6.838: Shape Analysis, 2021
Instructor: Justin Solomon
TAs: David Palmer and Paul Zhang

Low-Dimensional Visualizations of High-Dimensional Linear Functionals

David Anthony Bau

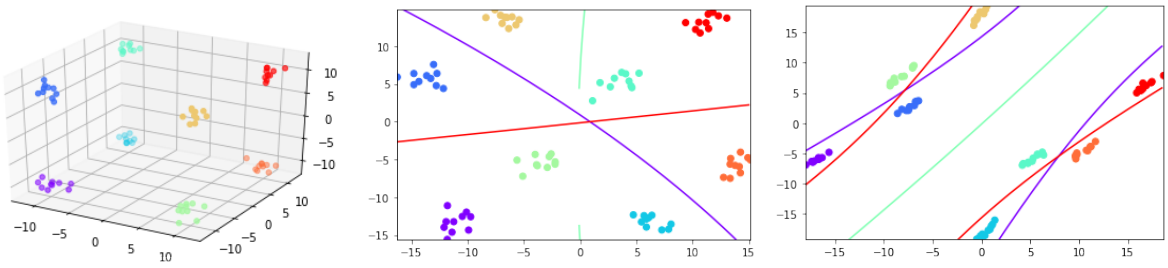

Figure 1: *Problem: embed the corners of the 3D cube (left) onto the 2D plane, such that you can draw conic sections on the 2D plane that have the same cluster separation properties as the xy, yz, and xz planes do in 3 dimensions? Traditional embedding methods (center) do not allow this by but we can modify them (right) to make it possible.*

Abstract

Dimensionality reduction techniques are used to create intuitive visualizations high-dimensional data. Frequently, one wants to also visualize decision boundaries for functions defined on the high-dimensional data. Classical techniques for decision boundary visualization treat the dimensionality-reducing embedding as fixed, and visualize the decision boundary afterward. In this paper, we gradient methods for doing dimensionality reductions that are aware of high-dimensional linear functions and arrange the embedding to make easy-to-understand low-dimensional visualizations of those functions.

1. Introduction

A *dimensionality reduction* technique takes as input a set of points $S \subseteq \mathbb{R}^n$ and outputs a mapping $f : S \to \mathbb{R}^m$ for some $m < n$. Typically, the set $\{f(s)\}$ approximates some properties of S, such as point-to-point distances. Dimensionality reduction techniques are frequently used with $m = 2$ to visualize the structure of high-dimensional data, such as images, word embedding vectors, or the hidden layers of a neural network.

In many cases, the visualized data will be the input for some high-dimensional function $g : \mathbb{R}^n \to T$ for some output set T, such as a classifier (with $T = \{0,1\}$) or the next layer of a neural network (with $T = \mathbb{R}^s$ for some s). In these cases one will also want a visualization of g that matches the visualization of the input S. This takes the form of some $\overline{g} : \mathbb{R}^m \to T$ such that $\overline{g}(f(s)) \approx g(s)$ for all $s \in S$.

Traditional techniques for this involve computing an inverse $f^{-1} : \mathbb{R}^m \to \mathbb{R}^n$ and defining the function $\overline{g}(x) = g(f^{-1}(x))$.

Techniques like this assume that f is computed before we even know about g. Sometimes, however, the chosen f can make it very difficult to compute a good or smooth g. In this project, we will explore methods for computing f and $\overline{g}$ at the same time, so that f can be chosen to make $\overline{g}$ smooth. Since the f are typically computed by minimizing some objective $E(f)$ function, we will do this by adding parameters for $\overline{g}$ and minimizing instead $\alpha E(f) + \beta \sum_{s \in S} \|\overline{g}(f(s)) - g(s)\|_2^2$ for some α, β.

To limit the scope of investigation, we will investigate just the cases where g is a linear functional. This is relevant to cases where the data is the input to a linear classifier, or to a single layer of a neural network. We will try to find an $\overline{g}$ that is a small-degree polynomial function – specifically, a quadratic one, making its decision boundaries conic. Our approach will be to use gradient descent with an additional error term representing the fidelity of quadratic representations of the high-dimensional linear functions.

2. Related Work

2.1. High-dimensional function visualization

Existing work on visualizing high-dimensionaly functions in low dimensions typically takes the form of *decision boundary* visu-

alization. This technique visualizes a high-dimensional function whose range is $\{-1,1\}$; but in theory the techniques could also be used on arbitrary functions. One example of a classical of doing this is described in [MRHT18], which depends on the local inversion method iLAMP described in [LT16]. This finds a dimensionality reduction by, for each point, applying a linear inverse of the whole embedding that is optimal locally around that point. Concretely, it computes $f^{-1}(p)$ as

$$M_p, t_p = \min_{M_p, t_p} \frac{1}{\|f(x)-p\|^2} \|M_p f(x) + t_p - x\|^2$$

$$f^{-1}(p) = M_p p + t_p$$

This finds for each p a approximate linear inverse to f that is optimal with respect to a weighted mean squared error weighted by squared embedded distance to p.

There are also more complex local inversion methods used, such as a neural one described in [REHT19]. As shown in [MRHT18] and [REHT19], these inversion methods frequently produce non-smooth regions and decision boundaries, and embeddings frequently have to be reseeded and restarted until the boundaries are roughly interpretable. In this project, we hope to avoid this restarting-and-selecting process by incorporating information about the desired visualized functions *during* the dimensionality reduction process.

An alternative visualization method, implemented in the Github repository [TMvK19], visualizes only the decision *boundary* and not the concrete value of the function. To do this, it selects *keypoints* exactly along the decision boundary (i.e. where $g(x) = 0$ if the sign of g is used as the classifier), and adds them to the pool of points to be dimensionality-reduced. This gives a scatter of points in the target space that represent to the decision boundary.

This method has the advantage of including information about the decision boundary in the dimensionality reduction procedure – since the keypoints are part of the dimensionality-reduced manifold, the dimensionality reduction will (to some degree) respect their relationship to other points. However, it will generally do so by respecting *distance to the decision boundary* rather than which side of the decision boundary to be on, which can lead to visualization errors. This can also only visualize the *boundary* of a decision function and not the *values* of a continuous map. In this project we investigate a way to incorporate knowledge about a function to visualize, while guaranteeing smoothness of the embedded function and retaining information about the function's concrete value.

2.2. Dimensionality reduction

The dimensionality reduction techniques we will modify are classical Multidimensional Scaling [Tor52] and t-SNE [vdMH08], because these both take the form of minimizing an densely-computed error function, making it easy to regularize the error to allow for a visualized functional. We will also compute exmaples on a heat kernel Laplacian method [BN03] for comparison.

Multidimensional Scaling minimizes the mean squared error between squared distance

$$\arg\min_y \sum_{i,j} \| \|y_i - y_j\|^2 - \|x_i - x - j\|^2 \|^2.$$

It can be shown that this is equivalent to minimizing the Frobenius norm of the difference between self-similarity matrices:

$$= \arg\min_Y \|YY^T - XX^T\|_F$$

and we will call this second error the "multidimensional scaling error". The multidimensional scaling error is convex and can be solved using spectral methods, but we will not do this since we need to add a new, non-convex term to it representing the fidelity of the visualiztation of a high-dimensional functional.

Meanwhile, t-SNE minimizes the KL divergence between two distributions over pairs of point indices. The first is the fixed Gaussian distribution P, where:

$$P(i,j) = \frac{e^{-\|x_i - x_j\|^2}}{\sum_{i \neq j} e^{-\|x_i - x_j\|^2}}$$

the second is the learned Student-T distribution Q, where:

$$Q(i,j) = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{i \neq j} (1 + \|y_i - y_j\|)^{-1}}$$

Finally, the heat-kernel Laplacian method we give for comparison is described in [BN03]. This doesn't straightforwardly take the form of optimizing an error, and instead a spectral method, given by first constructing a graph G from the union of nearest-neighbor clusters. The edges (i,j) of G are given weights proportional to a heat kernel $e^{-\frac{1}{t}\|x_i - x_j\|^2}$ and the spectrum of the graph Laplacian is used to create a low-dimensional graph embedding. We will use this as a third baseline to compare our modified methods to.

3. Technical Approach

3.1. Linear functionals to quadratic forms

Our investigation will be restricted to g that are linear, and $\overline{g}$ that are quadratic. That is, each g is:

$$g(x) = v_g \cdot x$$

for some vector g; and each $\overline{G}$ is:

$$\overline{g}(y) = \begin{bmatrix} y^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix}$$

for some 2×2 matrix A_g. This corresponds to translating hyperplane decision boundaries to conic sections. We measure the fidelity of the approximation using mean squared error:

$$E_{x,y}(\overline{g},g) = \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y_i \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2$$

Note that that the quadratic form $\begin{bmatrix} y^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix}$ is linear in A_g, which means that for a fixed y, solving for A_g amounts to solving the ordinary least squares problem:

$$A_{\overline{g}} = \arg\min_{A_g} \sum_i \|L_i(A_g) - v_g \cdot x_i\|^2$$

where L_i is the linear operator sending $A_g \mapsto \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix}$.

3.2. Fixed-embedding methods for comparison

As a baseline for comparison, we will evaluate the efficacy of a two-stage approach, where a dimensionality reduction method is used first and the optimal A_g is found afterward. We use TSNE, MDS, and the heat-kernel graph Laplacian method to find embeddings, and then compute their amenability to an $\overline{g}$ by measuring $\min_{\overline{g}} E_{x,y}(\overline{g},g)$.

3.3. Gradient descent

Our approach for designing an embedding method will be to take an existing embedding method that is based on optimizing an error function $E_x(y)$ (we will here examine MDS and TSNE), and using gradient descent to optimize a new $E'_x(y)$ that incorporates information about a potential A_g.

The simplest way to do this is to use the gradient methods to learn A_g and y simultaneously, differentiating:

$$E'_x(y,A_g) = \alpha E_x(y) + \beta \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y_i \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2$$

where $E(y,x)$ is the error from the original gradient method. The other possible method is to take advantage of the fact that, for fixed y, the optimal A_g can be found by ordinary least squares, and that there is therefore a differentiable closed form for A_g in terms of (x,y,z). We can thus write differentiable error function:

$$E'_x(y) = \alpha E_x(y) + \beta \min_{A_g} \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y_i \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2.$$

3.4. Modified MDS (MDS+)

Recall that multidimensional scaling minimizes error function

$$E_x(y) = \|yy^T - xx^T\|_F.$$

We can't immediately combine $E_x(y)$ with our new term $\min_{A_g} \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2$. This is because $E_x(y)$ is quadratic in the number of samples n, whereas our new term is linear in n; accordingly we take means over entries for both terms, making the MDS error

$$E_x(y) = \frac{1}{n^2} \|yy^T - xx^T\|_F$$

and out total error

$$E'_x(y) = \alpha \frac{1}{n^2} \|yy^T - xx^T\|_F + \frac{\beta}{n} \min_{A_g} \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2.$$

The two terms now have equal degree in both x and n, so should in convergence be comparable in magnitude.

To speed up optimization, we use the classical MDS embedding (equivalently, PCA) as the initial guess for Y.

3.5. Modified TSNE (TSNE+)

TSNE is already a gradient method, so we can add our error function to it without decreasing performance. We need to make our new error term comparable in magnitude to the existing error

$$\sum_{i,j} P(i,j) \log \frac{Q(i,j)}{P(i,j)}$$

This error takes the form of an expectation:

$$\mathbb{E}_P[\log \frac{Q(i,j)}{P(i,j)}]$$

Accordingly we expect this to be constant in the number of samples n. We thus add a multiple of our error that is constant in n:

$$E'_x(y) = \alpha \sum_{i,j} P(i,j) \log(\frac{Q(i,j)}{P(i,j)}) + \frac{\beta}{n} \min_{A_g} \sum_i \| \begin{bmatrix} y_i^T & 1 \end{bmatrix} A_g \begin{bmatrix} y \\ 1 \end{bmatrix} - v_g \cdot x_i \|^2.$$

Our implementation of this modified TSNE is based on the PyTorch implementation at the Github repository [mxl]. In keeping with ordinary TSNE, we use random initialization.

4. Results

4.1. Hypercube corners test

First, we test each method's ability to approximate the three dividing planes $x = \frac{1}{2}, y = \frac{1}{2}, z = \frac{1}{2}$ on an embedding of the eight corners of the cube $\{0,1\} \times \{0,1\} \times \{0,1\}$.

Accordingly, the three functions g_x, g_y, g_z are represented each by vector $[1,0,0]$, $[0,1,0]$, and $[0,0,1]$, respectively, and the data is generated by creating 8 clusters of size 100 each with variance $\frac{1}{10}$

Metric	MDS	Laplacian	MDS+	TSNE+
MDS (3-cube)	101.3	174.4	106.6	170.0
FEE (3-cube)	33.6	9.7	3.5	0.80
MDS (4-cube)	141.9	202.9	145.6	198.3
FEE (4-cube)	50.1	38.9	28.1	18.7
MDS (blob)	0.91	1.72	0.90	2.81
FEE (blob)	0.30	0.35	0.29	0.25
MDS (MNIST)	473146	2562071	469558	2567997
FEE (MNIST)	2630.62	3685.92	2581.56	1118.63

Table 1: *Average losses over 10 runs with different randomly-generated data for various embedding methods and various tests. MDS means "multidimensional scaling error"* $\frac{1}{n^2}\|yy^T - xx^T\|_F$. *FEE means "functional embedding error"* $\frac{1}{n}\sum_{i,j}\|\overline{g_j}(y_i) - g_j(x_i)\|^2$. *Note how MDS+ achieves much better embeddings of the linear functionals, without incurring too much of a cost to the MDS metric embedding loss.*

around each corner of the unit cube. Examples of results are shown in Figure 2, with aggregate results shown in Table 1.

We also run this test on the (significantly harder) task of embedding the corners of the 4-cube alongisde the 4 functionals represented by the four axes in 4-space. The results are also shown in Table 1; we can see that MDS+ still outperforms the classical MDS and static Laplacian methods.

4.2. Arbitrary orthogonal functionals on a Gaussian blob

Here, we test each method's ability to approximate an arbitrary linear functional that is unrelated to the structure of the underlying data. Here, we generate a single Gaussian cloud around the origin, and test each method's ability to (again) embed the three functionals represented by the vectors $[1,0,0]$, $[0,1,0]$, and $[0,0,1]$. We want an embedding that comes with three conics dividing the plane into the appropriate eight regions. In this case, there is no indication simply from the distance matrix that this should be done – so using the information in the linear functionals will be crucial.

Results shown in Table 1, with examples in Figure 3. We can see that MDS+ has trouble with this test, but TSNE+ can successfully partition the space into the required eight regions separated by three conics. Notice that TSNE+ does this at the expense of the metric structure of the data – for instance, there is no longer a well-defined "cluster center" that central points are closer to. MDS+ is presumably less willing to sacrifice this metric structure and so struggles to put the data into the correct partitions. All methods find this problem harder than the cube corners test.

4.3. MNIST test

Here, we train a classifier for MNIST digits logistic regression, and attempt to draw conic decision boundaries around a dimensionality-reduced MNIST using the parameters of this classifier.

We used the MNIST dataset available through OpenML (784-dimensional data with 70000 samples). We trained a logistic regression model $p(v) = \text{softmax}(Wv)$ using cross-entropy loss and

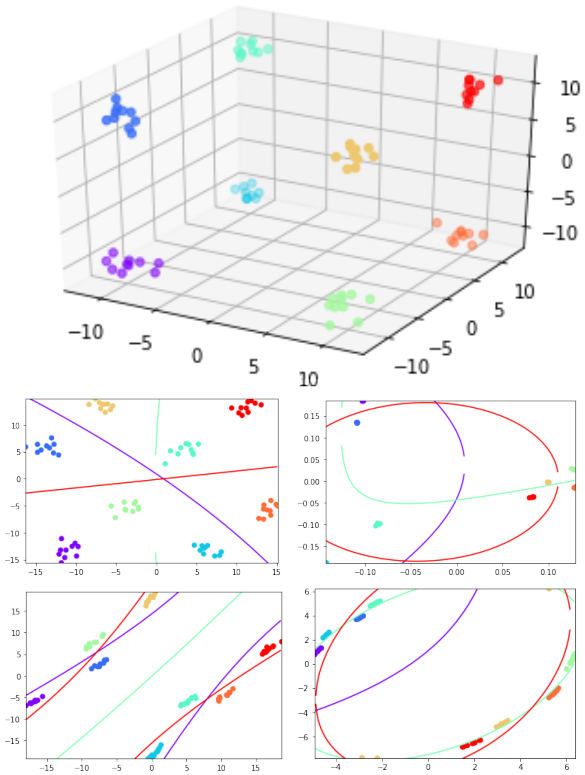

Figure 2: *The cube corners test. (Top) data in 3D. (Top left) best conic decision boundaries for fixed MDS embedding. (Top right) best conic decision boundaries for fixed Laplacian embedding. (Bottom left) solution generated by MDS+. (Bottom right) solution generated by TSNE+. Note how the static MDS embedding does not get some decisions right – e.g. the purple and green clusters should be separated by the purple boundary but are not.*

the Adam optimizer with learning rate 3e-4. The final training set error for the logistic regression was 0.94 (compared to 2.3 for a random classifier). It correctly classifies $62,706$ out of the $70,000$ samples for 89% accuracy.

The MNIST test then consists of taking a random 500-point subsample of the MNIST dataset, embedding it, and determining the fidelity of quadratic representations of the 10 linear functionals that make up the learned weight matrix W.

The results are shown in Table 1, with examples in Figure 4. We can see that MDS+ does not significantly improve the functional embedding error (it doesn't change much at all), but that TSNE+ does. However, as seen in Figure 4, this is done with significant cost to the metric structure of the data and does not yield better visualizations. Note that we used the same hyperparameters for the graph Laplacian here and its embedding was very bad in both metrics – better hyperparameters might have yielded a better embedding.

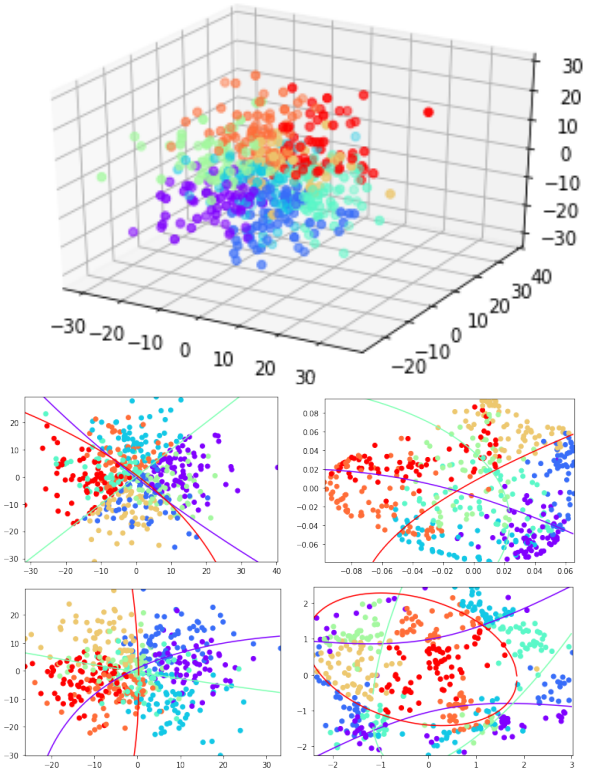

Figure 3: *The blob test. (Top) data in 3D. (Top left) best conic decision boundaries for fixed MDS embedding. (Top right) best conic decision boundaries for fixed Laplacian embedding. (Bottom left) solution generated by MDS+. (Bottom right) solution generated by TSNE+. Note how TSNE+ is almost able to give each color its own section – it does this by sacrificing the idea of a "cluster center" that points close to zero must be close to. This corresponds to its high MDS loss but low FEE loss in Table 1.*

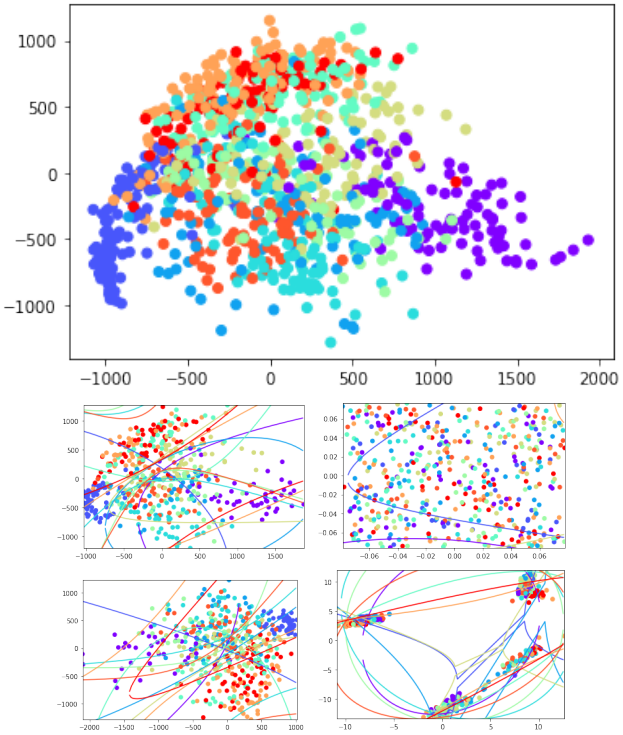

Figure 4: *The MNIST test. Each color represents a different number. Again, (top left) MDS, (top right) graph Laplacian, (bottom left) MDS+, (bottom right) TSNE+.*

5. Conclusions and future work

We have presented two possible methods for doing dimensionality reduction in a way that respects a high-dimensional function. These methods take advantage of the fact that the optimal quadratic $\overline{g}$ to represent a given high-dimensional function g can be computed in a closed, differentiable way. This allows us to design gradient methods that optimize for the fidelity of the optimal quadratic.

We have shown that these methods can successfully embed the corners of a hypercube in such a way that the axis planes can be represented by conics. We have furthermore explored their properties when applied to a Gaussian blob and the MNIST dataset, finding that TSNE+ can be effective at partitioning a Gaussian blob even though MDS+ is not; and that none of the methods substantially improve the MNIST visualization.

We note that any of these methods can be trivially applied to high-dimensional functions that are *not* linear, and to any finite-dimensional vector space of functions in the low-dimensional space (so long as we have a finite basis for the space of functions, the problem of finding the optimal function will still be OLS). More work might be done to see if other high-dimensional functions (such as entire neural networks), and low-dimensional parameterizations lend themselves to this kind of visualization.

It is also possible that the simple form of the FEE error might be taken advantage of to find some faster optimization method for MDS+ rather than gradient descent.

Any of these methods might also be applied to target embedding dimensions that are not 2. This might give rise to e.g. embeddings into 3 dimensinos with 3-dimensional conic sections (hyperboloids, ellipsoids, etc). as decisions boundaries.

Any visualizations of this kind, when faithful, could help to diagnose misclassifications and help machine-learning model designers see the structure of their models.

References

[BN03] BELKIN M., NIYOGI P.: Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation 15*, 6 (06 2003), 1373–1396. URL: https://doi.org/10.1162/089976603321780317, arXiv:https://direct.mit.edu/neco/article-pdf/15/6/1373/815527/089976603321780317.pdf, doi:10.1162/089976603321780317. 2

[LT16] LEHMANN D. J., THEISEL H.: General projective maps for multidimensional data projection. In *Proceedings of the 37th Annual Confer-*

ence of the European Association for Computer Graphics (Goslar, DEU, 2016), EG '16, Eurographics Association, p. 443–453. 2

[MRHT18] M. RODRIGUES F. C., HIRATA R., TELEA A. C.: Image-based visualization of classifier decision boundaries. In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)* (2018), pp. 353–360. doi:10.1109/SIBGRAPI.2018.00052. 2

[mxl] MXL1990:. https://github.com/mxl1990/tsne-pytorch. 3

[REHT19] RODRIGUES F. C. M., ESPADOTO M., HIRATA R., TELEA A. C.: Constructing and visualizing high-quality classifier decision boundary maps. *Information 10*, 9 (2019). URL: https://www.mdpi.com/2078-2489/10/9/280, doi:10.3390/info10090280. 2

[TMvK19] TAMAS MADL R. D., VAN KOOTEN P.: Plotting high-dimensional decision boundaries. https://github.com/tmadl/highdimensional-decision-boundary-plot, 2019. 2

[Tor52] TORGERSON W. S.: Multidimensional scaling: I. theory and method. *Psychometrika 17*, 4 (Dec 1952), 401–419. URL: https://doi.org/10.1007/BF02288916, doi:10.1007/BF02288916. 2

[vdMH08] VAN DER MAATEN L., HINTON G.: Visualizing data using t-sne. *Journal of Machine Learning Research 9*, 86 (2008), 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html. 2