On KDE-based brushing in scatterplots and how it compares to CNN-based brushing

Chaoran Fan and Helwig Hauser
University of Bergen, Norway, ii.uib.no/vis

Abstract
In this paper, we investigate to which degree the human should be involved into the model design and how good the empirical model can be with more careful design. To find out, we extended our previously published Mahalanobis brush (the best current empirical model in terms of accuracy for brushing points in a scatterplot) by further incorporating the data distribution information that is captured by the kernel density estimation (KDE). Based on this work, we then include a short discussion between the empirical model, designed in detail by an expert and the deep learning-based model that is learned from user data directly.

CCS Concepts
- Human-centered computing → Interaction techniques;
- Computing methodologies → Optimization algorithms;

1. Introduction
Linking and brushing is a prevalent interaction technique for data exploration and analysis in coordinated multiple views. Becker and Cleveland [BC87] were the first to come up with a theory of brushing, defined as an interactive method to select data points by using simple geometries (square, circle, or a polygon).

Since brushing is central in modern visual analytics systems, it has attracted a considerable amount of research and most of the proposed techniques can be evaluated by the following two criteria.

- **efficiency**—how fast is the brushing interaction; does it enable a fluid data exploration [EVMJ∗11,TKBH17]? Clicking one point, for example, is a minimal interaction that is also most efficient.
- **accuracy**—to which degree does the brushing interaction lead to an accurate selection of the data subset, which the user actually wished to select? The lasso, for example, is a brushing tool which can be used to specify the brush region with 100% accuracy.

In order to optimize both criteria in one technique as well as possible, data-driven interaction techniques became popular recently. This type of method is usually based on a fast, sketch-based user interaction (for example, click-and-drag) and a heuristic, which transforms the sketching interaction into the actual data selection, based on the underlying data visualization. To improve the accuracy, the parameters of the technique can be automatically optimized by using data from a user study.

Our previously published work—Mahalanobis brush [FH17]—is a typical example of a data-driven technique, taking the local covariance information into account, forming the basis for a local Mahalanobis metric. The parameters are then optimized based on data from a user study. The quantitative evaluation shows that it can achieve ≈92% accuracy based on a fast interaction (click-and-drag). Later, inspired by the success of deep learning methods in a wide range of fields, we developed a CNN-based brushing technique [FH18] and achieved the so far best accuracy (≈97%). For this method, we converted the interaction and data information into an image-based input to the network and allowed the network to learn a model itself.

As machine learning [FH18] had outperformed empirical modeling [FH17], we were curious about how much the empirical model can be improved with a more sophisticated design and whether it can beat the deep learning approach. In this paper, we report our attempt to construct a best-possible empirical model by further extending the Mahalanobis brush, incorporating kernel density estimation (KDE) [Par62], and informing a clustering step that returns one of the clusters as the data selection. The main contribution of this paper includes our extension of the empirical model for brushing points in a scatterplot, and a comparison of the two approaches, as well as an according discussion.

2. Related work
Brushing in scatterplots is often based on simple geometric shapes such as a rectangle or an ellipse to select data items, or a lasso is used to specify the brush accurately. Several extensions to simple brushing have been proposed over the years.

Martin and Ward [MW95] suggest logical combinations of brushes, including unions, intersections, negations, and XOR operations, enabling the user to configure composite brushes.

MyBrush was suggested by Koytek et al. [KPV∗17] in order to extend brushing and linking by incorporating personal agency. It
offers users the flexibility to configure the source, link, and target of multiple brushes.

Similarity brushing [NH06b, MKO’08] is a classical example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user uses a swift and approximate gesture (for ex., drawing an approximate shape that the data should follow) and then a similarity measure is designed to identify, which data items actually are brushed.

Our previously published Mahalanobis brush [FH17] is an extension of the original Mahalanobis brush, introduced by Rados et al. [RSM’16], in which the user clicks into the center of a coherent data subset to select it. The local covariance information is used to form a selection metric. The main drawbacks of this method are a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size. Both issues were successfully addressed in our method [FH17].

Later, we exploited machine learning and developed CNN-based brushing in scatterplots [FH18]. This technique uses the same interaction as the Mahalanobis brush [FH17], while achieving the so far best accuracy.

3. KDE brushing in scatterplots

Figure 1 shows an overview of the new, KDE-based brushing technique. We keep the simple click-and-drag interaction for sketching the data subset (click into the middle of the targeted data subset and drag the pointer to the outer boundary of the subset). The click-point \( s = (s_x, s_y) \) and the end-point \( e = (e_x, e_y) \) of the drag-interaction provide us with a first hint concerning the size of the data subset, which the user wishes to brush. Similarly to the Mahalanobis brush [FH17], we first consider a circular data subset, centered around the start-point of the interaction, 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 that refines this data subset selection, based on the local covariance information. After a sufficiently close convergence of this iteration, we make a selection of data points, based on a kernel density estimation of the data, using the local covariance information as a basis for specifying the kernel. In the following, we go into more details with respect to the individual components of our solution.

The Mahalanobis distance is a distance measure introduced by P. C. Mahalanobis [Mah36], which is used for helping with the identification and analysis of patterns in the data. The Mahalanobis distance between vectors \( a \) and \( b \) can be defined by

\[
d_E(a, b) = \sqrt{(a - b)\Sigma^{-1}(a - b)}
\]  

(1)

Initially, we consider a circular area with the radius \( \alpha \cdot d_E(s, e) \), where \( \alpha \) is a weighting factor and \( d_E(s, e) \) is the Euclidean distance between \( s \) and \( e \). All data points within this circle are used to compute the first instance of the local covariance information, \( \Sigma_1 \). Next, we consider all points within a Mahalanobis ellipse, based on \( \Sigma_1 \) and sized according to \( d_E(s, e) \). Usually, this leads to a new data subset, which is similar to the data subset as determined by the initial circle, but more closely following the underlying data structure. To obtain an even better sample, we refine the sample iteratively by replacing them with the points in the Mahalanobis ellipse which is updated every iteration according to the covariance of the samples in last iteration. However, we observe that this process sometimes can lead to small fluctuations, including/excluding a few data points in consecutive iterations. Therefore, we stabilize the convergence by enabling the partial consideration of data points, leading to a solution that is based on the weighted covariance matrix [Gou09].

The details of this procedure are introduced in [FH17].

4. Kernel Density Estimation

Kernel density estimation (KDE) is a popular method for data analysis in the field of statistics, which was introduced by Rosen-
Assuming that \( \{x_i\}_{i=1}^n \) is a sample of \( n \) \( d \)-dimensional vectors drawn from a common distribution, described by a particular density function, then KDE can be used to estimate this density function as

\[
f_H(x) = \frac{1}{n} \sum_{i=1}^{n} K_H(x - x_i)
\]

with \( H \) being the \( d \times d \) kernel matrix (symmetric and positive definite). The choice of matrix \( H \) is the single most important factor affecting the main characteristics of \( f_H \) [WJ95]. Figure 2 shows four results from a 2D KDE with increasing size of \( H \).

\[
K_H(x) = \left( |H|^{-\frac{1}{2}} \right) \sum_{i=1}^{n} \left( |H|^{-\frac{1}{2}} K \left( |H|^{-\frac{1}{2}} x_i \right) \right)
\]

In order to realize an appropriate scaling of our kernel, we make use of an eigendecomposition of \( \Sigma = \Lambda \Lambda^\top \) with eigenvectors \( V \) and eigenvalues \( \Lambda \). This leads to the following, scaled versions of \( |H|^{-\frac{1}{2}} \) and \( H^{-\frac{1}{2}} \):

\[
|H|^{-\frac{1}{2}} = \beta \phi \Lambda^{-\frac{1}{2}} \quad \text{and} \quad H^{-\frac{1}{2}} = (\beta \phi \Lambda)^{-\frac{1}{2}} V^\top
\]

Figure 3 demonstrates the influence of different kernel sizes on the brushing results. The true positives (correctly selected), true negatives (correctly non-selected), false positives (falsely selected) and the false negatives (falsely left out) are colored in yellow, white, pink and purple, accordingly. We see that 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 \( \phi \).

Since we aim at a KDE that will provide one cluster in order to select 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 \( s \) and through \( e \)—as a measure of comparison, we are using the dice coefficient between the Mahalanobis ellipse \( E \) and the KDE cluster \( C(\phi) \) [Dic45]:

\[
s(\phi) = \frac{2 |E \cap C(\phi)|}{|E| + |C(\phi)|}
\]

where \( |E| \) and \( |C(\phi)| \) are the sizes of the Mahalanobis ellipse and the KDE cluster, respectively (evaluated as a grid-based measure). The example shown in Figure 4 demonstrates the influence of different kernel sizes on the brushing results. The true positives (correctly selected), true negatives (correctly non-selected), false positives (falsely selected) and the false negatives (falsely left out) are colored in yellow, white, pink and purple, accordingly. We see that there are more false negatives when the kernel size is too small (showed in the left) with a low similarity between the gray KDE cluster and the Mahalanobis ellipse. Conversely, more false positives appear when the kernel size is too big (on the right).
Our technique, as described so far, has two not-yet-optimized parameters: $\alpha$ (initial selection, determining the context of local data shape analysis; too small values of $\alpha$ lead to underselection while too large values to overselection) and $\beta$ (overall scaling parameter on top of $\phi$, influencing the kernel size). In order to achieve as accurate as possible brushing, we have to get information about how users would use our technique to brush and what they actually wanted to select from the dataset (ground truth). As the interaction and brushing target designed in our method are same as the previously published Mahalanobis brush [FH17], we directly use the data from the published user study. Based on this information, we then did an optimization of $\alpha$ and $\beta$.

In the previously published user study, six representative datasets are provided. The datasets selected for the user study are based on scagnostics [TT88], aiming at a healthy spread of scatterplots of different type. For the user study, 50 participants were invited and asked to do 12 selections each. In every case, a particular scatterplot (one out of six) and a particular request (choose a large cluster/a small cluster/an elongated cluster) were given. Then the participant was instructed to choose a target data subset to select (ground truth), then also providing the corresponding click-and-drag interaction, which this participant would use to select the target group. Accordingly, 600 brushing cases were collected from the user study.

6. Optimization and Evaluation

From the 600 selections that we got from the published user study, we randomly chose 400 selections as training data. As a measure for how well the computed selection $S(\alpha, \beta)$ agrees with the user goal $G$, we compute the dice coefficient of these two sets:

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

If the computed selection $S(\alpha, \beta)$ matches the user goal $G$ 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 from the sketching interaction, we were able to conduct an optimization of $\alpha$ and $\beta$ according to the following procedure (not involving users anymore): Based on varying choices of $\alpha$ and $\beta$, we could execute our selection heuristic, using the datasets from the user study and the recorded interaction data, leading to a particular $S(\alpha, \beta)$—this was then straightforward to compare to $G$ as collected during the user study, leading to a corresponding $s(\alpha, \beta)$. We started with a large matrix of different combinations of the two parameters, covering a domain, which for sure was big enough. Inspecting the $s$-values for all these combination 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 optimal values for both parameters: $\alpha = 1.05$ and $\beta = 1.05$.

The whole optimization process is done offline once and it takes 4 hours for the computation.

Using the optimized parameters, we did an accuracy comparison with the previously published Mahalanobis brush [FH17] using the interaction information from the published study. The overall accuracy for the new technique is $\approx 90\%$ which is slightly lower than the one of the Mahalanobis brush ($\approx 92\%$) and does neither outperform the CNN-based brush [FH18] with its accuracy of $\approx 97\%$. So far, we have been unable to evaluate, why this the case – we had assumed that more carefully considering the local data distribution should help to further improve the technique’s accuracy (as a nonlinear method, the KDE should have better power to adapt to nonlinear structures in the data). Unfortunately, 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 imagine that another solution can in fact achieve a further improved accuracy.

As another point of this discussion, we note that empirical modeling comes with the advantage of being explainable (for example, we know how different values of $\alpha$ and $\beta$ influence the results in our approach), while the excellent performance of the deep learning model comes at the price 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.

7. Conclusion and future work

In this paper, we present our attempt to improve the Mahalanobis brush by incorporating KDE to further improve its accuracy and we investigate the influence of human expertise during model design. Although more information is taken into account for modeling the KDE-based model, we have not seen an improvement compared to the simpler Mahalanobis brush. Based on this result, we realize that the incremental cost of incorporating KDE could be an over-design problem. In addition, when compared with deep learning, we found that its black-box mechanism leads to a poor interpretability, while the results based on the empirical model are explainable. It is unclear, however, how to weigh this factor in, when comparing the overall performance.

In the future, we see potential in taking innovative advantage of both sides—empirical modeling and machine learning. We imagine, for example, to automatically learn the kernel size or to design the deep learning input on the basis of the KDE. In addition, we are also interested to investigate the reasons why the KDE-based method performs worst than the Mahalanobis brush.
References


