Institut für Visualisierung und interaktive Systeme
Abteilung Visualisierung
Universität Stuttgart
Allmandring 19
70569 Stuttgart

Bachelorarbeit

Cluster-Based Color Linking in Scatterplot Matrix

Mark Reissenberger

Course of Study:	Data Science
Examiner:	Prof. Dr. Michael Sedlmair
Supervisor:	Dr. Quynh Quang Ngo, Dr. Michael Aupetit
Commenced:	October 1st, 2024
Completed:	March 31st, 2025

Abstract

This thesis contributes a novel visual analysis approach for multi-dimensional data by enhancing the utility of scatterplot matrices (SPLOMs), focusing on automatic cluster detection and trend/pattern/functional dependency identification utilizing suitable color linking strategies among views. Traditional SPLOMs effectively perform analysis tasks, including identifying dimensionality within or between-cluster structures and trend/spatial correlation across 2D subspaces. Despite being effective for the tasks, traditional SPLOMs often require extensive manual exploration through brushing and linking among the views to uncover patterns across multiple dimensions. To address these limitations, this bachelor thesis studies several automatic assigning color strategies for view linking.

In particular, colors from a 2D color map will be assigned to points of the scatterplot of interest, taking advantage of two specific clustering techniques: Mean Shift and Self-Organizing Maps (SOM). We utilize Mean Shift to assign colors to scatterplot points to observe spatial information between cluster structures. Meanwhile, SOM is used and adapted to maximize the number of colors to be used, at the same time preserving the spatial information to support observing trend/correlation patterns within-cluster structures among 2D subspaces.

We proposed a visual analysis approach with multiple link views from the two strategies. A main and comparison view are created which allows users to directly compare clusters and patterns between different projections without repeatedly scanning through the entire SPLOM, thus significantly reducing cognitive load and manual effort. The Master View allows users to select a specific scatterplot for detailed analysis, automatically assigning colors based on cluster detection results. The Comparison View enables users to juxtapose another scatterplot against the Master View, facilitating quick comparative analysis between different dimensional combinations. Interactions in our visual analysis approach occur through dynamic linking among these views. When clusters are automatically identified and colored in one view (the Main view), these colors propagate consistently across all other views, including the Comparison view and SPLOM. Users can interactively adjust clustering parameters, select subsets of clusters for further refinement, and apply clustering algorithms again on these subsets. Such interactions enable users to iteratively drill down into specific parts of their data, exploring cluster structures at varying granularities. The primary goal of these interactions and workflows is to facilitate rapid identification of meaningful clusters and functional dependencies across multiple dimensions, while minimizing manual selection efforts and enhancing exploratory efficiency.

An instantiated interface is also implemented, integrating the Mean Shift algorithm for automatic cluster detection and Self-Organizing Maps (SOMs) for topology-preserving color assignments. This enhances the SPLOM by two side-by-side views of SPLOM entries for quick comparisons. Additionally, the user is able to control the color diversity interactively and adjust the display of the data in various ways. This allows for an intuitive and highly customizable workflow, which can adjust views, color diversity, and clustering techniques. The interface also supports interactive refinement, allowing users to isolate subsets of clusters and reapply clustering algorithms for more granular analysis.

The thesis showcases several case studies from several synthetic datasets; the respective computational performance evaluation is carried out for each strategy. Through the qualitative and quantitative evaluations, the visual analysis approach demonstrates significant improvements in

cluster detection accuracy, functional dependency identification, and cognitive load reduction compared to traditional SPLOM interfaces. A robust tool for effective multi-dimensional data analysis within Jupyter Notebooks is also provided (with code) and attached to the thesis submission.

Contents

1 Introduction **11**

2 Background **17**
2.1 Multi-Dimensional Data Representation 17
2.2 Brushing and linking 17

3 Related Work **19**

4 Method **23**
4.1 Clustering Algorithms 23
4.2 Coloring with Color Maps 27
4.3 Visual Summary 28
4.4 Functions 30

5 Evaluation **37**
5.1 Qualitative Evaluation 37
5.2 Quantitative Evaluation 50

6 Implementation **53**
6.1 Storage and Representation of the color map and data space 53
6.2 Mean Shift Clustering and Coloring 53
6.3 Bandwidth Thresholds Calculation 55
6.4 SOM and Coloring 58

7 Discussion **61**

8 Conclusion **63**

Bibliography **65**

List of Figures

1.1 Dataset 1 (monochrome views) . . . 11
1.2 Brushing and linking Dataset 1 (views): Brushing and linking reveals splits, however they can't be investigated further since they are occluded by points from another distribution . . . 12
1.3 Dataset 2: Monochromatic SPLOM with noise covering functional dependencies 13
1.4 Dataset 2 (SPLOM):Brushing the right cluster reveals the dependency in the v, y and u, x plots . . . 14

4.1 Color Map Mean Shift (Dataset 2): Colors of the cluster are based on the color coordinates of the cluster centers (black circles) . . . 24
4.2 Weight movement SOM for different epochs: The first epochs training the SOM. In the beginning, the weights collapse quickly to bring colors, which are relatively far away from clusters, closer to the data. Then, updates are done less aggressively and the weights spread out more within clusters. . . . 26
4.3 Color Map 1 . . . 27
4.4 Color Map 2 . . . 27
4.5 Widgets and Colormap view of the interface . . . 28
4.6 Main and Comparison views of the interface . . . 28
4.7 The SPLOM of the interface. Individual scatter plots can be displayed as the main or comparison view . . . 29
4.8 Cluster selection interface (start): The dropdown allows for drilling down into the data by cluster color . . . 31
4.9 Cluster selection interface (end): Selected points are rescaled to the color space and clustering/coloring is done on this subset. Deselected points remain visible. . 32
4.10 SOM initial weights configuration: Weights are colored rectangles and points grey circles. Data is from the IRIS dataset. . . . 33
4.11 SOM IRIS coloring (before training): Coloring before training is equivalent to coloring by nearest neighbor on the downsampled color map. . . . 33
4.12 SOM IRIS weights (10 epochs): Convergence of weights to the data is slow to minimize topology violations. . . . 34
4.13 SOM IRIS coloring (10 epochs): Coloring becomes more diverse as an increased percentage of the color map becomes a bmu . . . 34
4.14 SOM IRIS weights (52 epochs): At this point training terminates since the entropy from 4.1.2 stabilizes from this point onwards. This example also shows that parts of the color map can stay far away from the data. . . . 34
4.15 SOM IRIS coloring (52 epochs): Coloring at the end of training reaches its maximum diversity. Nearly the entire color map has been mapped onto the data with little topology distortions. . . . 34

4.16 Hybrid function for IRIS (start): The user wants to apply SOM on the points from the top two clusters which have been determined with Mean Shift. 35
4.17 Hybrid function for IRIS (end): After applying the cluster selection, SOM is applied on the subset. The deselected points remain visible. 35
4.18 Hybrid function initial SOM weights: The initial configuration of weights automatically rescales to the cluster selection. 36
4.19 Hybrid function SOM weights after training: Usage of the color map is maximized for any subset selection. 36

5.1 Use Case 1 with Mean Shift (start): Mean Shift directly reveals the split between B1 and B2 if two clusters are selected. Cluster A can be filtered out and then Mean Shift applied on (y, z) 38
5.2 Use Case 1 with Mean Shift (end): Running Mean Shift on the filtered new view allows the user to further explore clusters B1 and B2. 39
5.3 Use Case 1 with SOM: The high diversity of colors makes the cluster identification very difficult and a lack of filter options hinders the user from identifying the indices 40
5.4 Use Case 1 with the hybrid function: While it seems to offer the best of both worlds, the hybrid case lacks precision when compared to Mean Shift. 41
5.5 Use Case 2 with Mean Shift: Mean Shift offers no advantage over brushing and linking in spotting the functional dependencies. 43
5.6 Use Case 2 with SOM: The orange/yellow and light green cluster is just as “smoothly colored” in (u, v) as it is in (x, y). This instantly reveals the functional dependencies. 44
5.7 Use Case 3 Data: The split rectangle is “reattached” in the (u, v) view. 45
5.8 Use Case 3 with Brushing and linking: Brushing allows for a quick spotting of the reattached subclusters, but lookups in the SPLOM are still required to spot the functional dependencies. 46
5.9 Use Case 3 with Mean Shift: Mean Shift allows for a quick discovery of clusters, but offers no advantage in spotting functional dependencies. 47
5.10 Use Case 3 with SOM (before training): At the start of training there is practically no separation visible for the subclusters in (u, v) 48
5.11 Use Case 3 with SOM (end of training): Color diversity increases at the end of training, but the color gradients at the boundaries remain small. 49
5.12 Use Case 3 with the hybrid function: The application of SOM on only the subclusters doesn’t significantly increase the boundary color gradients. 49

Listings

6.1 Mean Shift Clustering and Coloring 54
6.2 Heuristic in bandwidth threshold calculation 55
6.3 Binary Search for calculating bandwidth thresholds 56
6.4 Direct and Interpolated Coloring for SOM 58
6.5 The calculation of BMU entropy 58

1 Introduction

Dimensionality Reduction techniques have been utilized to visually analyze multi-dimensional data to spot trends and clusters or to visually represent the data in an understandable way. This is especially important for labeling previously unlabeled data. One of the visualization techniques is scatterplot matrix (SPLOM), which represents multi-dimensional data in all 2D combinations of the data by orthogonal projection to the respective 2D subspaces. Several techniques can then be used in order to detect multi-dimensional clusters.

The Gestalt Laws [War04] describe a range of pre-attentive detection processes which can be employed to make the visual detection of similar patterns easier. The law of proximity enables us to quickly perceive clusters in scatterplots. However, this does not help with determining whether the points from these clusters also form clusters in other scatterplots of the SPLOM. This is addressed in the brushing and linking technique by making use of another Gestalt law, the law of similarity. When a set of points is selected, they are clearly identified in a distinct color or opacity across all scatterplots in the SOM. The user can then pre-attentively spot how and if these points form clusters in the high dimensional space.

The drawback of brushing and linking is that the user needs to do a lot of manual work, selecting all clusters. It also makes the spotting of patterns in the following examples difficult. We show here an example in Figure 1.1, the data consists of three dimensions in the 3D space. In the (x, y) scatterplot, two clusters can be identified.

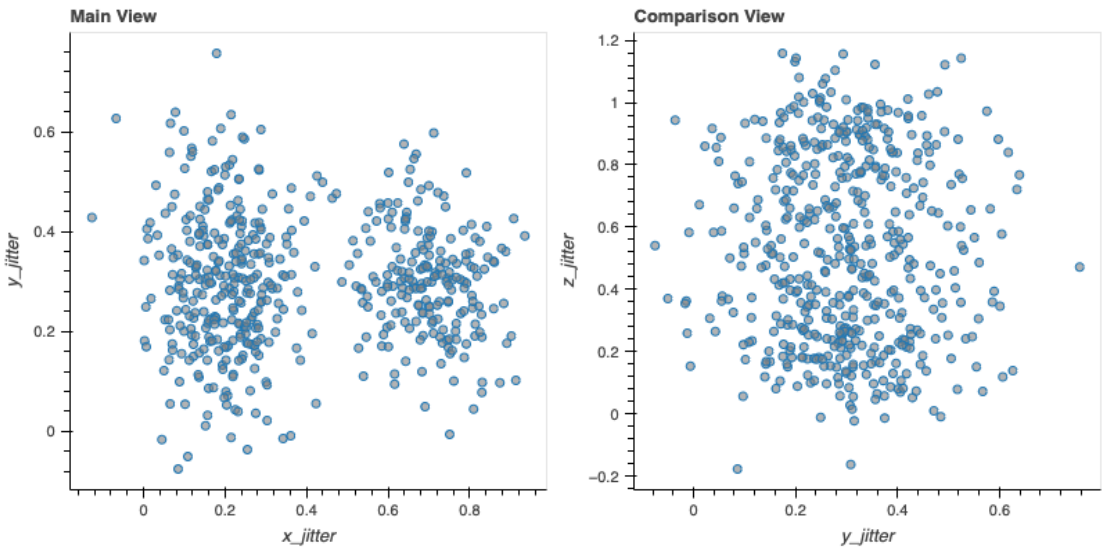


Figure 1.1: Dataset 1 (monochrome views)

However, one cluster from this view is further split into two clusters in another view. In the other view, the points from another cluster occlude these splits. Now the user may want to select the individual splits to check if these subclusters are also present in other scatterplots of the SPLOM. This would be impossible because the points from another cluster occlude these splits. Additionally, the user has to remember where these splits lie in the second scatterplot. This cluster occlusion is therefore a significant issue due to a lack of filtering and drill-down options over several iterations.

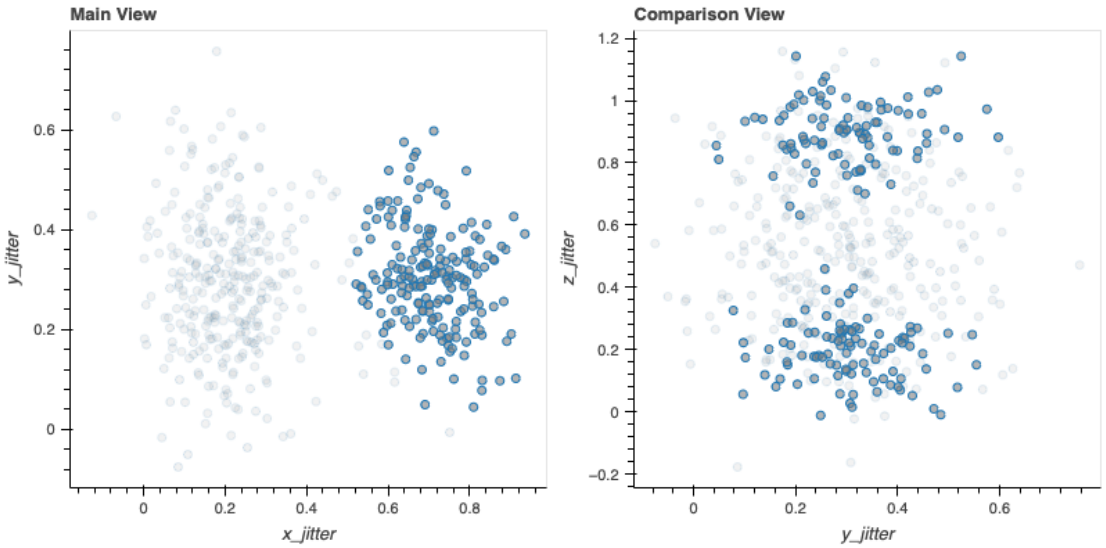


Figure 1.2: Brushing and linking Dataset 1 (views): Brushing and linking reveals splits, however they can't be investigated further since they are occluded by points from another distribution

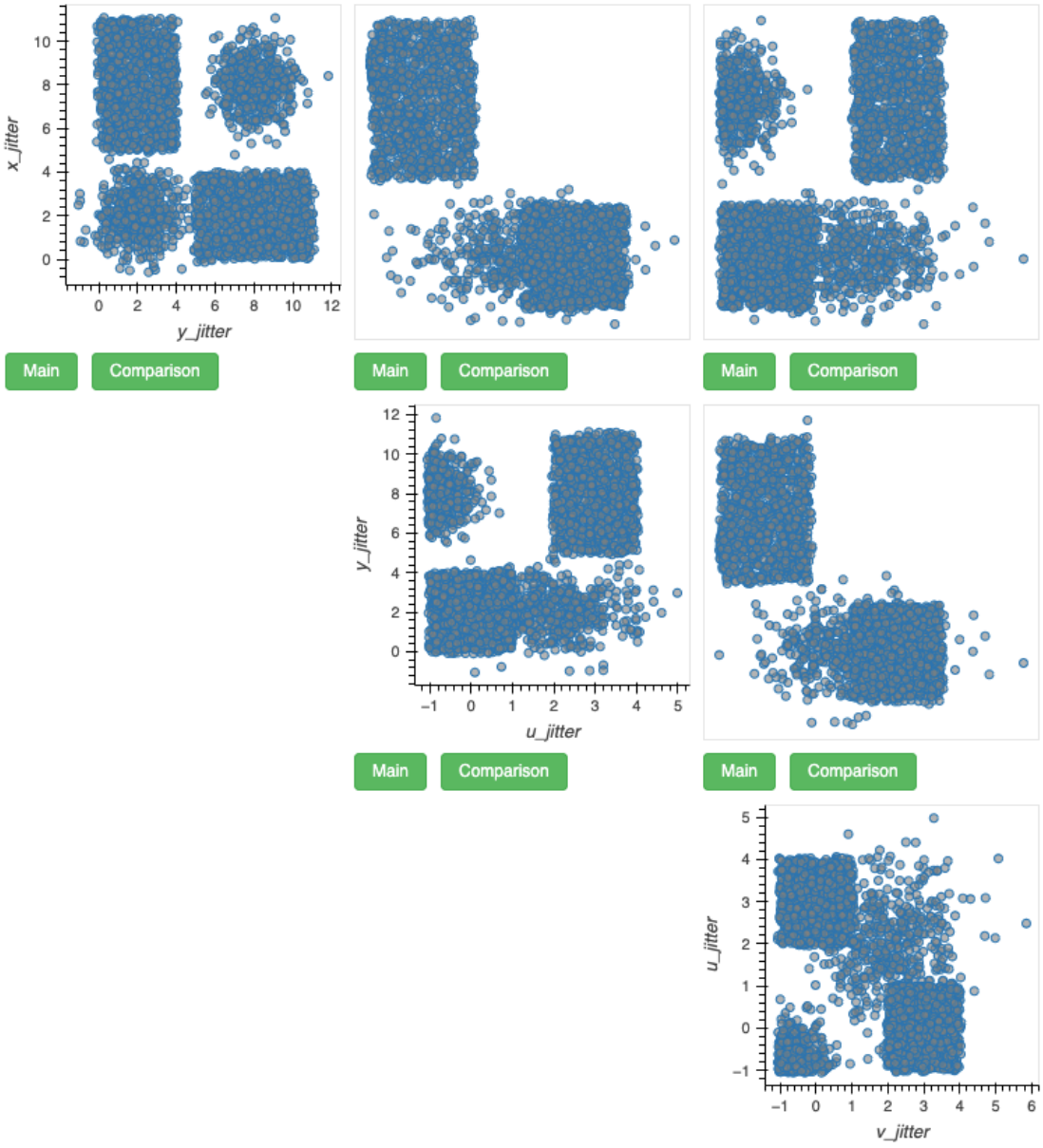


Figure 1.3: Dataset 2: Monochromatic SPLOM with noise covering functional dependencies

Another issue arises when attempting to spot functional dependencies with brushing and linking. In Figure 1.3, there are two Gaussian clusters and two uniform (noise) clusters in the x, y view. The points in the top right cluster also form a cluster in the u, v space; however, there is no functional dependency between their (x, y) and (u, v) values. The points in the bottom left cluster do have a functional dependency between, namely:
$u = sin(0.25x$ and $v = cos(0.25y)$
The first issue here is that the user does not know which cluster to select for brushing and linking and where these dependencies may lie. So the user has to select in the worst case all clusters and then look in the whole SPLOM for shapes indicating some kind of dependency between features.

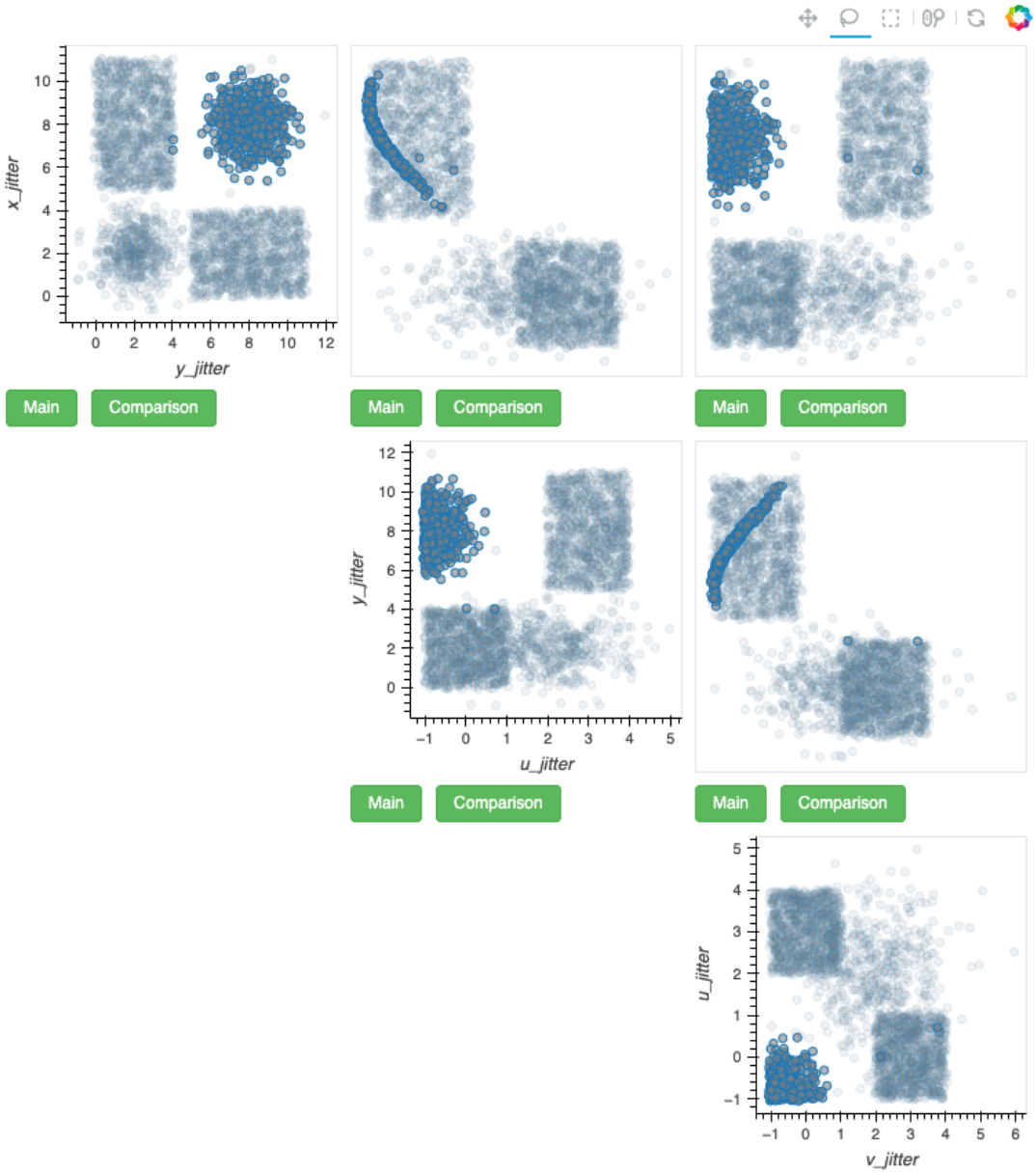


Figure 1.4: Dataset 2 (SPLOM):Brushing the right cluster reveals the dependency in the v, y and u, x plots

So it is certainly possible in this case but can be very time-consuming, especially if there are many dimensions and the SPLOM becomes more complex.

Another issue in cluster detection is topological preservation. There are many libraries and easy-to-implement algorithms which detect clusters mathematically and assign distinct colors to points from different clusters. However, these colors, while distinct, do not tell us anything about how close or far away these clusters are to each other. It would be beneficial if clusters which are close to each other have a color which is more similar to each other than two clusters which are further apart.

The goal of this thesis is therefore to develop a visual analysis approach addressing these issues by reducing manual effort and cognitive load during multi-dimensional data analysis tasks. Specifically, the proposed approach aims at automating cluster detection and identification of functional dependencies across multiple dimensions while preserving topological relationships between clusters through appropriate color assignments.

To achieve this goal, this thesis proposes a visual analysis interface usable within a Jupyter Notebook environment that provides the following functionalities:

1. Automatic identification and clusters coloring for a given view and propagation of this color to their respective points in the rest of the SPLOM,
2. enabling the interactive optimization of color assignments for points, emphasizing cluster structures and maximizing color utilization from a defined color space,
3. allowing users to interactively select subsets of identified clusters and rerun clustering algorithms specifically on these subsets,
4. facilitating the identification of functional dependencies between features without requiring exhaustive scanning through the entire SPLOM, especially in noisy scenarios.
5. preservation of topological relationships within normalized data spaces through careful color assignment strategies,
6. reduction of exhaustive manual exploration and increased precision in detecting clusters and dependencies compared to traditional brushing and linking methods.

What follows is an explanation of the basic concepts of multi-dimensional data representation and visual analysis. Afterward, a brief review of related work in the literature will be undertaken, and it will be shown why these approaches won't satisfy the requirements for the proposed interface. Thereafter, the proposed interface, which can maximize colors being used from a color map while preserving its topology, will be described by its functionalities from a high-level perspective.

To evaluate how effectively our visual analysis approach meets these goals, several qualitative use cases with synthetic datasets are conducted to demonstrate improvements over traditional SPLOMs in terms of accuracy, cognitive load reduction, and interaction efficiency. Additionally, quantitative evaluations assess computational performance and scalability limits for practical usability.

2 Background

Since this thesis aims to make multi-dimensional cluster detection for several data distributions easier, it is important to first look at some underlying concepts which are important to understand the current related work.

2.1 Multi-Dimensional Data Representation

Multi-dimensional datasets, characterized by more than three attributes in either the domain or range, present unique challenges in visualization and analysis [LMW+17]. The increasing complexity of data generated from massive simulations, sensing devices, and advanced computing resources has necessitated the development of sophisticated visualization techniques [LMW+17].

One effective method for visualizing multi-dimensional data is the scatterplot matrix (SPLOM), which has gained significant attention in the field of data visualization [Mun14]. Scatterplot matrices, also known as trellis displays, were introduced by Richard Becker et al. in 1996 as a framework for visualizing multi-dimensional data. A SPLOM "is a well-established visualization technique that organizes scatter plots into a grid or matrix format to explore pairwise relationships between variables in multidimensional data sets" [LAE+12]. A SPLOM consists of a grid where each cell represents a scatter plot of two variables. For a dataset with n numerical variables, the matrix will have n^2 cells, with the diagonal typically showing the distribution of a single variable.

SPLOMs provide a comprehensive overview of all pairwise relationships in a dataset, allowing for the identification of patterns, correlations, and outliers. However, SPLOMs are not easily scalable. While effective for moderate dimensionality, SPLOMs can become cluttered and difficult to interpret as the number of dimensions increases [LMW+17]. Using color, opacity, and related preattentive features to represent additional variables can increase the information density of SPLOMs [LMW+17]. One way to add these features is by an interaction method called Brushing and linking.

2.2 Brushing and linking

Brushing and linking was introduced by Becker and Cleveland in 1987 as an interactive method for exploring multivariate data [War09]. This process selects (brushes) data points in one plot, which then highlights (links) the matching data in the other plots [War09].
There are several ways to brush data, with the simplest way being a sweeping across data points with the mouse. In this case, the sweeping motion can form a rectangle or the user draws a custom shape around the points, sometimes also called lasso selection [War09].

Linking is the process of highlighting corresponding data points across multiple visualizations. This connection allows users to observe relationships between different dimensions or representations of the data. The highlighting itself can be done by many preattentive mechanisms, such as a distinct color or a change in opacity [War09]. The effects of brushing and linking can be observed in detail in 1. In these examples, it becomes apparent that applying brushing and linking in a SPLOM is a powerful tool for visual filtering and pattern recognition in multi-dimensional data analysis.

3 Related Work

Visually detecting multi-dimensional clusters is a challenge that has produced several attempted solutions in the past.

Elmqvist et al.'s "Rolling the Dice" interface [EDF08] represents a significant advancement in the visual exploration of multi-dimensional datasets using SPLOMs. Their approach addresses several key challenges in multidimensional data analysis through various innovative features. The interface provides multiple ways for users to navigate through the data space, including the ability to reorder the SPLOM, which can help in identifying patterns or relationships that might not be apparent in a fixed arrangement. This reordering functionality is particularly useful for grouping related dimensions together, potentially revealing clusters or trends across multiple attributes.

One of the most innovative features is the use of 3D animated transitions when switching axes in a 2D plot. This animation creates a smooth, intuitive visual link between different 2D projections within the SPLOM, helping users maintain a mental model of the data's structure across different dimensional views. This potentially aids in the identification of complex, multi-dimensional patterns or clusters.

The "Query Sculpting" feature represents an advanced implementation of brushing and linking techniques. It allows users to select and highlight points of interest, save these selections as persistent "queries", assign distinct colors to different queries, and adjust the visibility of non-selected points. This functionality enables users to build up a complex understanding of the data by iteratively selecting and comparing different subsets.

Additionally, the interface includes algorithms for discovering correlations between dimensions. This feature can automatically reorder the rows of the SPLOM based on these correlations, potentially revealing relationships or structures in the data that might not be immediately apparent in a randomly ordered SPLOM.

While the range of interactions offered by Elmqvist et al.'s interface is impressive and useful for many use cases, it still has some limitations. Despite its advanced features, the interface still relies heavily on manual exploration. Users must actively navigate through the data space, create queries, and interpret the results themselves. The wealth of features and interaction possibilities, while powerful, may also increase the cognitive load on users, potentially making it challenging for novice users or in time-sensitive analysis scenarios. The effectiveness of manual exploration and visual interpretation may decrease as the dimensionality and size of the dataset increase, potentially limiting its applicability to very multi-dimensional or large-scale datasets. Furthermore, the interface does not incorporate automatic pattern detection or clustering algorithms, which could potentially speed up the exploration process, especially for large or complex datasets.

In contrast to Elmqvist et al.'s approach, our proposed interface introduces automated cluster detection and color assignment strategies rather than relying exclusively on user-driven queries and manual selections. While their method provides sophisticated manual exploration tools through interactive navigation and query sculpting, our approach aims at reducing manual effort by automatically identifying clusters with Mean Shift clustering and assigning colors based on cluster centers' positions in a color map. Moreover, while Elmqvist et al.'s system can reorder dimensions based on detected correlations to facilitate exploration, our approach specifically targets the identification of functional dependencies across views through topology-preserving color assignments using SOMs. Thus, our method explicitly addresses issues such as noise occlusion and cognitive load reduction by providing automatic visual cues rather than requiring extensive manual interaction or prior knowledge from users.

Chegini et al.'s work [CSG+18] presents an advanced interface for pattern detection in scatterplot matrices (SPLOMs). Their approach focuses on enabling users to interactively explore and identify patterns across multiple dimensions. The core functionality of their interface allows users to select a set of points within a specified bounding box in a scatter plot. Once a selection is made, the system calculates similarities based on the distance between feature vectors derived from descriptors. This similarity calculation incorporates a weighted combination of distance metrics and minimum thresholds for pattern matching quality, providing a flexible framework for defining what constitutes a similar pattern.

The key advantage of this approach is its ability to help users check for the existence of clusters and correlations across multiple dimensions by simply selecting interesting patterns in a single view. This significantly reduces the need for exhaustive manual exploration of the entire SPLOM, as patterns identified in one plot can be automatically searched for in other dimensions. The interface thus offers a more efficient way to explore multi-dimensional data compared to traditional SPLOM navigation techniques.

However, despite its sophisticated design and undeniable ease of use, Chegini et al.'s interface has several limitations when compared to the requirements set for the proposed thesis work. Firstly, while it reduces manual exploration, it still relies heavily on the user's ability to identify initial patterns of interest. This requires the user to have some prior knowledge or intuition about what to look for, which may not always be possible, especially in exploratory data analysis scenarios. Secondly, the presence of noise in the data can significantly hinder the user's ability to spot patterns in the first place. Occlusion caused by noisy data points may obscure important structures, making them difficult to select and, consequently, to find in other dimensions.

Another limitation of the interface is its apparent lack of tools for drilling down further within spotted patterns. Once a pattern is identified and similar patterns are found in other dimensions, the interface does not seem to provide robust mechanisms for exploring subsets of these patterns in greater detail. This limitation could make it challenging to uncover more nuanced or nested patterns within the data.

In contrast, our proposed approach differs from Chegini et al.'s method primarily by focusing on automatic cluster detection rather than relying on user-driven selections of local patterns. Our approach leverages Mean Shift clustering to automatically detect clusters within selected views and assigns colors based on cluster centers' positions in a color map. Unlike Chegini et al.'s method,

which depends on user-defined pattern selections and iterative relevance feedback, our approach aims to automate initial cluster detection and coloring, thereby reducing reliance on prior knowledge or initial user input.

In conclusion, while Chegini et al.'s work represents a significant advancement in pattern detection for SPLOMs, it still leaves room for improvement in terms of automation, noise handling, and detailed exploration of identified patterns. As will be demonstrated in the evaluation chapter, these limitations may make some patterns very difficult to spot, especially in complex or noisy datasets.

Koytek et al.'s MyBrush [KPV+17] represents a significant advancement in brushing and linking techniques for coordinated multiple views (CMV). The core innovation of MyBrush lies in its incorporation of personal agency, allowing users unprecedented control and flexibility over brushing and linking interactions.

MyBrush deconstructs the brushing and linking process into three configurable components: source, link, and target. Users can manipulate these components through local menus, providing a high degree of customization. The source specifies the data being brushed, the link manages how relationships between the source and target are expressed, and the target displays the elements related to the source [KPV+17].

This granular control enables users to tailor visual attributes of the source, customize link expressions (e.g., through color, lines, or animation), and modify target display methods (e.g., highlighting or de-aggregation). Such flexibility is particularly valuable when exploring multi-dimensional data, where clusters and dependencies may only become apparent through specific dimensional combinations or visual representations. By allowing users to interactively explore various projections and visual mappings, MyBrush facilitates the discovery of hidden data structures that might otherwise remain obscured.

In contrast to Koytek et al.'s MyBrush interface, our proposed approach emphasizes automation rather than extensive manual configuration. While MyBrush provides powerful customization capabilities for brushing interactions, it still requires users to manually define brushes and interpret relationships between data points across views. Our interface instead focuses on the automatic identification of clusters using Mean Shift clustering and optimized color assignment using SOMs, significantly reducing manual effort in initial exploration phases.

While MyBrush significantly enhances user control in brushing and linking interactions, it still leaves room for improvement in terms of automation and guidance in pattern detection. The development of an interface that combines MyBrush's flexibility with more automated pattern detection and analysis capabilities could further enhance the effectiveness of multi-dimensional data exploration.

4 Method

To solve these shortcomings, a new interface will be developed which is further described in this chapter. The description will largely be structured by functionality. However, before these functionalities are explained, it is first important to understand the two algorithms and color maps being used and why they are particularly suited for the task.

4.1 Clustering Algorithms

4.1.1 Mean Shift Algorithm

The Mean Shift Algorithm is a powerful non-parametric technique for data analysis and clustering. Originally proposed by Fukunaga and Hostetler in 1975, it has gained significant attention in various fields, including computer vision, image processing, and pattern recognition [CLC22].
It identifies modes in the underlying probability distribution of data points by iteratively shifting data points towards regions of higher density, effectively discovering clusters in the data [Car15]. The algorithm's core principle is based on kernel density estimation, where each data point contributes to the overall density estimate [Car15].

The update equation for point iterations in the standard Mean Shift is given by:

$$m(x) = \frac{\sum_{i=1}^{n} K_h(x_i - x)x_i}{\sum_{i=1}^{n} K_h(x_i - x)} \tag{4.1}$$

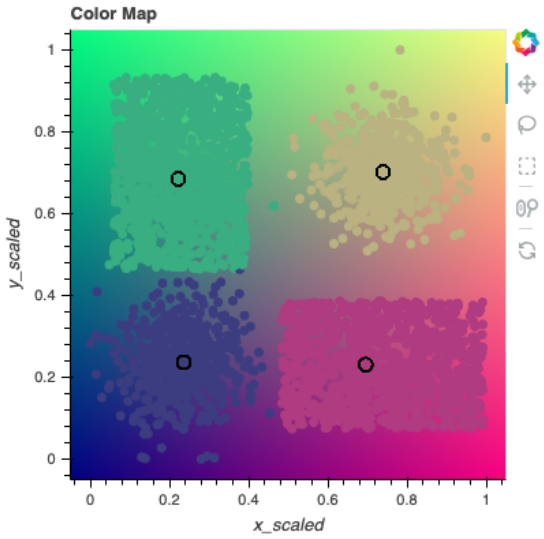
where:

$m(x)$	is the mean shift vector
x_i	are the data points in the feature space
$K_h(x)$	is the kernel function with bandwidth h
h	is the bandwidth parameter
n	is the number of data points

The kernel function K_h is typically defined as:

$$K_h(x) = \frac{1}{h^d} K(\frac{\|x\|^2}{h^2}) \tag{4.2}$$

where d is the dimension of the feature space. [CLC22]



h

Figure 4.1: Color Map Mean Shift (Dataset 2): Colors of the cluster are based on the color coordinates of the cluster centers (black circles)

One of the strengths of the algorithm is its ability to detect clusters which can take on arbitrary shapes and not just spherical clusters as in K-means clustering. So conceptually speaking, we want to leverage this ability and combine it with one of the color spaces discussed in 4.2. This will allow us to propagate a density-based and topology preserving coloring across all scatterplots, which does not rely on specific cluster shapes.

However, to implement this, the Mean Shift algorithm still requires the bandwidth parameter to be set. There are many heuristics like Scott's or Silverman's rule to select the value of this parameter [Car15]. Since these rules don't take into account varying densities across the dataset, the bandwidth will be selected by a "quantile of pairwise distances" heuristic. This approach first computes the maximum distance for every point to its 0.3 * N nearest neighbors by Euclidean distance, where N refers to the number of samples and 0.3 is a quantile parameter which can be modified. The estimated bandwidth is the average of these maximum distances. While the calculation of all these pairwise distances can be computationally expensive, it is possible to mitigate this issue by subsampling the data.

While this heuristic is useful for an initial clustering, it is not perfect and the user may want to increase or decrease the amount of identified and colored clusters further. Since the value of the bandwidth parameter doesn't directly tell the user how many clusters it will produce, bandwidth thresholds for all cluster amounts between one and ten will be calculated. This thresholds calculation is further explained in the Implementation chapter.

In order to speed up the Mean Shift algorithm, the seeds x of the Kernel functions do not represent the exact locations of all points, but instead correspond to the locations of a discretized version of the points. In this process, the points are grouped into bins on a grid, with the level of granularity determined by the bandwidth. This somewhat reduces the accuracy of Mean Shift but leads to a significant performance improvement.
In the interface, the Mean Shift algorithm can be applied on a normalized 2D set of the data. The

determined cluster centers are then applied to a color map which also lies in the normalized 2D space. This determines the color for all points which the algorithm associates with that cluster. An example is shown in Figure 4.1.

4.1.2 Self Organizing Maps

Self-Organizing Maps (SOMs), also known as Kohonen maps, are a type of artificial neural network introduced by Finnish professor Teuvo Kohonen in the early 1980s [CORV18]. SOMs are designed to perform unsupervised learning, primarily for dimensionality reduction and data visualization purposes [Yin08].

The development of SOMs was inspired by the topographic mapping observed in the cerebral cortex. This biological basis aims to model how sensory inputs are processed and organized in the brain, particularly in visual perception. SOMs operate on the principle of competitive learning, where neurons compete to respond to input stimuli, resulting in a self-organized representation of the input space [Yin08].

SOMs can represent multi-dimensional input data in a lower-dimensional (typically 2D) space while preserving topological relationships. During training, it maintains the relative distance between data points, which ensures that similar inputs activate neighboring neurons in the output space [CORV18].
The algorithm starts by initializing weight vectors for each neuron in the map which is present in the data space to train on. Then the Best Matching Unit (BMU) for each data point is identified based on the Euclidean distance to each weight vector. These weights are then updated by the formula:

$$W_v(s+1) = W_v(s) + \theta(v,t)\alpha(s)[D(t) - W_v(s)] \tag{4.3}$$

where:

$W_v(s)$	is the weight vector of neuron v at step s
$\theta(v,t)$	is the neighborhood function around winning neuron t
$\alpha(s)$	is the learning rate at step s
$D(t)$	is the input vector

The neighborhood function $\theta(v,t)$ is typically defined as:

$$\theta(v,t) = \exp\left(-\frac{\|r_v - r_t\|^2}{2\sigma^2(s)}\right) \tag{4.4}$$

where:

r_v	is the position of neuron v in the grid
r_t	is the position of the winning neuron t
$\sigma(s)$	is the neighborhood radius at step s

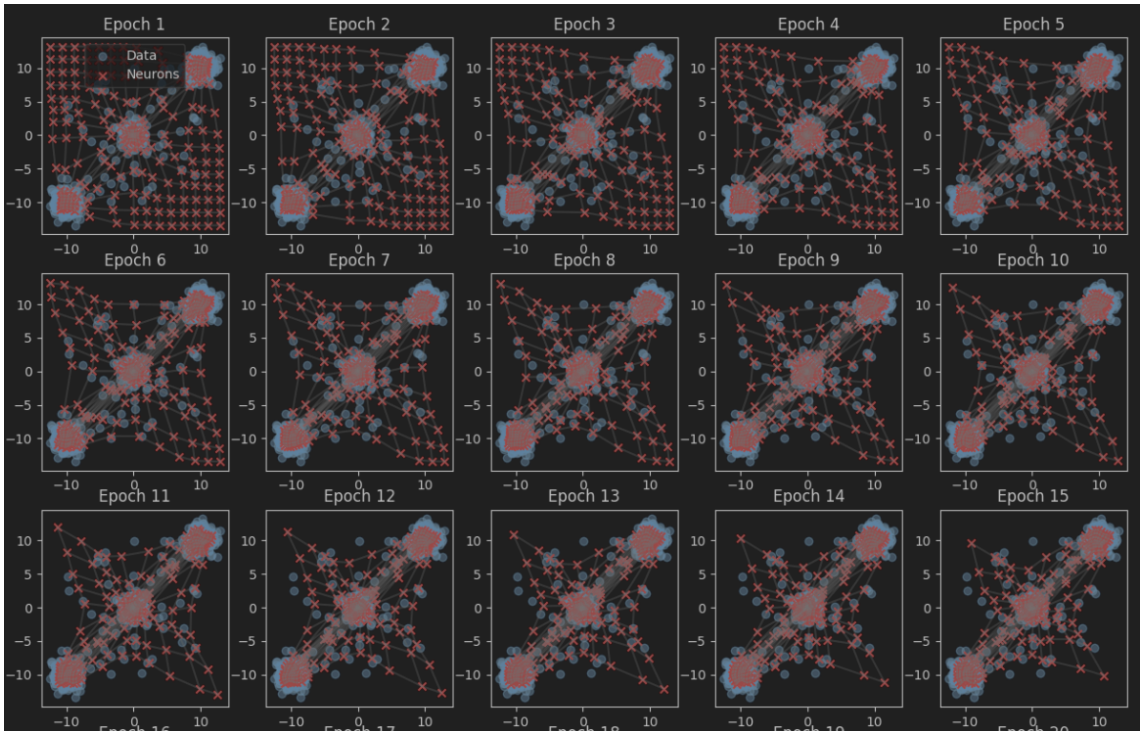


Figure 4.2: Weight movement SOM for different epochs: The first epochs training the SOM. In the beginning, the weights collapse quickly to bring colors, which are relatively far away from clusters, closer to the data. Then, updates are done less aggressively and the weights spread out more within clusters.

Conceptually speaking, the goal with implementing the SOM is to maximize the utilization of the color map and expose functional dependencies by leveraging color gradients within clusters. While Mean Shift applies the color space only based on the cluster centers themselves, SOM assigns colors for each point individually (relative to local density). The weights are in this case a representation of the color space.

For the implementation itself, the initial configuration of weights is a 15x15 grid of neurons, which is scaled to the 2D data space. This initial grid represents a downsampled color map, so each weight is not just a vector but also represents a color. The key advantage of SOM lies in its ability to mainly preserve the topology of the initial configuration of weights. This means that given the right parameter settings, the initial grid layout (with "lines" connecting neighboring weights) will be transformed in a way that minimizes crossings of these lines. To ensure a minimization of these crossings or topology violations, the parameters for SOM are dynamically set according to the following phases:

1. in the first phase where more than 95 percent of the weights are outside of the convex hull of the data, updates are relatively aggressive
2. once most weights are within the hull, learning is significantly slowed down in order to reduce the risk of topology violations and allow for a gradual increase in the amount of best matching units per cluster.

The exact parameter settings in each of these phases are further explained in the Implementation chapter. Figure 4.2 showcases the movement of weights for the first epochs and applied to the IRIS dataset.
The amount of epochs is also set dynamically. After every epoch, the entropy in regards to the BMU distribution is calculated. This distribution describes the amount of points each weight is a BMU for. Once phase 2 is reached and the SOM has been trained for at least 10 epochs, the standard deviation of entropy for the last 10 epochs is calculated. If the current entropy falls within that standard deviation, training stops since entropy has stabilized. Otherwise, training continues.

4.2 Coloring with Color Maps

Since in this interface, coloring of clusters should be done based on both the similarity between (Mean Shift) and within (SOM) clusters, the question arises which color space should be used.

Bernard et. al. [BSM+15] identified seven relevant tasks for multivariate data analysis and mapped them to six quality assessment measures for 2D colormaps. These are:

1. Finding specific items or locations based on color.
2. Identifying the color of a specific data point.
3. Comparing different data values or ranges based on their colors.
4. Detecting a known group or class within the visualization.
5. Identifying groups of data points by interpreting colored information.
6. Comparing sets or clusters of data points.
7. Comparing elements within sets or clusters.

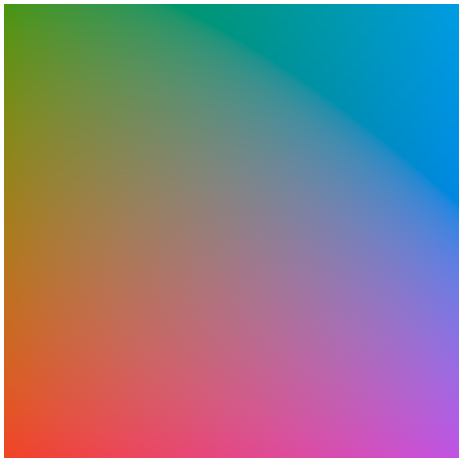


Figure 4.3: Color Map 1

Figure 4.4: Color Map 2

They then evaluated several colormaps based on these measures both for visualizations on a black and white background. In this interface, two of the colormaps with the highest overall score for white backgrounds are used (Figure 4.3 and Figure 4.4).

4.3 Visual Summary

The interface consists of four main elements. The widgets to adjust the settings with a view which shows how points are colored (Figure 4.5), two plots allowing for the comparison between two views (Figure 4.6), and the SPLOM itself (Figure 4.7). It is created with functions from the bokeh library. It is shown in a Jupyter notebook file as a cell output. Therefore, the user can seamlessly integrate the interface into the data exploration and analysis flow which the Jupyter notebook enables.

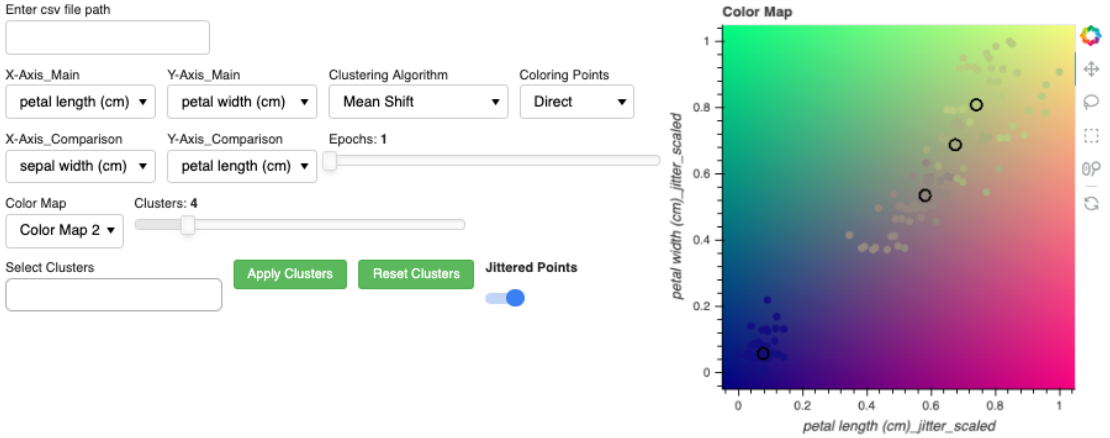


Figure 4.5: Widgets and Colormap view of the interface

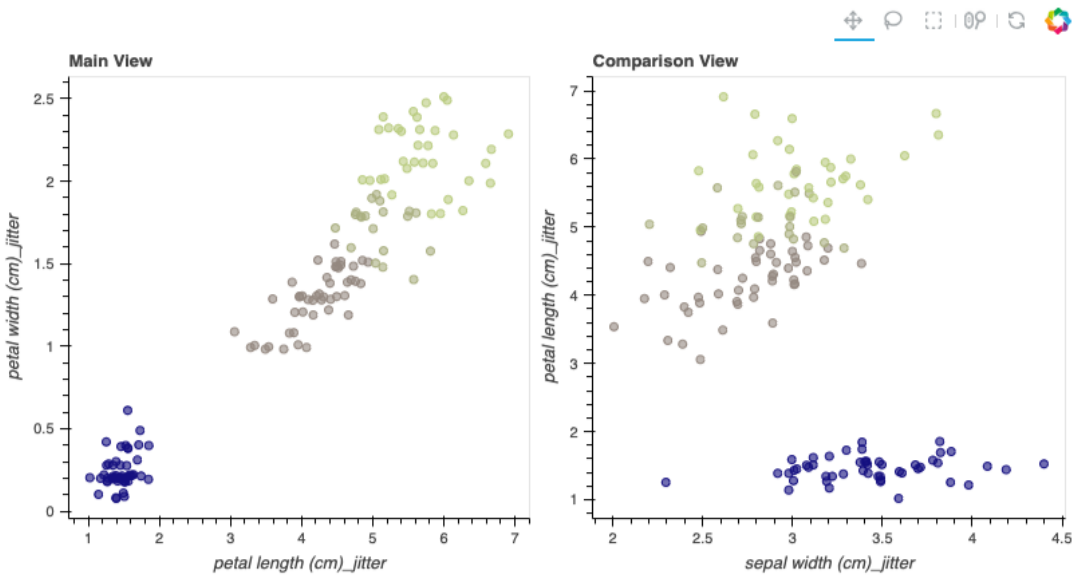


Figure 4.6: Main and Comparison views of the interface

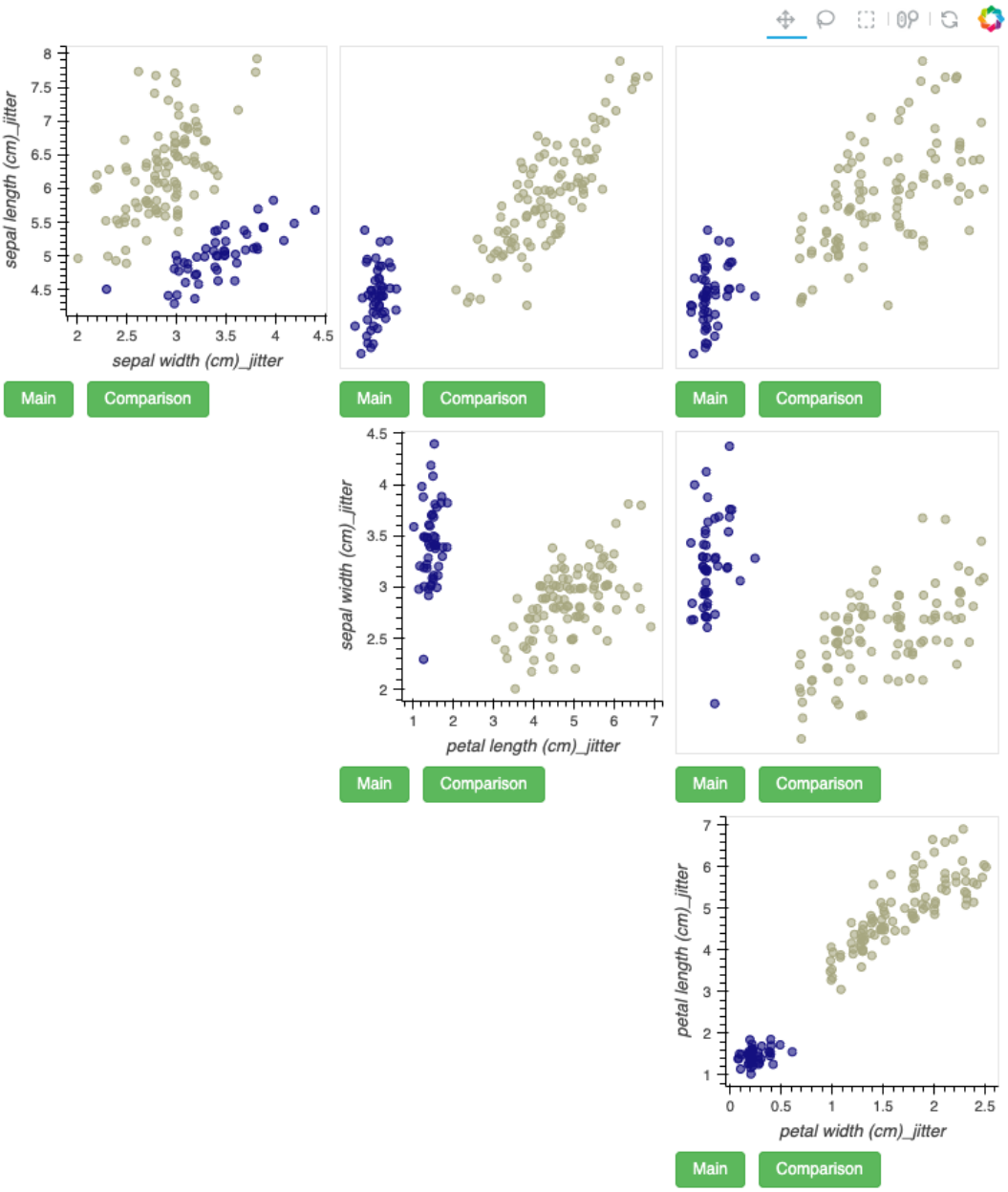


Figure 4.7: The SPLOM of the interface. Individual scatter plots can be displayed as the main or comparison view

4.4 Functions

Several functionalities are implemented in this interface. In this chapter, they will be described from the most basic function, which is also possible with other interfaces, to more sophisticated and unique features.

4.4.1 Basic SPLOM with brushing and linking

The first function, which lays the foundation for more sophisticated use cases, is the ability to read a csv file with numerical features and display these features in a SPLOM. This basic function has already been shown in the introduction (1.2, 1.3) The user can also select the Main and Comparison views through widgets and choose whether or not to “jitter” the displayed points. “Jittering” of points displays the values with a slight noise addition to all features that are normally distributed. This allows the user to better estimate the density of points in data with a low resolution and therefore many duplicates. The exact implementation of this noise is further explained in the Implementation chapter 6.
As shown in the Introduction, the interface also supports brushing and linking interactions, both across the Main/Comparison Views and the SPLOM itself.

4.4.2 Cluster Detection and Selection with Mean Shift

In order to automatically detect and color clusters, the Mean Shift algorithm, as described in 4.1.1 can be applied. The algorithm always runs on the 2D data selected as the Main view. Additionally, the user can use a slider to select the number of clusters to display/color. It is also possible to change the color map.

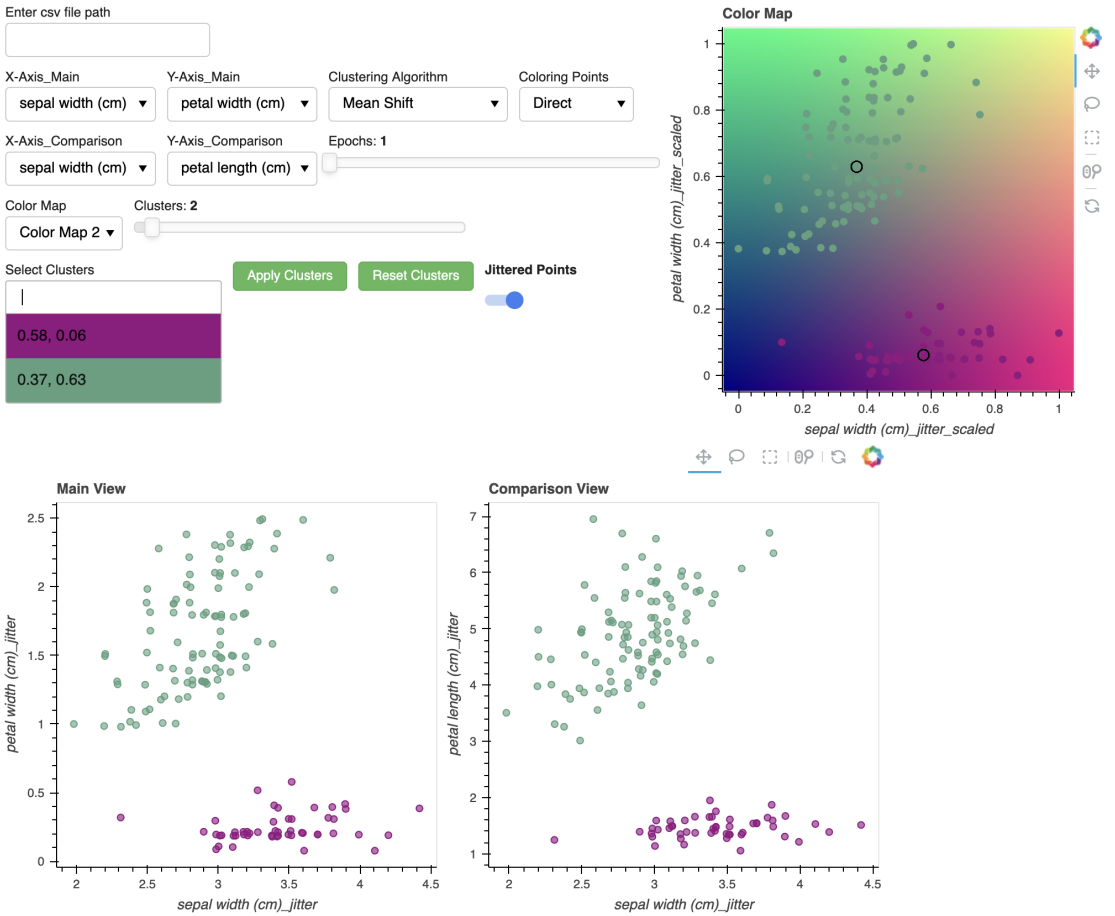


Figure 4.8: Cluster selection interface (start): The dropdown allows for drilling down into the data by cluster color

A key feature with Mean Shift is the ability to filter for a subset of clusters. This can be achieved by selecting a subset of points by colors through a dropdown menu (Figure 4.8).

When applying this selection, the data space for clustering is transformed to the selected subset and clustering is redone. In order to maximise the color space, the subset of data is rescaled to the color map space. While this changes the Color Map view, the deselected points are still visible in the Main/Comparison Views and the SPLOM. However, they are smaller and more transparent (Figure 4.9). This enables the user to preattentively distinguish them from the selected subset.

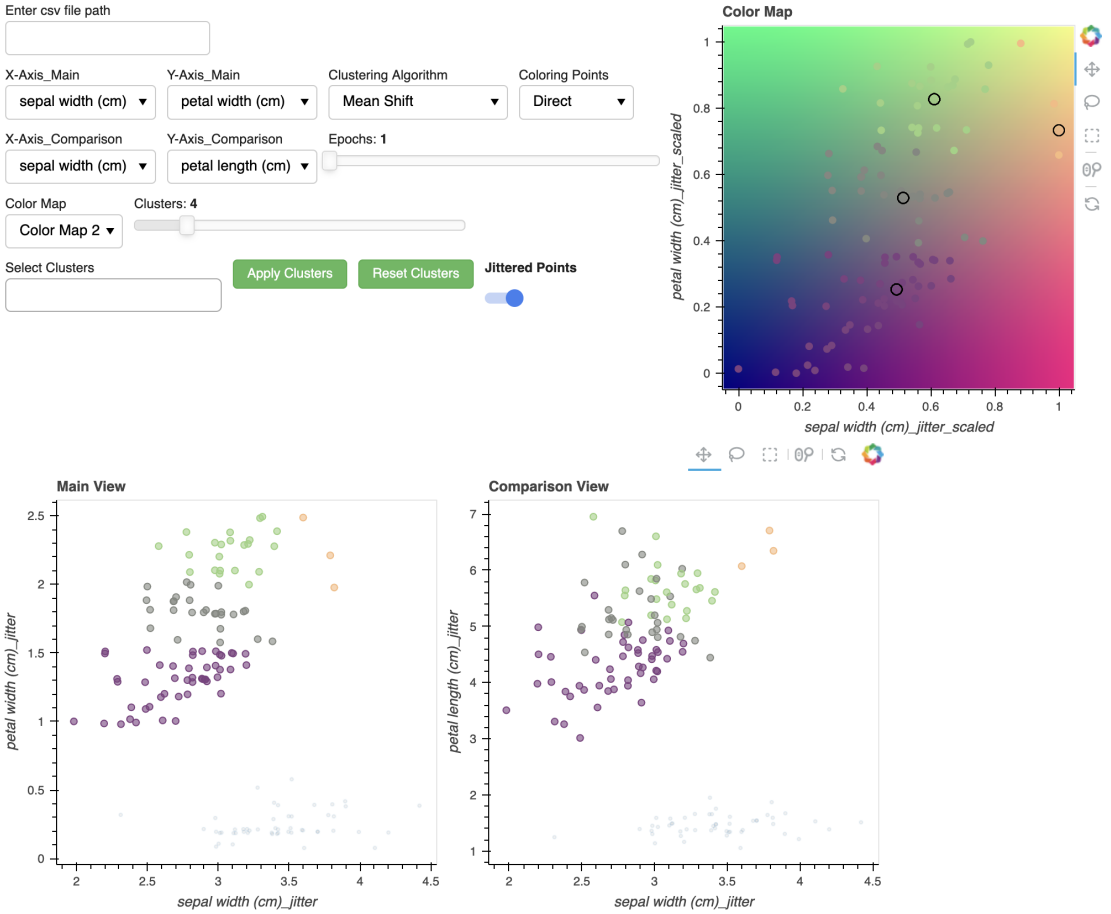


Figure 4.9: Cluster selection interface (end): Selected points are rescaled to the color space and clustering/coloring is done on this subset. Deselected points remain visible.

The user can continue to drill down in this way even further or reset all cluster selections, which recomputes the clustering and coloring for the original data.

4.4.3 Coloring with a Self-Organizing Map

In order to fully make use of the color space, a Self-Organizing Map has to be used. By default, the interface shows a direct coloring of points after the first Epoch. This means that each point is assigned to the color of the weight which is the best matching unit (bmu). It is also possible to use cubic interpolation of multiple weights in the neighborhood by RGB channel values of the colors. A detailed description of this feature follows in 6. Since the amount of weights (15x15) is set for computational efficiency, interpolation of colors may be prudent if the amount of points is significantly higher than the amount of weights, or to smooth out topology distortions or points whose bmu is relatively far away. This coloring over time is showcased in the figures Figure 4.10 to Figure 4.15.

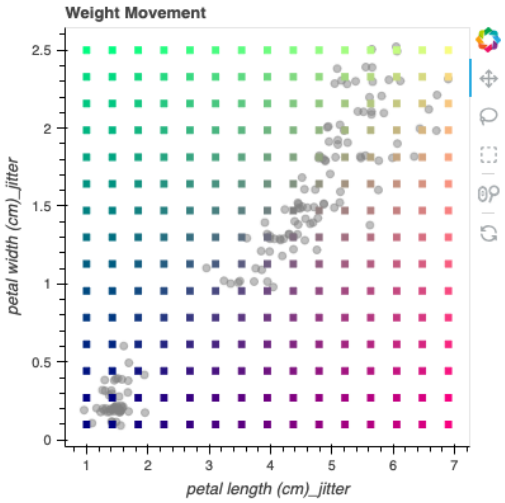


Figure 4.10: SOM initial weights configuration: Weights are colored rectangles and points grey circles. Data is from the IRIS dataset.

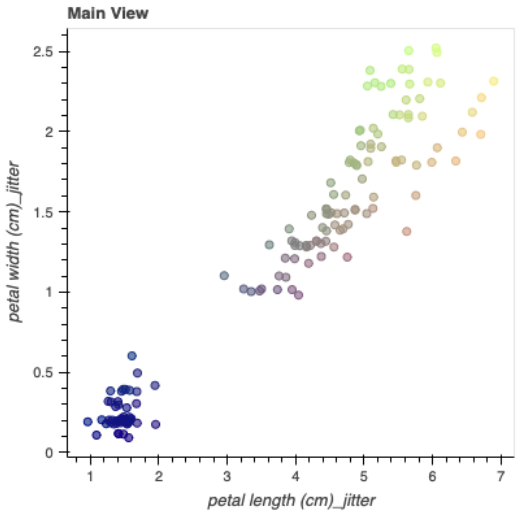


Figure 4.11: SOM IRIS coloring (before training): Coloring before training is equivalent to coloring by nearest neighbor on the downsampled color map.

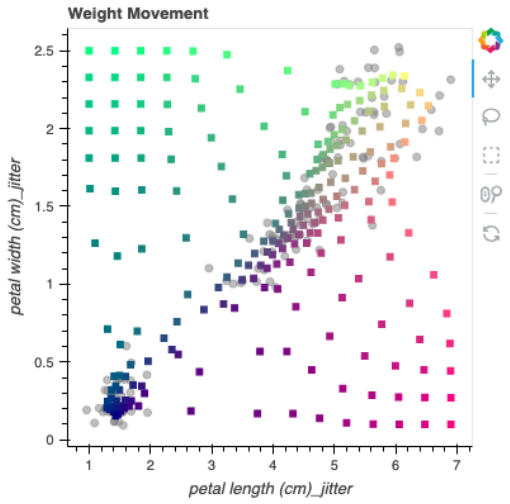


Figure 4.12: SOM IRIS weights (10 epochs): Convergence of weights to the data is slow to minimize topology violations.

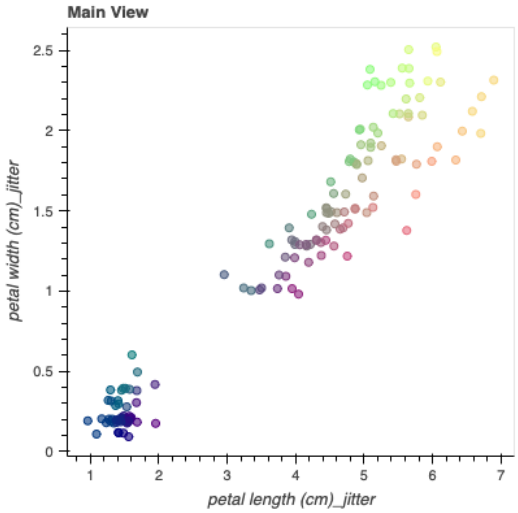


Figure 4.13: SOM IRIS coloring (10 epochs): Coloring becomes more diverse as an increased percentage of the color map becomes a bmu

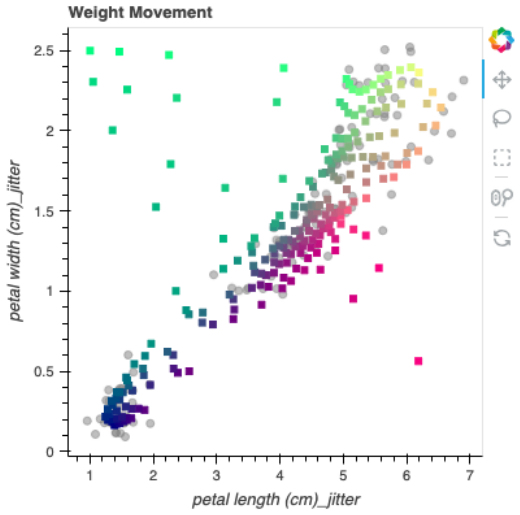


Figure 4.14: SOM IRIS weights (52 epochs): At this point training terminates since the entropy from 4.1.2 stabilizes from this point onwards. This example also shows that parts of the color map can stay far away from the data.

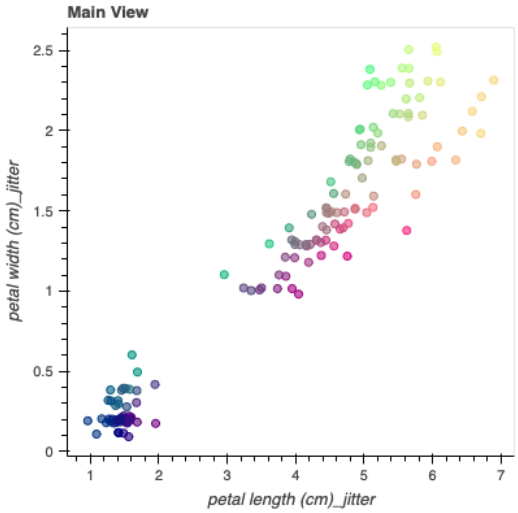


Figure 4.15: SOM IRIS coloring (52 epochs): Coloring at the end of training reaches its maximum diversity. Nearly the entire color map has been mapped onto the data with little topology distortions.

4.4.4 The hybrid use case: combining cluster filtering with SOM

The final key functionality of this interface lies in the possibility of combining the Mean Shift and SOM functionalities. This hybrid use case starts just as the Mean Shift functionality. First, clusters are identified. The user can adjust the amount of clusters as before.

Then, a subset of clusters is selected. This time, however, we want to run SOM and not Mean Shift on this subset. This can be done by selecting SOM as the clustering algorithm after the cluster selection has been applied. As with Mean Shift, the SOM is only trained on this subset and its initial configuration rescales accordingly. This ensures that the usage of the color map space in assigning colors to points is maximized again. This process is shown in Figure 4.16 and Figure 4.17.

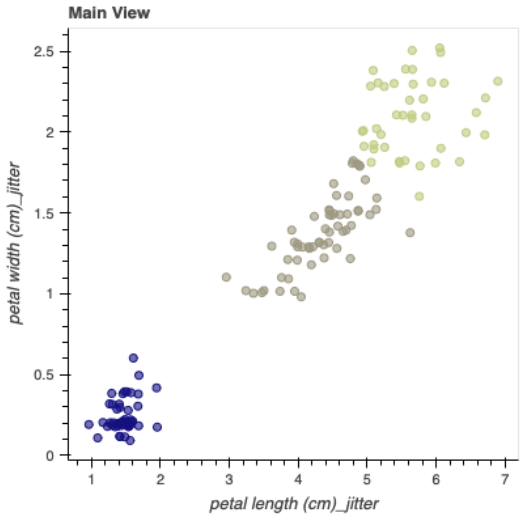


Figure 4.16: Hybrid function for IRIS (start): The user wants to apply SOM on the points from the top two clusters which have been determined with Mean Shift.

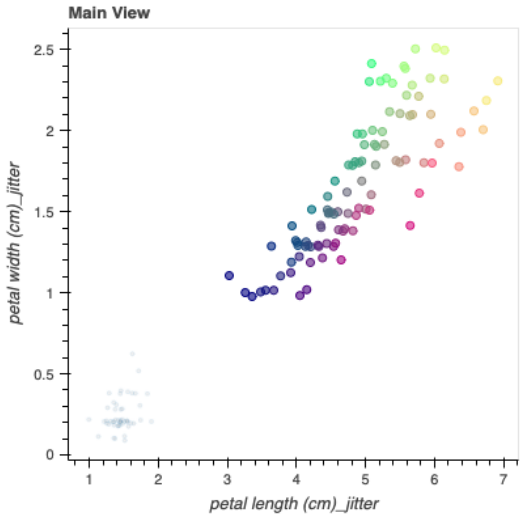


Figure 4.17: Hybrid function for IRIS (end): After applying the cluster selection, SOM is applied on the subset. The deselected points remain visible.

Using the interface in this hybrid way allows the user to leverage the strengths of both the Mean Shift and the SOM functionalities. Deselected points are still visible in the Main/Comparison views and SPLOM, but remain monochromatic and more transparent. This weight movement for this subset is shown in Figure 4.18 and Figure 4.19. As with Mean Shift, a reset of the filter is possible, which applies the SOM to the whole data again.

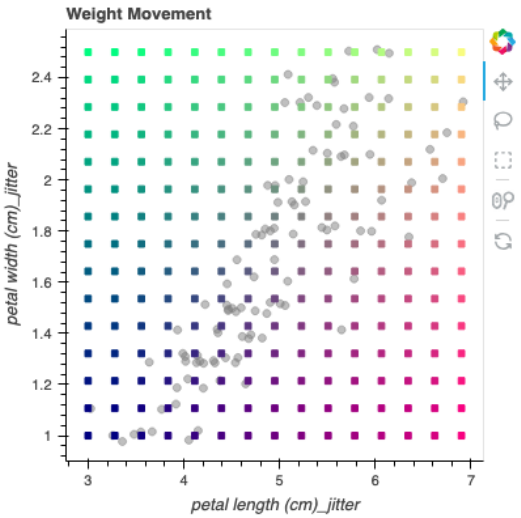


Figure 4.18: Hybrid function initial SOM weights: The initial configuration of weights automatically rescales to the cluster selection.

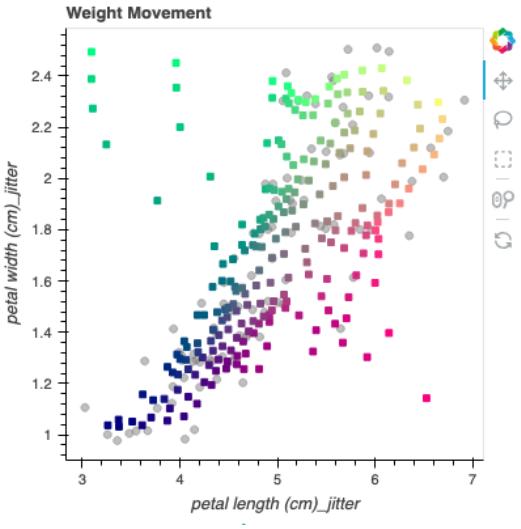


Figure 4.19: Hybrid function SOM weights after training: Usage of the color map is maximized for any subset selection.

5 Evaluation

The Evaluation of this interface is split into two parts. The qualitative and quantitative analysis. In the qualitative analysis, several use cases with unique data distributions are evaluated with the interface. For each use case, we start with the classical data analysis by just linking and brushing monochrome points in the Views and SPLOM. Then we allow the user to work with Mean Shift and attempt a solution. The same procedure is repeated by switching to SOM. In one case, the hybrid approach from 4.4.4 will be employed.

In the quantitative evaluation, the interface will be tested for the runtime of its functions when scaling the data both vertically and horizontally. Since interfaces shouldn't keep the user waiting for too long, it is important to understand the limitations for practical usability in terms of the size of datasets.

5.1 Qualitative Evaluation

In this section, the evaluation metrics will be the following:

1. Can all clusters and functional dependencies with their indices be identified at all and how precise is that identification? An identification of indices can either be done by brushing the points or identify them by color assignment. The fact if the indices is filtered out or not can also be used for identification.
2. How many interactions (mouse clicks/movements) are needed in order to find these patterns? In case the user doesn't know where to look for these patterns, the worst case in terms of the amount of interactions is assumed.

5.1.1 Use Case 1

The distribution for Use Case 1 is the same as in the first example of the Introduction (1.1). In the (x, y) view, two distinct clusters (Cluster A and Cluster B) can be identified, while in the other views, no clear separation between clusters is visible. One of the clusters, however, is further split up into two subclusters B1 and B2. Indeed, the data consists of the three clusters A, B1 and B2 in multi-dimensional space. The assumption is made that the (z, x) view doesn't directly reveal the three clusters due to noise. It offers no advantage to the (y, z) view and is therefore ignored for every approach. Additionally, the goal is to limit lookups in the SPLOM anyway.

Brushing and Linking

As seen in 1, brushing and linking makes it possible to spot the split of the clusters B1 and B2 (1.2) however, a further selection of these subclusters becomes very difficult since they themselves are occluded by cluster A. The data analyst would have to memorize each point of cluster B1 and B2 and then try to only select these in the (y, z) view to get the indices. This approach requires an extensive amount of interactions, if possible at all due to occlusion. The other requirements from the Introduction are also not fulfilled.

Mean Shift

Applying the Mean Shift functionalities makes the task much easier. The clusters in (x, y) are automatically identified and colored (Figure 5.1). The amount of identified clusters may have to be adjusted. This reveals the split in (y, z). Then, the user can filter out cluster A and rerun the clustering algorithm on the (y, z) view (Figure 5.2). Thus, only four to five interactions are needed to identify all three clusters by color assignment (the original color for cluster A remains in the data; it is just not displayed).The amount of interactions is five in the worst case.

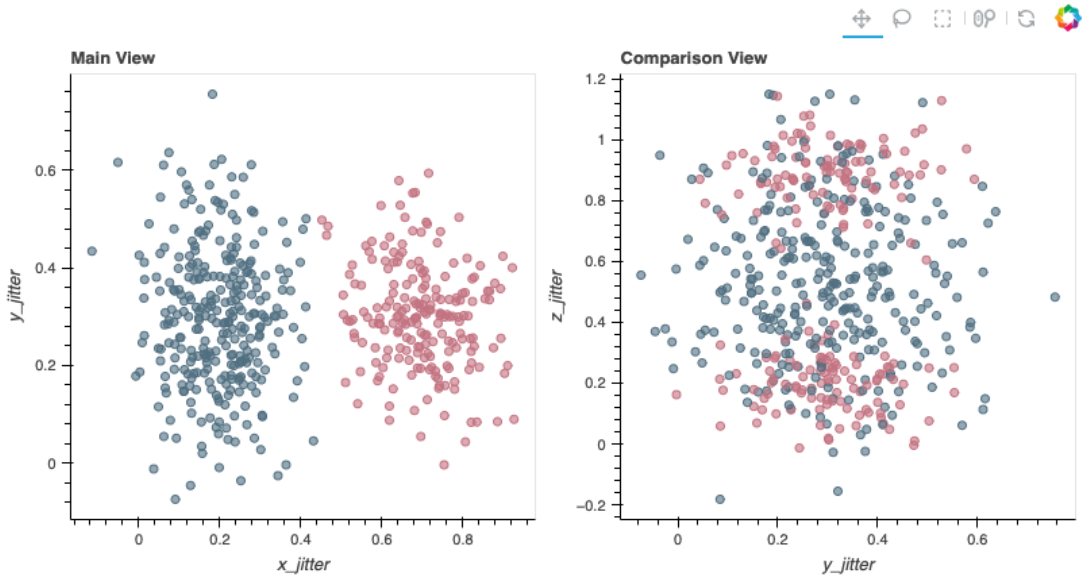


Figure 5.1: Use Case 1 with Mean Shift (start): Mean Shift directly reveals the split between B1 and B2 if two clusters are selected. Cluster A can be filtered out and then Mean Shift applied on (y, z)

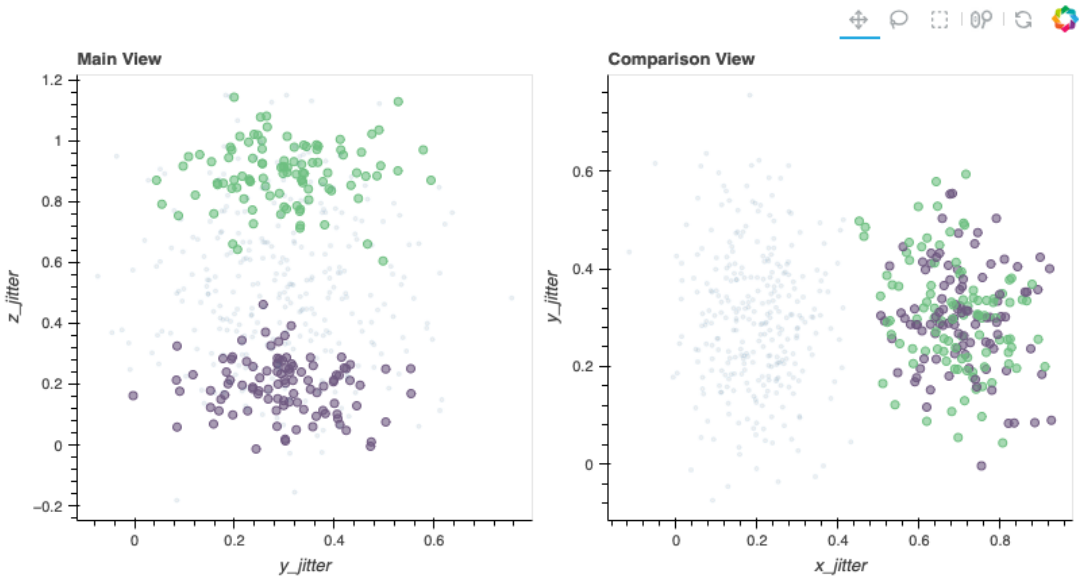


Figure 5.2: Use Case 1 with Mean Shift (end): Running Mean Shift on the filtered new view allows the user to further explore clusters B1 and B2.

SOM/Hybrid Case

Applying SOM by itself (Figure 5.3) doesn't help much for this use case. Since the clusters are relatively spread out over most of the data space in (x, y), the colors within the clusters are very diverse. It is difficult to even spot the split in (y, z) and an identification of the three clusters by color is near impossible. Since there is no filtering possible, the user will clearly fail this task if they choose to only use SOM.

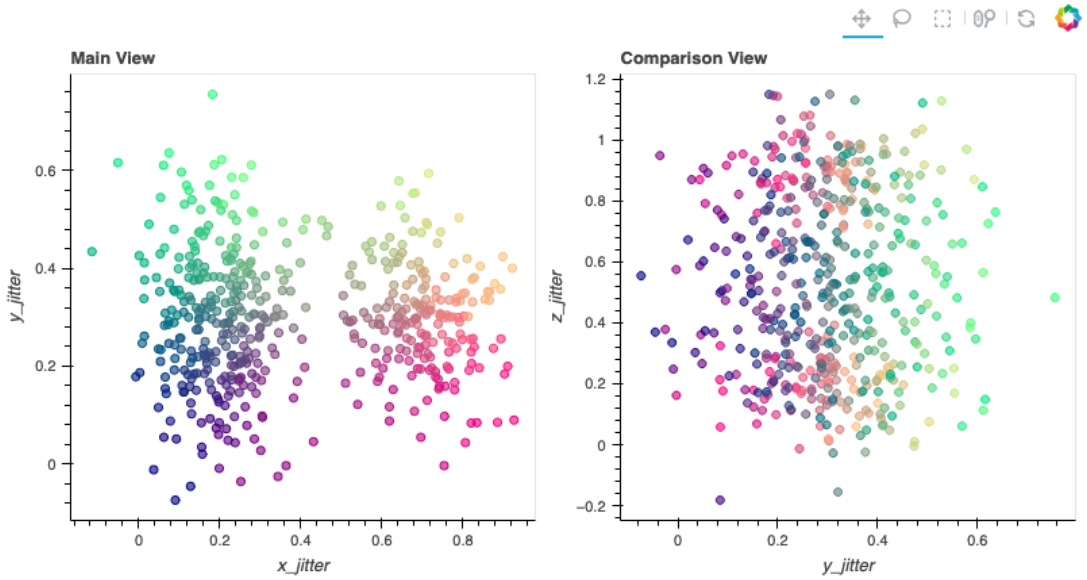


Figure 5.3: Use Case 1 with SOM: The high diversity of colors makes the cluster identification very difficult and a lack of filter options hinders the user from identifying the indices

However, the hybrid function can also be used. In this case, the Mean Shift case is followed until cluster A has been filtered out. Then, SOM is applied on the (y, z) view. Cluster A is identified by the fact that it has been deselected, while cluster B1 is identified by shades of green and yellow colors, and cluster B2 by shades of blue and red colors.

While this maximizes the usage of the color space for the B1 and B2 clusters, it is not very precise since identification is done based on approximate color ranges which may overlap. Precision can be increased by allowing brushing for the subclusters to identify the indices.

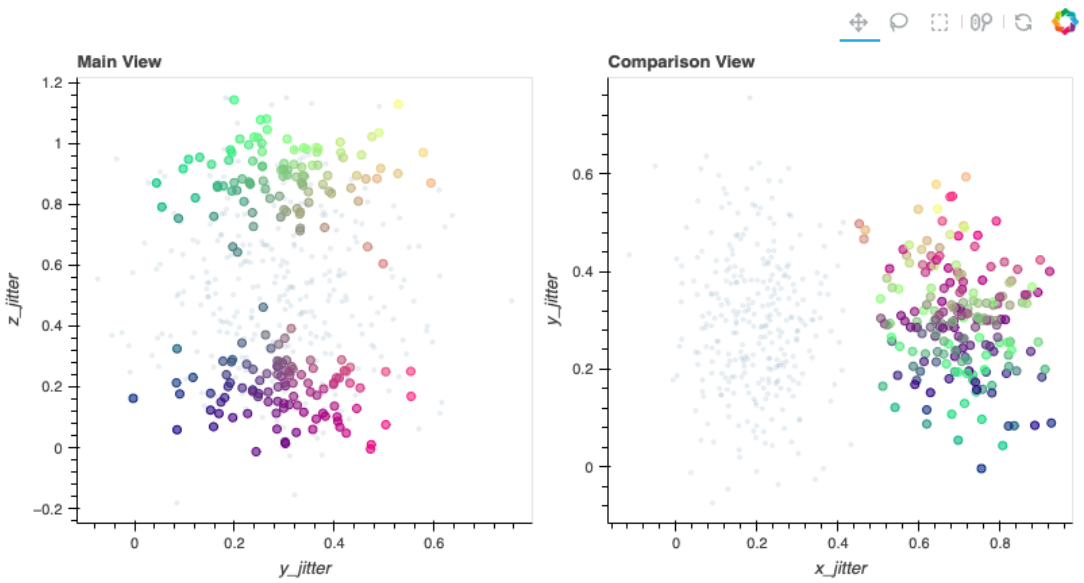


Figure 5.4: Use Case 1 with the hybrid function: While it seems to offer the best of both worlds, the hybrid case lacks precision when compared to Mean Shift.

5.1.2 Use Case 2

The data for the second Use Case has also been described in the Introduction. Here the challenge lies in spotting the cluster in (x, y) with functional dependencies between x, u and y, v. In this example, the whole SPLOM can be used; however, the goal is to minimize lookups in the SPLOM.

Brushing and Linking

As seen in 1.4, the dependency can be spotted when brushing the right cluster and looking for shapes in the SPLOM for the (u, x) and (v, y) plots, which indicate a functional dependency. In the worst case, this requires four brushes and lookups in the SPLOM. Precision is also relatively high, since the clusters are clearly separated in this example.

Mean Shift

Mean Shift doesn't fare better than brushing and linking in this case. It still requires the user to try out selecting one cluster after another and do lookups in the SPLOM for each selection (Figure 5.5). Precision is just as high as in brushing and linking. However, the amount of interactions is higher at eight in the worst case with four lookups in the SPLOM since cluster selections have to be reset before filtering again.

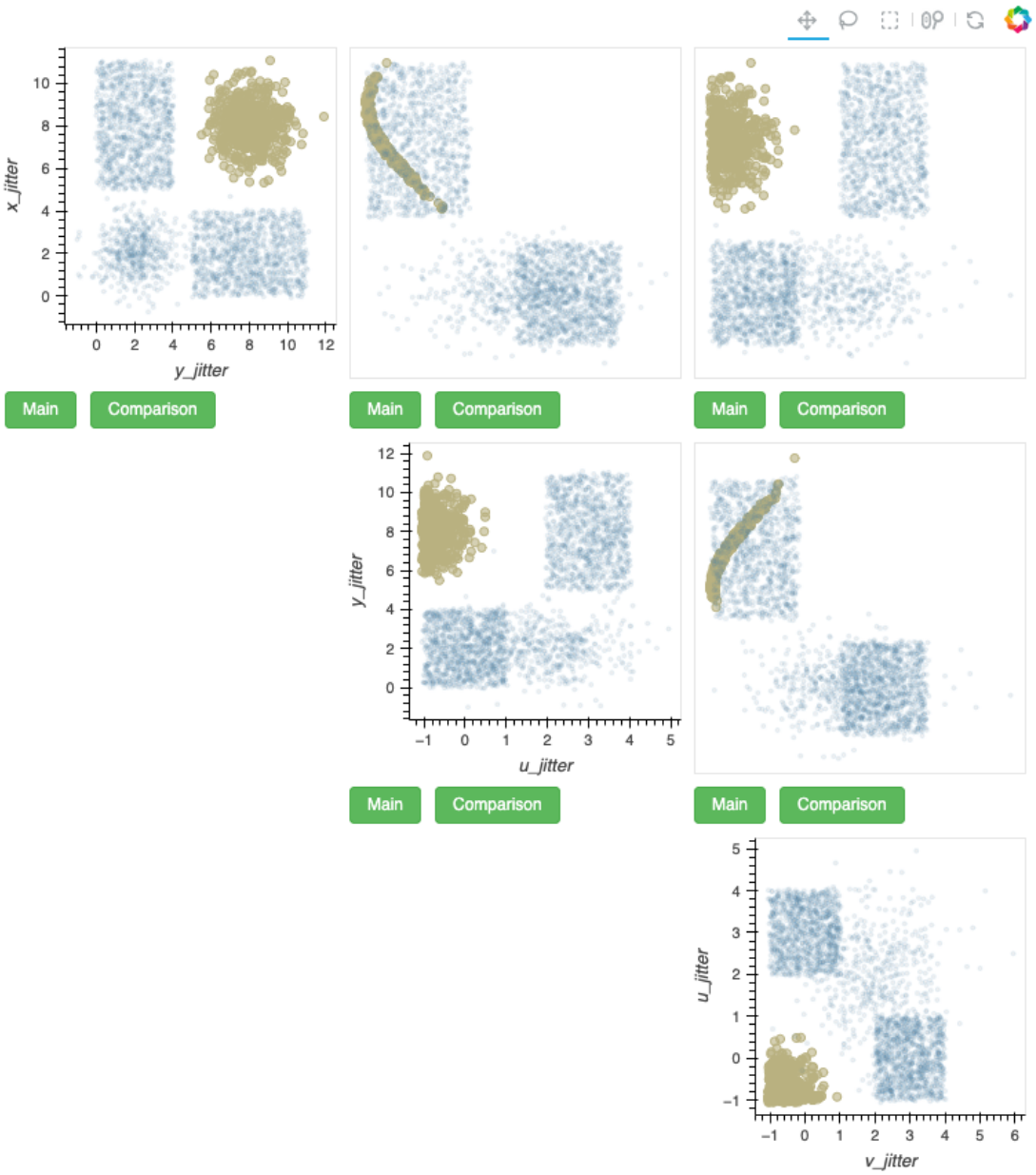


Figure 5.5: Use Case 2 with Mean Shift: Mean Shift offers no advantage over brushing and linking in spotting the functional dependencies.

SOM

Applying SOM offers by far the quickest way to spot the functional dependencies (Figure 5.6). This is because SOM ensures that the color gradient between neighboring points is small (since this is also the case for neighboring pixels in the color map). If a functional dependency from view X to view Y is introduced which is semi-linear, the magnitude of color gradients between neighboring points in Y will remain the same. This is not the case for points without functional dependencies where this color gradient is random. In other words, clusters in Y which are just as "smoothly colored" as in X reveal functional dependencies.

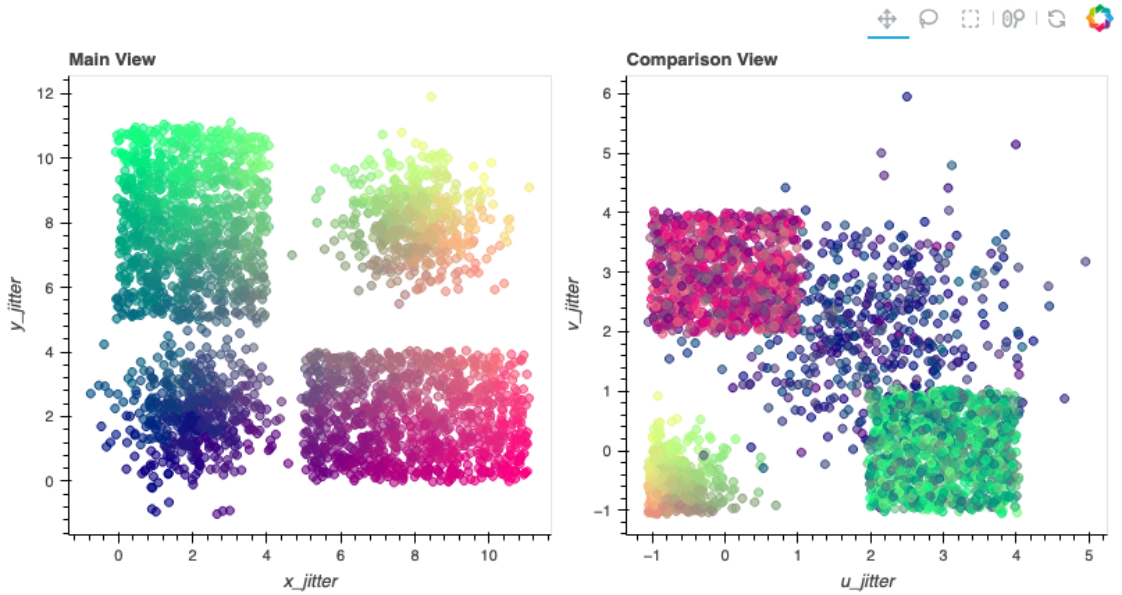


Figure 5.6: Use Case 2 with SOM: The orange/yellow and light green cluster is just as "smoothly colored" in (u, v) as it is in (x, y). This instantly reveals the functional dependencies.

In this case it is not even necessary to perform one interaction or lookup in the SPLOM. Precision is limited compared to the previous methods since only a color range is provided. However, this can again be enhanced by allowing for brushing to select the indices.

5.1.3 Use Case 3

The third use case is similar to the second one. There are the same functional dependencies for one multi-dimensional cluster and none for another cluster in the 4D space. However, there also exists a cluster with a uniform distribution, both in x and y. This "rectangle" cluster is split on its diagonal in the (x, y) view which clearly separates it into two subclusters. In the (u, v) view, these subclusters are nearly identically mapped, but the subclusters are "reattached" by shifting the lower subclusters upwards to form a rectangle in (u, v). Furthermore, all clusters lie on the diagonal of the (x, y) view. This is visualized in Figure 5.7

The goal in this case is to identify not just the functional dependencies, but also these subclusters as distinct in all views. Lookups in the SPLOM should be minimized.

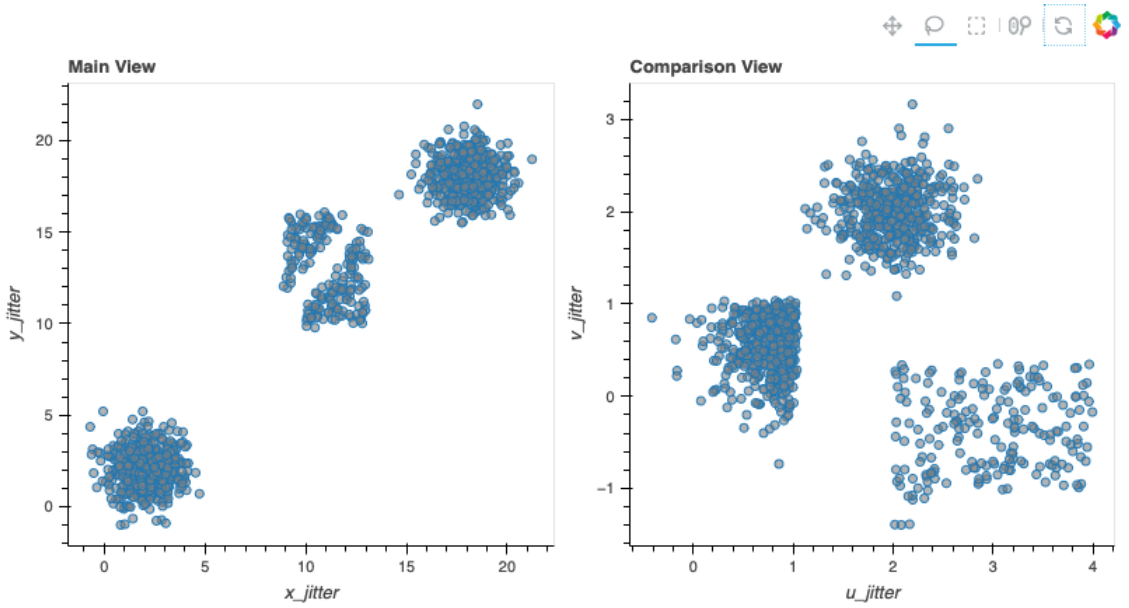


Figure 5.7: Use Case 3 Data: The split rectangle is "reattached" in the (u, v) view.

Brushing and Linking

Employing brushing and linking in this Use Case is capable of spotting the subclusters being reattached in (u, v) as seen in Figure 5.8, but the identification of the functional dependencies requires lookups in the SPLOM again. In the worst case, four interactions and lookups are necessary.

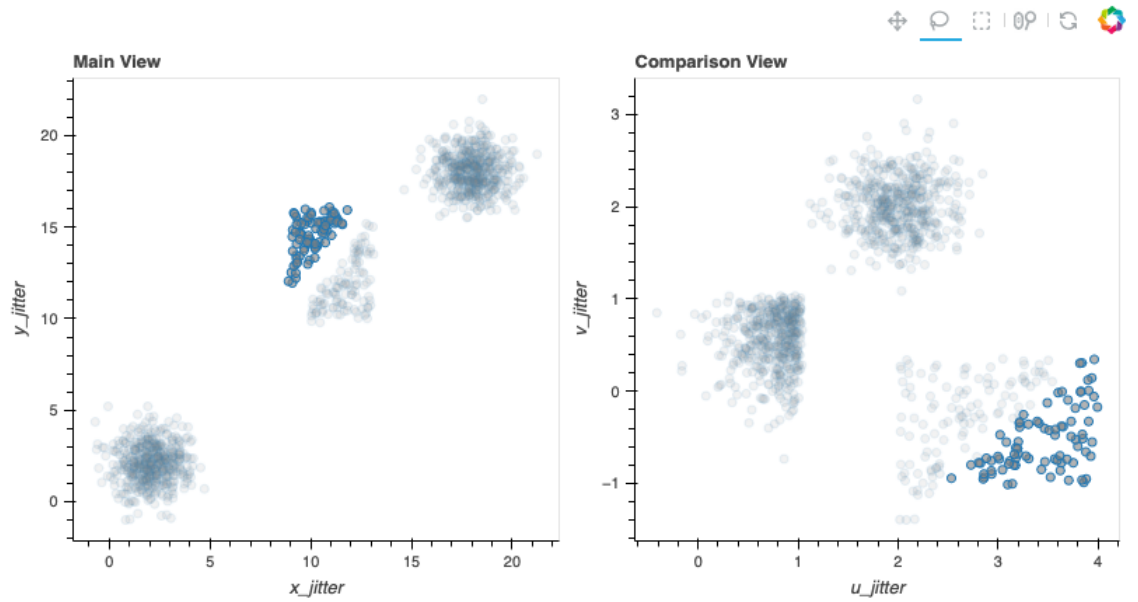


Figure 5.8: Use Case 3 with Brushing and linking: Brushing allows for a quick spotting of the reattached subclusters, but lookups in the SPLOM are still required to spot the functional dependencies.

Mean Shift

Mean Shift offers a significant advantage in assigning distinct colors to both subclusters automatically, if the right cluster amount is selected (Figure 5.9). While this fulfills the task of identifying the clusters, the spotting of functional dependencies is only possible by lookups in the SPLOM. Just as in Use Case 2, Mean Shift requires even more interaction and the same amount of lookups in the worst case as brushing and linking.

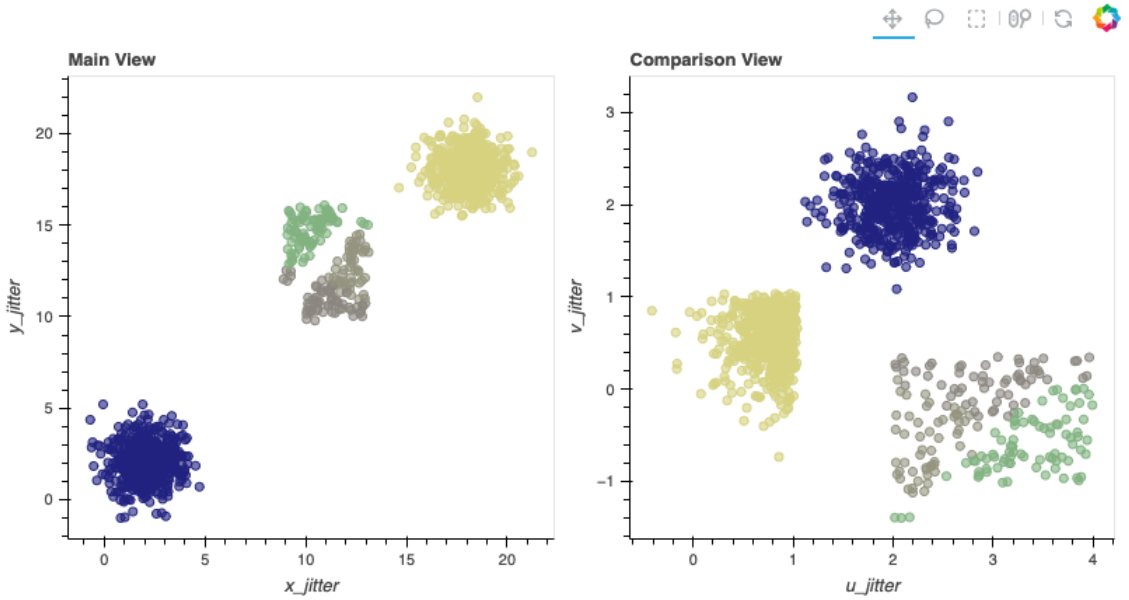


Figure 5.9: Use Case 3 with Mean Shift: Mean Shift allows for a quick discovery of clusters, but offers no advantage in spotting functional dependencies.

Cluster detection also isn't perfect. While Mean Shift is capable of identifying clusters regardless of shape, misclassifications along the edges are still possible.

SOM/Hybrid Case

As in Use Case 2, the application of SOM allows for a quick spotting of the functional dependencies with no additional interactions or lookups required . However, while there is a clear separation between the subclusters, its close proximity doesn't propagate this to the other views. Coupled with the functional dependency, the subclusters in the (u, v) don't seem distinct there anymore.

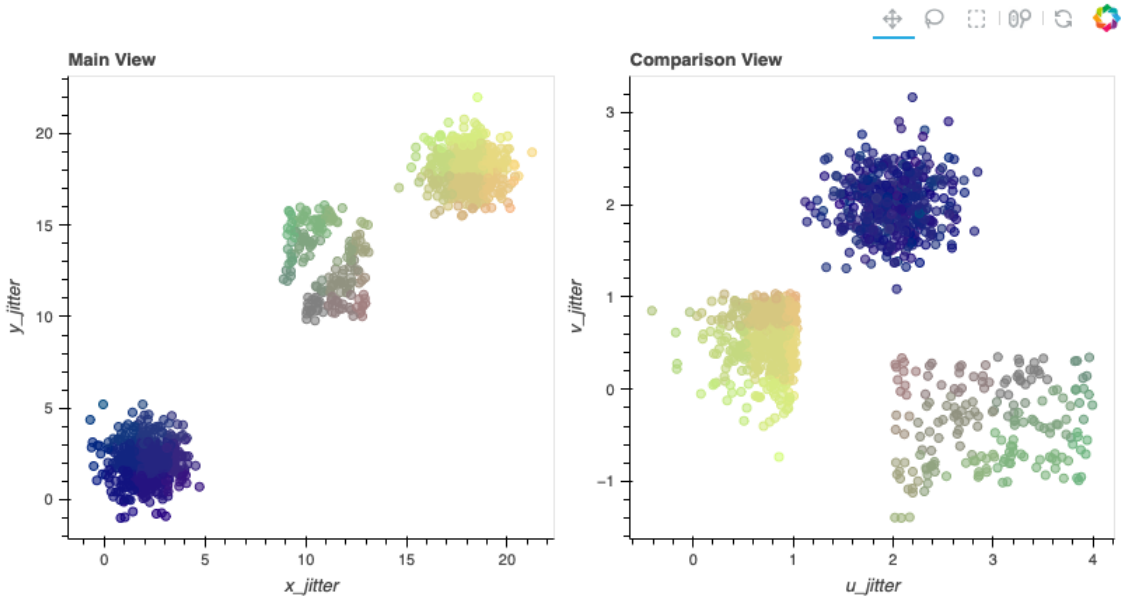


Figure 5.10: Use Case 3 with SOM (before training): At the start of training there is practically no separation visible for the subclusters in (u, v)

This is especially visible at the beginning of training when color diversity is low (Figure 5.10). The color gradient between points at the boundary of the subclusters remains relatively small in the (u, v) view at the end of training (Figure 5.11). The increase in color diversity is present in all clusters, whether they are split or not.

The hybrid case doesn't fare much better when SOM is applied only on the subclusters (Figure 5.12). While the color gradients at the boundary increase, they are still not big enough to allow for a clear separation.

Maximizing the color map while maintaining topology comes at a price in this particular use case. While Mean Shift clearly separates the two clusters, SOM doesn't make the distinction between a neighbor within the same cluster which is close and one from another cluster which is much further away.

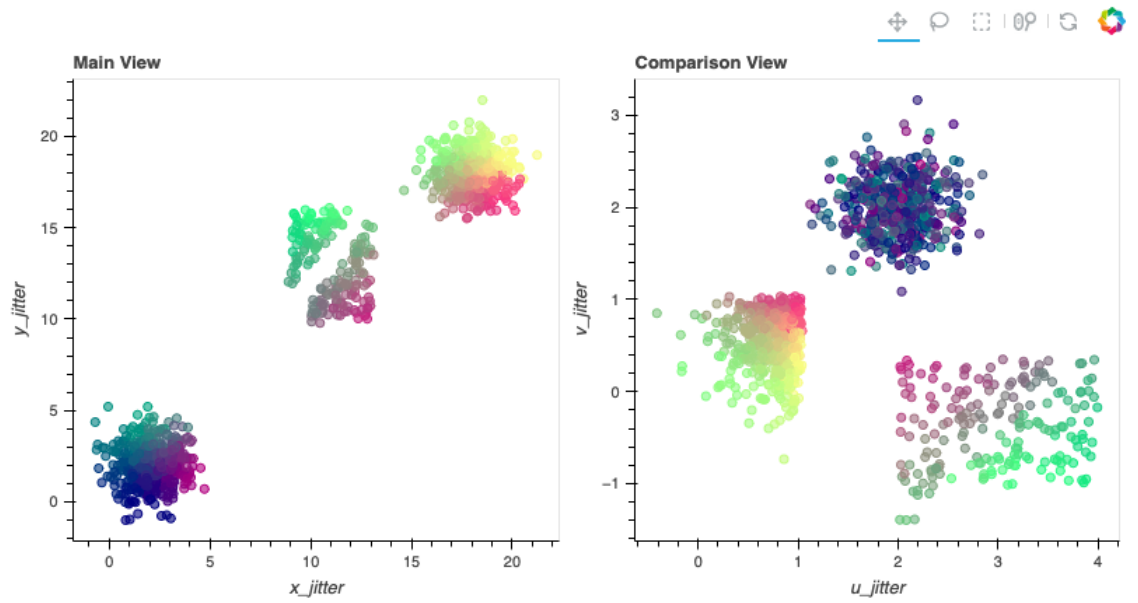


Figure 5.11: Use Case 3 with SOM (end of training): Color diversity increases at the end of training, but the color gradients at the boundaries remain small.

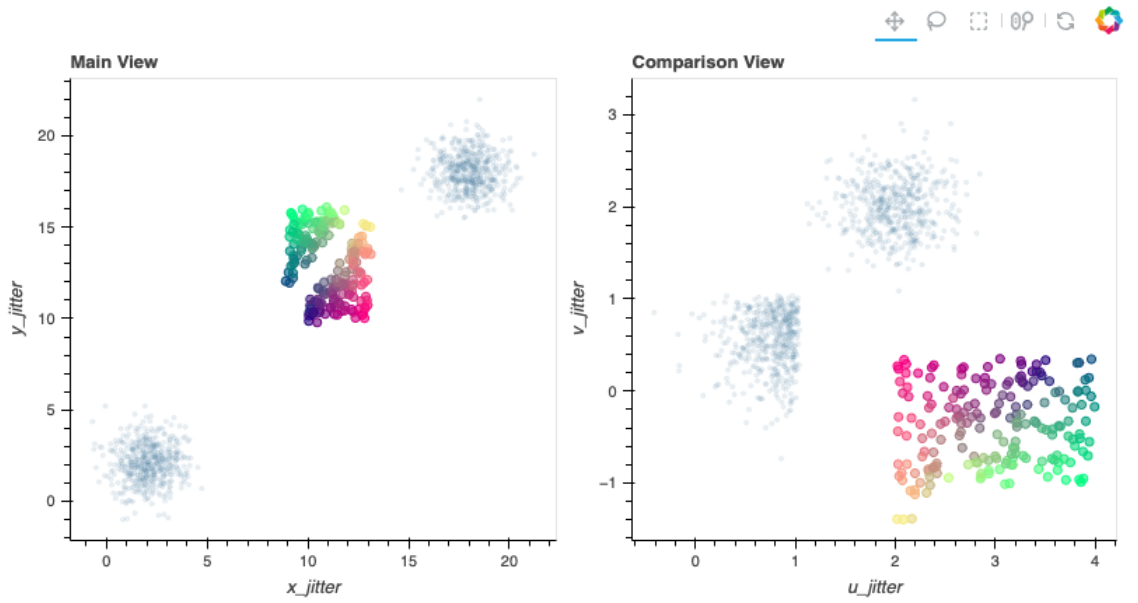


Figure 5.12: Use Case 3 with the hybrid function: The application of SOM on only the subclusters doesn't significantly increase the boundary color gradients.

5.2 Quantitative Evaluation

In this section, the runtime of certain interface functions is measured with an increasing row count (vertical scaling) and column count (horizontal scaling). This is important to understand the runtime behavior of certain implementations.

In vertical scaling, only the Mean Shift and SOM algorithms are tested. For Mean Shift, this also includes the bandwidth thresholds calculation. Since clustering is only done on a 2D view at a time, the runtime for more columns doesn't have to be evaluated.

Horizontal scaling is only done up to 40 columns, since a SPLOM grows exponentially in size with the amount of columns and its usefulness for pattern detection decreases with this size [LAE+12]. It will also only test loading and displaying the SPLOM for a vertical dimension which doesn't exceed 90 seconds in applying the Mean Shift or SOM algorithm.

The datasets are the following:

1. two Gaussian clusters in multi dimensional space which don't overlap and have an equal amount of points
2. the distribution from Use Case 1. The amount of points between clusters is the same. Horizontal scaling repeats the 3D distribution.
3. the distribution from Use Case 2. Again, horizontal scaling repeats the same distribution and the amount of points between clusters is the same.

Mean Shift is only evaluated for the first and second dataset, while SOM is only evaluated for the second and third dataset. Runtimes over 90 seconds are grouped together, and further vertical scaling isn't continued for this dataset and clustering algorithm once that threshold has been reached.

The code is executed on a Mac Book Air in JupyterLab with an Apple M2 Chip and 16 GB of RAM.

5.2.1 Vertical Scaling

Runtimes aren't measured on a continuous scale for these algorithms. Since subsampling data is a common practice in data analytics, the goal of this evaluation isn't the estimation of the computational complexity. The objective is rather to get an approximate idea of how usable the interface is for different magnitudes of instances.

Mean Shift

Table 5.1: Mean Shift Runtimes (Dataset 1)

Sample Size	Time (seconds)
500	3,5
5000	14,3
50000	>90

Table 5.2: Mean Shift Runtimes (Dataset 2)

Sample Size	Time (seconds)
500	3,9
5000	13,5
50000	>90

These results give a clear indication that the sample size shouldn't exceed around 5000 instances in order to conduct a practical analysis with the interface. If possible, a smaller sample size around 500 instances would be even better, since clustering is repeated if a different Main View or a subset of clusters is selected. It's also apparent that different distributions don't significantly affect the Mean Shift runtimes for the same sample size.

SOM

Table 5.3: SOM Runtimes (Dataset 2)

Sample Size	Time (seconds)
500	3,8
5000	57,1
50000	>90

Table 5.4: SOM Runtimes (Dataset 3)

Sample Size	Time (seconds)
2000	11,1
20000	>90
200000	>90

These results show that SOM requires a much smaller sample size than Mean Shift in order to remain practical.

It should be noted that the runtime of SOM also depends on the distribution of the data. Since the amount of epochs is not set in advance, training time can vary significantly between datasets with the same amount of instances. Training of the SOM on the dimensions petal length and petal width of the IRIS dataset takes 6,1 seconds, although only 150 instances are present in that dataset. This is due to the fact that in this particular case SOM is being trained for about 70 epochs. In the second dataset, training terminates at approximately 10 epochs.

5.2.2 Horizontal Scaling

For horizontal scaling, the distribution of the data itself isn't of interest, since the goal of this evaluation is to determine how runtime increases if more plots have to be displayed. Therefore, only the distribution of the first dataset is used and repeated up to 40 columns. Clustering and coloring is done with Mean Shift since the runtime of SOM varies between views and distributions more heavily. Since the interface should be practically usable, only sample sizes of 500 and 5000 instances are evaluated.

The following tables show the measured runtimes for all operations from data loading, over Mean Shift clustering and displaying all elements of the interface.

Table 5.5: Execution Times for 500 Rows and different column amounts

Columns	Time (seconds)
5	4
10	4,6
15	5,9
20	9,4
25	15,9
30	28,2
35	41,3
40	58,4

Table 5.6: Execution Times for 5000 Rows and different column amounts

Columns	Time (seconds)
5	14,8
10	17,9
15	23,6
20	35,7
25	53,8
30	70,7
35	>90
40	>90

The practical limits of the interface become apparent, especially with a sample size of 4000 instances. While some runtimes seem to make the practical usage very difficult, higher dimensionalities should be avoided anyway, especially if the objective is to reduce cognitive load. More than 10 dimensions, for example, require the user to zoom out in order to see the whole SPLOM.

6 Implementation

The implementation is split into two main python files. The data layer file handles extract, transform, and load (ETL) tasks, the application of the clustering algorithms, and assigning colors to the points. The frontend file handles the creation of the visualizations and the execution of the correct callbacks to ensure interactivity. There is also a server file which hosts the color map URLs locally, in order to properly display them with bokeh.

Several libraries apart from bokeh are used. These include pandas and numpy for efficient data transformation and storage. Scikit-learn is used for the implementation of the Mean Shift algorithm itself. Meanwhile, sklearn-som is used for the implementation of the self-organizing map algorithm. Within the training of these SOMs, the scipy package is used for controlling the parameters. Finally, PIL and flask are used for loading and hosting the color maps respectively.

In the following sections, the clustering/coloring algorithms are explained in more detail, together with the calculation of bandwidth thresholds.
Before going into detail, it's important to understand how the color map and points with their respective colors are stored.

6.1 Storage and Representation of the color map and data space

The color map image is stored as a numpy array with the same dimensions as the color map by pixels. Each entry stores the color value of the pixel it represents as a hex code string.

The data and its assigned colors are stored in the same pandas dataframe. The colors for Mean Shift per point and the colors for each epoch of the SOM are stored. Each point has a boolean cluster selection value. This value is by default True and False if the user deselects the cluster this point belongs to. In this case, it stays False until the User clicks on the "Reset Clusters" button.

6.2 Mean Shift Clustering and Coloring

Mean Shift Clustering itself is mainly implemented in the Scikit-Learn package. Since the color maps are quadratic in size, the data of the selected view has to be normalized first. If no cluster amount is selected, the bandwidth is estimated by the Scikit-Learn function "estimate bandwidth" with the quantile argument set at 0,2. Clustering is then done, which returns cluster centers with their corresponding colors. Once the cluster center coordinates per point are determined, the actual coloring can begin.

```
# The selected dimensions x and y have to be normalized first
df = min_max_scaler(df, [x, y])

X = df[[x_scaled, y_scaled]]

if bandwidth is None:
    bandwidth = sk_estimate_bandwidth(X, quantile=0.2)

# Bin seeding is enabled to speed up the algorithm
mean_shift = sk_MeanShift(bandwidth=bandwidth, bin_seeding=True)

mean_shift.fit(X)

cluster_centers = mean_shift.cluster_centers_
labels = mean_shift.labels_

# Cluster center coordinates are mapped per point
df.loc[:, "label"] = pd.Series(labels)
df.loc[:, "cluster_center_x"] = df.apply(lambda row: cluster_centers[row["label"], 0], axis=1)
df.loc[:, "cluster_center_y"] = df.apply(lambda row: cluster_centers[row["label"], 1], axis=1)

# The following casting functions ensure a proper index calculation of the color map per
coordinate
def y_coord_cast(y, color_map):
    y = int((1 - y) * len(color_map)) # The vertical dimesnion is mirrored in the color map
    if y < 512:
        return y
    # Edge case handling
    elif y == 512:
        return 511

def x_coord_cast(x, color_map):
    x = int((x) * len(color_map))
    if x < 512:
        return x
    # Edge case handling
    elif x == 512:
        return 511

# Mapping cluster center coordinates to color per point
df.loc[:, "color"] = df.apply(
    lambda row: color_map[
        y_coord_cast(row["cluster_center_y"],color_map),
        x_coord_cast(row["cluster_center_x"], color_map)
    ],
    axis=1
)
```

Listing 6.1: Mean Shift Clustering and Coloring

In the coloring stage, the points with their cluster center coordinates are mapped onto the numpy array which represents the color map. The rounded result of multiplying the x coordinate with the length of the color map provides the horizontal index of the numpy array. Meanwhile, the rounded result of multiplying 1 - the y coordinate with the length of the color map provides the vertical index of the numpy array. This rounding may reduce accuracy; however, the resolution of the color maps is at least 512x512 pixels which makes rounding mistakes barely visible. The following code further illustrates the implementation. It is important to note that it is not an exact representation of the code being used. It rather condenses the relevant parts of the "data transformer" and "color for cluster" functions for Mean Shift without cluster selections.

6.3 Bandwidth Thresholds Calculation

To make the interface more usable, bandwidth values for all cluster amounts between one and ten are calculated. This required the development of a new algorithm which can determine these bandwidth thresholds in an efficient manner (under 2 seconds for the IRIS dataset). Increasing the bandwidth always produces either the same or fewer clusters, while decreasing the bandwidth results in either the same or more clusters. It is also known that the bandwidth range for a normalized data space is set in the interval (x, 1], where x refers to the minimum pairwise distance between all points. Higher bandwidths are, of course, possible, but a bandwidth of 1 is guaranteed to only produce one cluster if the data space is appropriately normalized.
Now that an interval has been defined and we know that the cluster amounts this interval produces are sorted inversely, a good search algorithm to choose is the binary search with a logarithmic runtime. However, first experiments showed that the initial lower boundary isn't optimal. Since the objective of applying Mean Shift lies in clearly distinguishing between clusters through a distinct coloring, a high amount of clusters isn't necessary. Consequently, ten clusters have been set as the maximum in this interface. Therefore, all bandwidths which are lower than the bandwidth that produces ten clusters shouldn't have to be included at all in this binary search.

```
# Heuristic to search for a high lower limit bandwidth for all clusters
heuristic_loop = 0
for x in np.arange(0.05, 1, 0.005):

    ms = sk_MeanShift(bandwidth=x, bin_seeding=True)
    ms.fit(data)
    labels = ms.labels_
    n_clusters = len(np.unique(labels))
    if n_clusters > 10:
        min_bandwidth_all = x
        heuristic_loop += 1
    if n_clusters <= 10:
        if heuristic_loop == 0:
            min_bandwidth_all = 0.0001
            heuristic_loop += 1
        else:
            break
```

Listing 6.2: Heuristic in bandwidth threshold calculation

This is why a fast heuristic approach has been developed, which aims to find a lower limit for the actual search. This heuristic starts with a very low bandwidth like 0,05 and then performs Mean Shift and counts the amount of clusters. If the amount of clusters is higher than the maximum amount of clusters we want to determine the bandwidth for, the process continues for a higher bandwidth. The goal is not to find the actual lower limit for the bandwidth range, but one that is "good enough". Experiments on the datasets which were previously described showed that this heuristic leads to a significant improvement in runtime over the naive approach that doesn't modify the lower limit.

In the binary search itself, the issue of numerical instability had to be addressed. For every loop of the binary search, the midpoint between the current maximum and minimum is determined. Then, the algorithm checks if this value (the amount of clusters it produces) is the value we are looking for or larger or smaller than this value. Since these search areas are continuous and not discrete, the value to search for is by calculating min + ((max - min) / 2). A potential issue lies in the numerical instability of the division operation. This can lead to the new float value being the same as the old one, which would result in an infinite loop.
The solution lies in introducing a nudge mechanism which activates once the new bandwidth is identical to the old one when searching for the same cluster amount. Then, a beta distribution which lies within the delta of that search area is introduced. This distribution is heavily biased towards the midpoint we want to calculate (a=5 and b=5). A random value from this distribution is drawn and added to the minimum. Since all operations are numerically stable this time, the new value will be guaranteed to change to the approximate range we intend.

```
cluster_counter = 0
for cluster_no in range(1, max_clusters + 1):
    # Skip if the cluster number has already been calculated
    if cluster_no in thresholds.keys():
        continue
    tries = 0
    while n_clusters != cluster_no:
        cluster_counter += 1
        # Perform MeanShift clustering
        ms = sk_MeanShift(bandwidth=bandwidth, bin_seeding=True)
        ms.fit(data)
        labels = ms.labels_
        n_clusters = len(np.unique(labels))
        if n_clusters not in thresholds.keys() and n_clusters <= max_clusters:
            thresholds[n_clusters] = bandwidth

        # Cluster amount too big, so increase bandwidth
        if n_clusters > cluster_no:
            if len(thresholds) == 0:
                bandwidth = bandwidth + ((max_bandwidth - bandwidth) / 2)
            else:
                if min_bandwidth < bandwidth:
                    min_bandwidth = bandwidth
                new_bandwidth = min_bandwidth + ((max_bandwidth - min_bandwidth) / 2)

                # Nudge mechanism if bandwidth hasn't changed due to numerical instability
                if new_bandwidth == bandwidth and tries > 0:
```

```
                    range_width = max_bandwidth - min_bandwidth
                    # Use a beta distribution to bias the nudge toward the center
                    nudge_amount = np.random.beta(a=5, b=5) * range_width
                    bandwidth = min_bandwidth + nudge_amount
                else:
                    bandwidth = new_bandwidth
                    tries += 1

        # Cluster amount too small, so decrease bandwidth
        elif n_clusters < cluster_no:
            if len(thresholds) == 0:
                bandwidth = min_bandwidth + ((bandwidth - min_bandwidth) / 2)
            else:
                for cluster_dict, bandwidth_dict in thresholds.items():
                    if cluster_dict > n_clusters and cluster_dict < cluster_no and
bandwidth_dict < max_bandwidth:
                        max_bandwidth = bandwidth_dict
                if max_bandwidth > bandwidth:
                    max_bandwidth = bandwidth
                new_bandwidth = min_bandwidth + ((max_bandwidth - min_bandwidth) / 2)

                # Nudge mechanism if bandwidth hasn't changed due to numerical instability
                if new_bandwidth == bandwidth and tries > 0:
                    range_width = max_bandwidth - min_bandwidth
                    # Use a beta distribution to bias the nudge toward the center
                    nudge_amount = np.random.beta(a=5, b=5) * range_width
                    bandwidth = min_bandwidth + nudge_amount
                else:
                    bandwidth = new_bandwidth
                    tries += 1

    thresholds[n_clusters] = bandwidth
    max_bandwidth = bandwidth
    min_bandwidth = min_bandwidth_all
return (x_col, y_col), thresholds
```

Listing 6.3: Binary Search for calculating bandwidth thresholds

For IRIS data, this threshold calculation terminates in under two seconds for all 2D views. It is therefore an efficient algorithm to calculate bandwidths up to ten clusters for small sample sizes.

6.4 SOM and Coloring

Another original algorithm is the way the SOM is trained. As previously explained, the initial weight configuration is a 15x15 grid scaled to the data. As with Mean Shift, only the most important parts of this clustering and coloring are shown here.
The parameters of the SOM are set by default to 0,3 as the step size for updating the weights and 1,3 as sigma, which controls how far updates propagate. Over the course of training, only sigma will be modified. Mapping of weights to their color is done as in Mean Shift clustering, but in this case as RGB values to enable interpolation.// //Once training is done for the epoch, the colors for each point can be either assigned directly by its closest weight (BMU) or interpolated. Direct coloring essentially works like in Mean Shift, but with the point's BMU color instead of its cluster center color. Interpolating the color performs a cubic interpolation for all color channel values over the training data and current position of the weights with their color values. Cubic interpolation can be useful if the number of instances is larger than the number of weights, which is fixed at 225.

```
.......

if coloring == "Interpolated":
    # Interpolate each color channel separately
    interpolated_colors = np.zeros((len(training_data), 3))
    for i in range(3):  # RGB channels
        interpolated_colors[:, i] = griddata(
            som.weights,  # current position of neurons
            weight_colors_rgb[:, i],  # colors for all grid points
            training_data,  # BMU coordinates of training data
            method='cubic',
            fill_value=np.nan
        )

.......

elif coloring == "Direct":
    bmu_indices = np.array([som._find_bmu(x) for x in training_data])
    x_coords = bmu_indices % neuron_grid_size
    y_coords = bmu_indices // neuron_grid_size
    # Get the corresponding colors from the downsampled color map
    colors = [color_map_downsampled[y, x] for y, x in zip(y_coords, x_coords)]
    sel.loc[:, f'color_{str(epoch)}'] = colors
```

Listing 6.4: Direct and Interpolated Coloring for SOM

Once at least 95 percent of weights are within the convex hull of the training data, sigma is adjusted to 0,3 to avoid overfitting and topology violations. At this point, the entropy of the BMU distribution becomes relevant in order to determine when to stop training.

```
def calculate_bmu_entropy(bmu_indices, n_neurons):
    """
    Calculate the entropy of the BMU distribution.

    Parameters:
```

```
    bmu_indices: array of BMU indices for each data point
    n_neurons: total number of neurons in the SOM

    Returns:
    float: entropy value
    """
    # Count how many times each neuron is a BMU
    bmu_counts = np.bincount(bmu_indices, minlength=n_neurons)

    # Convert counts to probabilities
    bmu_probs = bmu_counts / len(bmu_indices)

    # Calculate entropy (using base 2 for information theory convention)
    return entropy(bmu_probs, base=2)
```

Listing 6.5: The calculation of BMU entropy

For each epoch, this entropy is calculated. From the tenth epoch onward, the mean and standard deviation (std) for the last ten epochs are calculated. Once the current entropy falls within the interval [mean - std, mean + std] and most weights are within the convex hull, training terminates.

7 Discussion

While the interface fulfills the requirements from 1 when compared to normal brushing and linking approaches, it still has several issues to contend with.

Firstly, the Mean Shift algorithm is not executed on the actual 2D data space, but rather on its normalized data space. If the aspect ratio of the selected view is high, the color gradient between different clusters may not be an accurate representation of their absolute distance from each other. However, the topology of the color space would still be maintained when comparing the assigned colors relative to each other. Since Mean Shift can recognize clusters regardless of shape, stretching or compressing clusters in one dimension shouldn't have an effect on accuracy.

Another issue with Mean Shift lies in the coloring process itself. Clusters which are clearly separated, but whose cluster centers lie relatively close to each other, get assigned to a relatively similar color. While this is the intention, this can lead to clusters having a near identical color, although the clusters are clearly distinct, but just not far away from each other. This is also an issue which may be exacerbated by a high aspect ratio. This problem may be solved by changing the color map, or modifying them to a higher aspect ratio.

There are also challenges when applying SOM, especially in cluster detection tasks. As shown in 5.1.3, cluster detection only performs well if the relative distance between clusters is high enough. On the other hand, it has also become clear that the strengths of SOM lie in the detection of functional dependencies. In pure cluster detection tasks, the Mean Shift algorithm can be used and fares much better.

Future research should also evaluate the performance of SOM if many outliers are present. This may significantly reduce the color diversity of actual clusters or increase training time, since the convex hull is calculated for every point. This can lead to updates becoming very small at a point when most weights are still relatively far away from the actual clusters. This would either lead to the algorithm stopping too early, or move too slowly if the BMU entropy doesn't stabilize. This can be mitigated by first identifying cluster centers with Mean Shift, and then ignoring points which are far away from these centers, when calculating the convex hull.

Additionally, SOM has only been shown to work well in detecting functional dependencies when they are semi-linear. As can be seen for Use Case 2 (1.4), the settings for the trigonometric functions together with the range of the data don't produce the characteristic wave function. Other functions may lead to the projection in (u, v) being less "smoothly colored" which could impair the spotting of these kinds of dependencies.
The parameters for the different stages of SOM training are hardcoded and based on experiments with different values for these datasets. These settings may produce worse colorings for other datasets. Future research should focus on optimizing these parameter settings further. Dynamically

training the SOM based on whether the weights are in the convex hull and the BMU entropy should also be expanded on further. Other methods should be developed and evaluated on a broader range of datasets and distributions. It has been shown that for the same amount of instances, SOM can take very different epochs to complete. This is another reason to adjust the SOM training process in a way that produces more consistent results.

Generally speaking, a User Study should be performed for the interface which evaluates its usability and performance more robustly.

8 Conclusion

This thesis addressed various challenges data analysts face in trying to detect patterns in multi-dimensional data with standard brushing and linking techniques. In a short overview of related work, gaps in automation and usability of current interfaces became apparent.

Therefore, a novel interface with the Mean Shift and SOM algorithm has been developed. Both algorithms assign colors based on a color map, while preserving the topology of this color space. For Mean Shift, the location of the cluster centers determines the position on the color map. This, in turn, colors all points belonging to this cluster accordingly. For SOM, the color map is first downsampled to 15x15 pixels, which gives the weights their color value. These weights are initially a grid of the same size, which is overlaid on the data to train on. The coloring of the data per epoch is either determined by the color of the closest weight or by cubic interpolation. Additional functionalities, such as interactive cluster selection and hybrid use cases combining Mean Shift with SOM, were introduced to enhance the user experience. A new and efficient algorithm to calculate bandwidth thresholds for different cluster amounts has been developed. This created a user-friendly, intuitive, and interactive tool for visual data analysis.

In the evaluation, the qualitative effectiveness of the interface has been demonstrated through various use cases. It also showcased that Mean Shift is best at identifying clusters, while SOM is better at identifying functional dependencies. Meanwhile, the quantitative evaluation did show scaling issues; however, these can be mitigated by subsampling the data. The results showed that the interface significantly reduced cognitive load and manual effort compared to traditional SPLOMs, particularly in tasks involving functional dependency identification. However, challenges such as cluster misclassification in Mean Shift and limited separation in SOM for closely spaced clusters were also identified.

This thesis introduced an interface that significantly reduced manual exploration and cognitive load in detecting both clusters and functional dependencies for multi-dimensional data. It is able to preserve the topology of a given color space and allows for a flexible interactive refinement in visual data analysis. To build upon this research, alternative clustering methods that address Mean Shift's sensitivity to bandwidth selection and SOM's challenges with closely spaced clusters should be studied. Also, adaptive mechanisms for dynamically tuning SOM parameters based on dataset characteristics should be developed, especially for datasets with many outliers. Computational efficiency of the interface should also be improved to handle larger datasets if subsampling isn't an option. Most importantly, robust user studies are required to evaluate the interface's effectiveness and learning curve in real-world scenarios.
In conclusion, this thesis presents a novel approach to enhancing SPLOM interfaces for multi-dimensional data visualization by combining advanced clustering techniques with intuitive interaction mechanisms. While there are areas for improvement, the proposed interface lays a strong foundation for future advancements in visual analytics tools.

Bibliography

[BSM+15] J. Bernard, M. Steiger, S. Mittelstädt, S. Thum, D. A. Keim, J. Kohlhammer. "A survey and task-based quality assessment of static 2D colormaps". In: *Proceedings of SPIE-IST Electronic Imaging*. Ed. by D. L. Kao. Article Number: 93970M. Springfield: SPIE, 2015. ISBN: 978-1-62841-487-5. DOI: 10.1117/12.2079841 (cit. on p. 27).

[Car15] M. Á. Carreira-Perpiñán. "A review of mean-shift algorithms for clustering". In: *CoRR* abs/1503.00687 (2015). arXiv: 1503.00687. URL: http://arxiv.org/abs/1503.00687 (cit. on pp. 23, 24).

[CLC22] C. Cariou, S. Le Moan, K. Chehdi. "A Novel Mean-Shift Algorithm for Data Clustering". In: *IEEE Access* 10 (Jan. 2022), pp. 14575–14585. DOI: 10.1109/ACCESS.2022.3147951. URL: https://hal.science/hal-03593568 (cit. on p. 23).

[CORV18] M. Cottrell, M. Olteanu, F. Rossi, N. N. Villa-Vialaneix. "Self-Organizing Maps, theory and applications". In: *Revista de Investigacion Operacional* 39.1 (2018). hal-01796059, pp. 1–22 (cit. on p. 25).

[CSG+18] M. Chegini, L. Shao, R. Gregor, D. J. Lehmann, K. Andrews, T. Schreck. "Interactive Visual Exploration of Local Patterns in Large Scatterplot Spaces". In: *Computer Graphics Forum* 37.3 (2018), pp. 99–109. DOI: https://doi.org/10.1111/cgf.13404. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13404. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13404 (cit. on p. 20).

[EDF08] N. Elmqvist, P. Dragicevic, J.-D. Fekete. "Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1148–1539. DOI: 10.1109/TVCG.2008.153 (cit. on p. 19).

[KPV+17] P. Koytek, C. Perin, J. Vermeulen, E. Andre, S. Carpendale. "MyBrush: Brushing and Linking with Personal Agency". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2017), pp. 605–615. DOI: 10.1109/tvcg.2017.2743859 (cit. on p. 21).

[LAE+12] D. J. Lehmann, G. Albuquerque, M. Eisemann, M. Magnor, H. Theisel. "Selecting Coherent and Relevant Plots in Large Scatterplot Matrices". In: *Computer Graphics Forum* 31.6 (2012), pp. 1895–1908. DOI: https://doi.org/10.1111/j.1467-8659.2012.03069.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03069.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03069.x (cit. on pp. 17, 50).

[LMW+17] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, V. Pascucci. "Visualizing High-Dimensional Data: Advances in the Past Decade". In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (2017), pp. 1249–1268. DOI: 10.1109/TVCG.2016.2640960 (cit. on p. 17).

[Mun14] T. Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC Press, 2014. ISBN: 9781498759717. URL: https://books.google.de/books?id=NfkYCwAAQBAJ (cit. on p. 17).

[War04] C. Ware. *Information Visualization: Perception for Design*. New York, NY, USA: Elsevier, 2004. ISBN: 1558608192 (cit. on p. 11).

[War09] M. Ward. "Linking and Brushing". In: (Jan. 2009). DOI: 10.1007/978-0-387-39940-9_1129 (cit. on pp. 17, 18).

[Yin08] H. Yin. "The self-organizing maps: Background, theories, extensions and applications". English. In: *Studies in Computational Intelligence|Stud. Comput. Intell.* Vol. 115. United States: Springer Nature, 2008, pp. 715–762. DOI: 10.1007/978-3-540-78293-3_17 (cit. on p. 25).

All links were last followed on February 9, 2025.

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