Image-based Visualization of Classifier Decision Boundaries

Francisco Caio M. Rodrigues*, R. Hirata Jr.*, A. C. Telea†
*Institue of Mathematics and Statistics, University of São Paulo
†Johann Bernoulli Institute, University of Groningen

***Abstract*—Understanding how a classifier partitions a high-dimensional input space and assigns labels to the parts is an important task in machine learning. Current methods for this task mainly use color-coded sample scatterplots, which do not explicitly show the actual decision boundaries or confusion zones. We propose an image-based technique to improve such visualizations. The method samples the 2D space of a dimensionality-reduction projection and color-code relevant classifier outputs, such as the majority class label, the confusion, and the sample density, to render a dense depiction of the high-dimensional decision boundaries. Our technique is simple to implement, handles any classifier, and has only two simple-to-control free parameters. We demonstrate our proposal on several real-world high-dimensional datasets, classifiers, and two different dimensionality reduction methods.**

I. Introduction

Machine learning (ML) methods are showing huge success when solving problems in many application areas, such as medical diagnosis [1], [2], recommendation systems [3], text classification [4], and image classification [5], [6]. However, many such methods work largely as "black boxes", so approaches that aim to shed insights into their operation are becoming increasingly important [7], [8].

One particular task in this context regards understanding how an ML classifier separates the input space in which its observations live into distinct regions which are next assigned different (discrete) labels. We consider here the case when this input space can be modeled by a (compact) subset $D \subset \mathbb{R}^n$ – that is, every observation $\mathbf{x}_i$ is described by n so-called features. Most ML classification techniques use such a representation [9]–[11]. Formally put, such a classifier can be described by a function $f : D \rightarrow C$, where C is a set of class labels. f is learned from a discrete training set $T \subset D$, and extrapolated to the entire input space D. A particularly important aspect of a classifier f is its so-called *decision boundaries*, *i.e.*, sets of points in D where f changes value. In other words, decision boundaries partition D into a set of areas $\cup_i D_i = D; D_i \cap D_j = \varnothing, \forall i \neq j$, so that f takes the same value over all points in a given D_i. Such decision boundaries are compact, typically continuous, subsets of D, due to the construction of the classifier function f.

Although very important for understanding a classifier, reconstructing decision boundaries for (visual) inspection is far from trivial [12], [13]. To *extract* the analytical descriptions of such boundaries, one would need to study the explicit representation of f, which is hard to reverse-engineer from the parameter values learned by a trained classifier. Depicting such boundaries is challenging, since in general they are complex manifolds embedded in $\mathbb{R}^n$. In general, such boundaries are implicitly shown by drawing 2D scatterplots of the training set with observations colored by assigned class labels, using dimensionality reduction methods [14]. However, since such sets are sparse (have relatively few observations), decision boundaries are only *implied* as the 2D areas where points with different labels meet. This leaves many open questions, the most important being what does a classifier precisely do for observations which fall *between* the labeled samples – that is, are not part of the training set.

In this paper, we propose to address the problem of extracting and visualizing decision boundaries of ML classifiers, as follows. First, we represent decision boundaries in a 2D (image) space, using the dimensionality reduction idea that underlies 2D scatterplot visualizations. However, in contrast to scatterplots, we propose a *dense* visualization, where each pixel encodes information relating to the n-dimensional subspace it maps, combining direct and inverse mappings (projections) from $\mathbb{R}^n$ to $\mathbb{R}^2$. Next, we adapt the sampling density of D so as to guarantee a user-specified degree of confidence per drawn pixel, which also handles the non-uniform density of the training set. Finally, we propose several visual encodings to depict sample density, classifier confusion, and actual labels assigned to decision areas D_i, and how these extrapolate the information learned from a training set T. Our method is generic, *i.e.*, can handle any feature-based classifier method f; does not depend on the dimensionality of the feature space; scales computationally well to large datasets; and has only two user-settable parameters with intuitive meanings.

The remaining of this paper is organized as follows. In Section I-A, we briefly review related work on dense maps in ML. Section II details the construction of our dense map visualization. Section III presents the results of our technique for real world datasets and classifiers. Section IV discusses our method. Section V outlines directions for future improvements.

A. Related Work

As outlined in Sec. I, the simplest and still most popular way to get insight in decision boundaries is to draw a 2D projection of a (discrete) training set $T \subset D$, where points are categorically colored to show class labels. Projections reduce the samples $\mathbf{x}_i \in D$ to points $\mathbf{y}_i \in \mathbb{R}^2$, $P(\mathbf{x}_i) = \mathbf{y}_i$, by applying dimensionality reduction, which in turn aims to preserve either distances [15]–[17] or neighborhoods [18]. A full discussion of projection methods is out of our scope here, and for this we refer to a recent survey [19].

While such scatterplots are simple and efficient to compute, they have a major drawback: depending on the size of T (sample count), how T is distributed over D, and how well the projection P preserves distances or neighborhoods, gaps will appear between the points of the resulting 2D scatterplot. One can only *guess* where the actual decision boundary passes through the labeled samples. Since such boundaries can, in general, have complex shapes, inferring them from the sparse sampling of a scatterplot is very challenging and error-prone.

To mitigate this general issue of scatterplots, the so-called image-based methods, or dense maps, have been proposed. The key idea is to color every pixel $\mathbf{y} \in \mathbb{R}^2$ of the target image to represent information pertaining to it in D. This approach is, for instance, used by the well-known TensorFlow toolkit [20]: Every pixel $\mathbf{y}$ of an image is color-coded to indicate the class label, and corresponding weight, that a neural network achieves for a sample $\mathbf{x}$. However, this works only because the input space (for the toy examples used in [20]) is two-dimensional, so the $\mathbb{R}^n$ to $\mathbb{R}^2$ mapping is trivial.

Image-based dense maps have been used for other tasks than decision boundary inspection in ML. For instance, Martins *et al.* propose several dense maps to encode the per-pixel errors created by dimensionality reduction methods [21], [22]. Similar per-pixel methods are used to encode the (categorical) identity of dimensions that make close points in a projection similar to each other [23]. Variants of this idea have been proposed to handle categorical data, using a Voronoi cell sampling of the image space, rather than a uniform pixel grid [24]. Key advantages of image-based methods are their ability to use every available pixel to show information, which increases the chance that complex data patterns are spotted without the need for the user to "guess" what happens between discrete samples; the lack of occlusion present in discrete methods; and the ability to handle large datasets by aggregating data over the available pixels. While such methods can handle arbitrary high-dimensional datasets, none of them was adapted to show actual classifier decision boundaries.

Closer to our scope, Hamel has proposed dense self-organizing image-based maps to visualize classifier decisions for high-dimensional feature spaces [12]. However, this method only handles support vector machine (SVM) classifiers; more importantly, the actual decision boundaries, while plotted on the respective maps, seem to be manually constructed by the user rather than by the method. Migut *et al.* actually construct and visualize such decision boundaries for high-dimensional data classification [13]. However, they use for this a sequence of 2D projections, each along a pair of dimensions; hence, the user has to mentally infer the actual position of the high-dimensional boundaries by interactively correlating multiple projections.

II. Dense Map Construction

We denote by I the resulting color image (dense map) generated by the method described in this section.

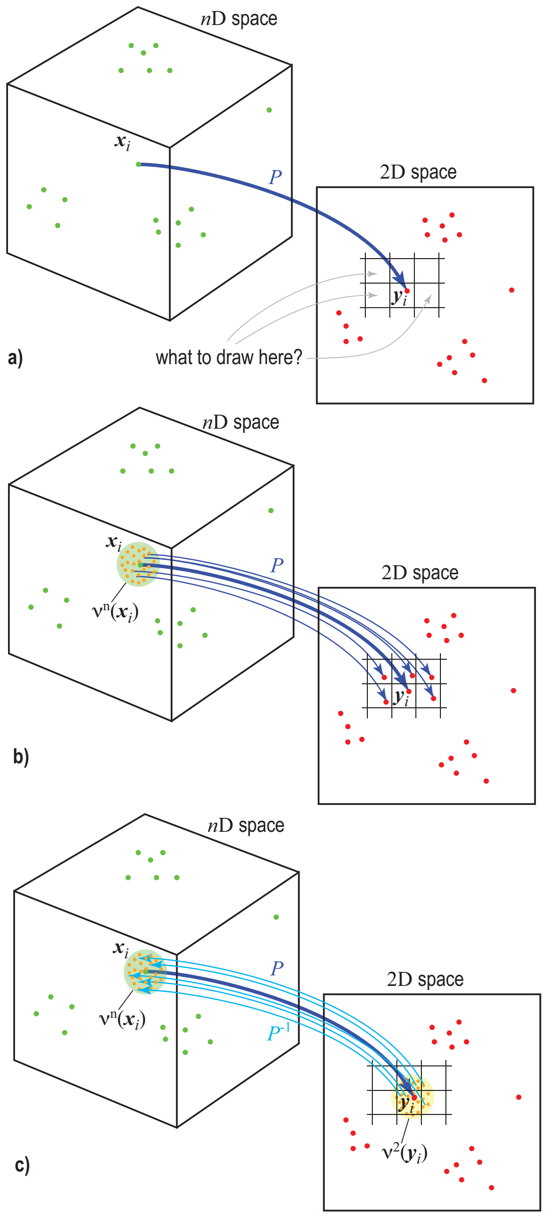

Fig. 1. Scatterplot problem (a). Dense map construction by scattering (b) and gathering (c) approaches.

Ideally, the image of D through P, denoted $P(D)$, would cover the entire I, thereby delivering (at least) one label $f(\mathbf{x})|\mathbf{x} \in D$ for any pixel $\mathbf{y} \in I$. However, training set $T \subset D$ only *sparsely* sample D. Visualizing $P(T)$ yields a typical class-label color-coded scatterplot rather than a dense map (Fig. 1a). To construct a dense map that covers all pixels of I, two approaches can be taken. First, we can create new observations $\mathbf{x}_j \in D$ by densely sampling the neighborhoods $\nu^n(\mathbf{x}_i)$ of all samples in T, computing their labels $f(\mathbf{x}_j)$, projecting $\mathbf{x}_j$ to 2D, and color-coding the pixels $\mathbf{y}_j = P(\mathbf{x}_j)$ by the labels $f(\mathbf{x}_j)$ (Fig. 1b). This method of *scattering* data from $\mathbb{R}^n$ to $\mathbb{R}^2$ does not guarantee that all pixels of I get covered, unless potentially very large neighborhoods ν^n are used, which is expensive. Alternatively, we can supersample I with $N \geq 1$ samples per pixel $\mathbf{y}$; compute the point $\mathbf{x} \in D$ that projects to $\mathbf{y}$, by using an (approximate) inverse projection function P^{-1}; and color $\mathbf{y}$ to summarize all labels of points that project onto that pixel (Fig. 1c). This *gathering* method guarantees that every pixel depicts information from at least N high-dimensional samples.

We choose for our goal the gathering strategy, as follows. Let N be the user-specified minimal number of samples per pixel desired. Larger N values increases the confidence of our visualization, as we have more information to decide the value of each pixel. Given a sparse labeled set of samples T, we first compute its scatterplot $P(T)$. This delivers $n(\mathbf{y})$ labels per pixel $\mathbf{y}$, where $n(\mathbf{y}) = 0$ for most pixels, given the above-mentioned sparsity. To ensure our target of N samples per pixel, we synthesize $\max(N - n(\mathbf{y}), 0)$ 2D points randomly spread over each pixel $\mathbf{y}$, and compute their $\mathbb{R}^n$ counterparts using P^{-1}. Pixels which are already densely covered by points in $P(T)$ need fewer additional samples, whereas pixels not at all covered by $P(T)$ receive N additional samples each.

At this point, every pixel is covered by at least N labels. We next encode the sample density, classifier confusion, and classifier decision at each pixel, as follows.

Decision: We define the decision d for a pixel $\mathbf{y}$ as the majority class label for all samples $\mathbf{y}_i$ in $\mathbf{y}$, *i.e.*

$$d(\mathbf{y}) = \text{argmax}_{k \in C} \sum_{\mathbf{y}_i \in \mathbf{y}} [f(P^{-1}(\mathbf{y}_i)) = k] \tag{1}$$

where [] denotes Iverson's bracket. We encode d in the hue $H(\mathbf{y})$ via a categorical color map, as follows. For each class label $k \in C$, we define a basic hue $H_T(k)$ and a slightly lighter version thereof $H_{synth}(k)$. If a pixel $\mathbf{y}$ has the majority label k, and there are points in the original input dataset T that project over $\mathbf{y}$, we set $H(\mathbf{y}) = H_T(k)$, otherwise, we set $H(\mathbf{y}) = H_{synth}(k)$. This way, we can distinguish between pixels covered by the scatterplot $P(T)$, which use $H_T(k)$, and pixels for which we needed to synthesize additional samples, which will use $H_{synth}(k)$.

Confusion: We define the confusion $c(\mathbf{y})$ for all samples of a pixel $\mathbf{y}$ as the ratio between the number of samples of the class label having most instances over $\mathbf{y}$ and the total sample count for that pixel, *i.e.*,

$$c(\mathbf{y}) = \frac{\max_{k \in C} \sum_{\mathbf{y}_i \in \mathbf{y}} [f(P^{-1}(\mathbf{y}_i)) = k]}{n(\mathbf{y})}. \tag{2}$$

Higher c values indicate more consensus for the samples over a pixel, whereas lower values indicate that the respective pixel is close to, or on, a decision boundary. We encode confusion into the saturation $S(\mathbf{y})$: Colorful pixels indicate areas where f chooses consistently a single label (depicted by hue, see above), whereas gray pixels indicate areas close to decision boundaries.

Density: We define the density of samples, ρ, for a pixel, $\mathbf{y}$, as the total number of samples covering this pixel's extent. These can be either samples in T or additionally generated samples created as described above. Visualizing ρ is useful as it tells us which dense-map areas have more information (samples), thus, we can be more confident about. Let ρ_{max} be the highest sample density over all pixels of I. We could directly encode ρ into the value (brightness) $V(\mathbf{y}) = \rho/\rho_{max}$. However, a problem of this design is that inherently darker hues, *e.g.* blue, will exhibit less brightness variation than brighter hues, *e.g.*, yellow, so density variations for the dark-hue labels will be hard to see. Hence, we choose to encode ρ in both brightness and saturation, as follows. First, we normalize ρ to $[0, 1]$ by computing

$$\overline{\rho} = \max\left(\frac{1}{20}\frac{\rho}{\rho_{avg}}, 1\right) \tag{3}$$

where ρ_{avg} is the average sample density over all pixels in I. Next, if $\overline{\rho} \in [0, 0.5]$, we compute V by linearly interpolating between a low brightness value $V_{min} = 0.1$ and the maximal $V = 1$. If $\overline{\rho} \in [0.5, 1]$, we set $V = 1$ and compute the saturation S by linearly interpolating between full saturation $S = 1$ and a low saturation $S_{min} = 0.2$. The net effect is that low densities will appear as darker hues; average densities will show the full brightness of the corresponding hue; and high densities will increasingly brighten the respective hue towards white. The values V_{min} and S_{min} are chosen so that we do not reach pure black or pure white, so the user does not confuse the emerging colors with those corresponding to maximal confusion values (grays). The decision zones of f will appear as "shaded cushions" whose domes indicate high density areas, akin to the results shown (by a different implementation and for a different goal) in [23].

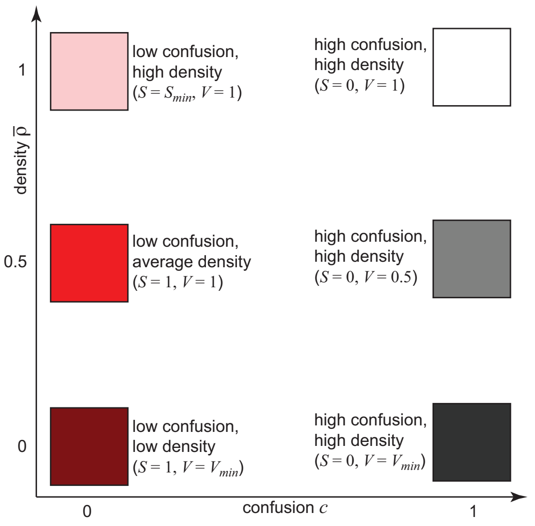
density $\overline{\rho}$
1
0.5
0
low confusion, high density ($S = S_{min}$, $V = 1$)
high confusion, high density ($S = 0$, $V = 1$)
low confusion, average density ($S = 1$, $V = 1$)
high confusion, high density ($S = 0$, $V = 0.5$)
low confusion, low density ($S = 1$, $V = V_{min}$)
high confusion, high density ($S = 0$, $V = V_{min}$)
0
confusion c
1

Fig. 2. Color scheme encoding decision, confusion and density values.

Figure 2 summarizes the HSV color synthesis proposed above to encode decision, confusion, and density for a class label mapped to the hue red (for illustration purposes). Along the y axis, we see how brightness increases to map higher density values. When the normalized density $\overline{\rho}$ is equal 0.5 and confusion is zero, we get a pure fully saturated red color. Lower density values map to darker reds, while higher densities map to increasingly whitish reds. Along the x axis, we see how saturation decreases to map increasing confusion values. When confusion is maximal, we only see gray tints.

A. *Parameter setting*

Our proposed dense map depends on two free user parameters: the resolution R of the target image I and the minimal desired number of samples per pixel N. We next explore the insights delivered by varying these parameters on a simple two-class dataset. This dataset is a subset of the well-known MNIST benchmark [25], created by keeping only the images of the digits 0 and 1 (for further details on MNIST, see Sec. III-B). For illustration purposes, we trained a Logistic Regression classifier (f) on this dataset, achieving a 99.8% accuracy. Any other classifier could be used – leading, of course, to different dense maps showing the behavior of that classifier. For projection, we used LAMP [15].

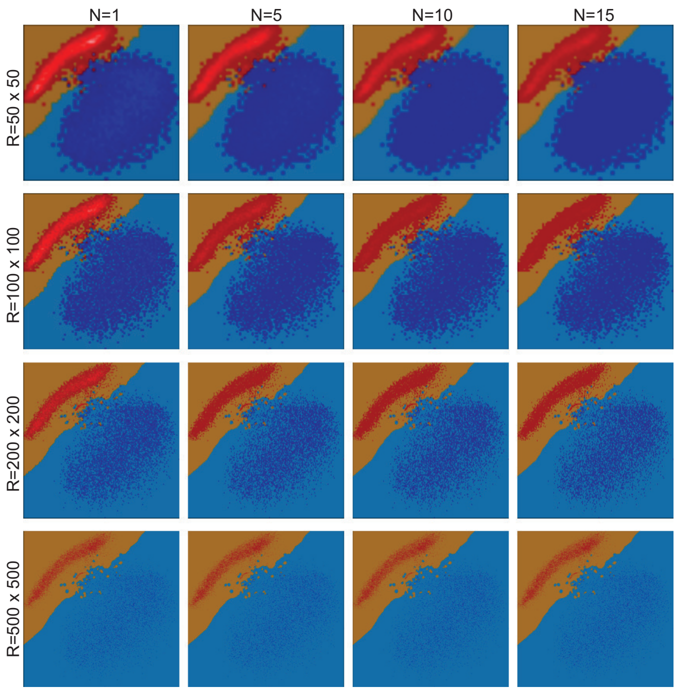

Fig. 3. The effect of varying both resolution R (rows) and number of minimum number of samples per pixel N (columns).

Figure 3 shows the impact of varying R and N on the dense map. Several observations follow, first and foremost, we see that the differences between dense maps for different parameter values are small and, more importantly, vary continuously with the parameters. This tells that our dense map construction is stable with respect to parameter choice, which is very important for its practical usability. Secondly, we see how the brightness bump, visible on the top-left image ($N = 1$, $R = 50 \times 50$ decreases as either R or N increase. This is expected, and interpreted as follows: for low N and R values, density variations of the raw *input* dataset D are visible, since there are no additional samples needed to construct the dense map. For our example, the red brightness bump tells that the "red" class samples are overall much denser than the "blue" class samples. Such images can be seen as a direct, depiction of D. As either N or R increases, the number $n(\mathbf{y})$ of samples in D per pixel $\mathbf{y}$ will decrease becoming eventually lower than N (the minimal number of desired samples per pixel), so we need to synthesize additional samples. When adding these extra samples, the overall spatial density of samples over the image becomes relatively constant, converging to N in the limit, so we see less brightness variation. We can interpret such high-N, high-R images as converging to the actual continuous decision boundaries in the limit. As R increases, we also see how the decision boundaries become more refined, showing more fine-scale details. Separately, the fact that we see few desaturated (gray) colors in the images tells us that the depicted classifier is quite consistent – that is, it assigns the same class to close samples.

To better understand confusion zones, Fig. 4 shows a zoomed-in view on the same dataset, but uses a simpler color coding than Fig. 3 – hue encodes the majority class label per pixel $d(\mathbf{y})$ (Eq. 1) and saturation encodes confusion $c(\mathbf{y})$ (Eq. 2). Hence, whitish pixels indicate zones where the classifier has a high confusion. As we increase the sampling density N, confusion bands appear more pronouncedly along the red-blue decision boundary, which is expected, since close to this boundary the classifier needs to change decisions. We also see small confusion areas *within* the compact blue zone, which indicate that this classifier has likely "drawn" the decision boundary in a too simple and inaccurate way.

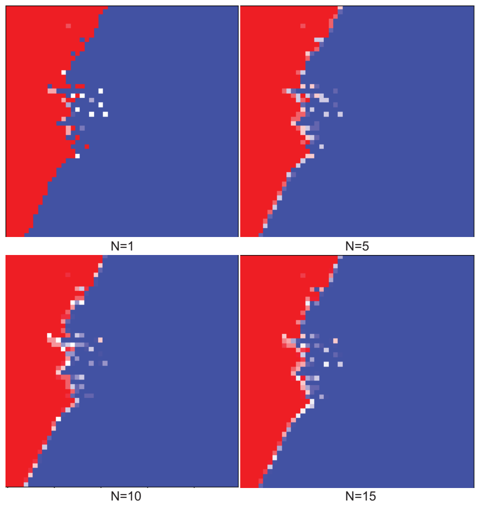

Fig. 4. Confusion zones (bright pixels) along the decision boundaries as function of minimum samples per pixel N.

B. *Implementation details*

We implemented our dense maps in Python using t-SNE [18] and LAMP [15] for the direct projection P and iLAMP [26] for the inverse projection P^{-1}. LAMP and iLAMP are simple to implement, and, as shown in several works, achieve higher accuracies in preserving distances than similar-type projection methods [15], [21]. In contrast, t-SNE has a better ability to separate high-dimensional clusters [18] than LAMP. LAMP and t-SNE are further compared in Sec. III-B.

III. Experiments and Results

We illustrate our technique by applying it to two high-dimensional datasets, four classifiers, and two dimensionality reduction techniques, as follows.

A. Segmentation dataset

The Image Segmentation Dataset [27] contains 2310 image instances with 19 features each, divided into 7 classes. Features measure image attributes, *e.g.*, color intensity mean, contrast, hue, and saturation. Classes relate to types of outdoor images, *i.e.*, brickface, sky, foliage, cement, window, path and grass. We trained three different classifiers, *i.e.* Logistic Regression (LR), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN), all implemented in *scikit-learn* [28], on this dataset, with the aim of comparing their decision boundaries. As parameters, we used the default ones in *scikit-learn*, except radial basis functions (SVM), and $k = 5$ nearest neighbors (KNN). Data was split into 70% training samples and 30% test samples. The obtained accuracies were 89% (LR), 87% (SVM), and 95% (KNN).

Figure 5 (top row) shows three LAMP projections for the three classifiers, each categorically colored by the training and test-set labels. This is a typical way that ML practitioners use to assess how the classifiers "divide" the data space into different zones for different classes. From these images, we only see small-scale differences between the three classifiers. Moreover, as already explained, such images are subject to occlusion and overplotting. Also, from these images, it is not clear what a classifier would decide for a sample which is relatively far away from existing ones. The dense maps, generated at a resolution $R = 500^2$ pixels, and with a minimum number of $N = 5$ samples per pixel, attempt to overcome these issues (Fig. 5, bottom row). They show us several insights. First, there is no overplotting in these images, so we are sure that each pixel carries the exact information pertaining to the samples that fall over it. Secondly, the differences between the decision zones corresponding to the seven classes are now much easier to see. For instance, for all classifiers, class 1 (orange) has a quite smooth and clear decision boundary touching mainly classes 3 (dark blue) and 5 (yellow-green). However, subtle differences between the classfiers also show up – for instance, the class-1 decision zone of LR contains a few isolated islands for class 2 (green) and class 3 (dark blue). These islands are different for SVM and KNN. Separately, we see that the decision boundaries for the other classes, most notably class 0 (blue) and class 5 (yellow-green) are much more jagged, for all classifiers. Verifying the actual classification results shows, indeed, that instances in these classes are harder to classify than in *e.g.* class 1 or class 6. Finally, we see some interesting differences between the dense map of KNN and the other two (Fig. 5, white stippled lines): the decision boundary of class 4 (purple) is visibly stretched upwards in zone A for KNN, whereas for LR and SVM, the purple zone is much smaller and does not protrude upwards through the class-0 (blue) area. Similarly, the decision boundary of the same class 4 protrudes significantly upwards in the green area in zone B for KNN, but not for the other two classifiers. Finally, the decision boundary for class-0 (blue) protrudes significantly downwards in the purple area for KNN, but not for the other two classifiers. Note that these differences cannot be explained by the projection, since we use the same projected points for all three classifiers. Overall, the decision boundaries of KNN show larger, and more mixed, per-class areas, except for class 1. This explains both the increase in accuracy of KNN *vs* the other two classifiers, but also for which regions (types of images) of the data space these differences occur. Note, again, that spotting such differences using only standard color-coded projection scatterplots is very hard, or even impossible.

B. MNIST dataset

Our second dataset, MNIST, is a standard dataset in ML consisting of 70K handwritten digit images, commonly employed to evaluate the performance of machine learning image classifiers, split into 60K training and 10K testing images [25]. Each 28×28 pixels grayscale image can be interpreted as a point in a 784-dimensional space. We use this dataset to explore two other questions, as follows.

First, we show that dense maps can also be used for deep learning classifiers, apart from the more classical ones such as LR, SVM, or KNN. For this, we built a simple Convolutional Neural Network (CNN) composed of two convolutional layers, one max-pooling layer and two densely connected layers. This CNN was implemented and trained using Keras [29], and obtained an accuracy of 99.2% on the test data after 14 training epochs. We next computed a 2D projection from a subset of 2000 samples of the training dataset using t-SNE (Figure 6). We did not use the full training dataset as t-SNE is quite slow (quadratic in the number of data points). If we had only this projection to analyse the decision boundaries assigned by the classifier, what would then be the conclusions to be drawn?

For instance, consider the top-right class-2 (green) and class-0 (blue) groups (marked A and B in Fig. 6). These appear equally well separated from the rest of the projection, equally compact, and have a similar (low) number of other-class points embedded in them. As such, with only the sparse projection information, we would likely conclude that the decision areas and boundaries for these two classes are quite similar. Likely, the user seeing this projection would draw the decision zones corresponding to A and B much like the dashed lines shown in Fig. 6. Let us look at the dense map for this dataset. Figure 7 shows it, at a resolution $R = 300^2$ pixels, computed for four different values of the per-pixel sample density N. We see now that the decision zones and boundaries of class-2 and class-0 are, in fact, much more complex than we could infer from the scatterplot. In particular, we see a non-negligible number of small "islands" corresponding to other labels than 0 and 2 embedded in the zones for these two labels. Also, we see

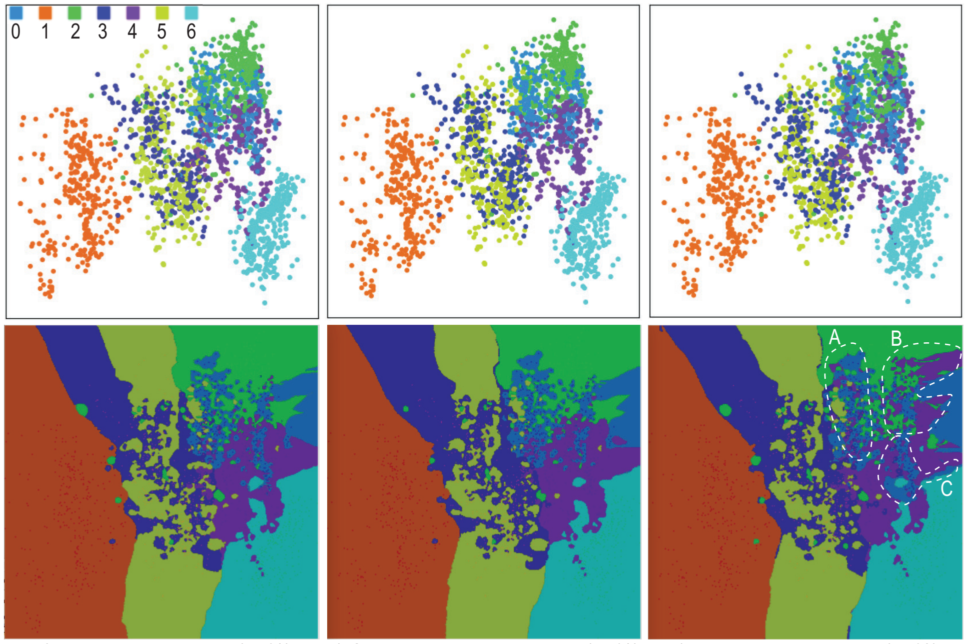

Fig. 5. Projections (top) and dense maps (bottom) for *segmentation* dataset, three classifiers.

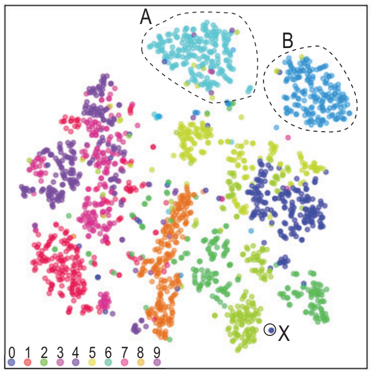

Fig. 6. MNIST projected using t-SNE, color coded by class labels.

that the class-0 zone is much more compact than class-2 – it contains a single small island for class 5 (Fig. 7, marker C).

Separately, let us consider the question of what happens close to outlier training samples. Take, for instance, the point marked X in the scatterplot (Fig. 6). What label would be assigned to a digit image that projects there? We see that we have an isolated class-3 outlier and the closest samples are the relatively large class-8 group (orange). So, based on the scatterplot, one would reckon that some class-3 decision boundary surrounding the outlier point will be created. However, what is the exact shape and size of this decision boundary? We cannot answer this question using only the scatterplot. The dense map gives us precisely this answer: the point X falls within an "island" decision zone that corresponds to class 3 (dark blue). This island is quite large, so it tells us that the impact of a *single* outlier in the training set is important in such sparsely-sampled areas. Note that this is not an approximate result: our method indeed synthesized a group of samples that project around (and on) the point X, ran it through the classifier, and obtained class 3 as a result. Moreover, we see that the dense maps are practically identical for different per-pixel sample densities, which increases the confidence that the class-3 island we see is indeed there. We could not have obtained this insight using the scatterplot only.

Another use of our dense maps is in showing how the decision boundaries *change* during training of a classifier. Figure 8 shows four such dense maps, for four different epochs (E) of training the CNN for the MNIST dataset, using stochastic gradient descent, with a learning rate of 0.001. For the first epoch (Fig. 8a), we see how the decision boundaries are highly jagged, while clear decision zones are mainly visible for the outlier samples. This confirms the insight that during a deep neural network training, outliers are handled the easiest, as surrounding them by decision boundaries is far easier than "drawing" such boundaries through a compact area of very similar samples [14]. From epoch 5, decision boundaries get significantly more refined in the central dense-sample region. The dense maps for epochs 10 and 15 clearly show how the training converges, as the decision boundaries basically stabilize. The dense maps correlate very well with the testing accuracies reported in Fig. 8. Such images generalize the simple 2D animations of 2D dataset classifications provided by TensorFlow [20] to nD datasets and arbitrarily complex networks. They help directly seeing when further training does not bring added value (in our case, from $E = 10$ onwards).

Finally, let us consider the projection algorithm choice. In Sec. II-A, we have shown that LAMP is a good choice for a simple two-class dataset. Above, we have shown that t-SNE works well for the high-dimensional MNIST dataset. We consider the same MNIST dataset (and classifier), but use

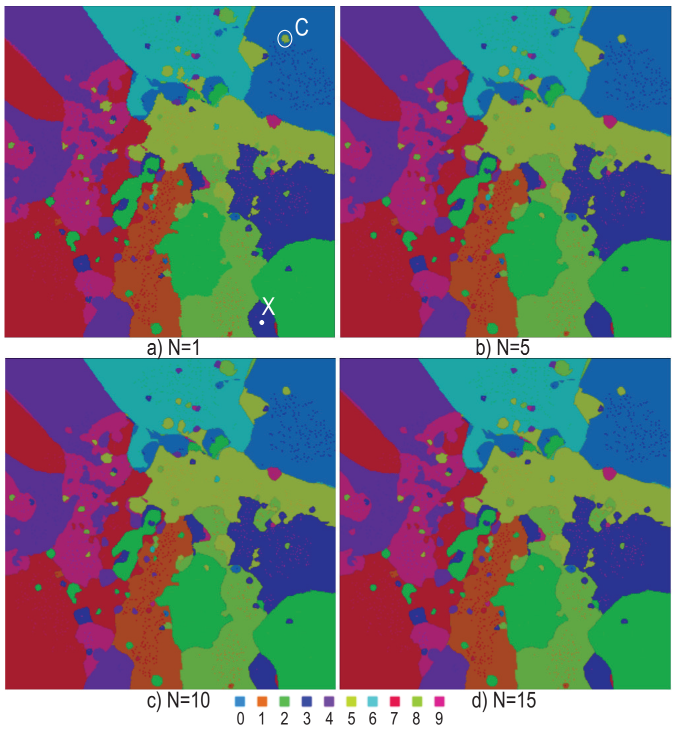

Fig. 7. dense maps for MNIST dataset classified by CNN, different sample density values N.

LAMP for the 2D projection instead of t-SNE. The resulting projection (Fig. 9a) shows clearly more class mixing than the t-SNE projection (Fig. 6). The explanation follows: t-SNE aims to preserve the high-dimensional nearest *neighbors* in the projection. Also, t-SNE pre-processes the data by PCA dimensionality reduction prior to projection, to make the 2D embedding task easier [18]. In contrast, LAMP aims to preserve high-dimensional Euclidean *distances* between points. So, for hundreds of dimensions (like the 784 ones in MNIST), LAMP yields far less cluster separation in the projection, even if the high-dimensional data is well separated. A poor-separation projection leads, next, to a dense map showing fragmented decision zones with complex borders (see Fig. 9b, which uses the same N and R values as Fig. 7d). So, we conclude that for low-dimensional datasets, LAMP and t-SNE are comparably good (with LAMP being significantly faster); for high-dimensional datasets, t-SNE should be definitely used instead of LAMP.

IV. Discussion

We discuss next several important aspects of our image-based visualization of classifier decision boundaries.

Genericity: Our dense maps work for any classifier and dataset, as long as the data can be represented by a feature vector. No specific constraints on data dimensionality, type, classifier internals, or classifier training, exist. For instance, we can visualize the decision boundaries of an insufficiently trained classifier and compare them with those of a better trained one, to tell us how training shifts these boundaries (see example in Sec. III-B).

Robustness: We have shown that our method is robust even for sparsely-sampled data spaces (limited number of training samples). Our dense map construction guarantees a user-specified number of labeled samples N per pixel, at a user-given resolution R. As N increases, every point of the 2D image becomes equally densely sampled, meaning that we also have the same confidence everywhere on our dense maps.

Projection: Our dense maps obviously depend on the quality, of the projection being used. A projection that, for instance, does not respect well high-dimensional distances or neighborhoods will also yield an artificially confused dense map. Yet, two key observations must be made. First, this (well known) limitation of projections applies equally to using scatterplots when visualizing a classifier's results, so our dense maps do not add any extra problem here. Secondly, the projection space should be seen as an *abstract*, and not Euclidean, space, in which decision zones are depicted. That is, *topological* tasks like finding the neighbors of a decision zone along its boundary, finding islands, and finding confusion zones, can be completed well even if a projection doesn't perfectly preserve point neighbors and/or inter-point distances. All our experiments showed that t-SNE is a good projection in this respect, in line with earlier findings in the same direction [14], [18], [19].

Limitations: Our current implementation cannot handle tens of thousands of points at interactive rates. The reason hereof is the already-mentioned high computational complexity of t-SNE. Yet, recent t-SNE accelerations could be used [30], [31], when such techniques become publicly available. Also, we need to compute the projection only *once* for a given dataset. Separately, we only tested our method with two projections (t-SNE and LAMP) and one inverse projection technique (iLAMP). Considering additional techniques is an important aspect for future investigations.

V. Conclusion

We have presented a technique to visualize decision boundaries of arbitrary machine learning classifiers. For this, we use an image-based approach where every pixel of the 2D output space is attributed a color to show the exact behavior of the classifier in the corresponding region of the high-dimensional space. For this, we use a combination of direct and inverse dimensionality-reduction methods, and we also propose several visual encodings of the classification result, confusion, and sample density. Our method is simple to implement and can handle any classifier and feature-based dataset with no changes. We demonstrate our method on several datasets, classifiers, and using two different projection techniques.

Several future work directions are possible. The dense maps can be augmented to show more information, such as explicit misclassification regions, and high-dimensional distances or neighborhoods. This would help understanding *why* a classifier

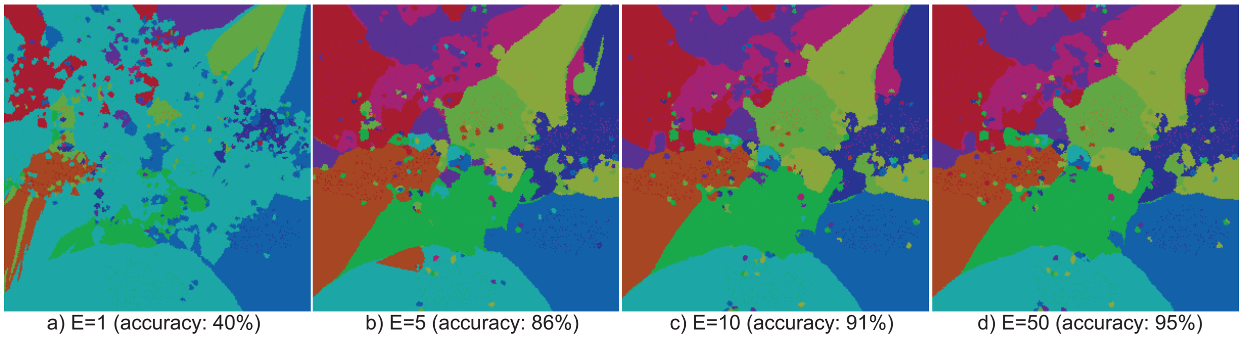

Fig. 8. dense maps for four training epochs E, CNN classifier, MNIST dataset.

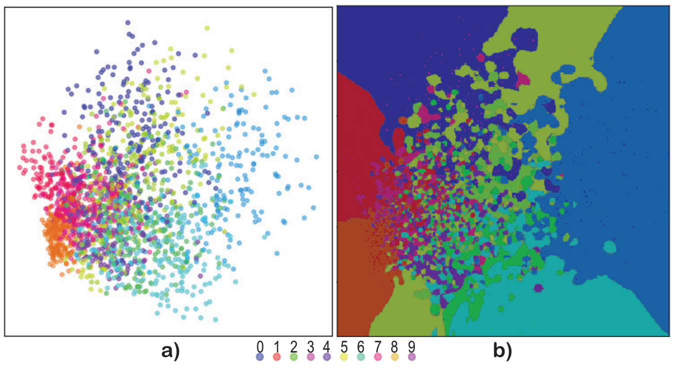

Fig. 9. LAMP projection and dense map for MNIST dataset, CNN classifier.

constructed its decision boundaries in a certain way and, thus, help in improving them. Separately, dense maps can be extended in an active learning approach to propose to the user areas where new labels would be needed to *e.g.* reduce confusion or increase classification accuracy.

Acknowledgment

Funding was provided by CAPES and FAPESP (2015/24485-9, 2015/01587-0).

References

[1] X. Hu, H. Cammann, H.-A. Meyer, K. Miller, K. Jung, and C. Stephan, "Artificial neural networks and prostate cancertools for diagnosis and management," *Nature Reviews Urology*, vol. 10, no. 3, pp. 174–182, 2013.

[2] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in Medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[3] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*. Springer, 2007, pp. 325–341.

[4] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE CVPR*, 2014, pp. 580–587.

[7] R. Féraud and F. Clérot, "A methodology to explain neural network classification," *Neural Networks*, vol. 15, no. 2, pp. 237–246, 2002.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proc. ACM SIGMOD KDD*, 2016, pp. 1135–1144.

[9] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[10] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[12] L. Hamel, "Visualization of support vector machines with unsupervised learning," in *Proc. Computational Intelligence and Bioinformatics and Computational Biology (CIBCB)*. IEEE, 2006.

[13] M. Migut, M. Worring, and C. J. Veenman, "Visualizing multi-dimensional decision boundaries in 2D," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 273–295, 2015.

[14] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE TVCG*, vol. 23, no. 1, pp. 101–110, 2017.

[15] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato, "Local affine multidimensional projection," *IEEE TVCG*, vol. 17, no. 12, pp. 2563–2571, 2011.

[16] M. Cox and T. Cox, "Multidimensional scaling," in *Handbook of Data Visualization*. Springer Handbooks Comp. Statistics, 2008.

[17] J. Tenenbaum, V. de Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[18] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J Mach Learn Res*, vol. 9, pp. 2579–2605, 2008.

[19] L. V. der Maaten, E. Postma, and H. V. den Herik, "Dimensionality reduction: A comparative review," *Technical Report TiCC TR 2009-005, Tilburg Univ., the Netherlands*, 2009.

[20] D. Smilkov and S. Carter, "TensorFlow playground," 2018, https://playground.tensorflow.org.

[21] R. Martins, D. Coimbra, R. Minghim, and A. Telea, "Visual analysis of dimensionality reduction quality for parameterized projections," *Computers & Graphics*, vol. 41, pp. 26–42, 2014.

[22] R. M. Martins, R. Minghim, and A. C. Telea, "Explaining neighborhood preservation for multidimensional projections," in *Proc. Computer Graphics and Visual Computing (CGVC)*. Eurographics, 2015.

[23] R. da Silva, P. Rauber, R. Martins, R. Minghim, and A. Telea, "Attribute-based visual explanation of multidimensional projections," in *Proc. EuroVA*. Eurographics, 2015, pp. 95–102.

[24] B. Broeksema, A. Telea, and T. Baudel, "Visual analysis of multi-dimensional categorical data sets," *Computer Graphics Forum*, vol. 32, no. 8, pp. 158–169, 2013.

[25] Y. LeCun and C. Cortes, "MNIST handwritten digits dataset," 2018, http://yann.lecun.com/exdb/mnist.

[26] E. P. dos Santos Amorim, E. V. Brazil, J. Daniels, P. Joia, L. G. Nonato, and M. C. Sousa, "iLAMP: Exploring high-dimensional spacing through backward multidimensional projection," in *Proc. IEEE VAST*, 2012, pp. 53–62.

[27] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J Mach Learn Res*, vol. 12, pp. 2825–2830, 2011.

[29] F. Chollet, "Keras machine learning framework," 2018, https://github.com/fchollet/keras.

[30] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova, "Approximated and user steerable t-SNE for progressive visual analytics," *IEEE TVCG*, vol. 23, no. 7, pp. 1739–1752, 2017.

[31] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova, "Hierarchical stochastic neighbor embedding," *Computer Graphics Forum*, vol. 35, no. 3, pp. 570–580, 2016.