SN Computer Science (2024) 5:279
https://doi.org/10.1007/s42979-024-02604-y

ORIGINAL RESEARCH

Seeing is Learning in High Dimensions: The Synergy Between Dimensionality Reduction and Machine Learning

Alexandru Telea[1] · **Alister Machado**[1] · **Yu Wang**[1]

Received: 4 September 2023 / Accepted: 30 December 2023

Abstract
High-dimensional data are a key study object for both machine learning (ML) and information visualization. On the visualization side, dimensionality reduction (DR) methods, also called projections, are the most suited techniques for visual exploration of large and high-dimensional datasets. On the ML side, high-dimensional data are generated and processed by classifiers and regressors, and these techniques increasingly require visualization for explanation and exploration. In this paper, we explore how both fields can help each other in achieving their respective aims. In more detail, we present both examples that show how DR can be used to understand and engineer better ML models (seeing helps learning) and also applications of DL for improving the computation of direct and inverse projections (learning helps seeing). We also identify existing limitations of DR methods used to assist ML and of ML techniques applied to improve DR. Based on the above, we propose several high-impact directions for future work that exploit the analyzed ML-DR synergy.

Keywords Multidimensional projections · Visual quality metrics · Explainable AI

Introduction

Machine learning (ML) has become one of the indispensable instruments in data-driven science and virtually any data-intensive application domain in our society. Recent advances in the field have made it possible to create models that predict or generate, with high accuracy, an increasing range of phenomena stemming from fields as diverse as image analysis and generation, natural language processing, medical diagnosis, and economical and societal trends. In parallel, developments in deep learning (DL), supported by the massive increase of power of modern GPU computing, have made the construction and deployment of powerful ML models increasingly scalable and affordable.

At a high level, and without loss of generality, ML models can be described as engines which process *high-dimensional data*—that is, collections of observations (samples) consisting of tens up to millions of individual measurements (dimensions) of a given phenomenon. Such data occur throughout the ML pipeline—it is present in the input of the models (e.g., images consisting of millions of pixels); in the internal working of such models (e.g., the so-called activations of neural units in the many intermediate layers of a DL model), and also in the models' output (e.g., the image created by generative AI techniques from given inputs). As such, it is not surprising that understanding high-dimensional data, and how it is transformed by ML models, is a key goal and challenge in ML.

In a separate field, exploring and understanding high-dimensional data are one of the top goals of information visualization (infovis) [1–3]. During the last decades, many techniques have been proposed to this end, including scatterplots and scatterplot matrices [4, 5], parallel coordinate plots [6], table lenses [7, 8], and glyphs [9]. However, most

A. Machado and Y. Wang contributed equally to this work.

This article is part of the topical collection "Recent Trends on Computer Vision, Imaging and Computer Graphics Theory and Applications" guest edited by Kadi Bouatouch, A. Augusto Sousa, Thomas Bashford-Rogers, Mounia Ziat and Helen Purchase.

✉ Alexandru Telea
a.c.telea@uu.nl

Alister Machado
a.machadodosreis@uu.nl

Yu Wang
y.wang6@uu.nl

[1] Department of Information and Computing Science, Utrecht University, Princetonplein 5, Utrecht 3584CC, The Netherlands

Published online: 21 February 2024

such techniques are fundamentally limited in the size of the datasets they can depict: They can show either datasets having many samples with few dimensions, few samples having many dimensions, but not both.

Dimensionality reduction (DR) techniques, also called multidimensional projections (MPs), are a family of visualization techniques that aim to solve the aforementioned visualization scalability issue in both the sample and dimension count. Simply put, given a high-dimensional dataset consisting of several samples, DR techniques create a low-dimensional (typically 2D or 3D) scatterplot in which close points correspond to similar samples in the input dataset. This allows users of DR visualizations to identify salient *patterns* in the dataset in the form of clusters of closely packed scatterplot points, clusters of points with different shapes or densities, or outlier points [10–12]. Tens, if not hundreds, of DR techniques have been designed to cater for the various functional and non-functional requirements inherent to the DR process, such as computational scalability, ability of treating data of various types and dimensionality, handling time-dependent data or data with missing values, ability of depicting new samples along existing ones (out-of-sample property), stability in the presence of noise, ability of capturing specific aspects present in the input dataset, and ease of use [12–14].

At a first glance, ML and DR are separate fields with different goals. ML is chiefly concerned with *learning* a model to predict the behavior of some phenomenon from existing samples thereof. In a more general setting, this has been extended to additional tasks such as data representation (autoencoders) or generative AI. For the purpose of our discussion next, we will mainly focus on prediction tasks, either in a classification or regression setting. DR aims at depicting, or *seeing*, the samples of such a phenomenon. We argue that these two goals are, however, strongly related, and actually advances in one field directly support requirements of the other field in both directions. Simply put, we argue that *seeing is learning*, in both directions of the implication, as outlined below:

- *Learning Needs Seeing* The ML field generates complex models, whose 'black-box' behavior is increasingly hard-to-understand by both their developers and users. Understanding such models is increasingly important for fine-tuning their behavior but also gaining trust in their deployment. Such understanding can be massively aided by seeing (visualizing) their structure and operation. Since ML models revolve around high-dimensional data, and DR techniques are ideally suited for depicting such data, DR techniques are a candidate of choice for visualizing them;
- *Seeing Needs Learning* Existing DR techniques are increasingly challenged by the already-mentioned sum of requirements they have to cope with. Few, if any, of such existing techniques can cope with all these requirements. In contrast, many ML techniques are designed upfront to handle such requirements, especially computational scalability, accuracy, stability, and out-of-sample ability. Given these, it makes sense to use ML techniques to learn the high-to-low-dimensional mapping and thereby assist the DR task.

In this paper, we extend our previous work [15] that explores the commonalities of ML and DR techniques to bring more evidence of existing, emerging, and potentially new interactions between these two fields, but also highlight important limitations of current ML-DR solutions. We proceed by introducing our two fields of interest—ML and DR—with an emphasis on their commonalities ("Background"). We next explore in "Seeing for Learning: DR Assists ML" how learning (ML) is supported by seeing (DR), especially in the creation of visual analytics (VA) solutions for explainable artificial intelligence (XAI). Subsequently, we study in "Learning for Seeing: ML Assists DR" the converse connection, that is, how seeing (DR) is supported by leaning (ML). We further outline in "Future Exploitations of the ML-DR Connection" new, emerging, connections between the two fields that point to promising future research directions in which the DR and ML fields can benefit from each other. Finally, "Conclusions" concludes the paper.

Background

In this section, we aim to provide a general introduction to ML and DR concepts, notations, and principles, with a focus on highlighting the commonalities between the two fields, which will be further explored in the remainder of the paper.

Notations Let $D = \{\mathbf{x}_i\}$ be a dataset of n-dimensional samples, also called observations or data points $\mathbf{x}_i, 1 \leq i \leq N$. A sample $\mathbf{x}_i = (x_i^1, \ldots, x_i^n)$ is a tuple of n components x_i^j, also called feature, variable, attribute, or dimension values. For exposition simplicity, we next consider that $x_i^j \in \mathbb{R}$ (other data domains are treated similarly for the purpose of our discussion). We denote by $Z \subset \mathbb{R}^n$ the spatial subset where samples of a given phenomenon are found. For instance, considering image data, only positive values (possibly bound by a maximum) can denote pixel intensities. Following this notation, D can be depicted as a table with N rows (one per sample) and n columns (one per dimension). Typically, these dimensions are considered to be *independent* variables, i.e., whose values are measured from the behavior of a given phenomenon over Z. Atop of these, D can have one or more dimensions (columns) of so-called *dependent* variables, also called labels or annotations. We next consider a single such dependent variable $y_i \in A$, where A is the domain

of definition of the labels, unless specified otherwise. We denote the annotated dataset D by $D_a = [D|\mathbf{y}]$.

Machine Learning Basics Given a so-called test set $D_T \subset D_a$, machine learning (ML) techniques aim to create a function (also called a model) $f : \mathbb{R}^n \rightarrow A$ which predicts the label values for most (ideally, all) samples in D_a, i.e., $f(\mathbf{x}_i) = y_i$, for $(x_i^1, \ldots, x_i^n, y_i) \in D_T$. Models f are built using a so-called training set $D_t \subset D_a$, $D_t \cap D_T = \emptyset$ so as to maximize the aforementioned results on the test set. ML models can be further split into *classifiers*, for which A is typically a categorical dataset; and *regressors*, for which A is typically a subset of $\mathbb{R}$. For regressors, f typically strives that $f(\mathbf{x}_i)$ is as close as possible to y_i, whereas for classifiers exact equality is aimed at.

Many methods exist to measure the performance of ML models. The most widespread such methods measure several so-called quality metrics on the training set (training performance) and, separately, on the unseen test set (testing performance). Common metrics include accuracy, precision, recall, F-score, and Cohen's kappa score. More advanced methods take into account hyperparameters that allow optimizing between precision and recall, e.g., the Receiver Operator Characteristic (ROC) curve and area underneath [16–18].

Dimensionality Reduction Basics A dimensionality reduction technique, or multidimensional projection P, is a function that maps every $\mathbf{x}_i \in D$ to a point $P(\mathbf{x}_i) \in \mathbb{R}^q$. For convenience, we next denote by $P(D) = \{P(\mathbf{x}_i)|\mathbf{x}_i \in D\}$ the projection of an entire dataset D. For visualization purposes, $q \in \{2, 3\}$, i.e., $P(D)$ is a 2D, respectively, 3D, scatterplot. At a high level, all projection techniques P aim to preserve the so-called *structure* of the dataset D, so that users can infer this structure by visualizing $P(D)$, following a well-known inverse mapping principle in data visualization [2]. Forms of such structure include, but are not limited to, clusters of similar samples; clusters having different sample densities; similarities between different samples; and outlier samples. Structure-preserving projections map (some) of these data properties to the corresponding properties of their generated scatterplots. Usually, projections do not use data annotations (even when these are available), but only independent variables—more on this aspect to be discussed further in "Deep Learning Projections".

Since data structure preservation entails several aspects, as outlined above, different so-called *quality metrics* have been devised to capture the abilities of a given P. A quality metric is a function $M(D, P(D)) \rightarrow \mathbb{R}^+$ that tells how well the scatterplot $P(D)$ captures a given aspect present in the dataset D. At a high level, such metrics can be grouped into (1) measuring *distance* preservation between pairs of samples, respectively, pairs of projection points, in $\mathbb{R}^n$ and $\mathbb{R}^q$, respectively, such as normalized stress and the Shepard diagram correlation [19]; and (2) measuring if neighborhoods (groups of close points) in D are mapped to neighborhoods in $P(D)$, such as trustworthiness and continuity [20], false and missing neighbors [21], and the Kullback–Leibler divergence [22]. The latter class is extended for projections of labeled data $P(D_a)$ by metrics such as neighborhood hit and class consistency [23, 24]. Detailed surveys of projection quality metrics are given in Refs. [10, 12, 14, 25].

Interaction Between ML and DR

As mentioned in "Introduction", our central statement is that *learning* (accomplished using ML) and *seeing* (visualization accomplished using DR) are intimately related to each other. This assertion, illustrated by Fig. 1, is explored next in detail.

How DR Helps ML and Conversely

Machine Learning Pipeline The central box (Fig. 1 blue) shows a technical view on the typical ML pipeline which maps an input real-valued dataset D into class labels or another real-valued signal by means of a classifier, respectively, regressor. Such ML pipelines can be next deployed to assist a wide variety of tasks. In our work here, we do not further detail these, but rather focus on how DR techniques can be used to assist the technical aspects of a typical, task-generic, ML pipeline; and conversely, how ML techniques can generically assist constructing better DR methods. As explained earlier in "Introduction", ML models operate on high-dimensional data. The green arrows atop this pipeline point to various visualization methods that use DR to depict such data. Using such visualizations, one can literally 'see' how the model learns. We further exemplify the use of such visualizations for ML tasks such as semi-automatic labeling ("Pseudolabeling for ML Training"), assessing classification difficulty ("Assessing and Improving Classifiers"), and assessing training of DL models ("Understanding DL Models").

Dimensionality Reduction Pipeline The bottom box (Fig. 1 yellow) shows how ML regressors can be used to create better DR projections of any high-dimensional data. Examples of this process include (self-)supervised projections and sensitivity analyses ("Deep Learning Projections"), inverse projections ("Deep Learning Inverse Projections"), and quality analysis for inverse projections ("Prospects of ML Assisting DR: Learning to See Better"). Once such DR methods have been created, they can be used for assisting ML engineering tasks, as shown by the red arrow in Fig. 1.

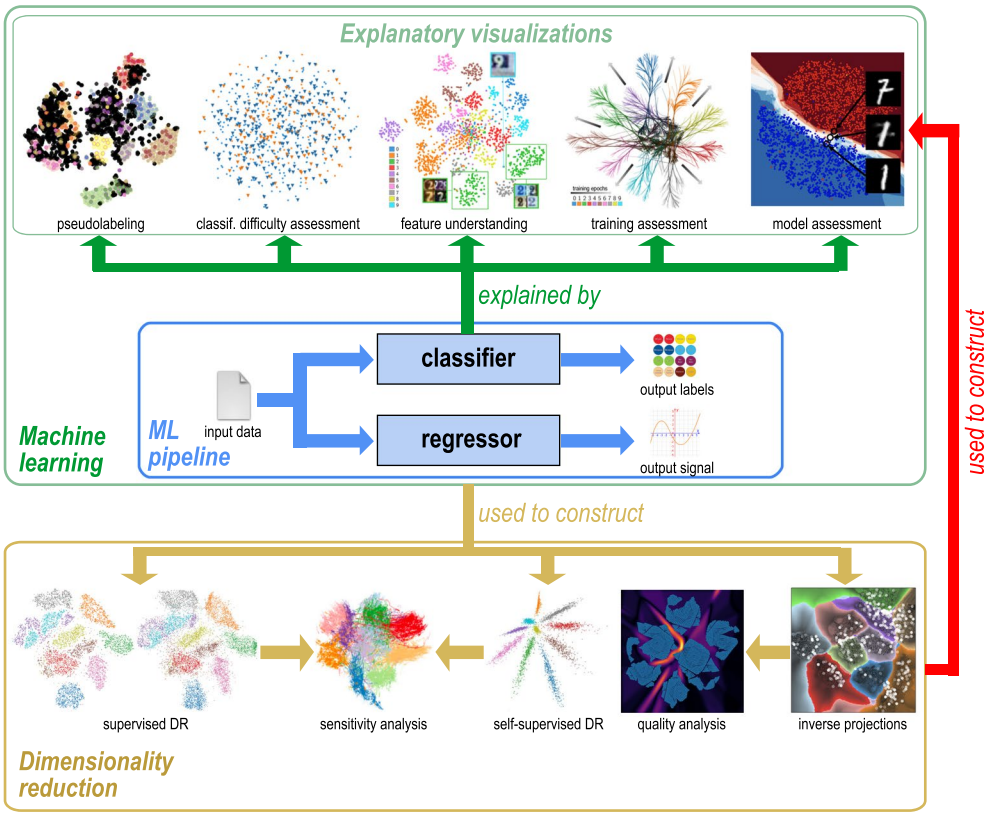

Fig. 1 Two-way interaction between machine learning (ML) and dimensionality reduction (DR) workflows. ML algorithms can be used to construct DR techniques. In turn, these can be used to construct explanatory visualizations for ML. See "Interaction Between ML and DR"

Common Aspects of DR and ML

"How DR Helps ML and Conversely" and Fig. 1 have outlined how DR can help ML and conversely. As such, it is not surprising that DL and ML share many common aspects. We detail next such commonalities, grouped in functional and non-functional ones, following a systems engineering perspective [26].

Functional Commonalities

Functional aspects describe how a system should operate. As already outlined, both ML models f and DR projection methods P are specialized cases of *inference* involving high-dimensional data. More specifically, P can be seen as a particular type of regressor from $\mathbb{R}^n$ to $\mathbb{R}^2$. Given this, we next use the notation X to jointly denote an ML model or DR algorithm, when distinguishing between the two is not important.

Non-functional Commonalities

Non-functional aspects describe how a system should behave in practice. Without claiming full coverage, we identify the following key aspects that both ML and DR techniques X strive to achieve in their operation. We also outline cases where these two classes of techniques achieve the respective requirements up to different degrees, thereby pointing to potential synergies where one technique family can be used to assist the other.

Genericity X should be readily applicable to any dataset D—that is, of any dimensionality, attribute types, and provenance application domain.

Accuracy X should deliver highly accurate results (inferences for ML; projection scatterplots for DR) as gauged by specific quality metrics in the two fields.

Scalability X should scale well computationally with the number of samples N and dimensions n—ideally, X should be linear in both N and n. In practice, X should be able to handle datasets with millions of samples and hundreds of dimensions on commodity hardware at interactive rates. This further on enables the use of X in *visual analytics* (VA) scenarios where the iterative and interactive exploration of complex hypotheses via data visualization is essential. We discuss this aspect further in "Seeing for Learning: DR Assists ML" and "Learning for Seeing: ML Assists DR".

Understandability For a technique to be useful and usable in practice, its operation should be easily understandable by

its intended users. This requirement takes different forms for ML and DR techniques. In general, ML techniques have an easy-to-understand *output*—they are designed to infer features having a clear meaning, e.g., the classes present in a dataset. However, due to their often black-box nature, the way in which they *operate* to do this is far less understandable, leading to challenges for their design, deployment, and acceptance (see "Understanding DL Models"). In contrast, most DR methods have a relatively clear way of operation. However, their output—a raw scatterplot, in the minimal case—is hard to interpret and requires additional explanatory mechanisms [21, 27–31].

Understandability is subtly related, but not identical, to the concept of *interpretability*. As mentioned above, we refer to understandability as the 'low level' ability of the intended users of a technique or tool to grasp how the tool works, at a basic level, so they are able to deploy it in practice. Interpretability operates at a higher conceptual level and refers to the ability of the users to reason about how the tool operates *internally* when executing its work. For ML models, for instance, linear regression is arguably more interpretable than deep neural networks due to its inherent linear model. Similarly, PCA's operation based on a global and linear data transformation is easier to understand than local and/or non-linear DR techniques such as t-SNE. In our further discussion, we mainly focus on the lower level of understandability.

Out of Sample (OOS) An operator X is said to be OOS if it can extrapolate its behavior beyond the data from which it was constructed. In ML, this usually means that the model f extrapolates from a training set D_t to an unseen test set D_T and beyond. By analogy, a projection P is OOS if, when extending some dataset D with additional samples $D' \nsubseteq D$, the projection $P(D \cup D')$ ideally keeps the samples of D at the locations they had in $P(D)$, i.e., $P(D \cup D') = P(D) \cup P(D')$. If P is OOS, this helps users to maintain their 'mental map' obtained by studying $P(D)$ when they further study $P(D \cup D')$. As most ML methods are OOS by design, they can be potentially used to design OOS projections (see next "Learning for Seeing: ML Assists DR").

Stability Small changes in the input dataset D should only lead to small changes in the output dataset $X(D)$. If not, spurious perturbations in D can massively affect the resulting inference $X(D)$ thereby rendering such results potentially unusable and/or misleading. Similarly, large-scale changes in D should arguably lead to correspondingly large changes in $X(D)$. Stability is related but not the same as OOS: An OOS algorithm is stable by definition but not all stable algorithms are OOS [14, 32]. Most ML methods are OOS by design, a property which is not shared by many projection techniques—therefore, opening up an interesting case for using ML for DR. We discuss stability and OOS in more detail in "Deep Learning Projections" and "Prospects of ML Assisting DR: Learning to See Better".

Ease of Use Visualization methods aim, by construction, to be easily usable by a wide range of users and with minimal or no programming effort. In contrast, building—and especially debugging and fine-tuning—an ML pipeline can be challenging for practitioners with limited training in ML. As such, this offers opportunities for using visualization (and DR in particular) to ease the task of ML practitioners.

Availability X should be readily available to practitioners in terms of documented open-source code. While sometimes neglected, this is a key requirement for ML and DR algorithms to become adopted and impactful in practice.

Table 1 compares how ML and DL techniques satisfy in general the above requirements. Scores are given on a 5-point Likert scale (++: best; −−: worst), according to our own experience. Besides genericity, where both ML and DR algorithms score equally well, all other requirements are met complementarity by the two algorithm families. This supports our earlier point that the two technique classes can support each other, if combined properly.

We next explore these commonalities and contrasts by first discussing how DR is used to help ML ("Seeing for Learning: DR Assists ML") and next how ML is used to create better DR algorithms ("Learning for Seeing: ML Assists DR").

Seeing for Learning: DR Assists ML

Many examples of visualization applications that assist ML workflows exist, most often coming in the form of complex multiple-view visual analytics systems [33–36]. An exhaustive presentation thereof is out of the scope of this paper. Rather, we focus in the following on selected use-cases where DR techniques have been used (with minimal

Table 1 Comparison of how ML and DR methods satisfy desirable requirements (**Gen**ericity, **Acc**uracy, **Scal**ability, **UnderOut** (understandability of output), **UnderAlg** (understandability of algorithm), **OOS**, **Stab**ility, **Ease** of use, **Avail**ability)

Methods	Gen	Acc	Scal	UnderOut	UnderAlg	OOS	Stab	Ease	Avail
ML	++	++	++	++	−−	++	++	0	++
DR	++	0/+	0	−−	+	−−	−	++	++

additions) to assist ML workflows: assessing and improving classifiers ("Assessing and Improving Classifiers"), pseudo-labeling for enriching training sets ("Pseudolabeling for ML Training"), exploring deep learning models ("Understanding DL Models"), and exploring classifier outputs via decision boundary maps ("Decision Boundary Maps").

Assessing and Improving Classifiers

One of the simplest, and still most frequently used, application of DR in ML is to project a labeled training or test set D_a with points $\mathbf{x}_i$ colored by their ground-truth labels y_i or labels $f(\mathbf{x}_i)$ inferred by some classifier f. The rationale for this use-case is straightforward: A projection places similar samples close to each other; a classifier labels similar samples similarly; hence, the visual structure of the projection helps several tasks:

- see how (and where) are *misclassified* samples distributed over the extent of a test set D_T (to next elicit what makes them hard to classify);
- see how well a training set D_t *covers* the data space (to, e.g., determine where extra training samples are needed);
- see how well a training set D_t is *separated* into different same-label sample groups (to next predict the classification ease).

The two first tasks are quite straightforward. In contrast, the last task is particularly interesting. The intuition that a projection $P(D)$ which is well separated into compact same-label groups indicates that D is easy to classify is quite old. Yet, a formal study of this correlation was only relatively recently presented [37]. In the respective work, the authors show that, given a range of classifiers, a dataset D whose projection $P(D)$ has well-separated classes (as measured by the neighborhood hit metric [23]) is far easier classifiable than a dataset whose projection shows intermixed points of different labels (low neighborhood hit). The projection $P(D)$ becomes a 'predictor' for the ease of classifying D, helping one to assess classification difficulty *before* actually embarking on the expensive cycle of classifier design-train-test.

Figure 2 illustrates the above usage of projections. Images (a) and (b) show the two-class Madelon dataset [38] ($n = 500$ dimensions, $|A| = 2$ classes) classified by KNN and Random Forests (RFC), respectively, with samples projected by t-SNE [22] and colored by class labels. The two projections show a very poor separation of the two classes, in line with the obtained low accuracies $AC = 54\%$ and $AC = 66\%$ (also visible by the misclassified samples, marked as triangles). Images (c) and (d) show the same dataset where extremely randomized trees [39] was used to select $n = 20$ dimensions. The projections show a much higher visual separation of the two classes, in line with the higher accuracies $AC = 88\%$ and $AC = 89\%$ obtained. Many other examples in [37] show that projections can predict classification accuracy quite well.

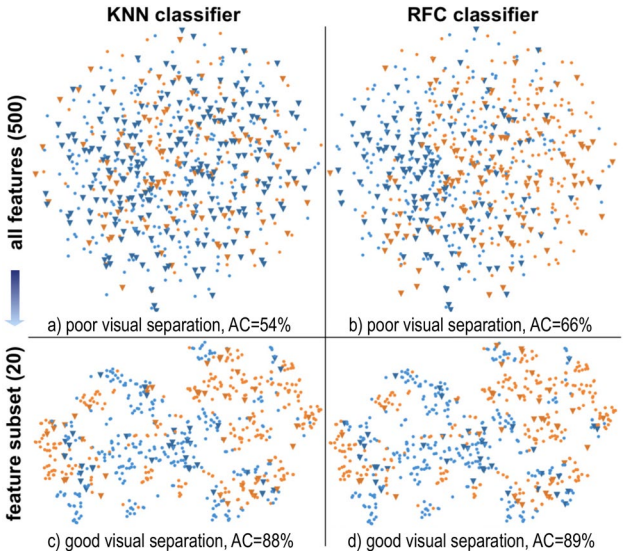

Fig. 2 Classification difficulty assessment via projections [37]

Pseudolabeling for ML Training

If projections are good predictors of classification accuracy, it means that their low-dimensional (2D) space captures well the similarity of the high-dimensional samples. This leads to the idea of using projections to create, rather than just explain, ML models. A first attempt was shown by Bernard et al. [40] in the context of an user evaluation that compared classical active learning with a user-supported procedure they dubbed Visual Interactive Labeling (VIL). Next after that, Benato et al. [41] proposed a very similar approach to VIL, called visual pseudolabeling, aimed to assist building a classifier from a training set having only very few labeled points: The entire training set, including unlabeled points, is projected and the user explores the projection to find unlabeled points tightly packed around labeled ones. Next, the user employs a tooltip to study the attributes of these points to confirm that they have the same class as the surrounded labeled one. If so, the user simply assigns that label to the unlabeled points. This workflow minimizes the user's labeling effort to quickly lead to sufficiently-large labeled sets for training the desired model. Interestingly, *automatic* label propagation in the embedded space using state-of-the-art methods [42, 43] leads to *poorer* results than user-driven labeling, which confirms the added value of the human-in-the-loop, and thus of the projections.

However, optimal results are obtained when humans and machine *cooperate*, rather than aim to replace, each other.

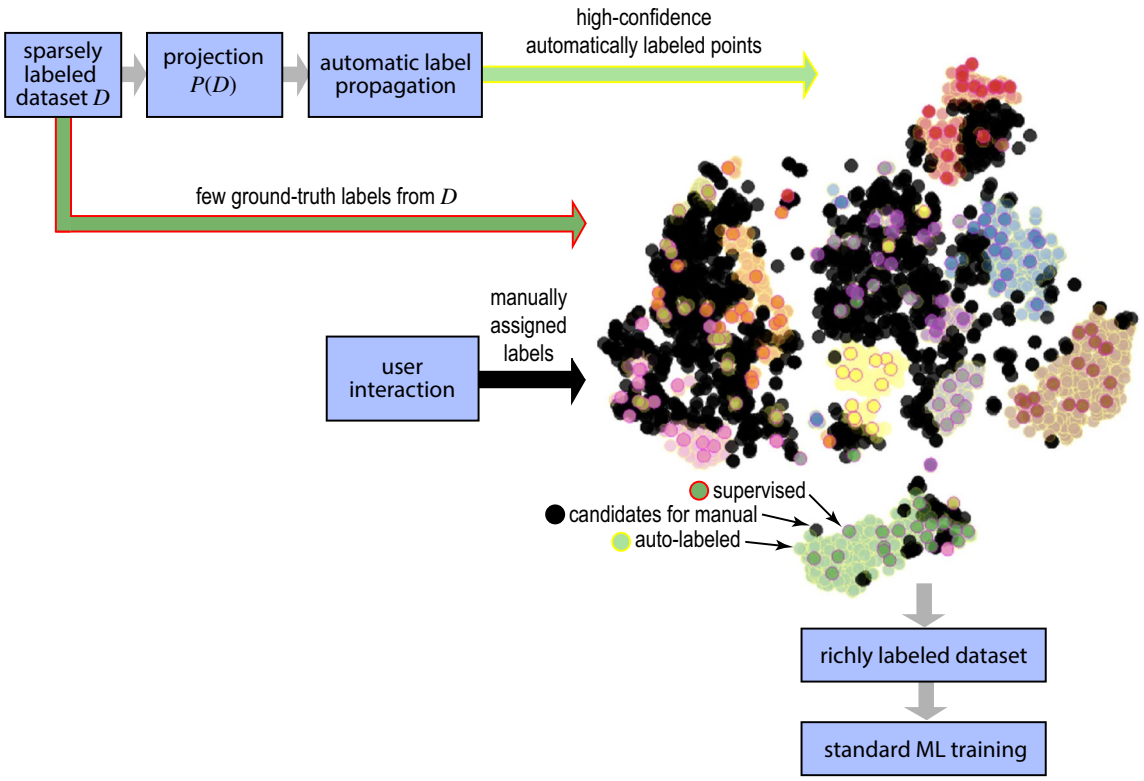

Fig. 3 Semi-automatic label propagation for constructing training sets. An algorithm propagates ground-truth labels from a small set of supervised samples towards unlabeled neighbor samples in the projection. When this algorithm is uncertain, samples are left for manual labeling [44]. See "Pseudolabeling for ML Training"

Benato et al. [44] refined the above workflow to (a) use automatic label propagation [42, 43] for the projection points where the propagation confidence is high; and (b) expose only the remaining unlabeled points to manual labeling (see Fig. 3). This way, many 'easy to label' points are handled automatically and the user's effort is channeled towards the hard cases, further reducing the manual labeling effort. This strategy also leads to increasing model accuracy and, again, surpassed confidence-based label propagation into the high-dimensional space.

Understanding DL Models

Deep learned (DL) models, with their millions of parameters, are among the hardest artifacts in ML to understand [45, 46]. Visualization has been listed early on as the technique of choice for explainable AI (XAI) for DL models [47]. A recent survey [33] outlines a wide spectrum of visual analytics techniques and tools used for DL engineering, grouped along how they support the tasks of training analysis (TA), architecture understanding (AU), and feature understanding (FU). Given the diversity of these tasks, the variety of the proposed visualization solutions—e.g. matrix plots [48], icicle plots [49], parallel coordinate plots [50], stacked barcharts, annotated networks [51], activation maps [52]—is not surprising.

Projections occupy a particular role among such visualizations due to their ability to compactly capture high-dimensional data—in the limit, a projection needs a single pixel to represent an n-dimensional point, for any value of n. As such, they are very suitable instruments to depict several aspects of a DL model. For example, in Fig. 4a, every point denotes a high-dimensional sample in D, in this case a digit image from the SVHN dataset [37]. The points, colored by their ground-truth classes, have as dimensions all activations of the last hidden layer—also called *learned features*—of a DL model trained to classify this dataset. We notice a good separation of same-class images (the projection contains compact same-color groups), which tells that the model's training went well. We also see, for each color (class), two distinct such groups. This tells that the model has learned to split images of the *same* digit into two subclasses. Upon inspection, illustrated by the tooltips in the figure, we see that the model has learned *by itself* to separate dark-on-bright-background digits from bright-on-dark background ones. Such findings would be hard to get without the projection-based visual exploration tool. Moreover, such findings can help fine-tuning the model to increase performance—in

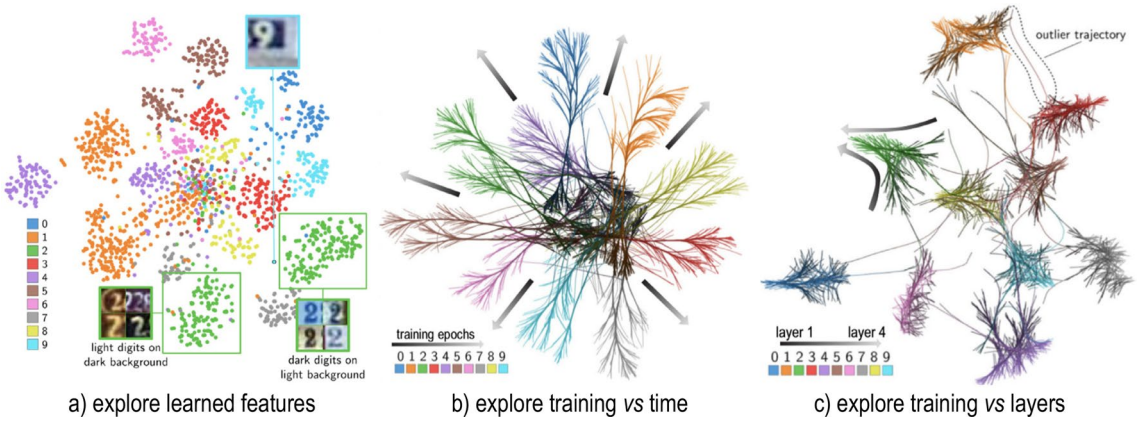

Fig. 4 Projections for understanding DL models. Exploring (**a**) activations of similar instances, (**b**) evolution of activations over training epochs, and (**c**) evolution of activations over network layers [55]. See "Understanding DL Models"

this case, eliminate the learning of the background-vs-foreground artificial separation for same-class digits.

Figure 4b explores a different DL aspect, namely how the model learns. For every epoch, a projection of all training-set samples is made, using as dimensions the samples' last hidden layer activations, similar to image (a). To maintain temporal coherence, i.e., have similar-value samples from the same or different epochs project to close locations, a *dynamic* projection algorithm, in this case dt-SNE [53], was used (see further "Prospects of ML Assisting DR: Learning to See Better"). Next, same-sample points from all epochs are connected by a trail. As the last step, trails are bundled in 2D [54] to reduce visual clutter. The resulting image (b) shows how the projection literally 'fans out' from a dark clump (in the middle of the image), which represents the similar activations of all samples in the first epoch, to separate clusters of same-label points (in the final epoch). This effectively summarizes the training success—the model has increasingly learned to separate the classes throughout its training. We also see some challenges of this model: The purple bundle (digit 4) is less well separated from the others, which indicates difficulties in classifying this digit.

Figure 4c shows a similarly constructed visualization but where the trails connect projections of test set image activations through all network's hidden *layers*. Bundles start fanned out but apart from each other, indicating that the trained model successfully separates the classes even after its first layer. Same-color trails in a bundle progressively fan in and also stay separated from trails in other bundles, indicating that, as we go down the model towards its further layers, class separation only becomes better, i.e., that the chosen network architecture is indeed good for the classification task at hand.

Decision Boundary Maps

A key aspect of ML classification models are points in their input data space $Z \subset \mathbb{R}^n$ where f changes output, i.e., the inferred class. Given the continuity assumption behind most ML models, such points are located on hypersurfaces (manifolds) embedded in Z, also called *decision surfaces* (see the light blue surfaces in Fig. 5a). These partition the Z space into compact regions where the classifier has the same output, also called *decision zones*.

As described so far, projections $P(D)$ depict a *discrete* set of samples D, optionally color-coded to show the behavior of a ML model f. For the dataset D represented by the green points in Fig. 5a, this would yield the red-points scatterplot in Fig. 5b. Such images, however, do not explicitly show where decision boundaries are—we know that they occur somewhere between the red dots, but not where precisely. Depicting such boundaries, along the color-coded training and/or test sets of f, significantly improves the understanding of how f actually behaves. This can help ML engineers to find where in the input space more training samples are needed to improve a classifier or, conversely, assess in which such areas would samples be misclassified.

Basic Idea of Decision Boundary Maps

Decision boundary maps (DBMs) propose such a visual representation for both decision zones and boundaries for any classifier. Intuitively put, DBMs map the entire space Z (as classified by f) to 2D rather than the discrete sample set D, as follows. Given a training and/or test set D, a direct projection P is used to create a 2D embedding thereof. After that, given an image space $I \subset \mathbb{R}^2$, a mapping $P^{-1} : I \rightarrow \mathbb{R}^n$ is constructed to reverse the effects of P. The mapping P^{-1} is then used to 'backproject' each pixel $\mathbf{y} \in I$ to a high-dimensional

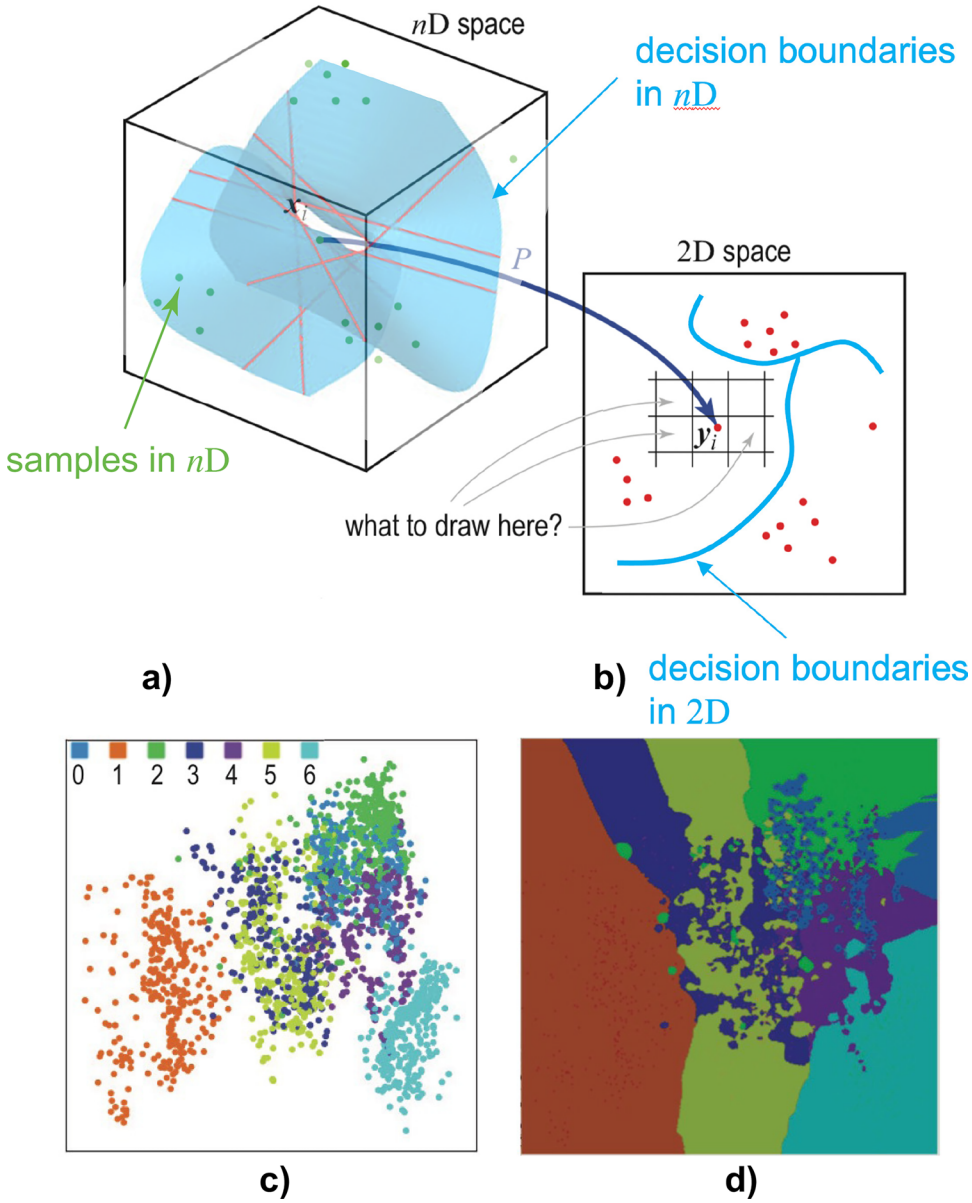

Fig. 5 Decision boundary maps. **a** A high-dimensional dataset with its decision boundary hypersurfaces. **b** Projecting the samples (green) and decision boundaries (light blue) of this dataset yields the red 2D points, respectively, light blue 2D curves. **c** Example of such a 2D projection with samples colored by the class inferred by a ML model. **d** The decision zones for this classifier are depicted in the 2D projection space as same-color areas. See "Decision Boundary Maps"

point $\mathbf{x} = P^{-1}(\mathbf{y})$, $\mathbf{x} \in Z$. Finally, each pixel $\mathbf{y}$ is colored by the label $f(\mathbf{x})$ assigned to it by the trained classifier to be explored. Same-color areas emerging in I indicate f's decision zones; pixels on the frontiers of these areas show f's decision boundaries. Figure 5d shows this for a KNN classifier trained to produce the test set depicted by the projection in Fig. 5 for a six-class problem.

The key to DBM construction is creating the mapping P^{-1} for a given direct projection P. In principle, any combination of P and P^{-1} can be used to construct a DBM for any given classifier by directly following the per-pixel procedure outlined above. However, earlier studies have shown that, for certain classification problems where one has ground-truth information about the expected outcomes—for example, in the sense of the shapes, sizes, and smoothness of the decision zones that a given classifier should create for that dataset—certain direct projections P and P^{-1} combinations work better [56, 57]. We discuss these aspects separately in "Deep Learning Inverse Projections".

Enhancements of Basic DBMs

DBMs can be further enhanced to encode, via brightness, the classifier's confidence at every 2D pixel (Fig. 6a, c) or

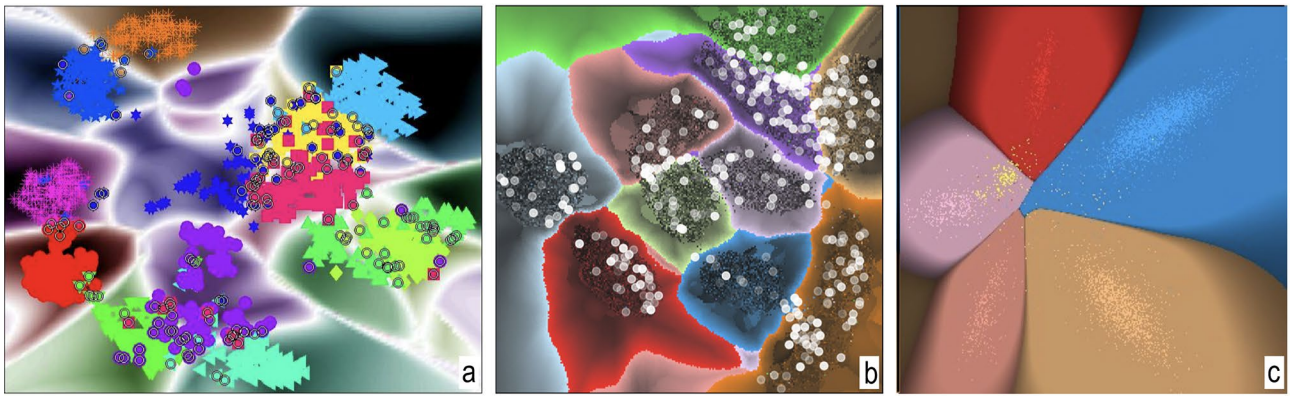

Fig. 6 Decision boundary maps for classifier analysis with luminance encoding classifier confidence (**a**, **c**) [57, 58], respectively, distance-to-decision-boundary (**b**) [56]. See "Decision Boundary Maps"

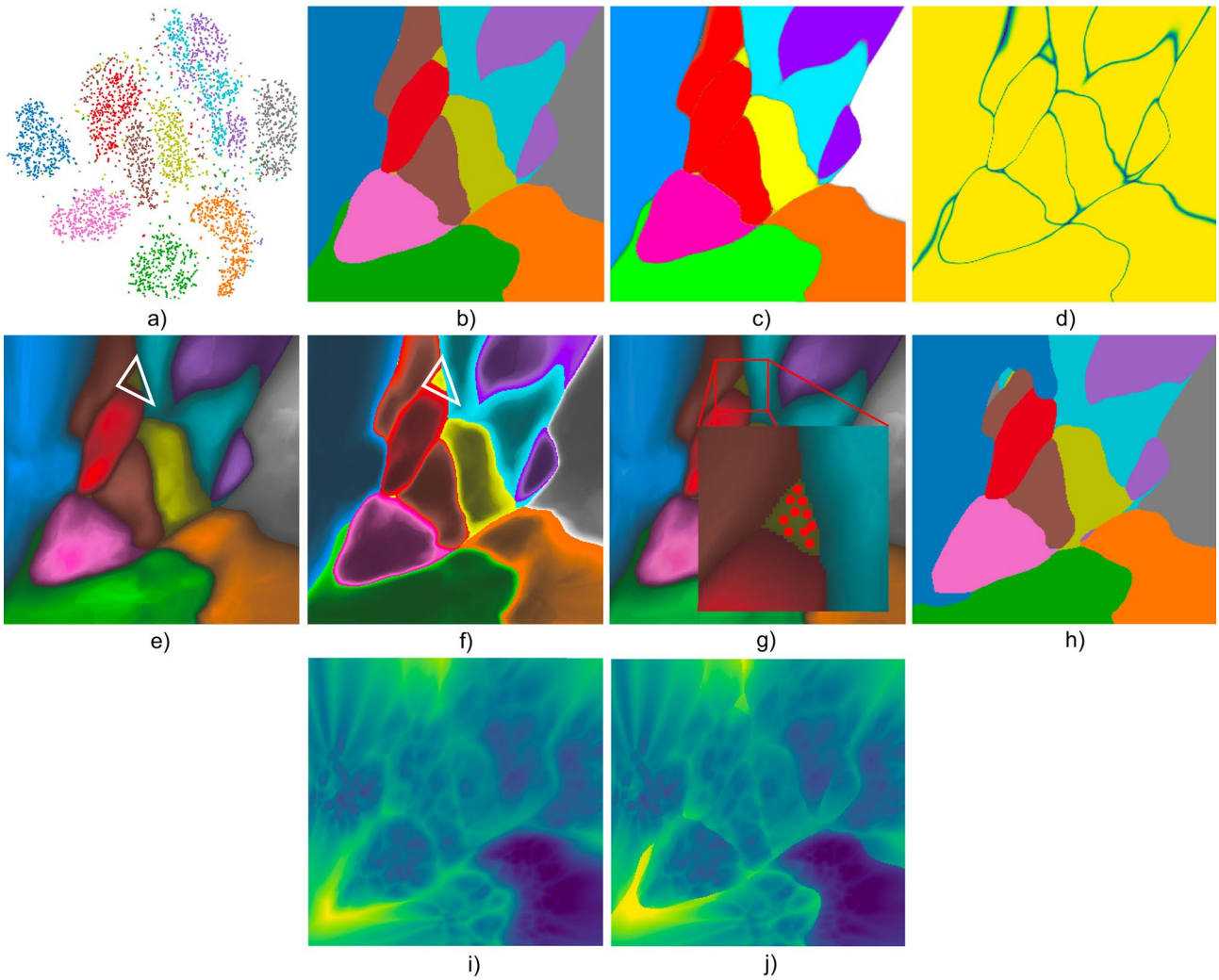

Fig. 7 Explanatory visualizations for interpreting DBMs. See "Decision Boundary Maps"

the actual distance, in Z, to the closest decision boundary (Fig. 6b). The appearing brightness gradients tell which areas in the projection space are more prone to misclassifications. Importantly, this does not require actual training or test samples to exist in these areas—rather, such samples are synthesized by P^{-1}.

Interpreting confidence or distance-enhanced DBMs is, however, not trivial, as illustrated next by the example in Fig. 7. Image (a) shows the MNIST digit dataset [59] ($n = 782$ dimensions, $|A| = 10$ classes) projected to 2D using t-SNE and classified by a neural network. Image (b) shows the DBMs for this problem. Image (c) enhances this by encoding the classifier confidence encoded into brightness (dark=lower confidence). For clarity, image (d) shows the confidence information separately (green=low confidence; yellow=high confidence). The images (c) and (d) convey the impression that the visualized classifier is highly, and equally, confident in all areas except very close to its decision boundaries.

Combining this information with the distance-to-closest-decision boundary reveals a different story. Image (e) shows this distance. In contrast to earlier techniques [56] (Fig. 6b) which use expensive iterative-search in the high-dimensional space to locate, for each pixel $\mathbf{y}$, the distance from $\mathbf{x} = P^{-1}(\mathbf{y})$ to its closest decision boundary, we use here a simpler, and much faster approach. For each such point $\mathbf{x}$, we synthesize its closest adversarial example $\mathbf{a} \in Z$ and approximate the sought distance as $\|\mathbf{x}-\mathbf{a}\|$ using DeepFool [60]. This is orders of magnitude faster than iterative search and allows generating the desired distances in sub-second time on a commodity PC. Examining image (e), we see that the distance-to-boundary evolves very differently for the different decision zones and has complex patterns even in a single such zone, indicating that certain points are far closer to decision boundaries than others. For example, the red decision zone, although appearing very close to its neighbors in the raw projection (Fig. 7a, is quite bright, telling that it is farther away from its surrounding decision boundaries, than the other, darker, zones. Image (f) shows the same information, but with inverse brightness mapping than in (e). This highlights zones close to decision boundaries, i.e., where the classifier may have trouble. We see, for example, a small yellow decision zone (marked by a white triangle). This zone, which is also disconnected from the other, larger, yellow decision zone (thus, for the same class), is very bright in image (e), indicating that it is very close to decision boundaries. This likely indicates potential model instabilities in this area.

To explore this hypothesis, we perform a simple experiment, as follows. We select ten pixels in the above-mentioned small yellow region, synthesize their corresponding data samples by P^{-1}, and add to them a wrong label—corresponding to the cyan color instead of the correct, yellow, label (see Fig. 7g, with the selected pixels marked in red). We next add these mislabeled points to the training set, retrain the model, and visualize its DBM. The result (Fig. 7h), shows how the small yellow region has become cyan, which is potentially not surprising given our newly added labels. More interesting, however, we see large changes in decision zones of *different* classes adjacent to the yellow region: the dark-blue zone grows significantly to cut away a portion of the brown zone. This shows that few data changes in a small decision zone, potentially flagged by our DBM visualization as unstable, change indeed the overall behavior of the classifier even outside this zone.

Annotating a DBM with the classifier confidence and/or distance to closest decision boundary does not, however, reveal all information that characterizes different decision zones. Indeed, one additional such information involves how close the DBM points are to the actual training points that the classifier was constructed from. We illustrate the added value of this information next. Images (c–f) in Fig. 7 show several large decision zones, e.g., the green and orange ones, which look quite similar from the perspective of confidence and distance to boundary—their inner pixels appear to be quite confident and far away from the surrounding decision boundaries. To gain more insight, we can visualize, for each pixel $\mathbf{y}$, the distance of its corresponding data sample to the closest training-set point, i.e.

$$d_{D_t}(\mathbf{y}) = \min_{\mathbf{x} \in D_t} \|P^{-1}(\mathbf{y}) - \mathbf{x}\|. \tag{1}$$

Figure 7i shows the distance d_{D_t} for our MNIST classifier, with dark blue indicating small distances and yellow large ones, respectively. We immediately see that all pixels of the orange decision zone are very close to the training-set, whereas pixels in all other zones appear much farther away. This indicates non-linear behavior of the DBM construction algorithm—the visible sizes of the decision zones in the DBM do not indicate actual sizes in the data space. Differently put, the orange decision zone is much closer 'wrapped around' training-set points than the other zones. This indicates that, all other aspects being equal, one should have more trust in the classifier behavior in the orange zone, as its depicted points are much closer to the training set that the classifier was built from.

Additionally, we see in image (i) a bright yellow band at the bottom of the corresponding pink decision zone. This tells that points around this decision boundary (between the pink and green zone) are quite far away from *any* training-set point. As such, even if the confidence of the classifier appears quite high in this area, apart from points very close to the decision boundary (see image (d)), the classifier extrapolates much farther away from its training data here, so, it is more prone to errors. Note that we would expect the confidence to drop as the data points become further apart

from the training set (intuitively, what the classifier learned from that training set is now 'stretched' to account for very different data), but this is not the case for this classifier. Our visualizations show that purely relying on classifier confidence is not sufficient for users to gain enough understanding of what the classifier does in specific situations and, hence, whether they trust (or not) the classifier in those situations.

Besides the above, we see, within each decision zone, a varying color pattern consisting of dark 'cells' separated by slightly brighter bands. These indicate how the respective sub-areas in a decision zone have been created by samples in the training-set—much like the visualization of a Voronoi diagram whose sites are the training-set samples.

Figure 7j shows a final variation of the explanatory visualizations for DBMs. Here, instead of depicting the distance of a map pixel to the closest training-set point, we show the distance to the closest training-set point of the same class as the pixel itself.

$$d_{D_t}^{\text{sameclass}}(\mathbf{y}) = \min_{\mathbf{x}\in D_t | f(\mathbf{x})=f(P^{-1}(\mathbf{y}))} \|P^{-1}(\mathbf{y}) - \mathbf{x}\|. \quad (2)$$

The distance $d_{D_t}^{\text{sameclass}}$ shows how far away samples that map to a decision zone are from training-set samples that led to the creation of that zone in the model f. We see that image (j) is quite similar to image (i). This is a positive finding, as it tells that pixels in a decision zone correspond to data points which are close to the training samples for that zone, which is indeed what a good DBM should show. In the same time, we see that the contrast between the orange and green zones, visible as dark blue, respectively, bright green in image (j), has increased. This tells that the decision boundary between the orange and green zones is far *closer* to the orange training samples than to the green ones—an insight which the basic confidence or distance-to-boundary maps do not reveal.

Coverage Study of DBMs

As explained earlier in this section, and outlined in Fig. 5, DBMs aim to project the decision zones of a classifier acting upon high-dimensional data into two-dimensional color patches. It is important to stress that this is a far more challenging problem than the one given to a 'plain' projection that acts upon a typical dataset D. Indeed, in virtually all cases, such datasets represent a carefully chosen *sampling* of a high-dimensional phenomenon, e.g., in terms of a training or test set. In contrast, a DBM aims, in theory, to project the *entire* high-dimensional space into 2D. A second difference regards how the data fed to projections, respectively, DBM methods, is created. For projections, the aforementioned sampling is typically carefully controlled by the creator of the respective training or test sets. For DBMs, the sampling of the high-dimensional space is, as explained earlier, done automatically based on the inverse projection P^{-1} being used. In summary, it is not evident which parts of the data space a DBM truly represents. Knowing this is crucial to further interpreting a DBM.

To gain more insight on this phenomenon, we executed a simple experiment. We generate a three-dimensional dataset having six concentrated blobs of samples, each blob having a separate class, following a Gaussian distribution. We next classify this dataset to obtain an 100% accuracy—which is expected, given the clear class separation. The choice of the used classifier is further not relevant given the simple nature of this dataset. Finally, we create DBMs for this classifier using the three available DBM techniques that we are aware of—the original decision map algorithm [56], the supervised version thereof called SDBM [57], and the DeepView technique [61] (more on these in "Deep Learning Inverse Projections"). Since the original dataset is three-dimensional, we can directly visualize it, and also the decision zones created by the three DBM techniques. To visualize these decision zones, we simply take all pixels corresponding to a decision zone in the 2D image, consider the quad mesh they form in that image via pixel adjacencies, backproject these pixels to 3D, and draw the respective quad mesh.

Figure 8a–c shows the obtained decision zones for the three aforementioned DBM techniques, color coded by their corresponding classes. Surprisingly, in all cases, these zones appear as residing on a *surface*, whereas, knowing the structure of the underlying dataset, they should actually be *volumetric* zones that partition the 3D space into six regions corresponding to the six class labels. In other words, the existing DBM techniques only choose a very specific two-dimensional surface-like subspace Z' of the entire data space to visualize. For clarity, note that this surface Z' is not the same as the actual decision *boundaries* of the studied classifier. These boundaries cannot be directly shown by the studied DBM visualizations. Rather, only their *intersections* with the artificial surface Z' are shown as the curves that separate different-color patches in Z'.

Image (d) illustrates this for the DBM map shown in image (a). Here, we sketch how the decision zones of three classes (yellow, blue, and purple) would arguably look like in 3D. As said, these are volumetric objects that enclose the training samples of their three respective classes. The border between the yellow and blue zones is the decision boundary B_{yb} between these classes, which is a *surface*. However, only the curve-like intersection $B_{yb} \cap Z'$ of this surface, indicated by the black curve in the figure, is shown by the DBM. Similarly, the border between the yellow and purple zones is the decision boundary B_{yp} between these two classes. However, in this case, the DBM does not show anything, since the surface Z' it constructs does not reach to that area of the 3D space, i.e., since $B_{yb} \cap Z' = \varnothing$ (dotted black curve in the

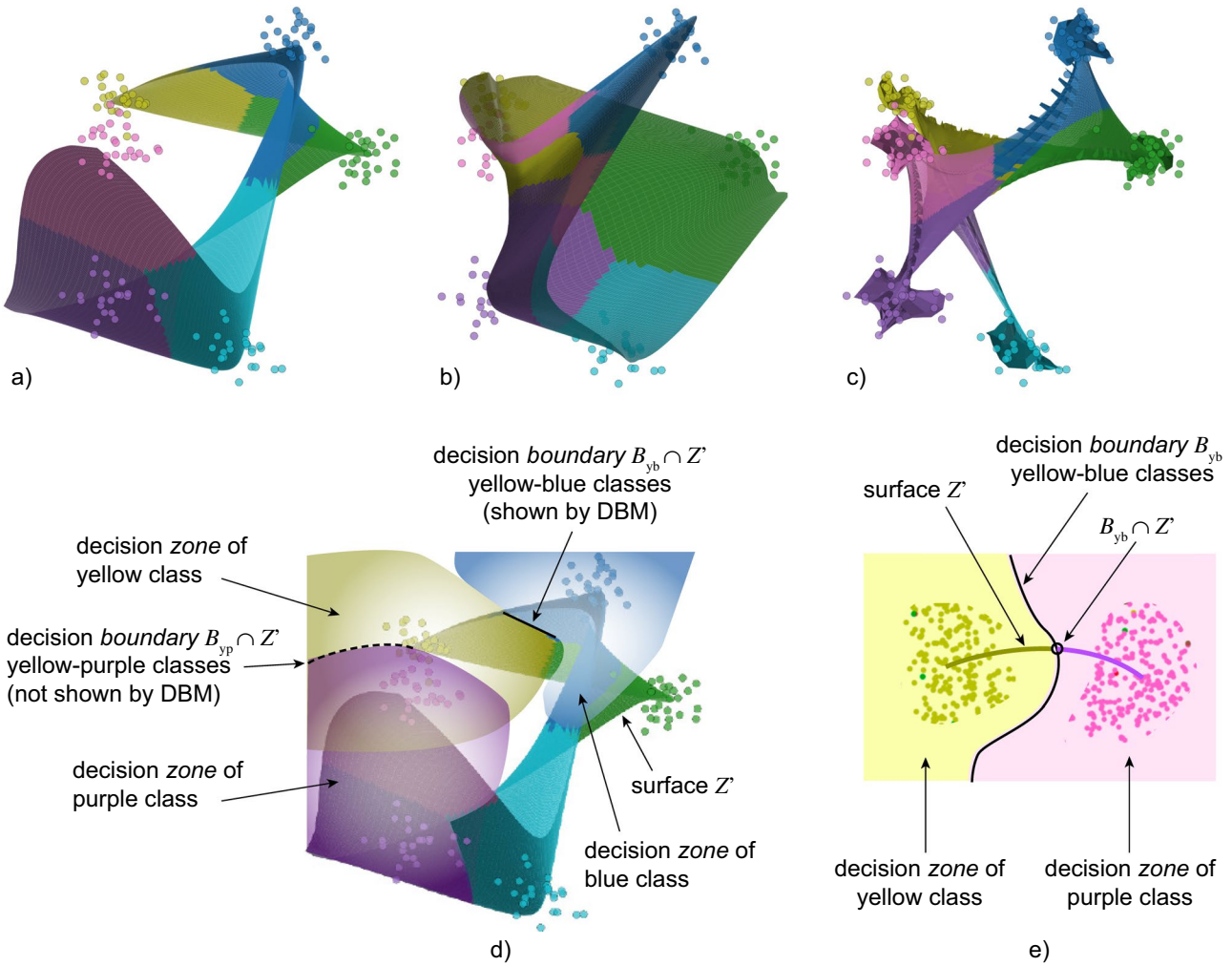

Fig. 8 Decision map methods construct and visualize their classifiers only over an implicitly constructed *surface* embedded in the high-dimensional space. **a** DBM [56]. **b** SDBM [57]; **c** DeepView [61]. **d** Limitations of decision map visualizations. **e** 2D sketch of **d**. See "Decision Boundary Maps"

figure). This is due to the finite size of the 2D image that these methods construct.

Figure 8e summarizes the above by a simpler, lower dimensional, 2D sketch (all quantities in Fig. 8d thus become one dimension lower). We see here the decision zones (2D yellow and pink surfaces), actual decision boundary B_{yb} (1D curve), surface Z' constructed by the DBM method (1D curve), and the part of the decision boundary that a DBM method can depict ($B_{yb} \cap Z'$, 0D point). As stated earlier, DBM methods only visualize a *subset* (0D point in this sketch) of the actual decision boundaries (1D curves in this sketch).

Summarizing our findings: (1) the way that a DBM method constructs the surface Z' will strongly influence which parts of the actual decision zones of a classifier will be offered for visualization; (2) only a part of the actual decision boundaries of a classifier are visualized by DBM methods; and (3) different DBM methods will produce different decision map visualizations for the same dataset and classifier—therefore, potentially leading to different interpretations. To our knowledge, none of these three findings have been outlined by earlier work on decision maps.

A final observation from Fig. 8 is that the above-mentioned surfaces Z' appear to *smoothly* connect the samples D used by the direct projection P that go into generating the inverse projection P^{-1}. Intuitively put, they look like minimal tension surfaces [62] that pass close to samples in D. This further suggests that, if the data to classify Z lives in high dimensions on a surface, and if D closely samples this surface, DBM methods will work predictably well and, also, deliver similar results—so, the choice of the DBM method to use is less relevant. Conversely, if D contains points that cannot be fit along such a surface—in other words, the sampled phenomenon has intrinsic

dimensionality higher than two—DBM methods may generate very different results depending on the actual dataset and DBM algorithm. This matches our earlier observation concerning the challenge of DBM methods to 'squeeze' a high-dimensional space into a 2D image. Designing more refined DBM algorithms that offer users a way to control which part of the high-dimensional space they sample to construct their classifier explanations is, thus, an important direction for future work (see next "Prospects of DR Assisting ML: Seeing to Learn Better").

Putting It All Together: Visual Analytics Workflow

At this point, we can further detail the general visual analytics workflow of ML assistance by DR techniques introduced in Fig. 1 (green arrows). Figure 9 does this by refining the workflow for using DR to assist ML proposed earlier by Rauber et al. (Fig. 1 in [37]) to include the DR-based techniques discussed above in this section—all which are novel in comparison with [37]), apart from the classification ease analysis (see next). This gives a practical guideline on how to use the presented visualization techniques in practical ML engineering. In the following, numbers in the text indicate steps in the workflow Fig. 9.

- The process starts by acquiring a dataset that one wants to further analyze, e.g., classify, using ML.
  - If not enough labeled samples are available (1), pseudolabeling ("Pseudolabeling for ML Training") can be used to create additional ones (2), else the process continues with the available data (3).
- From these data, features are typically extracted by various processing operations (4).
- Next, DR is used to construct a projection (5) from the data.
- The projection is visually studied to assess whether the data form sufficiently separated classes to suggest a feasible classification problem ("Assessing and Improving Classifiers").
  - If so (7), a ML architecture is chosen to design and train a model. Else, the workflow reverts to extracting better features (6).
- DR-enabled techniques are next used to assess whether the training performed well ("Understanding DL Models").
  - If training is found unsatisfactory (9), the model is further inspected ("Understanding DL Models") to find whether it has a poor design (10) or was fed by poor features (11).

    In the former case, the model goes to redesign stage; in the latter, different features are extracted.
  - If the model's training was positively assessed (12), the flow continues with standard ML evaluation (testing).

    Upon measuring satisfactory performance (13), the workflow ends with an operational model ready for use.
    If testing performance is found too low (14), decision map techniques ("Decision Boundary

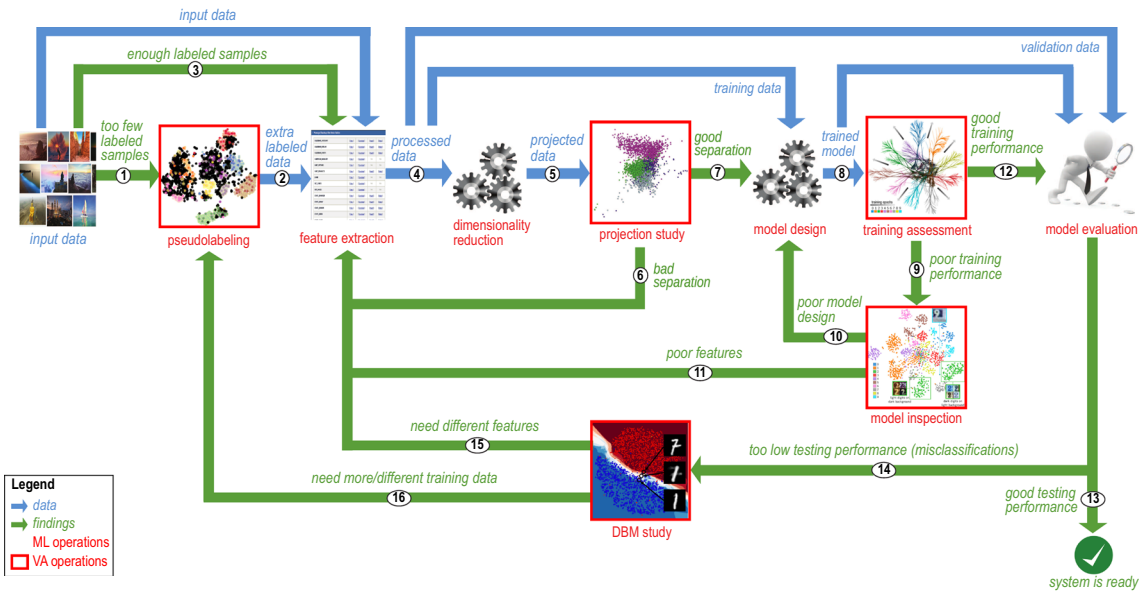

Fig. 9 Workflow of using DR techniques to assist ML (see Fig. 1, green arrows). Visual analytics (VA) operations enabled by DR are marked by red-outlined boxes

Maps") can be used to find out whether different features (15) or if more (or different) training data is needed (16). In both such cases, the workflow continues from the respective earlier steps, as indicated in the figure.

Learning for Seeing: ML Assists DR

Seeing for Learning: DR Assists ML has shown several examples of how DR visualizations help in several use-cases of ML engineering. In this section, we outline the opposite path, i.e., how ML techniques can be used to create DR visualizations so as to surpass limitations of existing DR algorithms. We discuss two classes of such methods for creating projections ("Deep Learning Projections"), respectively, inverse projections ("Deep Learning Inverse Projections").

Deep Learning Projections

Tens of DR techniques have been developed in the visualization community. However, choosing such a technique to apply in practice, for instance for the ML use-cases outlined in "Seeing for Learning: DR Assists ML", is challenging, as few comparisons of such techniques exist following all desirable requirements listed in "Common Aspects of DR and ML". A recent survey [14] addressed this question at scale for the first time by comparing 44 projection techniques P over 19 datasets D from the perspective of 6 quality metrics M, using grid-search to explore the hyperparameter spaces of the projection techniques. Equally important, all its results—datasets, projection techniques, quality metric implementations, study protocol—are automated and freely available, much like similar endeavors in the ML arena. Following the survey's results, four projection methods consistently scored high on quality for all datasets (UMAP [63], t-SNE [22], IDMAP [64], and PBC [65]), with several others close to them. However, none of the top-ranked surveyed techniques also met the OOS, computational scalability, and stability criteria. As such, we can conclude that better DR techniques are needed.

Following the analogy with ML regressors and given that such regressors meet the OOS, scalability, and stability criteria ("Interaction Between ML and DR"), it becomes interesting to consider ML for building better projection algorithms. Autoencoders [66] do precisely that and meet all requirements in "Interaction Between ML and DR" except quality—the resulting projections have in general poorer trustworthiness and continuity than state-of-the-art methods like UMAP and t-SNE. Figure 10 illustrates this: The well-known MNIST dataset, which is well separable into its 10 classes by many ML techniques, appears, wrongly, poorly separated when projected by autoencoders. Following [37] (see also "Assessing and Improving Classifiers", we can conclude that autoencoders are a poor solution for DR.

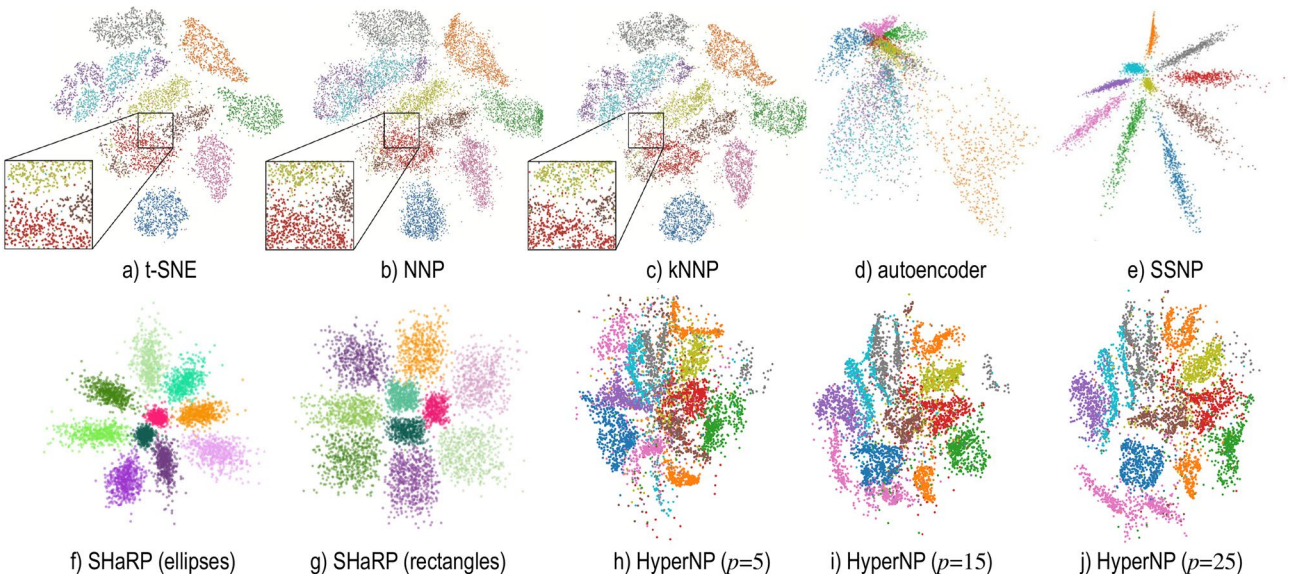

Fig. 10 Projection of MNIST dataset with (**a**) t-SNE [22] and with deep learning methods: **b** NNP [68], **c** kNNP [70], **d** autoencoders [66], **e** SSNP [71], **f**, **g** SHaRP [72] with elliptic, respectively, rectangular, cluster shapes, and **h**–**j** HyperNP [73] imitating t-SNE for three different perplexity values p

Basic Idea of Learning Projections

The idea of using deep learning to create an OOS projection is quite old. Pekalska et al. [67] proposed to do this to approximate Sammon's mapping in a way that could be extended to approximate also other DR techniques. More recently, Espadoto et al. [68] proposed Neural Network Projections (NNP), a supervised approach to learning DR: Given any dataset D and its projection $P(D)$ computed by the user's technique of choice P, a simple three-layer fully connected network is trained to learn to regress $P(D)$ when given D. Despite its simplicity, NNP can learn to imitate any projection technique P for any dataset D surprisingly well. While NNP's quality is typically slightly lower than state-of-the-art projections like t-SNE and UMAP, it is a *parametric* method, stable as proven by sensitivity analysis studies [69], OOS, linear in the sample count N and dimensionality n (in practice, thousands of times faster than t-SNE), and very simple to implement.

OOS and Sensitivity Analysis

As explained earlier, the OOS and sensitivity (stability to small changes of the input) are related, but not identical, desirable properties. We illustrate both properties for NNP next, noting also that all other similar deep-learned projection algorithms (kNNP, SNNP, SHaRP, autoencoders) share by construction the same properties, since they use very similar neural network architectures.

Figure 11 (top two rows) illustrates NNP's out-of-sample ability. The top row shows t-SNE projections of the MNIST dataset for an increasing number of samples (from 2K to 100K). As visible, the projection continuously changes, making it hard for users to maintain their mental map of the studied data. The row below shows NNP trained to mimic t-SNE. We see that the shape of the projection and location of its ten clusters (one per class) stays the same as more samples are added. A drawback of this is that the cluster separation is lower than for the t-SNE projection as more samples are added. This is expected since NNP was trained

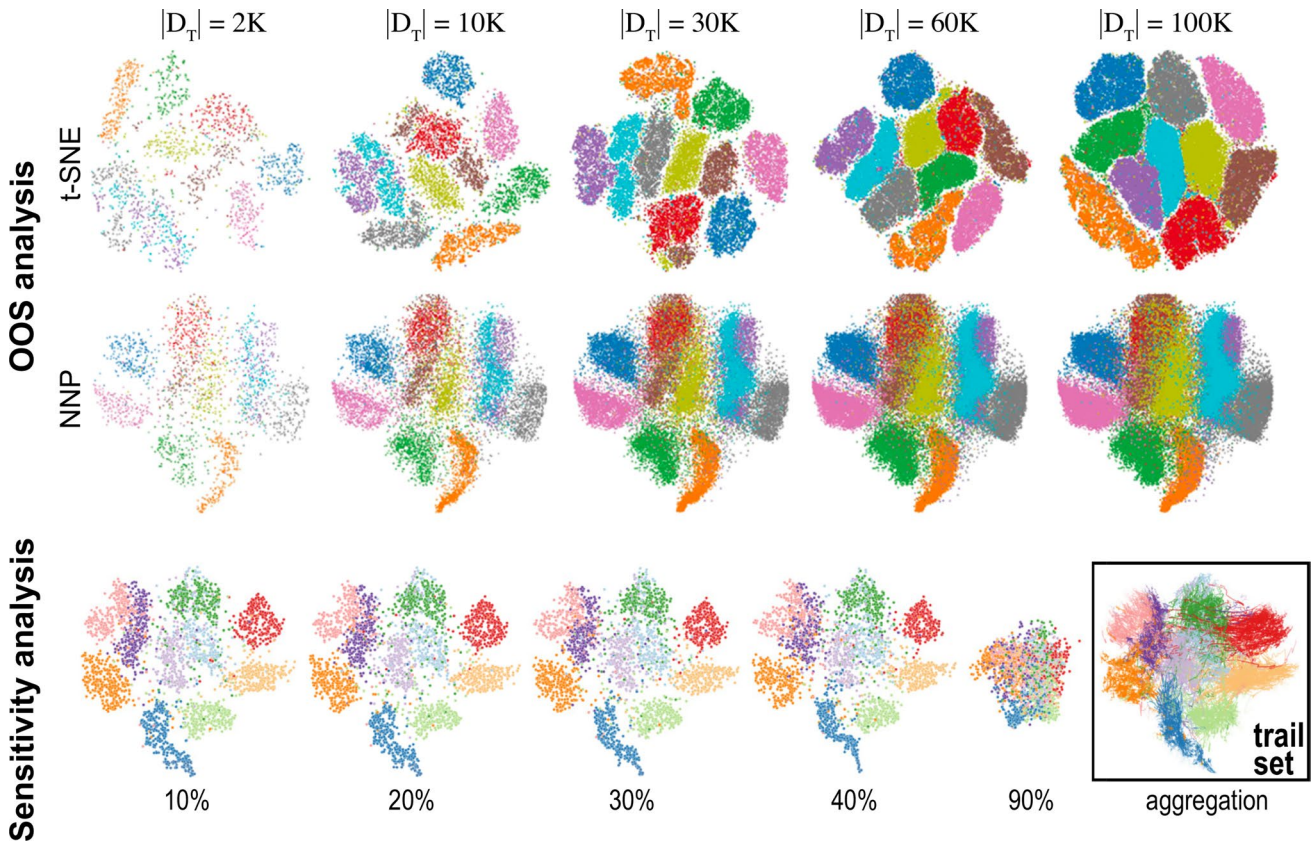

Fig. 11 Top two rows: NNP out-of-sample (OOS) analysis. The top row shows projections of the MNIST dataset using t-SNE for increasing numbers of samples in the test set D_T. The projection does not maintain stability. The second row shows how NNP maintains stability as new samples are added to the test set. Bottom row: NNP sensitivity analysis when removing between 10% and 90% of the dataset's dimensions. NNP can robustly depict the data structure even when a large part of the input information is missing. See "Deep Learning Projections"

only on the initial set of 2K samples, i.e., it did not have a chance to see the additional ones.

Figure 11 (bottom row) illustrates NNP's stability. An NNP model is trained to project the MNIST dataset, after which is asked to project MNIST images where an increasingly larger number of dimensions (pixel values) have been cancelled, i.e., set to zero. Surprisingly, NNP can capture the cluster structure of the data (10 classes for the 10 digits) up to 40% cancelled dimensions. The aggregated image shows the 'movement' of the points in the NNP projection as increasingly more dimensions get dropped. Similar insights are obtained for other input dataset perturbations such as sample jitter, translation, and scaling. At a higher level, we see sensitivity analysis as a very powerful, yet underexplored, technique—well known in the ML repertoire—to assess the quality of DR projections.

Refinements of NNP

Subsequent refinements of NNP aim to keep the attractive aspects of the method (speed, OOS, genericity, stability, simplicity) while increasing its *quality* and controlling the visual *appearance* of the resulting projections (see further Fig. 10). kNNP [70] enhances the projection quality, measured following the metrics introduced in "Background", by learning to project sets of neighbor samples rather than individual samples. SSNP [71] works in a self-supervised way, similar to autoencoders, thus dispenses of the need of a training projection. The self-supervised information comes either in the form of ground-truth labels (when available) or pseudolabels computed by clustering the input data. Since based on an autoencoder structure, SSNP can also create inverse projections (see next "Deep Learning Inverse Projections"). SDR-NNP [74] increases NNP's ability to separate clusters of different observations by pre-sharpening the input training set D_t via mean shift [75]. SHaRP [72] refines SSNP to allow one to control the shapes of clusters of similar observations to match desired templates such as ellipses, rectangles, or triangles. Finally, HyperNP [73] extends NNP by learning the behavior of a projection technique P for all its hyperparameter values, thereby allowing users to explore this parameter space at interactive rates. All the above results prove that DL is a serious contender for generating projections that comply with all requirements set forth by practice.

Deep Learning Inverse Projections

Following the success of DL for constructing projections P outlined above, it becomes immediately interesting to consider their use for the complementary problem of computing *inverse* projections P^{-1}. Introduced in "Decision Boundary Maps" for constructing DBMs, inverse projections have additional uses, e.g., generating synthetic samples for data augmentation scenarios [76] and hypothesizing the unexplored regions of a sampled data space for, e.g., shape or image morphing applications [77].

Definition Formally put, given a direct projection function $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$, with $q \ll n$ typically, the inverse projection is just the inverse of that function, i.e., a function $P^{-1} : \mathbb{R}^q \rightarrow \mathbb{R}^n$ so that $P^{-1}(P(\mathbf{x})) = \mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^n$. Unfortunately, computing such an exact inverse function is not possible for virtually any of the existing projection algorithms P for one or several of the following reasons:

- *Non-injectivity* Typical projection functions P are not injective—that is, they can map different points in $\mathbb{R}^n$ to the same location in $\mathbb{R}^q$. This is a direct, and probably unavoidable, consequence of the fact that $q \ll n$;
- *Non-parametric nature* While we are talking about P as being a *function* between two spaces ($\mathbb{R}^n$ and $\mathbb{R}^q$), many projection algorithms do not work in this fashion. Rather, they map a given *sampling*, or dataset, $D \subset \mathbb{R}^n$ to another dataset $P(D) \subset \mathbb{R}^q$. When the dataset D changes, the mapping can change as well—that is, the same point $\mathbf{x} \in \mathbb{R}^n$ can be mapped to different locations in $\mathbb{R}^q$, depending on which other points it comes along with in D. This is precisely the lack of OOS ability discussed in "Deep Learning Projections" and illustrated, for t-SNE, in Fig. 11. Only a (small) subset of projection techniques are parametric, i.e., comply with the functional definition given above;
- *Inverse problem* Even for cases when an algorithmic functional definition of P is available and P is injective, computing its inverse can be quite challenging since inverse problems do not always have guaranteed unique and/or stable solutions.

To address the above three problems, the practical approach to computing P^{-1} is by various forms of approximation, as follows:

- *Non-injectivity* This aspect is usually neglected by practical algorithms that compute P^{-1}. That is, if two (or more) points $\mathbf{x}_i \in \mathbb{R}^n$ map to the same location $\mathbf{y} \in \mathbb{R}^q$, $P^{-1}(\mathbf{y})$ will yield a single value in $\mathbb{R}^n$;
- *Non-parametric Nature* Since most projections P are of this nature, inverse projections are usually defined in terms of a given dataset $D \subset \mathbb{R}^n$. For such a dataset, an inverse projection is a function that aims to yield $P^{-1}(P(\mathbf{x})) \approx \mathbf{x}$ for all $\mathbf{x} \in D$. However, inverse projections *need* to be parametric themselves, i.e., applicable to any other values $\mathbf{y} \in \mathbf{R}^q$ apart from $P(D)$, otherwise they would not be useful for the tasks mentioned at the

beginning of this section. For such 'unseen' points $\mathbf{y}$, P^{-1} is usually defined to work in a smooth manner—the closest $\mathbf{y}$ is to a known direct mapping $P(\mathbf{x})$, the closest should $P^{-1}(\mathbf{y})$ be to $\mathbf{x}$.

- *Inverse Problem* Computing P^{-1} is usually done by minimizing suitably-chosen (convex) error functions that model the goal $P^{-1}(P(\mathbf{x})) \approx \mathbf{x}$ for all $\mathbf{x} \in D$ mentioned above. This simplifies and accelerates the computation problem using existing numerical optimization methods.

Early Inverse-Projection Methods

Likely one of the earliest methods for computing inverse projections is by training an autoencoder to jointly perform P and P^{-1} [66]. While this method is simple to implement and fast to compute, it does not allow inverting any user-chosen direct projection P. Moreover, projections created by autoencoders are of lower quality than other state-of-the-art techniques (see Fig. 10 and related text and also the evaluation in [14].

Mamani et al. [78] presented a method that uses inverse projections to transform the high-dimensional data space based on manipulations of the two-dimensional, projection, space. This allows users to execute several potentially complex operations that affect the invisible high-dimensional data based on a simple interface that allows direct manipulation of 2D projection points. However, the proposed method does not directly allow inversely projecting new points—that is, 2D points which are not the direct projection of existing data points.

Amorim et al. [77, 79] presented iLAMP, a method for inverting the LAMP [19] projection technique using distance-based interpolation functions to mimic the relative position of a point in 2D *vs* its neighbors in $P(D)$ onto the high-dimensional data space. iLAMP gives good results but is relatively slow and can only handle the LAMP projection.

Schulz et al. [58, 61] proposed DeepView to compute inverse projections using UMAP [63] as direct projection P. In contrast to UMAP, however, similarities of points are computed by combining their high-dimensional attributes with the outcome of a classifier f (similar to SSNP). The inverse projection is then computed also by UMAP from the projection of the given dataset $P(D)$ and then extrapolated to the entire 2D space by minimizing the Kullback–Leibler divergence (similar to t-SNE). DeepView yields quite smooth results (see its application for DBM construction in Fig. 6a) but is over an order of magnitude slower than the other P^{-1} techniques described here.

Deep Learning Inverse Projections

Following the success of NNP for direct projections ("Deep Learning Projections"), Espadoto et al. [80] computed inverse projections by simply 'switching' the input and output of NNP, i.e., given a dataset D that projects to a 2D scatterplot by some technique P, train a regressor to output D when given $P(D)$. This technique, called NNInv inherits all the desirable properties of NNP ("Deep Learning Projections") and also produces higher-quality inverse projections than autoencoders and iLAMP. The usage of NNInv is illustrated in the DBM construction in Fig. 6b. Further variations of this design include SSNP [71] (used to construct the DBM in Fig. 6c and SHaRP [72]. Both techniques use an autoencoder basis so produce both a direct and inverse projection (see also "Deep Learning Projections"). However, their quality is higher than plain autoencoders and also than plain NNInv given their (self-)supervised operation based on class (pseudo)labels. Figure 12 illustrates this by comparing NNinv (first row) with SHaRP (second and third rows) for the construction of a decision map for a simple k-nearest neighbors (KNN) classifier ($k = 21$) for four datasets of varying dimensionalities n. This is a novel insight as SHaRP has, so far, not been gauged on its performance for computing decision maps. We see that, similarly to SSNP (shown in Fig. 6c), SHaRP produces decision zones with smoother boundaries than NNInv, which are closer to the known ground-truth smooth boundaries (hyperplanes) that a KNN classifier should have.

Applications of Inverse Projections

NNInv was further explored in detail for visual analytics scenarios involving dynamic imputation and exploring ensemble classifiers [81]. Figure 13 shows the latter use-case: In the image, each pixel is backprojected and ran through a set of nine classifiers, trained to separate classes 1 and 7 from the MNIST dataset. The pixel is then colored to indicate the classifiers' agreement. Deep blue, respectively, red, zones show areas where all classifiers agree with class 1, respectively, 7. Brighter areas indicate regions of high classifier disagreement—which are thus highly difficult to decide upon and are prime candidates for ML engineering, regardless of the used classifier.

Future Exploitations of the ML-DR Connection

Reflecting upon the current achievements of using ML for DR and conversely, we see a bright future ahead for research where the two directions assist each other. We illustrate this with a few selected, non-exhaustive, examples of such potential ML-DR synergies.

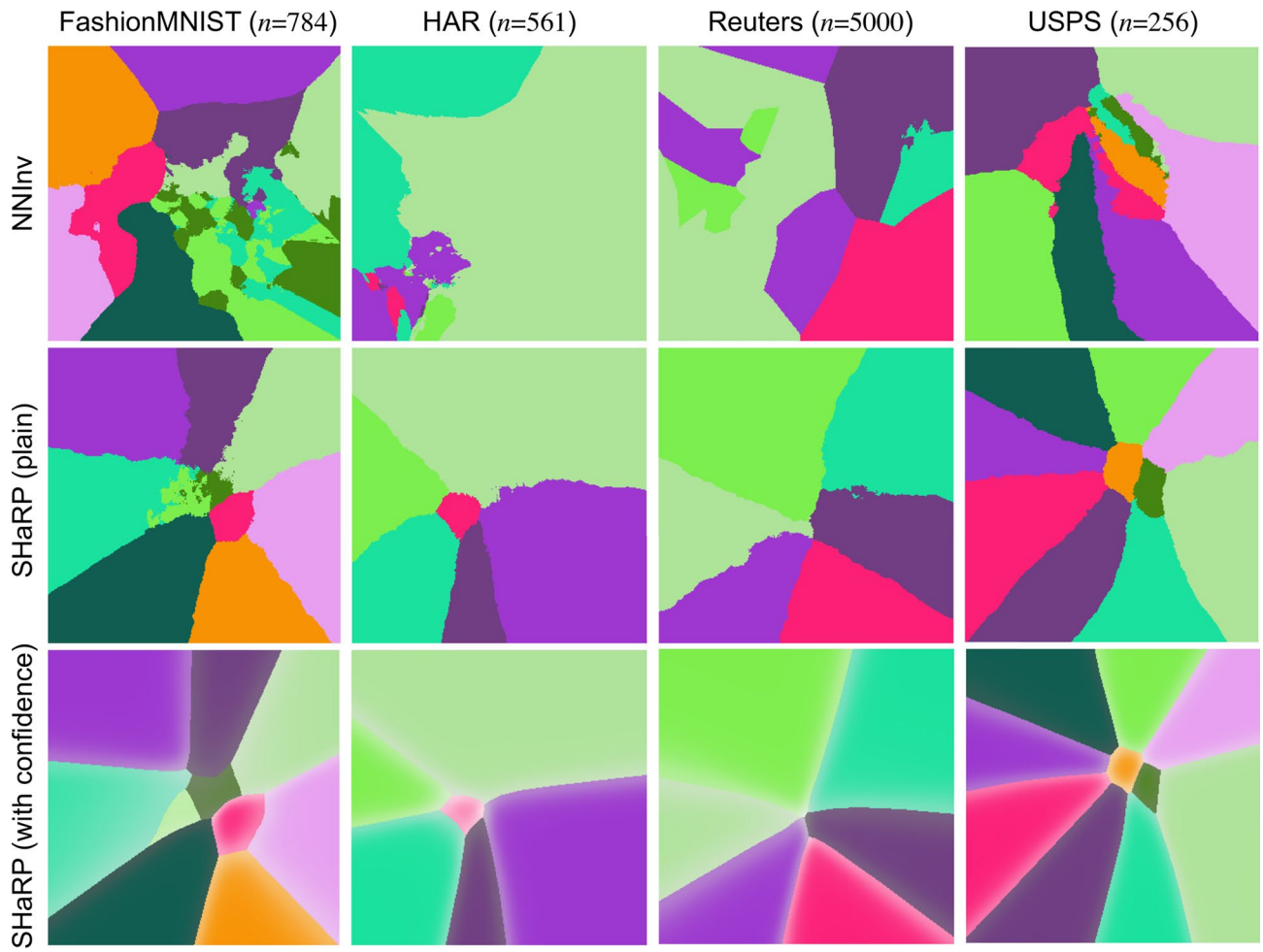

Fig. 12 Comparison of decision maps constructed by NNinv [80] (top row) and SHaRP [72] (middle row: plain map; bottom row: map with classifier confidence encoded into color saturation) inverse-projection techniques, for a KNN classifier and four datasets (n indicates the dataset dimensionalities). See "Deep Learning Inverse Projections"

Prospects of DR Assisting ML: Seeing to Learn Better

Better DBMs One recent, and unexpected, result of our analysis is that all DBM methods only visualize a surface-like subset of the high-dimensional data space (see the coverage discussion in "Decision Boundary Maps"). This tells us that not only a small, surface-like, subset of a classifier's behavior is thus visualized, but also that how this subset is selected is not under the user's control, but automatically determined by the inverse-projection method being used by the DBM algorithm. As such, the insights users will get from DBM methods applied to various classifiers and datasets will be highly dependent on the DBM technique in use. Even more critically, imagine a given, trained, classifier f whose input space is sampled by different test datasets D, each being used next to construct a decision map image. Such images will likely be (very) different since they depend on inverse projections constructed, in turn, from different datasets. This can be highly misleading given that we are visualizing the same, fixed, classifier f.

It is likely impossible to devise a DBM method that densely samples an *entire* high-dimensional space to construct a classifier image. As such, we see two ways forward to improving DBMs:

1. construct an inverse projection with predictable, guaranteed, *behavior* for a given sampling D of the data space;
2. allow users to *control* the surface that is backprojected by the inverse projection to sample the classifier.

Option 1 relates to how the aforementioned backprojected surfaces are constructed by the function P^{-1}. All designs of inverse projections essentially minimize a cost which tells that points of a given projection $P(\mathbf{x})$ have to backproject, via P^{-1}, one to one, to those of a given dataset $D = \{\mathbf{x}\}$ [58, 61, 71, 72, 77, 79, 80]. Hence, such

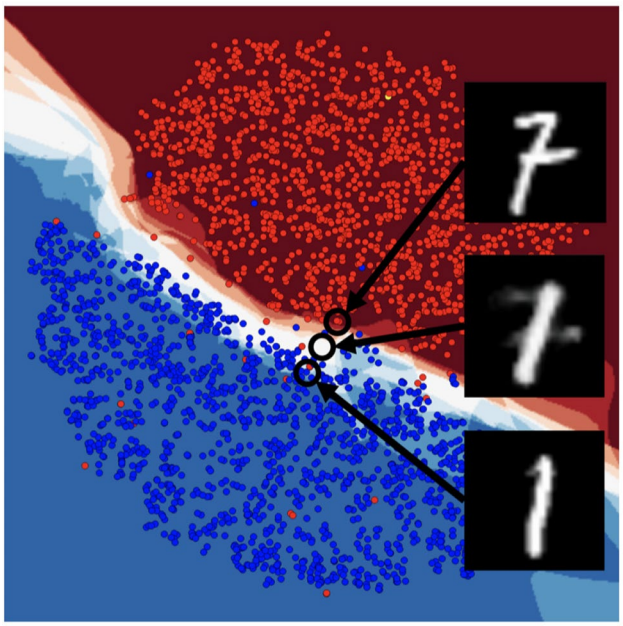

Fig. 13 Classifier agreement map for 9 classifiers, two-class problem (MNIST datasets digits 1 and 7). Dark colors indicate more of the 9 classifiers agreeing, at a pixel in the map, with their decisions (red = 1, blue = 7). Brighter, desaturated, colors indicate fewer classifiers in agreement (white=four classifiers output 1, the other five output 7, or conversely) [81]. See "Decision Boundary Maps"

backprojected surfaces can be thought of as level sets, or isosurfaces, of low values of the cost function. Analyzing them from this perspective can lead to important theoretical insights in their behavior.

Option 2 offers a more practical way forward. Simple ways can be devised to allow users to, e.g., shift this surface in given directions and/or by given amounts in the data space by means of interactive controls applied to the 2D image space. Similar ideas have been since long used, albeit in a different context, for the visualization of scalar functions of many variables [82]. The difference, in our case, is that we would start with a more complex surface, and would have to design intuitive ways to shift this surface in meaningful directions in the data space. This idea could be further extended by allowing for sampling a thick 'band' close to this surface. The challenge, in this case, would be to map this band to the 2D image space of a decision map.

DBMs in Use DBMs are not a goal in themselves, but a tool serving a goal. Apart from the scenarios depicted in Refs. [77, 79, 81], DBMs could be readily used in a visual analytics explorative scenario to drive a classifier's training. If computable in real-time, users could visualize the DBMs, find problematic areas with respect to how the decision boundaries wrap around samples, and next modify the training set by, e.g., adding or deleting labels, adding new augmented samples, or even moving samples. We envisage a tool in which users could effectively 'sculpt' the shape of decision boundaries by sample manipulation much as one edits 2D shapes by manipulating spline control points. This would offer unprecedented freedom and a wholly new way of fine-tuning classifiers to extend the approaches pioneered in Refs. [41, 44].

Visualizing Regressors All visualizations examples shown in this paper have covered only the depiction of classifiers that output a single categorical value. However, as "Background" mentions, ML also studies multi-valued classifiers and, further, single-valued and multi-valued regressors. Concerning decision maps, we are not aware of their extension to multi-valued classifiers. This could be achieved using multiple-view maps, one per classifier output, or categorical color-coding of all multi-valued class combinations in a single decision map. Concerning regressors, recent results have shown how to extend the decision map metaphor to visualize single-valued regressors [83, 84]. However, this research only used a relatively low-quality projection (PCA), so it could be readily explored how better direct and inverse projections, like the ones described in "Deep Learning Projections" and "Deep Learning Inverse Projections" could improve its results. Visualizing multi-valued regressors is a harder problem as several continuous values would need to be displayed at each pixel. To assist this, techniques developed earlier in scientific visualization (tensor visualization [85]) could offer an outcome.

Prospects of ML Assisting DR: Learning to See Better

Inverse-Projection Quality While many metrics exist to gauge the quality of direct projections ("Background"), there are no established ways to measure the quality of an inverse projection, apart from the simple mean-square-error (MSE) $\sum_{\mathbf{x}\in D} \|\mathbf{x} - P(P^{-1}(\mathbf{x}))\|$ [80]. This is not surprising since, as explained in "Deep Learning Inverse Projections", inverse projections are mainly used to infer, or hypothesize, what the data would be in locations where no ground-truth is present—much like classical ML models are used in regression. As such, defining what a good inverse projection should return in such areas is conceptually hard. Yet, possibilities exist. One can, e.g., use a ML approach where an unseen test set is kept apart from the construction of the inverse projection and is used to assess the quality of such a trained model using the aforementioned MSE. An equally interesting question is how to design a scale, or hierarchy, of errors. It is likely that differently inversely projected points $\mathbf{x}' = P^{-1}(P(\mathbf{x}))$ that deviate from its ground-truth location $\mathbf{x}$ by the same distance $\|\mathbf{x}' - \mathbf{x}\|$ are not equally good, or equally bad, depending on the application perspective. As

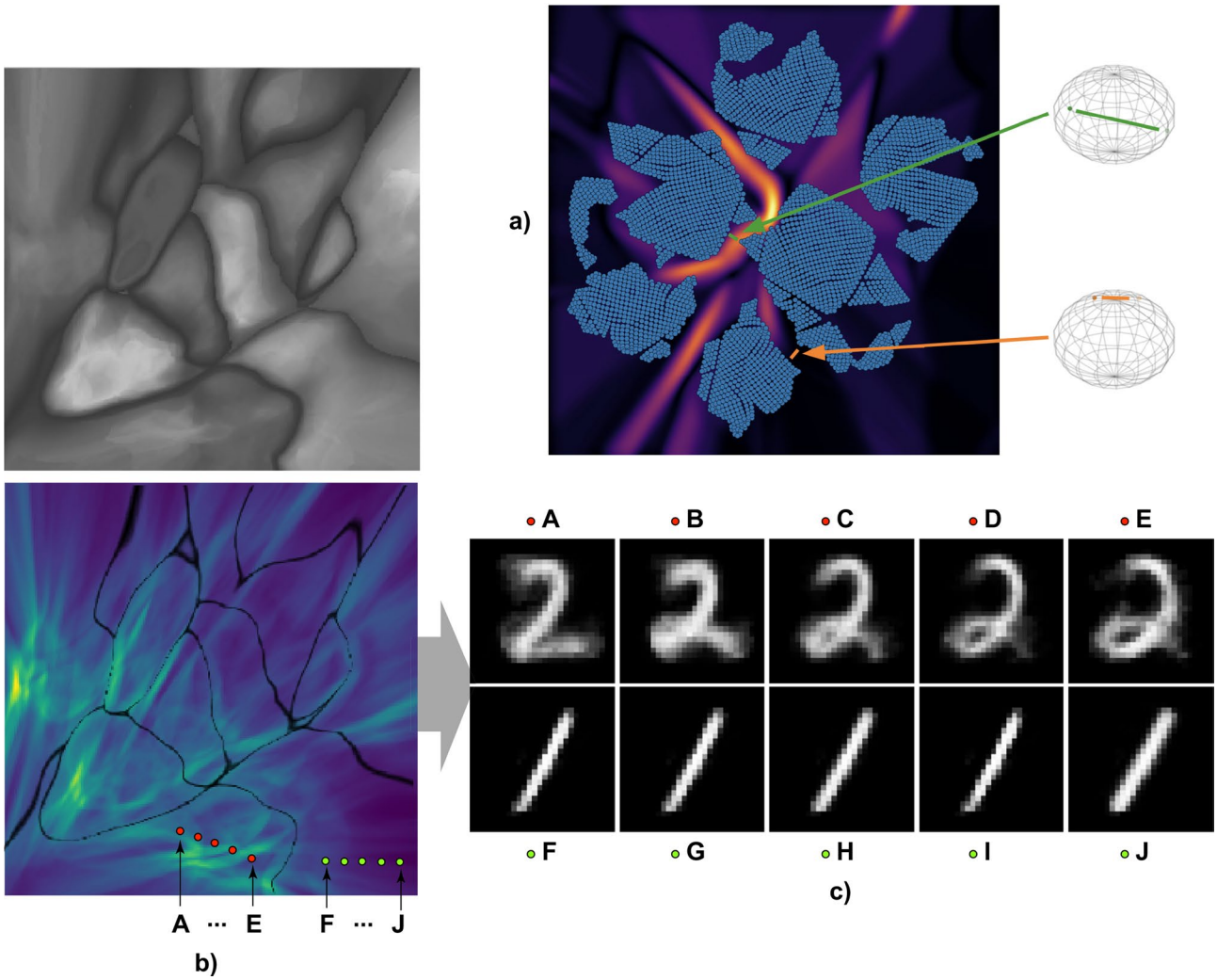

Fig. 14 **a** Gradient map of NNInv inverse projection constructed from a t-SNE projection of an uniformly sampled sphere. Hot, respectively, dark, regions indicate nearby 2D points that inversely project to far-apart, respectively, close, *n*D points (green line, top sphere; orange line, bottom sphere, respectively). **b** Gradient map of NNInv inverse projection used to construct the decision maps for the MNIST classification in Fig. 7, with distance-to-closest-boundary map at the top (grayscale). **c** Two regions of large, respectively, low, gradients are sampled by the red, respectively, green, points. The corresponding images generated by NNInv are shown and confirm the large, respectively, low, variations of the inverse projection in these areas. See "Prospects of ML Assisting DR: Learning to See Better"

such, inverse-projection quality metrics may need to be designed in an application-specific way.

Also, similarly to direct projections [14], the quality of inverse projections can be measured not only globally (by a single aggregate metric) but also locally, at every pixel. The explanatory visualizations in Fig. 7 can be thought as being such per-pixel quality maps (for classifiers). For inverse projections, we are aware of a single such per-pixel quality visualization—gradient maps [81]. Figure 14a shows this gradient map, which depicts the gradient magnitude of the P^{-1} function (in this case constructed with NNInv) at every pixel. Hot, respectively, dark, regions in the map indicate nearby 2D points which backproject far away from, respectively, close to, each other. Points in the hot regions thus indicate areas where the inverse projection may be unstable, and as such, potentially create misleading data. However, we cannot directly say that this is an error of the inverse projection P^{-1}. Such regions may correspond to areas where the *direct* projection P squeezed faraway data points to fit them in the 2D space—thus areas of low continuity [20]. Hence, analyzing inverse projection errors should go hand-in-hand with analyzing the errors of the direct projection it was computed for. For the latter, many per-pixel techniques are readily usable [10, 21, 25].

Figure 14b shows an additional use-case for gradient maps. The image depicts the gradient map of the NNInv inverse-projection method used to construct the decision map visualizations for the MNIST classifier explored in

Fig. 7. Atop of this gradient map, we overlaid the classifier confidence (Fig. 7d), so the dark bands in the image correspond to the classifier's decision boundaries. For clarity of exposition, we show atop image (b) the distance-to-closest-boundary (same information as encoded in the luminance in Fig. 7e). Image (b) tells us several insights. We see that large inverse-projection gradients occur both along decision boundaries but also deep inside the decision zones. Also, these large gradients are not correlated with areas of low, or high, distance-to-closest boundary. Hence, the gradient map tells additional information not present in earlier visualizations. This information helps seeing where a classifier will be exposed to high data variability, thus, meet more challenges. We show this by taking five points ($A \dots E$) in a low-gradient, and five others ($F \dots J$) in a high-gradient area, respectively. Figure 14c shows the MNIST images corresponding to these points. Indeed, we see how the respective digits vary significantly more in high-gradient areas than in low-gradient ones.

Learning styles All projection methods aim to encode the relative distance between data points in their resulting scatterplot. Atop of this, parametric projections aim to encode the actual data values. SHaRP extends this to force data clusters to specific shapes ("Deep Learning Projections"). Such strategies could be extended to map other data attributes, such as sample density or specific value ranges, to the size, shape, and/or position of point clusters in a projection. For DL methods, this could be done by refining their loss function. Additionally, SHaRP could be extended to create a hierarchy aware projection algorithm that would combine the advantages of treemaps and classical projections, extending earlier ideas in this class [86].

A second extension would be to design *local* cost functions that attempt to construct the projection by combining different criteria for different subsets of the input data—for example, to achieve a globally better projection that locally behaves like t-SNE in some areas and like UMAP in others. ML techniques can help here by, e.g., extending the HyperNP idea [73] to train from a set of projection techniques run on the same input dataset. Further inspiration can be gotten from recent ways in which DL is used for image synthesis and style transfer, e.g., [87].

Dynamic projections "Understanding DL Models" has briefly introduced dynamic projections. These are extensions of the standard, static, projection techniques which aim to handle a dataset consisting of high-dimensional points which maintain their identity while changing their attribute values through time. Dynamic projections have a wealth of applications—simply put, anywhere one wants to study high-dimensional data which changes over time. However, only a handful of dynamic projection techniques exist [32, 53, 88, 89], and their quality—as gauged by established quality metrics—is good in data structure preservation *or* data dynamics preservation but not both aspects. Designing a dynamic projection technique that accurately maps both data structure and dynamics is a grand challenge for the infovis community. Following the good results of using ML for DR ("Learning for Seeing: ML Assists DR", it looks highly interesting to explore ML (and in particular DL) to create dynamic projections. An issue here is that, since good ground-truth dynamic projections are relatively hard to construct, the supervised way (NNP-class methods) may be less preferable than the self-supervised (SSNP-like) direction.

Conclusions

In recent years, the research domains of dimensionality reduction (DR) and machine learning (ML) have came increasingly closer to each other, motivated by advances in ML techniques that help building better visualization algorithms, on the one hand, and by the need for visualization techniques to better explain the 'black box' behavior of ML (and in particular deep learning) methods. The two salient keywords that are often used to describe the two fields—seeing (for DR) and learning (for ML) have become increasingly connected.

In this paper, we have presented an overview of recent connections between the two fields, with a focus on techniques and methods in one field which assist tasks and use-cases in the other, and also satisfy overall desirable criteria as genericity, computational scalability, stability, and ease of use. We have made the case that the two fields are complementary, with key features being offered by methods in one field being required by methods in the other, therefore the potential for cross-fertilization. The first part of our overview ("Seeing for Learning: DR Assists ML") showed how DR can assist ML tasks by examples in assessing the behavior of general-purpose classifiers, pseudolabeling for creating large training sets, exploring the training and inference of deep learning models, and depicting the high-dimensional decision zones and boundaries of classifiers. The second part ("Learning for Seeing: ML Assists DR") showed how ML can assist DR by examples covering the deep learning of projections and inverse projections. We concluded our presentation by outlining several high-potential research directions at the crossroads of ML and DR based on the techniques discussed in this paper: using dense maps to explore and improve classifiers and regressors; using ML to create highly customized, high-quality projections for both static and dynamic data; and developing inverse projections to meet all the quality standards that current direct projection techniques have.

We see this convergence trend which unites research and researchers in DR and ML growing in the near future, with both areas positively feeding each other in terms of

research questions and tasks, and also solutions. A strong common mathematical background unites researchers in the two fields, making it easy to exchange research questions, ideas, and results. Also, tools and techniques in both areas become increasingly more available which eases the development of joint solutions. Such developments, jointly enabled by DR and ML researchers, will have impact far beyond these two fields.

Declarations

Conflict of interest On behalf of all the authors, the corresponding author states that there is no conflict of interest.

Human and animals participants This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Munzner T. Visualization analysis and design: principles, techniques, and practice. Boca Raton: CRC Press; 2014.
2. Telea AC. Data visualization—principles and practice. 2nd ed. Abingdon: CRC Press/Taylor and Francis; 2014.
3. Liu S, Maljovec D, Wang B, Bremer P-T, Pascucci V. Visualizing high-dimensional data: advances in the past decade. IEEE TVCG. 2015;23(3):1249–68.
4. Yates A, Webb A, Sharpnack M, Chamberlin H, Huang K, Machiraju R. Visualizing multidimensional data with glyph SPLOMs. CGF. 2014;33(3):301–10.
5. Lehmann DJ, Albuquerque G, Eisemann M, Magnor M, Theisel H. Selecting coherent and relevant plots in large scatterplot matrices. Comput Graph Forum. 2012;31(6):1895–908.
6. Inselberg A, Dimsdale B. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In: Proc. IEEE VIS. 1990. p. 361–78.
7. Rao R, Card SK. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In: Proc. ACM SIGCHI. 1994. p. 318–22.
8. Telea AC. Combining extended table lens and treemap techniques for visualizing tabular data. In: Proc. EuroVis. 2006. p. 120–7.
9. Borgo R, Kehrer J, Chung DHS, Maguire E, Laramee RS, Hauser H, Ward M, Chen M. Glyph-based visualization: foundations, design guidelines, techniques and applications. 2013.
10. Lespinats S, Aupetit M. CheckViz: sanity check and topological clues for linear and nonlinear mappings. CGF. 2011;30(1):113–25.
11. Sorzano C, Vargas J, Pascual-Montano A. A survey of dimensionality reduction techniques. arXiv:1403.2877 [stat.ML]. 2014.
12. Nonato L, Aupetit M. Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. IEEE TVCG. 2018;25(8):2650–73.
13. Cunningham J, Ghahramani Z. Linear dimensionality reduction: survey, insights, and generalizations. JMLR. 2015;16:2859–900.
14. Espadoto M, Martins R, Kerren A, Hirata N, Telea A. Toward a quantitative survey of dimension reduction techniques. IEEE TVCG. 2019;27(3):2153–73.
15. Telea A. Beyond the third dimension: how multidimensional projections and machine learning can help each other. In: Proc. IVAPP. 2023.
16. Botchkarev A. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: properties and typology. Interdiscip J Inf Knowl Manag. 2019;14:45–79.
17. Jiang T, Gradus J, Rosellini A. Supervised machine learning: a brief primer. Behav Ther. 2020;51(5):675–87.
18. Thiyagalingam J, Shankar M, Fox G, Hey T. Scientific machine learning benchmarks. Nat Rev Phys. 2022;4:413–20.
19. Joia P, Coimbra D, Cuminato JA, Paulovich FV, Nonato LG. Local affine multidimensional projection. IEEE TVCG. 2011;17(12):2563–71.
20. Venna J, Kaski S. Visualizing gene interaction graphs with local multidimensional scaling. In: Proc. ESANN. 2006. p. 557–62.
21. Martins R, Coimbra D, Minghim R, Telea AC. Visual analysis of dimensionality reduction quality for parameterized projections. Comput Graph. 2014;41:26–42.
22. van der Maaten L, Hinton GE. Visualizing data using t-SNE. JMLR. 2008;9:2579–605.
23. Paulovich FV, Nonato LG, Minghim R, Levkowitz H. Least square projection: a fast high-precision multidimensional projection technique and its application to document mapping. IEEE TVCG. 2008;14(3):564–75.
24. Sips M, Neubert B, Lewis J, Hanrahan P. Selecting good views of high-dimensional data using class consistency. CGF. 2009;28(3):831–8.
25. Aupetit M. Visualizing distortions and recovering topology in continuous projection techniques. Neurocomputing. 2007;10(7–9):1304–30.
26. Sommerville I. Software engineering. Sebastopol: O'Reilly Publishing; 2015.
27. da Silva R, Rauber P, Martins R, Minghim R, Telea AC. Attribute-based visual explanation of multidimensional projections. In: Proc. EuroVA. 2015.
28. Coimbra D, Martins R, Neves T, Telea A, Paulovich F. Explaining three-dimensional dimensionality reduction plots. Inf Vis. 2016;15(2):154–72.
29. Marcilio WE, Eler DM. Explaining dimensionality reduction results using Shapley values. arXiv:2103.05678 [cs.LG]. 2021.
30. Tian Z, Zhai X, Driel D, Steenpaal G, Espadoto M, Telea A. Using multiple attribute-based explanations of multidimensional projections to explore high-dimensional data. Comput Graph. 2021;98(C):93–104.
31. Thijssen J, Tian Z, Telea A. Scaling up the explanation of multidimensional projections. In: Proc. EuroVA. 2023.
32. Vernier E, Comba J, Telea A. Guided stable dynamic projections. Comput Graph Forum. 2021;40(3):87–98.
33. Garcia R, Telea A, Silva B, Torresen J, Comba J. A task-and-technique centered survey on visual analytics for deep learning model engineering. Comput Graph. 2018;77:30–49.

34. Hohman F, Kahng M, Pienta R, Chau DH. Visual analytics in deep learning: an interrogative survey for the next frontiers. IEEE TVCG. 2019;25(8):2674–93.
35. Yuan J, Chen C, Yang W, Liu M, Xia J, Liu S. A survey of visual analytics techniques for machine learning. Comput Visual Media. 2020;7:3–36.
36. Alicioglu G, Sun B. A survey of visual analytics for explainable artificial intelligence methods. Comput Graph. 2022;102(C):502–20.
37. Rauber PE, Falcão AX, Telea AC. Projections as visual aids for classification system design. Inf Vis. 2017;17(4):282–305.
38. Guyon I, Gunn S, Ben-Hur A. Result analysis of the NIPS 2003 feature selection challenge. In: Advances in neural information processing systems; 2004. p. 545–52
39. Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Mach Learn. 2006;63(1):3–42.
40. Bernard J, Hutter M, Zeppelzauer M, Fellner D, Sedlmair M. Comparing visual-interactive labeling with active learning: an experimental study. IEEE TVCG. 2018;24(1):298–308.
41. Benato B, Telea A, Falcão A. Semi-supervised learning with interactive label propagation guided by feature space projections. In: Proc. SIBGRAPI. 2018. p. 392–9.
42. Belkin M, Niyogi P, Sindhwani V. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J Mach Learn Res. 2006;7:2399–434.
43. Amorim WP, Falcão AX, Papa JP, Carvalho MH. Improving semi-supervised learning through optimum connectivity. Pattern Recognit. 2016;60(C):72–85.
44. Benato B, Gomes J, Telea A, Falcão A. Semi-automatic data annotation guided by feature space projection. Pattern Recognit. 2020;109:107612.
45. Shwartz-Ziv R, Tishby N. Opening the black box of deep neural networks via information. arXiv:1703.00810 [cs.LG]. 2017.
46. Azodi C, Tang J, Shiu S. Opening the black box: interpretable machine learning for geneticists. Trends Genet. 2020;36(6):442–55.
47. Tzeng FY, Ma K-L. Opening the black box—data driven visualization of neural networks. In: Proc. IEEE visualization. 2005.
48. Pezzotti N, Höllt T, Van Gemert J, Lelieveldt BPF, Eisemann E, Vilanova A. Deepeyes: progressive visual analytics for designing deep neural networks. IEEE TVCG. 2017;24(1):98–108.
49. Alsallakh B, Jourabloo A, Ye M, Liu X, Ren L. Do convolutional neural networks learn class hierarchy? IEEE Trans Vis Comput Graph. 2018;24(1):152–62.
50. Strobelt H, Gehrmann S, Pfister H, Rush AM. LSTMVis: a tool for visual analysis of hidden state dynamics in recurrent neural networks. IEEE TVCG. 2018;24(1):667–76.
51. Liu M, Shi J, Li Z, Li C, Zhu J, Liu S. Towards better analysis of deep convolutional neural networks. IEEE TVCG. 2016;23(1):91–100.
52. Chattopadhay A, Sarkar A, Howlader P, Balasubramanian V. Grad-CAM++: generalized gradient-based visual explanations for deep convolutional networks. In: Proc. IEEE WACV. 2018.
53. Rauber P, Falcao A, Telea A. Visualizing time-dependent data using dynamic t-SNE. In: Proc. EuroVis—short papers; 2016. p. 43–9.
54. Zwan M, Codreanu V, Telea A. CUBu: universal real-time bundling for large graphs. IEEE TVCG. 2016;22(12):2550–63.
55. Rauber P, Fadel SG, Falcão A, Telea A. Visualizing the hidden activity of artificial neural networks. IEEE TVCG. 2017;23(1):101–10.
56. Rodrigues FCM, Espadoto M, Hirata R Jr, Telea A. Constructing and visualizing high-quality classifier decision boundary maps. Information. 2019;10(9):280–97.
57. Oliveira AAM, Espadoto M, Hirata R, Telea A. SDBM: supervised decision boundary maps for machine learning classifiers. In: Proc. IVAPP. 2022.
58. Schulz A, Gisbrecht A, Hammer B. Using discriminative dimensionality reduction to visualize classifiers. Neural Process Lett. 2015;42(1):27–54.
59. LeCun Y, Cortes C, Burges C. MNIST handwritten digit database. AT &T Labs. http://yann.lecun.com/exdb/mnist. 2010. Accessed 15 Sept 2023.
60. Moosavi-Dezfooli S, Fawzi A, Frossard P. Deepfool: a simple and accurate method to fool deep neural networks. In: Proc. IEEE CVPR. 2016. p. 2574–82.
61. Schulz A, Hinder F, Hammer B. DeepView: visualizing classification boundaries of deep neural networks as scatter plots using discriminative dimensionality reduction. In: Bessiere C, editor. Proc. IJCAI. 2020. p. 2305–11.
62. Colding TH, Minicozzi WP. Shapes of embedded minimal surfaces. PNAS. 2006;103(30):11106–11.
63. McInnes L, Healy J, Melville J. UMAP: uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426v2 [stat.ML]. 2018.
64. Minghim R, Paulovich FV, Lopes AA. Content-based text mapping using multi-dimensional projections for exploration of document collections. In: Proc. SPIE. 2006. Intl. Society for Optics and Photonics.
65. Paulovich FV, Minghim R. Text map explorer: a tool to create and explore document maps. In: Proc. IEEE IV. 2006. p. 245–51.
66. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313(5786):504–7.
67. Pekalska E, Ridder D, Duin RPW, Kraaijveld MA. A new method of generalizing Sammon mapping with application to algorithm speed-up. Proc ASCI. 1999;99:221–8.
68. Espadoto M, Hirata N, Telea A. Deep learning multidimensional projections. Inf Vis. 2020;9(3):247–69.
69. Bredius C, Tian Z, Telea A. Visual exploration of neural network projection stability. In: Proc. MLVis. 2022.
70. Modrakowski T, Espadoto M, Falcao A, Hirata N, Telea A. Improving deep learning projections by neighborhood analysis. In: Communication in computer and information. 2021.
71. Espadoto M, Hirata N, Telea A. Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In: Proc. IVAPP. 2021.
72. Machado A, Behrisch M, Telea A. ShaRP: shape-regularized multidimensional projections. In: Proc. EuroVA. 2023.
73. Appleby G, Espadoto M, Chen R, Goree S, Telea A, Anderson E, Chang R. HyperNP: interactive visual exploration of multidimensional projection hyperparameters. CGF. 2022;41(3):169–81.
74. Kim Y, Espadoto M, Trager S, Roerdink J, Telea A. SDR-NNP: sharpened dimensionality reduction with neural networks. In: Proc. IVAPP. 2022.
75. Comaniciu D, Meer P. Mean shift: a robust approach toward feature space analysis. IEEE TPAMI. 2002;24(5):603–19.
76. Rodrigues FCM, Jr, RH, Telea A. Image-based visualization of classifier decision boundaries. In: Proc. SIBGRAPI. 2018.
77. Amorim E, Brazil E, Daniels J, Joia P, Nonato L, Sousa M. iLAMP: exploring high-dimensional spacing through backward multidimensional projection. In: Proc. IEEE VAST. 2012.
78. Mamani GMH, Fatore FM, Nonato LG, Paulovich FV. User-driven feature space transformation. Comput Graph Forum. 2013;32(3):291–9.

79. Amorim E, Brazil E, Mena-Chalco J, Velho L, Nonato LG, Samavati F, Sousa M. Facing the high-dimensions: inverse projection with radial basis functions. Comput Graph. 2015;48:35–47.
80. Espadoto M, Rodrigues FCM, Hirata NST, Jr, RH, Telea A. Deep learning inverse multidimensional projections. In: Proc. EuroVA. 2019.
81. Espadoto M, Appleby G, Suh A, Cashman D, Li M, Scheidegger C, Anderson E, Chang R, Telea A. UnProjection: leveraging inverse-projections for visual analytics of high-dimensional data. IEEE TVCG. 2021;29(2):1559–72.
82. Wijk JJ, Liere R. Hyperslice: Visualization of scalar functions of many variables. In: Proc. IEEE visualization. 1993. p. 119–25.
83. Espadoto M, Rodrigues FCM, Hirata N, Telea A. OptMap: using dense maps for visualizing multidimensional optimization problems. In: Proc. IVAPP. 2021.
84. Espadoto M, Rodrigues FCM, Hirata NST, Telea AC. Visualizing high-dimensional functions with dense maps. SN Comput Sci. 2023. https://doi.org/10.1007/s42979-022-01664-2.
85. Weickert J, Hagen H. Visualization and processing of tensor fields. Berlin: Springer; 2005.
86. Duarte F, Sikanski F, Fatore F, Fadel S, Paulovich FV. Nmap: a novel neighborhood preservation space-filling algorithm. IEEE TVCG. 2014;20(12):2063–71.
87. Luan F, Paris S, Shechtman E, Bala K. Deep photo style transfer. In: Proc. IEEE CVPR. 2017.
88. Vernier E, Garcia R, Silva I, Comba J, Telea A. Quantitative evaluation of time-dependent multidimensional projection techniques. Comput Graph Forum. 2020;39(3):241–52.
89. Neves TTT, Martins RM, Coimbra DB, Kucher K, Kerren A, Paulovich FV. Fast and reliable incremental dimensionality reduction for streaming data. Comput Graph. 2022;102:233–44.