On sketch-based selections from scatterplots using KDE, compared to Mahalanobis and CNN brushing

Chaoran Fan
University of Bergen

Helwig Hauser
University of Bergen

***Abstract*—Fast and accurate brushing is crucial in visual data exploration and sketch-based solutions are successful methods. In this paper, we detail a solution, based on kernel density estimation (KDE), which computes a data subset selection in a scatterplot from a simple click-and-drag interaction. We explain, how this technique relates to two alternative approaches, i.e., Mahalanobis brushing and CNN brushing. To study this relation, we conducted two user studies and present both a quantitative three-fold comparison as well as additional details about the prevalence of all possible cases in that each technique succeeds / fails. With this, we also provide a comparison between empirical modeling and implicit modeling by deep learning in terms of accuracy, efficiency, generality and interpretability.**

INTRODUCTION

Linking and brushing is a widely adopted interaction technique for visual data exploration in coordinated multiple views [1]. Over 30 years ago, Becker and Cleveland [2] defined brushing as an interactive method to select data points by using simple geometries on a data visualization such as a square, circle, or a polygon. In coordinated multiple views, brushing usually leads to a consistent highlighting of the selected data in all linked views, amounting to an important form of focus+context visualization [5].

Since brushing is central to visual analytics, a substantial amount of research has been devoted to it. The many available forms of brushing can be categorized into four different types:

- *simple geometries*—this is the most common category, including the rectangular brush on scatterplots, line brushing on data graphs, etc.
- *lassoing*—the user selects a data subset by drawing a geometrically detailed lasso around the subset's visualization in a view.
- *logical combinations of simple brushes*—the user refines the data selection iteratively by using multiple brushes and combining them using logical operators such as AND and OR.
- *sketch-based brushing*—the user sketches a shape onto a visualization and a selection heuristic is used (often based on a similarity measure) to determine which data are selected.

To evaluate a brushing technique, the following two criteria are of particular importance:

- *efficiency*—how fast is the brushing algorithm and how much time does the users spend on

the selection process; is the interaction fluid?

- *accuracy*—to which degree does the interaction lead to an accurate selection of the data subset as the user actually intended to select?

Clicking on a mark in a visualization to select the corresponding data point, for example, is highly efficient for selecting individual data points. High accuracy is possible, when using a geometrically detailed lasso to select data points in a scatterplot. In many cases, it is not trivial, or possible, to optimize for both criteria concurrently.

Many brushing techniques are indeed fast—we think of brushing to be fast, if only one click or only very few atomic interactions are needed to specify the brush, leading to a swift user–computer dialogue during exploration / analysis. The use of simple brushing geometries (rectangle, circle, etc.) and sketch-based brushes, where only a quick gesture is used for brushing, are examples of fast techniques. A common disadvantage of these methods is, however, that it can be difficult to accurately brush a targeted data subset.

Certainly, we also have brushing techniques that are accurate—likely with lassoing and the logical combination of simple brushes being the most prominent examples. With such a technique, it is straight-forward to accurately select a subset of interest. This benefit of being accurate, however, commonly comes at the cost of reduced efficiency—specifying a lasso point-by-point, for example, easily becomes a lengthy unit task by itself, interrupting the data exploration process.

To optimize both criteria for one technique as much as possible, data-driven methods are an interesting option. One typical kind of such a solution is based on sketch-based user interaction. The sketching of a simple shape, for example a line segment by a click-and-drag interaction, is complemented with a heuristic that estimates the actual data selection from the sketch. To achieve high accuracy, the parameters of such a technique can be optimized on the basis of data from a user study [7], [8].

The Mahalanobis brush [7], [9] is an example of a data-driven technique, using local covariance information as basis for a Mahalanobis metric that determines which points are closest to the sketch. The parameters of Mahalanobis brushing can be optimized based on data from a user study [7]. Quantitative evaluation shows that it can achieve ≈92% of accuracy, based on a fast click-and-drag interaction. Inspired by the success of machine learning, a brushing technique based on deep learning (DL) with a convolutional neural network (CNN) was developed [8] and achieved a substantially improved accuracy (≈97%). Both the interaction as well as the data visualization were represented as images to subject them as input to the CNN, and letting the network learn the model based on data from a user study.

With Mahalanobis brushing [7] and CNN brushing [8], two principally different approaches are given, i.e., representing the principles of empirical modeling (based on reasoning) vs. implicit modeling (based on deep learning). Since CNN brushing resulted in a significantly higher accuracy [8], and since Mahalanobis brushing is not based on any advanced distance metric, we were interested in studying to which degree empirical modeling can in fact compete with deep learning in this context.

In this paper, we now provide details of our attempt to construct an improved empirical model by further extending the Mahalanobis brush, incorporating kernel density estimation (KDE) [10], to inform a clustering step which then returns one of the clusters as the data selection [11]. Additionally, we contribute an in-depth, threefold comparison between the Mahalanobis brush, the CNN brush, and the new KDE brush.

Related work

Selecting data subsets by brushing is one of the most common types of interaction in visual analytics, where often data points are selected in one display, while the same information is highlighted in linked views (linking and brushing) [1].

Often, a combination of mouse motions and button clicks are used to realize brushing [3]. Less common methods are based on eye / head tracking or gestures, as for example in virtual reality [4].

Several extensions to simple brushing have been proposed, including techniques to formulate complex brushes by combining simple brushes using logical operators, including the union, intersection, and negation of brushes, and enabling the user to iteratively refine the data selection.

Furthermore, brushes are often adapted to interact with a particular aspect of the visualization

mapping. Hauser et al. [6], for example, developed angular brushing on parallel coordinates to select data points, whose representation include lines with a particular angle.

MyBrush was suggested by Koytek et al. [12] in order to extend brushing and linking by incorporating personal agency, allowing to configure the source, link, and target of multiple brushes.

Sketching is a natural and intuitive way for users to identify data subsets of interest in a visualization. Similarity brushing [13] is an example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user performs a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure is used to identify, which data items are actually selected. The main advantage of sketch-based brushing is the fast interaction, which, however, is not perfectly accurate, usually.

As another sketch-based solution, the Mahalanobis brush was presented as an interesting option for brushing scatterplots [9]. In this technique, a simple sketch (a click near the center of the data subset that should be selected) is used to brush the data. The link between the interaction and the actual selection is computed on the basis of the underlying data (local covariance information is used to determine the overall shape and orientation of the selection): all data points near the click-point, measured according to a local Mahalanobis metric, are selected.

While this technique is giving good results (≈65% accuracy [7]), it still has limitations, including a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size. To improve the accuracy of this approach, the Mahalanobis brush was extended by optimizing its parameters using data from a user study and getting rid of the off-line parameter [7]. In terms of efficiency, the average interaction (click-and-drag) time spent for the new Mahalanobis brushing is only 41% of Lasso [7]. However, this improved solution is still linear and has therefore difficulties with complex structures that would require a more flexible approach.

Inspired by the success of deep learning in a wide range of applications, especially in image processing, we then developed CNN brushing [8], using deep learning to establish the link between the user sketch and the actual data selection, achieving very high accuracy. However, as a general model, it learns the "average behavior" from different users, and thus is not able to match every single user's brushing preferences 100%. To address this issue, we further improved CNN brushing and achieved a solution which is able to turn the general model into a tailored model for a specific user [14]. To achieve this, we take the user into the loop to iteratively refine the brushing model with additional data which the user provides while using the brushing technique.

Despite the opportunity to achieve really good results with the help of deep learning, related approaches to solve interactive visualization tasks are still rare. The limited utilization of deep learning for visual analytics solutions is likely based on three reasons: (1.) Understanding a deep learning based model is challenging due to its "black box" nature. (2.) Usually, high accuracy of DL-based prediction requires large amounts of training data, which often is difficult to acquire. (3.) There is no established common understanding of how to determine the right DL solution as knowledge of topology, training method and required hyperparameters. Consequently, it is often difficult to efficiently make good use of deep learning—especially, when non-standard tasks are to be supported.

KDE brushing on scatterplots

Fig. 1 shows an overview of the new KDE-based brushing technique. We keep the simple click-and-drag interaction for sketching the target data subset: click into the middle of the subset and drag the pointer swiftly to the outer boundary of it. The click point $\mathbf{s} = (s_x, s_y)^\top$ and the end point $\mathbf{e} = (e_x, e_y)^\top$ of the drag interaction indicate the size of the target subset that the user wishes to select. As with the Mahalanobis brush [7], we first consider a circular subset, centered around $\mathbf{s}$, and estimate the shape and orientation of the data in this region by looking at the local covariance information. We then start a short iteration to refine this data subset selection, based on the local covariance information—ideally, we would like to directly use the resulting user selection as the data subset, of course, amounting to a chicken-and-egg type

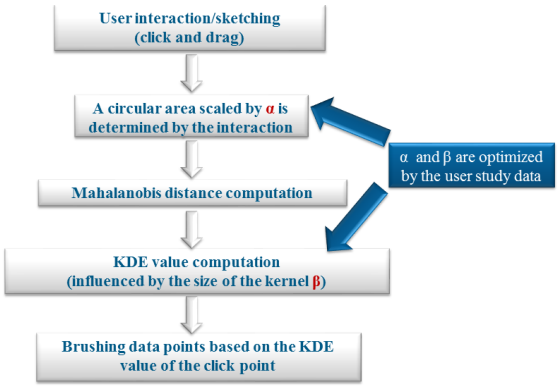

Figure 1. Overview of KDE brushing: the user clicks into the middle of the subset to be selected and drags the pointer to the border of the subset (sketching interaction); then a selection of points around the click-point is determined, based on the estimated density of the data; two parameters, α and β, related to the sample size and the size of the KDE bandwidth, influence the results and we optimize them based on a user study with 50 participants.

of problem, since the eventual selection is not known in advance. After a sufficiently close convergence of this iteration, we make a selection of data points, based on a kernel density estimation (KDE), using the local covariance information as a basis for specifying the kernel. The choice of KDE is based on three assumptions: (1.) Data-driven density information, captured by KDE, should improve results over the linear model in Mahalanobis brushing. (2.) A non-linear model should be suited to select general shapes. (3.) The modes (local maximum points) of a 2D KDE, at the right scale, can represent clusters of data points, corresponding to selection targets. In the following, we provide more details about the individual components of this approach.

Mahalanobis distance computation

Since the Mahalanobis distance is central to KDE brushing, we briefly review it first. Introduced by Mahalanobis in 1936 [15], it is based on the correlation between data variables to help with the identification and analysis of multivariate patterns. It is unitless and scale-invariant, which is a useful way for determining the similarity of an unknown sample to a known one. It differs from the Euclidean distance, which measures the distance with the available data. The Mahalanobis distance between vectors $\mathbf{a}$ and $\mathbf{b}$ is defined by

$$d_{\mathbf{\Sigma}}(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^{\top}\,\mathbf{\Sigma}^{-1}\,(\mathbf{a}-\mathbf{b})} \qquad (1)$$

where $\mathbf{\Sigma}$ is the covariance matrix of the sample. The locations of equal Mahalanobis distance from a central reference vector $\mathbf{x}$ form an ellipse around $\mathbf{x}$ in 2D.

In our proposed KDE brushing, the local covariance structure of a data subset around the click-point $\mathbf{s}$ is used as the basis of the kernel specification. Therefore, it is an important part of our approach to determine, which data subset should be used for this computation. We perform this in two steps.

Initially, we consider a circular area with radius $\alpha \cdot d_E(\mathbf{s},\mathbf{e})$, where α is a weighting factor and $d_E(\mathbf{s},\mathbf{e})$ is the Euclidean distance between $\mathbf{s}$ and $\mathbf{e}$. All data points within this circle are used to compute the first instance of the local covariance information, $\mathbf{\Sigma}_1$.

Next, we consider all points in a Mahalanobis ellipse, based on $\mathbf{\Sigma}_1$ and sized to $d_{\mathbf{\Sigma_1}}(\mathbf{s},\mathbf{e})$. Usually, this leads to a new subset, which is similar to the initial one, but fitting the underlying data structure more closely. To obtain an even better sample, we refine the sample iteratively by replacing it with the points in the Mahalanobis ellipse that is updated every iteration according to the covariance of the samples in last iteration.

While this process usually converges quickly, we observed that it sometimes can lead to small fluctuations, including and excluding a few points in consecutive iterations. To stabilize the convergence of the covariance matrix optimization, we enable the partial consideration of data points during the computation, leading to a solution that is based on the weighted covariance matrix. The elements σ_{jk} of the weighted covariance matrix $\mathbf{\Sigma}_w$ can be defined as:

$$\sigma_{jk} = \frac{\sum_i \omega_i (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)}{1 - \sum_i \omega_i^2} \qquad (2)$$

where ω_i is the normalized weight ($\omega_i \geq 0$) for vector $\mathbf{x}_i$ in a weighted sample with $\sum_i \omega_i = 1$ and $\bar{\mathbf{x}}$ being the weighted mean vector, given by $\sum_i \omega_i \mathbf{x}_i$. During the iteration, the weight of each point is updated and the points that are stable in the Mahalanobis ellipse are assigned a higher weight than less stable points. This process results

This article has been accepted for publication in IEEE Computer Graphics and Applications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MCG.2021.3097889

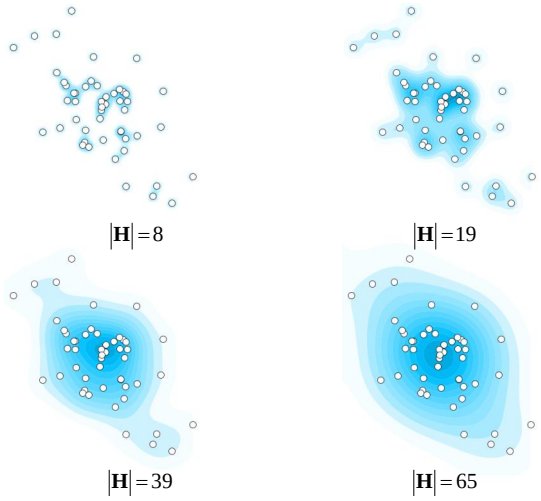

Figure 2. Changing the kernel size in KDE: bigger kernels, i.e., larger $|\mathbf{H}|$, bring forth larger structures in the data, while smaller kernels represent details.

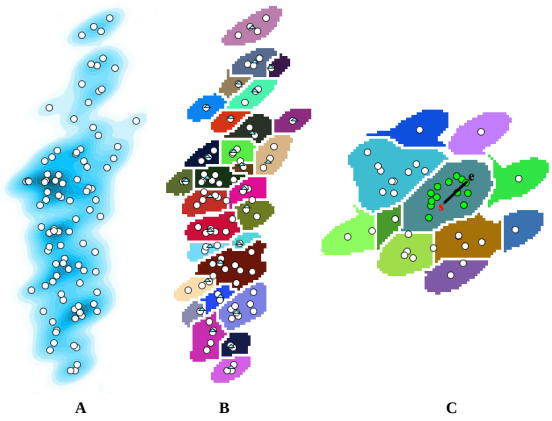

Figure 3. A: KDE of a dataset (relatively small kernel). B: Clustering related to the modes of A, shown as small cyan triangles. C: The one cluster, corresponding to the mode nearest to $\mathbf{s}$, determines which data points are selected (indicated as green points).

in a well-converged covariance matrix after a few interations. More details of this procedure (weight function design, singular matrix handling, etc.) are described in an earlier publication [7].

Density estimation

Kernel density estimation (KDE) is a popular method for data analysis [10]. It is a non-parametric way to estimate the probability density function of a random variable. KDE can be used, for example, to make inferences about data, based on a finite sample, without imposing any a priori structure on the data.

Given that $\{\mathbf{x}_i\}_{1\le i\le n}$ are n samples of d-dimensional vectors from a common distribution, KDE can be used to estimate their density as

$$f_{\mathbf{H}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}} \cdot (\mathbf{x} - \mathbf{x}_i) \qquad (3)$$

with $\mathbf{H}$ being a $d \times d$ bandwidth matrix (symmetric and positive definite). The choice of matrix $\mathbf{H}$ is the most important factor, critically affecting the characteristics of $f_{\mathbf{H}}$. Fig. 2 shows four results from 2D KDE with increasing determinant of $\mathbf{H}$, where $f_{\mathbf{H}}$ reveals details for small $|\mathbf{H}|$, while larger data structures dominate for larger $|\mathbf{H}|$.

$K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-\frac{1}{2}} K(\mathbf{H}^{-\frac{1}{2}}\mathbf{x})$ is the kernel function, with $K(\mathbf{x})$ being a symmetric multivariate density function with $K(\mathbf{x}) \ge 0$ and $\int K(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1$. A variety of kernels has been studied, including the uniform kernel, the triangle, normal / Gaussian, and Epanechnikov kernel, as well as others. The choice of the kernel function is actually not as important as the choice of the size (and shape) of $\mathbf{H}$. Being interested in the local mode of the data distribution, we use the normal kernel for KDE brushing.

To consider the local data distribution, when modeling kernel matrix $\mathbf{H}$, we make direct use of the converged covariance matrix $\mathbf{\Sigma}_w$, leading to the following anisotropic kernel function:

$$K_{\mathbf{H}}(\mathbf{x}) = \frac{e^{-\frac{1}{2}\mathbf{x}^\top \mathbf{\Sigma}^{-1}\,\mathbf{x}}}{\sqrt{(2\pi)^d |\mathbf{\Sigma}|}} \qquad (4)$$

To realize an proper scaling of the kernel, we use the eigendecomposition of $\mathbf{\Sigma}_w = \mathbf{V}\,\mathbf{\Lambda}\,\mathbf{V}^\top$ with eigenvectors $\mathbf{V}$ and eigenvalues $\mathbf{\Lambda}_{ii}$. This leads to the scaled versions of $|\mathbf{H}|^{-\frac{1}{2}} = |\beta\phi\mathbf{\Lambda}|^{-\frac{1}{2}}$ and $\mathbf{H}^{-\frac{1}{2}} = \mathbf{V}\,(\beta\phi\mathbf{\Lambda})^{-\frac{1}{2}}\,\mathbf{V}^\top$. Used with an isotropic kernel function $K(\mathbf{x}) = (2\pi)^{-\frac{d}{2}} e^{-\frac{1}{2}\mathbf{x}^\top\mathbf{x}}$, this corresponds to KDE with an accordingly scaled kernel matrix. We optimize the scaling of $\mathbf{H}$ by choosing the two scaling parameters ϕ and β by two separate methods: On the one hand, we use a data-driven approach to determine ϕ. On the other hand, we optimize β as a general parameter using the data from the user study.

Selecting a data subset using clustering

The modes of KDE represent groups of data items (at the scale determined by $|\mathbf{H}|$). We use clustering (each mode leading to one cluster) to identify the one group of data items, which is associated with the click-and-drag interaction, and select it.

The clustering is applied to a grid-based canvas where each grid has its KDE value. During the clustering, we use a simple watershed algorithm [16]: Starting with the mode with the highest KDE value, we iteratively include neighboring locations into the corresponding cluster, lowering the threshold iteratively. In every step, we either join a neighboring location to an existing cluster, or create a new cluster, if the new location is not adjacent to an existing cluster. In addition, for the clustering, we only apply the algorithm to the points in a square area with center being the start point $\mathbf{s}$ and side length being $2 \cdot r \cdot \omega$ (r is the length of interaction). Examining the user study data, we found $\omega = 1.5$ to be a reasonable value that the corresponding square area can cover all the user goals. Fig. 3B shows an according clustering result for a KDE with a relatively small kernel (shown in Fig. 3A) where the different clusters are shown in different colors and the corresponding KDE modes are located by small blue triangles. Fig. 3C shows an example of how data points are then selected (the points in the same cluster, corresponding to click-point $\mathbf{s}$, are selected and highlighted in green).

Optimizing the kernel size

The number of modes of a KDE is strongly related to the kernel size: the bigger the kernel, the fewer modes. Thus, there is a strong relation between the size of the kernel and the size of the cluster, which is used to select the data points. In the following, we describe a data-driven approach to determining an appropriate scaling factor ϕ.

Since we aim at a KDE that provides one cluster with the targeted data points, we optimize the size of the bandwidth kernel so that the size of the resulting cluster matches the size of the Mahalanobis ellipse around $\mathbf{s}$ and through $\mathbf{e}$ as closely as possible—as a measure of comparison, we are using the dice coefficient between the Mahalanobis ellipse $\mathbf{E}$ and the KDE cluster $\mathbf{C}(\phi)$:

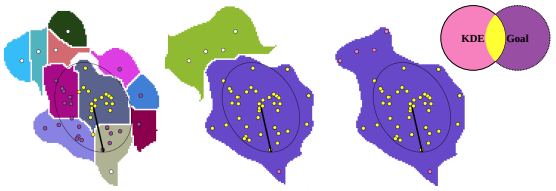

Figure 4. Clustering based on varying kernel sizes. Left: too small kernel, $s(\phi) = 0.63$; Middle: optimal size, $s(\phi) = 0.72$; Right: too big kernel, $s(\phi) = 0.64$.

$$s(\phi) = \frac{2\,|\mathbf{E} \cap \mathbf{C}(\phi)|}{|\mathbf{E}| + |\mathbf{C}(\phi)|} \tag{5}$$

where $|\mathbf{E}|$ and $|\mathbf{C}(\phi)|$ are the sizes of the Mahalanobis ellipse and the KDE cluster, respectively (evaluated grid-based). In our experiment, the searching domain for ϕ was $[\frac{1}{10}, 3]$.

The example in Fig. 4 illustrates the influence of different kernel sizes (sensitivity wrt. β) on the resulting selection. True positives (correctly selected), true negatives (correctly omitted), false positives (falsely selected), and false negatives are colored in yellow, white, pink, and purple, respectively. There are more false negatives when the kernel size is too small (Fig. 4, left) with a low similarity between the gray KDE cluster and the Mahalanobis ellipse. More false positives appear when the kernel size is too big (Fig. 4, right).

Data for parameter optimization

KDE brushing, as described so far, has two not-yet-optimized parameters: α (size of the initial selection, determining the context of the local data shape analysis) and β (overall scaling parameter on top of ϕ, influencing the kernel size). To achieve an as accurate as possible brushing result, we need information about how users would use our technique to brush and what they actually intend to select as ground truth. As the interaction and brushing targets are the same as with Mahalanobis brushing [7], we can use the same user study data to optimize α and β.

In the user study [7], six diverse datasets were used. To select these datasets, we examined their scagnostics [17], aiming at a good spread of scatterplots with varying characteristics. Scagnostics are useful to characterize scatterplots and help with the identification of relevant structures due

This article has been accepted for publication in IEEE Computer Graphics and Applications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MCG.2021.3097889

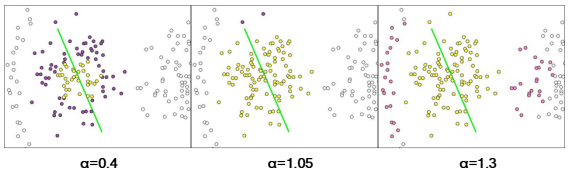

Figure 5. The influence of different values of α: too small values lead to underselection (left; false negatives in purple), too large values lead to overselection (right; false positives in pink).

to density, skewness, shape, outliers, etc. For the user study, four scatterplots were chosen from Boston Housing data with 14 variables and 91 different scatterplots, so that their scagnostics were maximally different from each other [7]. The other two datasets were one with Gaussian clusters and one with path-based spectral clusters (difficult due to the bent, elongated outer cluster).

For the study, 50 participants were asked to do 12 selections each. In every case, one scatterplot (of the six) and a rather general request ("choose a large cluster", "choose a small cluster", or "choose an elongated cluster") were given to cover a reasonable variety of cases. The actual choice of what to select was left to the user. Including a request to select an elongated cluster, was also to provoke non-trivial cases, as common in real-world applications, on top of the more simple standard cases.

During the study, the user was instructed to select a target subset (as ground truth, reported by the participants using a lasso tool), before then also providing the corresponding click-and-drag interaction that this participant would use to select the target group. Accordingly, 600 brushing cases were collected from the user study (the details are attached in the supplementary material), of which we used 400 for the optimization (see below).

Optimization

To better understand the result of this parameter optimization, we also investigated the influence of parameter α with respect to the resulting selection (see Fig. 5). The green line is the diameter of the circular area determined by the user sketch. We see that there are more false negatives (colored in purple as underselection) when α is too small. Conversely, more false positives (colored in pink when overselecting) appear, when the value of α is too big. The influence of the kernel size (scaled by β) is demonstrated in Fig. 4. We found that α and β were the most critical parameters to optimize in our technique.

Of the 600 selections from the user study [7], we randomly chose 400 as training data. As a measure for how well the selection $\mathbf{S}(\alpha, \beta)$ agrees with the user goal $\mathbf{G}$, we computed the dice coefficient for these two sets:

$$s(\alpha, \beta) = \frac{2|\mathbf{G} \cap \mathbf{S}(\alpha, \beta)|}{|\mathbf{G}| + |\mathbf{S}(\alpha, \beta)|} \tag{6}$$

If the computed selection $\mathbf{S}(\alpha, \beta)$ matches the user goal $\mathbf{G}$ perfectly, $s(\alpha, \beta) = 1$; in the case of a complete mismatch, $s(\alpha, \beta) = 0$.

Having collected the ground truth (lasso data) from the study and the click-and-release-points of the sketching interaction, we were able to conduct an optimization of α and β according to the following procedure (not involving users): Based on varying choices of α and β, we could execute our selection heuristic, using the datasets from the user study and the recorded interaction data, leading to a particular $\mathbf{S}(\alpha, \beta)$ per case—it was then straight-forward to compare $\mathbf{S}(\alpha, \beta)$ to $\mathbf{G}$ as collected during the user study, leading to a corresponding accuracy $s(\alpha, \beta)$. We started with a large matrix of different combinations of the two parameters, covering domain $[\frac{1}{10}, 10]$ that was certainly big enough. Inspecting the s-values for all these cases lead us to further examining a more detailed subset of the parameter space (basically, we refined our optimization hierarchically, doing the refinement manually). Eventually, we ended up with the following (near-optimal) values for both parameters: $\alpha = 1.05$ and $\beta = 1.05$. The optimization was done offline once.

Evaluation

Using the optimized parameters, we did an in-depth comparison in terms of accuracy, efficiency, generality and interpretability between the Mahalanobis brush [7], the CNN brush [8], and the KDE brush, using the interaction information from the user study [7] considering 252 400 points in all 600 selections.

Accuracy

Table 1 shows quantitative evaluation results for the three brushing techniques, according to a number of different measures [18]:

- **TP**: true positives (correctly selected points), total number and in percent
- **FP**: false positives (falsely selected points), total number and in percent
- **TN**: true negatives (correctly left out points), total number and in percent
- **FN**: false negatives (falsely left out points), total number and in percent
- **Accuracy**: correctly selected or left out, relative to all, $(\mathbf{TP}+\mathbf{TN})/\mathit{all}$ (the higher, the better)
- **Recall**: how much of the goal is selected, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FN})$ (higher $\leftrightarrow$ less underbrushing)
- **FPR (fall-out)**: how much of the non-goal is selected, $\mathbf{FP}/(\mathbf{FP}+\mathbf{TN})$ (lower $\leftrightarrow$ fewer **FP**)
- **FOR (false omissions)**: how much of the non-brushed was goal, $\mathbf{FN}/(\mathbf{FN}+\mathbf{TN})$ (lower $\leftrightarrow$ fewer omissions)
- **TS (threat score)**: how much of $\mathit{brush} \cup \mathit{goal}$ is **TP**, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FP}+\mathbf{FN})$ (higher $\leftrightarrow$ better)
- **Precision**: how much of the selection is goal, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FP})$ (higher $\leftrightarrow$ less overbrushing)
- **$\mathbf{F_1}$ score**: harmonic mean (precision, recall), $2\mathbf{TP}/(2\mathbf{TP}+\mathbf{FP}+\mathbf{FN})$ (higher $\leftrightarrow$ better)
- **MCC (Matthews correlation coefficient)**: measuring the quality of binary classification (the higher, the better), $(\mathbf{TP}\cdot\mathbf{TN}-\mathbf{FP}\cdot\mathbf{FN})/\sqrt{(\mathbf{TP}+\mathbf{FP})(\mathbf{TP}+\mathbf{FN})(\mathbf{TN}+\mathbf{FP})(\mathbf{TN}+\mathbf{FN})}$

According to the quantitative evaluation in the top part of Table 1, CNN brushing performs best with respect to **TP**, **FP**, **TN**, and **FN**. When comparing the two empirical models, KDE brushing gives more **TP** than Mahalanobis brushing and fewer **FN**, while it leads to more **FP** and fewer **TN**. This could indicate that KDE brushing is better at recognizing the user's brushing goal, but at the cost of more false negatives (some overbrushing).

By looking at the bottom part of Table 1, showing eight different measures for judging the quality of the classification, CNN brushing also outperforms the two empirical models in all measures. Comparing the two empirical models, KDE brushing seems to outperform Mahalanobis brushing in **recall** (how much of the goal is brushed) and in the **false omission rate** (how much of the non-brushed view is actually goal), while Mahalanobis brushing appears to be better in all other six measures.

In addition to comparing each method with the goal, we also did a threefold comparison to see the relation of the four related sets (actual goal, brushed by the Mahalanobis brush, the KDE brush, and the CNN brush), shown in Fig. 6A. The Venn diagram in Fig. 6B is an illustration of the threefold comparison: the thick green line surrounds the actual goal, the dashed violet line surrounds all that's brushed by the Mahalanobis brush, the dashed orange line surrounds what the KDE brush selects, and the dashed pink link surrounds, what the CNN brush selects (in the shown schematic, the areas do not correspond to the proportions of the respective cases).

For each point in the 600 cases from the user study [7] – 252 400 points, altogether – we check whether it belongs to the brushing goal (green), whether the Mahalanobis brush selects it (violet), whether the KDE brush selects it (orange), and whether the CNN brush selects it (pink), leading to $2^4 = 16$ possible situations per point. The relative prevalence of these situations is shown in Fig. 6A, leaving out the dominating "good" cases of *all techniques brush a goal point*, **TP**(all), 19.45% of all, *all techniques leave out a non-goal point*, **TN**(all), 73.86%, *all techniques select falsely*, 0.12% of all cases, and *all techniques fail to select*, 0.05% of all cases, emphasizing the situations, where at least one brushing technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**). The label of each situation indicates its characteristics – if one technique has a problem, then this is indicated (for ex., "FN(Mah): 1.30%" indicates the situation, when only the Mahalanobis brush fails to select a goal point); when two techniques have a problem, the opposite is done (for ex., "TN(Mah): 0.11%" indicates the situation, when only Mahalanobis brushing leaves out a non-goal point, while both other techniques incorrectly select it). As an additional mark in Fig. 6A, a combination of emojis is used to indicate the situation: a frowny indicates a problem (color: technique, filled: **FP**, empty: **FN**), while a smiley shows that the technique succeeded (filled: **TP**, empty: **TN**). Below, we briefly address all cases:

This article has been accepted for publication in IEEE Computer Graphics and Applications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MCG.2021.3097889

Table 1. Quantitative evaluation of the three brushing techniques, based on a dozen measures, computed for all 600 selections from the original user study [7] (emphasizing the best result in bold):

	TP	TP (%)	FP	FP (%)	TN	TN (%)	FN	FN (%)
Mahalanobis	50 737	20.10%	5 189	2.06%	191 682	75.94%	4 792	1.90%
KDE	52 436	20.77%	9 583	3.80%	187 288	74.20%	3 093	1.23%
CNN	**55 321**	**21.92%**	**929**	**0.37%**	**195 942**	**77.63%**	**208**	**0.08%**

	accuracy	recall	FPR	FOR	TS	precision	F_1	MCC
Mahalanobis	96.05%	91.37%	2.64%	2.44%	83.56%	90.72%	91.04%	88.51%
KDE	94.98%	94.43%	4.87%	1.62%	80.53%	84.55%	89.22%	86.18%
CNN	**99.55%**	**99.63%**	**0.47%**	**0.11%**	**97.99%**	**98.35%**	**98.98%**	**98.70%**

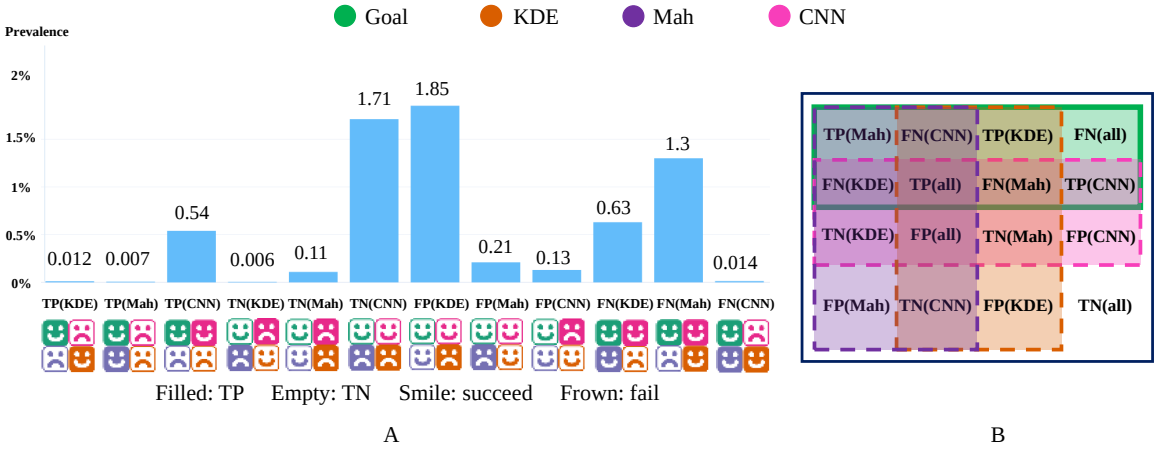

Figure 6. A three-way comparison of four sets (A): *actual goal* (green), *Mahalanobis brush* (violet), *KDE brush* (orange), *CNN brush* (pink). The cases of *all techniques correctly brush the goal* (19.45% of all), *all techniques leave out the non-goal correctly* (73.86%), *all techniques select falsely* (0.12%) and *all techniques fail to select* (0.05%) are not shown, focusing on those cases, where at least one technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**); Venn diagram, illustrating the relation between the sets (B).

- **TP**(all), **TN**(all): in most cases, all three techniques do the right thing, i.e., select a goal point or leave out a non-goal point: 19.45% are consistently well-selected goal points, 73.86% are consistently left-out non-goal points; thus, altogether, 93.31% of all cases are "good" for all three techniques!
- **FN**(KDE), **FN**(Mah.), **FN**(CNN): the indicated brush is the only one failing to select a goal point; of these, the case where the Mahalanobis brush underselects is most prevalent: 1.30%, compared to 0.63% and 0.014% of underbrushing by the KDE and CNN brushes.
- **FP**(KDE), **FP**(Mah.), **FP**(CNN): the indicated brush is the only to falsely select a non-goal point; of these, clearly the case where the KDE brush overbrushes is most prevalent: 1.85%, compared to 0.21% (Mah.) and 0.13% (CNN).
- **TP**(KDE), **TP**(Mah.), **TP**(CNN): the indicated brush succeeds to select the goal, while the other two fail; of these three, clearly the case where only the CNN brush succeeds is most prevalent: 0.54%, compared to 0.012% (KDE) and 0.007% (Mah).
- **TN**(KDE), **TN**(Mah.), **TN**(CNN): the indicated brush succeeds in not selecting a non-goal point, while the other two select it falsely; of these, also the case where only the CNN brush is right is most prevalent: 1.71%, compared to 0.11% (Mah) and 0.006% (KDE).
- **FP**(all), **FN**(all): in these rare cases, all three techniques do the wrong thing (select falsely or fail to select), amounting to 0.12% and 0.05%, respectively.

Efficiency

We evaluate the actual efficiency of sketch-based brushing in two ways: the time users spend

Table 2. Quantitative evaluation of the three brushing techniques, based on a dozen measures, computed for all 120 selections from the new user study [8] (emphasizing the best result in bold):

	TP	TP (%)	FP	FP (%)	TN	TN (%)	FN	FN (%)
Mahalanobis	18 989	21.90%	1 549	1.79%	65 921	76.03%	241	0.28%
KDE	18 927	21.83%	1 859	2.14%	65 611	75.68%	303	0.35%
CNN	**19 100**	**22.03%**	**1 286**	**1.48%**	**66 184**	**76.34%**	**130**	**0.15%**

	accuracy	recall	FPR	FOR	TS	precision	F_1	MCC
Mahalanobis	97.94%	98.75%	2.30%	0.36%	91.39%	92.46%	95.50%	94.25%
KDE	97.51%	98.42%	2.76%	0.46%	89.75%	91.06%	94.60%	93.10%
CNN	**98.37%**	**99.32%**	**1.91%**	**0.20%**	**93.10%**	**93.69%**	**96.43%**	**95.44%**

on the interaction and the computation cost of the method. The three methods adopt the same click-and-drag interaction and the average time spent is around 41% of using a lasso [7]. In terms of computation cost, CNN brushing and Mahalanobis brushing are similarly fast for small subsets. It takes, for example, around 20ms when brushing 2000 points, while the computation cost for KDE brushing is around 110ms. In the case of larger datasets, the CNN brush takes only 180ms when brushing one million points, while Mahalanobis brushing and KDE brushing take comparably long 110s and 300s for 100 000 points, respectively, which is too slow for a fluid interaction. Accordingly, all three methods enable a smooth and fluid interaction for small datasets while Mahalanobis brushing and KDE brushing are too slow for large data ($> \approx 100\,000$ points).

Generality

Generality is used to describe a model's ability to react to new and previously unseen data. To further substantiate the evaluation of the three techniques in terms of generality, we also tested the three methods on new data from another user study [8], which has not been used for training of the models at all. This user study used six new datasets (including compound data, personal hiking data, aggregation data, the omnipresent Iris data, R15 data and spirals shape data) and 10 users provided 12 selections each, leading to 120 selections in total (details in the supplementary material). The corresponding quantitative comparison is shown in Table 2 and Fig. 7, considering 86 700 points in total.

Comparing the quantitative evaluation based on the two user studies, we see that CNN brushing produces many more **FP** in the follow-up study (0.37% → 1.48%), leading also to a higher fall-out value (0.47% → 1.91%). Mahalanobis brushing and KDE brushing produce much fewer **FN** in the follow-up study (1.9% → 0.28% and 1.23% → 0.35%, respectively), while CNN produces more **FN** (0.08% → 0.15%). With respect to the other measures, KDE's **threat score** got better in the follow-up study (80.53% → 89.75%) and became more similar to the others. Mahalanobis' **threat score** improved also (83.56% → 91.39%), while CNN's **threat score** worsened (97.99% → 93.1%). Besides that, Mahalanobis' **recall** got better in the follow-up study (91.37% → 98.75%) and became more similar to the others (all methods are very good). CNN's **precision** and **accuracy** went down in the follow-up study (98.35% → 93.69%, 99.55% → 98.37%) and became more similar to the others. In addition, the **FPR** and **FOR** of KDE brushing are both largely reduced (4.87% → 2.76%, 1.62% → 0.46%), becoming more similar to the other two methods. In terms of the $\mathbf{F_1}$ **score** and **MCC**, both Mahalanobis and KDE brushing improved (91.04% → 95.5%, 88.51% → 94.25%; 89.22% → 94.6%, 86.18% → 93.1%), while CNN brushing got worse (98.98% → 96.43%, 98.7% → 95.44%).

For the threefold comparison, we see a performance decline of CNN brushing in **TP**(CNN) (0.54% → 0.08%), **TN**(CNN) (1.71% → 0.46%), **FP**(CNN) (0.13% → 0.15%) and **FN**(CNN) (0.14‰ → 0.51‰). For the empirical models, KDE is better in **TP**(KDE) (0.12‰ → 0.14‰), **TN**(KDE) (0.06‰ → 0.21‰), **FP**(KDE) (1.85% → 0.37%) and **FN**(KDE) (0.63% → 0.18%), while Mahalanobis brushing is better in **TN**(Mah) (0.11% → 0.42%), and **FN**(Mah) (1.3% → 0.1%) but not in **TP**(Mah) (0.07‰ → 0.05‰), and **FP**(Mah) (0.21% → 0.41%).

While almost all measures got better for both

This article has been accepted for publication in IEEE Computer Graphics and Applications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MCG.2021.3097889

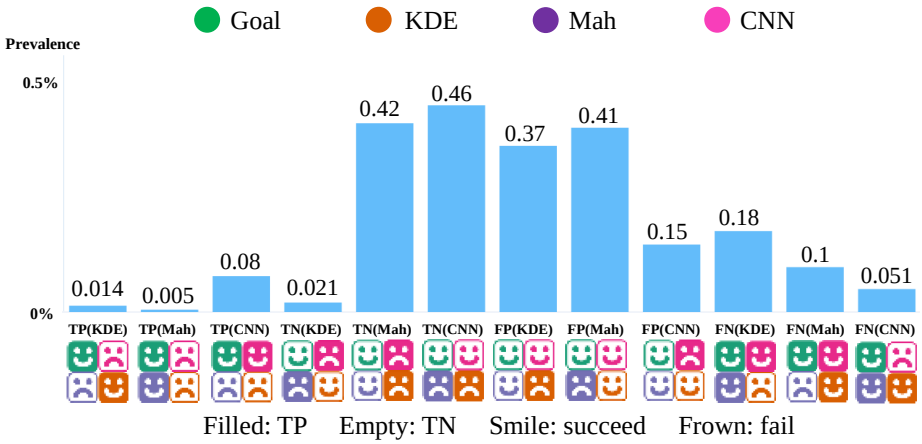

Figure 7. Threefold comparison between the Mahalanobis brush, KDE brush and the CNN brush based on the user's goal from a follow-up user study (details in the text). Note that all relative prevalences are below 0.5%.

Mahalanobis brushing and for KDE brushing in the second user study, they all got worse for CNN brushing – at least a bit. It is important to see, however, that CNN brushing still outperformed both other methods in all indicators (even though they are much more similar in the follow-up study). This could reveal one disadvantage of the CNN brush, namely that it is less general, when compared with the empirical models.

Interpretability

Interpretability refers to how much a human can understand the model's process and result. Although the three brushing methods achieve all good results (>90% accuracy) in two studies, we were curious to see how the methods perform in particularly difficult cases (for example, when brushing the large ring shapes in the original study or the spiral shapes in the follow-up study). Therefore, we had a closer look at special cases and found that the average accuracy for brushing the ring shapes are around 53%, 51% and 95% for KDE brushing, Mahalanobis brushing and CNN brushing, respectively, while they achieve 54%, 50% and 41% accuracy when selecting the spiral shapes. We can see that KDE brushing is slightly better than Mahalanobis brushing in these cases due to its nonlinear model while the performance of CNN brushing falls in the follow-up cases, possibly because of issues related to overfitting.

In addition to the "bad cases" analysis, we observed that some prediction results of CNN brushing are unreasonable. CNN brushing can, for example, accurately find the border of a cluster while some scattered points inside the cluster are not selected. This situation is not possible in Mahalanobis brushing and KDE brushing, representing an increased uncertainty and low interpretability of the DL-based model.

Summary

In general, we could not see that KDE brushing would significantly outperform Mahalanobis brushing based on the comparison between KDE brushing and Mahalanobis brushing, even though we see slightly better results for KDE brushing in the follow-up study and in some nonlinear-shape cases. An according assumption was originally made, because we thought that more carefully considering the local data distribution should help to further improve the technique's accuracy (as a nonlinear method, KDE brushing should have much better abilities to adapt to nonlinear structures in the data). So far, we cannot rule out that we have overlooked another limitation when realizing the KDE-based approach – either a conceptual one, or a limitation of our implementation. Accordingly, we see it still possible that another solution could achieve a further improved accuracy. To outperform CNN brushing, however, seems like a tall order.

We note that empirical modeling comes with the advantage of an explainable result (for example, we know how different values of α and β influence the results), while the excellent performance of the DL-based model comes at the

cost of a poor interpretability (including some uncertainty concerning the stability of its predictive power). This comparison leads to the interesting question of how much accuracy we are willing to sacrifice for a good interpretability.

Conclusion and future work

In this paper, we presented our attempt to improve Mahalanobis brushing by incorporating kernel density estimation to increase its accuracy. Although more information is taken into account for modeling the KDE-based model, we have not seen a significant improvement compared to the simpler Mahalanobis brush. Based on this result, we think that the increased cost of incorporating KDE could have come with an overdesign issue. When compared with deep learning, we found that its black-box nature results in a questionable interpretability (but with excellent accuracy), whereas the results based on the empirical model are explainable (even though not as good as the ones based on the learned model). Considering its reduced robustness, the DL-based method appears to be (a bit) less stable and a bit more unpredictable, even though it does have the best performance, after all. It is unclear, however, how to weigh in all factors, when comparing the overall performance for model selection.

In the future, we see several opportunities to further extend our work, including:

- Combining advantages of both sides, i.e., empirical modeling *and* deep learning. We imagine, for example, to automatically learn the kernel size or to design the deep learning input on the basis of the KDE.
- Investigating more closely, why KDE brushing did not outperform the Mahalanobis brush, and make a new attempt to further improve it.
- A sensitivity study with respect to the (optimized) parameters of the empirical models.
- Exploring other machine learning approaches to develop a new brushing technique which outperforms CNN brushing.

We hope, also, that this work can inspire further related research, especially in visualization for model design and model selection.

Acknowledgements

We thank the participants in the user studies for helping with our research. Parts of this work have been done in the context of CEDAS, Center for Data Science, at the University of Bergen, Norway.

REFERENCES

1. Roberts, Jonathan C.: “State of the art: Coordinated and multiple views in exploratory visualization”; Fifth int’l conf. on coordinated and multiple views in exploratory visualization (CMV 2007), pp. 61–71, 2007 (conf. proc.)
2. Becker, Richard A., and William S. Cleveland: “Brushing scatterplots”; Technometrics 29(2): 127–142, 1987 (journal article)
3. Martin, Allen R., and Matthew O. Ward: “High Dimensional Brushing for Interactive Exploration of Multivariate Data”; Proc. of the 6th Conf. on Visualization, pp. 271–279, 1995 (conf. proc.)
4. Yang, Ming-Hsuan, and Narendra Ahuja: “Face detection and gesture recognition for human-computer interaction”; The Int’l Series in Video Computing, Vol. 1, Springer, 2001 (book)
5. Hauser, Helwig: “Generalizing focus+context visualization”; In “Scientific visualization: The visual extraction of knowledge from data”, pp. 305–327, Springer, 2006 (book chapter)
6. Hauser, Helwig, Florian Ledermann, and Helmut Doleisch: “Angular brushing of extended parallel coordinates”; Proc. of IEEE Symp. on Information Visualization, pp. 127–130, 2002 (conf. proc.)
7. Fan, Chaoran, and Helwig Hauser: “User-study based optimization of fast and accurate Mahalanobis brushing in scatterplots”; Proc. of the conf. on Vision, Modeling and Visualization (VMV 2017), pp. 77–84, 2017 (conf. proc.)
8. Fan, Chaoran, and Helwig Hauser: “Fast and accurate CNN-based brushing in scatterplots”; Computer Graphics Forum 37(3): 111–120, 2018. (journal article)
9. Radoš, Sanjin, Rainer Splechtna, Krešimir Matković, Mario uras, M. Eduard Gröller, and Helwig Hauser: “Towards quantitative visual analytics with structured brushing and linked statistics”; Computer Graphics Forum 35(3): 251–260, 2016 (journal article)
10. Parzen, Emanuel: “On estimation of a probability density function and mode”; The annals of mathematical statistics 33(3): 1065–1076, 1962 (journal article)
11. Fan, Chaoran, and Helwig Hauser: “On KDE-based brushing in scatterplots and how it compares to CNN-

based brushing"; Proc. of Machine Learning Methods in Visualisation for Big Data 2019 (conf. proc.)

12. Koytek, Philipp, Charles Perin, Jo Vermeulen, Elisabeth André, and Sheelagh Carpendale: "Mybrush: Brushing and linking with personal agency"; IEEE Trans. on Visualization and Computer Graphics 24(1): 605–615, 2017. (journal article)
13. Novotný, Matěj, and Helwig Hauser: "Similarity brushing for exploring multidimensional relations"; Journal of WSCG 14: 105—112, 2006 (journal article)
14. Fan, Chaoran, and Helwig Hauser: "Personalized sketch-based brushing in scatterplots"; IEEE Computer Graphics & Applications 39(4): 28–39, 2019 (journal article)
15. Mahalanobis, Prasanta C.: "On the generalised distance in statistics"; Proc. National Institute of Science, India, Vol. 2: 49–55, 1936 (journal article)
16. Florek, Martin, and Helwig Hauser: "Quantitative data visualization with interactive KDE surfaces"; Proc. of the 26th Spring Conf. on Computer Graphics, pp. 33–42, 2010. (conf. proc.)
17. Tukey, John W., and Paul A. Tukey: "Computer graphics and exploratory data analysis: An introduction"; Proc. of the Sixth Annual Conf. and Exposition: Computer Graphics, Vol. 85, No. 3, pp. 773–785, 1985. (conf. proc.)
18. Powers, David M. W.: "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation"; arXiv:2010.16061 (arXiv preprint)

Chaoran Fan is currently working toward the Ph.D. degree at the Department of Informatics, University of Bergen, Bergen, Norway. His research interest focuses on information visualization, specifically for improving the user interaction in visual analytics by leveraging the machine learning knowledge. Contact him at fantasyfans2012@gmail.com

Helwig Hauser is professor in visualization at the University of Bergen, Norway, where he also leads the Center for Data Science, CEDAS. His research interests include visual data exploration and analysis, as well as visualization in general. Contact him at Helwig.Hauser@UiB.no