Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Simultaneous Visual Analysis of Multidimensional Data and Attributes

Marcus Philip Rottschäfer

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	Dipl.-Inf. Christoph Schulz, Moataz Abdelaal, M.Sc.
Commenced:	October 5, 2018
Completed:	April 5, 2019

Abstract

As more and more data becomes available through a variety of sources, for example, the Internet of Things, the need for analyzing high-dimensional data increases permanently. Visual analysis of multi-dimensional data offers an intuitive approach to better understand datasets, as humans are generally good at reasoning over spatial objects. To access a better comprehension of a dataset, data analysts often use visualization techniques in the data exploration phase. Especially convenient are dimensionality reduction techniques. They can be used for visualizations that depict the structure of the dataset and the relationship of the high-dimensional datapoints. However, visualizations based on dimensionality reduction often suffer from limited interpretability of the low-dimensional map. This is because the map does not show the relationship the low-dimensional datapoints have with the high-dimensional attributes. The Data Context Map algorithm proposed by Cheng and Mueller [CM16] solves this issue by providing a low-dimensional map that accounts for the datapoint relationship, the attribute relationship, and the datapoint-attribute relationship. In this thesis, we present an implementation of the Data Context Map algorithm, enhanced by an interaction technique called Probing [SDMT16] that allows to further reduce the error of the low-dimensional map locally. This implementation makes it possible to create visualizations of high-dimensional datasets and its attributes that improve the interpretation and the understanding of the data. Decision-making processes can be enhanced as high-dimensional data can be better analyzed. We apply the augmented Data Context Map to a variety of real-world datasets to show the advanced interpretability of the low-dimensional map.

Contents

List of Figures

List of Tables

List of Abbreviations

AKDE Adaptive Kernel Density Estimation. 17, 25

CM Composite Distance Matrix. 31

CMDS Classical Multidimensional Scaling. 19

CPU Central Processing Unit. 21

DCM Data Context Map. 14, 29

EVD Eigenvalue Decomposition. 18

Glimmer MDS Glimmer Multidimensional Scaling. 19, 21

GPU Graphics Processing Unit. 19

KDE Kernel Density Estimation. 24

MDS Multidimensional Scaling. 17, 19

Metric MDS Metric Multidimensional Scaling. 19

NWKR Nadaraya–Watson Kernel Regression. 17, 26

PCA Principal Component Analysis. 17, 18

PDF Probability Density Function. 24

SNE Stochastic Neighbor Embedding. 23

SVD Singular Value Decomposition. 18

t-SNE t-Distributed Stochastic Neighbor Embedding. 13, 23

UMAP Uniform Manifold Approximation and Projection. 52

1 Introduction

Visualization of high-dimensional data refers to methods that create informative graphical representations of high-dimensional datasets. In times of big data and the information age, more and more data becomes available. These enormous amounts of data need to be analyzed to extract knowledge and to gain further insight. With the understanding of the datasets in hand, decision-making processes can be enhanced, leading to better results throughout a variety of domains.

The visualization of high-dimensional data is an intuitive approach to get a better understanding of a dataset. As stated by Grinstein et al. [GTC], the visualizations are especially important in the data exploration phase, where a first look is taken at a dataset to evaluate further applicable data mining approaches. With a graphical representation, one is given an easy and almost non-technical alternative to gain insight into the structure of a dataset. Due to this, visualizations are used throughout a variety of domains [MH08], ranging from biology and medicine (e.g. visualizing DNA sequences [HGM+97]) over business use cases (sales team performance data [CM16]) to word vector visualizations [Heu16].

Dimensionality reduction is a popular way to find a visualization of high-dimensional data. The objective of a dimensionality reduction algorithm is to map a high-dimensional dataset into a lower-dimensional dataset (for visualizations e.g. 2-3 dimensions), where most of the inherent structure of the dataset is kept. This gives a low-dimensional embedding of the data, typically shown in a scatterplot, that highlights clustering and relationships of the original dataset. With recent advancements in the field, for example with t-Distributed Stochastic Neighbor Embedding (t-SNE) [MH08] or other non-linear techniques, dimensionality-reduced visualization of data becomes more and more popular. An example is given in Figure 1.1. The scatterplot shows the low-dimensional embedding obtained from applying a dimensionality reduction algorithm to the four-dimensional Iris dataset [Fis36].

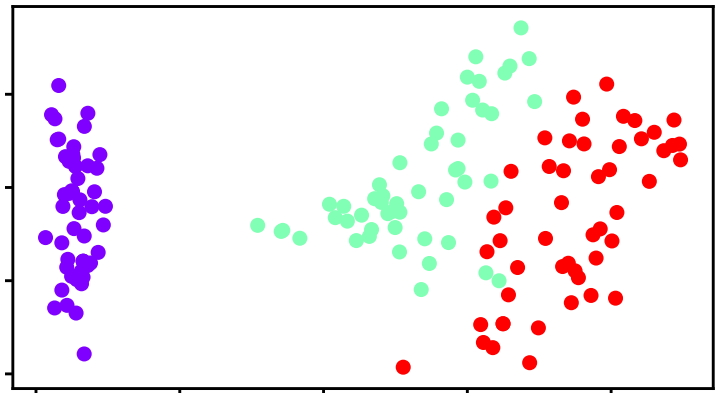


Figure 1.1: Scatterplot of two-dimensional datapoints, obtained by applying a dimensionality reduction algorithm (t-SNE) to the four-dimensional Iris dataset [Fis36].

Plotting a 4-dimensional datapoint into a scatterplot is not possible, that is why a reduction to 2 dimensions is needed. The results show that there are three clusters (corresponding to the groupings of the three colors), giving a better insight into the structure of the dataset.

Although these visualizations of dimensionality-reduced data are very useful to understand clusterings, relationships of datapoints in regards to each other and the overall structure of the dataset, they typically come with some drawbacks involved. For example, it is not possible to examine the relationship the original high-dimensional dataset's features have with the resulting low-dimensional map. The features are also called attributes and they each describe one dimension of the high-dimensional datapoint. This missing relationship can make the interpretation of the low-dimensional map more difficult [CM16]. In the example given above (Figure 1.1), we can observe that there are several clusterings within the dataset that are presented by the low-dimensional map. However, we only know that there are these clusters and how they relate, we don't know what makes this grouping a cluster and how that cluster can be understood. If we were provided a way of showing a low-dimensional map with the datapoints depicted in the *context* of the original attributes, the interpretation of the map could be further simplified and improved. We could infer properties about the high-dimensional dataset due to the relationship of the attributes with the datapoints depicted in the low-dimensional map.

This is what the Data Context Map (DCM) algorithm [CM16], realized by the implementation presented in this thesis, is capable of. For a given high-dimensional dataset, it produces a dimensionality-reduced map that shows the datapoints in the context of the attributes. This means that the resulting scatterplot does not only depict the relationship of the datapoints with each other like it would in a normal dimensionality-reduced map (Figure 1.1), but the resulting map also accounts for the relationship of the datapoints with the attributes, and the attributes with each other. The general idea is to augment the low-dimensional plot of datapoints with additional points that each correspond to the attributes. From the location of specific datapoints and specific attribute representation dots, further knowledge about the dataset can be inferred. This gives a map that can help in data exploration processes and datapoint selection tasks. A more detailed explanation of the DCM algorithm, as well as an example, are provided in Chapter 3.

In this thesis, we further present an improvement applied to the low-dimensional map created by DCM. Like the Probing proposed by Stahnke et al. [SDMT16] or the ProxiLens proposed by Heulot et al. [HAF13], this extension allows for an error correction in the low-dimensional map. The correction works by plotting the datapoints (and attribute representations) to match the real high-dimensional distances towards a selected point or a weighted average point within a selected radius in the map. This further improves the interpretability of the low-dimensional map, as the error necessarily occurring due to the dimensionality reduction can be eliminated, at least for a specific point or a small neighborhood of points. A more detailed explanation of the concept of Probing, in whose context the error correction is implemented, can be found in Section 2.4.

The results show that the DCM implementation presented in this thesis is correctly performing the algorithm described by Cheng and Mueller [CM16]. The interpretability of the low-dimensional map has been confirmed by applying the implementation to a selection of real-world datasets and by analyzing them. Further, the error-correcting interaction approach offers an additional possibility to enhance the interpretation of the map.

For the remaining part of this thesis, the implementation of the DCM and the theory behind the algorithm will be explained. Chapter 2 deals with the theoretical foundations involved and provides an extensive introduction to dimensionality reduction techniques. Chapter 3 explains the DCM algorithm in detail. We present the implementation and underlying technologies in the subsequent chapter. Finally, Chapter 5 and Chapter 6 provide the verification of the implementation, maps for a selection of real-world datasets with the aim to better understand these datasets and a conclusion with some ideas, how the implementation could be further improved. For the rest of this chapter, we present the task description for this thesis.

1.1 Task Description

The goal of this thesis is to develop a method that allows showing low-dimensional datapoints (obtained with a dimensionality reduction technique) in the context of the original dataset's attributes. This will further improve the interpretability of the low-dimensional map. For this approach, the DCM algorithm proposed by Cheng and Mueller [CM16] will be used. As the algorithm is not publicly available, the first step is to implement the DCM so that a visualization of high-dimensional datasets can be achieved.

As a low-dimensional map has been obtained with the DCM implementation, the goal is to further improve the map with the interaction techniques (error correction for selected points) mentioned above. For this task, the error correction, point-wise and weighted average neighborhood-wise, needs to be implemented. Especially the weighted average neighborhood-wise point correction offers a more smoothed interaction with the map that helps in better understanding the dataset.

The implementation of the whole algorithm is then applied to a selection of real-world datasets, aiming to improve understanding and interpretability.

2 Related Work

This chapter introduces to the theory needed for the DCM. An overview of dimensionality reduction will be given in Section 2.1, followed by presenting a selection of dimensionality reduction algorithms like Principal Component Analysis (PCA) (Section 2.1.1), Multidimensional Scaling (MDS) (Section 2.1.2) and t-SNE (Section 2.1.3).

Afterwards, we present Adaptive Kernel Density Estimation (AKDE) and Nadaraya–Watson Kernel Regression (NWKR) in Section 2.2 and 2.3. Both are used to estimate a continuous map of values within DCM. The last section introduces to an interactive approach with dimensionality-reduced data called Probing (Section 2.4), that is used within this thesis to augment the DCM.

2.1 Projection Techniques for Dimensionality Reduction

Dimensionality reduction describes the process of reducing the number of variables within a dataset, with the goal to eliminate redundant variables and dependencies [LV07, p.2, 11]. This typically leads to a lower-dimensional embedding of the data that still captures much of the original data's structure. A lower-dimensional representation of the data offers some advantages. For example, we can store data more efficiently, as we removed redundant variables and can still account for most of the data with the remaining variables. But more importantly, dimensionality reduction can help to understand the intrinsic structure of a dataset. For example, taking a plane in the 2D space and twisting it to be shaped like an "S" (it is now represented in 3D), does not change the fact that the original structure is two-dimensional. The resulting 3D object has an intrinsic dimensionality of 2, the twist is not helpful in generally explaining the data and can be ignored. Good dimensionality reduction algorithms will find that the underlying structure of the object is a 2D plane. This small example for dimensionality reduction is called the Swiss Roll (e.g. [TDL00]).

As dimensionality reduction methods are trying to find this intrinsic representation, we can use them to better understand and visualize high-dimensional datasets. To achieve this, a mapping of the high-dimensional data $\{x_i\}_{i=1}^n, x_i \in \mathbb{R}^D$ into a low-dimensional representation $y_i \in \mathbb{R}^P$ is needed, typically $P < D$ and for visualization $P \in \{2, 3\}$. There exist different objectives for the reduction. For example, the MDS class of dimensionality reduction methods seeks a representation in which the distances of the low-dimensional datapoints equal the distances of the high-dimensional datapoints [SVM14].

The following sections explain some important algorithms for dimensionality reduction like PCA (Section 2.1.1), MDS (Section 2.1.2) and t-SNE (Section 2.1.3). One general remark for the notation in the following sections: The observations $\{x_i\}_{i=1}^n, x_i \in \mathbb{R}^D$ are often represented in a matrix $\mathbf{X} \in \mathbb{R}^{nxD}$, with each datapoint being a row vector. The low-dimensional embedding is therefore referred to as $\mathbf{Y} \in \mathbb{R}^{nxP}$, each low-dimensional datapoint $y_i \in \mathbb{R}^P$ being a row vector.

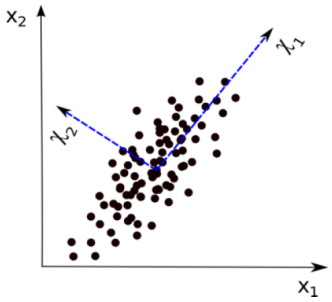


Figure 2.1: PCA applied to a 2D dataset. The blue-marked vectors $\mathcal{X}_1$ and $\mathcal{X}_2$ correspond to the principal components that PCA found, the directions of the highest variance within the data. Taken from [SVM14].

2.1.1 Principal Component Analysis

Principal Component Analysis (PCA) is known as one of the most popular procedures for dimensionality reduction [LV07, p.11]. It is a linear procedure, meaning that the low-dimensional mapping is a linear projection of the datapoints. Due to this, it works best when the high-dimensional data fit a linear subspace $\mathbb{R}^P$ [SVM14]. Originally developed by Pearson [Pea01] in 1901, a variety of improvements and non-linear extensions exist today (e.g. Kernel PCA [SSM98]).

The key idea of PCA is to find a number of P orthogonal vectors, which explain as much of the variance within the dataset as possible [SVM14]. This can be interpreted in a sense that PCA assumes the best low-dimensional embedding to be by the axes of the highest variance. Figure 2.1 shows a simple plot of applying PCA to a two-dimensional dataset. What can be observed is that the two vectors $\mathcal{X}_1$ and $\mathcal{X}_2$, which were found by PCA, correspond to the directions of the first- and second-most variance within the data. These vectors are also called the principal components. A reduction to one dimension would rank the datapoints according to $\mathcal{X}_1$.

For the algorithm, it is assumed that the data has a zero-mean. To make the data matrix $\mathbf{X} \in \mathbb{R}^{nxD}$ has a zero-mean, we set each row vector x_i to be $\bar{x_i} = x_i - \mu$ with $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ being the mean vector of all datapoints.

PCA can be approached by a variety of different criteria, which all lead to the same low-dimensional embedding [LV07, p.26]. Next to minimizing a reconstruction error, which can be reformulated as performing a Singular Value Decomposition (SVD) on the data matrix $\mathbf{X}$, one could try to preserve a maximum of variance. As Lee and Verleysen [LV07, p.28] explain in detail, this criterion leads to performing an Eigenvalue Decomposition (EVD) of the covariance matrix of $\mathbf{X}$. In an equation, given the matrix of high-dimensional datapoints $\mathbf{X} \in \mathbb{R}^{nxD}$, this is:

$$Cov(\mathbf{X}) = \mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{V}^T, \quad \mathbf{D} = diag(\lambda_1, \lambda_2, ..., \lambda_D) \tag{2.1}$$

where $\mathbf{V}$ is the matrix consisting of the eigenvectors $e_i \in \mathbb{R}^D$ with corresponding eigenvalue λ_i. As $Cov(\mathbf{X})$ is symmetric and positive semidefinite [LV07, p.29], the eigenvectors are orthogonal and the eigenvalues are real numbers. For the low-dimensional embedding $\mathbf{Y} \in \mathbb{R}^{nxP}$, we now need to somehow account for the P directions with the highest variance. This is done by defining a submatrix $\mathbf{V}_P$, which consists of the P-largest eigenvectors e_i of $\mathbf{V}$. Large in this regard refers to sorting the eigenvectors decreasingly by the corresponding eigenvalue λ_i.

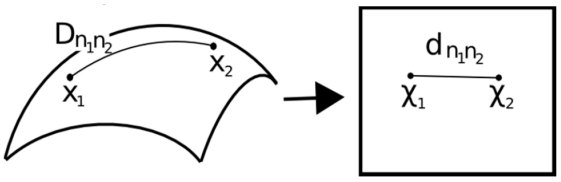


Figure 2.2: The mapping into the lower-dimensional space tries to preserve the distance of the datapoints in the high-dimensional space. Image taken from [SVM14].

Finally, the linear mapping $y_i = V_P^T x_i$ needs to be calculated. It obtains the low-dimensional representation for a high-dimensional datapoint x_i. Optionally, the previously subtracted mean μ can be added to the x_i again. With PCA, it is possible to invert the mapping to get a high-dimensional datapoint from its low-dimensional representation via $x_i = V_P y_i$. As a recent study suggests [RSS18], PCA is a good algorithm for a fast and computationally inexpensive dimensionality reduction. However, due to its linear nature, it is unsuited for more complex and non-linear datasets.

2.1.2 Multidimensional Scaling

The term Multidimensional Scaling (MDS) describes a whole set of so-called distance-preserving dimensionality reduction techniques, rather than being a single specific algorithm [LV07, p.73]. Distance-preserving in this context means that the criterion for the dimensionality reduction is to keep the original, high-dimensional distances in the embedding. Figure 2.2 visualizes this principle. The distance between two points x_1 and x_2 in the high-dimensional space is used to determine the distance of the embedding $\mathcal{X}_1$ and $\mathcal{X}_2$. The mapping tries to best preserve all the point's dissimilarities.

The idea behind this criterion is explained by Lee and Verleysen [LV07, p.69]: "In the ideal case, the preservation of the pairwise distances measured in a dataset ensures that the low-dimensional embedding inherits the main geometric properties of data[...]". An MDS-based low-dimensional embedding is generally good at capturing the global structure of the data, regarding all the point's distances. This is one reason why MDS is popular for exploratory data analysis, as it gives a globally consistent overview of the data [CM16].

In the following subsections, a selection of MDS-type algorithms will be presented. Starting with Metric Multidimensional Scaling (Metric MDS), a more powerful, Graphics Processing Unit (GPU)-based method called Glimmer Multidimensional Scaling (Glimmer MDS) will be explained in Section 2.1.2.

Metric MDS

This section explains the oldest form of Metric Multidimensional Scaling (Metric MDS), also known as Torgerson's Classical Multidimensional Scaling (CMDS) [Tor65]. The term Metric MDS generally comprises more than just this one algorithm [CC00, p.36]. While CMDS specifically assumes Euclidean distances of the input [DM11], other Metric MDS procedures exist that generalize to different loss functions and other distances measures (e.g. the SMACOF algorithm [DM11]).

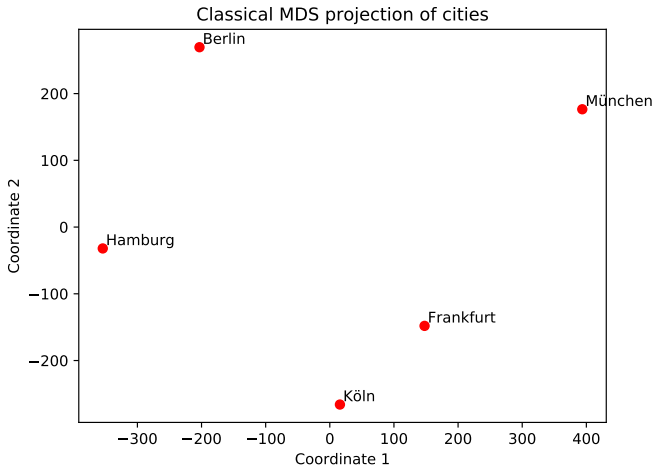


Figure 2.3: Classical MDS projection used to recover the locations of five German cities, given only their pairwise distances.

The CMDS minimizes a loss function called *strain*, that is similar, but not equal to the loss function minimized by the majority of MDS procedures called *stress* [IMO09]. An advantage is that CMDS has an analytical solution, while most of the more modern MDS variants require an iterative optimization technique [Wic03].

For the CMDS algorithm, it is assumed that a distance matrix $\mathbf{D} \in \mathbb{R}^{nxn}$ of the point's pairwise Euclidean distances is given. In other words, the algorithm ensures that in the resulting low-dimensional embedding, all the point's Euclidean distances approximate the Euclidean distances of the high-dimensional points. This can be used for example to create a map based on given distances between points, as shown in Figure 2.3. Given the pairwise distances between a number of German cities, CMDS has been used to recover the cities location on a two-dimensional coordinate system. The resulting map is unique except for rotation and scaling.

To perform CMDS, the distance matrix $\mathbf{D}$ first needs to be squared $\mathbf{D} = [d_{ij}^2]$ for each entry d_{ij} in $\mathbf{D}$. d_{ij} is the measured high-dimensional distance between datapoints x_i and x_j, normally the Euclidean distance. The next step is to *double-center* the distance matrix $\mathbf{D}$. By that, the algorithm ensures that an EVD of the distance matrix can be performed to obtain the low-dimensional coordinates [Wic03]. The double-centering can be achieved with

$$\mathbf{B} = -\frac{1}{2}\mathbf{JDJ} \quad \text{with} \quad \mathbf{J} = \mathbf{I} - \frac{1}{n}1_n 1_n^T \tag{2.2}$$

where $1_n 1_n^T \in \mathbb{R}^{nxn}$ is the matrix consisting of ones and $\mathbf{I} \in \mathbb{R}^{nxn}$ is the identity matrix. As the matrix is now double-centered, we can apply an EVD to find the low-dimensional embedding, $\mathbf{B} = \mathbf{VEV}^T, \quad \mathbf{E} = diag(\lambda_1, \lambda_2, ..., \lambda_D)$ with $\mathbf{V}$ is the matrix consisting of the eigenvectors $e_i \in \mathbb{R}^D$ with corresponding eigenvalue λ_i.

The matrix of the low-dimensional embeddings $\mathbf{Y} \in \mathbb{R}^{nxP}$ can now be calculated as

$$\mathbf{Y} = \mathbf{V}_P \mathbf{E}_P^{1/2} \tag{2.3}$$

where $\mathbf{V}_P$ is the matrix of the eigenvectors corresponding to the P-largest eigenvalues λ_i and $\mathbf{E}_P$ the diagonal matrix of those largest eigenvalues, sorted decreasingly on the diagonal [Wic03].

It can be shown [LV07, p.75] that CMDS and PCA give the same resulting low-dimensional embedding, as both minimize the same criterion. This is true as long as the input to CMDS is a matrix consisting of Euclidean distances [Gho06]. However, the input can be any measure of dissimilarity, only the embedding will depict the dissimilarities as Euclidean distances. This allows to apply PCA, even if the datapoints themselves aren't available, but just the dissimilarities of the data [LV07, p.76]. CMDS is a linear dimensionality reduction method and therefore suffers the same drawbacks as PCA when dealing with more complex, non-linear datasets.

Glimmer MDS

The Glimmer Multidimensional Scaling (Glimmer MDS) algorithm is a powerful approach to MDS. While the analytical solution of CMDS finds the minimum of the so-called *strain* loss-function (Section 2.1.2), Glimmer uses an iterative optimization scheme to find the minimum of the *stress* loss-function. Although similar, the stress function seems to be the slightly better metric to measure the MDS performance. Stress is defined as

$$stress = \sqrt{\frac{\sum_{i,j}(\hat{d}_{ij}^2 - d_{ij}^2)}{\sum_{i,j} d_{ij}^2}} \tag{2.4}$$

where d_{ij}^2 is the squared high-dimensional distance between x_i and x_j and $\hat{d}_{ij}^2$ is the projected squared low-dimensional distance between y_i and y_j [IMO09]. Glimmer MDS is a force-based MDS algorithm; in most cases, these algorithms produce a lower-stress result than CMDS [IMO09].

The class of force-based (also force-directed) MDS algorithms describes an approach to best preserve the pairwise high-dimensional distances with the help of a mass-spring simulation [IMO09]. In such a system, each low-dimensional datapoint is modeled as a particle that is connected to other particles with springs. These springs have an ideal, relaxed length corresponding to the high-dimensional distance d_{ij} between the two connected points. Initialized with randomly located low-dimensional points, the system connected by the springs can be simulated to reach a state of minimal energy. In this state, the springs are maximally relaxed and the resulting low-dimensional embedding can be obtained from the particle's positions. However, as Ingram et al. [IMO09] state, naive implementations of such systems are generally not that common, as they are computationally expensive ($O(n^2)$ for n datapoints per iteration) and are inclined to reach local minima.

This is where the Glimmer MDS algorithm offers improvements. Using the GPU, Glimmer is considerably faster than an implementation based on the Central Processing Unit (CPU). Also, using a linear-time termination condition based on an approximation of stress, the $O(n^2)$ computation cost of the regular stress function can be avoided. A multilevel strategy [IMO09] used by Glimmer MDS makes local minima solutions less likely to occur. Glimmer MDS is based on a stochastic force-directed algorithm proposed by Chalmers [Cha96]. Both algorithms run in $O(n)$ per iteration [IMO09], however, Glimmer exploits the speed gains achieved with parallelism on the GPU and uses an improved termination condition.

(a) Glimmer MDS, stress=0.157

(b) Pivot MDS, stress=0.928

Figure 2.4: Comparing the 2-dimensional embedding of a document meta data dataset with six clusters between Glimmer MDS and the fast, CMDS-approximating Pivot MDS. Taken from [IMO09].

The general idea of a stochastic force-based approach is to reduce the number of forces that apply to each datapoint, caused by the springs. While in the original force-based approach, the forces acting upon a datapoint are based on all other datapoints (modeled by springs), the stochastic approach only regards a constant number of randomly chosen datapoints per iteration. This reduces the computation cost to update a point's position to a constant $O(1)$ and is the main reason for the speed gains of Glimmer MDS and Chandler's algorithm [Cha96] over the basic force-based MDS approach.

To calculate the forces acting on a datapoint y_i, only two fixed-sized sets of other points are regarded. These sets are called the Near set and the Random set [IMO09]. The Random set consists of randomly chosen datapoints (particles) and that set will be newly sampled every iteration. The Near set, although initially also chosen at random, converges to the set of nearest neighbors over time. For each datapoint in the Random set, if it's high-dimensional distance to the current datapoint y_i is smaller than the distance of y_i to any of the points in the Near set, the Near set will be updated. This ensures that for the force calculation for a datapoint, global (Random set) and local (Near set) information will be regarded. The size of the sets can be chosen as e.g. 10 for the Random set and 5 for the Near set, as proposed by Chalmers [Cha96]. Glimmer MDS works by iteratively calculating the forces acting upon each of the datapoints, updating their low-dimensional positions accordingly and checking for the termination criterion. The termination criterion is derived from the so-called *sparse normalized stress* [IMO09], a stress that is faster to calculate than the normal stress (Equation (2.4)). In a recent comparison between different MDS variants and implementations [IMO09], Glimmer MDS outperformed the linear CMDS and its extensions as well as some other force-based MDS algorithms. Figure 2.4 depicts an extraction of this comparison. It shows the results of embedding a 28.374-dimensional, real-world dataset of document meta data [KH05] into two dimensions, using Glimmer MDS (2.4a) and Pivot MDS (2.4b), a fast approximation algorithm for CMDS [BP06]. The colors correspond to the ground-truth for six given clusters. The results show clearly that Glimmer MDS better captures the structure and the

clusters of the dataset, while CMDS hardly finds any clustering of the data that would be considered useful. Due to its powerful dimensionality reduction abilities, Glimmer MDS is used in the DCM algorithm to find the low-dimensional embedding (see also Section 3.2).

2.1.3 t-Distributed Stochastic Neighbor Embedding

Another powerful, non-linear dimensionality reduction algorithm is called t-Distributed Stochastic Neighbor Embedding (t-SNE). We include this algorithm in our work as it is an especially powerful technique for visualization purposes, mapping the high-dimensional data into two- or three-dimensional subspace.

t-SNE, proposed by Maaten and Hinton [MH08], became very popular in recent years due to its ability to deal with difficult, high-dimensional, real-world datasets. A comparison of different dimensionality reduction algorithms showed that the visualizations achieved with t-SNE often outmatch other dimensionality reduction techniques like MDS or even more complex, non-linear techniques [RSS18]. One drawback in comparison to e.g. MDS is that t-SNE doesn't capture the global structure as well as MDS does. However, it is better at isolating individual clusters, retaining more of the local structure of the dataset.

The general objective of t-SNE is to align two probability distributions, one over the high-dimensional space and another one over the low-dimensional target space. In that regard, it is very similar to its predecessor Stochastic Neighbor Embedding (SNE). For the algorithm, we first define the conditional probabilities that are used to represent the similarities of the high-dimensional datapoints. The probabilities are obtained from the high-dimensional Euclidean distances, however, any other distance metric could be used [MH08].

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}, \quad p_{i|i} = 0 \tag{2.5}$$

where x_i and x_j are two high-dimensional datapoints. Variance σ_i can be obtained by means of a binary search, including a hyperparameter "perplexity". The details would exceed the depth of this section and can be found in [MH08]. This conditional probability can be interpreted in how likely datapoint x_i would choose datapoint x_j as a neighbor. This likelihood is proportional to a Gaussian distribution with variance σ_i centered at the datapoint x_i. The probability $p_{j|i}$ increases, the closer x_j is towards x_i and is almost zero for high-dimensional points far apart.

Secondly, a probability distribution over the low-dimensional points $\{y_i\}_{i=1}^n$ will be defined. The idea is that if the distribution over the low-dimensional points can be aligned to match the distribution over the high-dimensional points (Equation (2.5)), a pretty good dimensionality reduction is achieved. This is where the most advancements of t-SNE over its predecessor SNE come into play. In order to simplify the optimization of the cost function in SNE, t-SNE uses a cost function calculated by the joint probability distributions over the high- and low-dimensional spaces, instead of the conditional probability distributions as given by Equation (2.5) for the high-dimensional part [MH08]. Therefore we set the joint probability $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$, which symmetries the conditional probabilities presented above.

The joint probability for the low-dimensional space is given by a Student t-distribution with one degree of freedom [MH08]. It can be computed using

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{2.6}$$

This gives the affinities for the low-dimensional representations. With the joint probabilities defined, the gradient can now be calculated. The gradient is used within an iterative optimization scheme to find the optimal positioning for the low-dimensional points with respect to the cost function given by:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log(\frac{p_{ij}}{q_{ij}}) \tag{2.7}$$

where $KL(P||Q)$ denotes the Kullback-Leibler divergence between the high- and low-dimensional probability distributions. It functions as a measurement of the difference of both probability distributions. What is now searched for is a minimization of the cost function with respect to the low-dimensional points, as then the low-dimensional distribution best approximates the high-dimensional distribution. The gradient is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{2.8}$$

It is used within a gradient descent procedure to retrieve the low-dimensional representations. The following equation can be iterated over until the changes to the low-dimensional dataset $Y = \{y_1, y_2, ..., y_n\}$ are negligible.

$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}) \tag{2.9}$$

where $Y^{(t)}$ denotes the low-dimensional dataset at the time t. η and $\alpha(t)$ are hyperparameters, indicating the learning rate (η) and the gradient descent's momentum ($\alpha(t)$). The initialization $Y^{(0)}$ can be sampled from a small-variance Gaussian distribution $\mathcal{N}(0, 10^{-4}I)$. Performing this algorithm achieves a powerful dimensionality reduction, especially useful for visualizations.

2.2 Adaptive Kernel Density Estimation

Density estimation describes a set of statistical methods that try to estimate the Probability Density Function (PDF) from a set of observed datapoints. One very simple approach is to create a histogram of the observed data to estimate the PDF. However, this approach may not be sufficient, as a histogram doesn't allow for a continuous density estimate [Sil18].

Kernel Density Estimation (KDE), first introduced by Parzen [Par62], is a method to estimate a continuous, smoothed PDF. Nonparametric approaches like KDE have the advantage that they don't presume a distribution of the datapoints [Sil18]. This makes them better suited for data for which one does not know the distribution beforehand, as it often occurs with real-world datasets. The estimated density $\hat{f}(x)$ for an unknown datapoint x, estimated with KDE, looks like the following:

$$\hat{f_h}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - X_i) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - X_i}{h}) \tag{2.10}$$

where $\{X_i\}_{i=1}^n$ denotes the observations and $K_h(u) = \frac{1}{h}K(\frac{u}{h})$ denotes a kernel function with bandwidth h.

In statistics, a kernel is a non-negative, real-valued function. For a given value x, it is used to weight distances from that value, where the higher the distance between the parameter and x, the lower the weight [HTF09, p.209]. A variety of kernels exist; in the simplest case, a kernel can be the uniform kernel, a rectangular window given by $K(u) = \frac{1}{2}, -1 \leq u \leq 1$. A commonly used kernel is the Gaussian Kernel $K(u) = \frac{1}{\sqrt{2\pi}}exp(-\frac{u^2}{2})$, which is inspired by the bell curve. For all kernels to be used in KDE, we must have that $\int_{-\infty}^{\infty} K(u)du = 1$ such that $\hat{f}_h$ will be a proper PDF [Sil18].

The bandwidth parameter h is an important choice for the kernel. It specifies the width of the kernel function, thereby determining how strong the neighborhood of a given point x is weighted. The higher h, the wider the kernel function. If we choose a high h, a wider range of the neighborhood of x will be regarded. This smooths the PDF, but may also lead to a bias in the estimation that hides important features [Sil18]. If h is chosen to be lower, we may overcome the bias with a more fine-grained, local estimation of the density function, but overfitting due to high variance is more likely to occur [Sil18].

Although carefully chosen, a fixed bandwidth h can still lead to an imprecise estimation. This is due to the density of the data itself. In sparse regions with only a few observations, a globally fixed h tends to support overfitting, as h is typically to low for a good smoothing of the neighborhood [Van+03]. The opposite occurs in dense regions where h is more likely to cause a bias because too many neighbors are highly weighted and regarded.

One approach to solve this issue is to use Adaptive Kernel Density Estimation (AKDE). These density estimation methods have a varying bandwidth for each sample point, which makes them better adapt to sparseness in the data [Van+03]. The AKDE method proposed by Van Kerm et al. [Van+03] uses a two-stage estimation technique. First, a globally fixed h is used to estimate the local bandwidths h_i for each point. $h_i = \lambda_i h$, where λ_i is the weight that shrinks or enlarges the global bandwidth. These weights are obtained as $\lambda_i = (G/\hat{f}_h(X_i))^{0.5}$ with $\{X_i\}_{i=1}^n$ being the data, $\hat{f}_h(X_i)$ as in Equation (2.10) and G being the geometric mean of all $\hat{f}_h(X_i)$ [Van+03].

After obtaining the local bandwidths h_i, the adaptive kernel density estimate for a new point x is given by

$$\tilde{f}(x) = \frac{1}{nh_i}\sum_{i=1}^{n} K(\frac{x - X_i}{h_i}) \tag{2.11}$$

AKDE is used as part of DCM for the value region estimation. Refer to Section 3.3 to see more on how it is used for this purpose.

2.3 Nadaraya–Watson Kernel Regression

Given two random variables X and Y, the objective of regression is to estimate the regression function $f(x) = E(Y|X = x)$ [HTF09; Wat64]. That is, for a (probably unseen) x, what is the corresponding value of y? In other words, regression methods try to find the relationship between X and Y.

Kernel Regression provides a non-linear, smoothed estimation $\hat{f}(x)$. Using kernels (see Section 2.2), a special weighted, local-neighborhood based estimate of a point's value can be made [HTF09, p.192]. As with AKDE (Section 2.2), the kernels must be specified with a bandwidth h, which determines the degree to which points close to x are weighted for the estimate. The choice of bandwidth is a crucial factor for the overall smoothing of the regression function [HTF09, p.193]. The same trade-off between variance and bias as described in Section 2.2 occurs.

Nadaraya–Watson Kernel Regression (NWKR) has been developed by Nadaraya [Nad64] and Watson [Wat64] and is a special form of Kernel Regression. Given a set of observations $\{(x_i, y_i)\}_{i=1}^n$, NWKR estimates a new x's value as

$$\hat{f}_h(x) = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{i=1}^n K_h(x - x_i)} = \frac{\sum_{i=1}^n K(\frac{x-x_i}{h}) y_i}{\sum_{i=1}^n K(\frac{x-x_i}{h})} \tag{2.12}$$

where K is any kernel function (see Section 2.2 for an example) and h the corresponding kernel width. The NWKR estimate of a point can be seen as a weighted average of the y_i's, with the weights decreasing with distance to the chosen point [Wat64]. An advancement that adapts the overall bandwidth h to each point to get more fine-grained, local bandwidths h_i, just as with AKDE, is also possible.

2.4 Probing Projections

The human brain is good at performing spatial reasoning of different objects perceived with vision. This capability is one of the main reasons to perform information visualization, e.g. with the help of dimensionality reduction techniques. If the human brain is good in intuitively understanding spatial relationships, groupings of objects and their separations, the idea is that a projection of high-dimensional data into two or three dimensions, achieved with dimensionality reduction, heavily helps in interpreting and analyzing the data [SDMT16].

Still, dimensionality-reduced data has two problems regarding the visualization. At first, it is not clear how exactly the positions of the low-dimensional datapoints should be interpreted. With PCA for example, one can conclude that the first axis of the resulting plot corresponds to the direction of the highest variance within the data; the second axis of the plot can be interpreted likewise. However, switching to another dimensionality reduction technique like Glimmer MDS or t-SNE, the meaning of the resulting X and Y axes of the plot is obscured and can not be understood easily. Clusters and the overall structure of the data can be seen, but it is not clear, for most dimensionality reduction techniques, what the axes of the resulting low-dimensional plot tell.

Another problem as stated by Stahnke et al. [SDMT16] is that the errors of the projection are rarely shown to the user. An error is naturally occurring for most projections due to fitting the high-dimensional data into a lower-dimensional subspace that can possibly not capture all of the high-dimensional space's variance. Although the optimization procedures of the algorithms are more and more improving to increase the projection accuracy, the errors of the low-dimensional embedding may be significant [SDMT16]. A simple scatterplot visualization of low-dimensional datapoints for example, does not help in examining the quality of the visualization in regard to errors of the projections.

Figure 2.5: The visualization shows the result of applying a Metric MDS scheme to a high-dimensional dataset. Selecting a datapoint (orange) corrects the low-dimensional distances to each other point to match the true high-dimensional distance. The lines show the amount of movement needed. Image taken from [SDMT16].

Stahnke et al. [SDMT16] propose the concept of Probing in an attempt to help data analysts and people dealing with dimensionality-reduced data for visualization purposes to better manage the two problems mentioned above. Probing refers to an interaction approach with visualizations that helps in both exploring the data and examining the quality of the representation [SDMT16]. If a visualization environment offers help for both activities, confidence can be gained in interpreting the low-dimensional embedding and by that in analyzing the original, high-dimensional dataset.

As mentioned previously, the low-dimensional embedding typically involves an error as to the exact representation of the high-dimensional data. A scatterplot of dimensionality-reduced data, e.g. two-dimensional, normally tries to locate the points so that some distance metric between the points is best-approximated. In the case of MDS for example, the stress value (see Section 2.1.2) is often used as a global error measure. In the simplest case, one would just show the raw stress measure along with the representation. However, being only shown the stress value makes it hard to relate the error to the actual visualization in the scatterplot. Approaches to visualize the errors of the embedding exist (e.g in a Shepard diagram [BG05]), but they focus on showing the error about the projection. What is needed for a more truthful and confident interpretation of the low-dimensional dataset however is that the error can be shown inside the visualization itself. By doing so, the data analyst can make more sense of how the error influences the embedding. This is one main idea of Probing [SDMT16]. Including the error of the embedding in the visualization can e.g. be achieved by showing the error for each datapoint. As Probing suggests, a datapoint must interactively be chosen from the scatterplot visualization. For one given datapoint, the positions of all other datapoints can be corrected to match the exact high-dimensional distance to the chosen point. An example is shown in Figure 2.5. After selecting the low-dimensional datapoint marked in orange, the positions of all other datapoints are corrected so that the resulting distance to the orange point corresponds to the high-dimensional distance. The grey and white lines show how much each

point needed to be moved. A longer line indicates that the representation error for that point was higher, as the difference between the low-dimensional distance and the high-dimensional distance is greater. However, it needs to be noted that this error correction only applies to one point. After the correction, the positions of the other points to each other are less informative than before. Only the relation of the chosen point to the other points can be better understood [SDMT16].

This kind of interaction with the low-dimensional embedding offers the possibility to better see how the error influences the positioning of a point. Probing allows interpreting the low-dimensional representation more confidently. The data analyst can account for errors in the embedding in a more intuitive, accessible way and therefore better analyze the underlying, high-dimensional dataset.

Next to the error correction mentioned above, the visualization environment presented by Stahnke et al. [SDMT16] offers some other features to better interpret the meaning and examine the quality of a low-dimensional representation. These features include clustering, examining the high-dimensional values, comparing values of single points and selections and accessing the distribution of the single attributes. However, for the DCM implementation, the error correction is of higher importance. Further details regarding the other features of Probing can be found in [SDMT16].

3 Data Context Map

Dimensionality reduction algorithms like the ones proposed in Section 2.1 allow creating a visualization, e.g. two-dimensional, of a given high-dimensional dataset. These algorithms take a set of data points and use each point's attributes to create a meaningful, lower-dimensional dataset. The attributes are also called features. Typically, this dataset is then shown as dots in a scatterplot where each dot corresponds to the projection of a single high-dimensional datapoint. While this procedure generally works well to discover clusters and hidden structures within the dataset, it is not possible to determine what relation the attributes of the dataset have with the visualization. There is no *context* of how the attributes relate to the low-dimensional representation of the data. The Data Context Map (DCM) proposed by Cheng and Mueller [CM16] describes an algorithm that allows a context-specific visualization of these attributes.

Visualizations that show the relationship of the attributes with the datapoints already exist. There are for example parallel coordinates plots [Weg90] and star plots, which both show each datapoint as a so-called polyline, crossing the attribute representations at different heights to show their corresponding value. There is also the star coordinates plot [Kan00], that uses 2D points to prevent a clutter of polylines, for example when the dataset is large enough. Radial visualization (Radviz) [HGM+97] also shows the data as 2D points in a radial fashion, pulled towards attributes for which it has a high value. However, all these methods are preserving the relations among the attributes and the datapoints, but not the relations among the datapoints themselves and the attributes themselves simultaneously [CM16]. This is what the DCM proposed in this chapter is capable of.

Figure 3.1 shows an example of the DCM based on a set of universities. The blue dots correspond to the low-dimensional representation of the universities. Each university has been described with 14 attributes (population, tuition, nightlife, housing, etc.), creating a 14-dimensional dataset for the input. What makes DCM now distinct in comparison to other high-dimensional data visualizations is that the 14 attributes are regarded as well. The red dots labeled with the corresponding dimension show how the attributes of the dataset correlate to the datapoints. The interpretation is as follows:

1. Attributes that are close to each other are highly correlated [CM16]. This has to be understood in a sense that correlated features often bring very similar information to the model, therefore these attributes are similar in the context of the datapoints. An example with the given map would be the tuple (tuition, academic). Both attributes are fairly close to each other. Intuitively, a university with a better academic education has a higher tuition cost. This correlation can be seen in the closeness of those two attributes.

2. The position of the datapoints does now not only depict clusterings and groupings of the high-dimensional data like it would in a normal visualization of dimensionality-reduced data, but it also shows how the values of the datapoints relate to the attributes. The closer a datapoint comes towards an attribute, the higher the datapoint's value for that attribute [CM16]. This allows to fairly easy select observations that have a high value in a specific attribute, without

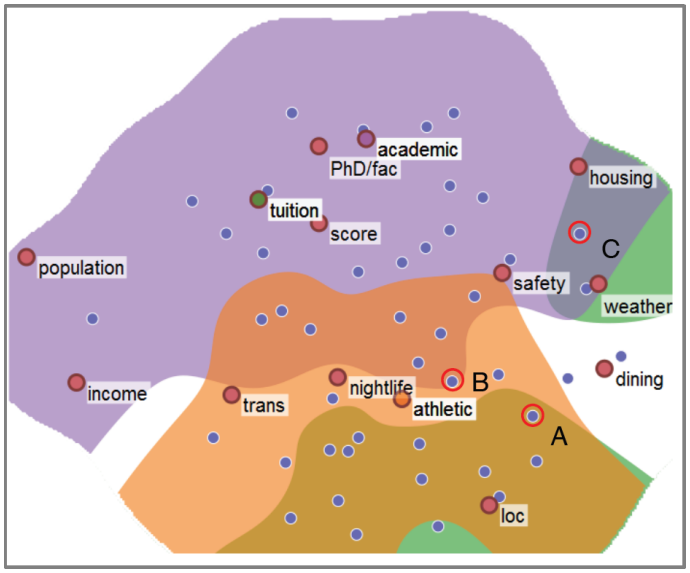


Figure 3.1: A 14-dimensional university dataset visualized with DCM. The blue dots correspond to the universities and the red dots correspond to the attributes of the dataset. Three partly overlapping spatial regions help in estimating the values for the selected attributes academic, tuition and athletic, respectively [CM16].

looking at the original high-dimensional dataset. Take for example the red-circled, selected point B. This low-dimensional point is very close to the attribute "athletic", therefore this university must have a high value in its athletic rating.

3. Clusters of datapoints can be made more sense of. Although there are even better datasets to find meaningful clusters (see for example Section 5.2.1), the idea can become quite clear by reference to Figure 3.1. Take the universities in the upper middle part of the map, mostly surrounded by the purple area. While a normal clustering would probably find that there are some "more prestigious" universities than others and would put them into a cluster, an analyst would very likely not know where that cluster can be found in the map. With the relationship to the attributes however, the meaning and position of a cluster can be fairly good understood. The universities in the upper middle part most likely are the more top-ranked universities, as they have the highest score, academic rating, and tuition cost. This grouping must therefore correspond to the elite universities of the dataset. So the DCM helps in finding not only that there is a cluster of datapoints, but also what makes this grouping a cluster and how that cluster can be interpreted.

Additionally, the map has been overlayed with colored areas like for example the big purple area in the upper half. These areas are defined by three preselected attributes, namely tuition (green), academic (purple) and athletic (orange). The areas show an estimation of the range of the corresponding attribute values. The green area for example (and it's blue and yellow overlap with the other areas) is modeled to include all universities that have a tuition cost of less than $18.000 US-Dollar [CM16]. This threshold is manually chosen. Setting another threshold for this attribute may influence the green area to shrink or grow in size, respectively. The other two areas are chosen with a corresponding threshold as well.

This segmentation of the map, together with the possible interpretations of the map presented above, allows for a better and more comprehensive understanding of the high-dimensional dataset. Professional data analysts may better analyze the dataset, as the map also accounts for the relationship of the attributes and the relationship of the attributes with the datapoints. Non-expert people like people casually looking for universities are provided help in complex decision-making. Not only can they better understand the low-dimensional representation due to the presence of attributes, but the iso-contouring segmentation of the map (colored areas) also help in easily identifying trade-offs [CM16].

The next Section 3.1 explains the construction of the Composite Distance Matrix (CM), the distance matrix for DCM that works as an input to a distance-preserving dimensionality reduction algorithm. Section 3.2 shows how that CM matrix is used to obtain the DCM, completed by Section 3.3, in which we present how the iso-contouring of the map is achieved.

3.1 The Composite Distance Matrix

The key to obtaining the DCM is to create an adequate mapping from the given matrix of high-dimensional datapoints $\mathbf{X} \in \mathbb{R}^{nxD}$ into a representation as depicted in Figure 3.1. n is the number of datapoints and D is the datapoint's dimensionality. This mapping is achieved by creating an appropriate distance matrix (CM) and feeding it into an applicative dimensionality reduction algorithm like an MDS scheme (Section 2.1.2). It is assumed that the data matrix $\mathbf{X}$ is normalized to [0, 1].

The DCM captures the relations between the datapoints, the attributes, and the attributes in the context of the datapoints. This means that the distance matrix CM needs to account for all three relationships. We achieve this by creating different single distance matrices, which we (non-trivially) fuse into one distance matrix CM. Fusing the matrices is non-trivial; the map cannot be created by just overlaying a datapoint-distance plot and an attribute-distance plot, as shown by Cheng and Mueller [CM16].

Obtaining the submatrices

Figure 3.2 shows the general construction of the CM matrix. It consists of the four submatrices **DD**, **DV**, **VD** and **VV**, whose creation we will discuss in the following paragraphs. n and D show the expansion of the matrix, giving clues about the size of the different submatrices, respectively.

DD is the classical distance matrix created from the dataset. Given the data matrix $\mathbf{X}$, it is the matrix $\in \mathbb{R}^{nxn}$ containing the pairwise distances between the datapoints. For each different type of relationship (datapoints, attributes, datapoints and attributes mutually), a different distance metric can be used [CM16]. However, for most cases, it is sufficient to use the Euclidean distance for this matrix. For two given datapoints $x, y \in \mathbb{R}^D$, the Euclidean distance, also called L_2 norm of the difference, is given by:

$$d(x,y) = ||x - y||_2 = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2} \tag{3.1}$$

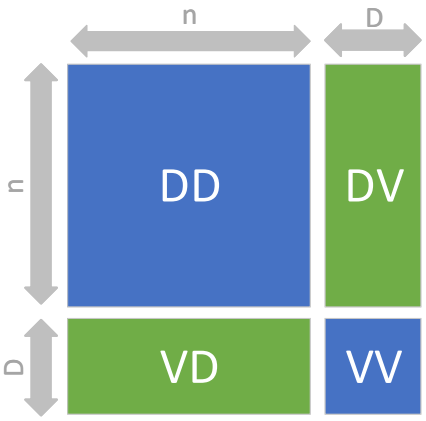


Figure 3.2: The four submatrices which the overall distance matrix CM exists of. CM has an extension of $(n + D)$ x $(n + D)$. Inspired by [CM16].

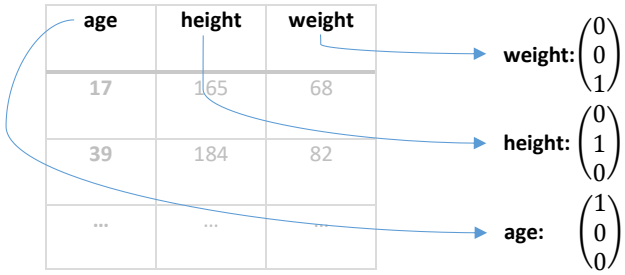


Figure 3.3: Three attributes "age", "height" and "weight" of an example dataset are turned into a 3D datapoint representation, allowing to calculate distance measures between the attributes and the datapoints of the dataset.

VV contains the pairwise dissimilarities of the attributes. There are D attributes given with the high-dimensional dataset. As attributes we take the column vectors $\{v_i\}_{i=1}^{D}, v_i \in \mathbb{R}^n$ of the data matrix $\mathbf{X}$. While the Euclidean distance measure could be used, it is more convenient to use for example the distance derived from the Pearson Correlation Coefficient [BCHC09] as a metric.

DV and **VD** are the matrices that make the whole derivation of CM special. These matrices provide the dissimilarity measures that allow the map to link the datapoint's location with the attributes location to create the comprehensive map that sets both entities into relation. The trick to creating a distance measure between the datapoints and the attributes is to regard the attributes as datapoints, as proposed by Cheng and Mueller [CM16]. They argue that each attribute axis of the dataset can be seen as a vector that consists of zeros, except for the dimension that this attribute corresponds to, set to one. Figure 3.3 visualizes this process for a simple, three-dimensional dataset, creating three vectors of unit length that can be used to calculate the distance measures with the dataset.

With this representation, any distance metric like Euclidean, Pearson or Cosine can be used to create the n x D distance matrix **DV**. Calculating the variable-to-data distance matrix **VD** can be achieved in a similar way. This time, the attribute axes are not the ones naturally proposed by the dataset, but

the n dimensions occurring by the data itself. The D x n matrix **VD** consists of the dissimilarities between the D column vectors $\in \mathbb{R}^n$ of $\mathbf{X}$ and the n unit vectors $\in \mathbb{R}^n$, created by regarding the n datapoints as attribute axes.

It needs to be made clear that for CM to be a proper symmetric distance matrix, **DV** and **VD** must be each other's transpose. This is only the case when using a metric called (1-value) [CM16]. However, when using for example Euclidean- or Correlation distance, the matrix (**DV** or **VD**) with the higher matrix norm will be chosen and the other one will be obtained by transposing it [CM16].

Correcting the transformation

The CM matrix obtained by the four submatrices (Figure 3.2) needs to be further transformed to be used in DCM. This is due to the fact that 1) the vectors used had unequal lengths (n and D) and 2) different distance metrics may have been applied [CM16]. The resulting map could be distorted [CM16]. This is why further transformation is needed. Cheng and Mueller [CM16] propose to balance the difference between the four submatrices by making them have an equal mean. A linear adjustment f preserves the distribution and topology of the data [CM16] and is therefore preferred. $\overline{\mathbf{M}}$ denotes the mean value of a matrix $\mathbf{M}$, where $\mathbf{M} \in \{\mathbf{DD}, \mathbf{DV}, \mathbf{VD}, \mathbf{VV}\}$

$$f(\mathbf{M}) = \mathbf{M}\frac{max(\overline{\mathbf{DD}}, \overline{\mathbf{DV}}, \overline{\mathbf{VD}}, \overline{\mathbf{VV}})}{\overline{\mathbf{M}}} \tag{3.2}$$

needs to be calculated for each submatrix. The scalar with which $\mathbf{M}$ is multiplied can be seen as a weight, making all matrices have the same (maximal) mean. The resulting matrices can be used to build the final CM matrix.

3.2 Creating the Data Context Map

As the correct distance matrix CM has been obtained, a distance-preserving dimensionality reduction scheme can be used to find the DCM. Although there exists a variety of applicable, distance-preserving algorithms, Cheng and Mueller [CM16] propose to use the iterative Glimmer MDS (see Section 2.1.2). There are two main reasons for this. The first reason is that the simpler, linear approaches like PCA and CMDS are often not able to deal with non-linear datasets that naturally occur in the real world. Secondly, an MDS approach is preserving the global structure of the dataset, as opposed to other techniques like t-SNE, the Locally-Linear Embedding family [RSS18] or further dimensionality reduction techniques. A globally consistent view of the dataset can be especially useful for visualization purposes.

Furthermore, the iterative nature of the stochastic force-directed MDS algorithms like Glimmer MDS allows more control in the optimization procedure. DCM uses this iterative nature to define different update schedules for the calculation of the low-dimensional map. While the original way (and still the most common way) is to apply Glimmer MDS *once* to the given distance matrix CM, different ways to obtain the low-dimensional representation can be used. Cheng and Mueller [CM16] altered the Glimmer MDS framework to define which points (datapoints or attributes) are allowed to be moved in the iterative procedure and which points are allowed to be in the Random set (the set that influences the position of the points that move). With these alterations, the low-dimensional

map can be influenced. For example, Glimmer MDS can be run twice, with only the datapoints used as input in the first run, and only the attributes being allowed to move in the second run. For this, the datapoints must be allowed in the Random set in the second run. Using this scheme instead of the original way achieves a low-dimensional representation, in which the error of the datapoint representations is lower than in the original DCM [CM16]. So in short, the different schemes enabled by the alteration of Glimmer MDS give a mapping that either better preserves the attribute relations, the datapoint relations, or the overall relations equally (the original way). For all DCM representations in this thesis, the equally-preserving scheme has been used (apply Glimmer MDS once to the whole CM matrix). However, these schemes give more freedom in designing the low-dimensional representation of the DCM.

3.3 Segmentation of the Map

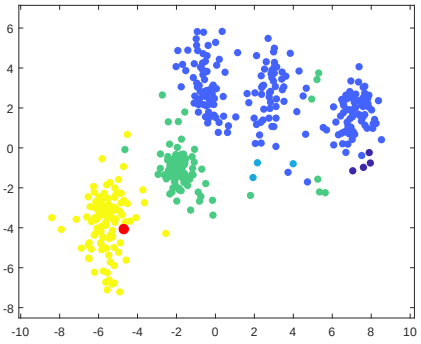
With the last step presented in Section 3.2, the complete two-dimensional representation of the points of the DCM can be obtained. What is missing for the complete map as presented in Figure 3.1 is the colored iso-contouring. This section explains how the colored regions, the segmentation of the map, can be calculated.

The idea behind these regions is that the selection of specific datapoints can further be simplified. With only the 2D point representation of the map, particular features of the datapoints can be filtered for by looking at datapoints close to the corresponding attribute, as described in more detail in Chapter 3. However, the closeness of a datapoint tells much more about that feature value than the distance to the other attributes [CM16]; this is partly due to the error necessarily occuring in the dimensionality reduction step. This means that the low-dimensional closeness is a much better metric than the distance and is a reason why it is more difficult to filter the datapoints for a combination of attributes. The iso-contouring of the map solves this problem by providing colored regions for specified value ranges. If a point locates within such a region, the user can conclude that its value for that feature must be within the specified range.

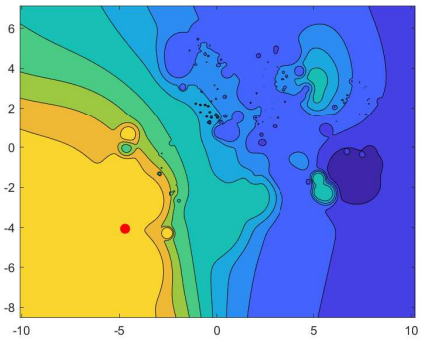
To create the colored areas, a continuous estimation of each of the dataset's attributes is needed. Then, different ranges can be easily specified for each attribute, creating the contour fields. Cheng and Mueller [CM16] propose to use a combination of AKDE (Section 2.2) and NWKR (Section 2.3) for the estimation. At first, AKDE is used to estimate the density of the dataset with adaptive kernels. While the theoretical explanation in Section 2.2 explains the 1D case for AKDE, we need a 2D density estimation now. This is achieved by using the spatial distance (Euclidean distance, $||x - X_i||_2$) as the difference $x - X_i$ for the equations in Section 2.2 [CM16]. The same applies to NWKR (Section 2.3). As far as DCM is concerned, a Gaussian Kernel has been used [CM16]. As the adaptive kernel bandwidths for each datapoint have been obtained with AKDE, these bandwidths are used for the NWKR algorithm to estimate the values for each attribute over the 2D space of the map. What we get is a continuous estimation for every attribute of the dataset.

Figure 3.4 shows an example, generated from the AutoMPG[1] dataset. The procedure described above was used to estimate the attribute "cylinders", which is a multi-valued discrete variable. The left image 3.4a shows the low-dimensional representation obtained by applying DCM. The datapoints

[1] https://archive.ics.uci.edu/ml/datasets/auto+mpg



(a) DCM plotting of the AutoMPG dataset, with the attribute "cylinders" highlighted.



(b) Estimation for the attribute "cylinders" based on the low-dimensional representation.

Figure 3.4: Estimating the attribute "cylinders" of the AutoMPG dataset with AKDE and NWKR. The left image shows the real values for that attribute, for each datapoint. The right image shows the estimation over a 256 x 256 grid. Yellow indicates a high value for the cylinders, blue a low value.

are colored corresponding to the real value of that attribute, with the attribute representation as the red dot. As can be noted, the clustering is fairly good. Points close to the red dot have a high value for the "cylinders" attribute, meaning they represent cars with many cylinders. What can be observed by comparing the left figure to the estimation of the attribute on the right is that the values are fairly good approximated. The red dot is in the center of a yellow contour, visualizing the estimation for a high number of cylinders. This is consistent with the actual representation of the dataset. Also, the estimation for lower values is fairly accurate. While NWKR estimates continuous values, the representation 3.4b has been discretized to show a fixed number of contours. The estimation has been made over a 256x256 grid over the 2D space shown in Figure 3.4a.

For the estimation, a threshold ϵ must be chosen. In the NWKR algorithm, only points that comply with this ϵ will be regarded, in a sense that

$$\sum_{i=1}^{n} K_h(x - x_i) \geq \epsilon \tag{3.3}$$

must hold [CM16]. We set a threshold (for the sum distance) for the denominator in Equation (2.12). All points that are to be estimated and don't comply with the constraint will be ignored. This ensures that points far away from the datapoints are not regarded. The estimated values can now be used to plot the actual colored areas as depicted in Figure 3.1.

Referring to Figure 3.4, a use case might be that someone wants to plot an area that includes all cars which have a value between 6 and 8 for the attribute "cylinders". In that case, the contour region would be created by taking the area that includes all estimates with values between 6 and 8 and plotting it. Due to the property of DCM, to arrange the datapoints with their value decreasing with distance to the corresponding attribute, the region will generally be a single, cohesive area. Further important to note is that because the estimation of the value region has been made from the low-dimensional DCM point representation, any selection will be accurate [CM16]. With the regions plotted into the low-dimensional representation, the DCM construction is now completed.

4 Implementation

This chapter explains the implementation of the DCM and the Probing presented in this thesis. A summary and an overview of the different modules is provided in Section 4.1, followed by a more detailed explanation of the important software components in the subsequent sections.

4.1 Application Components

This section introduces the different components of the implementation and explains the overall system architecture. The implementation of the DCM algorithm described in this thesis is made up of a selection of different programming languages and frameworks, rather than being written in one language specifically. There are various reasons for that. The first reason is that some of the sub-algorithms that are used within the DCM are only available in a specific programming language. Glimmer MDS [IMO09] for example is solely available in a Visual C++ implementation[1]. The AKDE package [Van+03] used by Cheng and Mueller [CM16] is part of the proprietary software package Stata[2]. A technology stack of different languages needed to be made up to not having to implement each sub-algorithm separately and to not rely on expensive, proprietary software.

Figure 4.1 gives an overview of the different modules involved in the DCM implementation. The UML component diagram shows how the general architecture is set up and how the modules are communicating with each other. What can be seen is that the main part of the software, named Data Context Map, is written in Visual C#. Its setup is explained in the following Section 4.1.1 in more detail. This part of the implementation is the most important part, as it determines the executable software and handles all the crucial transformations and calls to retrieve the DCM. However, as mentioned above, not all algorithms used within DCM are available in C#. Due to Glimmer MDS being only available in C++, the main component uses a CSV-file based communication to apply Glimmer MDS to the CM matrix calculated by DCM. Also, AKDE and NWKR are not available in C#. This is why the calculation of the contour regions has been moved to MATLAB, for which a free script is available[3]. All three components will be explained more deeply in the following subsections.

4.1.1 DCM Main Component

This is the main component of the whole implementation. This component creates the DCM and provides the connection that joins the different algorithmic parts. Assigned tasks are:

[1] https://www.cs.ubc.ca/labs/imager/tr/2008/glimmer/

[2] https://www.stata.com/

[3] https://de.mathworks.com/matlabcentral/fileexchange/58312-kernel-density-estimator-for-high-dimensions

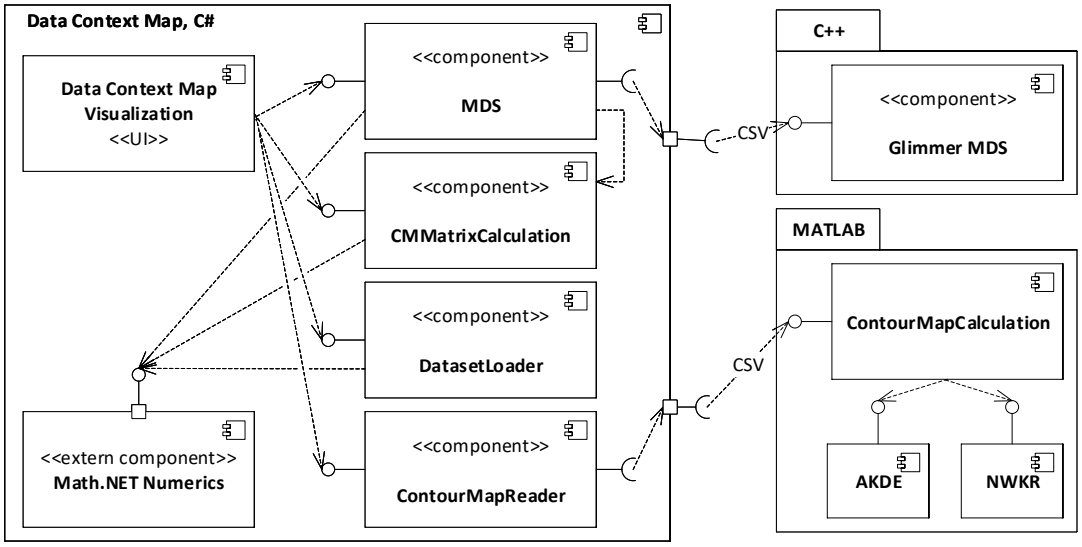


Figure 4.1: An UML Component Diagram showing the overall architecture of the implementation. Two subsystems of the DCM are implemented in MATLAB and C++, respectively. The main component is written in Visual C#.

- **DatasetLoader**: Loading different datasets for which a DCM visualization will be shown.
- **ContourMapReader**: Reading the contour maps calculated with the MATLAB component (Section 4.1.3). The ContourMapReader transforms the CSV-files created by the calculation in a format readable by the main component.
- **CMMatrixCalculation**: Creating the CM matrix based on a given dataset. This is a key part of the DCM algorithm. The CM matrix is further used to obtain the low-dimensional embedding.
- **DimensionalityReduction**: Using a dimensionality reduction scheme to retrieve the 2D representation used for the visualization and the Contour Map Calculation. This component calls the Glimmer MDS implementation further explained in 4.1.2. A generic scheme allows exchanging the used dimensionality reduction algorithm.
- **Probing**: The error correction for specific points, called Probing (Section 2.4), has been implemented in this component. The feature of Probing implemented here is explained in Section 4.2 more deeply, as it is an extension of the concept proposed by Stahnke et al. [SDMT16].
- **Visualization**: This part of the software concerns the visualization of the DCM instance. A 2D scatterplot is provided, showing the contour fields and datapoint/attribute representations. The visualization also accounts for selecting specific points for the Probing functionality and for showing the error corrected low-dimensional embedding.

All the different parts of this component rely on the Math.NET Numerics library[4] for matrix calculations and representations. The DCM Main Component provides the code that shows the DCM visualization as presented by Cheng and Mueller [CM16].

4.1.2 Glimmer MDS Component

Glimmer MDS is only available as a C++ project. In their code[1], Ingram et al. [IMO09] provide two different implementations, one being a CPU-running, Visual C++ variant and the other one being an OpenGL-based GPU implementation. In our code, we use the CPU variant, due to compatibility problems with the GPU implementation. However, changes to the code needed to be made to allow the CPU version to take dissimilarity matrices as input. This is a crucial feature, as the CM matrix produced by DCM consists only of the pairwise distances of the datapoints and attributes.

4.1.3 Contour Map Calculation Component

The calculation of the contour regions, namely performed by AKDE and NWKR, has been implemented in MATLAB. Cheng and Mueller [CM16] use an AKDE implementation based on the statistical software Stata[2], which is commercial software. Due to this, we chose a freely available MATLAB implementation[3] as a basis. The NWKR algorithm has been manually written in MATLAB, based on the weights calculated with AKDE. This component communicates with the C# main component via CSV-files, which are used to exchange the calculated matrices. The contour map calculation takes the original high-dimensional dataset and the low-dimensional point representation and outputs, for each attribute in the high-dimensional dataset, a 256x256 grid sized matrix of estimated values.

4.2 Probing Projections Implementation

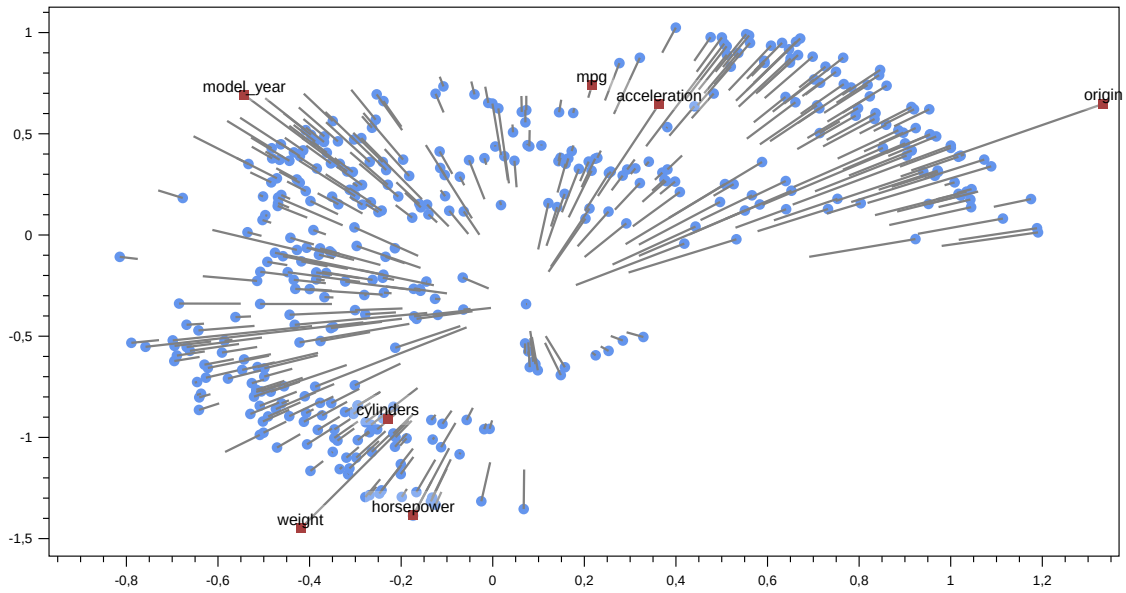
The Probing functionality (see Section 2.4) implemented in this thesis is not quite the functionality proposed by Stahnke et al. [SDMT16]. Our implementation is rather an extension. The original Probing allows for a specific selected low-dimensional datapoint to be error-corrected in the low-dimensional map so that the points distance to all other points is exactly matching the high-dimensional distance to each point. While this feature is still available (middle-clicking a datapoint in the map), an extension of this concept has been implemented. This extension allows for weighted average error correction. Instead of just correcting the distances for a single, selected point [SDMT16], the weighted average correction calculates an artificial, imaginative datapoint based on a predefined radius and all the low-dimensional points that are within this radius when a click anywhere on the map occurs. The weighting can be done equally among all datapoints within the radius (simple average), or it can be done with a weighting function like a kernel method (see Section 2.2). In our implementation, we use a Gaussian weighting function, giving points that are in the radius and are close to the click a higher weight than points close to the edge of the circle spanned by the radius.

[4] https://numerics.mathdotnet.com/

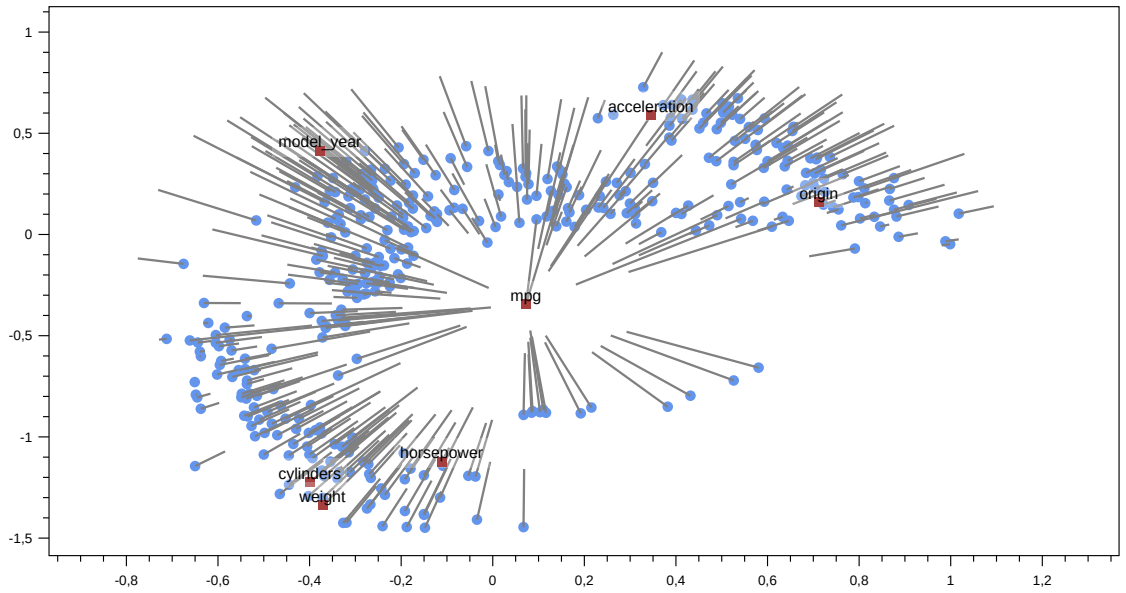
This feature allows applying the error correction to every point in the map, rather than being limited to the actual datapoints. Another advantage is that the error correction is not exclusively done for one selected datapoint anymore, but can be applied to a region of points. The radius needs to be set accordingly. This gives a more smoothed error correction, as the real high-dimensional distances towards a region of points will be shown. However, similar limitations as with the single-point Probing occur. The procedure only corrects the representation error for a local region specified by the radius. The distances between the points that are not part of that region are less informative than before. This correction only helps in understanding the error between the local region and the rest of the datapoints. It extends the error correction proposed in [SDMT16] from a single datapoint to be applicable to a small region of points.

The weighted average error correction for a region is available by right-clicking anywhere in the implementation's visualization. The radius can be set manually.

Figure 4.2 shows a comparison of the two Probing features by reference to the AutoMPG dataset. In the top image, the single-point Probing as proposed in [SDMT16] is shown. The distances are corrected just for the selected point, which is the almost centered blue point all lines are pointing to. The length of the lines indicates how much each other datapoint needed to be moved for the error correction. The bottom figure shows the weighted average error correction for a click in the same region. With a radius set to 0.2, not only the point from the upper image is regarded, but also the points that are within the radius. The correction is done for the whole region covered by the radius. What can be observed is that for example the attribute representation "weight" has a greater distance to the point in the single-point Probing than to the point weighted with its local neighbors. One possible interpretation is that the single point has a noticeably smaller value for the attribute "weight" then this point averaged with its neighborhood. This presumption can be confirmed when comparing the single datapoint with the imaginary, calculated weighted average datapoint. The single datapoint has a "weight" value of about 2470, while the averaged point has a value of about 3000.



(a) Probing interaction for a single chosen datapoint as proposed in [SDMT16].



(b) Performing the correction for the same datapoint, this time also regarding its neighbors within a radius of 0.2 around the point. The weighted average error correction performs a Gaussian weighting of all points in the radius around the click.

Figure 4.2: A comparison of the two Probing techniques. On top, the single point error correction and below the weighted average error correction for a DCM instance of the AutoMPG dataset.

5 Results

In this chapter, we are going to present and evaluate the results obtained from the implementation of the DCM. The first Section, 5.1, is about verifying that the implementation indeed gives a valid DCM representation. This is achieved by comparing the low-dimensional map obtained for a specific dataset with the map given for that same dataset in the original paper [CM16]. After that, Section 5.2 shows and evaluates the results of applying the DCM algorithm to different real-world datasets.

5.1 Verification of the Model

One important aspect of this thesis is to evaluate, whether the implementation described in Chapter 4 is correctly performing the DCM algorithm. This verification is crucial to apply the algorithm and to gain further insight into different datasets. We achieve this verification by the following means:

1. Choose a map generated by the DCM algorithm from the original paper [CM16] and obtain the underlying dataset.
2. Use that dataset to create a second low-dimensional map, this time with the implementation presented in Chapter 4.
3. Compare the two maps. If the second map matches the original map, the implementation is correctly calculating the DCM.

For the first point, it needs to be noted that the map of the university dataset, presented by Cheng and Mueller [CM16] and shown in a previous Chapter in Figure 3.1, can not be recreated. This is due to the dataset not being available publicly. Another possibility would be to use an artificial dataset like the 6 Gaussian test dataset [CM16] for the comparison. However, we think that a real-world dataset would be better suited, as 1) the comparison can be done for the exact same data and not for data generated from the same distribution and 2) the interpretation of a real-world data map is more intuitive and demonstrative than interpreting a Gaussian test dataset. For these reasons, we chose the AutoMPG[1] dataset as a baseline for the comparison. The AutoMPG dataset consists of 398 cars from the years 1970 to 1982. Each car is described with an eight-dimensional vector, accounting for the miles per gallon, cylinders, displacement, horsepower, weight, acceleration, model year, and origin of the car.

Figure 5.1 depicts the low-dimensional map for AutoMPG obtained from DCM as shown in the original paper [CM16]. Before applying our implementation to the same dataset, some issues need to be made aware of. Firstly, it needs to be noted that the original AutoMPG dataset has 8 attributes

[1] https://archive.ics.uci.edu/ml/datasets/auto+mpg

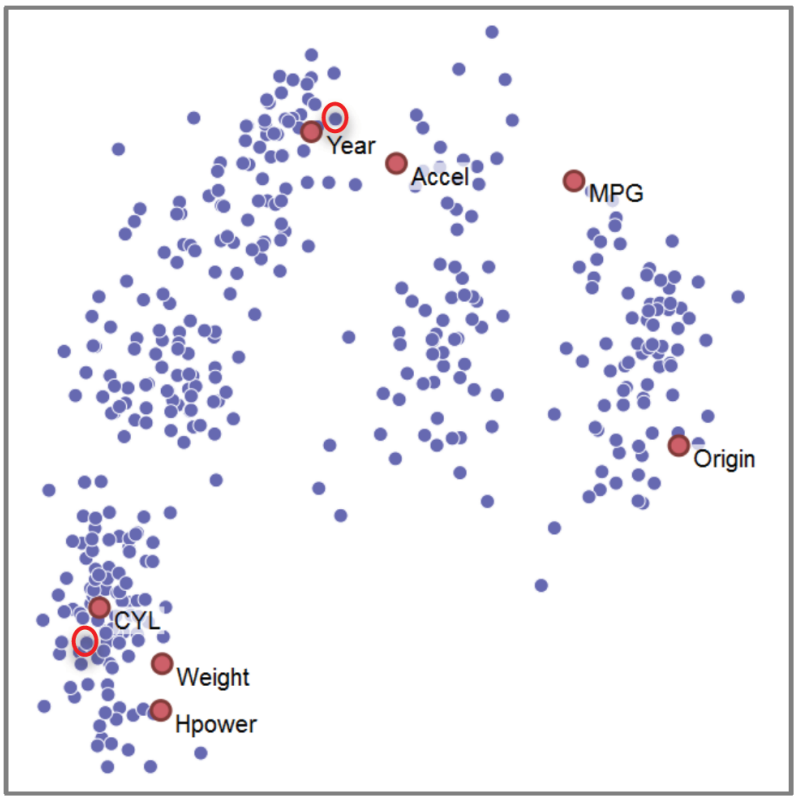


Figure 5.1: The DCM algorithm applied to the AutoMPG dataset in the original paper, taken from [CM16].

(without the class attribute), while the set used by the visualization has only 7 attributes [CM16]. The attribute "displacement" has been removed. Secondly, only 392 of the 398 instances have been regarded. This is where we can only assume which datapoints have been used, presumably the 392 datapoints that have no missing attribute value. 6 of the originally 398 cars have missing values for the horsepower attribute, as can be found in the dataset description[1].

The slightly modified dataset is used for the verification. Figure 5.2 shows the result of applying our own DCM implementation to this dataset. What can be observed is that the two maps in Figure 5.1 and 5.2 are not exactly alike. While there are four somehow distinct clusters in the original map, the custom map only shows the bottom cluster clearly. The other clusters are a bit more overlapping than in the original map. Also, the datapoints position is not exactly matching the points positions in the custom map. As all the datapoints in the custom map have been arranged a little differently, the exact positioning like in the original map has not been achieved.

However, the two results do also have a lot in common. At first, it can be seen that the custom map depicts a global structure close to the original map. Both maps show the bottom cluster distinct from the remaining datapoints, having those extend over the right side somehow equally. Further, and this is the most important aspect, the attribute representation in the custom map does nicely reflect the attribute representation in the original map. Take for example the three attributes "CYL", "Weight" and "HPower" in the original map. Their positions, even their positions among another, are very closely preserved in the custom map. The distances towards the other attributes, even the

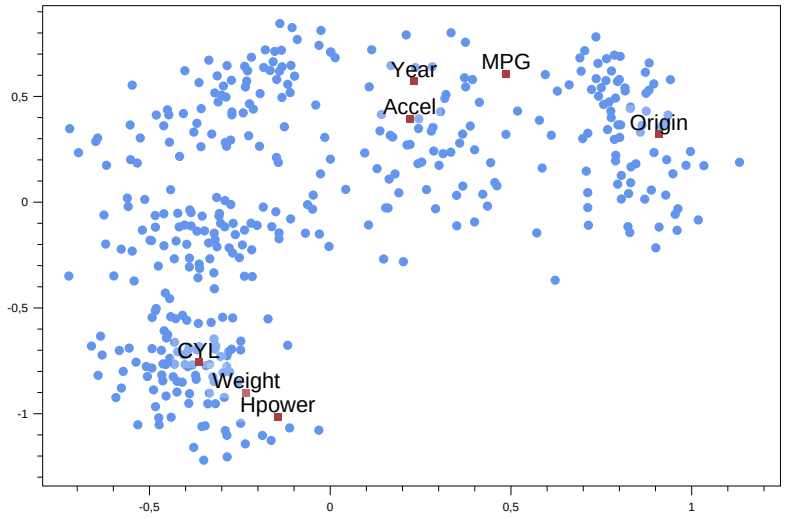


Figure 5.2: Low-dimensional representation generated for AutoMPG by the implementation presented in Chapter 4.

orientation and spatial relation of those ("Origin" on the very right side, "Year" on the upper left, guarding both "Accel" and "MPG" in the upper middle part) has fully been retained by the custom implementation. Interacting with the actual map, for example, selecting different datapoints, reveals that the datapoint-attribute relationship (higher value means closer distance to the attribute point) holds nicely, also in the custom map.

Due to the similarities regarding the plot and the positional relevance of the datapoints when interacting with the map, we find it safe to say that the implementation presented in Chapter 4 is calculating the correct low-dimensional DCM representation. The custom map has retained a vast majority of the features of the original map and all the features the DCM generally has.

What remains is the question to why the maps are not exactly looking alike. The true reasons can only be presumed. However, we identified some possible explanations for the difference:

1. It can't be said definitively that the 392 used cars are the exact subset of the AutoMPG dataset used by Cheng and Mueller [CM16]. Slightly different datapoints may gradually influence the location of the low-dimensional points.

2. The calculation of the CM matrix could give a different result than in the original paper. However, we find that explanation implausible, as most of the original map has been retained in the custom map, including all the important attribute-datapoints relationships.

3. For the original map, it can't be said which distance metrics exactly have been used for the different submatrices of the CM matrix (see Section 3.1 for more details). The choice of the distance metrics (e.g. Euclidean distance for the datapoint-datapoint relationship, Pearson correlation for the variable-variable relationship), is a crucial factor for the resulting low-dimensional map. As it has not been specified, which distance metrics have been used for the original AutoMPG map [CM16], we assumed a Euclidean distance for the datapoints, Pearson

correlation for the attributes and the (1-value) distance [CM16] for the datapoint-variable relation in the custom map. These distance metrics may however not reflect the original choice, hence explain the variance in the result.

4. The original Glimmer MDS algorithm performs a random shuffling (Fisher-Yates shuffle [Dur64]) of the input data. This random initialization leads to the algorithm always returning a slightly different resulting low-dimensional map. Although generally restricting the randomness in our own CPU Glimmer MDS adaption, this may have a significant influence on the resulting map. With the random shuffle included, multiple runs for the same dataset can be made and the best resulting map can be chosen. This is why it is extremely unlikely to achieve the exact representation as in Figure 5.1, as there are as many different initializations as permutations of the datapoints in the dataset. To obtain Figure 5.2, we ran the DCM algorithm a few times with shuffling included and chose the best map from those runs.

In summary, as can be concluded from this section, the implementation is correctly performing the DCM algorithm. Therefore, the next section will show some low-dimensional maps for different real-world datasets obtained with our DCM implementation.

5.2 Applying Data Context Map to Real World Datasets

This section shows the application of DCM to a selection of real-world data, aiming at improving the understanding of these datasets.

5.2.1 AutoMPG Dataset

The AutoMPG dataset visualized with DCM is already shown in Figure 5.2. What can be seen is that the cars align in five more or less ambiguous clusters. The most distinct cluster is the grouping in the bottom left, build by the datapoints and the attribute representations “CYL”, “Weight” and “HPower”. As all cars within that grouping have a high number of cylinders, a high value for weight and horsepower and a low origin (indicating North America), this cluster can be interpreted to be built by heavy American muscle cars. An exploration of the datapoints confirms that this type of car is located in that cluster.

While the three attributes “CYL”, “Weight” and “HPower” are strongly correlated (they are close to each other), they have a great distance towards the remaining four attributes “Year”, “Accel”, “MPG” and “Origin”. This tells that the two attribute groups are not correlated. As one would expect, cars with a high weight, horsepower, and many cylinders have a pretty low fuel efficiency (“MPG” is far away). They also have a low value for acceleration, as they typically achieve high speed very fast (4s for 100 km/h is faster than 15s). The location of the attribute “Year” on the other hand tells that newer cars tend to have a better MPG value, for example, because technology advances and the fuel efficiency becomes better. The map provides helpful insight into the structure and hidden relationships within the dataset.

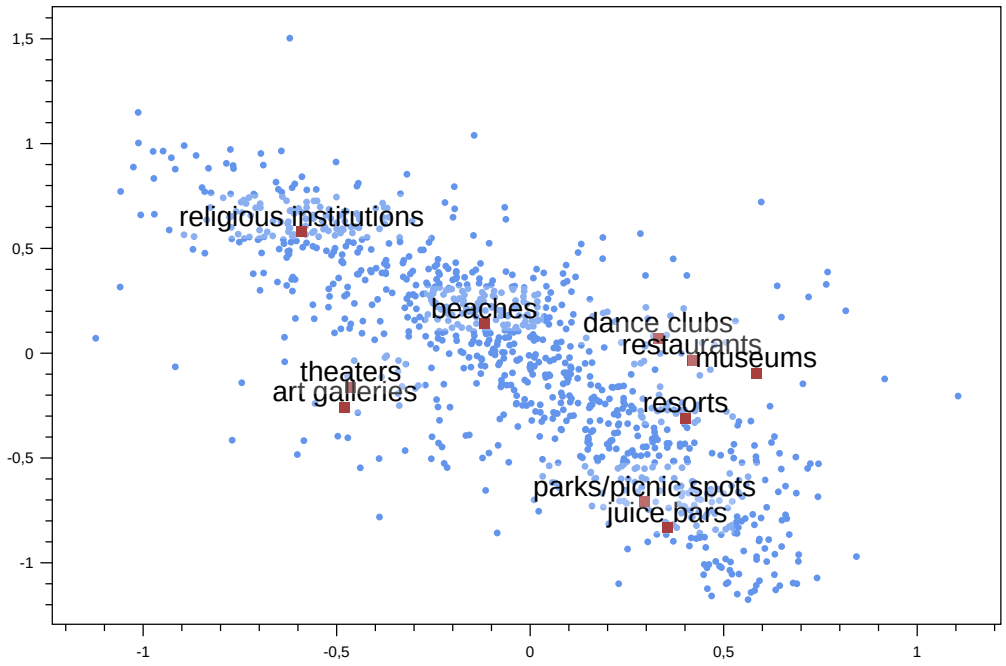


Figure 5.3: DCM representation generated for the Travel Reviews dataset. Each review is given for 10 categories, building the attributes visible in the map.

5.2.2 Travel Reviews Dataset

The Travel Reviews dataset[2] is a collection of 980 reviews for tourist destinations across East Asia. Each review is given for a single traveler and is made up of the average rating for that user for multiple locations. Possible ratings are Excellent(4), Very Good(3), Average(2), Poor(1), and Terrible(0). The attributes are given by the rating for 10 different categories, namely “art galleries”, “dance clubs”, “juice bars”, “restaurants”, “museums”, “resorts”, “parks/picnic spots”, “beaches”, “theaters”, and “religious institutions”. So each datapoint corresponds to a review of an East Asia travel for a single user. For the construction of the DCM, the Cosine distance has been used as a measurement for the datapoint-datapoint similarity. For the attribute-attribute relationship, we used the Pearson correlation distance and for the attribute-datapoint relation, the (1-value) distance [CM16] was used. We took a fixed random permutation as initialization for the Glimmer MDS instance, namely the order provided by the dataset itself. This makes the results easily reproducible.

Figure 5.3 shows the result of applying the DCM algorithm to the dataset. On the first view, it can be seen that there are no clear clusterings within the map. The grouping of datapoints around the attribute representation “religious institutions” however might be regarded as a cluster, same with the datapoints arranged around “theaters” and “art galleries”. Still, the clusters are blurry. What we would like to achieve now is to gain further insight into the dataset and its interpretation.

[2] http://archive.ics.uci.edu/ml/datasets/Travel+Reviews

Regarding the attribute representations, several clusters can be found. On the one side, there is a cluster of parks/picnic spots and juice bars and a cluster of dance clubs, restaurants, and museums, with the attribute "resort" between them. On the other side, there is a cluster of theaters and art galleries. "Religious institutions" and "beaches" are somehow separated. We interpret this as follows:

- Reviews, that have a good theater rating also have a good art galleries rating and vice versa. This is because the attributes are very close. So travelers, who like theaters, often also like art galleries.
- Those travelers tend to give not so good ratings to dance clubs, restaurants, and museums, as they are far away.
- Travelers, who rate parks, picnic spots, and juice bars nicely, are unlikely to give a high rating to religious institutions. Those clusters are very separated. Travelers of the theater/art gallery type are more likely to rate religious institutions well, these attribute representations are closer.
- All types of travelers have in common a similar rating of beaches, as this attribute is equally apart from the other attributes. This can be interpreted in that every type of traveler likes the beach more or less equally and rates it accordingly.

Most reviews are centered between the different attribute clusters, so most reviews are not that extreme for some specific attributes. Only a few outlier ratings are far apart from the others.

5.2.3 Forest Fires Dataset

The Forest Fires dataset[3] proposed by Cortez and Morais [CM07] consists of 517 forest fires observed in the Montesinho Natural Park in Portugal. Each fire is described with 13 attributes. These attributes are explained in Table 5.1.

The dataset is originally made for regression tasks with the aim to predict the attribute "area". However, as stated by [CM07], the regression task given by this dataset is a very difficult one, because the area attribute is hard to correlate with the other attributes. Therefore we use this dataset as a benchmark for the DCM algorithm. Figure 5.4 depicts the map generated with the DCM algorithm for the Forest Fires dataset. Again, a fixed random initialization (without Fisher-Yates shuffling for the Glimmer MDS algorithm) was used, given by the order of the dataset itself. The datapoint dissimilarity was measured with the Euclidean distance and the attribute dissimilarity was measured with the Pearson correlation coefficient. We used the (1-value) distance [CM16] for the datapoint-attribute relationship.

Preliminary analysis of the dataset revealed that, although small, the best correlation with "area" is given by "temp", "ISI" and "humidity". Our visualization confirms this correlation, at least for 2 out of the 3 given attributes, because "ISI" and "humidity" are close to the attribute "area". Intuitively this makes sense. The "ISI" attribute measures the expected rate of fire spread, a high value for this attribute is likely to cause a large area of forest to be burned. The same argument holds for the temperature. Apparently, the humidity also has an effect on the area. While the majority of

[3] https://archive.ics.uci.edu/ml/datasets/forest+fires

Attribute	Description
X	x-axis coordinate (from 1 to 9)
Y	y-axis coordinate (from 1 to 9)
month	Month of the year (January to December)
day	Day of the week (Monday to Sunday)
FFMC	FFMC code
DMC	DMC code
DC	DC code
ISI	ISI index
temp	Outside temperature (in °C)
humidity	Outside relative humidity (in %)
wind	Outside wind speed (in km/h)
rain	Outside rain (in mm/m^2)
area	Total burned area (in ha)

Table 5.1: Description of the 13 attributes of the Forest Fires dataset, taken from [CM07]

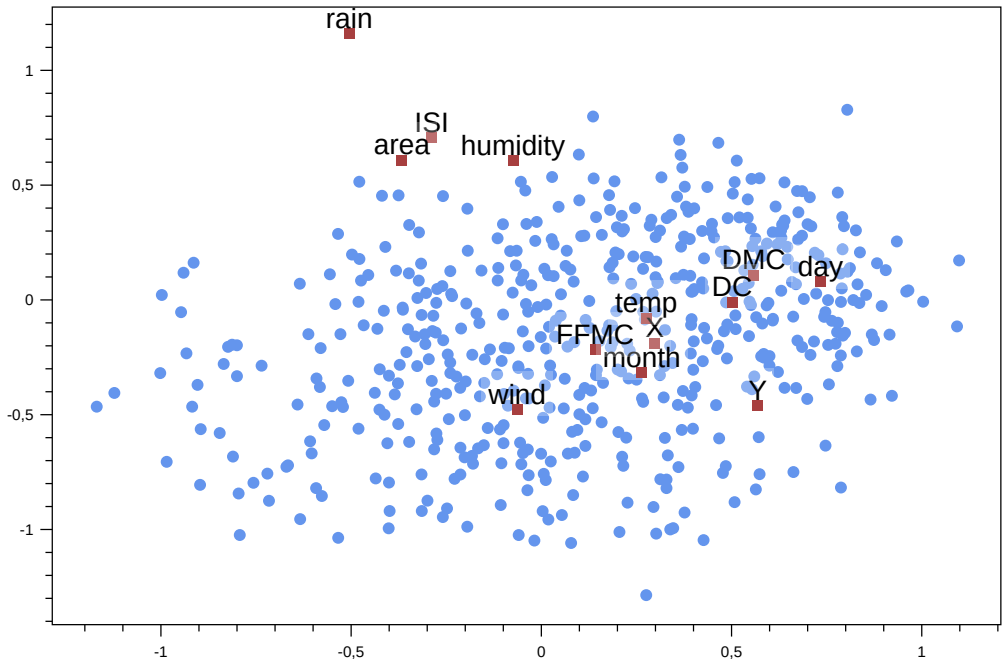


Figure 5.4: The map created by DCM for the Forest Fires dataset. Each fire is described with 13 attributes.

the remaining attributes are more or less centered within the datapoint cloud and do not give away much interpretation, the attribute “rain” was plotted far away from the other points. This can be interpreted in that “rain” has very little to no correlation with the other attributes, especially with the attributes in the middle of the point cloud (“wind”, “FFMC”, etc.).

The structure of the low-dimensional datapoints does not match a clear and unambiguous clustering of the fires. This can be understood in that there is not a really good clustering of the fires detectable within the dataset. Although the algorithm finds the grouping of correlated attributes, this dataset seems to be a hard task for DCM, as no real structure is observable, for example in contrast to the two datasets provided above. Still, the map helps to gain insight into some of the attributes that are more likely to cause a large forest fire. Further evaluation would be needed, for example by applying different weighting schemes to the submatrices that make up the CM matrix, to achieve a more insightful low-dimensional representation. Also, different distance metrics could be evaluated to further improve the map. For this visualization, Euclidean-, Cosine- and Pearson distances have been tried, although there possibly exists a variety of better-suited metrics. Onwards in Chapter 6, we present some ideas on how DCM could be further improved to deal with more complex datasets.

6 Conclusion

This thesis gave a detailed description of the theory and the implementation for the DCM algorithm, augmented by the improvements offered with the Probing interaction technique. Chapter 2 and Chapter 3 provided the theoretical foundations for the sub-algorithms and for the DCM algorithm itself, building the groundwork for the implementation presented in Chapter 4. In the last chapter, we verified the implementation and gave some examples of its application to real-world datasets.

Two main statements can be inferred from the results. Firstly, the implementation of the algorithm proposed in this thesis is correctly calculating the DCM. This has been evaluated with a "ground truth" DCM instance from the original paper [CM16]. As the implementation has been verified, it can be used to analyze high-dimensional datasets. Secondly, the application of DCM to real-world datasets showed that the method can help to gain insight into a dataset. Interpretability will be improved and data analysts, as well as people with a non-technical background, are provided a more intuitive way to analyze and understand datasets. Next to analyzing high-dimensional datasets, the method can be used for information selection tasks, where for example the contour lines of the DCM help to easily filter for specific attribute value ranges.

For the AutoMPG dataset for example, the DCM (Section 5.2.1) revealed some inherent structure in the dataset, namely a clustering of the cars into roughly five groups. Next to the occurrence of the clusters, the DCM also helped in identifying, how the clusters can be interpreted. As can be observed in the map, the occurrences and positioning of the attribute representations help in understanding what makes a grouping a cluster. For example, one cluster has been interpreted to be the set of American muscle cars, as the attribute representation locations gave away that those cars are all American, have a high weight, much horsepower, and many cylinders.

We further evaluated the performance of DCM on advanced datasets. The Forest Fires dataset [CM07] has been described by the authors as a difficult regression dataset because the area to predict is hard to relate to the other given attributes (see Section 5.2.3). The map showed that the dataset is indeed a hard task for DCM. Although at least for the area attribute, a small correlation with some other attributes has been found, the map generally does not help in understanding the dataset very well. Another hint for the difficulty of the dataset can be found when running the procedure multiple times. For a random initialization of the Glimmer MDS instance, which is the default, multiple runs return quite different low-dimensional maps. The maps vary much by the location of the datapoints and the attributes, a good dimensionality reduction would be less prone to these local minima. This shows that at least for the Glimmer MDS dimensionality reduction scheme, it has its difficulties with the low-dimensional embedding needed for the Forest Fires DCM instance.

In future work, we propose to look for further advanced dimensionality reduction schemes to be used for the DCM algorithm. Although Glimmer MDS is a fast and very powerful MDS scheme, we believe that there is even more potential for an accurate low-dimensional representation retrieval. One reason for the usage of Glimmer MDS as an advanced MDS variant was that the resulting

low-dimensional map precisely captures the global relationship of the dataset regarding the chosen distance metric. This is a property of all MDS algorithms, that the global structure is preserved, while focusing less on local structures in the data. While it is generally plausible to account for the global structure of a dataset, at least for visualization reasons, there are some arguments why it can be important to consider the local structures of a dataset as well. For example, for large datasets, it can be interesting to see how different clusters are structured internally, as this helps more in understanding the cluster itself.

This is why the recent development of powerful dimensionality reduction schemes like t-SNE (Section 2.1.3) could be further evaluated for the DCM algorithm. These algorithms are very good at finding clusterings in the data, even for complex datasets. And while the global structure is usually preserved, just what Glimmer MDS is excellent in, these algorithms also account for the local structure of the data. For example, it could be evaluated if a more advanced dimensionality reduction scheme gives a better DCM representation for the Forest Fires dataset, so that the dataset can be better analyzed. Especially interesting could be the performance of DCM when using the novel Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction scheme [MHM18]. This algorithm gives better visualizations than t-SNE while having superior runtime and preserving more of the global structure [MHM18].

Another aspect that could be evaluated for future work is the influence of different distance metrics that are used within DCM to create the CM matrix. For this thesis, we evaluated the Euclidean-, Pearson, Cosine, and (1-value) distance proposed by [CM16]. However, the choice of the distance metric is crucial for the resulting low-dimensional map. The (1-value) distance for example is used to determine the datapoint-attribute relationship within the map. A different metric could be used to influence the meaning of the locations of the low-dimensional points and attributes, creating maps in which the height of an attribute value is not or not only determined by the closeness towards the attribute representation.

Regarding the interaction techniques described in this thesis, it can be said that the error correction given by the Probing interaction (Section 2.4 and Section 4.2) further helps in the dataset analysis. The extension of Probing to apply the error correction to a local region in the map, instead of one selected datapoint, seems to be a promising approach to data exploration. We suggest that further research can be done, similar to the ProxiLens approach [HAF13].

This thesis provided an operative implementation of the DCM algorithm, enhanced by an error-correcting interaction approach called Probing. Next to the Probing described by Stahnke et al. [SDMT16], we implemented a weighted average local neighborhood selection that smooths the error correction to be applied to a small local region, instead of a specific datapoint only. The results are promising and they show, that the DCM can help in the analysis and exploration of multidimensional datasets.

Bibliography

[BCHC09] J. Benesty, J. Chen, Y. Huang, I. Cohen. "Pearson correlation coefficient". In: *Noise reduction in speech processing*. Springer, 2009, pp. 1–4 (cit. on p. 32).

[BG05] I. Borg, P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005 (cit. on p. 27).

[BP06] U. Brandes, C. Pich. "Eigensolver methods for progressive multidimensional scaling of large data". In: *International Symposium on Graph Drawing*. Springer. 2006, pp. 42–53 (cit. on p. 22).

[CC00] T. F. Cox, M. A. Cox. *Multidimensional scaling*. Chapman and hall/CRC, 2000 (cit. on p. 19).

[Cha96] M. Chalmers. "A linear iteration time layout algorithm for visualising high-dimensional data". In: *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE. 1996, pp. 127–131 (cit. on pp. 21, 22).

[CM07] P. Cortez, A. d. J. R. Morais. "A data mining approach to predict forest fires using meteorological data". In: (2007) (cit. on pp. 48, 49, 51).

[CM16] S. Cheng, K. Mueller. "The data context map: Fusing data and attributes into a unified display". In: *IEEE transactions on visualization and computer graphics* 22.1 (2016), pp. 121–130 (cit. on pp. 3, 13–15, 19, 29–35, 37, 39, 43–48, 51, 52).

[DM11] J. De Leeuw, P. Mair. "Multidimensional scaling using majorization: SMACOF in R". In: (2011) (cit. on p. 19).

[Dur64] R. Durstenfeld. "Algorithm 235: random permutation". In: *Communications of the ACM* 7.7 (1964), p. 420 (cit. on p. 46).

[Fis36] R. A. Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188 (cit. on p. 13).

[Gho06] A. Ghodsi. "Dimensionality reduction a short tutorial". In: *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37 (2006), p. 38 (cit. on p. 21).

[GTC] G. Grinstein, M. Trutschl, U. Cvek. "High-dimensional visualizations". In: Citeseer (cit. on p. 13).

[HAF13] N. Heulot, M. Aupetit, J.-D. Fekete. "Proxilens: Interactive exploration of high-dimensional data using projections". In: *VAMP: EuroVis Workshop on Visual Analytics using Multidimensional Projections*. The Eurographics Association. 2013 (cit. on pp. 14, 52).

[Heu16] H. Heuer. "Text comparison using word vector representations and dimensionality reduction". In: *arXiv preprint arXiv:1607.00534* (2016) (cit. on p. 13).

[HGM+97] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, E. Stanley. “DNA visual and analytic data mining”. In: *Proceedings. Visualization’97 (Cat. No. 97CB36155)*. IEEE. 1997, pp. 437–441 (cit. on pp. 13, 29).

[HTF09] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN: 9780387848846. URL: https://books.google.de/books?id=eBSgoAEACAAJ (cit. on pp. 25, 26).

[IMO09] S. Ingram, T. Munzner, M. Olano. “Glimmer: Multilevel MDS on the GPU”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.2 (2009), pp. 249–261 (cit. on pp. 20–22, 37, 39).

[Kan00] E. Kandogan. “Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions”. In: *Proceedings of the IEEE Information Visualization Symposium*. Vol. 650. Citeseer. 2000, p. 22 (cit. on p. 29).

[KH05] A. Krowne, M. Halbert. “An initial evaluation of automated organization for digital library browsing”. In: *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*. ACM. 2005, pp. 246–255 (cit. on p. 22).

[LV07] J. A. Lee, M. Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007 (cit. on pp. 17–19, 21).

[MH08] L. v. d. Maaten, G. Hinton. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605 (cit. on pp. 13, 23, 24).

[MHM18] L. McInnes, J. Healy, J. Melville. “Umap: Uniform manifold approximation and projection for dimension reduction”. In: *arXiv preprint arXiv:1802.03426* (2018) (cit. on p. 52).

[Nad64] E. A. Nadaraya. “On estimating regression”. In: *Theory of Probability & Its Applications* 9.1 (1964), pp. 141–142 (cit. on p. 26).

[Par62] E. Parzen. “On estimation of a probability density function and mode”. In: *The annals of mathematical statistics* 33.3 (1962), pp. 1065–1076 (cit. on p. 24).

[Pea01] K. Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901). PCA beginnings, pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720 (cit. on p. 18).

[RSS18] M. Rottschäfer, V. Sabbatino, S. Söhnel. “Analyse und Vergleich von Projektionstechniken zur Dimensionsreduktion”. Fachstudie. University of Stuttgart, 2018 (cit. on pp. 19, 23, 33).

[SDMT16] J. Stahnke, M. Dörk, B. Müller, A. Thom. “Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions”. In: *IEEE transactions on visualization and computer graphics* 22.1 (2016), pp. 629–638 (cit. on pp. 3, 14, 26–28, 38–41, 52).

[Sil18] B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018 (cit. on pp. 24, 25).

[SSM98] B. Schölkopf, A. Smola, K.-R. Müller. “Nonlinear component analysis as a kernel eigenvalue problem”. In: *Neural computation* 10.5 (1998), pp. 1299–1319 (cit. on p. 18).

[SVM14] C. O. S. Sorzano, J. Vargas, A. P. Montano. “A survey of dimensionality reduction techniques”. In: *arXiv preprint arXiv:1403.2877* (2014) (cit. on pp. 17–19).

[TDL00] J. B. Tenenbaum, V. De Silva, J. C. Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *science* 290.5500 (2000), pp. 2319–2323 (cit. on p. 17).

[Tor65] W. S. Torgerson. “Multidimensional scaling of similarity”. In: *Psychometrika* 30.4 (1965), pp. 379–393 (cit. on p. 19).

[Van+03] P. Van Kerm et al. “Adaptive kernel density estimation”. In: *The Stata Journal* 3.2 (2003), pp. 148–156 (cit. on pp. 25, 37).

[Wat64] G. S. Watson. “Smooth regression analysis”. In: *Sankhyā: The Indian Journal of Statistics, Series A* (1964), pp. 359–372 (cit. on pp. 25, 26).

[Weg90] E. J. Wegman. “Hyperdimensional data analysis using parallel coordinates”. In: *Journal of the American Statistical Association* 85.411 (1990), pp. 664–675 (cit. on p. 29).

[Wic03] F. Wickelmaier. “An introduction to MDS”. In: *Sound Quality Research Unit, Aalborg University, Denmark* 46.5 (2003), pp. 1–26 (cit. on pp. 20, 21).

All links were last followed on April 4, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature