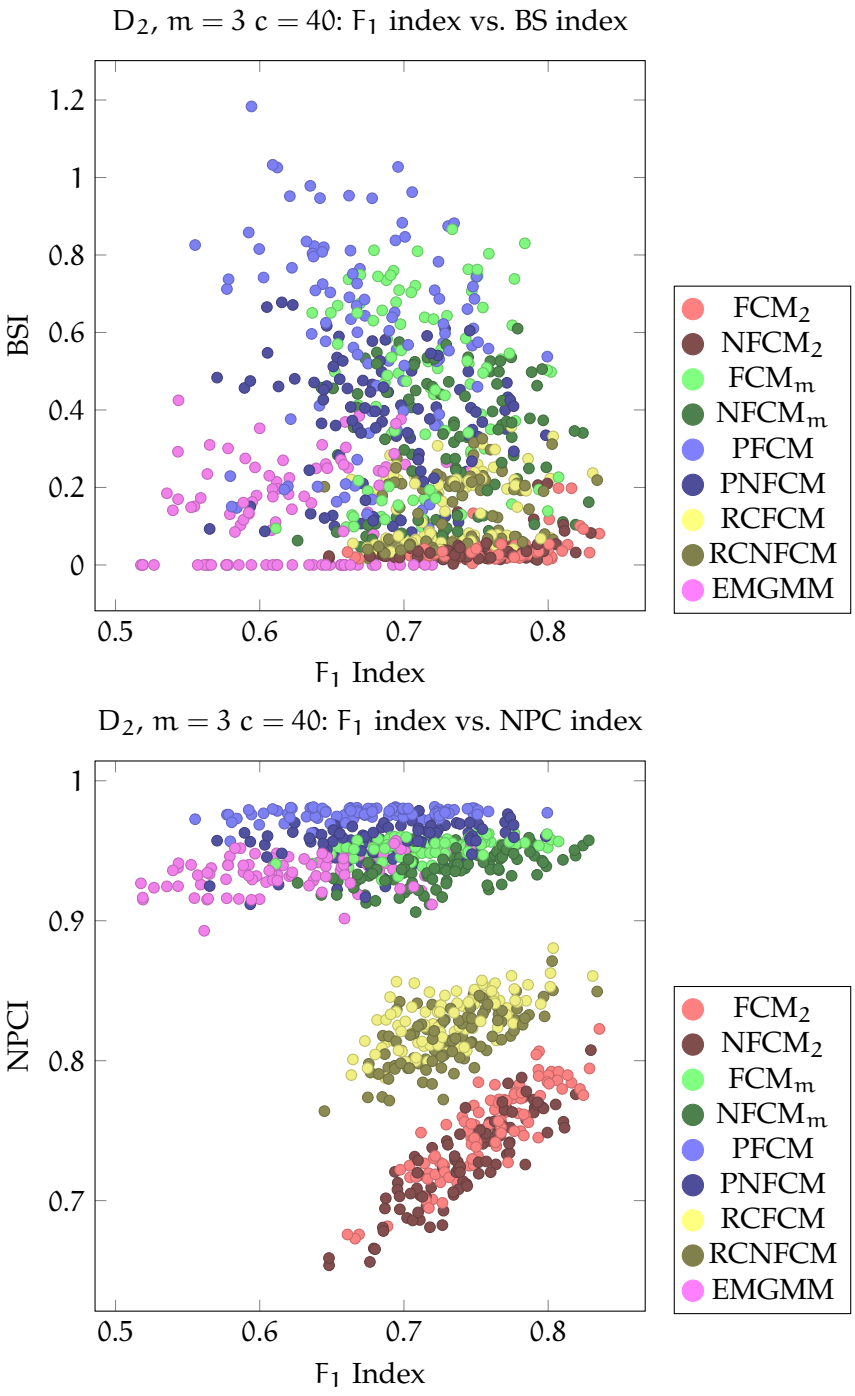


Figure 5.20: Correlation plots of the D_2 data set family of $m = 3$ dimensions, $c = 40$ classes using the BS (top) and NPC (bottom) indices on the y-axis and the score of the F_1 index on the x-axis. The index values are centred on the mean and scaled by the standard deviation.

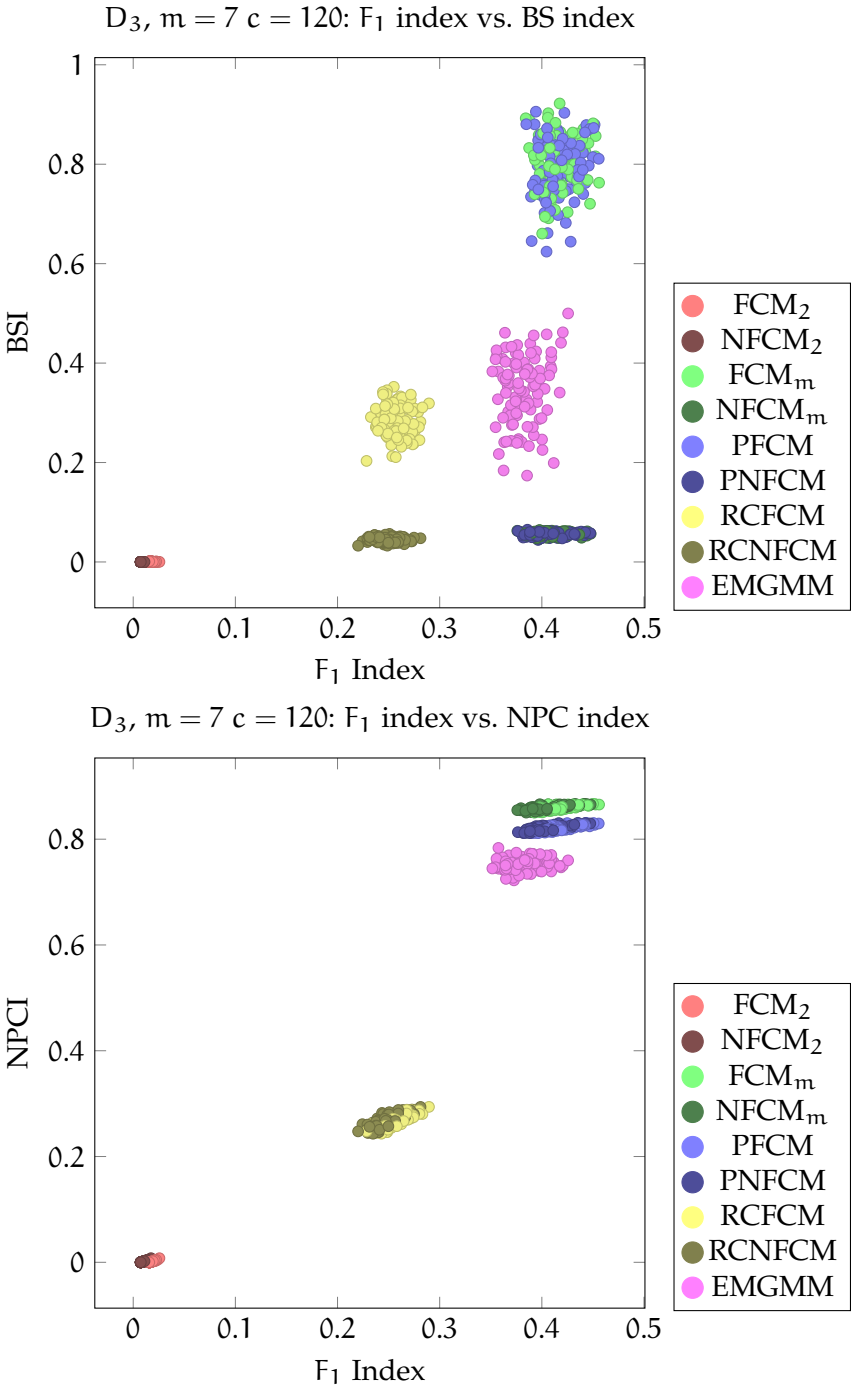
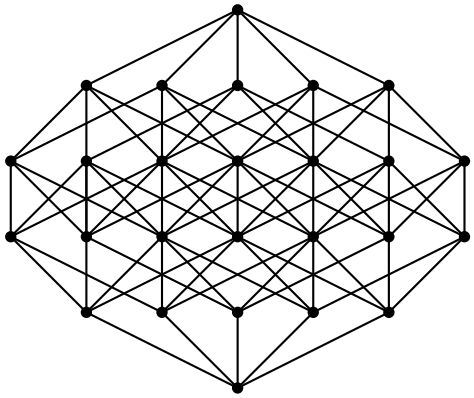


Figure 5.21: Correlation plots of the D_3 data set family of $m = 7$ dimensions, $c = 120$ classes/clusters using the BS (top) and NPC (bottom) indices on the y-axis and the score of the F_1 index on the x-axis. The index values are centred on the mean and scaled by the standard deviation.



6

CONCLUSIONS AND FUTURE RESEARCH

The purpose of data mining in general and clustering in particular is to transform raw data into knowledge. In high-dimensional feature spaces, this task is particularly difficult. The structure of high-dimensional feature spaces prevent an easy, direct application of clustering algorithms that work well on low dimensional feature spaces. An example is FCM_2 as can be seen in the introduction, see Figure 1.5.

In the past chapters, a lot of the relevant information for a successful clustering in a high-dimensional feature space have been discussed. A short and informative summary is presented in this chapter, beginning with the thesis questions that are introduced in Section 1.3. All four questions and their answers are presented, followed by Section 6.2 which holds a discussion of open questions and further research topics that came up throughout this thesis. This work closes with the conclusions in Section 6.3.

6.1 THESIS QUESTIONS: DISCUSSION

The thesis questions, introduced at the very beginning in Section 1.3 have been guiding the development and contents of this work. The questions and their respective answers are summarised within this section in the order of the question numbers.

6.1.1 *What is the Curse of Dimensionality?*

The first question Q_1: *"What is the 'curse of dimensionality' in the framework of clustering and under which circumstances does it occur?"* is a question for describing the mathematical background of the problems, that arise within high-dimensional feature spaces. The 'curse of dimensionality' is described by the occurrence of distance concentration in high-dimensional feature spaces. As defined in Equation (2.7), distance concentration occurs, if and only if the relative variance of distances becomes 0 for increasing dimensionality:

$$\lim_{m\to\infty} \mathrm{RV}\left(D^{p}_{\vec{q}^{(m)}}(\vec{X}^{(m)})\right) = 0$$

A similar definition, can be given for real data sets, based on estimators for variance and expectation. Note that distance concentration is described in terms of the distribution of distances w.r.t. a query point. It is not based directly on the (marginal) distribution of the data set which makes predictions based on the data distribution difficult.

This step is overcome partly by the analysis in Subsection 2.2.3. From Equation (2.20), it is concluded, that

$$\lim_{m\to\infty} \mathrm{RV}\left(\|\vec{q}^{(m)} - \vec{X}^{(m)}\|_p^p\right) = \lim_{m\to\infty} \frac{(\overline{\sigma}^{(m)})^2}{(\overline{\mu}^{(m)})^2}$$

where $\overline{\sigma}^2$ is the average pairwise correlation of the distances of two distinct features and $\overline{\mu}^2$ is the mean expected value of the distances of all dimensions, centred to a query point $\vec{q}$ and scaled to a variance of 1. In other words, distance concentration basically depends on the amount of pairwise correlation of the data generating marginal distributions. The more positive correlations are present, the less likely it is that distance concentration occurs. This might be explained with the notion, that positively correlated dimensions lead to a lower intrinsically dimensionality because the data is located in or close to a lower-dimensional linear subspace.

With this result, it is for example easy to understand why meaningless features lead to distance concentration. They do not correlate to the actual data or to each other, thus the average pairwise correlation approaches 0 faster than the average marginal distances. This result also answers the first part of the thesis question in a satisfying way: the conditions for the curse of dimensionality can be described in terms of pairwise correlations among the absolute values of individual dimensions, centred on a query point $\vec{q}$.

The second part of this question, the curse of dimensionality in the framework of clustering is a bit more ambiguous. A clear description of the tolerance of clustering algorithms to distance concentration is still missing. What is needed is a consistent mathematical description of the influence of distance concentration on prototype based clustering algorithms. One way of describing this influence is presented in the approximation in Equations (5.5) and (5.6). If the approximation is possible, this might be an indicator that the algorithm is sensible to distance concentration.

Another description of the influence of distance concentration are the objective function plots (see Figures 5.5, 5.6, 5.7 and 5.10), generated using the simplex data set (see Figure 5.2). A local minimum at the centre of gravity in these plots indicate a tendency that the prototypes will run into the centre of mass of the data set.

6.1.2 *Which Clustering Algorithm?*

The second thesis question Q_2: *"Which prototype based clustering algorithms can be used for high-dimensional data sets?"* can be answered by stating that FCM_m, PFCM and RCFCM are good candidates for real applications. As became clear already in the introduction, FCM_2 is not a good choice since it was part of the motivation to start this thesis in the first place. During the tests, EMGMM was less effective than the other algorithms that are analysed, which makes a use of EMGMM less promising in high-dimensional feature spaces. HCM seems to be a bad choice for any clustering problem with more than 2 classes, independently of the number of dimensions and shape of the classes. Very surprisingly, noise variations of all algorithms performed less well than the basic variants of the algorithms in many tests where noise was present. This result puts a big question-mark on the use of noise sensitive algorithms in high-dimensional clustering algorithms, a deeper analysis might be necessary here. Maybe a better way to estimate the noise distance might solve the issue within this thesis.

Among FCM_m, PFCM and RCFCM however, the question which algorithm is best cannot be answered independently from the data set that is supposed to be clustered. All three algorithms seem to be useful, though it is not a-priori clear, which is best in a given situation. FCM_m is a bit harder to set up because the cluster quality depends on the correct value of the fuzzifier parameter and it is hard to anticipate the correct fuzzifier value. For PFCM and RCFCM, the choice is easier because the clustering is more robust w.r.t. the fuzzifier parameter. However, the performance of FCM_m seems to be better for more complex data sets, see the results for data set families D_3 and D_4. This indicates that tests based on simple data set structures like Gaussian shaped classes are not indicative for the performance of a clustering algorithm in high-dimensional feature spaces. In practice, any of the three algorithms might be tried.

A more general advice can be given at the end of the next subsection.

6.1.3 *Influence of Dimensionality and Cluster Count*

The third thesis question Q_3: *"How is the clustering quality influenced by the number of dimensions and the number of clusters?"* can only be answered clearly for FCM_2. The FCM_2 algorithm has a strong dependency on the number of dimensions and the number of clusters under distance concentration, see for example in Figure 5.4. This is also supported by the deeper analysis presented in Section 5.1.2. The other algorithms seem to be less sensitive to the number of dimensions and classes/clusters. All algorithms perform less well for low

number of dimensions and high number of classes, as can be expected because the classes overlap and are not distinguishable. The overlapping is especially strong for data set family D_3, which uses the distorted classes and is designed to cause this problem.

The number of clusters seem to have a stronger influence on the clustering quality than the number of dimensions for FCM_m, PFCM, RCFCM, their noise clustering variations as well as EMGMM. In fact, with sufficiently high number of classes/clusters, the performance of the algorithms increase with the number of dimensions due to the problem of overlapping. So contrary to the effect for (N)FCM_2, it is worth increasing the number of useful features in a data set if the number of classes is very high. HCM provides a constant, low clustering quality, independently of the number of dimensions and clusters as long as the number of clusters is larger than three.

The membership value plots (see Figure 5.12), presented in Section 5.1.7 give a good indication on the performance of a fuzzy clustering algorithm in a high-dimensional environment. From Chapter 2, it is known that in a high-dimensional feature space, distance concentration is likely to be a problem. The membership values form a mathematical way of interpreting the relative distances between query points (prototypes) and data objects. The problems of distance concentration can be mitigated by interpreting small differences in relative distances with strong changes in membership. Or in more abstract terms, a fuzzifier function that magnifies the small differences in distances is better suited for high-dimensional feature spaces than a fuzzifier function without that magnification effect.

In this way, the question of how the number of dimensions influences the clustering process can be answered quite clearly: the concentration of distances causes the problems and is mitigated by an appropriate choice of the clustering algorithm. The influence of the number of classes/clusters is more subtle, but it does not seem to have a big influence on distance concentration resistant algorithms other than the problem of overlapping classes. However, such a solution to the thesis question is not well defined for all algorithms. For example the membership value plots method does not work well for EMGMM because the width parameter for clusters can alter the shape of the plots. For crisp clustering methods like HCM, the method is completely useless because the fuzzifier function is not continuous.

In the introduction, it was stated that the answer to this thesis question might give insight into the development of clustering algorithms that are resistant to distance concentration (see Section 1.3). This is indeed the case here, at least for fuzzy clustering algorithms. If a membership value plot like it is shown in Figure 5.12 can be produced for a new fuzzy clustering algorithm, its derivation in the middle between two prototypes indicates how well the algorithm can resist the concentration of distances.

6.1.4 *Cluster Quality Indices*

The last thesis question Q_4: *"Are internal cluster quality index measures useful to assess the quality of clustering results in high-dimensional feature spaces?"* can be answered quite clearly with 'no'. At least, if one or a combination of the most popular and widely used indices are considered: BS, DB, XB, NPC or NPE (see Section 4.3). The analysis presented in Section 5.3 as well as the data presented in Appendix G indicate, that internal indices can not be trusted. As has been discussed at the end of Section 5.3, the problem is not strongly connected to the number of dimensions or number of classes/clusters.

The indices can not be used reliably for ranking of clustering results (see Section 5.3.1 and Appendix E). Likewise, the true cluster quality indices are in too many cases negatively correlated to the true cluster quality, as measured by the external F_1 index (see Section 5.3.2 and Appendix F). Even worse, the values of the internal indices are often dominated by the selection of clustering algorithms that are compared (see Appendix G). All this together means, that the use of the BS, DB, XB, NPC and NPE indices are questionable and should be avoided in practice.

6.2 CRITIQUE AND FUTURE RESEARCH

In this section, the limitations of this work as well as unanswered questions and topics for future research are discussed. Some of the points that are listed below might contribute to the results in this work and have good potential for further research. A good scientific practice would be to challenge the results of this work, some of the points below indicate where such a challenge can best lead to new insight.

6.2.1 *Limitation of the theoretical result on the Euclidean Distance*

The results in the sections 2.2 of Chapter 2 are limited to distance functions, induced by a p-norm, raised to the power of p. These distance function are by far the most common distance functions in practical applications. The squared Euclidian distances is one of these distance functions and it is also required by the clustering algorithms, so the choice is well justified. However, if the results presented in Chapter 2 are as fundamental as they seem, it might be possible to extend the results to general distance functions and use the triangle inequality instead of the exact decomposition into its individual components. The potential to extend the assumptions to a more general level would be very welcome. If the results of Chapter 2 were proven for general distance functions, an extension to more general feature spaces (that

are not vector spaces for instance) might be possible. Therefore, the distance concentration argument would not require a transformation into $\mathbb{R}^m$, which opens up a far wider field of applications.

6.2.2 F_1 *measure*

I cannot find any argument why the F_1 should be questionable but that does not mean the decision is without drawbacks.

One of the most influential decision for the answering of thesis questions Q_2, Q_3 and Q_4 was to use the F_1 measure (presented in Section 4.2.2) as external quality index. The F_1 index values are interpreted as the true clustering quality. If, for any reason, this assumption is false, the results in Chapter 5 become questionable at best. If anyone has doubts in the results that are presented in this work, this might be the point to start an investigation. An independent confirmation of the presented results using a different external index measure would be also very useful.

6.2.3 *Selection of Clustering Algorithms*

The selection of clustering algorithms (see Chapter 3) is somewhat limited. There are so many clustering algorithms, that it would easily be possible to find 5 to 10 more prototype based clustering algorithms which would fill in the current scheme. So the selection of clustering algorithms might not be optimal, and for sure is not complete. However, the line has to be drawn somewhere. At the time when the selection of the 10 algorithms was made, I thought that kernel based clustering algorithms are not well suited for clustering high-dimensional data. Kernel based clustering algorithms transform the already high-dimensional feature space into an even higher-dimensional feature space which does not seem to be a good idea. In retrospect, this might not have been a valid assessment. Kernel based clustering algorithms might be a way of limiting the curse of dimensionality by selecting a suitable kernel function. The trick would be to find a good kernel function, which might be a topic for another research project.

Also other types of clustering algorithms are completely neglected, like hierarchical methods, density based algorithms like DBScan [Ester et al., 1996] or grid based clustering algorithms like OptiGrid [Hinneburg and Keim, 1999]. These algorithms are not easily comparable to prototype based clustering approaches because they do not provide a prototype or another natural centre of the clusters. Without this centre, the relative variance is not as naturally defined for these algorithms because they miss the necessary query point for the concentration of distances. Still, a comparative analysis of these methods with prototype based clustering methods might be very interesting.

For extending the number of algorithms and performing similar experiments, the data mining tool EDMOAL can be used, see Appendix B.

6.2.4 *Selection of Data Set Families*

As with the selection of algorithms, only a small selection of data set families could be considered in this work, see Chapter 4. There are good reasons to choose them. First, it was important to have a very easy data set family D_1, which can work as a sanity check. If a clustering algorithm fails to cluster a data set from this family, there is not much hope in applying it on any other, more complex data set. D_2 was chosen because instances from this data set family are often discussed in papers to prove the usefulness of clustering algorithms. It was expected that algorithms, performing well on this data set family do not perform well on more complex ones. This expectation is satisfied as was shown by the comparison with data set family D_3. D_3 was chosen because it simulates complex functional connections between dimensions within one cluster. In reality, no cluster is nicely normal distributed. The idea behind D_3 was, to produce a data set family that is a challenge for the clustering algorithms by producing classes that live in complex lower dimensional manifolds that overlap and are in general unpredictable. D_4 is used because the data set family is similar to the aircraft movement of the data set presented in Section 1.1.3 and to test how algorithms work in an environment that they are not directly designed for.

This selection of data set families are by no means complete and in reality, data sets have far more complex and unique properties. A wider range of data set families is on top of the wish-list if the results of this work should be extended in some way. Performing all the experiments took several months of computing time on 12 cores in parallel, more data set families would require even more computing time, which I simply did not decide to invest.

Again, for extending the data set families and performing similar experiments, the data mining tool EDMOAL can be used, see Appendix B.

6.2.5 *Missing Real World Data Set*

In this work, no real world data set is discussed. This has one negative and one positive aspect. First the negative aspect: the observations of this work do not have a strong link to the real world. This work could be improved by addressing multiple real world high-dimensional data sets and observe the effects that are discussed theoretic-

ally. Such an analysis however would require substantial work, maybe enough for an additional thesis.

The positive aspect is, that it was possible to create such a huge range of artificial experiments and present them in a useful way. The artificial data sets presented in Chapter 5, turned out to be quite useful because of the surprising results, especially related to the internal cluster quality index values. In fact if I had discussed one or more real world data sets, the investigation presented in this work would likely not have happened in this way. I might have missed the deeper problems of the internal indices for instance. Even worse, I probably would have used internal indices to determine the best way of clustering the real data set because I believed prior to this work that the internal indices can be trusted enough. As can be learned from the answer to thesis question Q_4, this trust is not well founded.

6.2.6 *Development of Reliable Internal Quality Indices*

As already mentioned in the last subsection, one of the surprises of this work is the insight how the internal quality indices can be biased. Especially the fact that the selection of clustering algorithms has such a strong influence on the usefulness of the index (see the correlation plots in Appendix F and the detailed plots in Appendix G). Even though the problem of the indices is only weakly connected to the number of dimensions, the problems are more apparent for a high number of dimensions. So the canonical task would be to develop an internal index measure, which does not introduce the observed bias. Due to time and space limitations however, I chose not to do this, so it remains an open research target.

Reliable internal indices can be regarded at least as important as the clustering algorithms themselves. Clustering algorithms almost never produce an optimal result because the objective function often falls into a local optimum. Without a way to test their results and compare different clustering algorithms, it is not possible to know how reliable any clustering result is. A good internal cluster quality index is therefore needed just as much as a reliable clustering algorithm.

An alternative to internal quality indices is to create a Monte-Carlo simulation [Irle, 2005] of the data set that should be clustered. Such a simulation can then be used to benchmark the clustering algorithms, similarly to the approach in Chapter 4. This approach is of course very time consuming and requires a lot of knowledge about the data set. It is therefore not likely to be used in practice. With the lack of a reliable internal index measure however, a Monte-Carlo approach might be the only option if the clustering result cannot be trusted.

6.2.7 *Distance Concentration Resistance of Clustering Algorithms*

A mathematical connection between the theory of distance concentration and the application of clustering algorithms is currently not well developed. As already indicated in the second half of Section 2.4 and again in the answers to the thesis questions Q_1 and Q_2 in Sections 6.1.1 and 6.1.2 the knowledge of the tolerance of a clustering algorithm to distance concentration is very limited. With the sufficient condition for distance concentration, described in Section 2.2.3, it is desirable to decide if distance concentration is a problem for a given algorithm. This requires the knowledge of how much distance concentration an algorithm can tolerate. I worked on several ideas in this direction, but unfortunately, I did not find a well defined metric. As mentioned above, the simplex analysis (see Section 5.1) and also by the membership value plots (see Section 5.1.7) might give a first indication on the tolerance of an algorithm to distance concentration, but this topic is worth further investigations.

6.2.8 *Distance Concentration Index for a Data Set*

In Section 2.4, a test for distance concentration is outlined. A gradient ascending algorithm was implemented to find the maximum relative variance in all test data sets, discussed in Chapter 4. Unfortunately, the standard gradient ascending algorithm did not deliver useful results, despite multiple reinitializations of the starting point. I chose not to include these results in Chapter 5, because they were not useful. The reason was, that the optimization algorithm converged too slowly due to a very weak gradient. It would probably be better to use some more sophisticated (e.g. swarm optimization) algorithm to find the maximal relative variance. A priori, it is not possible to know which algorithm would be best suited for this task, so the only true solution would be to implement and execute several different algorithms. Due to the already high execution time (several months) of the experiments in this work and the additional workload to implement such algorithms, I decided against it. An important factor for this decision is also the missing metric for the tolerance of distance concentration for prototype based clustering algorithms, discussed in the last subsection Section 6.2.7. Without the metric, a deeper analysis of the relative variance of a given data set would be only half the information that is needed.

6.2.9 *Critique on Publication Practice*

I want to address one more general topic related to the scientific practice in data mining. Many researchers that I talked to were very sur-

prised to see FCM_2 fail in high-dimensional feature spaces. In this work, I analysed in great detail why that happens, but this is not the point I want to make. The problem I see is that the failure of FCM_2 came as a surprise to many. It reveals an issue within the data mining community that is apparent in many papers that claim to have found new algorithms to solve a problem. In these papers, it is seldom written what the limitations of the new algorithm are, the focus is always on its capabilities. Often, new algorithms are introduced by showing how they can solve a problem better than other algorithms. In that sense, the limitations are discussed only if there is another algorithm that overcomes these limitations. I do not exclude myself from this practice, in fact, all papers that I contributed to and that announce new algorithms follow the same scheme, see [Rehm et al., 2010; Winkler et al., 2010a,b, 2011b,c, 2013]. The culture of only presenting positive results has lead to the situation that researchers can be surprised by limitations of standard algorithms like FCM_2 and HCM. This is not good because the usefulness of an algorithm in practice depends on its limitations and the knowledge thereof.

6.3 CONCLUSIONS

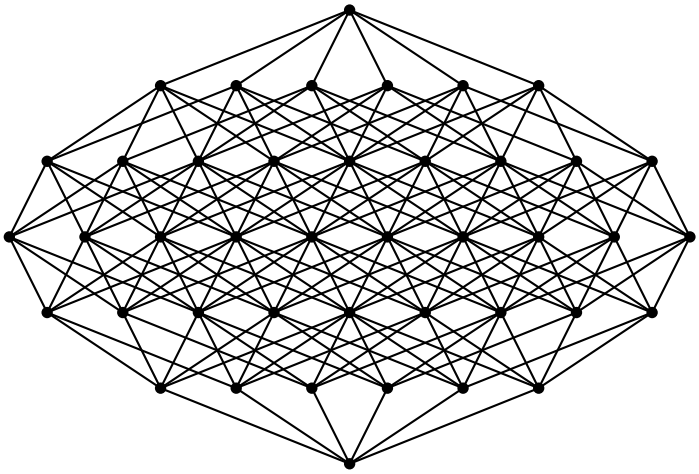
By answering the thesis questions Q_1, Q_2, Q_3 and Q_4 (see Section 6.1), many major difficulties of clustering in high-dimensional feature spaces are addressed. In Chapter 2, it is proven that distance concentration can be expressed by the two-dimensional marginal distributions for any high-dimensional probability distribution if a p-norm induced distance function is used. The approach of combining distance function and data distribution is new and was not published in this fashion before. It increases the understanding in effects related to the curse of dimensionality and made the proof of distance concentration for dependent features possible, which was also never accomplished before.

So far, the performance of clustering algorithms are almost exclusively compared based on small number of selected data sets. In contrast, a large number and variety of simulated data sets and clustering results have been produced for this work. This survey allows the comparison of some of the standard prototype based clustering algorithms with more specialized clustering algorithms in a meaningful way. The results of the simulated data sets align nicely with the idea that a steep gradient in membership values is advantageous for clustering in high-dimensional feature spaces.

The large amount of simulated data allowed furthermore an investigation on the performance of commonly used internal cluster quality measures. Until now, nobody has done such a survey. I show in this work that the values of the indices are biased by the selection of clus-

tering algorithms. In some cases, the indices are even negatively correlated with the true cluster quality which negates their usefulness. This result is not restricted to high-dimensional feature spaces and poses a serious problem for the clustering community.

The source code that was used to create the data sets and generate the results within this work is available as open source project EDMOAL. Similarly, the data is published under a creative commons licence which allows anyone to recreate, expand and challenge the results.



Appendix

A

ARTIFICIALLY GENERATED DATA SETS

In this appendix, a description of the randomly generated data objects is provided. The random generation of data objects utilizes pseudo random numbers, generated by the virtual machine in JAVA, utilizing the random number generating algorithm, described in [Knuth, 2005] Volume 2, Chapter 3: Seminumerical Algorithms, Section 3.2.1.. Since the generation of random numbers is still a deterministic process, a chance of a hidden structure in the data can not be eliminated. However, the probability of this happening is marginally low and is ignored.

The 4 families of data sets have several things in common. To answer Q_2, Q_3 and Q_4, all data sets are scalable in the number of dimensions m and the number of classes c. The data objects within the individual classes are always generated independently of any other class. This property is utilized by generating the classes before assembling the data sets. So for each number of dimensions m and for each data set family, a set of 250 classes are generated. The data sets within a data set family are then assembled using a random selection of the 250 pre-generated classes. The data objects live in $\mathbb{R}^m$ and are not strictly bound to the unit hypercube $[0,1]^m$, but are generated close to it so that no rescaling prior to applying the clustering algorithms is necessary. There are no missing values and no extreme outliers or invalid values in the data.

A.1 SPHERICAL NORMAL SHAPED CLASSES OF IDENTICAL SIZE

The first data set family D_1 is deliberately designed to be easy for clustering algorithms. It can be used as a sanity check whether or not a clustering algorithm produces a sensible result on high-dimensional data and it can be expected that any clustering algorithm generates almost perfect results on any data set of the D_1 family. The classes for D_1 are sampled from spherical normal distributions with identical variance $\sigma^2 = 0.01$ and 1000 of data objects per class. The expectation vector of the generating normal distribution are sampled uniformly from the unit hypercube and therefore does not need to be parametrized. By placing the expectation vector inside the unit hypercube, the

Of course, you know from my introduction in the first chapter, that at least standard FCM fails the 'sanity check' for $m > 10$.

data objects, sampled from the normal distributions might occasionally be sampled outside the unit hypercube, but they are still close enough such that no rescaling of the data sets is necessary.

The left-hand side of Figure 4.1 in page 85 shows a 2-dimensional example of D_1, with colour indicating the class information. In 2-dimensions, it is likely, that some classes overlap and it is quite possible that some classes are clustered within the same cluster due to the overlap. However, in dimensions higher than 5 or for a low number of classes, this data set family should be perfect for prototype based clustering and any decent clustering algorithm should be able to find a perfect representation of the classes.

A.2 SPHERICAL NORMAL SHAPED CLASSES OF VARIOUS SIZES

The second family of data sets D_2 is similar to the first, but more challenging. The classes are still sampled from spherical normal distributions, but with variable variance and number of data objects per class. The number of data objects is sampled from a uniform distribution on the interval between 200 and 1800. The variance of the normal distributions (classes) is also sampled from a uniform distribution on the interval between 0 and $\sigma_{max} = 0.02$, independently of the number of data objects. Finally, to each data set in D_2, a fixed fraction of noise data objects are added that are sampled from a uniform distribution on the unit hypercube $[0, 1]^m$. The number of noise data objects is set to be 1/10th of the total number of data object in a data set. Since the number of data objects per class is randomly sampled, the number of noise data objects is determined after the classes for one data set are selected. The right-hand side of Figure 4.1 shows a 2-dimensional example of D_2. As in case of D_1, data objects might be sampled outside of the unit hypercube, but due to the construction of the classes, the data objects can not be sampled far away from it.

A.3 DISTORTED CLASSES DATA SET

Data sets of the families D_1 and D_2 are very unrealistic. In reality, data sets do not contain neat and spherical classes, they can have all kinds of complex shapes. The family of data sets D_3 is designed to mimic such complex dependencies by producing very distorted and strangely shaped classes.

A data set of the D_3 family is constructed as before, from c, independently generated classes in a m-dimensional real vector space, bound to the unit hypercube. All classes contain a random number of data objects, again sampled from a uniform distribution on the in-

terval $[200, 1800]$. The final data sets of D_3 also contains noise data objects, obtained in the same way as for data sets in D_2.

The process of building the classes however, is much more complicated as in case of the D_2 data set family. Each class is generated individually and the shape of the class is generated iteratively. The iteration process is started with a sample from simple probability distributions, a mixture of an m-dimensional gaussian distribution and a normal distribution, bound on the unit hypercube $[0, 1]^m$. After initialization, unary and binary distortion functions are applied on the data objects of the class, forging distortions and pairwise dependencies within each iteration. The functions are randomly selected from a pool of predefined functions. In total, $t_{max} = 2 \cdot m$ iterations are performed, which consist of three individual steps:

1. For all dimensions $k \in \{1, \ldots, m\}$, randomly select one unary function $f \in \{u_1, \ldots, u_6\}$ and apply it on dimension k of each vector in a class $\forall \vec{x}^{(t)} \in C^{(t)}$: $\vec{x}'_k = f(\vec{x}^{(t)}_k)$, creating C' in the process.

2. For all dimensions $k \in \{1, \ldots, m\}$, randomly select one other dimension $r \in \{1, \ldots, m\}$, $r \neq k$ and randomly select one binary function $g \in \{b_1, \ldots, b_5\}$ and apply it on dimension k of each vector in the class $\forall \vec{x}' \in C' : \vec{x}''_k = g(\vec{x}'_k, \vec{x}'_r)$, creating C'' in the process.

3. For all dimensions, $k \in \{1, \ldots, m\}$ normalize the data to keep it well confined in the unit hypercube, creating the final value of $x_k^{(t+1)}$ for this iteration step: $\forall \vec{x}'' \in C'' : \vec{x}_k^{(t+1)} = \text{normalize}(\vec{x}''_k)$, generating $C^{(t+1)}$

Each iteration step is applied independently from the previous until $t_{max} = 2 \cdot m$ iterations are reached and the process is stopped. After the iteration process is finished, the data objects within the class are pushed in direction of one of the randomly selected corners of the unit hypercube.

A.3.1 *Initialization*

The data is initialized using two different distributions. The first half of the initial cluster data objects $\vec{x}_1^{(0)}, \ldots, \vec{x}_{\lfloor \frac{n}{2} \rfloor}^{(0)} \sim U([0, 1]^m)$ is sampled independently from a uniform distribution on the unit hypercube. The second half $\vec{x}'_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, \vec{x}'_n \sim N([0]^m, 1)$ is sampled from a spherical m-dimensional normal distribution. To map it to the unit hypercube, it is normalized by a linear normalization function $f_{\text{norm}} : \mathbb{R}^m \to [0, 1]^m$: for all normal distribution sampled data objects $j =$

$\lfloor \frac{n}{2} \rfloor + 1 \ldots n$, apply the normalization function for all dimensions $k = 1 \ldots m$ individually:

$$\vec{x}^{(0)}_{jk} = f_{\text{norm}}(\vec{x}'_{jk}) = \frac{\vec{x}'_{jk} - \vec{x}'_{\min k}}{\vec{x}'_{\max k} - \vec{x}'_{\min k}} \tag{A.1}$$

where $\vec{x}'_{jk}$ is the k'th dimension of the j'th data object and $\vec{x}'_{\min k}$ ($\vec{x}'_{\max k}$) is the minimal (maximal) entry of the k'th dimension of all the data objects that are sampled from the normal distribution $j = \lfloor \frac{n}{2} \rfloor + 1 \ldots n$. The initial cluster is $C^{(0)} = \{\vec{x}^{(0)}_1, \ldots, \vec{x}^{(0)}_n\}$. Figure A.1 shows the initial condition $C^{(0)}$ of the cluster as a projection on 2 dimensions.

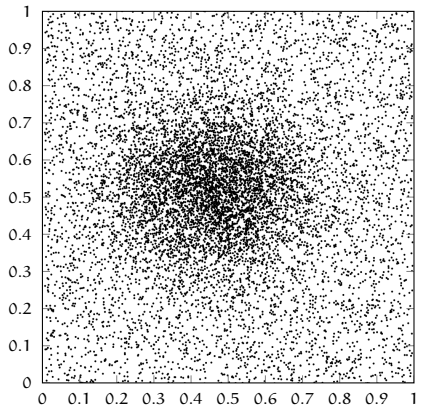


Figure A.1: Example distribution of the initialization of distorted classes. A sample of a combined uniform and normal distribution.

A.3.2 *Unary Functions*

It is very confusing to deal with corner cases like division by 0. This is effectively avoided by using functions on the unit interval only.

To keep the functions simple and avoid an uncontrolled spread of the values, all functions live in the unit interval. There are 6 different unary functions $u_1, \ldots, u_6 : [0,1] \to [0,1]$.

u_1: The first function is the identity $u_1(x) = x$.

u_2: The second function reverses the values, $u_2(x) = 1 - x$.

u_3: u_3 concentrates the values in the lower end of the $[0,1]$ interval. The function takes a parameter a that controls the level of concentration.

$$u_3(x) = \frac{(x+a)^2 - a^2}{(1+a)^2 - a^2}$$

If a would be set to 0, the slope of u_3 in 0 would be 0 as well, which would result in a strong concentration of values near 0. This is prevented by the parameter a. It is used to select

a specific section from a parable x^2. Due to the scaling between $[0, 1]$, the parameter basically defines how strong the concentration effect shall be. A small value for parameter a means a strong concentration effect, a large value means a small effect. With $a = 0.2$, good results were accomplished.

u_4: The opposite effect is accomplished with $u_4(x) = \sqrt{x}$. Being the square root, the function spreads the lower end of the unit interval. Here, no damaging over-concentration of values can occur, so no parameter is used.

u_5: The fifth function u_5 projects the data values in direction of the centre of the unit interval.

$$u_5(x) = \frac{1 + \sqrt{x} - \sqrt{1 - x}}{2}$$

u_6: The last function is somewhat special because it takes all current values of cluster C for a specific dimension k into account. It is used to give an opposite pole for the concentration effects of values by spreading them over the the entire interval. Let μ_k be the truncated mean of all values of the k'th dimension $\{\vec{y}_k \mid \forall \vec{y} \in C^{(t)}\}$ and σ_k be the corresponding sample standard deviation.

$$u_6(x) = \frac{1}{1 + e^{-\frac{x - \mu_k}{\sigma_k}}}$$

u_6 is part of the family of logistic functions.

The unary functions, u_3 to u_6 and their effects on the data are visualized in Figures A.2 and A.3. The u_3, u_4 and u_5 are pretty easy to grasp and they bring a certain random element into play whet it comes to producing dependencies by applying binary functions. u_6 is special in the sense, that it counteracts the concentration effects, introduced by the binary functions. In Figure A.3 in the left panel, a randomly generated and distorted cluster is presented. The distortion process was tweaked to generate a condensed cluster, which is then analysed and spread out by applying u_6, in x and y dimensions successively. The process is visualized by plotting the distortion function of u_6 over the data objects.

I experimented with many variation of functions and also random selection of parameters, not only in f_2. In the end, it turned out that this is not necessary and makes everything more complex than needed. If you look into the source code of EDMOAL, a process of randomly choose parameters is still present, but it is not used.

A.3.3 *Binary Functions*

The binary functions are defined analogously to the unary functions. There are 5 different binary functions $b_1, \ldots, b_5 : [0, 1] \times [0, 1] \to [0, 1]$. The binary functions are there to create dependencies among the initially uncorrelated dimensions of a cluster.

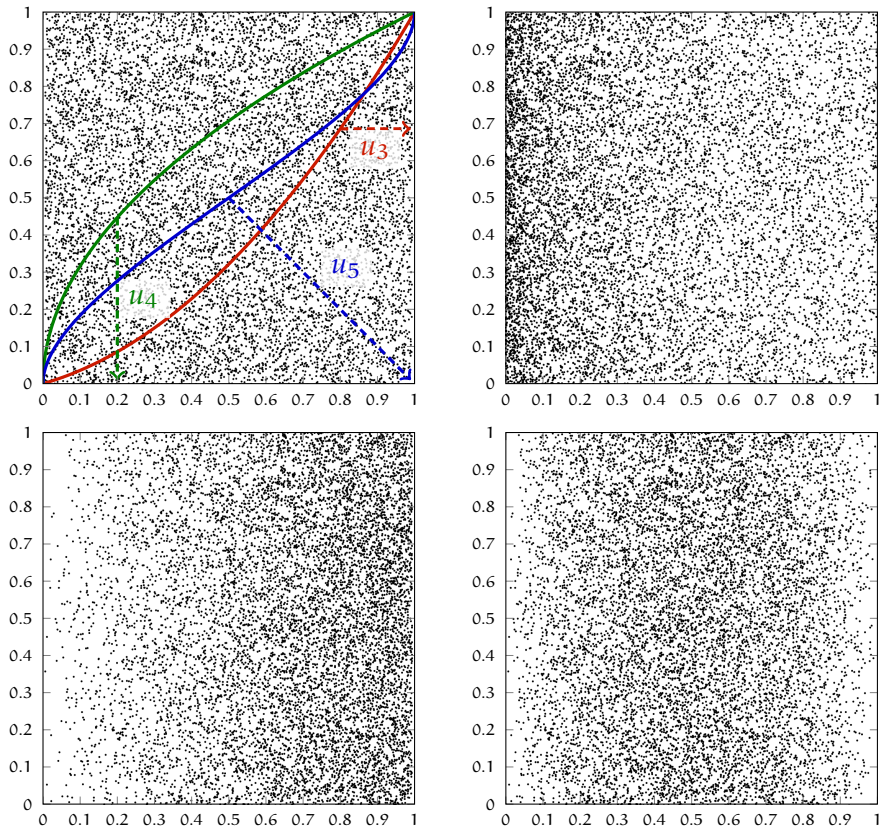


Figure A.2: Uniform distribution of data (top left), distorted on the x-axis by u_3 (top right), u_4 (bottom left) and u_5 (bottom right). The functions that were used to distort the x-axis indicate how the values were changed in order to achieve the distortion of the data.

b_1: The first function is again, the identity for the first parameter, $b_1(x, y) = x$ This function is needed for technical reasons and testing. it is not used during the distortion process of the classes.

b_2: $b_2(x, y) = \frac{x+y}{2}$ defines a addition between two values, scaled on the unit interval.

b_3: The third function provides a similar effect as b_2, utilizing the multiplication: $b_3(x, y) = \sqrt{x \cdot y}$.

b_4: The fourth function is not commutative, as the others before.

$$b_4(x, y) = x \cdot e^{y-1}$$

b_5: The last binary function simulates the shape of a part of a circle.

$$b_5(x, y) = \frac{x}{2} + \sqrt{\left(\frac{1}{2}\right)^2 - \left(y - \frac{1}{2}\right)^2}$$

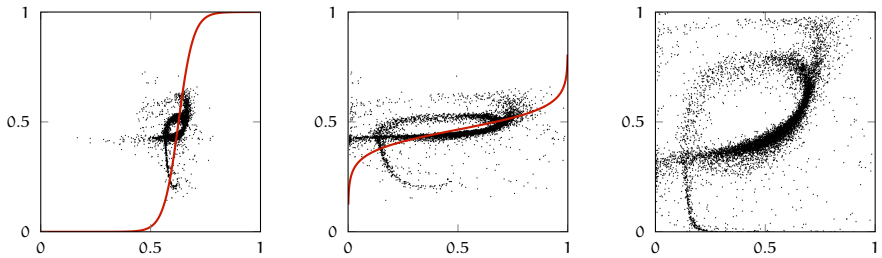


Figure A.3: Distortion function u_6 that spreads data objects, applied on both dimensions. The data (left panel) is distorted first on the x-axis (middle): $u_6(x) = \frac{1}{1+e^{-\frac{x-0.62708}{0.03385}}}$ and then the y-axis (right): $u_6(y) = \frac{1}{1+e^{-\frac{y-0.46507}{0.04941}}}$. The overlay printed functions indicate the respective distortion function, adjusted to fit the data.

It is derived from circle formula in 2 dimensions, $r^2 = x^2 + y^2$ with radius $r = \frac{1}{2}$ and centred on $\frac{1}{2}$, the centre of the unit interval.

In the image series of Figure A.4 the binary functions b_2 to b_5 are visualized.

A.3.4 *Normalization*

Unfortunately, I did not find a way to create distortion functions that do not require this artificial normalization process.

Many successive applications of the above discussed functions can lead to a contraction of values, condensing most data objects close to a single point. To prevent this, all data objects of a cluster are normalized on the unit interval by applying f_{norm} from Equation (A.1). As a result of the normalization, in all dimensions $k = 1, \ldots, m$ at least 2 data objects exist, with $x_k = 0$ or $x_k = 1$. Over time, this can render the normalization useless, because once, a value is set to 0 or 1, it is unlikely to change because both, unary and binary functions tend to keep extreme values intact. This effectively produces outliers, which is not intended by this procedure. To avoid this unwanted behaviour, extreme values of 0 and 1 are replaced in a random fashion. The new value is sampled from a normal distribution with expectation value and variance being the mean and sample variance of the current cluster data objects in the given dimension. Formally, let $\vec{x} \in C^{(t)}$ be a data object in cluster C of the t'th iteration and let the k'th value of $\vec{x}$ be in an extreme state $\vec{x}_k = 0$ or $\vec{x}_k = 1$. Then $\vec{x}_k$ is replaced with $\vec{x}'_k \sim N(\mu_k, \sigma_k^2)$, with μ_k being the mean of all values of the k'th dimension $\{\vec{y}_k \mid \forall \vec{y} \in C^{(t)}\}$ as well as σ_k^2 being the sample variance of all values of the k'th dimension $\{\vec{y}_k \mid \forall \vec{y} \in C^{(t)}\}$.

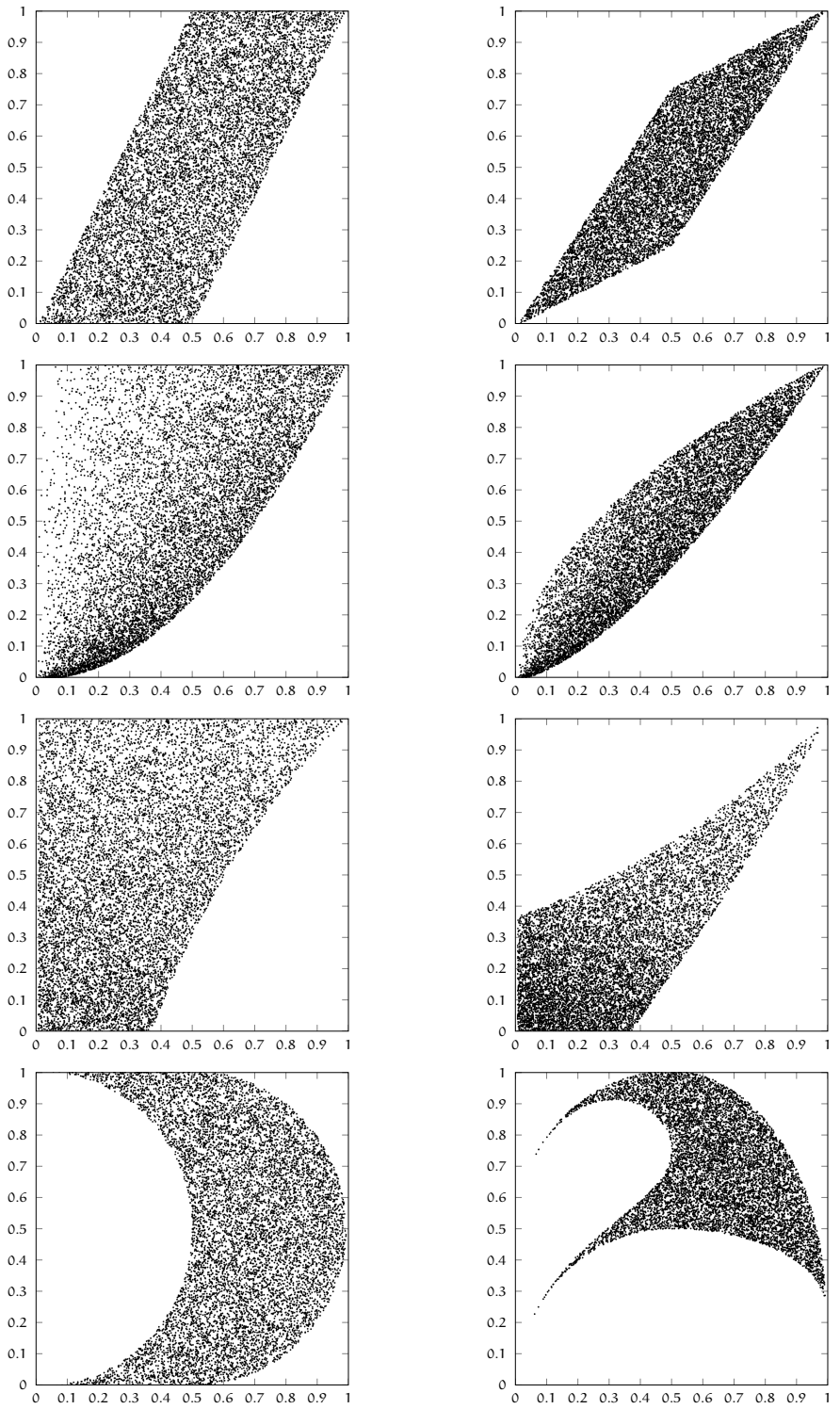


Figure A.4: Visualization of the binary distortion functions, applied on a uniform distribution, from top to bottom: b_2, b_3, b_4 and b_5. The left panels always show the binary functions applied on the x axis, with input form the y axis: $x' = b_\cdot(x, y)$. The right panels show the the reversed order, applied on the already distorted data: $y' = b_\cdot(y, x')$.

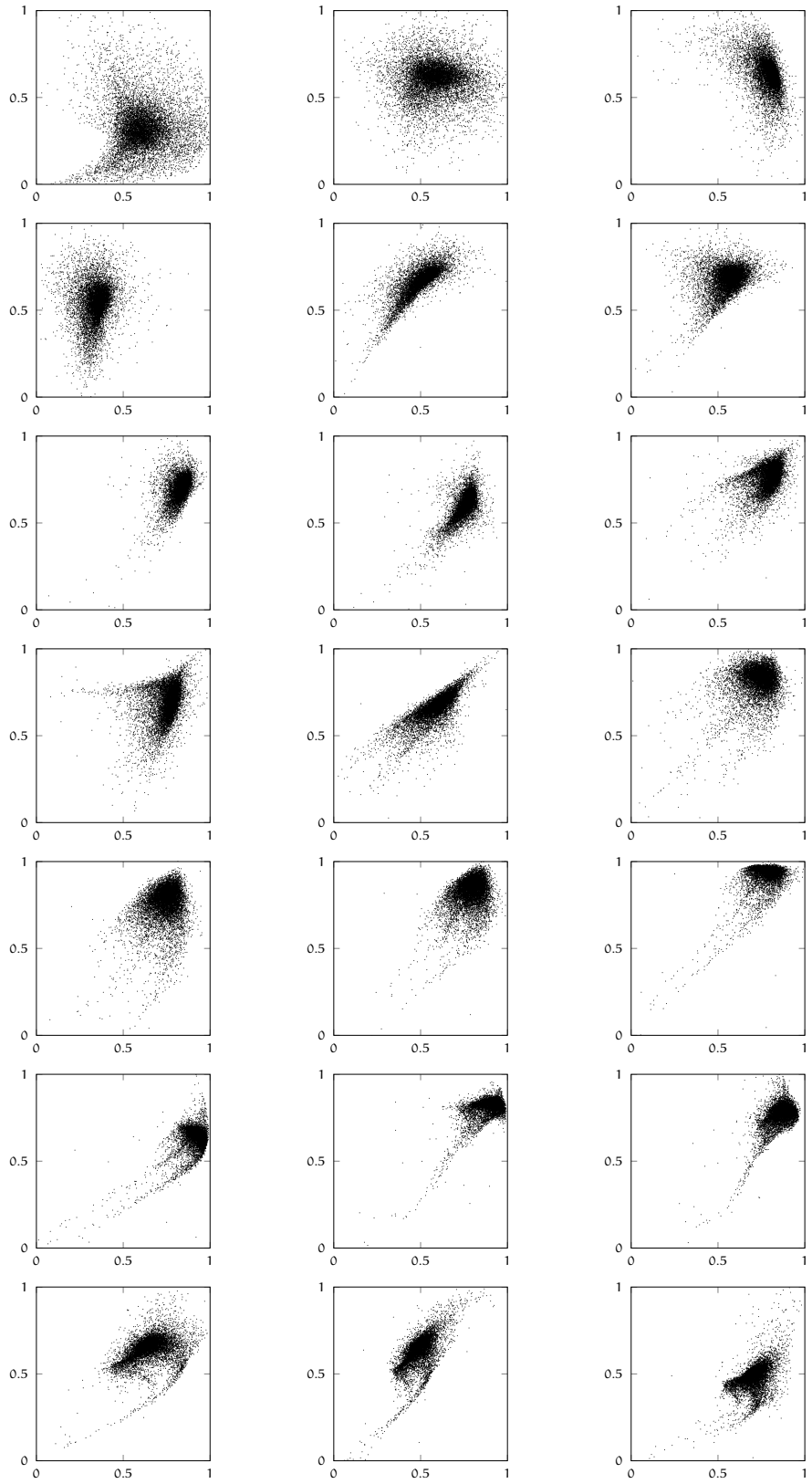


Figure A.5: Series of iterative distortion.

A.3.5 *Iteration*

Each iteration of the distortion process is independent of the previous ones. For each dimension, a unary function is randomly selected and applied on all data objects, thus altering the data values without introducing any correlations. The same is done with the binary functions, using 2 input dimensions and altering only one of them introduces correlations and in general dependencies among a pair of dimensions.

Formally, let $F^{(t)} = \{f_1^{(t)}, \ldots, f_m^{(t)}\}$, $f_k^{(t)} : [0,1] \to [0,1]$ be a set of m, randomly selected unary functions for the t'th iteration. Likewise, let $G^{(t)} = \{g_1^{(t)}, \ldots, g_m^{(t)}\}$, $g_k^{(t)} : [0,1] \times [0,1] \to [0,1]$ be a set of m, randomly selected, binary functions for the t'th iteration. Additionally, let the randomly generated permutation π of $1, \ldots, m$ with $\pi(k) \neq k$ define the index of the second argument of the binary functions.

To get from $C^{(t)}$, the class of data objects at its (t)'th iteration to its $(t+1)$'th iteration $C^{(t+1)}$, three steps are taken. First, the unary functions are applied, second the binary and last the normalization. Applying the unary functions on all data objects yields $C'^{(t)}$ with: $\vec{x}'_{jk} = f_k^{(t)}(\vec{x}_{jk})$, $j = 1, \ldots, n$ $k = 1, \ldots, m$. Applying the binary functions yields $C''^{(t)}$ with: $\vec{x}''_{jk} = g_k^{(t)}(\vec{x}'_{jk}, \vec{x}'_{j\pi(k)})$, $j = 1, \ldots, n$ $k = 1, \ldots, m$. Finally, normalizing the result yields the next level in the iteration steps $C^{(t+1)}$ with: $\vec{x}_{jk}^{(t+1)} = f_{\text{norm}}(\vec{x}''_{jk})$, $j = 1, \ldots, n$ $k = 1, \ldots, m$.

The quality of the result is very sensitive to the random selection process of the unary and binary functions. There are probably many more useful combinations than the presented, but since only one solution is sufficient, a detailed analysis of the influence of each parameter is not necessary. Let P^F (P^G) denote the discrete probability distribution for selecting the unary (binary) functions. For all dimensions, $k = 1, \ldots, m$, and all iterations $t > 0$, the unary (binary) distortion functions are sampled identical and independently from P^F (P^G). The probabilities of occurrence are in case of the unary functions:

$$P^F(f_k^{(t)} = u_1) = \frac{m}{m+41.7},$$

$$P^F(f_k^{(t)} = u_2) = \frac{0.5}{m+41.7},$$

$$P^F(f_k^{(t)} = u_3) = \frac{20}{m+41.7},$$

$$P^F(f_k^{(t)} = u_4) = \frac{10}{m+41.7},$$

$$P^F(f_k^{(t)} = u_5) = \frac{10}{m+41.7},$$

$$P^F(f_k^{(t)} = u_6) = \frac{1.2}{m+41.7}$$

and in case of the binary functions:

$$P^G(g_k^{(t)} = b_1) = 0,$$

$$P^G(g_k^{(t)} = b_2) = \frac{0.1m}{1+0.2m+\ln(m)},$$

$$P^G(g_k^{(t)} = b_3) = \frac{0.1m}{1+0.2m+\ln(m)},$$

$$P^G(g_k^{(t)} = b_4) = \frac{1}{1+0.2m+\ln(m)},$$

$$P^G(g_k^{(t)} = b_5) = \frac{\ln(m)}{1+0.2m+\ln(m)}.$$

The number of iterations t_{max} that are performed influences the result in an interesting way. The goal with the distorted data sets is, to generate data where the individual features (dimensions) are highly dependent (not necessarily correlated). In each iteration, each dimensions becomes dependent to at least one other dimension. The iterative nature of the algorithms provides an easy way to control the overall level of dependency. A value of $t_{max} = 2 \cdot m$ has been chosen to ensure a very high level of interdependent dimensions.

The series of images in Figure A.5 shows the distortion process on one cluster, applied on the initialization data, shown in A.1. Over time, the cluster is shaped in a way that is hard to reproduce using sampling of combinations of probability distributions.

A.3.6 *Finalization*

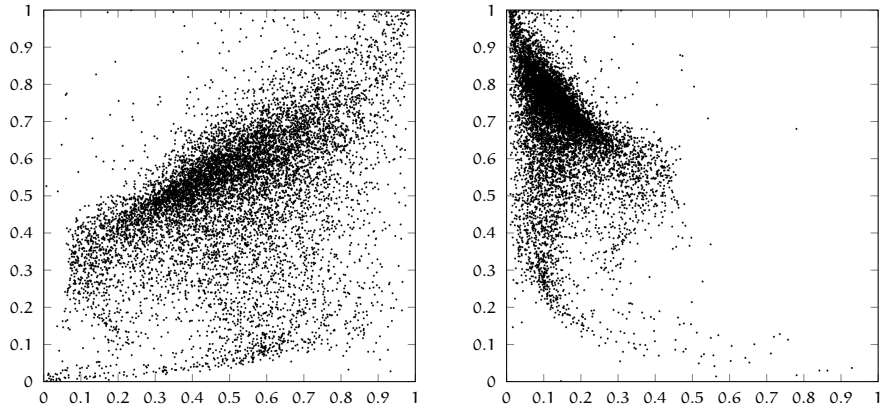


Figure A.6: Scaled and shuffled distorted cluster.

After a distorted classes are generated, several classes are compiled to form one data set. Due to nature of the distortion process, the centre of most classes are near the centre of the unit hypercube, which is not desirable. A finalization step is applied to spread the classes better and make there centres more distinct. The first step is to apply u_6 to all dimensions of a distorted cluster $C = C^{(t_{max})}$. This creates a cluster with a centre roughly in the centre of the unit hypercube. In a second step, the cluster is randomly pushed in direction of the $(1, \ldots, 1)$ corner by iterating 6 times unary functions with probability

distribution $P^F(f_k^{(t)} = u_1) = 0.7$ and $P^F(f_k^{(t)} = u_4) = 0.3$. Since all dimensions are independently altered, this gives a somewhat random concentration. Finally, one more iteration with $P^F(f_k^{(t)} = u_1) = 0.5$ and $P^F(f_k^{(t)} = u_1) = 0.5$ is applied, which gives a 50% chance of reversing a dimension. This last step randomizes the corner in which the class is pushed by the finalization step. In Figure A.6, the series of Figure A.5 is continued, first by applying u_6 (left-hand side) and then by shuffling the location of the cluster (right-hand side). More examples of clusters are shown in Figure A.7. The clusters come in so many different shapes an varieties, that it is impossible to cover them all, but at least, these examples give a good impression on what is possible.

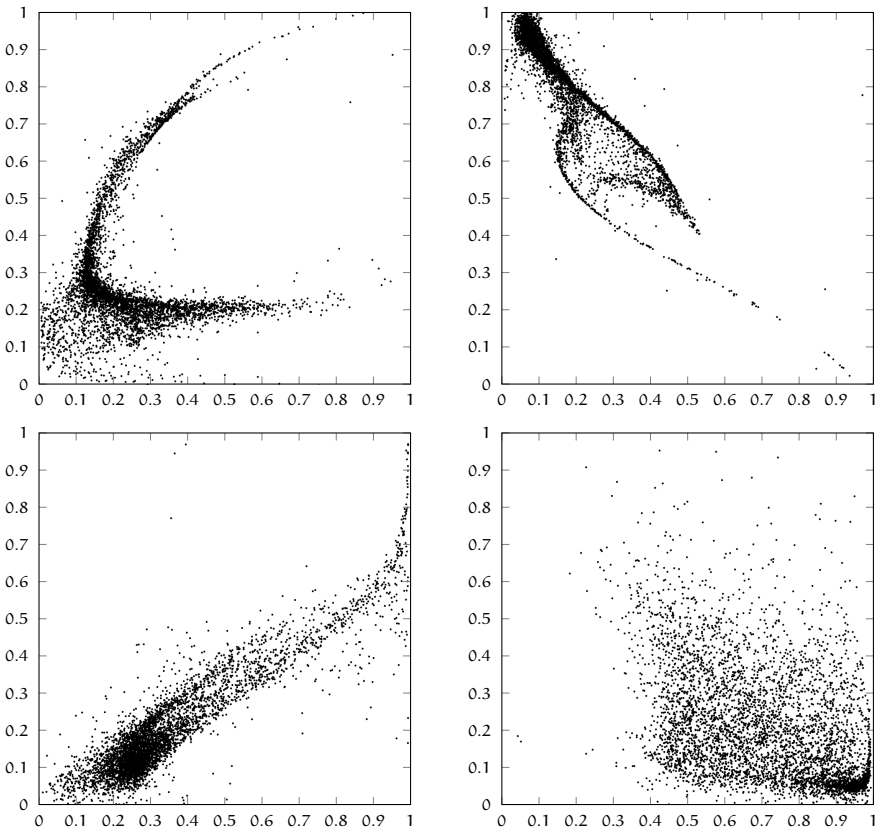


Figure A.7: Some more examples of distorted clusters.

This process of generating classes works quite well until around $m = 1000$ dimensions, at which point the 2-dimensional projection of the data distribution becomes very similar to a projection of a normal distribution. In other words, the distortion becomes so complex, that the sampling of a few thousand data objects is not enough to recognize the structure of the class. To cover even higher dimensions, the number of data objects must increase exponentially with the number of dimensions, as already mentioned in Section 2.5. Within the benchmark for this work, the number of dimensions is limited to 100,

which is well within the range of useful parameters for this method of artificial data generation.

A.4 CORNER CLASSES DATA SET

The fourth and final data set family D_4 for the benchmark consists of binary data objects that live in the corners of the unit hypercube $\{0,1\}^m$. Of course, there are dedicated approaches to analyse binary data like for example Frequent Pattern Mining [Aggarwal and Srikant, 1994]. But these approaches are not always best suited for clustering. In reality, it is very rarely the case that there is a clustering algorithm that is ideal for a given problem. In this context, it is particular interesting how the prototype based clustering algorithms deal with data, that is not really suited for prototype based clustering, such as the family of D_4 data sets. The data sets within the D_4 data set family are designed to be similar to Microarray data (see Section 1.1.6) or the aircraft movement data set, presented in Section 1.1.3.

The generation of classes for data sets within D_4 is done by generating seeds that are then randomly changed in order to generate data objects of the classes. Again, the number of data objects per class is sampled from a uniform distribution between 200 and 1800 data objects. As for D_2 and D_3, the data sets in D_4 have 10% randomly generated noise data objects.

The seed of one class is sampled from one of the corners of the unit hypercube, where a random number with expectation value $\sqrt{m}$ of the dimensions are sampled with 1 entries and the remaining with 0. Then, the data objects of that class are generated by randomly altering some entries of the seed, flipping some of the 1 entries to 0 and some of the 0 entries to 1. As for the data set family D_3, the classes in D_4 are generated independently from one another, hence the generation of classes is done one class at a time and a data set is assembled from previously generated classes.

A.4.1 *Cluster Seed Generation*

The clusters in D_4 are designed in such a way, that the number of 0-entries is (much) larger than the number of 1-entries. A seed $\vec{x}^{\text{seed}} \in \{0,1\}^m$, is used to generate the data objects that are relatively similar, but not necessarily identical for cluster C. So the number of 1-entries is sampled from a Poisson distribution with expectation value μ_s as long as the sampled value does not exceed $\frac{m}{2}$. Let $0 < s \leqslant \frac{m}{2} \in \mathbb{N}$ be the number of 1-entries of the seed, with $s' \sim \text{Poiss}(\mu_s)$ and $s = \min s', \frac{m}{2}$. Let $\pi \sim U(\text{Poly}(\{1,\ldots,m\}))$ be a randomly generated permutation, which is sampled from a uniform distribution on the set of all permutations Poly on the set of $\{1,\ldots,m\}$. The set of 1-

entries for the seed is defined by the first s values of the permutation π. $\vec{x}^{seed} = \{\vec{x}_1^{seed}, \ldots, \vec{x}_m^{seed}\}$ with for $k = 1, \ldots, s$: $\vec{x}^{seed}_{\pi(k)} = 1$ and for $k = s+1, \ldots, m$: $\vec{x}^{seed}_{\pi(k)} = 0$.

For example, a 10-dimensional seed with $s = 4$ 1-entries and a random permutation
$\pi =< 2, 7, 10, 9, 4, 6, 5, 8, 3, 1 >$ may be represented a string of values.

```
dimension: 1  2  3  4  5  6  7  8  9 10
seed:      0  1  0  0  0  0  1  0  1  1
```

A.4.2 *Cluster Data Generation*

The data is generated, using the seed as starting point and changing the entries slightly. Let the cluster $C = \{\vec{x}_1, \ldots, \vec{x}_n\}$ be a set of data object objects, $\vec{x}_j \in \{0,1\}^m$, $j = 1, \ldots, n$. Each data object is generated independently from the others.

Let $r_j'^1 \sim \text{Poiss}(\mu_0)$, $r_j^1 = \min r_j'^1, s$ and $r_j'^0 \sim \text{Poiss}(\mu_1)$, $r_j^0 = \min r_j'^0, m-s$ define two numbers $r_j^1, r_j^0 \in \mathbb{N}$ for the data object $\vec{x}_j$. The number r_j^1 describes how many elements of value 1 in the seed are set to be 0 in the data object $\vec{x}_j$, while r_j^0 defines the opposite, the number of 0-entries in the seed, that are set to be 1 when generating $\vec{x}_j$. Let $\pi_j^1 \sim U(\text{Poly}(\{\pi(1), \ldots, \pi(s)\}))$ be a randomly sampled permutation of the dimensions that are selected to have 1-entries in the seed. Likewise, let $\pi_j^0 \sim U(\text{Poly}(\{\pi(s+1), \ldots, \pi(m)\}))$ be a randomly sampled permutation of the dimensions that are selected to have 0-entries in the seed.

$\vec{x}_j$ is defined as follows:

$$\forall k = 1, \ldots, r^1 \colon \vec{x}_{j\pi(\pi^1(k))} = 0$$

$$\forall k = r^1 + 1, \ldots, s \colon \vec{x}_{j\pi(\pi^1(k))} = 1$$

$$\forall k = s+1, \ldots, s+r^0 \colon \vec{x}_{j\pi(\pi^0(k))} = 1$$

$$\forall k = s+r^0+1, \ldots, m \colon \vec{x}_{j\pi(\pi^0(k))} = 0$$

Data object 2 and 6 are identical. In set theory something like that would cause trouble because the elements of a set must be uniquely defined. Therefore, I consider the index of a data object as being part of its identity, even if that is not reflected in its mathematical description. Double entries are possible in reality as well and have no negative influence on the clustering process.

From the above mentioned 10-dimensional example seed, the following data objects might have been produced.

```
dimension:   1  2  3  4  5  6  7  8  9 10
seed:        0  1  0  0  0  0  1  0  1  1
data obj. 1: 0  0  0  0  0  0  1  1  1  1
data obj. 2: 0  1  0  1  0  0  1  0  0  0
data obj. 3: 0  1  1  0  0  0  1  0  1  1
data obj. 4: 0  1  0  0  0  0  1  0  0  1
data obj. 5: 0  1  1  1  0  0  1  0  1  1
data obj. 6: 0  1  0  1  0  0  1  0  0  0
```

A.4.3 *Finalization*

The D_4 data set family contains noise, however due to the binary nature of the data objects, the noise data objects is different than for D_2 and D_3. The data objects for the noise cluster $N \subset \{0,1\}^m$ are generated in exactly the same way as the seeds for the real clusters. In other words, the noise-seeds are treated as data objects and N is the set of all data objects, generated this way.

B

EDMOAL

EDMOAL is an acronym for 'Efficient Data Mining on Algebraic Limits'. It is the data mining tool that was used in order to generate the results in this work and is published with an open source license and hosted at github[1]. Anybody interested in the source code can find it there, the use of the distributed revision control software GIT is advised but not necessary.

EDMOAL contains all the clustering algorithms (and some more), discussed in this work (see Chapter 3). It also contains the internal and external cluster quality indices (see Sections 4.2 and 4.3) as well as the algorithms for generating the artificial data sets (see Section 4.1 and Appendix A). It does not however, include the execution of the benchmark experiments (see Section 4.4) nor the code for organizing and analysing the results (see Chapter 5). The setup of the experiments have no place in an API (see below) and also, the source code is not well documented and also not very readable.[2]

B.1 MOTIVATION FOR EDMOAL

B.1.1 *Why yet an other data mining tool?*

Many algorithms in data mining tools are implemented based on arrays or lists of floating point values. Data that can not be expressed by such an array, standard implementations of data mining algorithms often can not be used or an inaccurate transformation to floating point values has to be found. On the other hand, most algorithms only require a certain algebraic structure on the data objects, which is not necessarily limited to tuples of real numbers. So the limitation to floating point numbers is not a requirement of the algorithm, but it is imposed by its implementation. The same is true for data structures, which sometimes require a certain algebraic structure rather than a specific form like a floating point array.

Also it was great fun developing it. I learned a lot and it is very satisfying that it runs so well.

EDMOAL is build to apply algorithms and data structures to their full potential, limited only by the algebraic structure underlying the

1 `https://github.com/Roland-Winkler/EDMOAL`

2 I can send it via eMail if required, please contact me: `roland.winkler@gmail.com`

data. For example if a data mining algorithm only examines the distances between data objects, it is not necessary to have a vector space algebra. DBScan is an example for such an algorithm. However, all the clustering algorithms using prototypes as cluster representatives need a vector space on the data in order to compute the locations of the prototypes. For alternating optimization clustering algorithms, even an Euclidean vector space is required. But some algorithms only require the algebraic structure for what ever data that should be analysed. In case of data structures, a k-d-tree for example needs an m-dimensional vector structure while a ball-tree requires only a norm and a centred ball tree requires a normed vector space.

Keeping only the algebraic limits in mind for all algorithms and data structures, enables it to design one algorithm for all kinds of data, provided an algebraic structure is available for the algorithm. Most other data mining tools to my knowledge are not capable of that or only very limited. Therefore, I created EDMOAL in order to provide an algorithm and algebraic base that is not limited by floating point numbers.

B.1.2 *The efficient E in EDMOAL*

The algorithms in EDMOAL are designed to save first and foremost computation time where possible without obfuscating the code too much. Also they are designed to use as little memory as possible. For example for Fuzzy-c Means, it is not required to store the membership matrix during the clustering process, so it is not done.

In most other data mining tools, algorithms do not utilize a data structure in order to speed up data mining processes. For example DBScan implemented naively is in the runtime complexity class $\mathcal{O}(n^2)$ with n being the number of data objects. Implemented on a ball-tree requires only $\mathcal{O}(n \cdot k \cdot \log_2(n))$ with k being the minimal number of data objects for clusters, which is much faster for large data sets.

The algorithms in EDMOAL are designed to utilize a data structure if that is necessary for the algorithm. In most cases however, the data structure is not always determined by the algorithm, it just requires some functionality like for example a nearest neighbour query. How that query is implemented is not always important for the algorithm, hence it is abstracted into the data structure interface.

B.1.3 *EDMOAL as an API*

EDMOAL is not intended to be used as a stand alone tool. Providing a GUI for the general purpose user requires a tremendous amount of work, which I can not provide. Also EDMOAL should provide functionality for as many projects and scientists as possible. Adding

a GUI limits its use more than it extends it. Therefore, there will be no GUI like in Weka, Knime or RapidMiner for EDMOAL. EDMOAL is not intended to be an alternative or competitor to these tools. It can however be used as an API within such tools and might be an enrichment to them at one day.

The implemented graphical representation of some algorithms is intended to be used for debugging purposes only. It might be used for screenshots in scientific papers, but it needs to be adjusted by the user if he wants more functionality than it currently has. There might be an other project in the future to provide suitable graphical output for EDMOAL algorithms, but it is not on the TODO list and I will certainly not provide one.

EDMOAL is intended to be an API and should be used as such. For a quick start, the package "dataMiningTestTrack" contains some examples how to use the algorithms on some artificially generated data.

B.2 THE BASIC STRUCTURE

B.2.1 *Indexed Data Set*

The core of all algorithms are the classes IndexedDataSet and IndexedDataObject which fix the order of the data objects. There are no algorithms yet that can deal with dynamic data sets (that is, data sets that change during the analysis process). The IndexedDataSet, once build and sealed, can not be changed any more, hence providing a $1:1$ linked relationship between index and data object, which provides an $\mathcal{O}(1)$ access to the index of a given data object. It also makes sure the results of the algorithms are interpreted properly because the index can be used to store membership values, etc. in a separate indexed list.

The IndexedDataObject is therefore a container for any class that may hold an data object without restricting the data object in any way. This provides maximal freedom to other developers that want to utilize their existing class structure and apply EDMOAL on them. Data mining algorithms and data structures within EDMOAL are build on IndexedDataSets and IndexedDataObjects.

B.2.2 *Data Structures*

All data structures are required to implement an interface that masks their functionalities. If algorithms use data structures, they should only use the functionality interfaces unless they require a specific implementation of a data structure. This way, a user may implement

additional data structures and use them within already existing algorithms.

B.2.3 *Data Mining Algorithms*

Similarly to data structures, data mining algorithms are embedded in a interface hierarchy describing its capabilities. The interface is named "DataMiningAlgorithm" and requires an algorithm to provide a function that is called to apply it. All algorithm classes implement that interface as well as any other interface that support additional functionality.

C

DATA AND RESULTS

As described in Section 4.4 and Appendix A, a lot of data is generated and stored for this work. Likewise, all clustering results and corresponding cluster quality index values are stored. In this chapter, the structure of the folders and an interpretation of the files is given.

C.1 DATA REPOSITORY

The data and the results are available at:

`https://escience.aip.de/vis/simulated-clustering-data/`

The data is available in `data/` and the clustering results as well as the cluster quality index values are stored in `results/`. A preview for both data and results is available in `previews/`. The previews contain $\frac{1}{20}$th of a data set family and results respectively. Both data set and results contain a lot of individual files which are compressed using the zip format. The data is compressed to roughly 40% of its original size, except the data of the corner data set family which is compressed to roughly 5% its original size. Please download all parts of sequence of zip files and unpack using a suitable program. The zip files are created in such a way, that all data and result files should be put in the same main folder, the necessary sub-folders are generated when unzipping. All zip files together require approximately 65 GB space on the hard disk. Unpacked, the total amount of data and result files require around 155 GB.

The naming of the data family and result files is a bit different to the notation used in the text of this work so far. Data set family *MixNormal* refers to D_1, *MixNormalNoise* is a name for D_2, *Distorted* specifies D_3 and *Corner* refers to D_4. The folder structure and content of the data files are described in the following sections.

C.2 DATA FILES

The data files for the data set families are stored in folder `data/` using the following directory structure:

```
data/
  <family>/
    002D...100D/
      000...020/
        cluster_000.csv
        ...
        cluster_249.csv
        init.csv
        meta.ini
        noise.csv
        properties.csv
        statistics.ini
```

Where `<family>` stands for the four data set families D_1 to D_4, which are addressed as: `MixNormal`, `MixNormalNoise`, `Distorted` and `Corner`. The folders `002D` to `100D` refer to data of the specified dimensionality (usually attributed as symbol m in this work) and the subdirectories `000` to `020` specify the data set index (symbol s in Section 4.4.2). The preview data set family files, only contain files for data set index `000` but the folder structure is kept in tact.

At the time I generated these files, I did not differentiate between classes and clusters. I can not change the file names because it would brake the source code analysing the data.

The files in one of the data set folders consist of 250 files for classes and some files with additional information. The files holding the classes are called `cluster_000.csv` to `cluster_249.csv`. The data sets are constructed using between 2 to 200 classes, randomly selected from the pool of 250 classes stored in the directory. $2,500$ possible initial positions for prototypes are stored in the file `init.csv` while the file `noise.csv` contains $50,000$ noise data objects.

The `properties.csv` file contains information about the shape of the classes, including the maximal radius of the class. The maximal radius is the maximal distance between a data object of the class and the center of gravity of the class. From that file, some statistics are are calculated and stored in `statistics.ini`, only the maximum of the maximal radius is used later for estimating the noise distance, see Section 4.4.2. Finally, the `meta.ini` file contains some general information about the data that is stored in the folder.

C.3 RESULT FILES

The result files have a deeper structured directory tree than the data files because there are multiple data sets generated from the classes and multiple clustering runs per data set. The files are stored in the folder called `clusterResults/` which is located at the same level as the `data/` folder if the zip files are extracted as described above. The clustering results contain only the prototype positions of the respective clustering algorithms (and in case of EMGMM the variance for each prototype) because storing the membership values on a per data

object basis would require too much space. The membership values can be reconstructed using the prototype information and the knowledge of the clustering algorithm.

The directory structure of `clusterResults/` is as follows:

```
clusterResults/
  <family>/
    002D...100D/
      000...020/
        C002...C200/
          R000...R005/
            convergence.ini
            dataSetup.ini
            EMGMM_clusterVariance.csv
            <algorithm>_protoPos.csv
            score.ini
```

Where `<family>` stands again for the four data set families D_1 to D_4, addressed as: `MixNormal`, `MixNormalNoise`, `Distorted` and `Corner`. `<algorithm>` stands for EMGMM, FCM2, FCMdim, HCM, NFCM2, NFCMdim, PFCM, PNFCM, RCFCM and RCNFCM which refer to the clustering algorithms that were used to find the prototype positions. The folders `002D` to `100D` refer to data of the specified dimensionality (usually attributed as symbol m in this work) and the subdirectories `000` to `020` specify the data set index (symbol s in Section 4.4.2). Further subdirectories `C002` to `C200` refer to the number of classes/clusters (symbol c) that were used to generate the clustering results and `R000` to `R005` refer to the clustering run with initial prototype positions 1 to 5, addressed with symbol r in Section 4.4.2. The preview clustering results again only contain files for data set index `000` but the folder structure is kept in tact.

The file `dataSetup.ini` contains the information which classes, noise data objects and prototype initial positions were used for this clustering run. The `convergence.ini` file contains for each clustering algorithm the maximal distance that a prototype travelled between two iteration steps. It can be used to verify that the clustering process indeed converged with only tiny steps left at the end of the clustering process. The final positions of the prototypes are stored in the files `<algorithm>_protoPos.csv`, for each clustering algorithm individually. Only for EMGMM, additional information about the variance of individual classes were necessary to have complete information, which is stored in `EMGMM_clusterVariance.csv`.

Finally, the file `score.ini` contains the information about the clustering result. In this file, for each clustering algorithm the following information is stored:

ITERATIONS: The number of iterations the clustering algorithm was running before it was terminated.

OBJFUNC VALUE: The objective function value after the clustering process finished.

F1: The F_1 index value of the clustering result, indicating the true clustering quality (see Section 4.2)

F1 DEFUZZ: The F_1 index after defuzzifying the membership values with the winner takes all strategy: a data object is assigned crisply to that cluster, that has the highest membership value.

BSI, DBI, XBI, NPCI, PEI: The values for the internal cluster quality indices, see Section 4.3.

The subdirectory `perfectProto/` that sits on the same levels as `R000` to `R005` contains artificial clustering results. These results were created using the centre of mass of a class as the initial location for the prototypes. No iterations of the clustering algorithms were done so that the initial prototype positions were also the final prototype positions and the score values were calculated using this clustering result as input. This can be seen as an approximation of the best result a clustering algorithm could produce on the given data set. This data is not evaluated within this work and is also not contained in the scoreList, discussed in the next section.

C.4 SCORE LIST

The file `scores.zip` contains all the score files, described in the last Section while retaining the `clusterResult` directory structure. Since this is a lot of small files that is somewhat cumbersome to use, all that data is also available in one large list, containing also the relevant meta information, see Table C.1. The score list is contained in the zip file `scoreList.zip` and has a size of about 125 MB. The table contains $1,000,000$ rows ($800,000$ of which are used within this work) and 15 columns, see Table C.1. Each row contains the meta information and scores for one individual experiment, but not the position of the prototypes. The contents of this list are used to generate the plots, seen in Appendices D, E and F as well as in many images shown in Chapters 5.

Please be careful when opening the file with a text editor or table calculation program. Due to its size not all programs are able to open it properly.

Column name	Description	Value Range
DataSet	Data set family	$0, \ldots, 3$
Experiment Repetition	Index of the repetition of an experiment	$0, \ldots, 19$
Dim	Number of dimensions of the data set	$2, \ldots, 100$
Cluster Count	Number of classes in the data set	$2, \ldots, 200$
Algorithm	Algorithm index	$0, \ldots, 9$
Initialization	Initialization index	$0, \ldots, 4$
Iterations	Number of iterations before terminating the algorithm	$10, \ldots, 30$
ObjFunc Value	Objective function value after clustering	$\geqslant 0$
F1	F_1 external index value	$[0, 1]$
F1 defuzzified	F_1 external index value if fuzzy membership values are defuzzified with winner takes all strategy	$[0, 1]$
BSI	Bezdec Separation Index value	$\geqslant 0$
DBI	Davies-Bouldin Index value	$\geqslant 0$
XBI	Xie-Beni Index value	$\geqslant 0$
NPCI	Normalised Partition Coefficient Index value	$[0, 1]$
PEI	Normalised Partition Entropy Index value	$[0, 1]$

Table C.1: Columns of the scoreList data file.

D

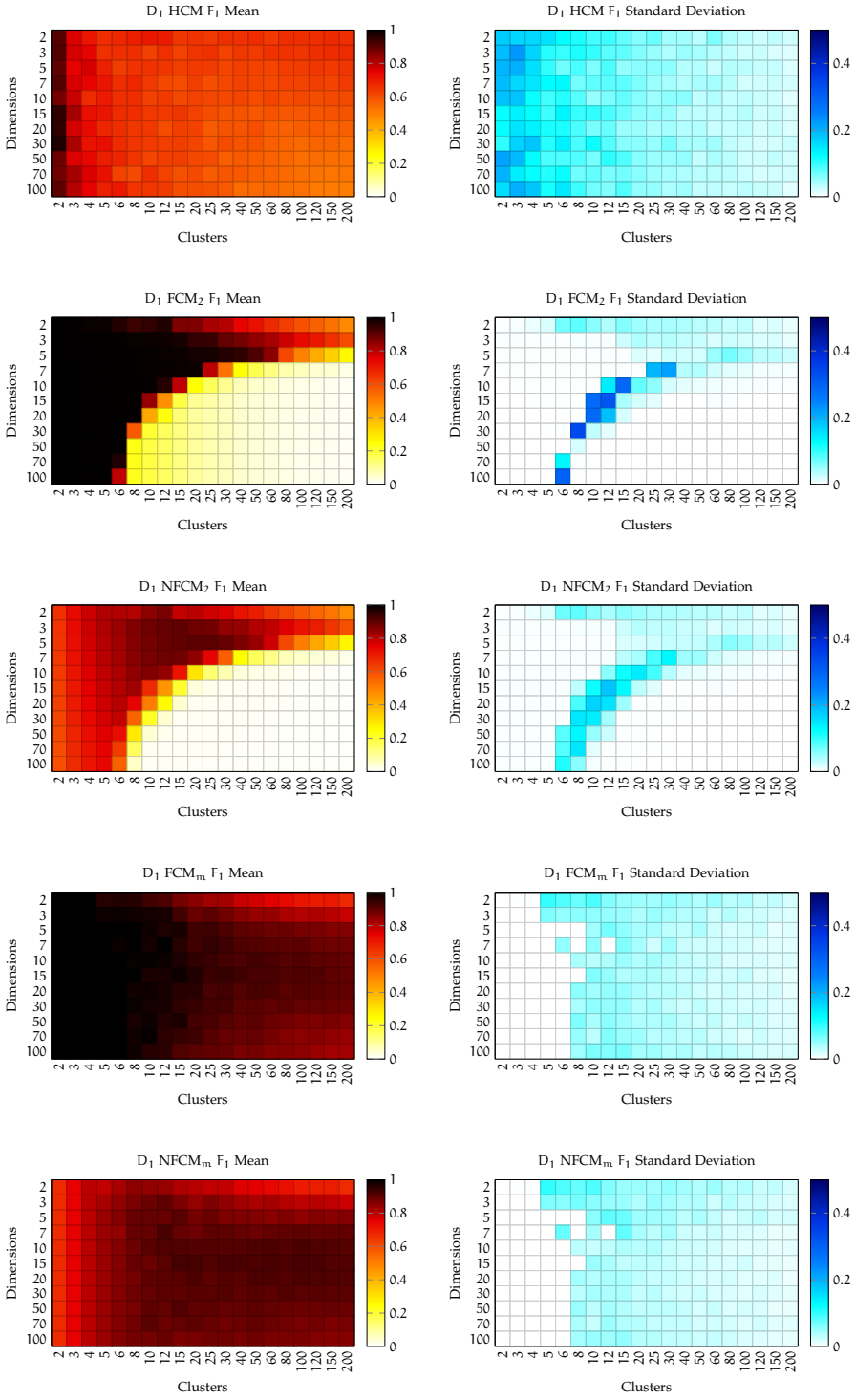
QUALITY OF CLUSTERING ALGORITHMS

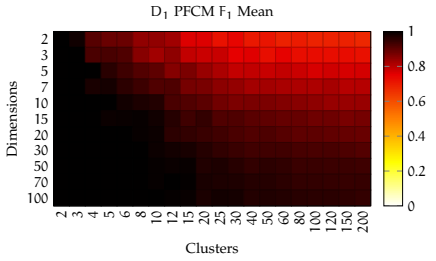
In this appendix, the quality of the 10 clustering algorithms (see Chapter 3) are visualized for the 4 different data set families (see Section 4.1). The plots show the mean F_1 values that can be interpreted as clustering performance as well as the standard deviation to indicate the spread of the mean. For each cell in a diagram 20 different data sets of a data set family with corresponding number of dimensions and classes are considered. Each of the 20 data sets is clustered 5 times with different initializations. The best clustering results of these 5 clustering runs is used. The mean of the 20 best F_1 values are shown colour coded in the left-hand side panels. The corresponding standard deviation is shown on the right-hand side panels with two different colour sets to make the difference visually easy to distinguish. The procedure is also described at the beginning of Section 5.1 in the main text.

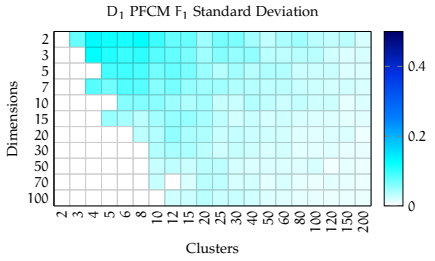
There are 4 groups of panels, one group for each data set family and two panels for each of the 10 algorithms in a group. On the mean panels on the left-hand side of the pages, the darker the colour, the higher is the mean cluster quality. On the standard deviation panels on the right-hand side of the pages, a dark colour represents a high standard deviation which means that the mean value in the corresponding plot on the left has a large spread. The first 4 rows of the panels for data sets of the D_4 data set family are left empty because no data was generated for these tiles (dimensions 2, 3, 5 and 7). To keep the visual representation consistent, these rows are left in the diagram but do not hold any information.

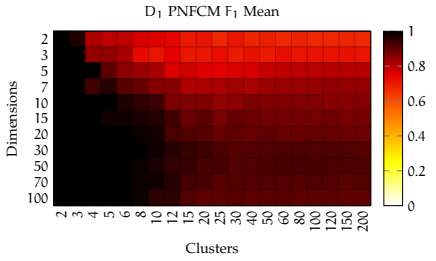
The raw data is stored in a table and is available online, see Appendix C.4.

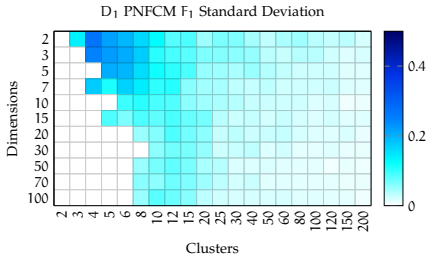
D.1 DATA SET FAMILY D_1

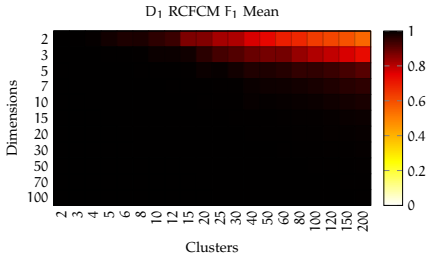


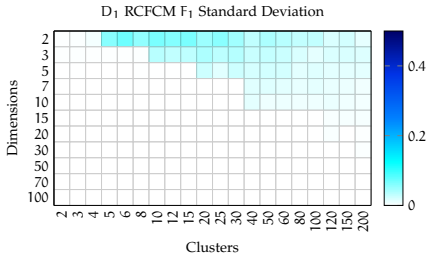

D_1 PFCM F_1 Mean
Dimensions
Clusters

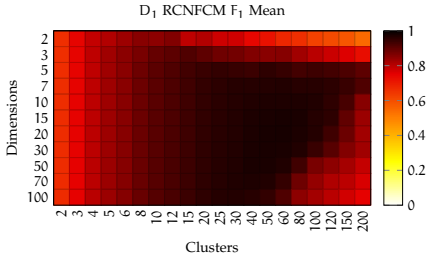

D_1 PFCM F_1 Standard Deviation
Dimensions
Clusters

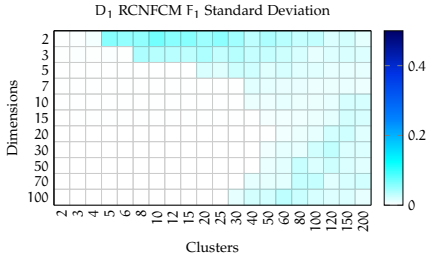

D_1 PNFCM F_1 Mean
Dimensions
Clusters

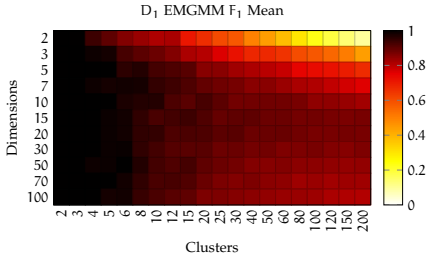

D_1 PNFCM F_1 Standard Deviation
Dimensions
Clusters

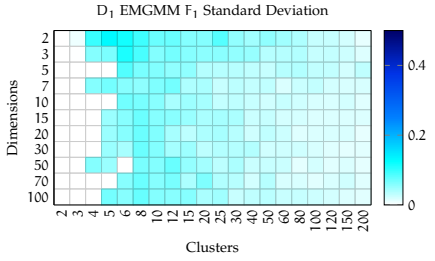

D_1 RCFCM F_1 Mean
Dimensions
Clusters


D_1 RCFCM F_1 Standard Deviation
Dimensions
Clusters

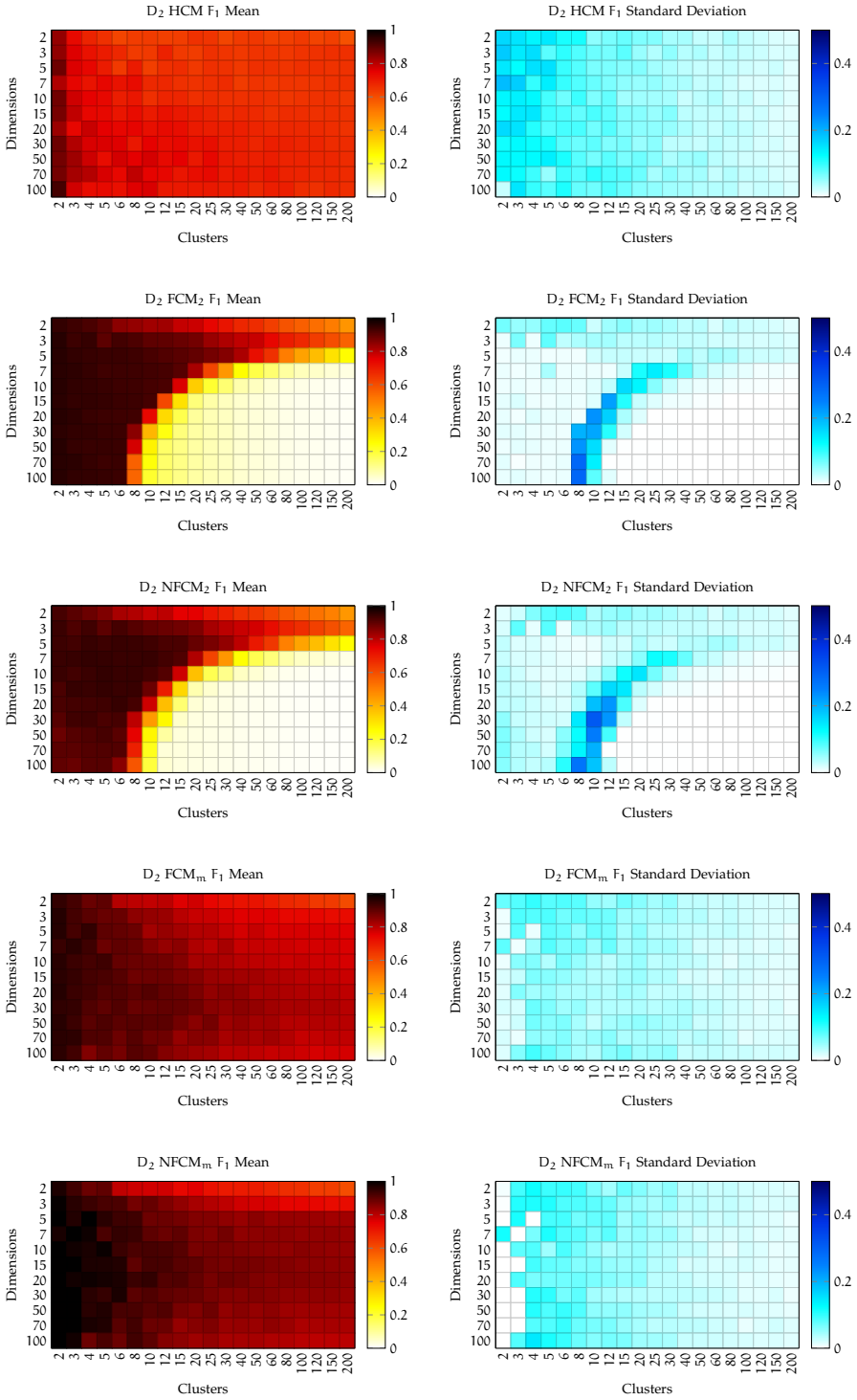

D_1 RCNFCM F_1 Mean
Dimensions
Clusters

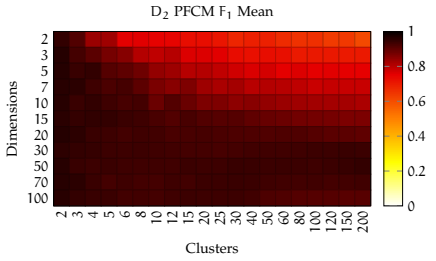

D_1 RCNFCM F_1 Standard Deviation
Dimensions
Clusters

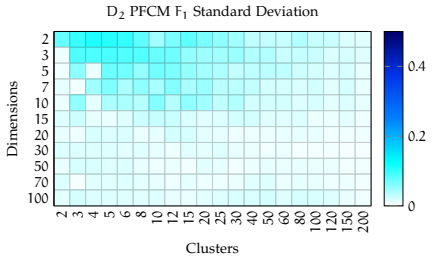

D_1 EMGMM F_1 Mean
Dimensions
Clusters

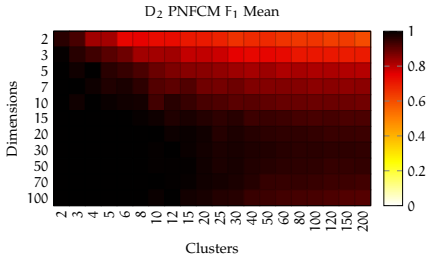

D_1 EMGMM F_1 Standard Deviation
Dimensions
Clusters

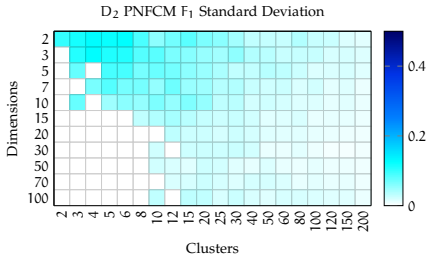
D.2 DATA SET FAMILY D_2

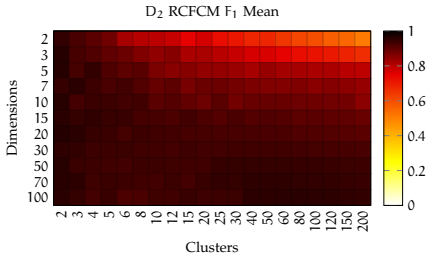


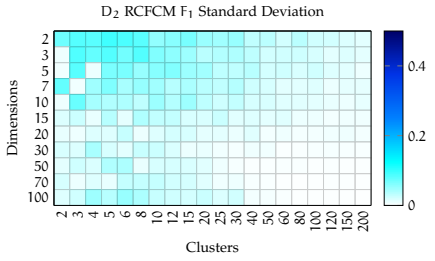

D_2 PFCM F_1 Mean
Dimensions
Clusters

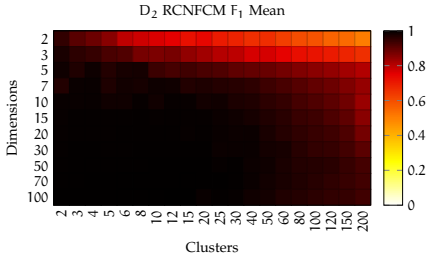

D_2 PFCM F_1 Standard Deviation
Dimensions
Clusters

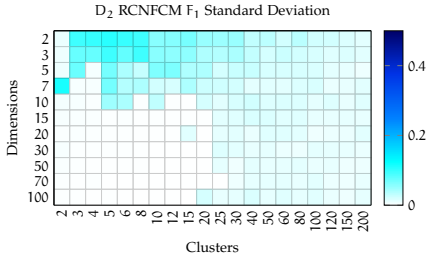

D_2 PNFCM F_1 Mean
Dimensions
Clusters

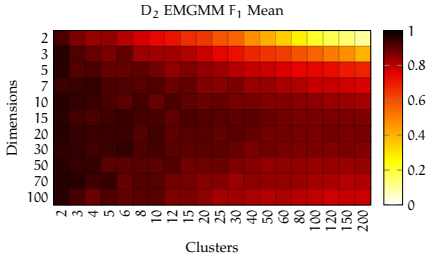

D_2 PNFCM F_1 Standard Deviation
Dimensions
Clusters

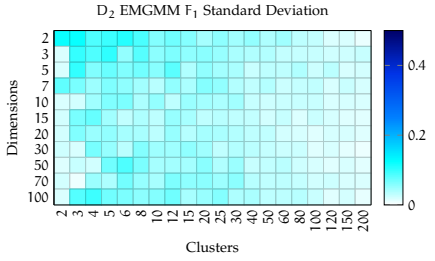

D_2 RCFCM F_1 Mean
Dimensions
Clusters


D_2 RCFCM F_1 Standard Deviation
Dimensions
Clusters

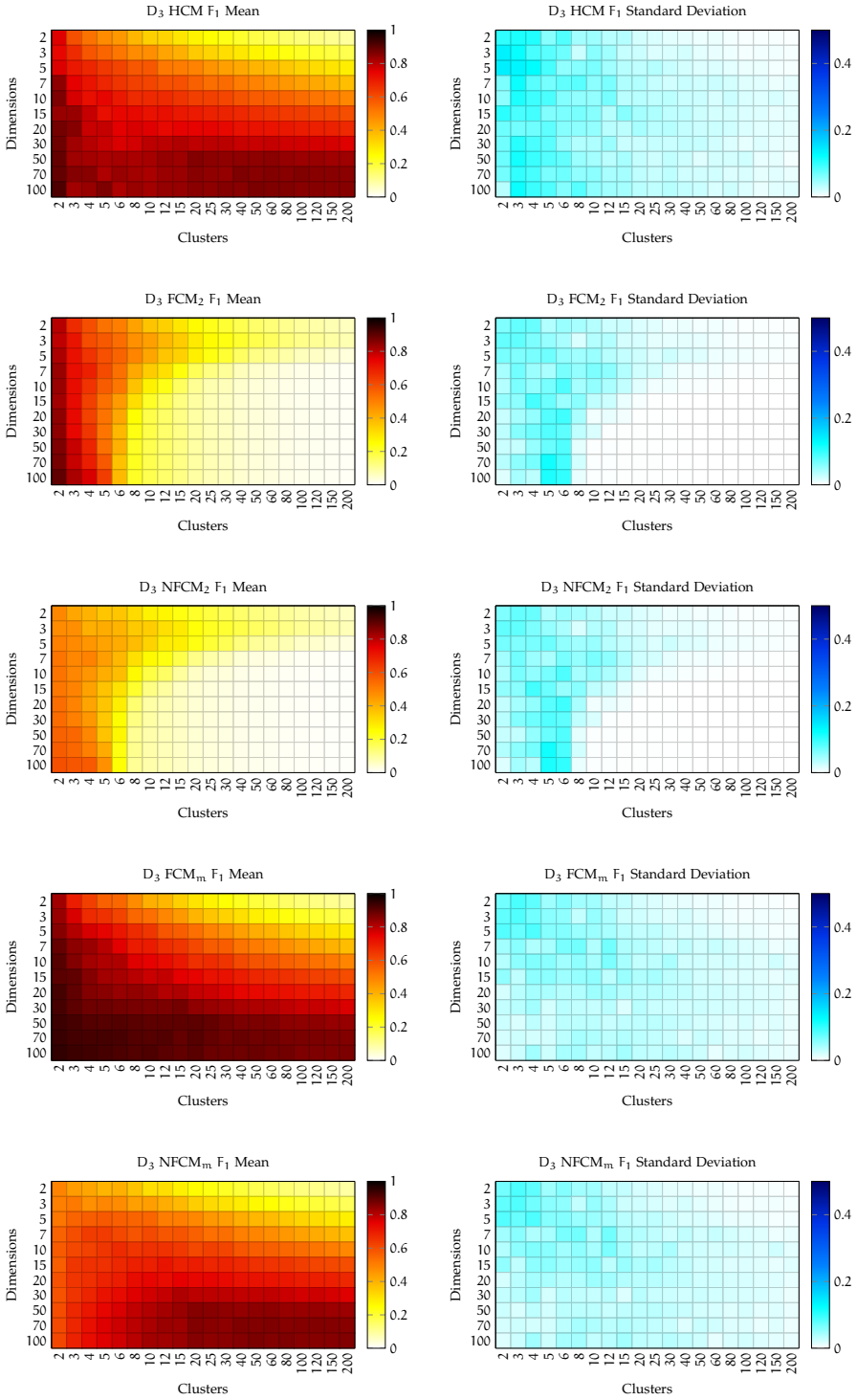

D_2 RCNFCM F_1 Mean
Dimensions
Clusters

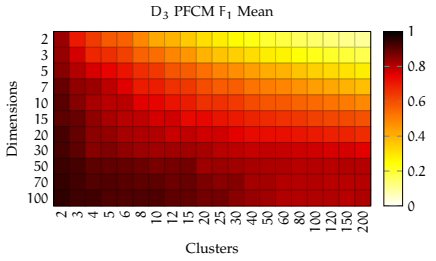

D_2 RCNFCM F_1 Standard Deviation
Dimensions
Clusters

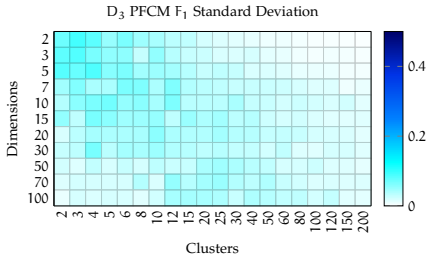

D_2 EMGMM F_1 Mean
Dimensions
Clusters

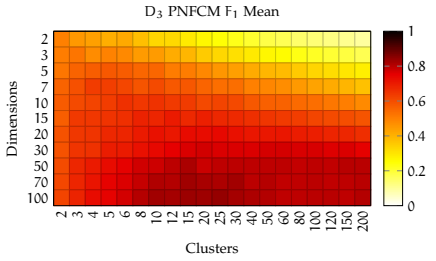

D_2 EMGMM F_1 Standard Deviation
Dimensions
Clusters

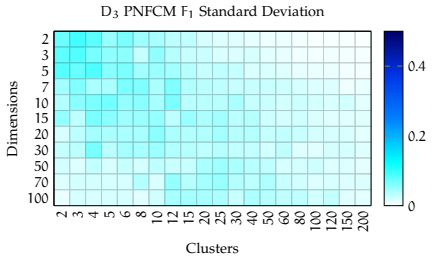
D.3 DATA SET FAMILY D_3

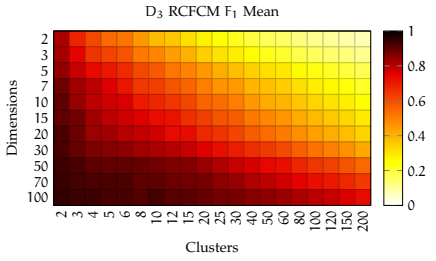


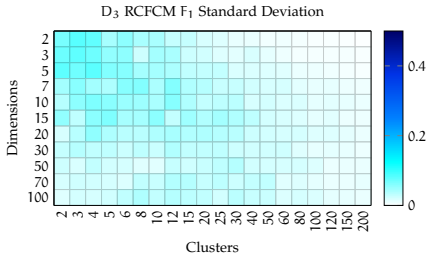

D_3 PFCM F_1 Mean
Dimensions
Clusters

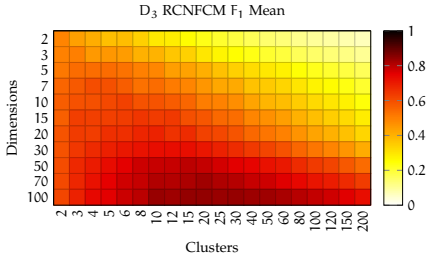

D_3 PFCM F_1 Standard Deviation
Dimensions
Clusters

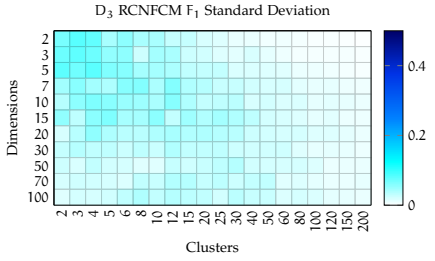

D_3 PNFCM F_1 Mean
Dimensions
Clusters

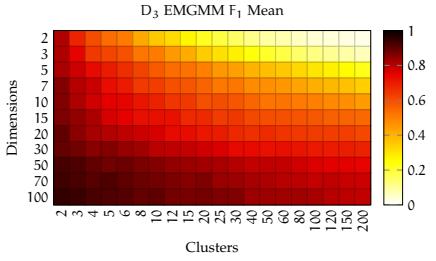

D_3 PNFCM F_1 Standard Deviation
Dimensions
Clusters

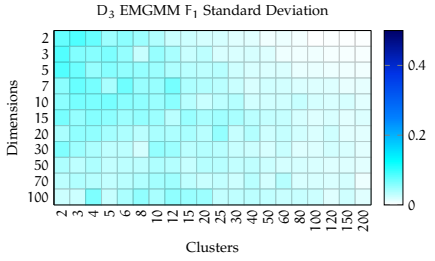

D_3 RCFCM F_1 Mean
Dimensions
Clusters


D_3 RCFCM F_1 Standard Deviation
Dimensions
Clusters

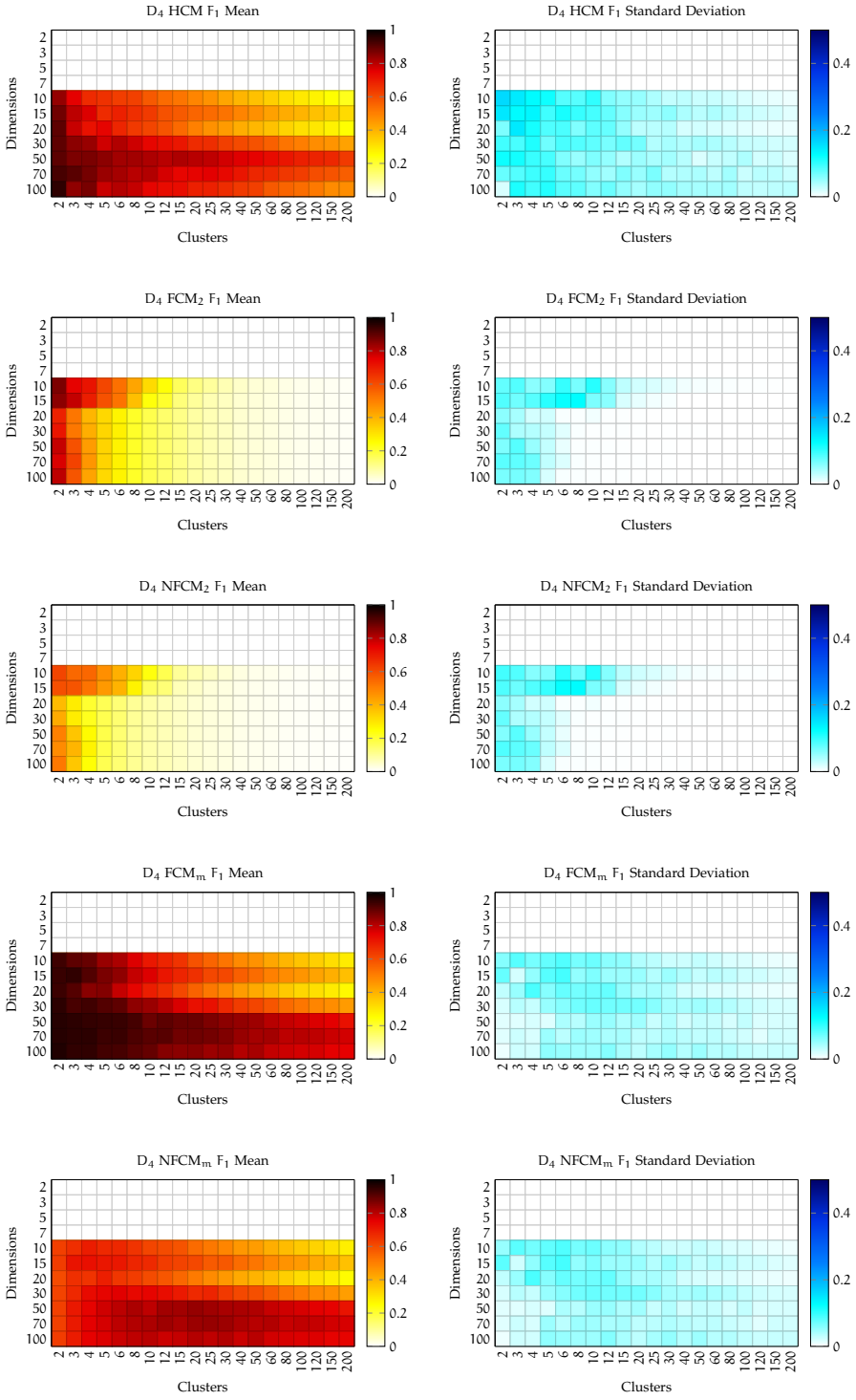

D_3 RCNFCM F_1 Mean
Dimensions
Clusters

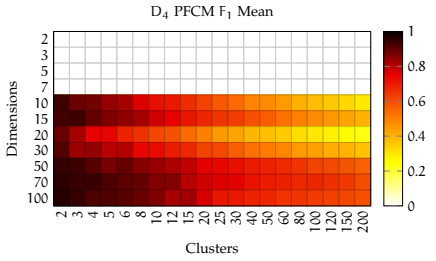

D_3 RCNFCM F_1 Standard Deviation
Dimensions
Clusters

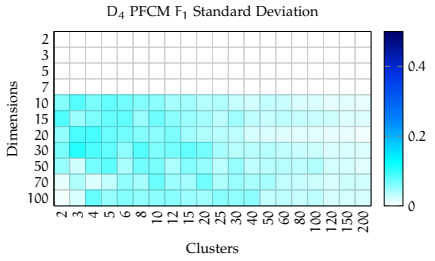

D_3 EMGMM F_1 Mean
Dimensions
Clusters

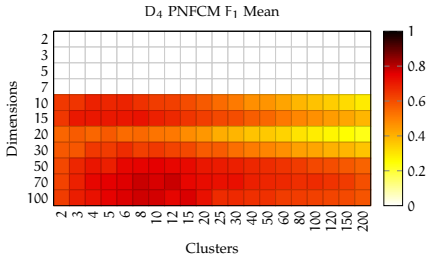

D_3 EMGMM F_1 Standard Deviation
Dimensions
Clusters

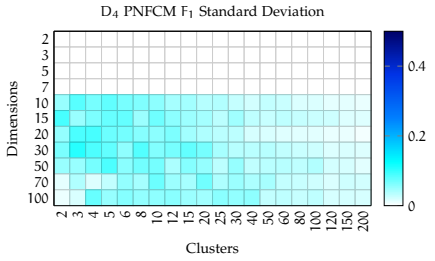
D.4 DATA SET FAMILY D_4

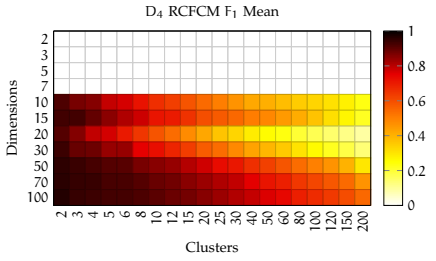


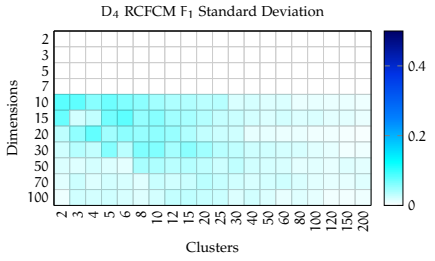

D_4 PFCM F_1 Mean
Dimensions
Clusters

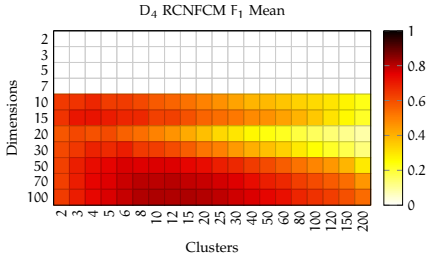

D_4 PFCM F_1 Standard Deviation
Dimensions
Clusters

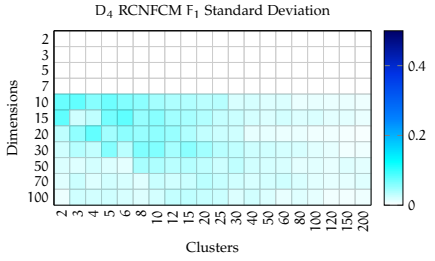

D_4 PNFCM F_1 Mean
Dimensions
Clusters

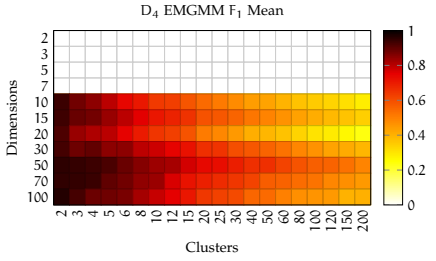

D_4 PNFCM F_1 Standard Deviation
Dimensions
Clusters

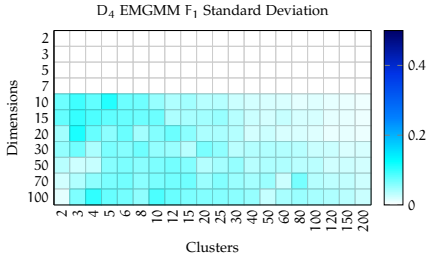

D_4 RCFCM F_1 Mean
Dimensions
Clusters


D_4 RCFCM F_1 Standard Deviation
Dimensions
Clusters


D_4 RCNFCM F_1 Mean
Dimensions
Clusters


D_4 RCNFCM F_1 Standard Deviation
Dimensions
Clusters


D_4 EMGMM F_1 Mean
Dimensions
Clusters


D_4 EMGMM F_1 Standard Deviation
Dimensions
Clusters

E

RANKING CAPABILITY OF INTERNAL INDICES

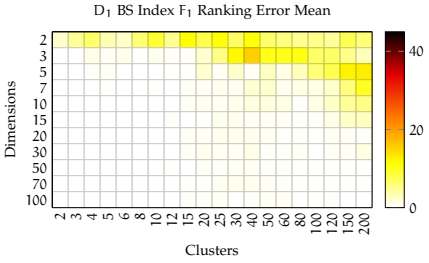
In this Appendix, the ability to correctly rank the results of the clustering algorithms is visualized for 5 different internal cluster quality indices (see Section 4.3). The plots show the mean ranking error on the left-hand side panels and the corresponding standard deviation on the right-hand side panels.

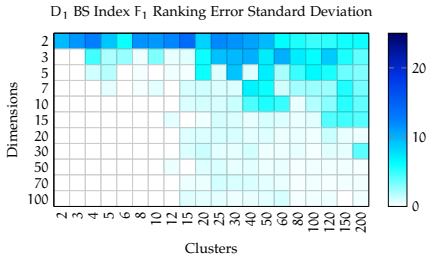
Each cell in a diagram, represents the performance of an index for clustering results of one data set family and a particular combination of dimensios and classes/clusters. There are 20 data sets per cell and each data set is clustered 5 times with different initializations by 9 clustering algorithms (HCM is excluded because it does only generate crisp clustering results). The ranking error of an index is measured as the ranking position of the best clustering result according to the F_1 measure (see Section 4.2). From these 20 ranking error values, the mean and standard deviation is calculated, please see Section 5.3.1 for more information on the procedure. Note that contrary to the panels in the last appendix, a dark colour in the left-hand side panels indicate bad ranking performance.

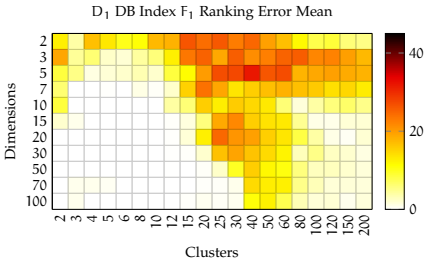
There are 4 groups of panels, one group for each data set family and two panels for each of the 5 cluster quality indices (see Section 4.3) in a group. On the mean panels on the left-hand side of the pages, the darker the colour, the higher is the mean ranking error. On the standard deviation panels on the right-hand side of the pages, a dark colour represents a high standard deviation which means that the mean value in the corresponding plot on the left has a large spread. The first 4 rows of the panels for data sets of the D_4 data set family are left empty because no data was generated for these tiles (dimensions 2, 3, 5 and 7). To keep the visual representation consistent, these rows are left in the diagram but do not hold any information.

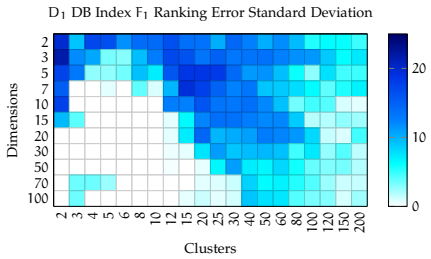
The raw data is stored in a table and is available online, see Appendix C.4.

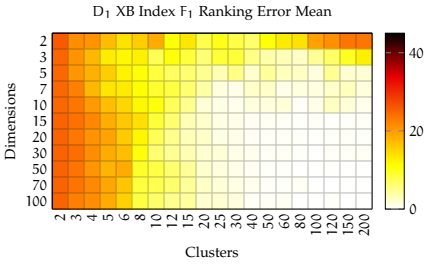
E.1 DATA SET FAMILY D_1

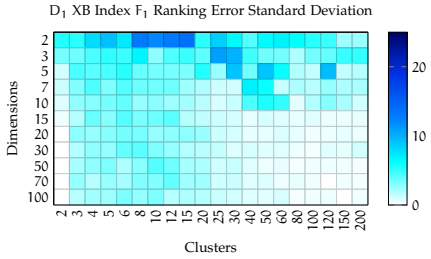


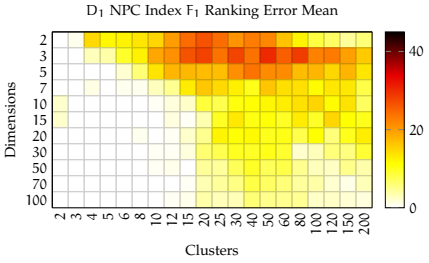


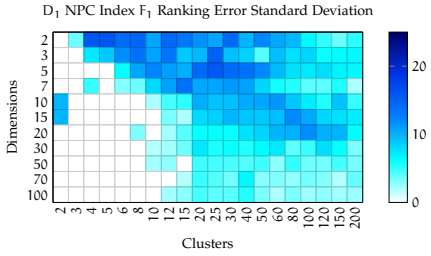


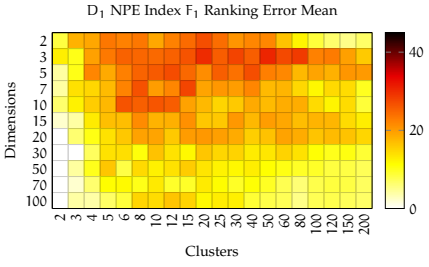


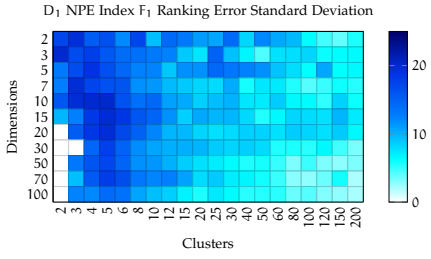




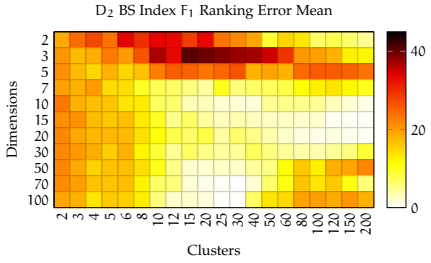


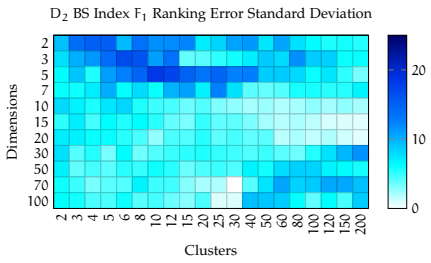


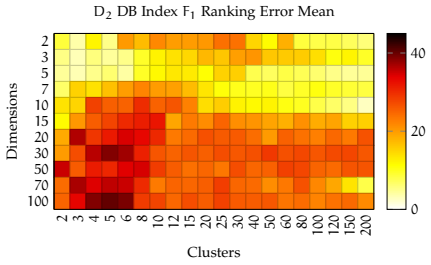


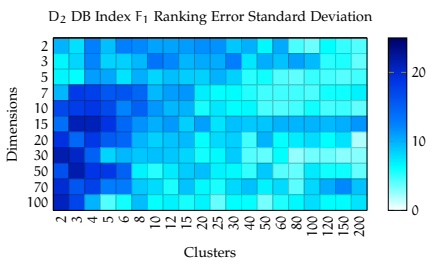


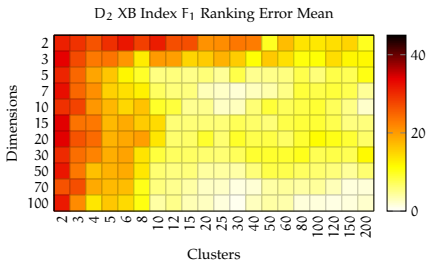
E.2 DATA SET FAMILY D_2

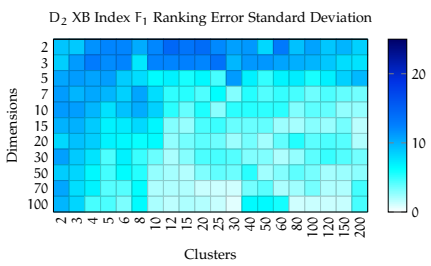


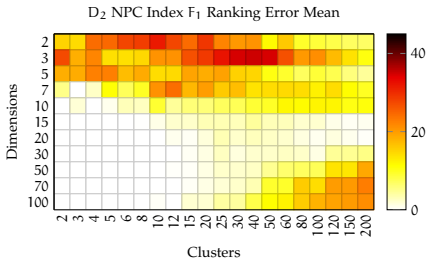


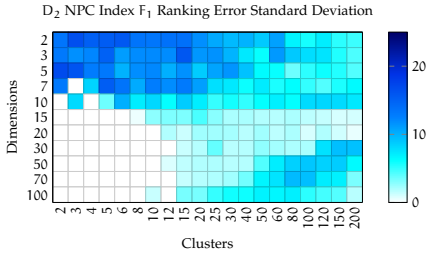


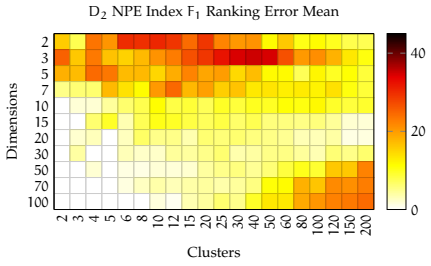


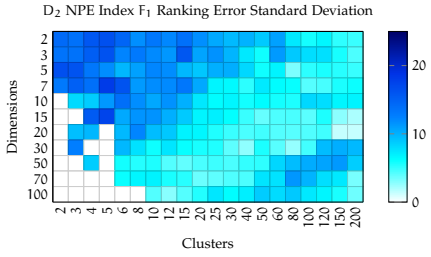




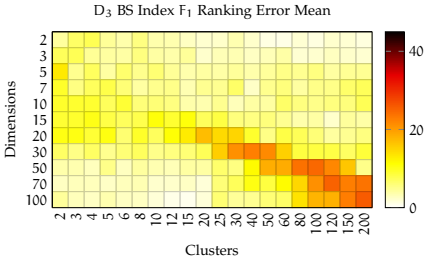


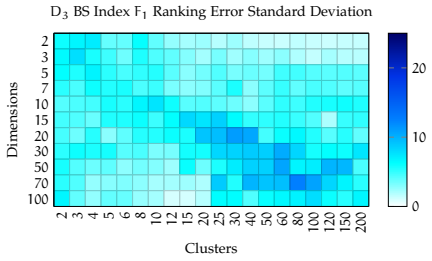


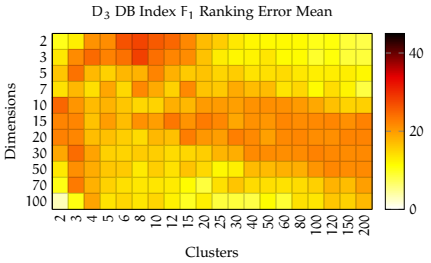


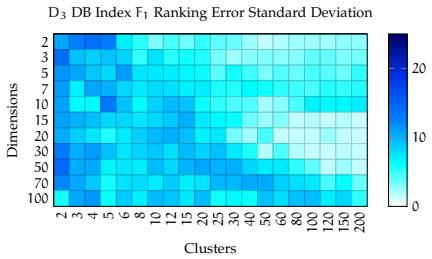


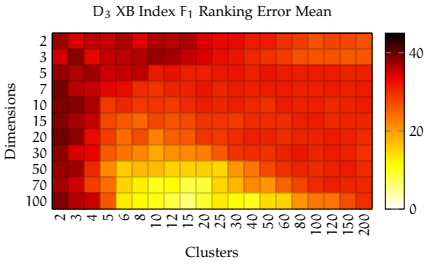
E.3 DATA SET FAMILY D_3

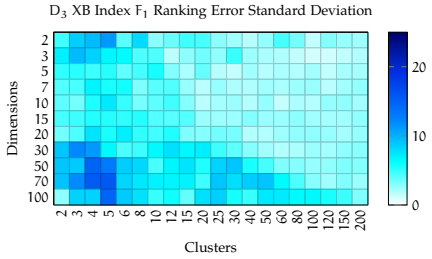


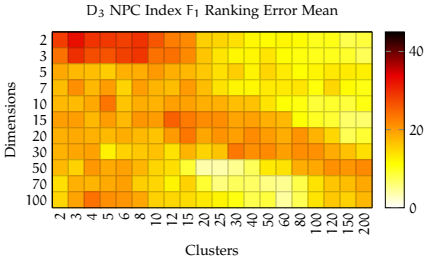


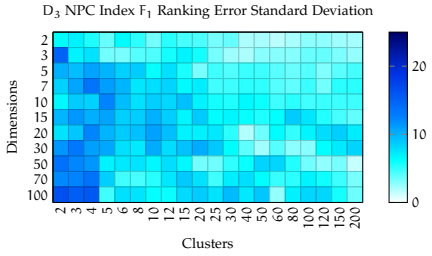


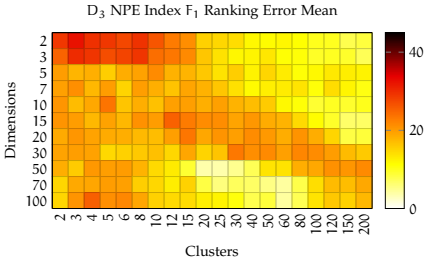


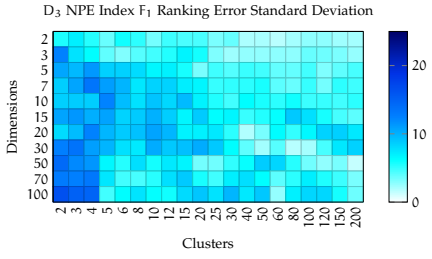




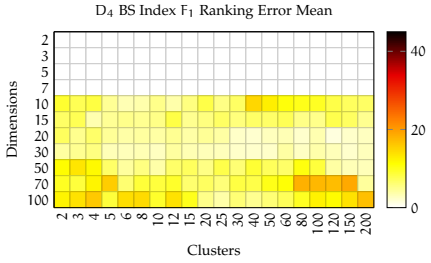


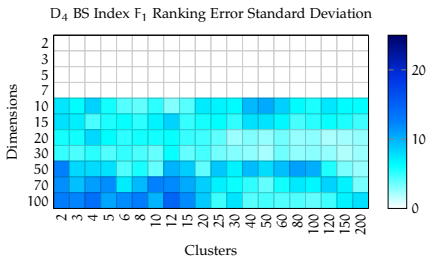


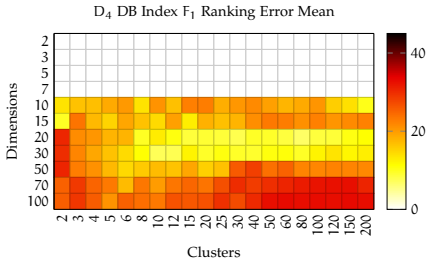


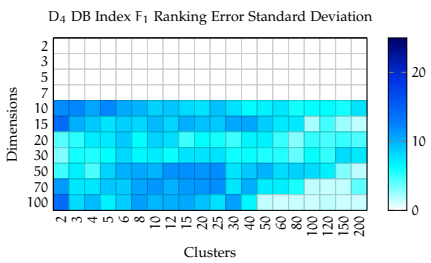


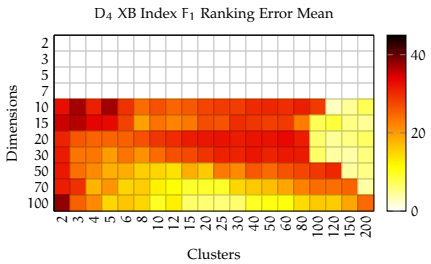
E.4 DATA SET FAMILY D_4

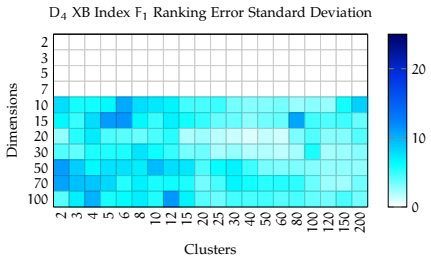


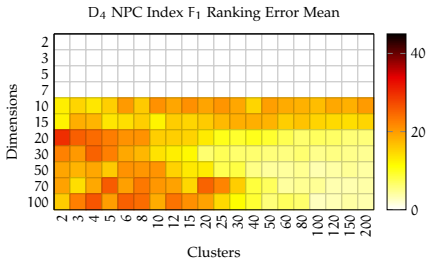


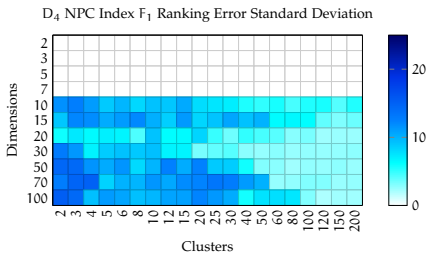


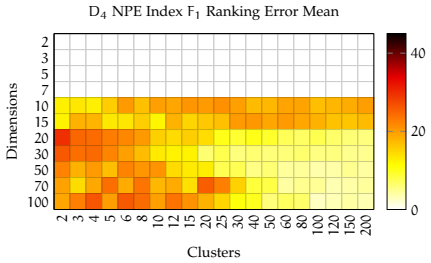


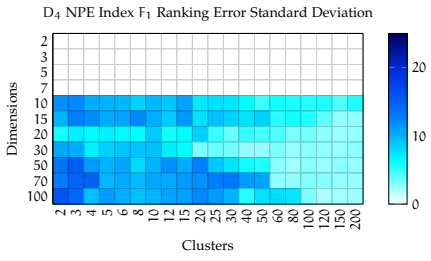












F

INTERNAL AND THE F_1 INDEX CORRELATIONS

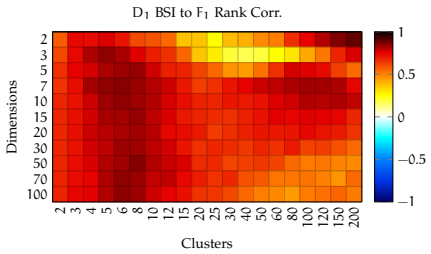
In this Appendix, the performance of the cluster quality indices (see Section 4.3) are visualized further by presenting the Spearman rank correlation coefficients of the index values with the F_1 measure (see Section 4.2).

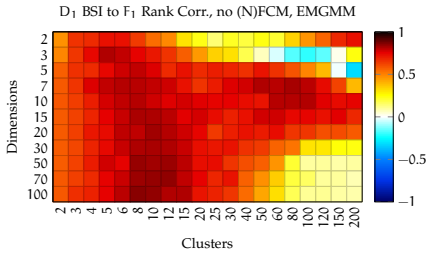
Each cell in a diagram, represents the correlation performance of an index for clustering results of one data set family and a particular combination of dimensios and classes/clusters. There are 20 data sets per cell and each data set is clustered 5 times with different initializations by 9 clustering algorithms on the left-hand side panels (HCM is excluded because it does only generate crisp clustering results) and 6 different clustering algorithms on the right-hand side panels (HCM, FCM, NFCM and EMGMM are left out). The spearman rank correlation is computed for all clustering results corresponding to one cell combined, that are $20 \cdot 5 \cdot 9 = 900$ on the left-hand side and $20 \cdot 5 \cdot 6 = 600$ on the right hand side. See Section 5.3.2 for more details on the procedure

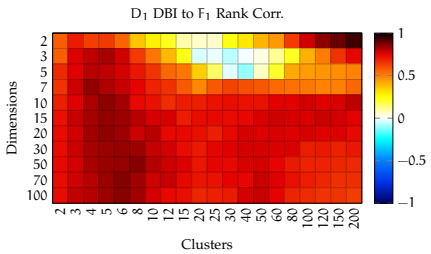
Since the correlation coefficient between an index and the F_1 measure can be both, positive and negative, the colour scale is very different to the previous panels. A positive correlation is good, represented in orange colours and no correlation is bad, which is represented in white. Negative correlation basically means that the internal index predicts the opposite of what it should do this is worse than bad and is coloured in blue. There are 4 groups of panels, one group for each data set family and two panels for each of the 5 cluster quality indices (see Section 4.3) in a group. The first 4 rows of the panels for data sets of the D_4 data set family are left empty because no data was generated for these tiles (dimensions 2, 3, 5 and 7). To keep the visual representation consistent, these rows are left in the diagram but do not hold any information.

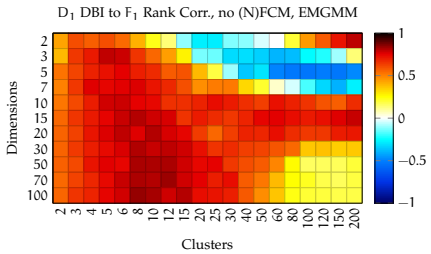
The raw data is stored in a table and is available online, see Appendix C.4.

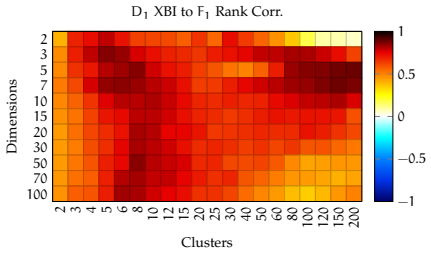
F.1 DATA SET FAMILY D_1

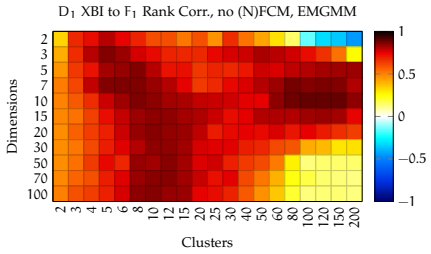


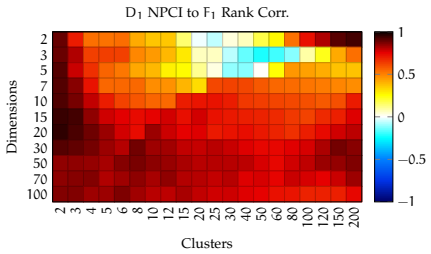


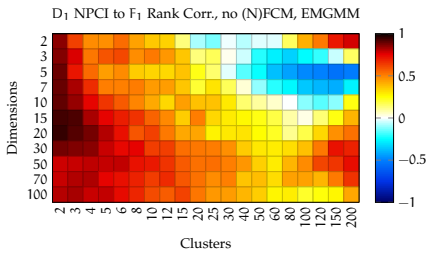


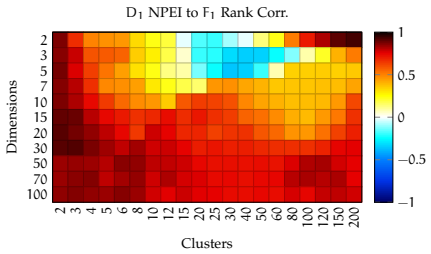


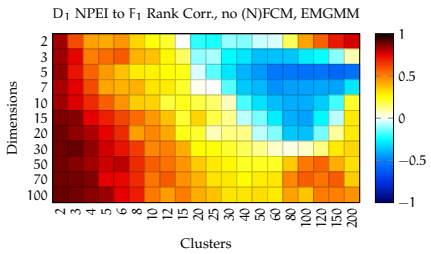




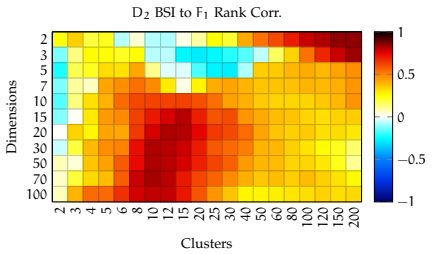


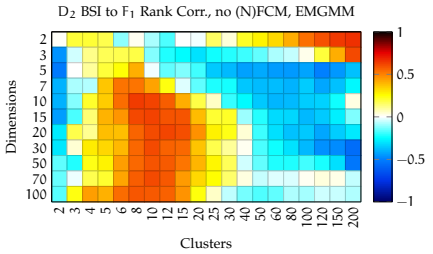


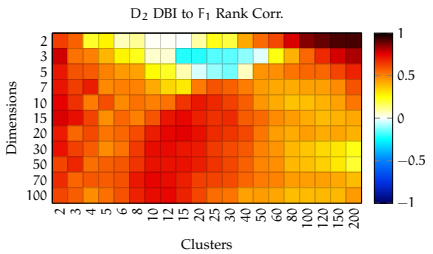


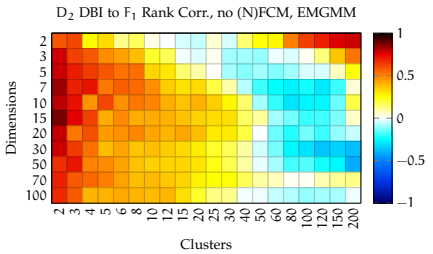


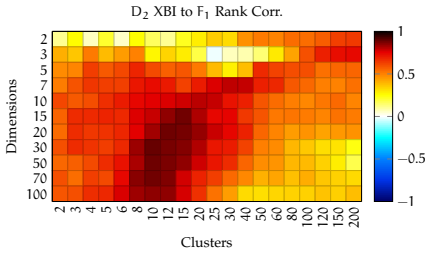
F.2 DATA SET FAMILY D_2

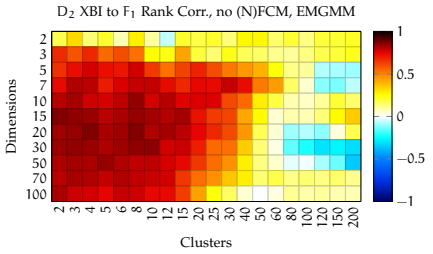


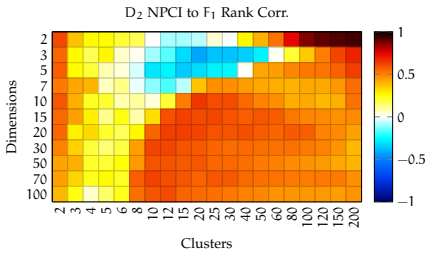


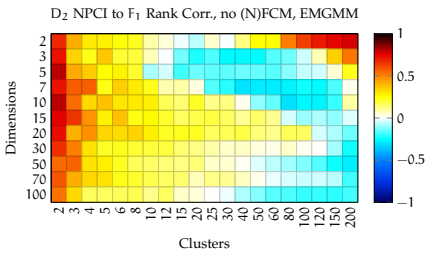


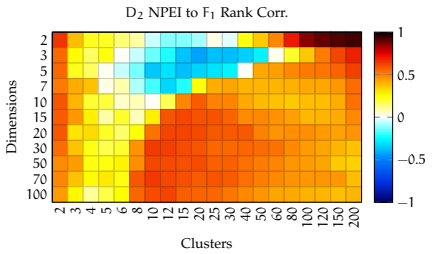


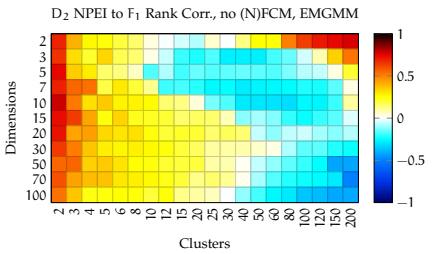




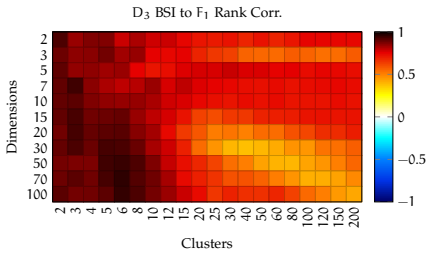


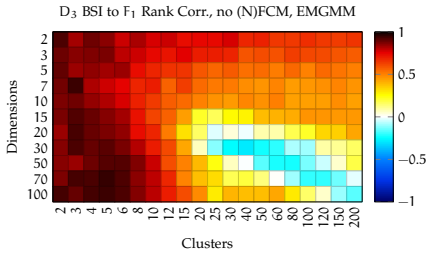


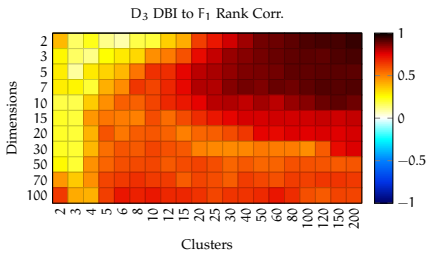


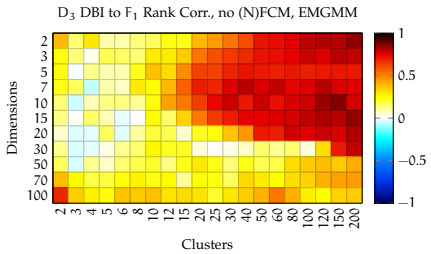


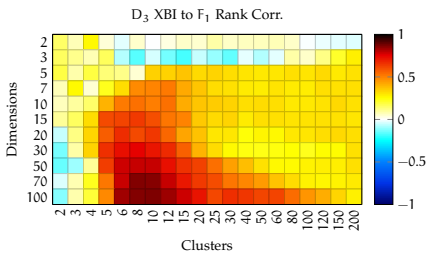
F.3 DATA SET FAMILY D_3

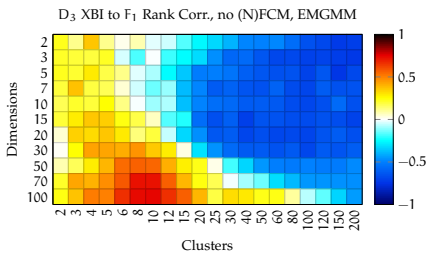


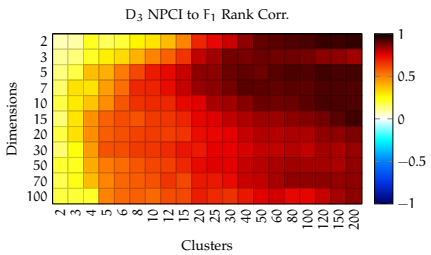


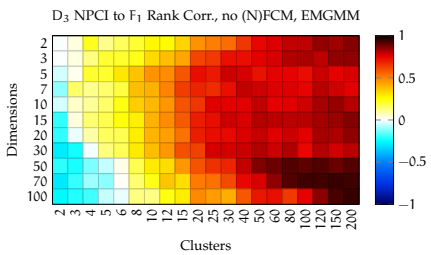


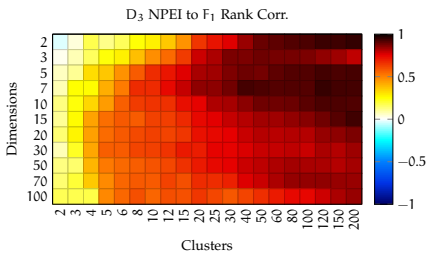


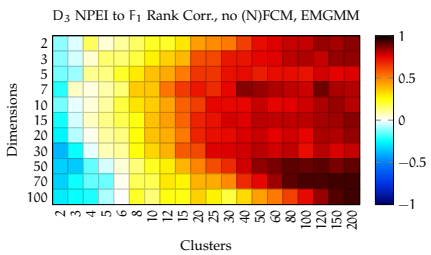




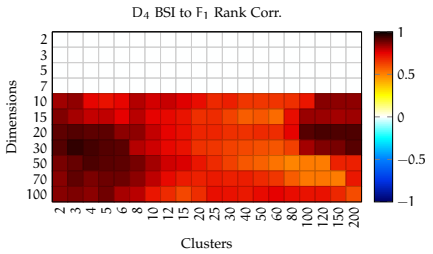


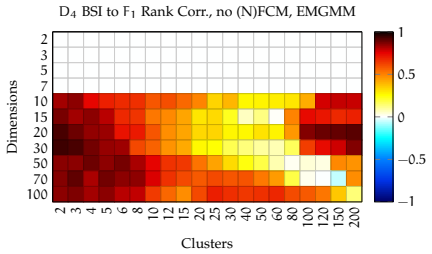


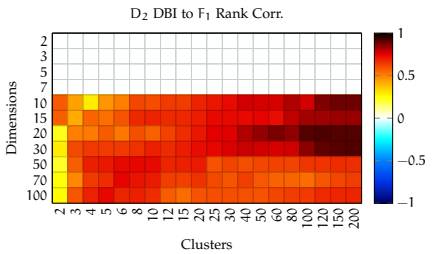


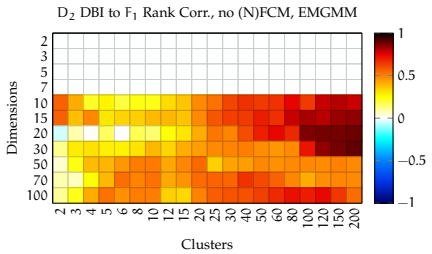


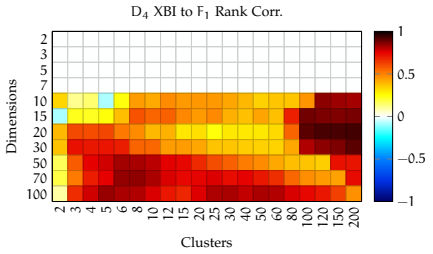
F.4 DATA SET FAMILY D_4

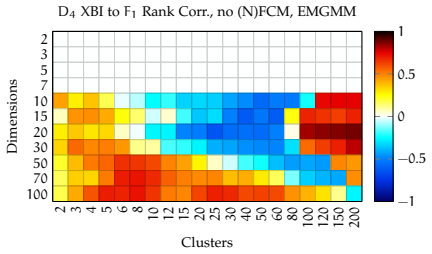


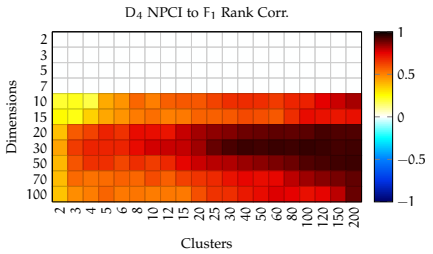


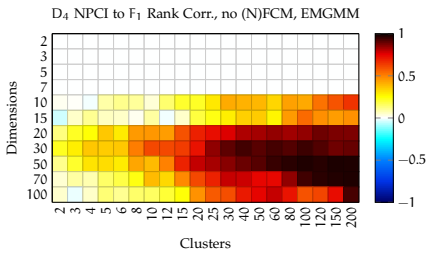


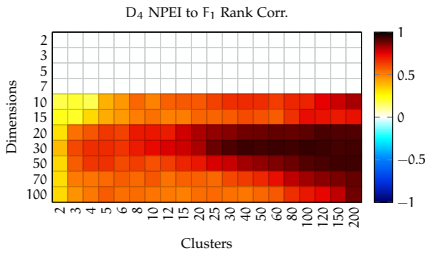


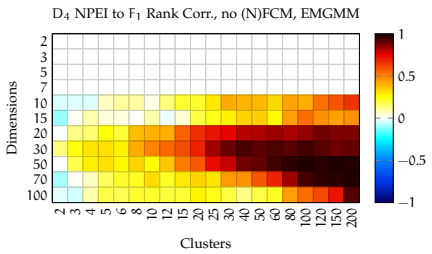












G

INTERNAL VS. THE F_1 INDEX PLOTS

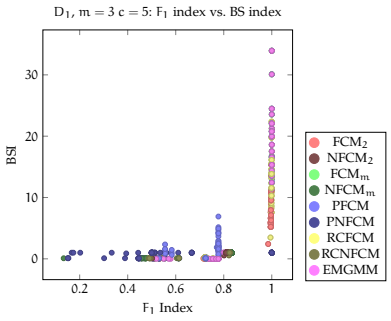
The plots in this appendix visualize the data behind the correlation plots, presented in Appendix F. The plots are used to find an explanation for the performance of the internal indices, presented in the last appendix. Each individual plot shows the F_1 index on the x-achsis and the value of an index on the y-achsis, see the end of Section 5.3.2 for more information.

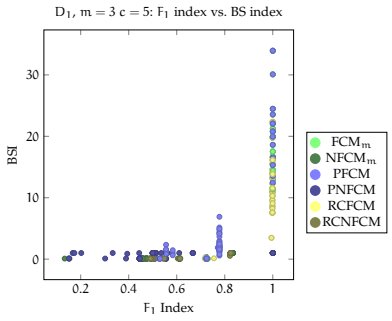
Similar to the rank correlation panels in the last appendix, the left-hand side plots in this appendix show the index values for all clustering algorithms but HCM and the right-hand side plots show also leave HCM, FCM, NFCM and EMGMM away. Each plot in this appendix visualized the data for calculating the rank correlation in one cell of one plot in the last appendix. The cells (addressed as tuple (m, c)) are for all 4 data set families and all 5 indices: $(3, 5)$, $(10, 100)$, $(15, 20)$ and $(50, 150)$. The plots on the left-hand side contains 900 clustering results and 600 on the right-hand side. Each dot in a plot corresponds to one clustering result and the clustering algorithms that were used to generate the clustering results are colour coded, as presented in the legend. The plots are grouped in 4 times 4 groups, sorted first according to the cell and second according to the data set family.

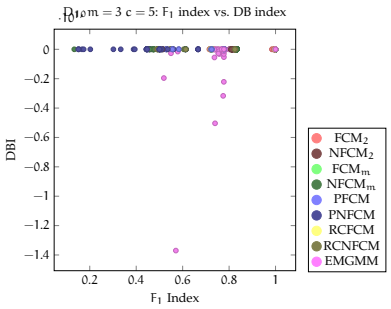
The raw data is stored in a table and is available online, see Appendix C.4.

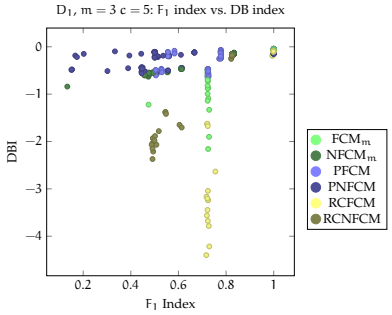
G.1 PLOTS FOR $m = 3$ AND $c = 5$

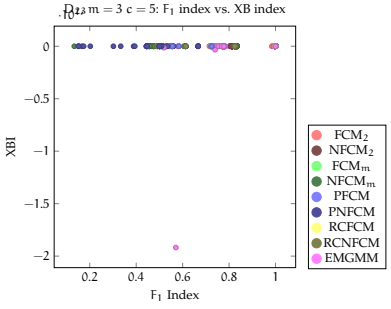
G.1.1 *Data Set Family* D_1

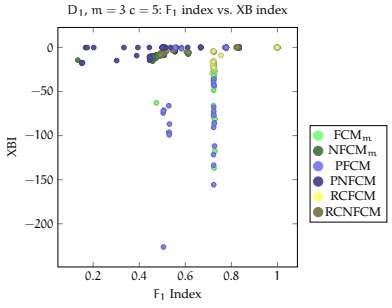


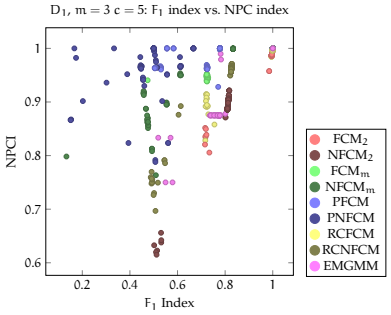


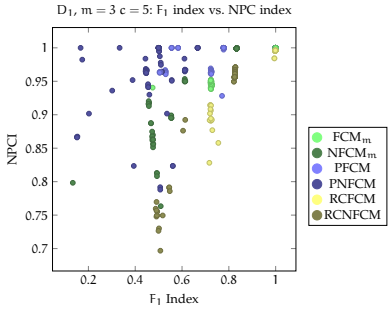


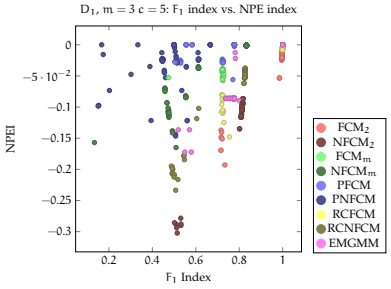


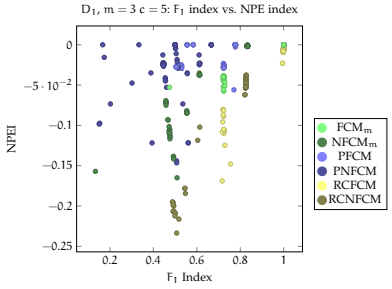




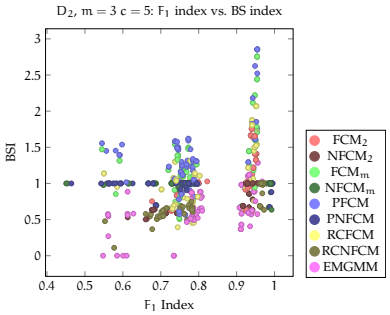


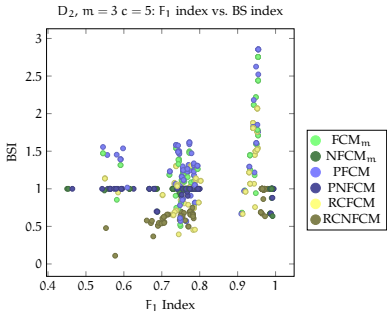


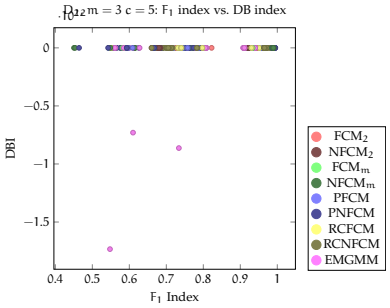


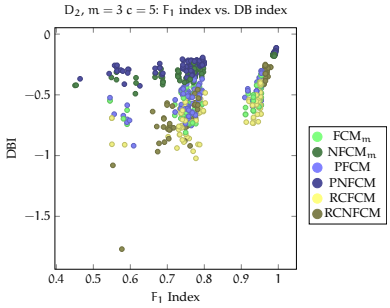


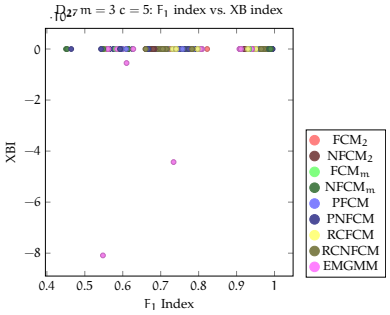
G.1.2 *Data Set Family* D_2

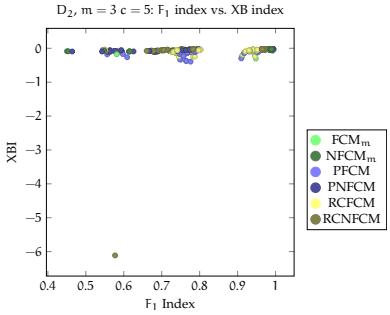


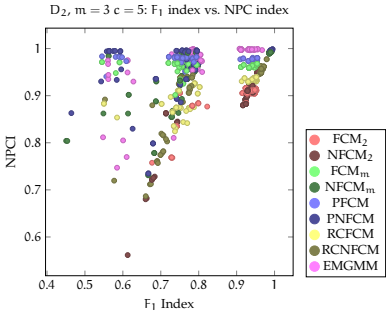


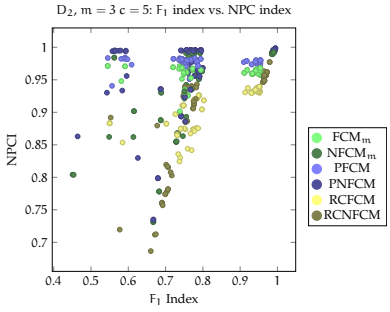


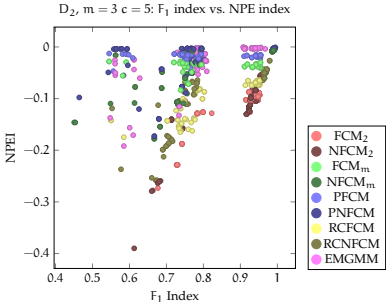


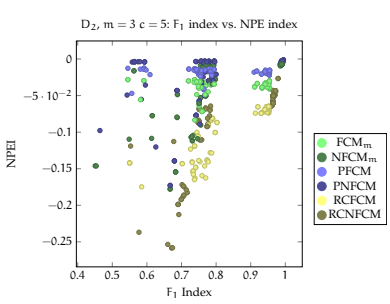




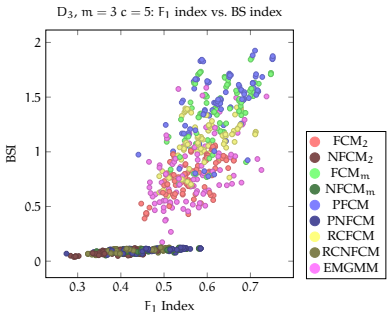


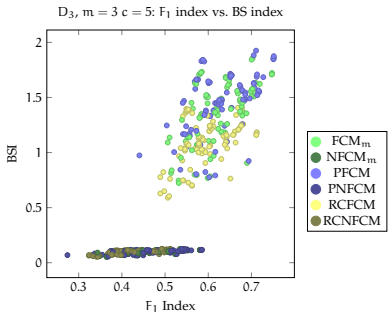


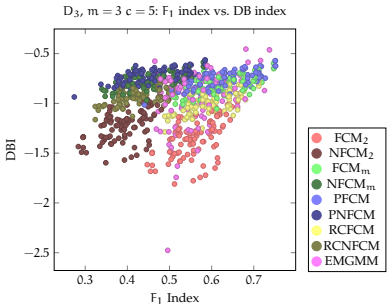


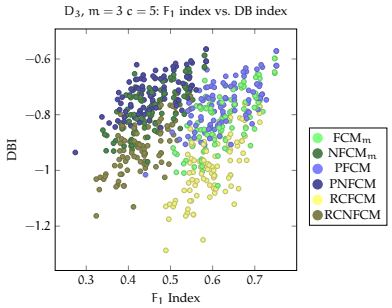


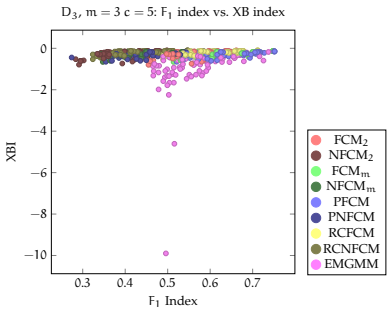
G.1.3 *Data Set Family* D_3

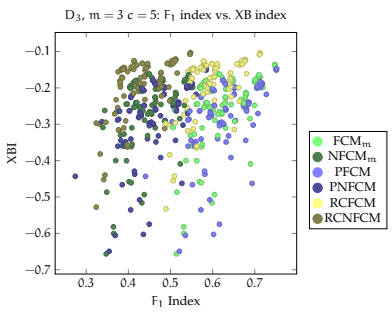


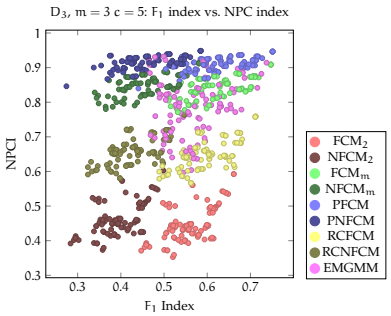


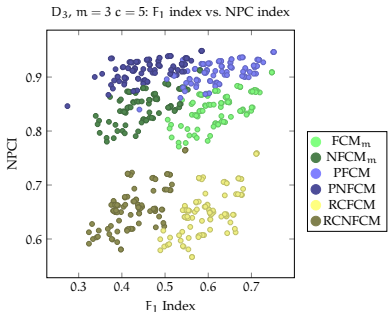


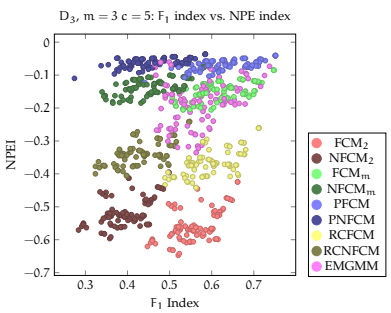


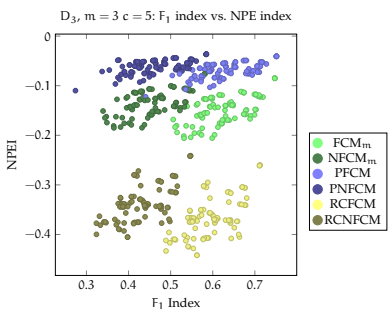










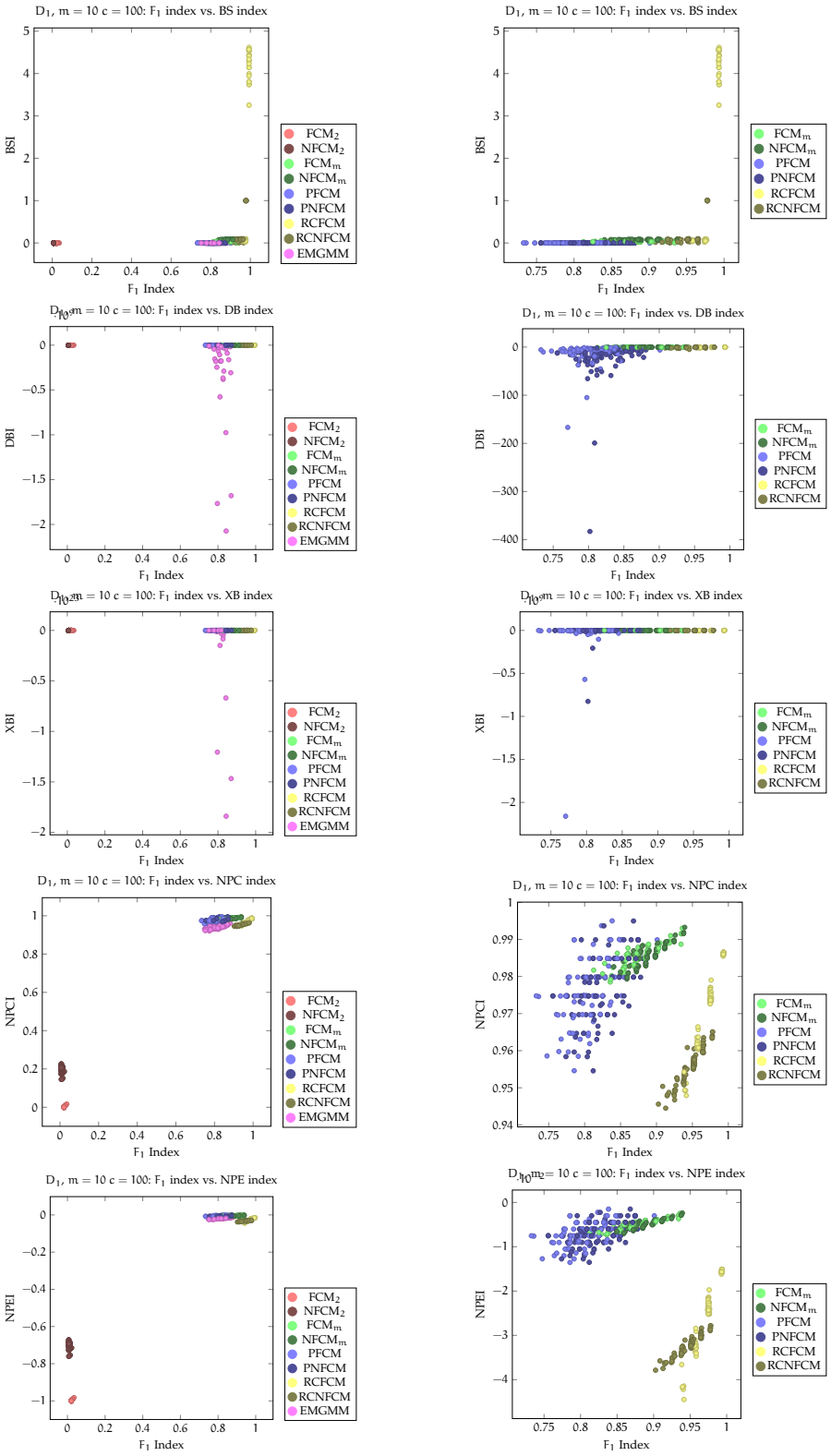


G.1.4 *Data Set Family* D_4

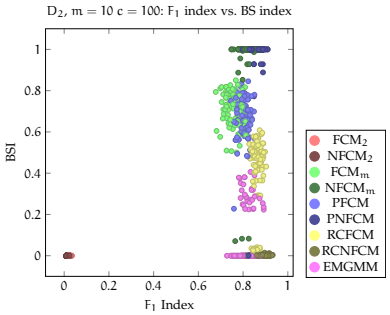
There are no data sets in data set family D_4, with $m = 3$ dimensions.

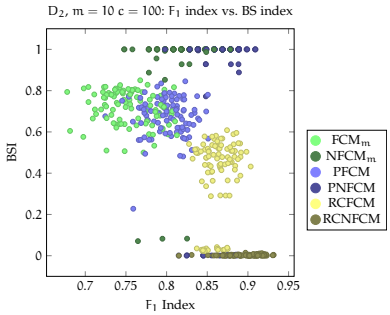
G.2 PLOTS FOR $m = 10$ AND $c = 100$

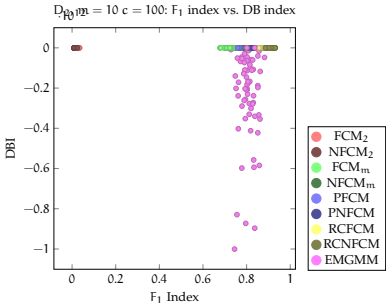
G.2.1 *Data Set Family* D_1

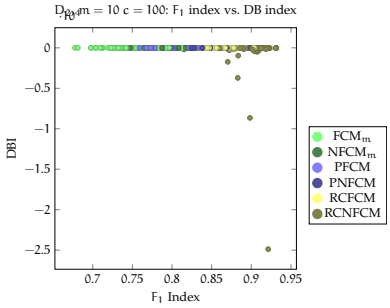


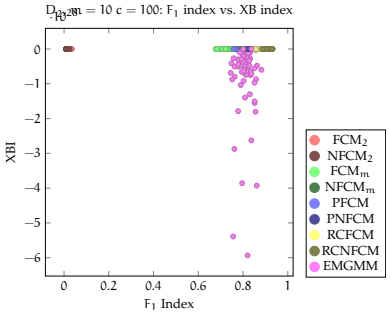
G.2.2 *Data Set Family* D_2

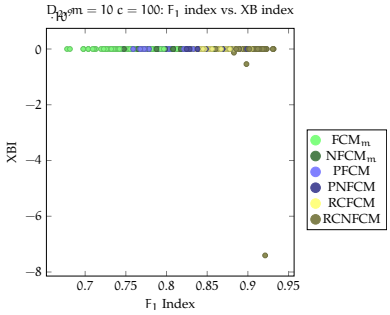


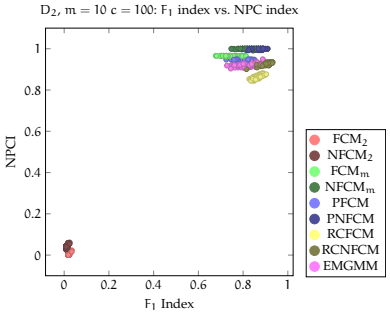


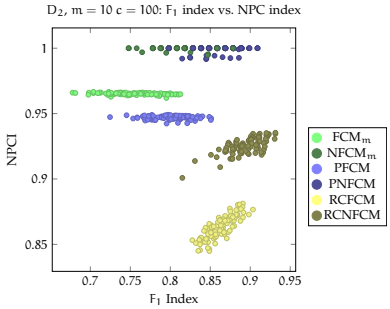


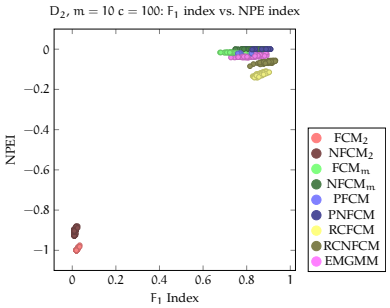


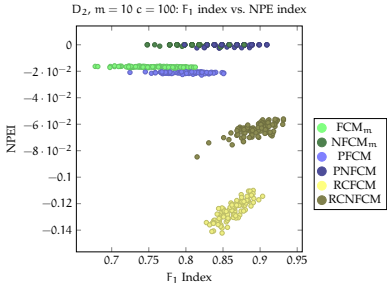




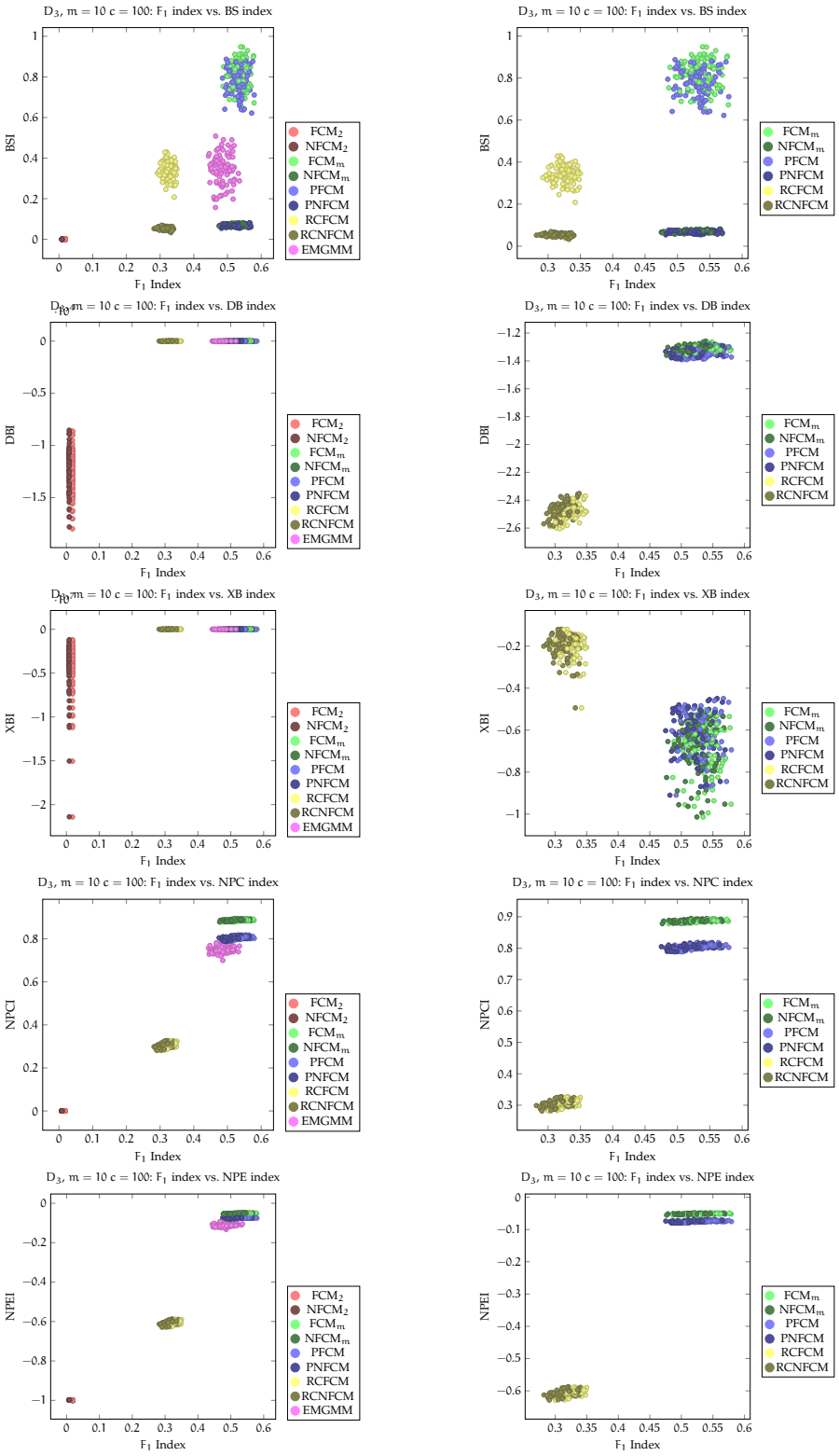




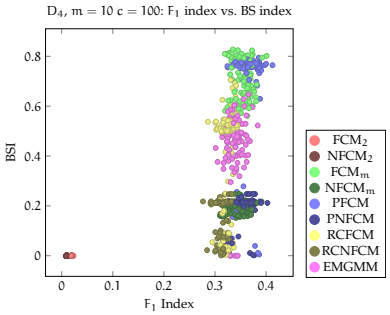


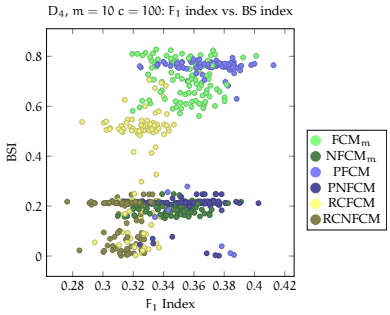


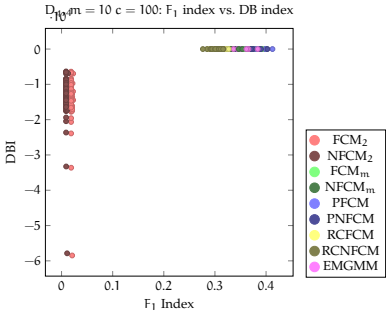
G.2.3 *Data Set Family* D_3

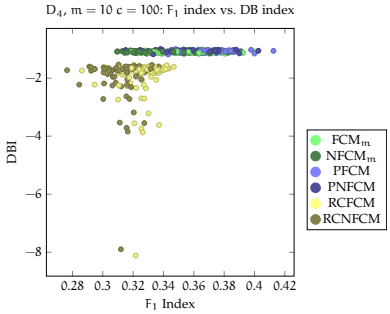


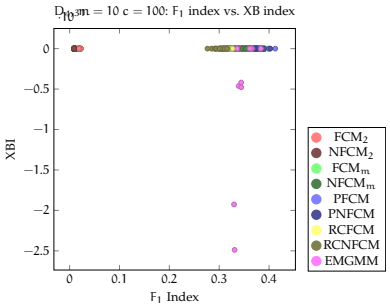
G.2.4 *Data Set Family* D_4

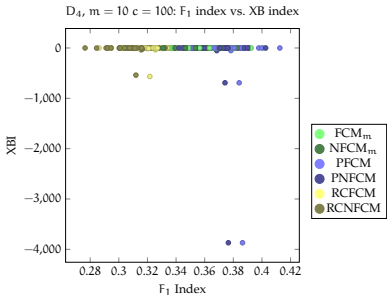


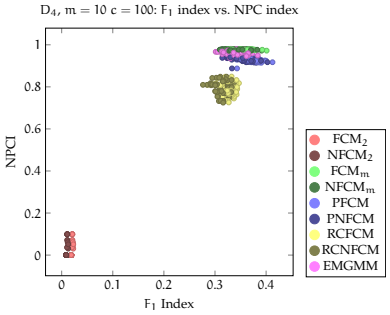


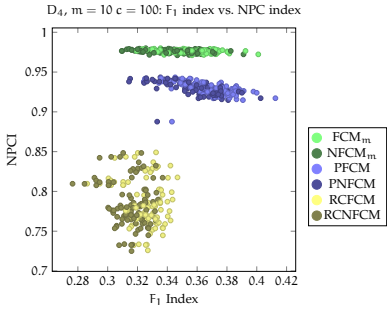


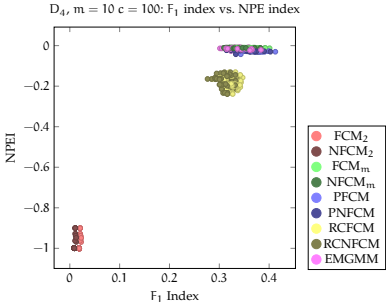


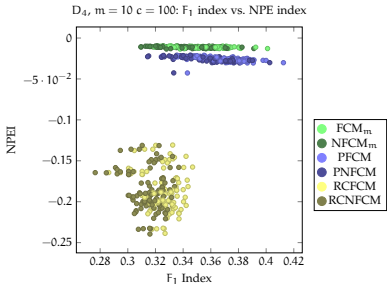






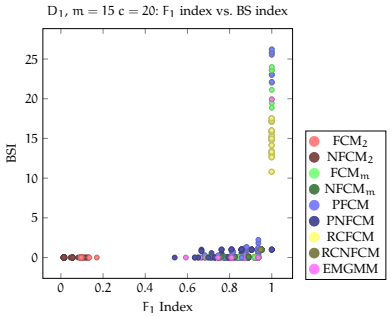


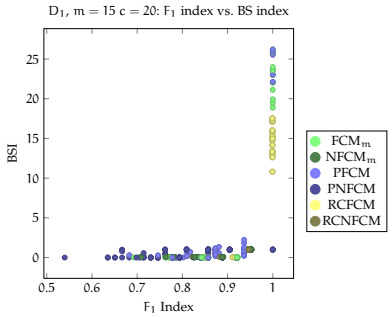


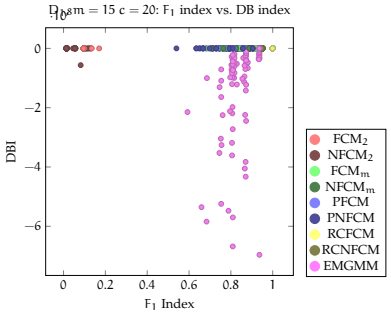


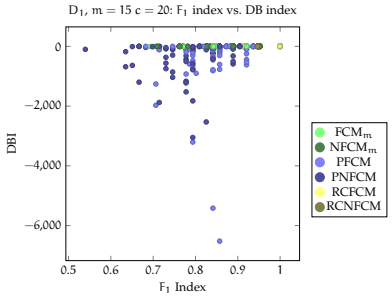
G.3 PLOTS FOR $m = 15$ AND $c = 20$

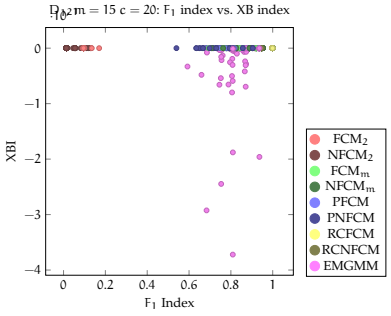
G.3.1 *Data Set Family* D_1

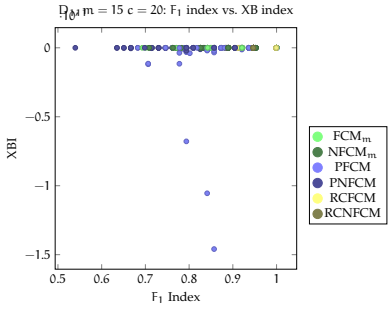


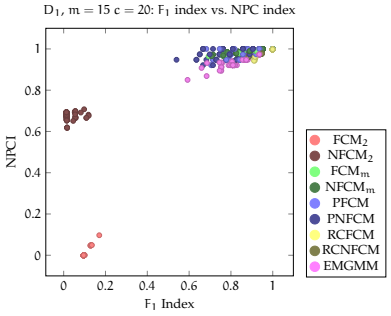


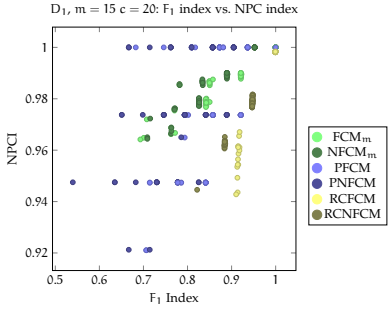


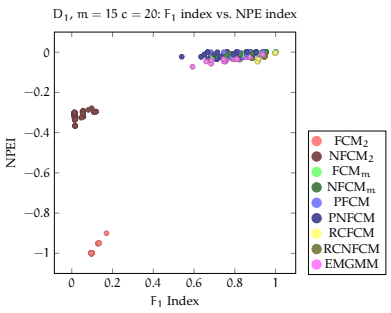


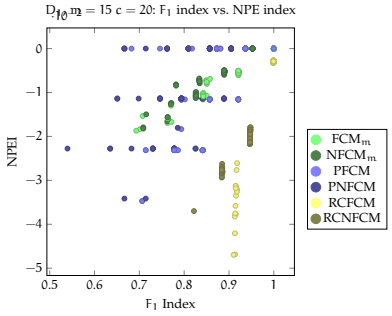




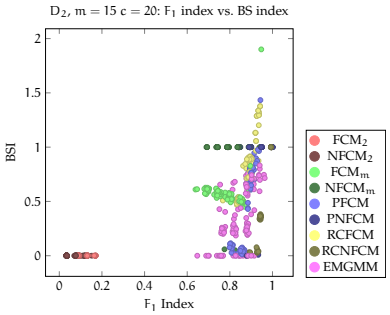


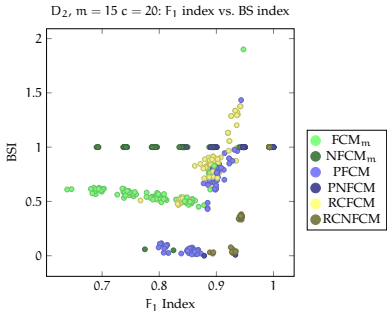


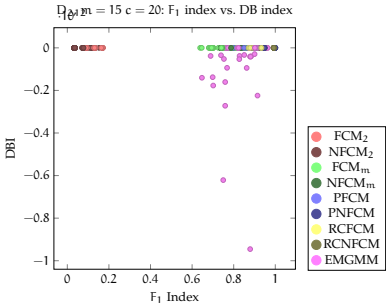


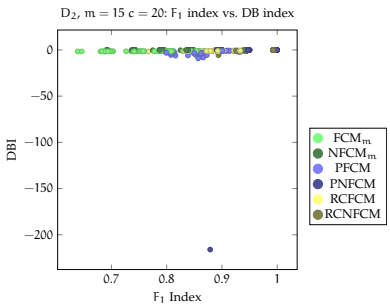


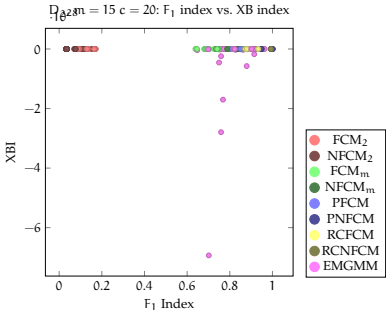
G.3.2 *Data Set Family* D_2

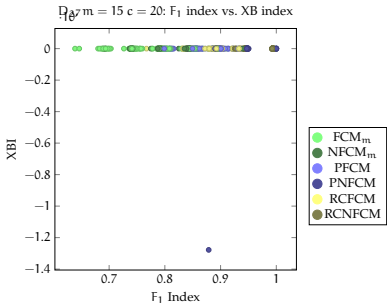


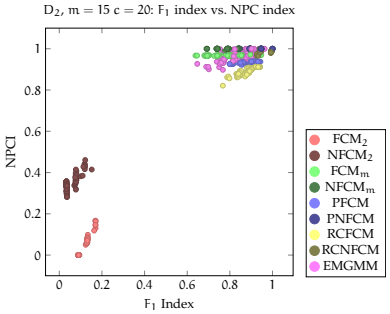


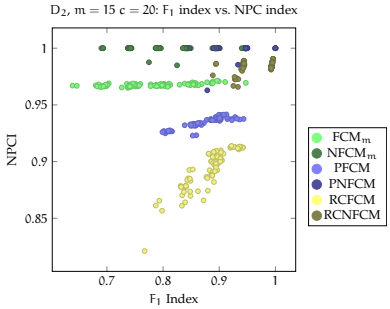


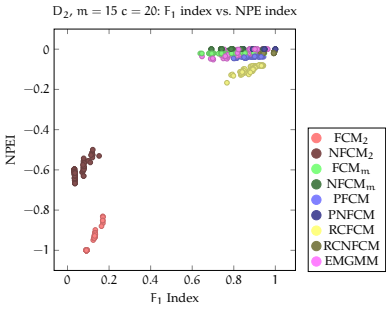


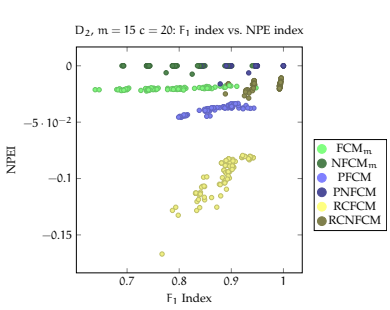




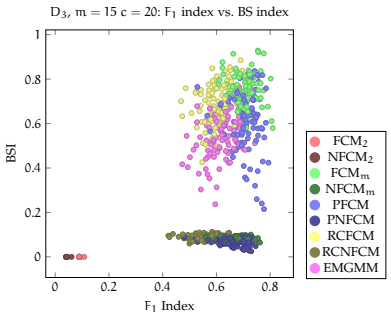


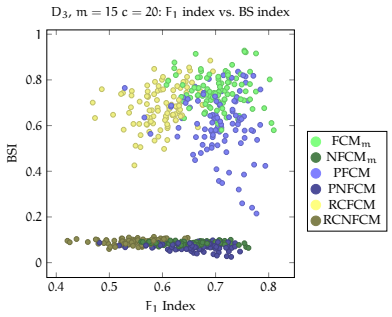


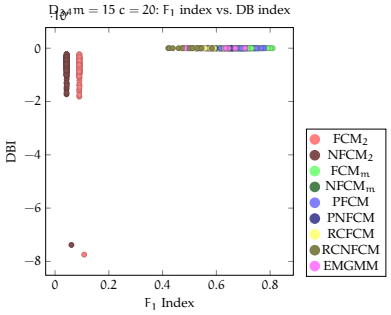


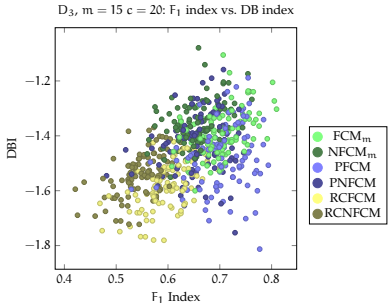


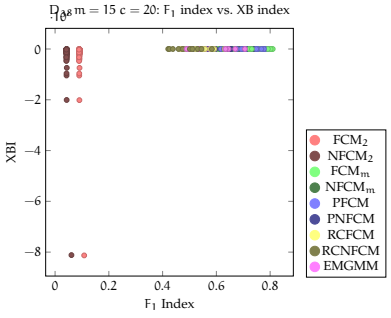
G.3.3 *Data Set Family* D_3

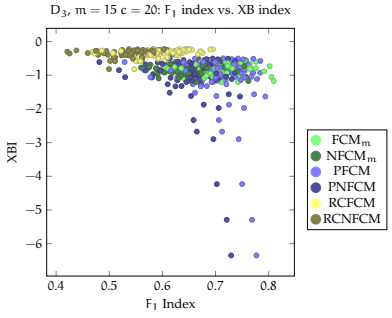


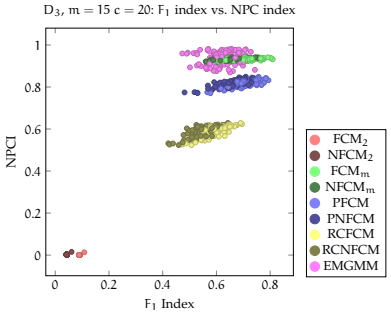


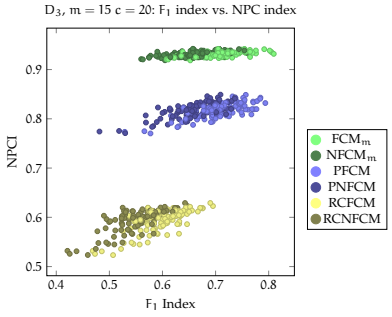


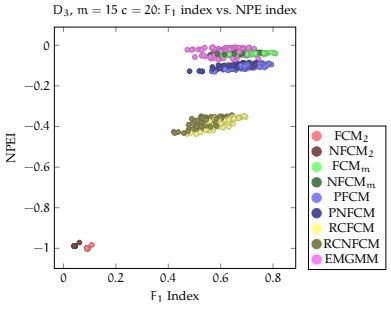


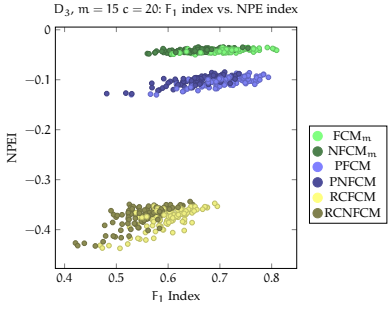




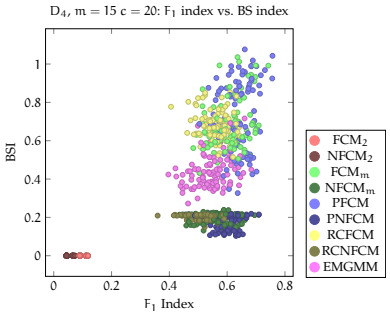


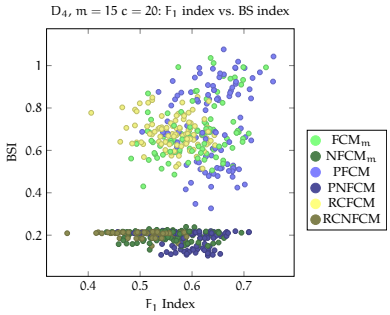


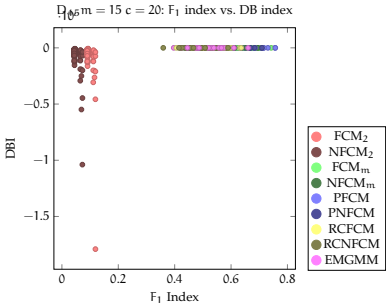


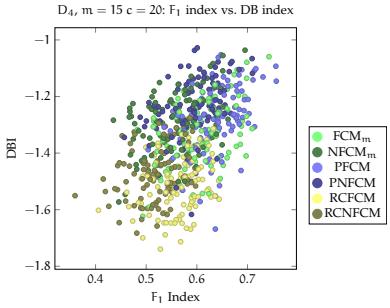


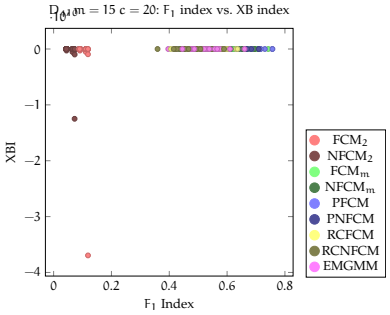
G.3.4 *Data Set Family* D_4

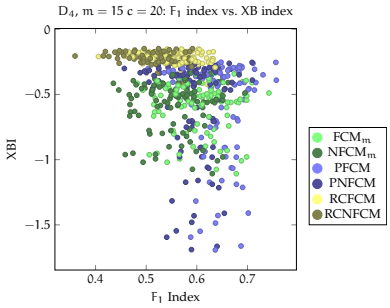


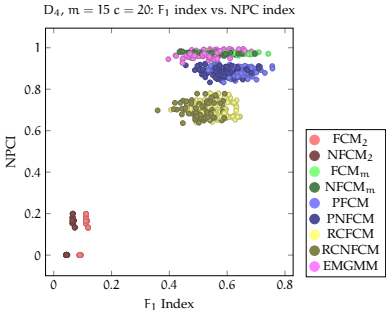


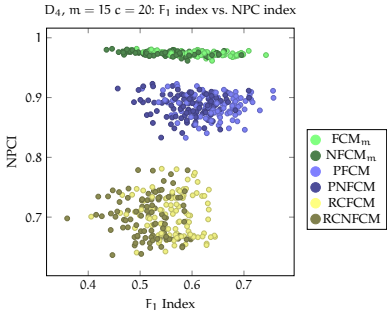


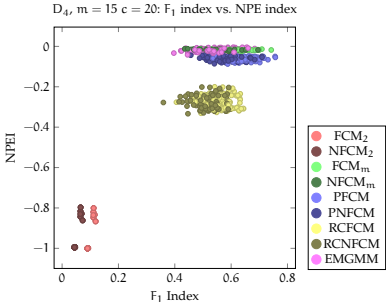


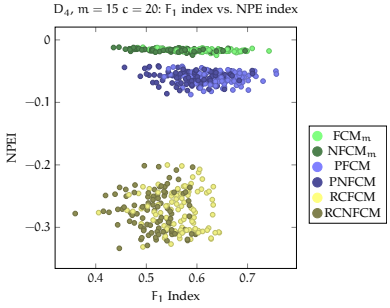






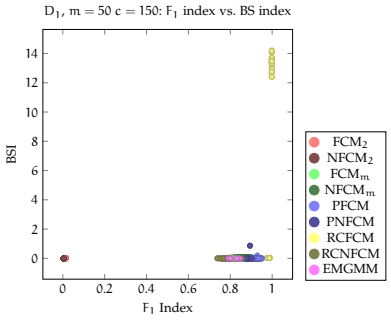


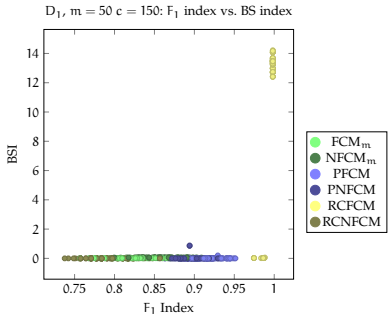


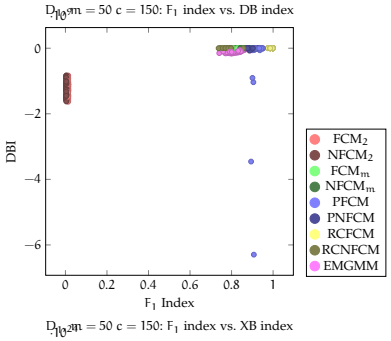


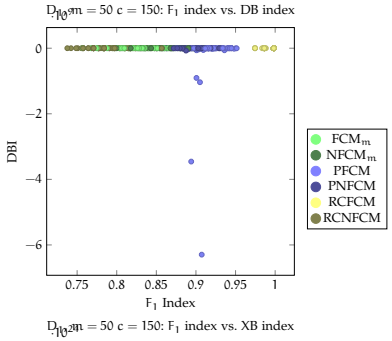
G.4 PLOTS FOR $m = 50$ AND $c = 150$

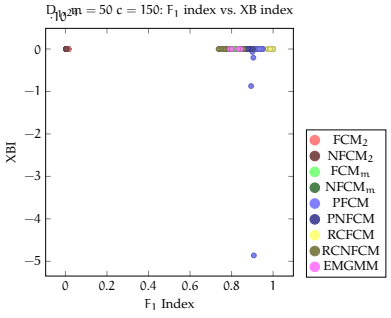
G.4.1 *Data Set Family* D_1

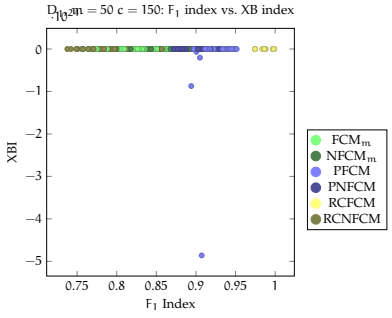


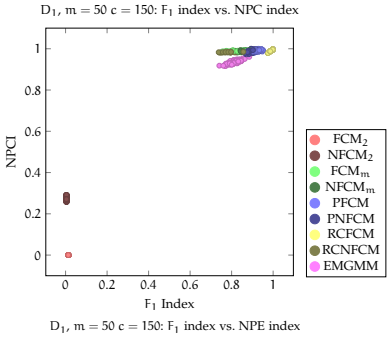


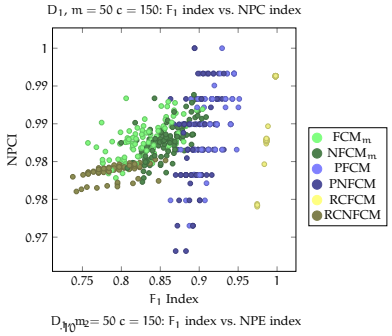


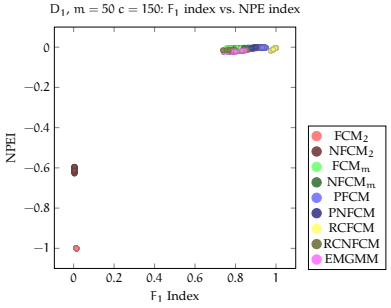


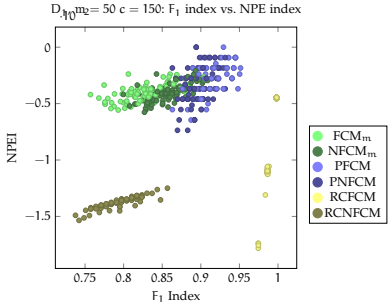




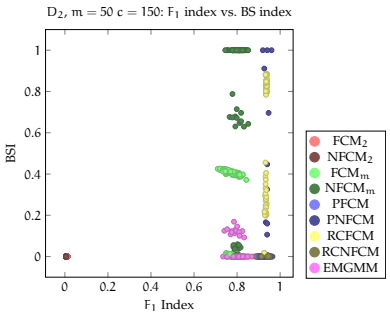


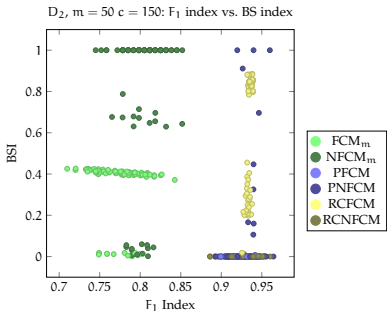


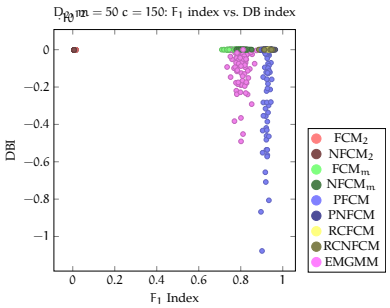


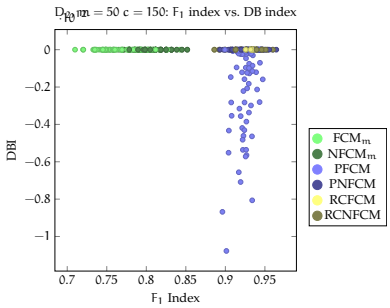


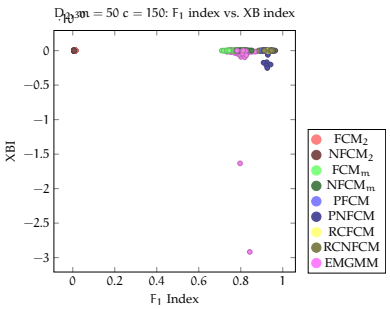
G.4.2 *Data Set Family* D_2

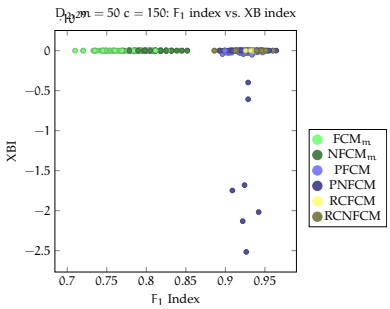


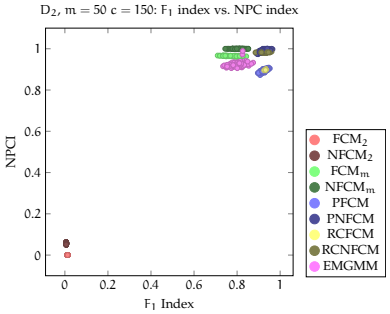


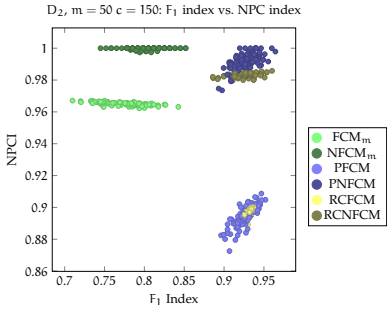


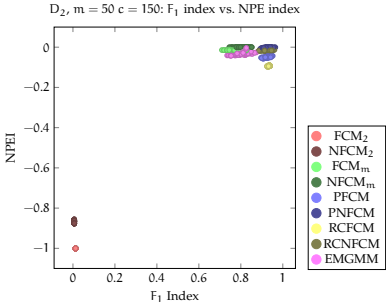


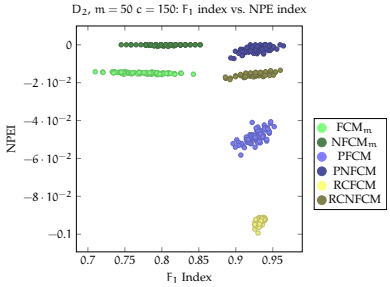




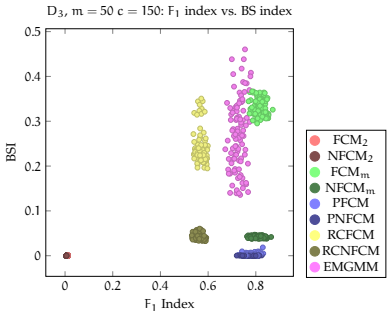


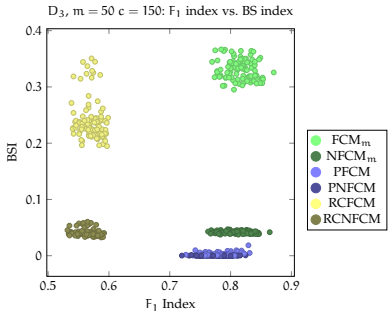


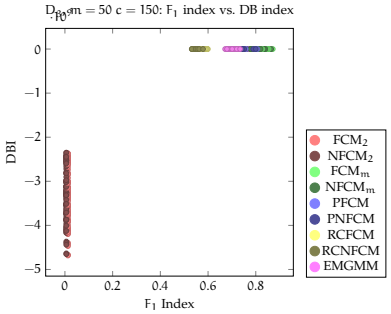


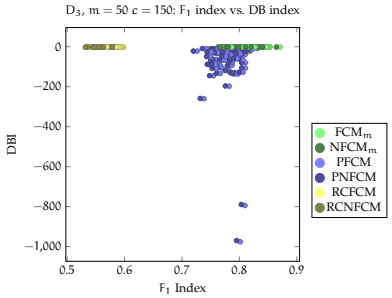


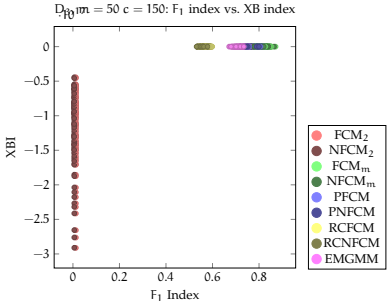
G.4.3 *Data Set Family* D_3

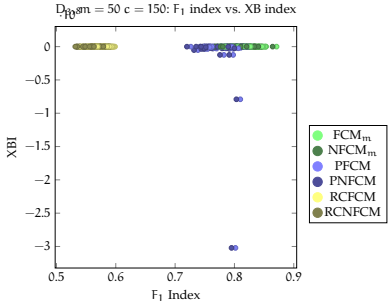


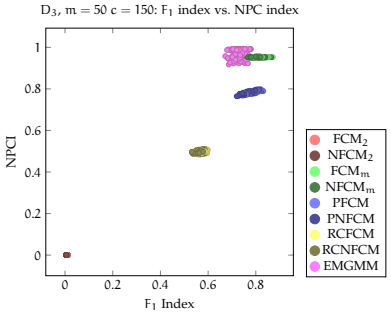


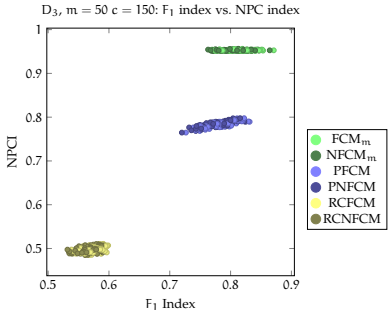


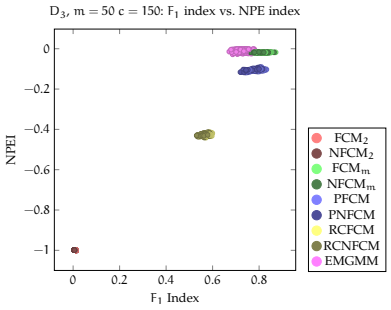


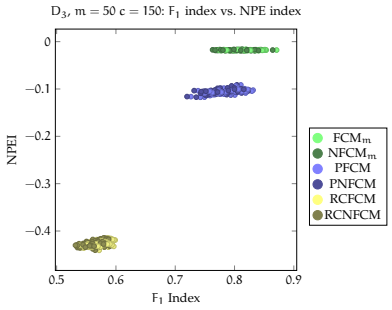




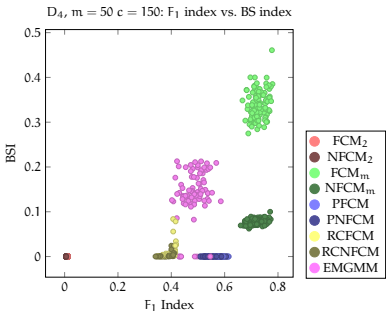


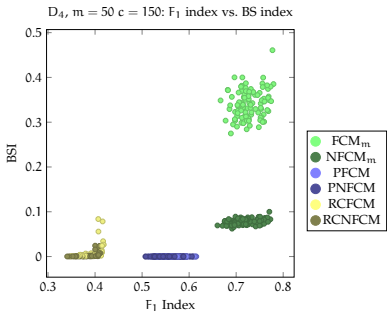


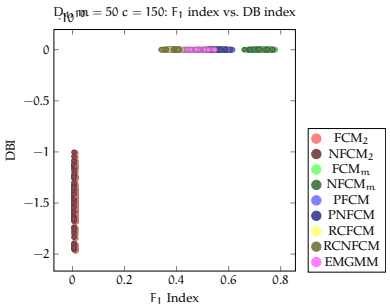


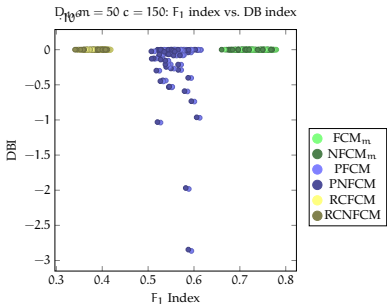


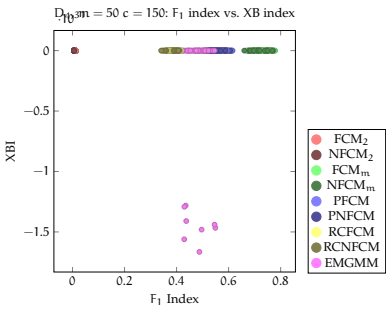
G.4.4 *Data Set Family* D_4

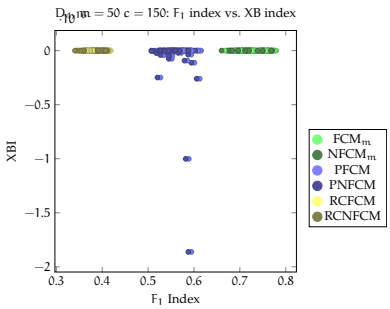


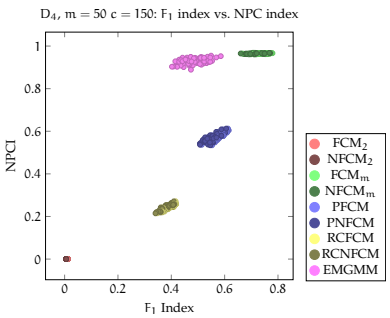


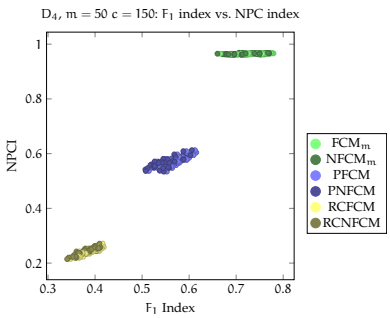


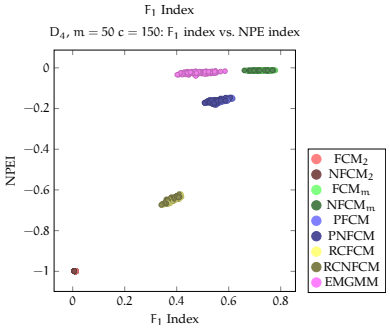


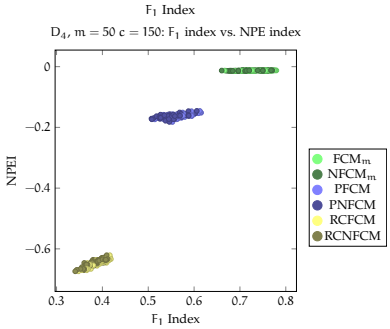












BIBLIOGRAPHY

Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

CharuC. Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 77–128. Springer US, 2012. ISBN 978-1-4614-3222-7. doi: 10.1007/978-1-4614-3223-4_4. URL http://dx.doi.org/10.1007/978-1-4614-3223-4_4.

Rakesh Aggarwal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8.

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009. ISSN 0885-6125. URL http://dx.doi.org/10.1007/s10994-009-5103-0. 10.1007/s10994-009-5103-0.

E. Anderson. The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, 1999. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/304181.304187.

Sitaram Asur, Duygu Ucar, and Srinivasan Parthasarathy. An ensemble framework for clustering protein–protein interaction networks. *Bioinformatics*, 23(13):i29–i40, 2007.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Eric Backer and Anil K. Jain. A clustering performance measure based on fuzzy set decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-3(1):66–75, Jan. 1981. ISSN 0162-8828. doi: 10.1109/TPAMI.1981.4767051.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press, 2001.

R.E. Bellman. *Dynamic Programming*. Dover Books on Mathematics. Dover Publications, 2003. ISBN 9780486428093. URL http://books.google.com/books?id=fyVtp3EMxasC.

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *Database Theory - ICDT'99*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer Berlin / Heidelberg, 1999. ISBN 978-3-540-65452-0. doi: 10.1007/3-540-49257-7_15. URL http://www.springerlink.com/content/04p94cqnbge862kh.

James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

James C Bezdek and Richard J Hathaway. Some notes on alternating optimization. In *Advances in Soft Computing–AFSS 2002*, pages 288–300. Springer, 2002.

JC Bezdek, WQ Li, Y Attikiouzel, and M Windham. A geometric approach to cluster validity for normal mixtures. *Soft Computing*, 1 (4):166–179, 1997.

B.R. Wilkins B.G. Batchelor. Method for location of clusters of patterns to initialise a learning machine. *Electronics Letters*, 5:481–483(2), October 1969. ISSN 0013-5194.

J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, University of California, Berkeley, 1997.

Christian Borgelt. *Data Mining with Graphical Models*. PhD thesis, Otto-von-Guericke University Magdeburg, 2000.

Christian Borgelt. *Prototype-based Classification and Clustering (Habilitationsschrift)*. PhD thesis, Otto-von-Guericke-University of Magdeburg, Germany, 2005. http://www.borgelt.net/habil.html.

Christian Borgelt. Resampling for fuzzy clustering. In *Proc. Symposium on Fuzzy Systems in Computer Science (FSCS 2006, Magdeburg, Germany)*, Magdeburg, Germany, 2006. Otto-von-Guericke-University of Magdeburg.

Christian Borgelt and Rudolf Kruse. Finding the number of fuzzy clusters by resampling. In *Proc. 16th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'06, Vancouver, Canada)*, Piscataway, NJ, USA, 2006. IEEE Press.

Ilja Bronstein and Konstantin Semendjajew. *Taschenbuch der Mathematik*. Nauka, Moskau und Teubner, Leipzig, 19. komplett überarbeitete auflage edition, 1979.

Sébastien Bubeck, Marina Meilă, and Ulrike von Luxburg. How the initialization affects the stability of the k-means algorithm. *ESAIM: Probability and Statistics*, 16:436–452, 2012.

Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Kaushik Chakrabarti and Sharad Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 89–100, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-715-3. URL `http://dl.acm.org/citation.cfm?id=645926.671852`.

Stephen L Chiu. Fuzzy model identification based on cluster estimation. *Journal of intelligent and Fuzzy systems*, 2(3):267–278, 1994.

MGCA Cimino, G Frosini, Beatrice Lazzerini, and Francesco Marcelloni. On the noise distance in robust fuzzy c-means. In *Proceeding of world academy of science, Engineering and Technology*, volume 1, pages 361–364. Citeseer, 2005.

Pierre Comon. Independent component analysis, a new concept? *Signal Process.*, 36(3):287–314, April 1994. ISSN 0165-1684. doi: 10.1016/0165-1684(94)90029-9. URL `http://dx.doi.org/10.1016/0165-1684(94)90029-9`.

Martin Cooke, Phil Green, Ljubomir Josifovski, and Ascension Vizinho. Robust automatic speech recognition with missing and unreliable acoustic data. *Speech communication*, 34(3):267–285, 2001.

Antonio Cuevas, Manuel Febrero, and Ricardo Fraiman. Estimating the number of clusters. *Canadian Journal of Statistics*, 28(2):367–382, 2000.

Rajesh N. Dave. Characterization and detection of noise in clustering. *Pattern Recogn. Lett.*, 12(11):657–664, 1991. ISSN 0167-8655.

David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(2): 224–227, 1979.

Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

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

D. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. Lecture delivered at the conference "Math Challenges of the 21st Century" held by the American Math. Society organised in Los Angeles, August 6-11, August 2000.

J.C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Cybernetics and Systems: An International Journal*, 3(3):32–57, 1973. doi: 10.1080/01969727308546046. URL http://www.informaworld.com/10.1080/01969727308546046.

Robert J. Durrant and Ata Kabán. When is 'nearest neighbour' meaningful: A converse theorem and implications. *Journal of Complexity*, 25(4):385 – 397, 2008. ISSN 0885-064X. doi: DOI:10.1016/j.jco.2009.02.011. URL http://www.sciencedirect.com/science/article/B6WHX-4VXB8V2-1/2/9e931f06c677abeb1a1589a8e759197c.

Martin Ester, Hans-Peter Kriegel, S. Jörg, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.1980.

Usama M Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. Advances in knowledge discovery and data mining. *The MIT Press*, 2 1996.

Fisher. Theory of statistical estimation. In *Proc. Cambridge Philosophical Society*, volume 22, pages 700–725. Cambridge University Press, Cambridge, 1925.

C. Fraley and A. E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, August 1998. doi: 10.1093/comjnl/41.8.578. URL http://dx.doi.org/10.1093/comjnl/41.8.578.

Damien François, Vincent Wertz, and Michel Verleysen. The concentration of fractional distances. *Knowledge and Data Engineering, IEEE Transactions on*, 19(7):873 –886, 7 2007. ISSN 1041-4347. doi: 10.1109/TKDE.2007.1037.

William J Frawley, Gregory Piatetsky-Shapiro, and Christopher J Matheus. Knowledge discovery in databases: An overview. *AI magazine*, 13(3):57, 1992.

Hichem Frigui and Raghu Krishnapuram. A robust algorithm for automatic extraction of an unknown number of clusters

from noisy data. *Pattern Recognition Letters*, 17(12):1223 – 1232, 1996. ISSN 0167-8655. doi: DOI:10.1016/0167-8655(96)00080-3. URL http://www.sciencedirect.com/science/article/B6V15-3WP2DPP-S/2/9cf46c6a0875ff2051d8d42cb25653fc.

Hichem Frigui and Raghu Krishnapuram. Clustering by competitive agglomeration. *Pattern Recognition*, 30(7):1109 – 1119, 1997. ISSN 0031-3203. doi: DOI:10.1016/S0031-3203(96)00140-9. URL http://www.sciencedirect.com/science/article/B6V14-3SNVHWM-M/2/3c5ec045f0a0662d00ab583aab028f9f.

Isak Gath and Amir B. Geva. Unsupervised optimal fuzzy clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7): 773–780, 1989.

Mark Girolami. Mercer kernel-based clustering in feature space. *Neural Networks, IEEE Transactions on*, 13(3):780–784, 2002.

Phil Green, Jon Barker, Martin Cooke, and Ljubomir Josifovski. Handling missing and unreliable information in speech recognition. In *AISTATS 2001, Florida*, 2001.

Donald E. Gustafson and William C. Kessel. Fuzzy clustering with a fuzzy covariance matrix. In *IEEE*, volume 17, pages 761–766, Jan. 1978. doi: 10.1109/CDC.1978.268028.

Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.8934.

Isabelle Guyon. *Feature extraction: foundations and applications*. Studies in fuzziness and soft computing. Springer-Verlag, 2006. ISBN 9783540354871. URL http://books.google.com/books?id=x5hdbK8bIG0C.

Mark A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato, 1999.

Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 506–517. Morgan Kaufmann, 1999. ISBN 1-55860-615-7.

Alexander Hinneburg and Daniel A. Keim. A general approach to clustering in large databases with noise. *Knowledge and Information Systems*, 5(4):387–415, November 2003. ISSN 0219-1377 (Print) 0219-3116 (Online). doi: 10.1007/s10115-003-0086-9. URL http://www.springerlink.com/content/cde7exf7u79fdfjl.

Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 506–515, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-715-3.

F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis*. John Wiley & Sons, Chichester, England, 1999.

Frank Höppner and Frank Klawonn. A new approach to fuzzy partitioning. In *In Proc. of the Joint 9th IFSA World Congress and 20th NAFIPS Int. Conf*, pages 1419–1424, 2001.

Frank Höppner and Frank Klawonn. Improved fuzzy partitions for fuzzy regression models. *Int. J. Approx. Reasoning*, 32(2-3):85–102, 2003.

Harold Hotelling and Margaret Richards Pabst. Rank correlation and tests of significance involving no assumption of normality. *The Annals of Mathematical Statistics*, 7(1):29–43, 1936.

A. Hotho, S. Staab, and A. Maedche. Ontology-based text clustering. In *In Proceedings of the IJCAI-2001 Workshop Text Learning: Beyond Supervision*, 2001.

Chih-Ming Hsu and Ming-Syan Chen. On the necessary and sufficient conditions of a meaningful distance function for high dimensional data space. In *SDM*, 2006.

Chih-Ming Hsu and Ming-Syan Chen. On the design and applicability of distance functions in high-dimensional data space. *Knowledge and Data Engineering, IEEE Transactions on*, 21(4):523 –536, 4 2009. ISSN 1041-4347. doi: 10.1109/TKDE.2008.178.

A. Irle. *Wahrscheinlichkeitstheorie und Statistik: Grundlagen - Resultate - Anwendungen*. Lehrbuch Mathematik. Teubner, 2005. ISBN 9783322871992. URL `http://books.google.de/books?id=-mGRLA9w6VYC`.

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

B. Jayaram and F. Klawonn. Can unbounded distance measures mitigate the curse of dimensionality? *Int. J. Data Mining, Modelling and Management*, x(x):xxx–xxx., x 2012. (accepted paper).

I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, October 2002. ISBN 0387954422. URL `http://www.worldcat.org/isbn/0387954422`.

A. Kabán. Fractional norm regularization: Learning with very few relevant features. *Neural Networks and Learning Systems, IEEE Transactions on*, 24(6):953–963, June 2013. ISSN 2162-237X. doi: 10.1109/TNNLS.2013.2247417.

Ata Kabán. On the distance concentration awareness of certain data reduction techniques. *Pattern Recognition*, 44(2):265 – 277, 2011. ISSN 0031-3203. doi: DOI:10.1016/j.patcog.2010.08.018. URL `http://www.sciencedirect.com/science/article/B6V14-50T41D8-1/2/ad372c0fe81d9b0e19fc39673348d5a6`.

Ata Kabán. Non-parametric detection of meaningless distances in high dimensional data. *Statistics and Computing*, 22(2):375–385, 2012.

William Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Masters thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968.

Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

Frank Klawonn. Fuzzy clustering: insights and a new approach. *Mathware and Soft Computing*, 11:125–142, 2004. URL `http://hdl.handle.net/2099/3642`.

Frank Klawonn and Frank Höppner. What is fuzzy about fuzzy clustering? understanding and improving the concept of the fuzzifier. In Michael R. Berthold, Hans-Joachim Lenz, Elizabeth Bradley, Rudolf Kruse, and Christian Borgelt, editors, *Advances in Intelligent Data Analysis V*, volume 2810 of *Lecture Notes in Computer Science*, pages 254–264. Springer Berlin / Heidelberg, 2003a.

Frank Klawonn and Frank Höppner. An alternative approach to the fuzzifier in fuzzy clustering to obtain better clustering. In *EUSFLAT Conf.*, pages 730–734, 2003b.

Donald Ervin Knuth. *The art of computer programming*. Pearson Education, 2005.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

T. Kohonen. *Self-organizing maps*. Springer series in information sciences. Springer, 2001. ISBN 9783540679219. URL `http://books.google.de/books?id=e4igHzyf078C`.

H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif., 1951. University of California Press.

HW Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52(1):7–21, 2005.

S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

Ludmila I Kuncheva and Dmitry P Vetrov. Evaluation of stability of k-means cluster ensembles with respect to random initialization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(11): 1798–1808, 2006.

Deval A Lashkari, Joseph L DeRisi, John H McCusker, Allen F Namath, Cristl Gentile, Seung Y Hwang, Patrick O Brown, and Ronald W Davis. Yeast microarrays for genome wide parallel genetic and gene expression analysis. *Proceedings of the National Academy of Sciences*, 94(24):13057–13062, 1997.

Zhongping Lee, Kendall L Carder, Curtis D Mobley, Robert G Steward, and Jennifer S Patch. Hyperspectral remote sensing for shallow waters. 2. deriving bottom depths and water properties by optimization. *Applied Optics*, 38(18):3831–3843, 1999.

Xiaoke Ma, Lin Gao, Xuerong Yong, and Lidong Fu. Semi-supervised clustering algorithm for community structure detection in complex networks. *Physica A: Statistical Mechanics and its Applications*, 389(1): 187–197, 2010.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In Sandip Das and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation*, volume 5431 of *Lecture Notes in Computer Science*, pages 274–285. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-00201-4.

Farid Melgani and Lorenzo Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *Geoscience and Remote Sensing, IEEE Transactions on*, 42(8):1778–1790, 2004.

K Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on*, 12(2):181–201, 2001.

Scott A Ness. Basic microarray analysis. In *Bioinformatics and Drug Discovery*, pages 13–33. Springer, 2006.

Dale A Ostlie and Bradley W Carroll. *An introduction to modern astrophysics*. Addison-Wesley, 2006.

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Siddheswar Ray and Rose H Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, pages 137–143, 1999.

Frank Rehm, Roland Winkler, and Rudolf Kruse. Fuzzy clustering with repulsive prototypes. *Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design*, pages 332–336, 2010.

Enrique H. Ruspini. A new approach to clustering. *Information and Control*, 15(1):22 – 32, 1969. ISSN 0019-9958. doi: DOI: 10.1016/S0019-9958(69)90591-9. URL http://www.sciencedirect.com/science/article/pii/S0019995869905919.

Tanwistha Saha, Carlotta Domeniconi, and Huzefa Rangwala. Detection of communities and bridges in weighted networks. In *Machine Learning and Data Mining in Pattern Recognition*, pages 584–598. Springer, 2011.

J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.*, 18:401–409, May 1969. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/T-C.1969.222678. URL http://dx.doi.org/10.1109/T-C.1969.222678.

M Sarkar and T Y Leong. Fuzzy k-means clustering with missing values. *Proceedings of the AMIA Symposium*, pages 588–592, 2001.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In Wulfram Gerstner, Alain Germond, Martin Hasler, and Jean-Daniel Nicoud, editors, *Artificial Neural Networks - ICANN'97*, volume 1327 of *Lecture Notes in Computer Science*, pages 583–588. Springer Berlin / Heidelberg, 1997. ISBN 978-3-540-63631-1. URL http://dx.doi.org/10.1007/BFb0020217. 10.1007/BFb0020217.

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

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

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

M. Shafiei, Singer Wang, R. Zhang, E. Milios, Bin Tang, J. Tougas, and R. Spiteri. Document representation and dimension reduction for text clustering. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 770–779, 2007. doi: 10.1109/ICDEW.2007.4401066.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

Vin D Silva and Joshua B Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in neural information processing systems*, pages 705–712, 2002.

Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer, 2005.

Joshua B. Tenenbaum, Vin Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000. ISSN 00368075. doi: 10.1126/science.290.5500.2319. URL `http://dx.doi.org/10.1126/science.290.5500.2319`.

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

C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, United Kingdom, second edition, 1979.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1073–1080. ACM, 2009.

Kiri Wagstaff. Clustering with missing values: No imputation required. In *Clustering, and Data Mining Applications (Proceedings of the Meeting of the International Federation of Classification Societies*, pages 649–658. Springer, 2004.

Roland Winkler, Frank Klawonn, Frank Höppner, and Rudolf Kruse. Fuzzy cluster analysis of larger data sets. In M.-J. Lesot A. Laurent, editor, *Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design*, pages 302–331. IGI Global: Information Science Reference, 2010a. ISBN 9781605668581.

Roland Winkler, Frank Rehm, and Rudolf Kruse. Clustering with repulsive prototypes. In *Advances in data analysis, data handling and business intelligence*, number 2 in Studies in Classification, Data Analysis and Knowledge Organization, pages 207–215. Springer, 2010b. ISBN 978-3-642-01043-9.

Roland Winkler, Annette Temme, Chrisoph Bösel, and Rudolf Kruse. Clustering radar tracks to evaluate efficiency indicators. In *Proceedings of the second ENRI Workshop on ATM and CNS*, pages 71–94, Tokyo, Japan, 11 2010c. ENRI.

Roland Winkler, Frank Klawonn, and Rudolf Kruse. Fuzzy c-means in high dimensional spaces. *IJFSA*, 1(1):1–16, 2011a.

Roland Winkler, Frank Klawonn, and Rudolf Kruse. M-estimator induced fuzzy clustering algorithms. In *Advances in Intelligent Systems Research*, volume 1 of *EUSFLAT*, pages 298–304. Atlantis Press, 7 2011b.

Roland Winkler, Frank Klawonn, and Rudolf Kruse. Fuzzy clustering with polynomial fuzzifier function in connection with m-estimators. In *Zeitschrift: Applied and Computational Mathematics*, volume 10 of *Special Issue on Fuzzy Set Theory and Applications*, pages 146–163. Azerbaijan National Academy of Sciences, 2011c.

Roland Winkler, Frank Klawonn, and Rudolf Kruse. Problems of fuzzy c-means clustering and similar algorithms with high dimensional data sets. In Wolfgang A. Gaul, Andreas Geyer-Schulz, Lars Schmidt-Thieme, and Jonas Kunze, editors, *Challenges at the Interface of Data Analysis, Computer Science, and Optimization*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 79–87. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-24465-0.

Roland Winkler, Frank Klawonn, and Rudolf Kruse. A new distance function for prototype based clustering algorithms in high dimensional spaces. In Paolo Giudici, Salvatore Ingrassia, and Maurizio Vichi, editors, *Statistical Models for Data Analysis*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 371–378. Springer International Publishing, 2013. ISBN 978-3-319-00031-2.

C. F. Wu. On the convergence properties of the em algorithm. *Ann. Stat.*, 11(1):95–103, 1983.

Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Transactions on pattern analysis and machine intelligence*, 13(8):841–847, 1991.

R Yager and D Filev. Generation of fuzzy rules by mountain clustering. *Journal of Intelligent and Fuzzy Systems*, 2(3):209–219, 1994.

L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965. ISSN 0019-9958. doi: DOI:10.1016/S0019-9958(65)90241-X. URL http://www.sciencedirect.com/science/article/B7MFM-4DX43MN-W3/2/f244f7a33f31015e819042700cd83047.

Dao-Qiang Zhang and Song-Can Chen. Clustering incomplete data using kernel-based fuzzy c-means algorithm. *Neural Processing Letters*, 18(3):155–162, 2003.

Daoqiang Zhang and Songcan Chen. Fuzzy clustering using kernel method. In *The 2002 International Conference on Control and Automation, 2002. ICCA, 2002*. Citeseer, 2002.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM, 1996.

EHRENERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 11.12.2015

Roland Winkler