EuroVis Workshop on Visual Analytics (2024)
M. El-Assady and H.-J. Schulz (Editors)

Computing Fast and Accurate Decision Boundary Maps

Cristian Grosu, Yu Wang, and Alexandru Telea

Department of Information and Computing Science, Utrecht University, Netherlands

Abstract

Decision boundary maps (DBMs) are image representations of the behavior of trained machine learning classification models. They are used in examining how the model partitions its data space into decision zones separated by decision boundaries and how this partition is influenced by the training data. However, all current DBM methods require significant computational effort, which precludes their use in interactive visual analytics scenarios. We present FastDBM, a set of techniques for the fast computation of DBMs. Our methods can accelerate any existing DBM algorithm by over one order of magnitude, yield results very similar to the original DBM methods, have a single parameter to set (with good presets), and are simple to implement. We demonstrate our method on various combinations of DBM techniques, datasets, and classification models.

CCS Concepts

• ***Human-centered computing*** → *Visualization design and evaluation methods;*

1. Introduction

Decision boundary maps (DBMs) are simple but effective instruments for the visual depiction of the behavior of trained classification models [TMW24]. Simply put, a DBM maps (a part of) the high-dimensional data space onto which the model operates to a 2D image space. Next, image pixels are used to represent the model's behavior at samples from the data space, by encoding the labels inferred by the classifier, and optionally classification confidence, via color hue, respectively color saturation. This way, the image shows so-called *decision zones*, *i.e.* areas where the classifier infers the same label; and *decision boundaries*, *i.e.*, locations in the data space where the model changes decision.

Several techniques for computing DBMs have been proposed [RHT18, SHH20, OEHJT22]. However, all these techniques share the fact that computing the DBM image, even at quite small resolutions of hundreds of pixels squared, is very time consuming (tens of seconds up to tens of minutes, depending on the DBM technique [WMT23]. In turn, this precludes the usage of such DBMs in visual analytics scenarios where users aim to *improve* a classification model by *e.g.* changing its hyperparameters or performing data pseudo-labeling in active learning settings [BTF18, SHH20, BGTF21]. Indeed, in such scenarios, users need to iteratively, and ideally interactively, change the classifier, compute a new DBM, visually explore it, and next decide how to fine-tune their next changes to arrive at the desired result. As such, the potential of DBMs for helping classifier engineering is currently limited by their computational scalability.

We address this problem by proposing FastDBM, a set of techniques that accelerate the computation of DBM images. Our techniques introduce only a very low error rate as compared to the ground-truth DBM computation – in practice, only a few tens of pixels, located on the decision boundaries, are different. Finally, our techniques are simple to implement, have no hidden parameters to be set, and can accelerate any existing DBM computation method for any classification model in a black-box manner, that is, without needing access to their internals. We demonstrate our proposal by speed and accuracy comparisons of FastDBM with ground-truth DBM computations for several datasets and classifiers.

2. Related work

Let $D = \{\mathbf{x}_i\} \subset \mathbb{R}^n$ be a high-dimensional dataset with samples $\mathbf{x}_i$. A classification model $M : \mathbb{R}^n \to C$, trained and/or tested on D, maps samples from the data space to a categorical (label) domain C. A *decision map* is an image I that captures M's behavior, computed as follows. Let $P : D \to \mathbb{R}^2$ be a dimensionality reduction, or projection, such as t-SNE [vdMH08], UMAP [MH18], PCA [Jol02], or any other similar methods [NA18, EMK*21]. Let $P(D)$ be the mapping of D to a 2D scatterplot computed using P. Using the pair $(D, P(D))$, one can compute a so-called inverse projection $P^{-1} : \mathbb{R}^2 \to \mathbb{R}^n$. Next, each pixel $\mathbf{y} \in I$ is mapped to a data sample $P^{-1}(\mathbf{y})$ and colored to reflect the model's inferred label $M(P^{-1}(\mathbf{y}))$ and, optionally, the classification confidence $c(P^{-1}(\mathbf{y})) \in [0,1]$ for that label. Figure 2a shows such a decision map for the well-known MNIST dataset [LeC10] with confidence encoded into color saturation. By examining I, one can reason about how the classifier's decision zones relate to its training data [OEHJT22, OEH*23]; and how far from these data are its predictions reliable [REJT19, SHH20].

Key to the computation of DBMs are the choice of P and P^{-1}, since P is used to compute P^{-1} and P^{-1} is used to compute the DBM, as explained above. Several combinations are possible. Early

on, Rodrigues *et al* [RHT18] used t-SNE and LAMP [JPC*11] for P and iLAMP [dBD*12] for P^{-1}, respectively. Since iLAMP is quite slow, Espadoto *et al* [ERH*19] proposed the NNInv technique to compute P^{-1} via deep learning. This approach was next refined in terms of quality of the produced DBMs by Self-Supervised Decision Maps (SDBM, [OEHJT22]) which exploited jointly deep learning P and P^{-1} [EHT21], based on an autoencoder approach [HS06]. Separately, DeepView [SHH20] proposed discriminative dimensionality reduction to construct DBMs using UMAP for P^{-1}.

Rodrigues *et al.* [REJT19] have presented a qualitative evaluation of DBM techniques involving 28 methods for P and two methods (iLAMP and NNInv) for P^{-1}. Their results showed that, among the studied techniques, t-SNE and UMAP (for P) and NNInv (for P^{-1}) produce DBMs which are best in line with the expected behavior of the visualized models. More recently, Wang *et al.* [WMT23] have presented a detailed evaluation of DBM techniques from the perspective of quality and computational scalability. Similarly to Rodrigues *et al.*, they found that t-SNE and UMAP (for P) and NNInv (for P^{-1}) yield very good results, surpassed only by DeepView. However, all these techniques are quite slow – on a typical commodity PC, computing a DBM for resolutions of 250^2 pixels takes about 10 seconds for all tested methods except DeepView; for DeepView, this took over two hours. As we shall see in Sec. 4, these costs increase quadratically with the DBM resolution – and higher resolutions are needed to create DBMs in which users can see subtle details such as the exact shape of decision boundaries close to groups of data samples. This makes current DBM methods poorly suitable for visual analytics scenarios that require fast recomputation of the DBM upon re-training of the studied model.

3. Fast DBM computation

We start by observing that, for an image I of $n \times n$ pixels, the complexity of the standard DBM method is $O(n^2K)$, where K is the cost of a single $M(P^{-1}(\cdot)))$ operation. Decreasing K is hard if we allow any generic inverse projections P^{-1} and classifier models M. Hence, to improve speed, we first focus on reducing the n^2 term.

A classification model M aims to fit its decision boundaries to surround same-class training points. Training points located close to each other in data space will typically have the same labels. As such, (1) decision zones are relatively large and compact subsets of $\mathbb{R}^n$. In the same time, (2) the boundaries of the decision zones must be sufficiently *smooth* so as to allow for generalization without overfitting. As a consequence of (1) and (2), a given M has in general a relatively *small* number of compact, smooth-boundary, decision zones (not necessarily one per class). A DBM should preserve the above properties so that users can infer the actual decision boundaries and zones from the visualized ones [OEHJT22, OEH*23]. As such, the visualized decision boundaries and zones in the DBM will also respect the above properties (small number of compact, smooth-boundary, decision zones). We next use this observation to devise three acceleration strategies which are presented next.

Binary split: We start by computing the DBM image, using any desired existing method (Sec. 2), at a resolution of B^2 blocks. Each such block represents a square of $\frac{n}{B} \times \frac{n}{B}$ pixels from I. For each block b, we evaluate the label $l_b = M(P^{-1}(\mathbf{y}))$ at its central pixel $\mathbf{y}$.

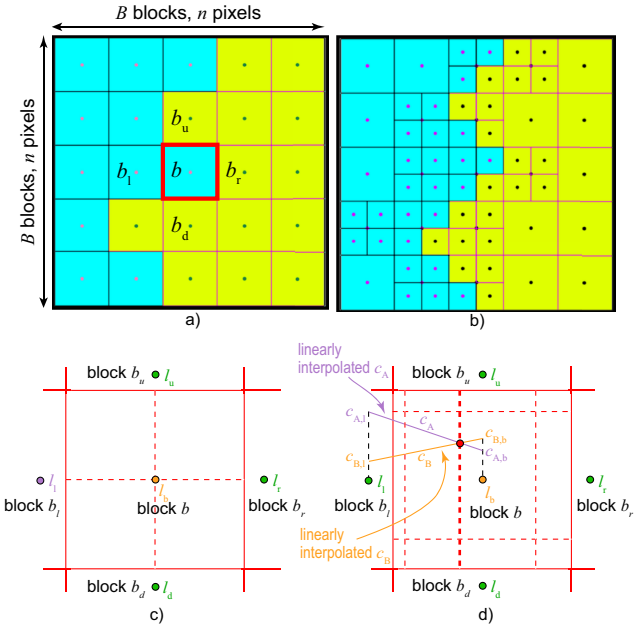

Figure 1: *Binary (a-c) and confidence-based (d) splitting heuristics.*

Figure 1a shows this for a binary classifier (cyan and yellow are the two classes; the current block b is in bold red). Let l_u, l_d, l_l, l_r be the labels computed similarly for the up, down, left, and right neighbor blocks of b. Let κ be the number of neighbors with labels different from l_b. If $\kappa = 0$, then b is surrounded by same-label blocks. If we assume that a decision zone in the DBM is locally *thicker* than $\frac{n}{B}$ pixels, no decision boundary can cross b. Hence, we can assign l_b to all pixels in b. If $\kappa > 0$, we split b into four equal smaller blocks (the resulting block-set is shown in Fig. 1b). We repeat the process, in an quadtree-like fashion, until we arrive at pixel-sized blocks or blocks that do not need splitting any more. During this process, we note that, to ensure an uniform refinement allover the image, (1) splitting larger blocks first is more advantageous than splitting smaller ones; and (2) splitting blocks having several neighbors with different labels is more advantageous than splitting blocks having a single such neighbor since the former cover more decision boundary fragments. Following the observations (1) and (2), we keep blocks to split in a priority queue sorted descendingly on $d \cdot d \cdot \frac{\kappa}{N}$ where d is the size of a block, κ its number of different-label neighbors, and N is its neighbor count, *i.e.*, 4 for blocks inside fhe DBM, 3 for blocks on the DBM border, and 2 for blocks on the DBM corners.

As Sec. 2 outlines, a DBM can also show at each pixel, apart from labels, the *confidence* of the visualized model M. Yet, per block, we have a single data sample $P^{-1}(\mathbf{y})$, computed at the block's center pixel $\mathbf{y}$. This is fine for class labels as these are constant for entire decision zones, thus also per block, as per our splitting heuristic. Yet, confidence varies even within a decision zone, being typically small close to decision boundaries and larger deeper in a zone (see *e.g.* Fig. 2a). Hence, confidence varies also within a block. We avoid computing additional confidence values apart from c at $\mathbf{y}$ by interpolating these values, computed at the blocks' centers $\mathbf{y}$, using nearest-neighbor, bilinear, or bicubic schemes.

Confidence split: The binary split follows a simple bisection idea to locate, up to pixel precision, the places where decision boundaries are. We can use the confidence values c at the block centers $\mathbf{y}$ to refine this process as follows. Take the block b configuration shown in Fig. 1c. Binary splitting would divide this cell along the dashed lines in the image. Now, consider the confidence values c_A and c_B for the inferred classes purple, respectively orange, sampled at the centers of the cells b_l and b, denoted next by $c_{A,l}$, $c_{B,l}$, $c_{A,b}$, and $c_{B,b}$ (see Fig. 1d). We can next bilinearly interpolate these values over the image space I and find the point where $c_A = c_B$ – that is, the red point in Fig. 1d where the purple and orange lines intersect. This is on (or close, depending on the precision of our interpolation) the decision boundary between the orange and purple classes. As such, it is likely a good point to split cell b. We do this split following the thick dashed line in Fig. 1d. We proceed in the same way for all class values with respect to all four boundaries of cell b. This yields a possible set of 2, 3, 4, 6, or 9 cells that split b as opposed to the four square cells created by the binary split. The dashed red lines in Fig. 1d indicate the splitting positions. Confidence over blocks b is next interpolated as in the binary split method.

Confidence sampling: Interpolating c over the image I allows us to devise another DBM construction method. Given an initial resolution of B^2 blocks, we compute c at block centers $\mathbf{y}$, for *all* inferred $|C|$ classes, and next interpolate these over I by nearest-neighbor, bilinear, or bicubic techniques – as described above, but at the single, initial, block resolution level. In contrast to binary and confidence split, we now do not refine the initial blocks. Next, for each pixel $\mathbf{y} \in I$, we sample the interpolated confidences for all $|C|$ classes and assign to $\mathbf{y}$ the class label with highest confidence.

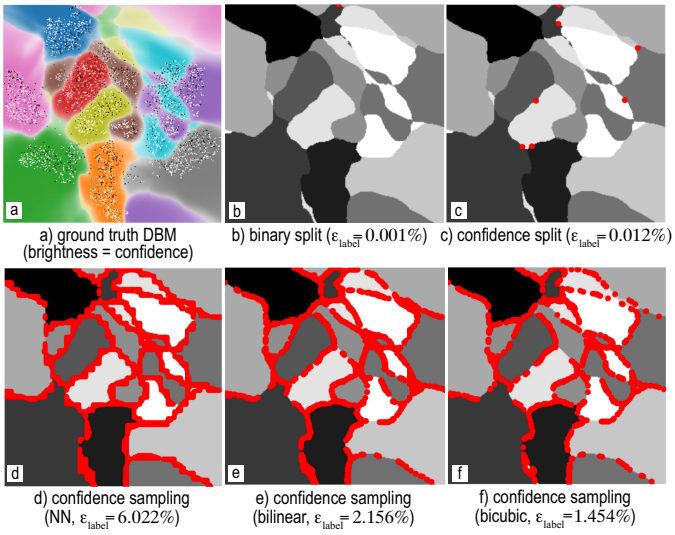

Figure 2: *a) Ground-truth DBM with labels (color) and confidence (saturation), MNIST dataset. b-f) Class assignment errors for FastDBM method variants,*

4. Evaluation

We compare our FastDBM results with ground-truth DBMs using two metrics, as follows.

Label errors: We would like to obtain the same labels for the same pixels in the FastDBM image I_{fast} and the ground-truth DBM image I. We evaluate this by the error

$$\varepsilon_{label} = \frac{1}{n^2} \sum_{1 \le i \le n, 1 \le j \le n} \delta(I(i,j), I_{fast}(i,j)),$$

where $\delta(a,b)$ is 0 if $a = b$ and 1 otherwise.

Confidence errors: Also, we would like that our interpolated confidence c_{fast} is as close as possible to the ground-truth one c. We evaluate this by the normalized MSE error

$$\varepsilon_{conf} = \frac{\sum_{1 \le i \le n, 1 \le j \le n} (c(i,j) - c_{fast}(i,j))^2}{\sum_{1 \le i \le n, 1 \le j \le n} c(i,j)^2}.$$

Figure 2 shows our results for the MNIST dataset [LeC10], classified with a simple classification deep learning network M (flatten layer, dense 10-unit layer and softmax activation, trained for 20 epochs, 3.5K training samples, 1.5K test samples); t-SNE and NNInv used for P and P^{-1}; and a DBM image size of $n = 256$ pixels squared, $B = 32$ blocks along each image dimension. Figure 2a shows the ground-truth DBM with labels and confidence color- respectively saturation-coded. Images (b-f) show the results of our binary split, confidence split, and confidence sampling methods, the latter using nearest neighbors (NN), bilinear, and bicubic interpolation. Red points show locations where the ground-truth labels differ from FastDBM's results. Values ε_{label} indicate the fraction these red pixels are of the total image. As visible, our methods yield practically the same DBMs, with only a small number of different pixels. The binary split method is best – only 8 pixels of the 256^2 are different. The confidence sampling method is worst; yet, for this method, all errors appear within 2-3 pixels from the decision boundaries. This is most likely since confidence varies smoothly inside decision zones but more rapidly close to boundaries (see Fig. 2a), so our interpolation has difficulties in the latter areas.

Figure 3 shows the computation time and label assignment and confidence errors ε_{label} and ε_{conf} for the above experiment, evaluated for different DBM resolutions n ranging from 250 to 2000 pixels squared. Note that our maximal resolution $n = 2000$ significantly exceeds all reported DBM results in the literature, which only use images of several hundred pixels squared. For confidence sampling, we only show the cubic method as this yields better accuracy than the nearest neighbor and bilinear variants – see Fig. 2 and related text earlier. Figure 3, together with additional results including more DBM techniques using different (P, P^{-1}) combinations in the supplementary material, lead to the following observations.

Speed-wise, our binary and confidence-sampling methods show near-linear behavior in n as opposed to the quadratic behavior of ground-truth DBM. The confidence-split method is also roughly linear in n and more than double the speed of the ground-truth DBM. the confidence-split method in between the two. The acceleration factors are significant – the binary and confidence-sampling methods are over one order of magnitude faster than ground-truth DBMs. Label errors ε_{label} are larger for the confidence sampling method. Also, all our three methods do not increase ε_{label} with the resolution n, so larger images do not contain more errors. Similarly, confidence errors ε_{conf} are largely constant with n. However, we see that our

confidence sampling method fares much better in ε_{conf} than the binary split and confidence split methods.

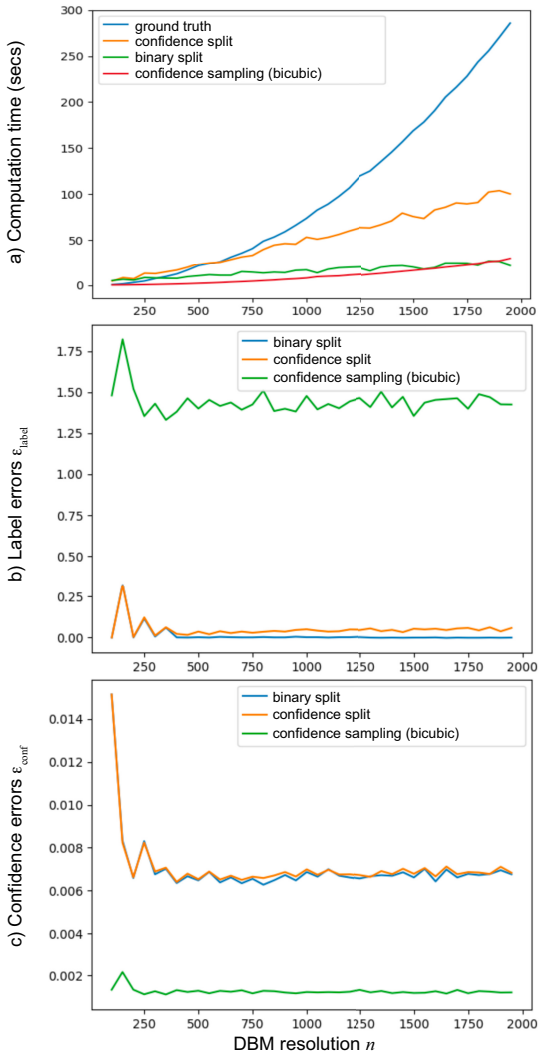

Figure 3: *Computation time (a), label errors ε_{label} (b), and confidence errors ε_{conf} (c) for the FastDBM methods, MNIST dataset.*

From the above, we conclude that the two most interesting methods for practical use are binary split and confidence sampling (bicubic variant). Binary split is the fastest method and also the one creating the fewest wrong labels in the DBM. Even though this method has the highest ε_{conf} errors, it is a good default method for users interested to get a very quick overview of their classification model's behavior, a task for which *exact* confidence values are less critical. Confidence sampling has roughly the same speed and has the lowest ε_{conf} errors. However, it creates label errors around the decision boundaries. As such, this method is more useful when one wishes to explore a classifier's behavior – including accurate confidence estimations – close to its training set, *i.e.*, farther away from decision boundaries.

Initial block size: The single free parameter of our method is the initial block resolution B, so how to set its value? A high B will limit the errors due to dense sampling of the image space I, but will be slow, since many blocks need to be evaluated via $M(P^{-1}(\cdot))$. A low B will be fast, but as noted in Sec. 3, DBM details smaller than $\frac{n}{B}$ pixels may be lost. Figure 4 shows an analysis of this trade-off regarding the size of B for our maximally considered resolution n. To generalize our findings, we use for this experiment additional combinations of P and P^{-1} to compute our ground-truth DBMs. These are autoencoders (AE, used for both P and P^{-1}); SSNP (used for both P and P^{-1}); and PCA, UMAP, and t-SNE (for P) with NNInv (for P^{-1}).

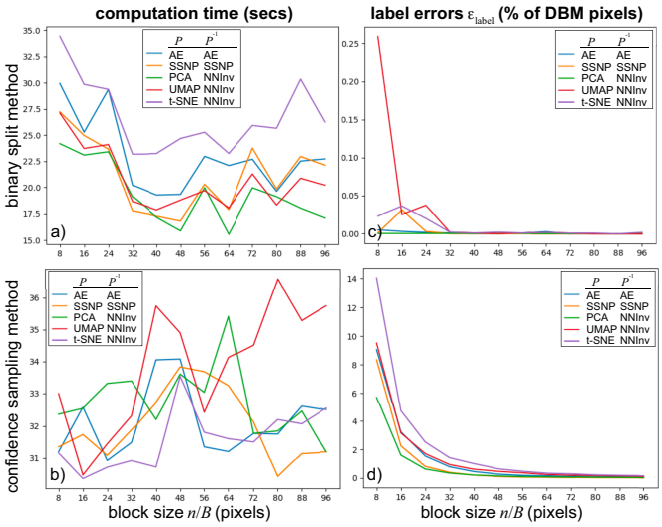

Figure 4: *FastDBM speed (a,c) and label errors (b,d) as function of block size n/B for DBMs computed by five (P, P^{-1}) methods, MNIST dataset.*

The binary split method is fastest for a block size of $B \simeq 50$ pixels (Fig. 4a). For confidence sampling, no exact B value yields a best value for computational time for all studied (P, P^{-1}) combinations, but differences are small (Fig. 4b). Label-error-wise, all block sizes above roughly 40 pixels yield (very) low errors for both methods (Fig. 4c,d). Hence, we conclude that a block size $n/B = 50$ pixels is a good general preset for FastDBM. We have verified this setting for several datasets including different sample and class counts.

Additional results: We evaluated the binary split method – which yields the best label approximation – with extra datasets, classifiers, and (P, P^{-1}) pairs used to compute the ground-truth DBM. Datasets include FashionMNIST [XRV17], HAR [AGO*12], and Iris [Fis88]. Classifiers included logistic regression (LR), support vector machines (SVM), k-nearest neighbors (kNN), decision trees (DT), random forests (RF), and the neural network (NN) we used for MNIST. Note that the accuracy of the trained classifiers used here is of no concern in this evaluation. Indeed, our aim is to see how well FastDBM approximates a ground-truth DBM, regardless of what this DBM shows. If the approximation is good, FastDBM can be used instead of the original DBM, being however much faster.

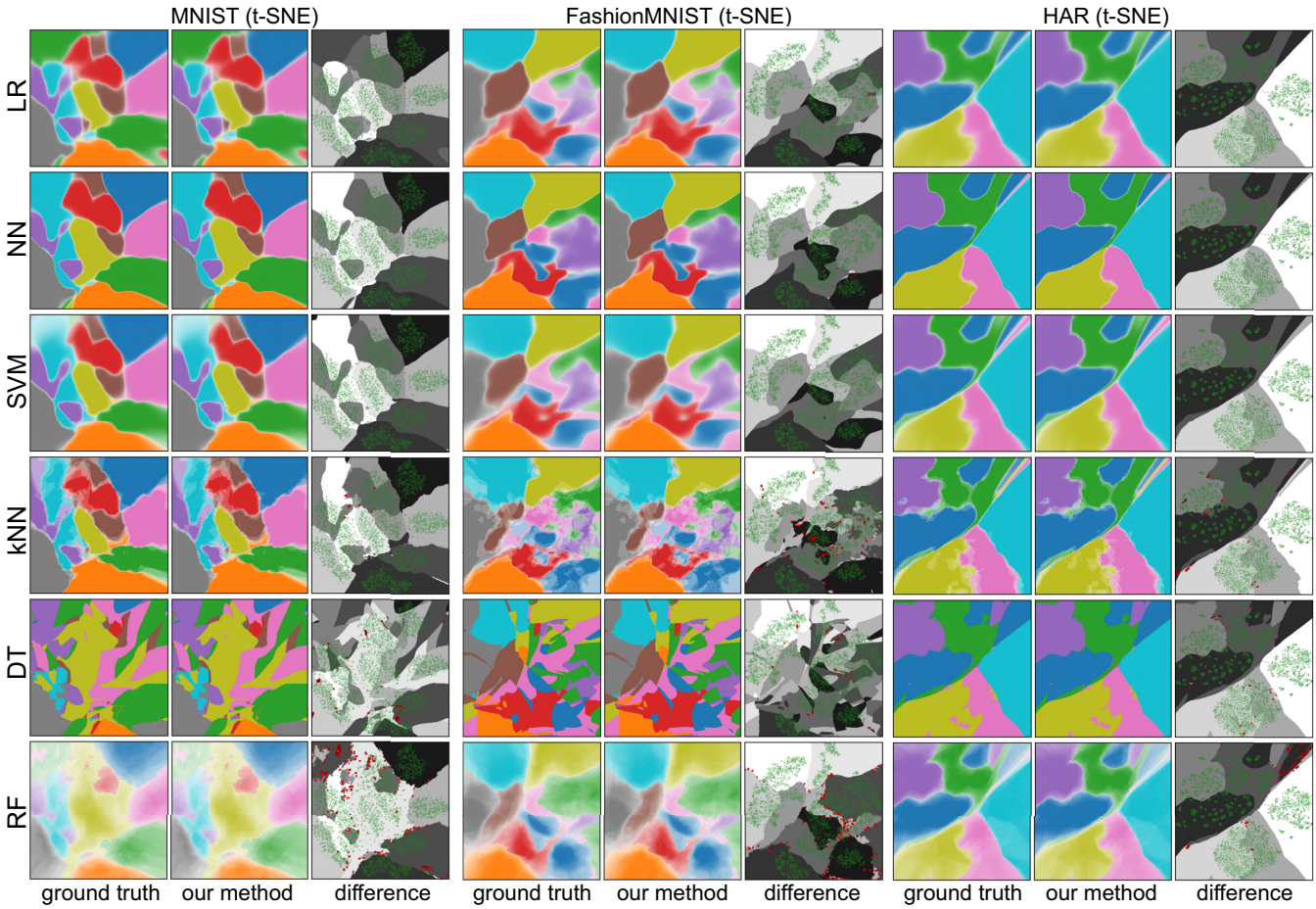

Figure 5: *Comparison between ground-truth DBM and our binary split method for three datasets and six classifiers, MNIST dataset.*

We created ground-truth DBMs at a resolution of $n = 400$ pixels squared by the (t-SNE, NNInv) and (UMAP, NNInv) pairs which are among the best combination for DBM computation (see again Sec. 2). Figure 5 shows, for the MNIST, FashionMNIST, and HAR datasets, the ground-truth DBMs, created with (t-SNE, NNInv); the DBMs created by our binary splitting; and the pixel-wise differences between the ground-truth and FastDBM encoded by red dots as in Fig. 2. In the difference images, we also show the training points used to construct the ground-truth DBM as green dots. Our method yields visually almost identical results in labeling as the ground-truth – there are only few red points in the difference images. This occurs consistently for quite different DBMs, *e.g.*, the smooth decision-zone ones created for LR, NN, SVM, and KNN, but also the far noisier one created for DT, and the overall low-confidence one created for RF. Additional results for all other tested combinations, present in the supplementary material, confirm this observation.

Implementation: Our FastDBM method is implemented in Python using the *sklearn* package and runs fully on the CPU. The full source code, including all datasets, ground-truth DBM computation methods, and experiments presented here, is publicly available for usage and replication purposes [GWT24].

5. Discussion and conclusion

We have presented FastDBM, a set of techniques that accelerate the computation of decision boundary maps (DBMs). FastDBM computes DBMs that are visually almost identical to ground-truth ones with a speed-up of over one order of magnitude. This allows the further deployment of such visualizations in interactive visual analytics workflows for classifier engineering. Our method variants offer trade-offs between the approximation of the visualized classifier labels and classifier confidence, and depend on a single free parameter for which we provide a good preset value. Importantly, our method can accelerate any current DBM computation technique, and can be applied to any trained classifier model, as it only requires access to the inverse projection function this technique uses, respectively to the black-box execution of the trained model.

In future work, we plan to further reduce the approximation errors of FastDBM while increasing its speed by more advanced sampling and interpolation schemes as well as GPU execution. In parallel, we are working on using FastDBM to extend visual active learning approaches [BTF18, BGTF21] such as to better support users in pseudo-labeling tasks for classifier engineering.

References

[AGO*12] ANGUITA D., GHIO A., ONETO L., PARRA X., REYES-ORTIZ J. L.: *Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine*. Ambient Assisted Living and Home Care. Springer, 2012, pp. 216–223. 4

[BGTF21] BENATO B. C., GOMES J. F., TELEA A. C., FALCÃO A. X.: Semi-automatic data annotation guided by feature space projection. *Pattern Recognition 109* (2021), 107612. 1, 5

[BTF18] BENATO B., TELEA A., FALCAO A.: Semi-supervised learning with interactive label propagation guided by feature space projections. In *Proc. SIBGRAPI* (2018). 1, 5

[dBD*12] DOS SANTOS AMORIM E. P., BRAZIL E. V., DANIELS J., JOIA P., NONATO L. G., SOUSA M. C.: iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2012), pp. 53–62. 2

[EHT21] ESPADOTO M., HIRATA N. S. T., TELEA A. C.: Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In *Proc. IVAPP* (2021), Hurter C., Purchase H. C., Braz J., Bouatouch K., (Eds.), SCITEPRESS, pp. 27–37. 2

[EMK*21] ESPADOTO M., MARTINS R. M., KERREN A., HIRATA N. S. T., TELEA A. C.: Toward a quantitative survey of dimension reduction techniques. *IEEE transactions on visualization and computer graphics 27*, 3 (Mar 01, 2021), 2153–2173. 1

[ERH*19] ESPADOTO M., RODRIGUES F. C. M., HIRATA N. S. T., JR R. H., TELEA A.: Deep learning inverse multidimensional projections. In *Proc. EuroVA* (2019). 2

[Fis88] FISHER R. A.: Iris plants database, 1988. UCI Machine Learning Repository. 4

[GWT24] GROSU C., WANG Y., TELEA A.: FastDBM implementation source code, 2024. https://pypi.org/project/decision-boundary-mapper. 5

[HS06] HINTON G. E., SALAKHUTDINOV R. R.: Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507. 2

[Jol02] JOLLIFFE I.: *Principal Component Analysis*. Springer, 2002. 1

[JPC*11] JOIA P., PAULOVICH F. V., COIMBRA D., CUMINATO J. A., NONATO L. G.: Local affine multidimensional projection. *IEEE transactions on visualization and computer graphics 17*, 12 (Dec 2011), 2563–2571. 2

[LeC10] LECUN Y.: MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist, 2010. 1, 3

[MH18] MCINNES L., HEALY J.: UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR abs/1802.03426* (2018). arXiv:1802.03426. 1

[NA18] NONATO L., AUPETIT M.: Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG 25*, 8 (2018), 2650–2673. 1

[OEH*23] OLIVEIRA A., ESPADOTO M., HIRATA R., HIRATA N., TELEA A.: Improving self-supervised dimensionality reduction: Exploring hyperparameters and pseudo-labeling strategies. In *Communications in Computer and Information Science* (2023), vol. 1691, Springer, pp. 135–161. 1, 2

[OEHJT22] OLIVEIRA A. A., ESPADOTO M., HIRATA JR R., TELEA A. C.: SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers. In *VISIGRAPP (3: IVAPP)* (2022), pp. 77–87. 1, 2

[REJT19] RODRIGUES F. C. M., ESPADOTO M., JR R. H., TELEA A.: Constructing and visualizing high-quality classifier decision boundary maps. *Information 10*, 9 (2019), 280–297. 1, 2

[RHT18] RODRIGUES F. C. M., HIRATA R., TELEA A. C.: Image-based visualization of classifier decision boundaries. In *31st Conference on Graphics, Patterns and Images (SIBGRAPI)* (2018), IEEE, pp. 353–360. 1, 2

[SHH20] SCHULZ A., HINDER F., HAMMER B.: DeepView: Visualizing classification boundaries of deep neural networks as scatter plots using discriminative dimensionality reduction. In *Proc. IJCAI* (2020), Bessiere C., (Ed.), pp. 2305–2311. 1, 2

[TMW24] TELEA A., MACHADO A., WANG Y.: Seeing is learning in high dimensions – how dimensionality reduction and machine learning can help each other. *SN Computer Science 5*, 279 (2024). 1

[vdMH08] VAN DER MAATEN L., HINTON G. E.: Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605. 1

[WMT23] WANG Y., MACHADO A., TELEA A.: Quantitative and qualitative comparison of decision map techniques for explaining classification models. *Algorithms 16*, 9 (2023), 438. 1, 2

[XRV17] XIAO H., RASUL K., VOLLGRAF R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR abs/1708.07747* (2017). arXiv:1708.07747. 4