Computers & Graphics 124 (2024) 104062

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on SIBGRAPI 2024

Human-in-the-loop: Using classifier decision boundary maps to improve pseudo labels

Bárbara C. Benato [a,*,1], Cristian Grosu [b,1], Alexandre X. Falcão [a], Alexandru C. Telea [b]

[a] Institute of Computing, University of Campinas, Campinas, Brazil
[b] Department of Information and Computing Sciences, Faculty of Science, Utrecht University, Utrecht, The Netherlands

ARTICLE INFO

Keywords:
Active learning
Visual analytics
Pseudo labeling
Decision boundary maps

ABSTRACT

For classification tasks, several strategies aim to tackle the problem of not having sufficient labeled data, usually by automatic labeling or by fully passing this task to a user. Automatic labeling is simple to apply but can fail handling complex situations where human insights may be required to decide the correct labels. Conversely, manual labeling leverages the expertise of specialists but may waste precious effort which could be handled by automatic methods. More specifically, automatic solutions could be improved by combining an active learning loop with manual labeling assisted by visual depictions of a classifier's behavior. We propose to include the human in the labeling loop by using manual labeling in feature spaces produced by a deep feature annotation (DeepFA) technique. To assist manual labeling, we provide users with visual insights on the classifier's decision boundaries. Finally, we use the manual and automatically computed labels jointly to retrain the classifier in an active learning (AL) loop scheme. Experiments using a toy and a real-world application dataset show that our proposed combination of manual labeling supported by visualization of decision boundaries and automatic labeling can yield a significant increase in classifier performance with a quite limited user effort.

1. Introduction

Data acquisition has been massively favored by many applications. Yet, getting a good amount of labeled data is crucial when training supervised classifiers, especially for deep neural networks (DNNs) [1,2]. While label acquisition is reasonably cheap for some applications, this can be costly for image data, particularly when specialists, *e.g.* from Medicine or Biology, need to carefully study each image.

Several strategies aim to tackle the problem of insufficient labeled data (a) automatically, (b) interactively, or (c) by combining (a) and (b). We concern ourselves with cases where we have only a few supervised image samples and also need to label unsupervised samples. As such, few-shot-learning solutions [3] are out of our scope.

a. Automatic labeling: Semi-supervised learning (SSL) [3] and pseudo labeling [4] are well-known methods in this class. SSL extracts label information from a few supervised samples while capturing additional information on data distribution from many unsupervised samples [3]. As such, SSL can both improve the performance of a DNN and increase the number of labeled samples. Deep learning approaches [5–8] have been used to propagate labels from a few supervised samples to many unsupervised ones by exploiting their feature-space distribution. The scarce label information is effectively 'propagated' over the training set, leading to *pseudo labels* – *i.e.*, labels that are not assigned by users but are used exactly as true labels for training. If we assume that the training set accurately captures the data distribution for the problem at hand, then such information can be enough to train high-performance classifier models. Yet, SSL and pseudo-labeling techniques still need hundreds to thousands of supervised samples for training and/or hyperparameter optimization [6–8]. Aiming to solve such problems, a deep feature annotation (DeepFA) [9,10] technique allows both learning deep features and labeling them using only dozens of supervised samples. DeepFA combines graph-based pseudo labeling and non-linear projection techniques to iteratively improve feature spaces. Although this approach allows users to visually inspect the (pseudo labeled) data by a projection, the user is not actively *involved* in the labeling.

b. Manual labeling: Crowdsource labeling is a popular way to involve users with labeling [11], but works well only for applications where specific knowledge is not important for the labeling task. Active learning (AL) [12] and visual-interactive machine learning [13–16] techniques play an important role in this context. In AL, a deep-learned

[*] Correspondence to: Institute of Computing, University of Campinas, Av. Albert Einstein, 1251, 13083-852 Campinas, SP, Brazil.
E-mail addresses: barbarabenato@gmail.com (B.C. Benato), cristi2019255@gmail.com (C. Grosu), afalcao@ic.unicamp.br (A.X. Falcão), a.c.telea@uu.nl (A.C. Telea).
[1] Both authors had equal contributions.

https://doi.org/10.1016/j.cag.2024.104062
Received 27 April 2024; Received in revised form 17 August 2024; Accepted 23 August 2024
Available online 30 August 2024

model selects a few samples based on some criteria and gives them to users to label [17]; the model is retrained with the user-labeled samples; and the loop repeats. The pseudo-labeled samples are expected to boost classifier performance after a few iterations. While AL-based deep learning solutions do include visual interactive labeling [17], these do not consider *iterated* user interaction. Overall, AL and visual-interactive labeling have been compared in the literature [14], but they have not been explored together.

c. Combined labeling: To amend the above issues, some methods combine automatic and interactive approaches. The human ability to surpass automatic pseudo-labeling techniques was first assessed in [18]. Later, combined manual and automatic labeling was proposed [19], using AL concepts of sample informativeness [12] to do automatic labeling in high-confidence areas, leaving low-confidence areas for manual labeling. Yet, an AL *loop* that incorporates user input was not considered.

Given these gaps, we propose in this work to include the human in the loop by (i) considering manual labeling in feature spaces produced by DeepFA, (ii) helping user labeling by visualizing the trained model's behavior, and (iii) using manual labels to retrain a classifier in an AL looping scheme. Specifically, we use Decision Boundary Maps (DBMs) [20] and direct-and-inverse projection errors [20] as visual aids to help users decide where, in a projection, to concentrate their manual labeling efforts. We present a controlled experiment that shows that our combination of automatic and manual labeling, supported by DBMs, allows users to improve classification performance by non-negligible factors, for both a simple and a challenging, real-world, dataset, and with very limited user effort. To our knowledge, this is the first application that measures how DBMs can be effectively used to improve classifier performance with only a few supervised samples, complex datasets, and combining manual and automatic labeling.

2. Related work

Preliminaries. Let $D = \{\mathbf{x_i}\}$, $1 \leq i \leq N$ be a dataset of N samples, in which $\mathbf{x} = (x^1, x^2, \ldots, x^n)$, $x^i \in \mathbb{R}^n$ and $1 \leq i \leq n$, a n-dimensional (nD) real sample. We call the values x^i, $1 \leq i \leq n$, the dimensions (attributes, variables, or features) of sample $\mathbf{x}$.

We call D *supervised* if there is a pair $(\mathbf{x_i}, c_i) \in D \times C$ for all $\mathbf{x}_i$ in D, where c_i is called the label of sample $\mathbf{x}_i$. In a supervised D, labels c_i are known as *true labels* and are manually assigned by a human or come from other trusted data sources. In an unsupervised dataset D, c_i is unknown. A label $c_i \in C$ can be assigned to a sample $\mathbf{x_i}$ by a *labeling process*. For classification problems, C is a categorical domain, and labels $c \in C$ are also known as classes. A classifier for D is a function $f : D \to C$ that maps samples to class labels in a supervised way.

Direct and inverse projections. Dimensionality reduction (DR) methods, also called *projections*, take a dataset D to create a scatterplot (or embedding) $P(D) = \{\mathbf{y}_i = P(\mathbf{x}_i) | \mathbf{y}_i \in \mathbb{R}^v\}$, where typically $v \in \{2, 3\}$. Without generality loss, we next consider $v = 2$, *i.e.*, we project data to 2D.

An inverse projection $P^{-1} : \mathbb{R}^v \to \mathbb{R}^n$ is a function that aims to 'revert' the effect of a given projection P. More formally, given a projection $P(D)$ of a dataset D, its inverse is a function $P^{-1} : \mathbb{R}^v \to \mathbb{R}^n$ that minimizes the cost $\sum_{\mathbf{x}_i \in D} d(P^{-1}(P(x_i)) - \mathbf{x}_i)$ for a given metric d (typically, MAE or MSE); and smoothly varies as the input of P^{-1} changes over $\mathbb{R}^v$. Several techniques have been proposed to construct inverse projections. Early on, autoencoders (AEs) minimized a reconstruction error [21] – the encoder part of the AE computes P, while the decoder part computes P^{-1}. iLAMP [22] explored local affine mappings to compute P^{-1} for the direct projection LAMP [23]. The NNP technique [24] used deep learning to train a regressor to produce $P(D)$ for any given dataset D and projection technique P. NNInv [25] swapped the roles of D and $P(D)$ – given a 2D scatterplot $P(D)$, NNP regresses it to the corresponding dataset D. Improvements of NNInv include Self-Supervised Neural Projection (SSNP, [26]), which learns both P and P^{-1} with strong cluster separation based on data (pseudo)labeling; and Shape-Regularized Multidimensional Projection (ShaRP, [27]), which does the same using a variational autoencoder design to constrain the shapes of the obtained point clusters in $P(D)$. Compared to iLAMP, deep-learning inverse projections (AE, SSNP, NNInv, ShaRP) are much faster and are parameter-free.

Active learning. Classical AL pipelines work as follows: An algorithm selects a set of samples based on specified criteria and passes them to a user for labeling or inspection. The user-provided information is given to the learner so the learner can improve itself by it [12]. The process iterates until some pre-established stopping criteria are reached, *e.g.*, a desired classification performance or a maximal user effort being spent.

Modern classifiers use mainly deep learning techniques such as deep CNNs [28,29], deep restricted Boltzmann machines [30], Bayesian CNNs [31], and DBNs [32,33] for the AL task. Additionally, some studies explored incremental CNN learning [28] and incremental dictionary learning [33] with few layers. Given our interest in labeling image data, we focus next on studies that consider AL and deep CNNs.

Given that AL strategies require many supervised samples for training the deep model in the first iteration [17], solutions have considered user interaction using projected spaces. A recent study [34] aimed to *simulate* user labeling in AL looping, using an improved semi-supervised extension of the t-SNE [35] projection, where t-SNE plays the role of user labeling. Yet, this work did not actually *involve* users in the looping.

Iwata et al. [13] proposed an interactive visual analytics (VA) AL framework which selects objects for the user to relocate to obtain a desired visualization. The main goal was to obtain better visualizations using AL rather than create labeled datasets to train classifiers. Bernard et al. [16] presented a systematic quantitative analysis of 10 different user strategies (called computational building blocks) commonly used for selecting samples to label using projections. These user strategies include labeling outliers, density regions, or cluster borders first (among other similar ones). The performance of such strategies is analyzed and compared with 7 AL strategies through experiments using different datasets. Later, Bernard et al. [14] compared the performance of the above-mentioned 10 visual-interactive and other 16 AL labeling strategies, including single and multiple classifiers. Their findings suggest that visual-interactive labeling can *outperform* AL when class distributions are well separated in projections. While such studies compared AL and visual-interactive labeling, they did not *combine* these techniques. In contrast, on our work we perform precisely this combination by first performing automatic labeling and next enabling the user to refine these labels using a set of visual depictions of a classifier's behavior which go beyond the raw projection of labeled samples used in Bernard et al.'s work. Finally, all above-mentioned studies only used relatively simple datasets. Real-world complex datasets provide more complex feature spaces which are more challenging to handle via AL (with or without VA-based techniques). In our work, we consider both a relatively simple dataset, and a complex, real-world, one.

Pseudo labeling. Being a special case of self-training, pseudo labeling was first proposed to fine-tune a pre-trained model [4]. Still, label propagation errors can negatively affect the classification performance of models trained with pseudo labels [18,36]. The confidence of the apprentice model was included in the loss function to mitigate such problems [5,37]. Recently, pseudo labeling approaches [4,7,8] essentially adopt the semi-supervised strategy with the apprentice model assigning uncertain (pseudo) labels to unsupervised samples. Such approaches have been also combined with different strategies, *e.g.*, self-supervised methods [38,39]. For the same goal, meta-pseudo-labeling [8] uses an auxiliary model (teacher) to generate pseudo labels to train the primary model (student). Yet, to get reasonable label propagation accuracy, such deep-learning-based methods require a training set with hundreds of supervised samples per class and a validation set with additional supervised samples for parameter optimization [4,6–8]. When only a *few* supervised samples (*e.g.*, dozens per class) are available, this requirement is a clear blocker for using such methods.

Recently, *Embedded Pseudo Labeling* (EPL) [18] proposed pseudo labeling starting from only dozens of supervised samples, without needing validation sets with more supervised samples. EPL projects the data D to 2D the latent feature space extracted from a deep neural network (DNN) using autoencoders [19] and pre-trained architectures [9]. Pseudo labels are next propagated in the 2D projection from supervised to unsupervised samples using the OPFSemi [40] method. OPFSemi's value for pseudo labeling was studied in [18,41,42], showing that it can surpasses many other similar methods. In brief, OPFSemi considers each sample as a node of a complete graph, weighted by a distance function (usually Euclidean) between samples. It defines the cost of a path connecting two nodes as the maximum arc weight along the path. From the training nodes, the supervised ones are used as seeds to compute a minimum-cost path forest, such that each seed defines a tree and assigns its label to the most closely connected unsupervised nodes of its tree. For additional details, we refer the reader again to the full description in [40].

Deep feature annotation. The quality of the pseudo labels produced by EPL is constrained by the initial feature space. If the feature learning step produces a feature space with poor visual separation, then pseudo labeling will fail. Deep feature annotation (DeepFA) [9,10] circumvented this problem by improving feature learning over EPL *iterations*, using only 1% of all supervised samples. In DeepFA, the teacher (OPFSemi, a semi-supervised classifier) uses modifications of a given latent feature space of the student (a DNN) along with iterations of a 2D projection (t-SNE [43]) for pseudo-labeling. At each iteration, pseudo labeled samples are used to retrain the DNN, modifying its latent feature space. A few iterations of the training loop with truly-and-artificially-labeled samples improve the DNN's generalization performance. Yet, this approach does not consider user manual labeling to improve its pseudo labels.

Decision Boundary Maps (DBMs). DBMs construct a *dense* visual representation of the behavior of a trained ML model f, allowing users to inspect f's behavior outside of some limited training or test set. Given a dataset D (which can be f's training set, test set, or a combination of both), a projection $P(D)$ is created. Next, an inverse projection P^{-1} is computed from D and $P(D)$, using any of the methods described earlier. The 2D space in which $P(D)$ lives is discretized in a pixel grid G. For every pixel $\mathbf{y} \in G$, its inverse projection $P^{-1}(\mathbf{y})$ is computed, and $\mathbf{y}$ is colored to show the inferred label $f(P^{-1}(\mathbf{y}))$. Fig. 1 (right) shows a DBM for the classifier f and projection $P(D)$ depicted in the 2D scatterplot in the left image. Same-color regions in the DBM show the *decision zones* of the classifier, *i.e.*, samples for which f infers the same label. Neighboring pixels having different colors in the DBM show the *decision boundaries*, *i.e.*, samples where f changes value.

Several techniques were proposed to compute DBMs. Early on, DeepView [44] (and next [45]) used UMAP to compute both P and P^{-1}, extrapolating P^{-1} to all DBM points by minimizing a Kullback–Leibler (KL) divergence that models similarity probabilities in both 2D and data space. DeepView yields smooth DBMs but is orders of magnitude slower than other DBM methods [46]. Rodrigues et al. [20] used t-SNE and LAMP for P and iLAMP for P^{-1}. Yet, their method creates many 'spurious islands', *i.e.*, small areas in a DBM image that appear as different decision zones from their surroundings. Such areas are highly improbable given the smooth nature of most classifiers. This method was refined by removing points in $P(D)$ with high projection errors, which lead to spurious islands; and by encoding the confidence of the classifier f in the DBM brightness [47]. Supervised Decision Boundary Maps (SDBM, [48]) used the SSNP projection [26] to construct DBMs with far smoother decision boundaries than [20], which are thus easier

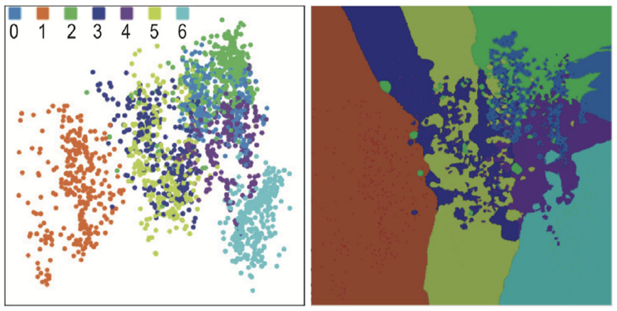

Fig. 1. Left: Projection scatterplot colored by labels inferred by a classifier f. Right: Decision map shows how f operates on additional points. Same-color areas are f's decision zones. Neighboring pixels of different colors show f's decision boundaries. *Source:* Image adapted from [20].

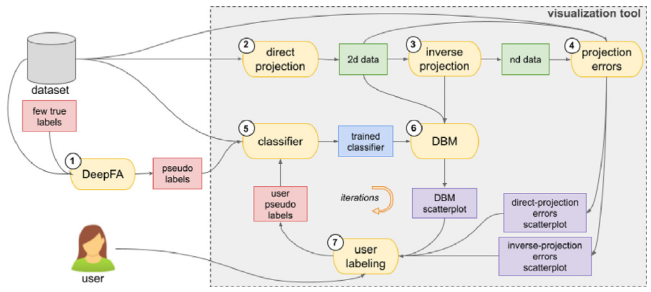

Fig. 2. Pipeline of the proposed interactive approach.

to explore visually. A recent study compared DBM methods on their ability to accurately display the behavior of a given classifier [46].

DBMs have been used to *explain*, but not *improve*, classifiers. The only exception we know [49] explored user interaction with DBMs in an AL looping to label samples. The author combined 2D projections and DBMs to manually label samples and improve a classifier over active learning iterations. Evaluation was performed on three synthetic datasets containing two and three classes, and a subset with four and ten classes of Fashion MNIST, respectively. Only one user did the evaluation using only one set/split dataset. In short, this work (1) computes classification performance over validation and test sets, which requires many supervised samples (as already explained); (2) does not consider combining automatic and manual labeling; (3) was tested only on relatively simple classification problems, and without a formal measurement of user effort. Our work next improves on all these aspects as follows. We propose to (1) compute classification performance over the labeled set – the one labeled by DeepFA and given as input in the first iteration – without the need for extra supervised samples for validation; (2) combine automatic and manual labeling, which saves the user significant time as one only needs to focus on a small subset of points to manually label; (3) validate our approach on a significantly more challenging, unbalanced, real-world dataset coming from biology; and (4) combine different sources of visual information to help the user with manual labeling.

3. Proposed method

We next detail our proposed pipeline that combines DBM techniques and manual labeling in an active learning loop to improve classifier training. The steps of our pipeline are as follows (see also Fig. 2).

(1) DeepFA: We start with a dataset having only a few true labels. We want to get a fully labeled dataset in the end, with minimal user labeling effort, so we use DeepFA (see Section 2) to create pseudo labels, with only 1% of supervised samples. We use these pseudo labels to train the classifier in step 5. For selecting these 1% samples, we used a stratified random approach. Other (AL) approaches for this sample selection could be used, *e.g.*, based on entropy, uncertainty, and diversity. Given the space constraints, we consider such additional experiments as part of future work.

(2) Direct projection: To embed data in 2D, we evaluated several different techniques: autoencoders [21], PCA [50], t-SNE [35], SSNP [26], and UMAP [51].

(3) Inverse projection: To compute a DBM, we need an inverse projection to map 2D points to data space (see Section 2). For this, we evaluated autoencoders [21], SSNP [26], and NNinv [25].

(4) Visualization errors: We use the data points, their projections, and the computed inverse projection function to measure errors of both P and P^{-1} (Sections 3.1 and 3.2).

(5) Classifier training: We train a classifier with 70% of samples (including pseudo labels created by DeepFA), keeping 30% to compute classification performance metrics. We can handle any classifier as our pipeline treats it as a black box $f : D \to C$; in our experiments, we use a deep neural network for f (Section 4).

(6) DBM computation: We visualize the trained classifier using the DBM techniques discussed earlier in Section 2.

(7) Manual labeling: Users can inspect the projection of the input dataset, the direct and inverse projection errors (computed at step 4), and the DBM of the trained classifier (computed at step 6) to decide which samples to manually label (Section 3.3). We use these newly-created pseudo labels to re-train the classifier – *i.e.*, the pipeline re-starts at step 5. Steps 5–7 are thus our active learning loop. Looping ends when the classifier has achieved a desired target performance or when the user decides that enough manual labeling effort has been put in the process.

3.1. Direct projection errors

Showing *local* projection errors can help users decide where to put their labeling effort. Assume a group of projection points is marked as having high errors. Then, the information shown in that area – class labels; decision zones and decision boundaries shown by a DBM – can be misleading. Let us refine these cases. Consider an area in a projection $P(D)$ showing inferred labels by color coding. Such an area can contain a mix of many colors – see Fig. 1 (left). This can lead users to believe that the classifier behaves poorly in that area. Yet, this color mix can be an *artifact* of P – the classifier may perfectly work in that area. The same is true for DBMs: An area in a DBM can show tortuous decision boundaries or many small-scale islands – see Fig. 1 (right). Such artifacts can be caused by P or P^{-1} errors rather than actual classifier problems.

To understand such cases, we compute the projection error metrics trustworthiness T [52] and continuity C [52] *locally* – that is, for every point $\mathbf{y} \in P(D)$. To simplify the user's task in assessing errors, we next combine these into

$$\epsilon(\mathbf{y}) = ((1 - T(\mathbf{y})) + (1 - C(\mathbf{y})))\,/2. \tag{1}$$

Ranging in $[0, 1]$, ϵ is easy to interpret: ϵ close to 1 tells that P is poor close to point $\mathbf{y}$; ϵ close to 0 imply a good P close to $\mathbf{y}$.

To *visualize* ϵ, we could simply color the projection points $\mathbf{y}$ by its values. Yet, this would make it hard to see *regions* of points which have high, respectively low, error values, especially in the presence of potential overplotting. A slightly better solution is to extrapolate ϵ up to a small, fixed, distance ρ from the points $\mathbf{y}$, using radial basis functions, as done in [53,54]. Yet, controlling ρ is tricky — too small ρ values yield large empty areas in the projection; too large ρ values yield overestimated error values.

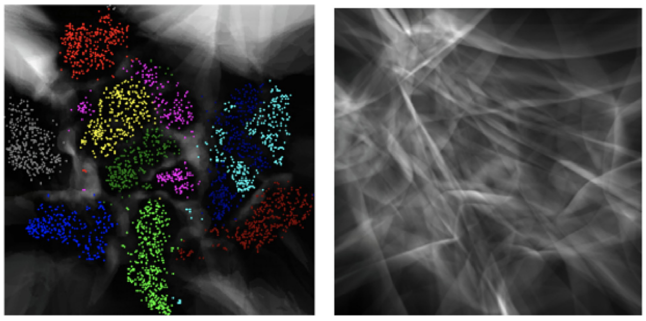

Fig. 3. Left: 2D projection for the MNIST dataset, points colored by class. Projection errors ϵ are encoded into brightness (dark=low, bright=high). Right: Inverse projection errors for the MNIST dataset (dark=low, bright=high).

We avoid such issues as follows. For a pixel $\mathbf{y} \in \mathbb{R}^2$, its corresponding data sample is $\mathbf{x} = P^{-1}(\mathbf{y})$. For both $\mathbf{x}$ and $\mathbf{y}$, we find their respective k-nearest neighbors in $\mathbb{R}^2$, respectively $\mathbb{R}^n$. With this, we can directly evaluate T and C and thus $\epsilon(\mathbf{y})$ at every pixel $\mathbf{y}$. Fig. 3(left) shows the combined projection error ϵ for the MNIST [55] dataset, with values mapped to brightness (high ϵ: bright; low ϵ: dark). As visible, projection errors are low close to most of the projection points. This is expected, as the projection technique used here (t-SNE) is known to have low errors everywhere on the MNIST dataset. As we go further from the projected points, we see how errors increase.

3.2. Inverse projection errors

Besides direct projection errors, also the inverse projection P^{-1} errors can adversely influence the insights users get from a DBM. Since all existing DBM methods use some P^{-1} method (Section 2), errors of P^{-1} will create errors in the DBM. We measure *locally* the quality of inverse projections using the *gradient map* technique in [25]. In detail, let $\mathbf{y}$ be a pixel in a DBM image, and $\mathbf{y}_r$ and $\mathbf{y}_b$ its right, respectively bottom, neighbors. The gradient map method computes the value

$$G(\mathbf{y}) = \sqrt{\|P^{-1}(\mathbf{y}_r) - P^{-1}(\mathbf{y})\|^2 + \|P^{-1}(\mathbf{y}_b) - P^{-1}(\mathbf{y})\|^2}, \tag{2}$$

which is the finite-difference approximation of the gradient norm of P^{-1} at $\mathbf{y}$. Interpreting G is simple: Low G values indicate low P^{-1} errors. Indeed, small changes in $\mathbf{y}$ correspond to small changes in the data space, which we expect from a well-behaved P^{-1}. High G values show errors in P^{-1} – small 'moves' in the image correspond to 'jumps' in the data space.[2] Fig. 3(right) shows the inverse projection error G for the same dataset and projection as in Fig. 3(left), encoded by brightness. We see that most image areas have low G values, except some thin 'bands' where G increases a lot.

3.3. VA tool for active learning

We next present the Visual Analytics (VA) tool that we constructed to assist users to interactively label samples to improve the training of a given classifier (Fig. 2, large gray box). Throughout the explanation of our tool's workflow, we refer to the steps (1)-(7) of the pipeline in Fig. 2.

Pre-labeling: Our tool starts with a fully labeled dataset D. If we have ground-truth for D, they can be directly used. If not, *i.e.*, when only a small fraction of D have supervised labels, we use DeepFA to

[2] More precisely, large values of the *gradient* of G, *i.e.* discontinuities in the G signal, show P^{-1} problems. In practice, we have seen that G consists of large low-G areas separated by thin 'bands' of high-G values. As such, we consider that high G values are a reliable indicator for the mentioned problems.

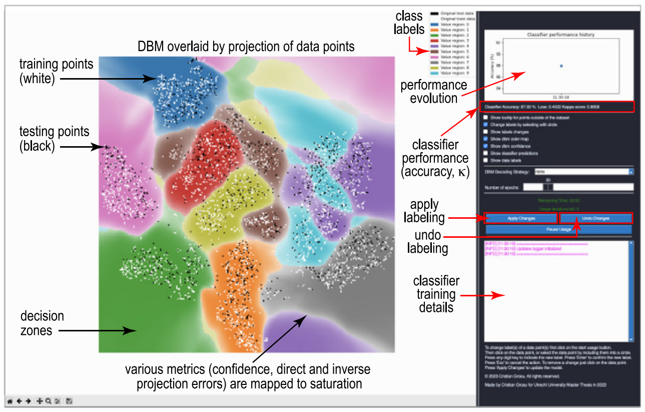
class labels
DBM overlaid by projection of data points
training points (white)
testing points (black)
performance evolution
classifier performance (accuracy, κ)
apply labeling
undo labeling
classifier training details
decision zones
various metrics (confidence, direct and inverse projection errors) are mapped to saturation

Fig. 4. Overview of the visual analytics tool's labeling window.

compute pseudolabels for the remaining points (1), using 1% of the supervised samples in D. This pre-labeling process is done only once, before our tool is started. After pre-labeling, the user uploads the pre-labeled dataset and the classifier to train and chooses general settings, *e.g.*, techniques to use for the DBM generation (explained next).

Interactive labeling: This is the main operation supported by our tool. It starts with displaying a DBM image for the classifier and labeled dataset provided at tool start-up. The DBM, (labeled) samples, P and P^{-1} error metrics, and tooltips showing details about specific samples, can be interactively explored in a *labeling window*.

Fig. 4 shows the labeling window. In the left part, the window shows the 2D projection of the loaded dataset (using the user-selected P method). Training and test points are shown in white, respectively black. The DBM for the classifier trained by the labeled set loaded by the tool (before any manual labeling), computed by a user-selected P^{-1} method, is also shown under the projection. We allow users to combine three additional metrics to show the DBM — the direct-projection error (Eq. (1)); the inverse projection error (Eq. (2)); and the confidence of the trained model [25,48]. Since all these measures share the same scale $[0, 1]$, we combine them by multiplication and map this combination to the DBM image's saturation. Fig. 4 (left) illustrates this by showing the classifier confidence mapped to saturation. We see that DBM areas close to decision boundaries have a lower classification confidence, as expected.

Hovering a pixel $\mathbf{y}$ in the DBM shows $P^{-1}(\mathbf{y})$ in a tooltip image. This allows users to inspect both actual samples (from the dataset D loaded by the tool) or inferred points corresponding to areas in the DBM outside $P(D)$. This helps next deciding how to manually label data samples. Top right in the window, we show the accuracy and Cohen's κ value for the *current* classifier, *i.e.*, trained with the current set of labeled points, including those the user has manually labeled so far. We also show the *evolution* of the classifier accuracy and κ over labeling iterations in a 2D line chart. This shows users whether their manual labeling actions have increased, or decreased, the classifier performance, and take corrective actions (more on this below).

Users can now start their first manual labeling iteration. Using the tooltip, currently-labeled samples in $P(D)$, DBM, confidence, and direct/inverse projection errors, they decide on a set of unlabeled points to which they want to assign one of the labels in C. This is not a deterministic process — if it were, we would not need the user's help but would automate labeling using *e.g.* techniques discussed earlier in Section 2. Rather, our claim is that, by studying all abovementioned information (scatterplot, DBM, confidence, errors), users can spot patterns in the data structure which help adding manual labels to increase the classifier's performance. We show this next in Sections 4 and 5.

At each iteration, users use a circle tool to select any set of points in the projection $P(D)$ to label (see Fig. 5a-c). When satisfied with the labeling, the user confirms this by an *apply labels* button. At this point, and at each iteration, the tool stores (i) the classifier accuracy and κ score, (ii) the labeled samples, and (iii) a screenshot of the labeling window. This data is used next to support undoing manual labeling changes and also to compute manual labeling performance, as discussed next.

Classifier re-training: After an iteration is completed, the classifier is re-trained to use the newly assigned pseudo labels, along the existing pseudo labels assigned by DeepFA. The training progress is shown in the tool (see Fig. 4) so users can spot possible problems. After re-training completes, the labeling window is updated to show the DBM of the newly-trained classifier. Note that position of points in the projection $P(D)$ does not change since users can only change *labels* of the data samples, but not the dataset D itself.

Fig. 5 shows the labeling window (d) before and (e) after a manual labeling iteration. Here, for illustration, the user manually selected a large set of points in the blue decision zone (marked by the black circle) and assigned them label 3 (red). Image (e) shows how the re-trained classifier has a large red decision zone that includes most points labeled by the user as red. Due to this 'brutal' re-labeling, the classifier's performance decreases significantly — accuracy drops from 0.8793 to 0.7927; κ drops from 0.8685 to 0.7689. This is, of course, expected, since the user has basically forced the disappearance of roughly the whole blue decision zone. In practice, manual labeling will select significantly fewer samples to label during an iteration.

If the classifier performance decreased (shown in the tool's interface) compared to the previous iteration, the user can decide to *undo* the last-performed labeling. The labeling window then changes to show the values (DBM, classifier performance, κ) before this iteration. The process continues until the user decides to stop it, either because of time constraints or because the desired classifier performance has been reached.

3.4. Implementation details

We next provide implementation details of our VA tool.

DeepFA: As explained, we create pseudo-labels for all the dataset samples, using only 1% of supervised samples and 5 DeepFA iterations. All other technical details concerning DeepFA follow the ones described in [9].

Direct projection methods: Our tool supports PCA [50], vanilla autoencoders (encoder part) [21], t-SNE [35], UMAP [51], and SSNP (encoder part) [26]. t-SNE and PCA use Scikit-learn [56]. UMAP uses the default implementation [57]. All parameters are set to their default values, except the perplexity of t-SNE, which we set to 30.

Inverse projection methods: Our tool supports vanilla autoencoders (decoder part), NNinv [25], and SSNP (decoder part). For NNinv, we use a fully connected neural network with architecture 2-32-64-128-512-α. We use $\alpha = 784$ for MNIST and $\alpha = 5000$ for P.cysts, respectively. Hidden layers use ReLU activation, except the last one which uses a sigmoid activation. The first layer uses an L_2 regularization penalty with constant set to 0.0002. Weights are initialized using the HeUniform kernel with bias set to 0.01. We train NNInv for 300 epochs (with early stopping) and mean squared error (MSE) as loss function. The decoder part of the autoencoder match the NNInv architecture and use the same activation functions and weight initializer. The encoder architecture is the decoder one, but flipped. We train our autoencoder for 300 epochs with MSE as loss function. SSNP uses an identical architecture to the autoencoder with the main difference of adding a data-dependent clustering layer — that is, its output has as many classes as the treated dataset has, *e.g.*, $C = 10$ for MNIST. The clustering layer uses the softmax activation function. SSNP uses two loss functions — MSE for data decoding (as the autoencoder) and sparse categorical cross-entropy loss (for data clustering). In the total loss, these two

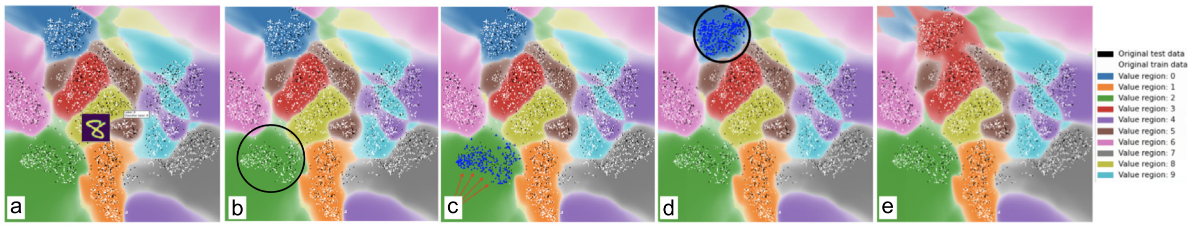

Fig. 5. Labeling procedure, MNIST dataset (a) Users can see the tooltip image for each point. (b) User selects a set of points in the green decision zone to label by drawing a circle with the mouse. Selected points (of any class) are colored blue by default to distinguish them from not selected ones. (c) Next, the user assigns label 2 (green) to these points. (d,e) Classifier retraining and DBM recalculation. (d) Initial state (classifier accuracy: 0.8793, κ: 0.8685). User selects a set of points in the blue decision zone and assigns them label 3 (red). (e) Situation after classifier retraining with the new manually added labels (classifier accuracy: 0.7927; κ: 0.7689).

Table 1
Combinations of direct P and inverse projection P^{-1} methods provided by our VA tool.

P	P^{-1}	Supported in our tool
t-SNE UMAP PCA	NNinv	yes
Autoencoder (encoder)	Autoencoder (decoder)	yes
	NNinv	no
SSNP (encoder)	SSNP (decoder)	yes
	NNinv	no

functions contribute with weights of 1, respectively 0.125. For further details, we refer to the original SSNP paper [26].

Direct and inverse projection combination: As outlined above, we have a total of 5 direct projections P (PCA, autoencoders, t-SNE, UMAP, SSNP) and 3 inverse projections P^{-1} (autoencoders, NNInv, SSNP), *i.e.* a total of 15 (P, P^{-1}) combinations. We deem some combinations to be less practical and/or useful than others. For instance, it has little sense to use SSNP as P^{-1} with a different P than the one SSNP provides, since SSNP *jointly* trains for P and P^{-1}. The same holds for autoencoders. Table 1 lists all (P, P^{-1}) that our VA tool supports.

DBM: We generate decision map using NNInv [47] and SSNP [26] at a resolution of 256 × 256 pixels.

Source code: Our VA tool, implemented in Python using *scikit.learn*, *TensorFlow*, and *Keras* is publicly available [58]. A short demo of the tool usage and 2d data generated in our experiments are also available [59].

4. User evaluation

We evaluated the efficiency and effectiveness of our VA tool for manual labeling by two user experiments. In these, we used our tool described as above, with two key constraints: (1) a labeling iteration can last max 3 min; (2) we allow a maximum of 5 iterations. Users can commit their manually assigned labels in an iteration at any time before 3 min. When this time elapses, no more manual labeling is allowed. We introduced these constraints to limit the total (and per-iteration) effort that users can put in manual labeling. While allowing a larger user effort could lead to better results, we argue that showing that our VA tool is effective under time constraints makes our claims of added value stronger — that is, we show that measurable improvement can be achieved by limited effort.

We next describe the classifier, datasets, and participants involved in our user evaluation.

4.1. Classifier

The classifier we use in our experiments is a neural network consisting of a flattening layer (having the input size), followed by a dense layer with a softmax activation function (with the unit count equal to the number of classes C in the considered dataset). We use 20 epochs for training and re-fitting. We chose, on purpose, a very *simple* classifier architecture, so as to offer the possibility for the classifier to generate sufficient errors, which next can be reduced by our VA-assisted labeling.

4.2. Datasets

Our evaluation comprises two experiments. In the first one, we use a ‘toy’ dataset and classification problem, to calibrate various settings of our tool. Next, we use a significantly more challenging, real-world, dataset to measure our tool’s effectiveness and efficiency. We next describe the two datasets.

4.2.1. Toy dataset: MNIST

We use a small subset of MNIST [55] ($N = 3500$, $C = 10$) to fine-tune our tool along two directions, as follows.

First, we note that the training-set size can affect the performance of a classifier, thus, also out tool’s perceived efficiency: A small training-set likely yields a poor classifier; then, our tool can *quickly* increase this poor performance; a large training-set will likely yield a high(er) classifier performance; further improving this performance by our VA tool is *potentially hard*. We analyze this effect of the training-set size using 5 such sets with 20%, 40%, 60%, 80%, and 100% of D’s samples (randomly selected), called next $D_{0.2}$, $D_{0.4}$, $D_{0.6}$, $D_{0.8}$, and $D_{1.0}$.

Secondly, our VA tool allows for five combinations of (P, P^{-1}) – namely (t-SNE, NNinv); (UMAP, NNinv); (PCA, NNinv); (autoencoder, autoencoder); and (SSNP, SSNP) (see Section 3.4, Table 1). For brevity, we next denote by (autoencoder), resp. (SSNP), the combinations where the same technique is used for both P and P^{-1}. To select the best combination of (P, P^{-1}) to use next with our tool, we next test our five training subsets D_i using all (P, P^{-1}) combinations.

4.2.2. Real-world dataset: P.cysts

We now use the best combination of (P, P^{-1}) found by the MNIST experiment to manually label a more challenging dataset. This dataset, called *P.cysts* [60], contains $N = 2696$ color microscopy images (200^2 pixels) of $C = 6$ species of human intestinal parasites in Brazil, responsible for public health problems. Classes and samples/class are given next: *E. coli* = 719, *E.histolytica* = 78, *E.nana* = 724, *Giardia* = 641, *I.butschlii* = 1501, and *B.hominis* = 189. *P.cysts*’ dimensionality is far higher than MNIST’s (200^2 *vs* 28^2 pixels), so we cannot directly feed *P.cysts* to a projection P and expect to obtain good results. Hence, we reduce *P.cysts* to a lower dimensionality $n = 512$ using a standard autoencoder approach, as done earlier in [18]. Next, we use this 512-dimensional dataset along the same manual labeling workflow as for MNIST.

4.2.3. Training, testing, and performance evaluation

In all our experiments, we split, randomly and in a stratified manner, the given input dataset D into D_{train} and D_{test} with a proportion of 80% to 20% respectively. This allows the computation of classification accuracy and κ. Importantly, we compute classification accuracy and κ over assigned pseudo-labels as if they were true labels — that is, we use no true labels in this computation.

Table 2
MNIST baseline. For different training sample counts D_i, we show DeepFA labeling performance (lower bound), classifier performance using pseudo labels, and classifier performance using true labels (upper bound).

	Metric	$D_{0.2}$	$D_{0.4}$	$D_{0.6}$	$D_{0.8}$	$D_{1.0}$
# of samples	$\lvert D\rvert$	700	1400	2100	2800	3500
labeling performance	acc	0.7757	0.7607	0.7723	0.7728	0.7737
classifier performance (pseudo labels)	acc	0.7507	0.7780	0.7987	0.7893	0.7540
	κ	0.7226	0.7531	0.7761	0.7658	0.7266
classifier performance (true labels)	acc	0.8347	0.8807	0.8960	0.9000	0.8947
	κ	0.8161	0.8673	0.8844	0.8888	0.8829

4.3. Participants

For both experiments, two users (denoted $U1$ and $U2$) are asked to perform manual labeling. $U1$ was closely involved in developing the VA tool, but has no prior knowledge on the real-world dataset *P.cysts* and its classification challenges. Conversely, $U2$ has detailed knowledge on *P.cysts* but was not involved in developing the VA tool. Both users have a good understanding of MNIST. This means that, for the first experiment (MNIST dataset), we can assume that $U1$ has some relative advantage over $U2$. For the second experiment (*P.cysts* dataset), we see no clear advantage of any of the users. Apart from the above, both users have quite similar profiles in terms of age and experience with machine learning and data visualization.

During both experiments, users were not able to exchange any information concerning their way of working and/or intermediate results, to avoid cross-learning or bias effects.

5. Experimental results

We next present the results of our two experiments.

5.1. Toy dataset: MNIST

5.1.1. Defining a performance baseline

We start by defining a *baseline* for our experiments. The classifier performance's *upper bound* for a given dataset is reached when training it using *all* true labels. Conversely, the performance *lower bound* is reached when training it using the DeepFA-generated pseudo labels, *i.e.*, those our tool takes as input (Fig. 2 (1)). This is the range we compare our VA tool's labeling with. We cannot expect that our tool can 'magically' perform some manual labeling that exceeds the upper bound performance; however, for our tool to be useful, it should yield a performance exceeding the lower bound as much as possible.

Table 2 shows the DeepFA labeling performance and classifier performance using DeepFA pseudo labels resp. all true labels, for all five subsets D_i of MNIST (see Section 4.2.1). We see that the labeling and classifier performance (with DeepFA pseudo labels) are around 0.77, largely independent of the used D_i subset. Also, we see that, when using true labels, accuracy and κ increase with the sample count.

5.1.2. Comparison among different techniques and users

Table 3 shows the classification accuracy and κ for pseudo labels generated by manual labeling using our VA tool, for both users, all five subsets D_i of MNIST considered, and all five combinations (P, P^{-1}) in Table 1, after five labeling iterations.

We see only small differences between $U1$ and $U2$ for the same dataset D_i and combination (P, P^{-1}) – both users yielded similar performance when given the same conditions. Hence, we next focus on the trends of performance *vs* dataset size $|D_i|$ and combination (P, P^{-1}). We notice that all projection techniques using only 20% of samples ($D_{0.2}$, 700 samples) achieve a similar (poor) result. This small sample count was not sufficient for the projections to provide a good visual representation for manual labeling. Separately, we see that not all (P, P^{-1}) combinations show a performance increase with the size of D_i. For instance, (PCA, NNinv) show poorer results in $D_{1.0}$ compared to $D_{0.2}$. In contrast, (t-SNE, NNinv) and (UMAP, NNinv) show a performance *increase* with the sample count. This suggests that these last two combinations are more suitable to generate visualizations for manual labeling.

Figs. 6 and 7 detail the performance values in Table 3 by showing accuracy and κ over all 5 labeling iterations. Blue curves show the classifier performance trained with user-assigned labels over iterations. Green, orange, and red curves show linear, logarithmic, and exponential trends, for interpretation ease: A blue curve close to the green line tells the user got a (roughly) *linear* increase of classifier performance over labeling iterations; a blue curve close to the orange curve shows that the user did better in early labeling iterations than in latter ones; a blue curve close to the red curve shows the user has difficulties in early labeling iterations but got better in latter ones.

Figs. 6 and 7 show similar trends for both users. While (tSNE, NNinv) and (UMAP, NNinv) show a more logarithmic trend – *i.e.*, user labeling was more effective in early iterations –, (autoencoder), (SSNP), and (PCA, NNinv) show an exponential trend – *i.e.*, the user performed better in latter iterations. This result may be caused by the visualization generated by the (P, P^{-1}) combination used. For the exponential trend case, the user might have done some significant wrong labeling in the first iterations, possibly because of the unintuitive DBMs generated by the (P, P^{-1}) combination used, but managed to correct this in latter ones. Importantly, since not *all* (P, P^{-1}) combinations show exponential trends, we cannot say that the key difficulty of users was in *learning* how to use the VA tool.

5.1.3. Added value of manual labeling

Let us analyze the added value of manual labeling compared to the baseline (Table 2). For this, we show in Table 4 the gain (difference) of classification performance obtained by manual labeling (Table 3) *vs* using DeepFA pseudo labels (Table 2). Positive values here tell that manual labeling exceeds the lower bound (DeepFA); negative values tell the opposite.

We see that (SSNP) and (PCA, NNInv) yielded the lowest gain for both users, while (autoencoder) yielded lower gains for $U1$. The highest gain was reached by (tSNE, NNinv) and (UMAP, NNinv) for both users, with (tSNE, NNinv) yielding 0.1 more in accuracy and κ as compared to (UMAP, NNInv).

We conclude that, for specific (P, P^{-1}) combinations, manual labeling can surpass the performance of DeepFA labeling, even when users are offered little time (15 min) to visually explore and label the data. The (P, P^{-1}) combinations can be roughly grouped in two classes — those which help manual labeling, *i.e.*, (tSNE, NNinv) and (UMAP, NNinv); and the remaining ones. Among the 'good' combinations, (tSNE, NNinv) consistently showed the best-added value for all considered dataset sizes and for both users.

5.2. Evaluation in a real-world problem: P.cysts

For the more complex *P.cysts* dataset, we only considered the (tSNE, NNinv) combination, which showed the best results when manual labeling the MNIST dataset (see Section 5.1.3).

5.2.1. Defining a performance baseline

As for MNIST, we first compute a baseline giving the lower and upper bound performances, reached by DeepFA labeling, respectively by using all true labels. In contrast to the MNIST experiments, we now use the full set of $|D| = 2696$ samples, due to the larger difficulty of the *P.cysts* dataset, and also given our earlier finding that too few samples can create problems for manual labeling (Section 5.1.2).

Table 5 shows this baseline and also the DeepFA labeling performance. We see that the difference between the upper and lower bounds is 0.0225 and 0.0334 for accuracy and κ, respectively. That is, DeepFA already yields a quite good result, leaving only a *small range* for manual labeling to improve.

Table 3
MNIST dataset. Results of classification accuracy and κ for classifiers trained with pseudo labels generated by manual labeling using our VA tool after 5 iterations, for two users, and different combinations of direct (P) and inverse (P^{-1}) projections.

P, P^{-1}	user	$D_{0.2}$		$D_{0.4}$		$D_{0.6}$		$D_{0.8}$		$D_{1.0}$	
		acc	κ	acc	κ	acc	κ	acc	κ	acc	κ
autoencoder	U1	0.8167	0.7961	0.8287	0.8094	0.8353	0.8169	0.8307	0.8117	0.8140	0.7932
	U2	0.8180	0.7976	0.7980	0.7754	0.8160	0.7954	0.8240	0.8043	0.8140	0.7932
SSNP	U1	0.8180	0.7976	0.8167	0.7961	0.8280	0.8087	0.8007	0.7784	0.7860	0.7621
	U2	0.7893	0.7657	0.7840	0.7598	0.7900	0.7664	0.7920	0.7687	0.7973	0.7746
t-SNE, NNinv	U1	0.8207	0.8005	0.8487	0.8317	0.8627	0.8473	0.8827	0.8695	0.8867	0.8740
	U2	0.8400	0.8221	0.8513	0.8348	0.8480	0.8310	0.8613	0.8458	0.8600	0.8443
UMAP, NNinv	U1	0.8167	0.7962	0.8727	0.8584	0.8667	0.8517	0.8887	0.8762	0.8633	0.8481
	U2	0.8333	0.8147	0.8460	0.8288	0.8260	0.8065	0.8473	0.8303	0.8487	0.8318
PCA, NNinv	U1	0.8127	0.7917	0.8120	0.7909	0.8260	0.8065	0.8127	0.7917	0.7620	0.7355
	U2	0.7673	0.7413	0.7720	0.7465	0.7867	0.7628	0.7673	0.7413	0.7333	0.7034

Table 4
MNIST dataset. Gain in classification performance obtained by manual labeling *vs* DeepFA pseudo labels, for two different users, different amounts of training samples D, and different combinations of direct (P) and inverse (P^{-1}) projections.

P, P^{-1}	user	$D_{0.2}$		$D_{0.4}$		$D_{0.6}$		$D_{0.8}$		$D_{1.0}$	
		acc	κ	acc	κ	acc	κ	acc	κ	acc	κ
autoencoder	U1	0.0660	0.0735	0.0507	0.0563	0.0366	0.0408	0.0414	0.0459	0.0600	0.0666
	U2	0.0640	0.0710	0.0440	0.0488	0.0620	0.0688	0.0700	0.0777	0.0600	0.0666
SSNP	U1	0.0673	0.0750	0.0387	0.0430	0.0293	0.0326	0.0144	0.0126	0.0320	0.0355
	U2	0.0386	0.0431	0.0060	0.0067	−0.0087	−0.0097	0.0027	0.0029	0.0433	0.0480
t-SNE, NNinv	U1	0.0700	0.0779	0.0707	0.0786	0.0640	0.0712	0.0934	0.1037	0.1327	0.1474
	U2	0.0893	0.0995	0.0733	0.0817	0.0493	0.0549	0.0720	0.0800	0.1060	0.1177
UMAP, NNinv	U1	0.0600	0.0736	0.0947	0.1053	0.0680	0.0756	0.0994	0.1104	0.1093	0.1215
	U2	0.0826	0.0921	0.0680	0.0757	0.0273	0.0304	0.0580	0.0645	0.0947	0.1052
PCA, NNinv	U1	0.0620	0.0691	0.0340	0.0378	0.0273	0.0304	0.0234	0.0259	0.0080	0.0089
	U2	0.0166	0.0187	−0.0060	−0.0066	−0.0120	−0.0133	−0.0220	−0.0245	−0.0207	−0.0232

Table 5
P.cysts baseline. Performance of DeepFA labeling, classification using the DeepFA pseudolabels (lower bound), and classification using all true labels (upper bound).

	Metric	Value
# of samples	$\|D\|$	2696
labeling performance	acc	0.8560
classifier performance (pseudo labels)	acc	0.8564
	κ	0.8043
classifier performance (true labels)	acc	0.8789
	κ	0.8377

Table 6
P.cysts dataset. Classification accuracy and κ for classifiers trained with pseudo labels generated by manual labeling using our VA tool after 5 iterations, two different users ($U1$, $U2$).

Iteration	acc		κ	
	U1	U2	U1	U2
0	0.8564		0.8043	
1	0.8538	0.8521	0.8019	0.7987
2	0.8581	0.8728	0.8096	0.8279
3	0.8590	0.8763	0.8107	0.8322
4	0.8616	0.8763	0.8131	0.8321
5	0.8676	0.8737	0.8211	0.8292

5.2.2. *Added value of manual labeling*

Table 6 shows the classification accuracy and κ values obtained after manual labeling by both users $U1$ and $U2$. Both users start from the same baseline value, *i.e.*, that offered by the DeepFA pseudolabeling. We see that both succeeded in using our VA tool to increase classification performance over the five available iterations – $U1$ increased accuracy by 0.0112 and κ by 0.0168; $U2$ increased accuracy by 0.0173 and κ by 0.0249. This shows that our VA tool offers added value even for more complex, and real-world, datasets.

Fig. 8 details the aggregated results in Table 6 over our 5 labeling iterations. We see trends similar to MNIST (Figs. 6, 7) – that is, exponential, respectively logarithmic, performance increase over iterations. These trends are now not related to different (P, P^{-1}) *combinations*, but to the two different *users*. For comparison ease, the last row of Fig. 8 shows the accuracy and κ trends for the two users ($U1$: blue; $U2$: orange) superimposed. The difference in these trends matches our knowledge about the users: $U2$ was familiar with the *P.cysts* dataset, which explains the quick gains obtained in early iterations and, likely that, at the end, $U2$ gets a slightly higher performance than $U1$. Results for $U1$ are more interesting: Even if this user had *zero* prior knowledge of the *P.cysts* dataset, $U1$ managed to get an almost as high performance as $U2$ with the same effort (time).

6. Discussion

We next discuss the key advantages and limitations of our proposed VA-based approach for manual labeling.

Genericity. Our approach can be applied to any dataset and classifier in a black-box manner — we require no details of the internal operation of the classifier. However, if the data dimensionality is too large – roughly, over a few thousand dimensions – directly using the tool's projection technique (t-SNE) to construct the DBM can be problematic. As the number of dimensions increases, so does the approximation of the dimensionality reduction technique. The amount of projection errors may also increase in this case. Our experiments include datasets with different numbers of dimensions to consider that. The point of our proposed solution is that approximation errors can be inspected by the user via the projection errors in our visualizations (Section 3.1). When these are too high, the dataset can be reduced to a smaller number of

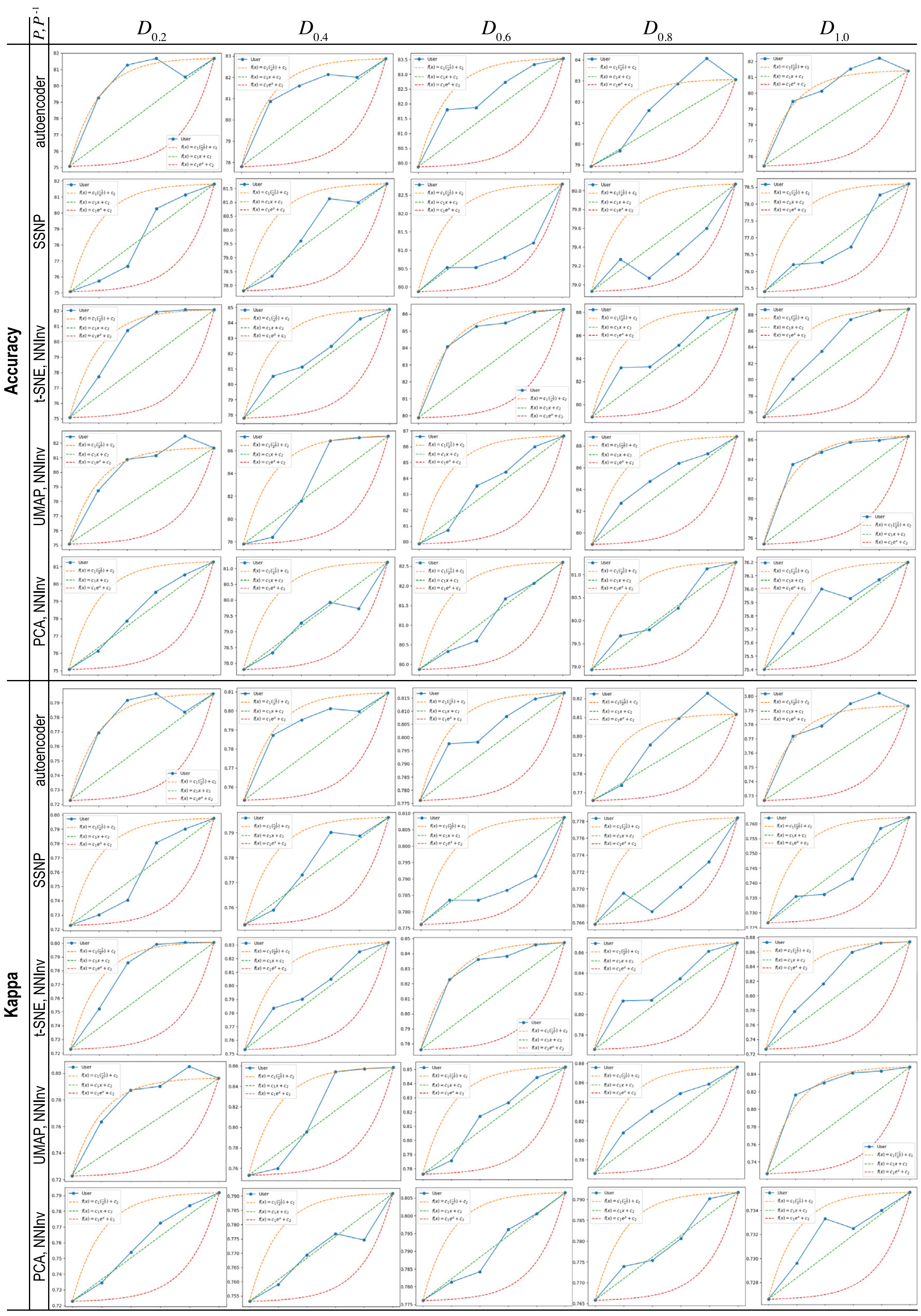

Fig. 6. Results of $U1$ for classification accuracy (top) and κ (bottom) over five iterations, using different direct and inverse projection techniques (P, P^{-1}) and input dataset fractions (D_i).

features using *e.g.* autoencoders prior to its use in the VA tool, as we did for *P.cysts* (see Section 4.2.2).

Effectiveness. While of limited extent (two datasets and two users only), we have shown evidence that our VA tool helps constructing classifiers with a higher performance than what can be obtained by automatic pseudolabeling algorithms such as DeepFA. This is, we believe, the strongest contribution of our work as it underlies that a *combination* of automatic techniques and human insight is optimal for ML

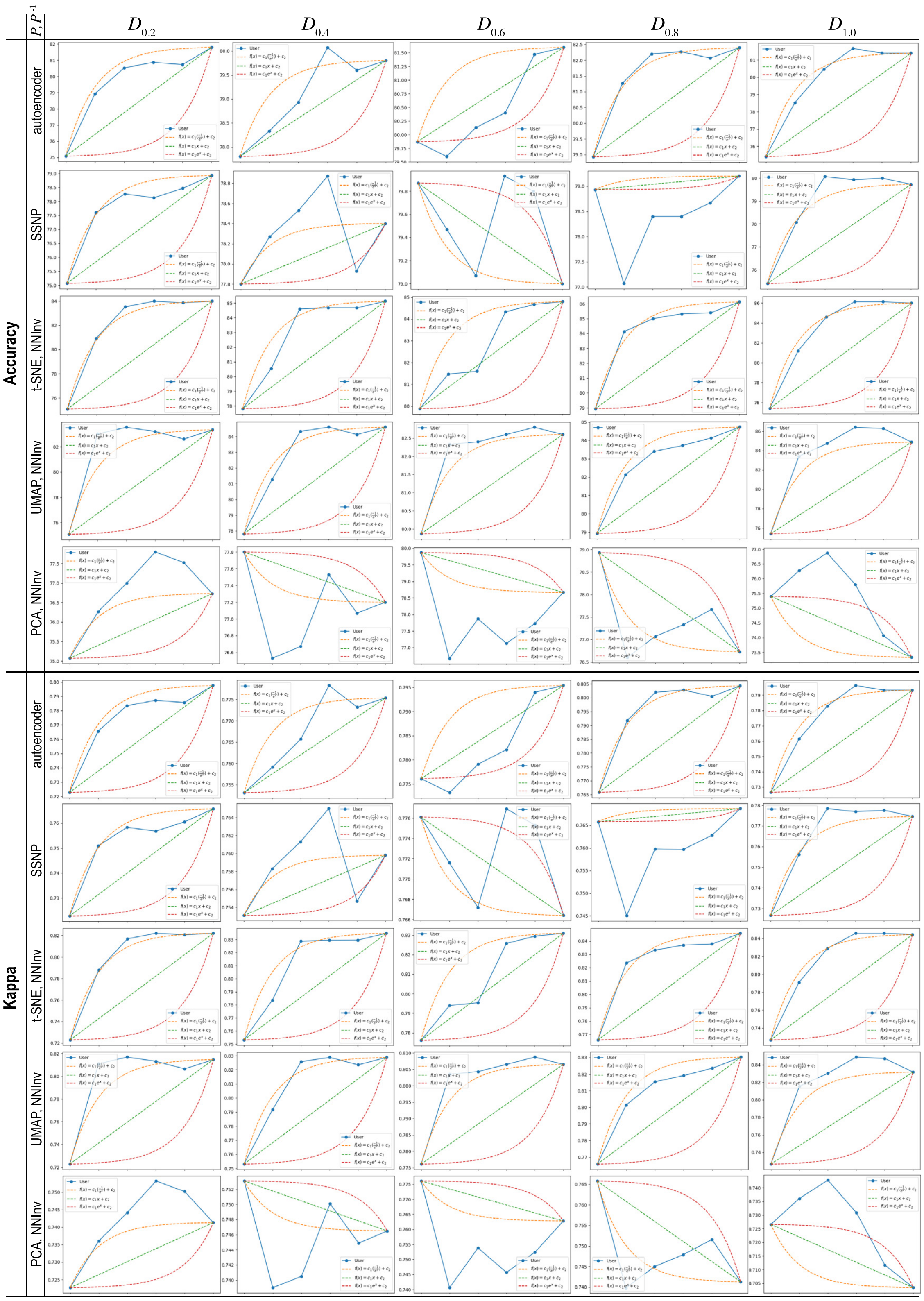

Fig. 7. Results of $U2$ for classification accuracy (top) and κ (bottom) over five iterations, using different direct and inverse projection techniques (P, P^{-1}) and input dataset fractions (D_i).

engineering problems. Yet, three main limitations exist here. First, our increase of effectiveness – the so-called *gain* that manual labeling offers *vs* automatic labeling – is quite small (a few percent points). Secondly, it is not clear how generalizable this gain is, *i.e.*, if it can be consistently observed for a wide spectrum of users, classifiers, and datasets. Thirdly, this gain may be reduced by the user errors, *i.e.*, the user may be prone to incorporate projection errors and assign wrong labels to samples when performing the task. Indeed, this is something we cannot avoid, mainly when dealing with solutions that involve human interaction and abstraction. Yet, we believe that our limited validation brings

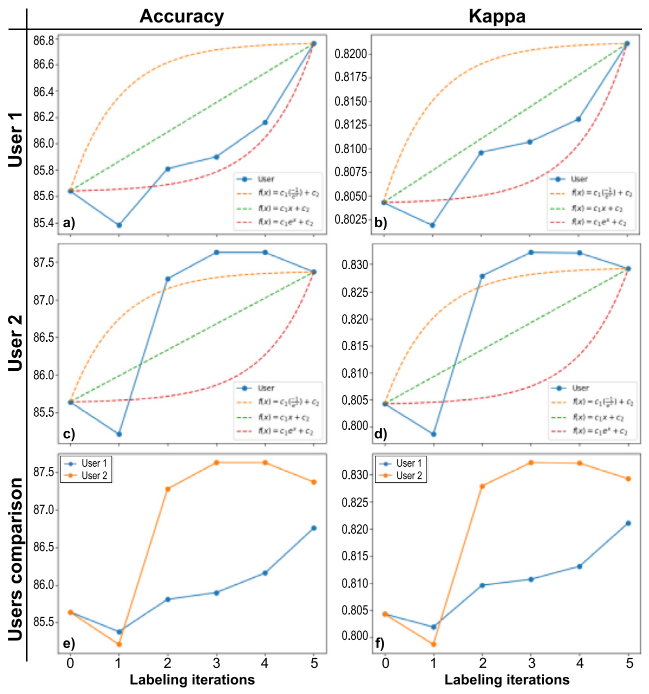

Fig. 8. *P.cysts* dataset. Comparing classification performance increase over five labeling iterations between two users.

sufficient evidence in support of our claims of usefulness of our VA tool for assisting with non-trivial classifier engineering: Our second dataset (*P.cysts*, Section 4.2.2) poses a very challenging classification problem due to the imbalance and similarities between impurities and parasite species — more so than when aiming to design classifiers for other real-world datasets where classes are better defined (no overlap between groups of different classes). Our experiments also bring evidence that even when prone to make mistakes and assign wrong labels, the user can *learn* during the process and still take advantage of the provided VA tools. Additionally, we used a *cross-validation scheme* with training and test sets for different sample counts in the dataset D: 20%, 40%, 60%, 80%, and 100%. By this, we consistently showed *classification improvements* for distinct subsets of our datasets. To strengthen these promising first results, we acknowledge that applying our VA tool and approach to additional datasets coming from different domains is needed.

Workflow. So far, our VA tool does not provide a specific way to 'instruct' users on how they can best do manual labeling, apart from the general interpretation of the visualizations described in Section 3 – *i.e.*, direct and inverse projection errors, confidence-annotated DBMs, and sample tooltips. Users learn to use our tool via trial-and-error, *i.e.*, select a few samples to label and monitor the change in DBMs and classifier performance that the new labels create. For this to work, we need a very *fast* execution of the label-retrain-visualize loop, so that one can do many such iterations quickly, including undo operations when negative effects are observed. In our current implementation, this loop takes a few up to ten seconds, depending on the used hardware (a mid-range PC) and dataset size. Recent DBM acceleration techniques [61] can reduce this time by one order of magnitude — an avenue we surely want to explore.

Besides a fast label-retrain-visualize loop, a second key potential improvement relates to *explaining* the DBM and its error maps. We have (anecdotally) noticed that users use the direct and inverse error maps mainly to ignore samples that fall in high-error areas. Still, completely skipping samples in such areas may miss high-potential labeling opportunities. We believe that additional interactive tools can help users to interpret error maps, thereby making users more comfortable with labeling additional samples, thus making our VA proposal more efficient and/or effective. Such tools can include *e.g.* tooltips showing which are the true nearest-neighbors, in data space, of a (set of) point(s) in the projection; or tooltips that show the actual distance to the closest decision boundary of every DBM point [47].

A final improvement point concerns the treatment of true labels. Currently, our VA tool does not distinguish between such labels and those pre-assigned by the DeepFA pseudo labeling step executed at the start of the VA workflow. As such, users may, during manual labeling, inadvertently change such true labels, thus unnecessarily decrease the performance of the training. A simple fix could prevent this issue from taking place.

Limitations. Earlier, we detailed specific limitations of our solution. We discussed the approximation errors of a projection technique (Genericity). We considered the restricted number of datasets and users, the gain in classification and labeling, and the human-added errors (Effectiveness). We discussed the trial-and-error way of labeling and the lack of a guideline to instruct users to perform the best manual labeling possible (Workflow).

Apart from them, we next consider some other limitations. Concerning the user experiments, we fixed the number of iterations and interaction time to measure and compare the labeling process properly. However, given that the user *learns* how to label while labeling, it would be interesting not to limit the number of iterations or interaction time and record user actions to be able to understand better the labeling process, challenges, and points to be improved. From that, we may provide a practical guideline to achieve the best labeling result possible. Additionally, we did not evaluate the impact of approximation errors in our approach. We argued that direct and inverse projection errors might impact visualization and, then, the user actions and labeling, but we did not actually *measure* how much these errors impact our results. We could compare our results with the user labeling in the projection error zones.

Another limitation is related to the application. We evaluated our approach when labeling samples for classification purposes. However, this approach can be extended to support labeling for segmentation purposes. Instead of using a dataset as input, we could use pixels as input and then label pixels instead of images. With that, the user can visualize the decision boundaries of a classifier when separating different pixels of a single image. It would be interesting to evaluate how the user can intervene or teach the classifier to learn what is relevant in a specific image by using pseudo labels.

7. Conclusion

In this work, we have shown how automatic and user-driven pseudo labeling methods can be combined in a single workflow to help constructing high-performance classifiers. Our proposed workflow starts by executing an existing pseudo labeling algorithm from a (very) small set of true labels to an entire training set. Next, users can examine this training set, together with the classification model that this set leads to (depicted as decision boundary maps), and manually label additional samples to improve classification performance.

We have shown two sets of experiments, on a simple dataset (MNIST) and a more complex one (*P.cysts*), both classified by a neural network. We have shown that using a specific combination of direct and inverse projection techniques to create DBMs (t-SNE and NNInv, respectively) users achieve an average increase in classification accuracy and κ of 8.21% and 0.00913 (MNIST), and 1.43% and 0.0209 (*P.cysts*), with a small effort — max 15 min of tool usage. We have also shown evidence telling that this gain does not depend on prior knowledge of the dataset and/or classification problem at hand, so it could be reached by a wide spectrum of users having general knowledge in machine learning and visual analytics. To our knowledge, this is the first study that explores decision boundary maps in a scenario aiming to improve classifier performance. As such, our work brings additional evidence to the value of DBMs.

Many future work avenues exist. Testing our VA approach with more users, datasets, and classifiers is needed to confirm (or refine) our findings and thus get a better idea of the added value it provides. Secondly, improvements in the depiction of the classifier behavior (via annotated decision maps), but also for the depiction of how the classifier actually uses the provided manual labels during training, can reduce the iterative effort needed to manually construct a good set of pseudo labels.

CRediT authorship contribution statement

Bárbara C. Benato: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Formal analysis. **Cristian Grosu:** Writing – original draft, Validation, Software, Formal analysis, Data curation. **Alexandre X. Falcão:** Writing – review & editing, Supervision, Resources, Funding acquisition. **Alexandru C. Telea:** Writing – review & editing, Supervision, Resources, Project administration, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my code in the paper (after acceptance to maintain anonymity). Data will be made available on request.

Acknowledgments

The authors acknowledge CAPES (# STIC-AMSUD 88887.878869/2023-00), CNPq (#304711/2023-3, #441950/2923-3), USP CEPID-CeMEAI (#2013/07375-0), FAPESP (#2023/14427-8, #2019/10705-8, #2022/12668-5).

References

[1] Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. Microsoft COCO: Common objects in context. In: Proc. ECCV. 2014, p. 740–55.
[2] Sun C, Shrivastava A, Singh S, Gupta A. Revisiting unreasonable effectiveness of data in deep learning era. In: Proc. ICCV. 2017, p. 843–52.
[3] Wang Y, Yao Q, Kwok JT, Ni LM. Generalizing from a few examples: A survey on few-shot learning. ACM Comput Surv 2020;53(3):1–34.
[4] Lee DH. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. In: Proc. ICML-WREPL. 2013.
[5] Iscen A, Tolias G, Avrithis Y, Chum O. Label propagation for deep semi-supervised learning. In: Proc. ICCV. 2019, p. 5070–9.
[6] Miyato T, Maeda Si, Koyama M, Ishii S. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. IEEE PAMI 2018;41(8):1979–93.
[7] Jing L, Tian Y. Self-supervised visual feature learning with deep neural networks: A survey. IEEE PAMI 2020;1.
[8] Pham H, Dai Z, Xie Q, Le QV. Meta pseudo labels. In: Proc. CVPR. 2021, p. 11557–68.
[9] Benato BC, Gomes JF, Telea AC, Falcão AX. Semi-supervised deep learning based on label propagation in a 2D embedded space. In: Proc. CIARP. Springer; 2021, p. 371–81.
[10] Benato BC, Telea AC, Falcão AX. Deep feature annotation by iterative meta-pseudo-labeling on 2D projections. Pattern Recognit 2023;141:109649.
[11] Zhang J, Wu X, Sheng VS. Learning from crowdsourced labeled data: a survey. Artif Intell Rev 2016;46:543–76.
[12] Settles B. Active learning literature survey. Computer sciences technical report 1648, University of Wisconsin–Madison; 2009.
[13] Iwata T, Houlsby N, Ghahramani Z. Active learning for interactive visualization. In: Carvalho CM, Ravikumar P, editors. Proceedings of the sixteenth international conference on artificial intelligence and statistics, vol. 31. 2013, p. 342–50.
[14] Bernard J, Hutter M, Zeppelzauer M, Fellner D, Sedlmair M. Comparing visual-interactive labeling with active learning: An experimental study. IEEE Trans Vis Comput Graphics 2018;24(1):298–308.
[15] Benato BC, Telea AC, Falcao AX. Iterative pseudo-labeling with deep feature annotation and confidence-based sampling. In: Proc. SIBGRAPI. IEEE; 2021, p. 192–8.
[16] Bernard J, Zeppelzauer M, Lehmann M, Müller M, Sedlmair M. Towards user-centered active learning algorithms. Comput Graph Forum 2018;37(3):121–32.
[17] Ren P, Xiao Y, Chang X, Huang PY, Li Z, Gupta BB, et al. A survey of deep active learning. ACM Comput Surv 2021;54(9).
[18] Benato BC, Telea AC, Falcão AX. Semi-supervised learning with interactive label propagation guided by feature space projections. In: Proc. SIBGRAPI. 2018, p. 392–9.
[19] Benato BC, Gomes JF, Telea AC, Falcão AX. Semi-automatic data annotation guided by feature space projection. Pattern Recognit 2021;109:107612.
[20] Rodrigues MFC, Hirata R, Telea A. Image-based visualization of classifier decision boundaries. In: Proc. SIBGRAPI. 2018, p. 353–60.
[21] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science 2006;313(5786):504–7.
[22] dos Santos Amorim EP, Brazil EV, Daniels J, Joia P, Nonato LG, Sousa MC. iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In: 2012 IEEE conference on visual analytics science and technology. 2012, p. 53–62.
[23] Joia P, Coimbra D, Cuminato JA, Paulovich FV, Nonato LG. Local affine multidimensional projection. In: Proc. IEEE TVCG. 2011, p. 2563–71.
[24] Espadoto M, Hirata N, Telea A. Deep learning multidimensional projections. Inf Vis 2020;9(3):247–69.
[25] Espadoto M, Appleby G, Suh A, Cashman D, Li M, Scheidegger C, et al. UnProjection: Leveraging inverse-projections for visual analytics of high-dimensional data. IEEE Trans Vis Comput Graphics 2023;29(2):1559–72.
[26] Espadoto. M, Hirata. NST, Telea AC. Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In: Proc. IVAPP. 2021, p. 27–37.
[27] Machado A, Telea A, Behrisch M. ShaRP: Shape-regularized multidimensional projections. In: Proc. EuroVA. 2023.
[28] Wang K, Zhang D, Li Y, Zhang R, Lin L. Cost-effective active learning for deep image classification. IEEE Trans Circuits Syst Video Technol 2017;27(12):2591–600.
[29] Yu F, Zhang Y, Song S, Seff A, Xiao J. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop, CoRR abs/1506.03365. 2015, URL http://arxiv.org/abs/1506.03365.
[30] Zhou S, Chen Q, Wang X. Active deep learning method for semi-supervised sentiment classification. Neurocomputing 2013;120:536–46.
[31] Gal Y, Islam R, Ghahramani Z. Deep Bayesian active learning with image data. In: Proc. ICML. 2017, p. 1183–92.
[32] Zou Q, Ni L, Zhang T, Wang Q. Deep learning based feature selection for remote sensing scene classification. IEEE Geosci Remote Sens Lett 2015;12(11):2321–5.
[33] Liu P, Zhang H, Eom KB. Active deep learning for classification of hyperspectral images. IEEE J Sel Top Appl Earth Obs Remote Sens 2017;10(2):712–24.
[34] Luus FPS, Khan N, Akhalwaya I. Active learning with TensorBoard projector, CoRR abs/1901.00675. 2019, URL http://arxiv.org/abs/1901.00675.
[35] van der Maaten L, Hinton G. Visualizing data using t-SNE. J Mach Learn Res 2008;9:2579–605.
[36] Arazo E, Ortego D, Albert P, O'Connor NE, McGuinness K. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In: Proc. IJCNN. IEEE; 2020, p. 1–8.
[37] Shi W, Gong Y, Ding C, Tao ZM, Zheng N. Transductive semi-supervised deep learning using min-max features. In: Proc. ECCV. 2018, p. 299–315.
[38] Zhai X, Oliver A, Kolesnikov A, Beyer L. S4l: Self-supervised semi-supervised learning. In: Proc. ICCV. 2019, p. 1476–85.
[39] Cascante-Bonilla P, Tan F, Qi Y, Ordonez V. Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning. 2020, arXiv preprint arXiv:2001.06001.
[40] Amorim W, Falcão A, Papa J, Carvalho M. Improving semi-supervised learning through optimum connectivity. Pattern Recognit 2016;60:72–85.
[41] Amorim W, Rosa G, Rogério, Castanho J, Dotto F, Rodrigues O, et al. Semi-supervised learning with connectivity-driven convolutional neural networks. Pattern Recognit 2019;128:16–22.
[42] Benato BC, Falcão AX, Telea A-C. Linking data separation, visual separation, and classifier performance using pseudo-labeling by contrastive learning. In: Proc. VISAPP. 2023.
[43] van der Maaten L. Accelerating t-SNE using tree-based algorithms. J Mach Learn Res 2014;15(1):3221–45.
[44] Schulz A, Gisbrecht A, Hammer B. Using discriminative dimensionality reduction to visualize classifiers. Neural Process Lett 2015;42:27–54.
[45] Schulz A, Hinder F, Hammer B. DeepView: Visualizing classification boundaries of deep neural networks as scatter plots using discriminative dimensionality reduction. In: Proc. IJCAI. 2020, p. 2305–11.
[46] Wang Y, Machado A, Telea A. Quantitative and qualitative comparison of decision map techniques for explaining classification models. Algorithms 2023;16(9).

[47] Rodrigues FCM, Espadoto M, Hirata Jr R, Telea A. Constructing and visualizing high-quality classifier decision boundary maps. Information 2019;10(9):280–97.
[48] Oliveira AAAM, Espadoto M, Hirata Jr R, Telea AC. SDBM: Supervised decision boundary maps for machine learning classifiers. In: Proc. IVAPP. 2022, p. 77–87.
[49] Rodrigues F. Visual analytics for machine learning: Computing and leveraging decision boundary maps [Ph.D. thesis], University of Groningen; 2020.
[50] Jolliffe IT. Principal component analysis. 2nd ed. Springer; 1986.
[51] McInnes L, Healy J, Melville J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. 2018, ArXiv e-prints.
[52] Venna J, Kaski S. Visualizing gene interaction graphs with local multidimensional scaling. In: Proc. ESANN, vol. 6. 2006, p. 557–62.
[53] Martins R, Coimbra D, Minghim R, Telea A. Visual analysis of dimensionality reduction quality for parameterized projections. Comput Graph 2014;41: 26–42.
[54] Martins R, Minghim R, Telea A. Explaining neighborhood preservation for multidimensional projections. In: Proc. CGVC. Eurographics; 2015.
[55] LeCun Y, Cortes C. MNIST handwritten digit database. 2010, http://yann.lecun.com/exdb/mnist.
[56] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12(Oct):2825–30.
[57] McInnes L, Healy J, Saul N, Grossberger L. UMAP: Uniform manifold approximation and projection. J Open Source Softw 2018;3(29):861.
[58] The Authors. Visual analysis tool for pseudolabeling – source code. 2024, https://pypi.org/project/decision-boundary-mapper.
[59] The Authors. Visual analysis tool for pseudolabeling – demo. 2024, https://github.com/barbarabenato/decision_boundary_mapper_demo.
[60] Suzuki C, Gomes J, Falcão A, Shimizu S, Papa J. Automated diagnosis of human intestinal parasites using optical microscopy images. In: Proc. symp. biomedical imaging. 2013, p. 460–3.
[61] Grosu C, Wang Y, Telea A. Computing fast and accurate decision boundary maps. In: Proc. EuroVA. Eurographics; 2024.