Learning Interpretable, Tree-Based Projection Mappings for Nonlinear Embeddings

Arman Zharmagambetov **Miguel Á. Carreira-Perpiñán**
Department of Computer Science and Engineering, University of California, Merced

Abstract

Model interpretability is a topic of renewed interest given today's widespread practical use of machine learning, and the need to trust or understand automated predictions. We consider the problem of optimally learning interpretable out-of-sample mappings for nonlinear embedding methods such as t-SNE. We argue for the use of sparse oblique decision trees because they strike a good tradeoff between accuracy and interpretability which can be controlled via a hyperparameter, thus allowing one to achieve a model with a desired explanatory complexity. The resulting optimization problem is difficult because decision trees are not differentiable. By using an equivalent formulation of the problem, we give an algorithm that can learn such a tree for any given nonlinear embedding objective. We illustrate experimentally how the resulting trees provide insights into the data beyond what a simple 2D visualization of the embedding does.

While predictive accuracy is usually the primary reason to learn a machine learning (ML) model, the ability to interpret the resulting model has always been important, and work in both statistics and computer science shows this. Early examples are the use of varimax rotation to facilitate the interpretation of factor analysis (Kaiser, 1958), or the use of expert systems in medical diagnosis (Neches et al., 1985). However, it is in the last few years, due to the widespread adoption of ML models in many practical applications, that model interpretability has become paramount. Indeed, new regulations have been proposed or are already in effect that require, in one form or another, that ML model decisions be explainable to end users. These include regulations by the EU (GDPR and AI Act), US Federal Trade Commission (FTC), etc.[1] This implies that companies that use AI should be prepared to handle audits of their systems[2]. In fact, some sensitive uses of ML models have long been regulated. An example is credit scoring. The FTC's 1970 Fair Credit Reporting Act and 1974 Equal Credit Opportunity Act both address automated decision-making, and require companies to disclose to the consumer the principal reasons why they were denied credit.

Model interpretability has seen much research in supervised settings (classification, regression) but much less in unsupervised ones, specifically in dimensionality reduction (DR) and nonlinear embeddings (NLE), which we focus on here. Two outputs of a DR procedure are relevant for interpretability. One is the low-dimensional projections of the data points (the *embedding*, which can be visualized (typically in 2D scatterplots) to find patterns in the data. This has been widely exploited and is indeed a main application of DR. The other output is the *projection mapping* from high- to low-dimensional points. This has received some attention for linear models such as PCA but very little otherwise, in particular for NLE methods such as t-SNE, which do not naturally define an out-of-sample mapping, rather they directly learn a low-dimensional projection for each training point, by optimizing an objective function of the latter. A further consideration is that interpretability is not a black-and-white concept. In learning an explainable DR mapping, it is useful to be able to control the complexity of the explanation so as to span a range from a very accurate, detailed but complex explanation, to a less accurate but simpler explanation that may capture important overall patterns.

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

[1] GDPR: `https://gdpr-info.eu`. AI Act: https://www.bbc.com/news/technology-56830779. FTC: https://www.ftc.gov/news-events/blogs/business-blog/2021/04/aiming-truth-fairness-equity-your-companys-use-ai.

[2] https://news.bloomberglaw.com/tech-and-telecom-law/bias-in-artificial-intelligence-is-your-bot-bigoted

There is another, important advantage of learning an interpretable projection mapping jointly with the embedding. While an NLE method such as t-SNE is precisely defined by an objective function of the low-dimensional coordinates, this does not mean that the latter offer a faithful projection of the original, high-dimensional data. Firstly, the result depends in an obscure way on the objective function and on hyperparameters such as the perplexity in t-SNE. Second, the resulting embedding may give a misleading view of the data. For example, t-SNE has a strong tendency to find clusters where none exist, as can be seen with manifolds of known structure, such as the Swiss roll (Carreira-Perpiñán, 2010). Many papers present t-SNE embeddings of data without perhaps realizing that t-SNE may introduce artifacts that do no exist in the data (this criticism extends to most NLE methods, not just t-SNE, of course). Augmenting the t-SNE embedding with an interpretable out-of-sample mapping, such as the sparse oblique trees we use here, allows one to understand how the high-dimensional input instances are projected to the embedding and understand whether that makes sense, thus helping to validate the embedding.

This opens several technical issues. What mapping should we use? In section 2 we argue that sparse oblique trees are ideally suited for DR interpretability and that they can provide explanations of controllable complexity. Section 3 describes how to learn such trees. How do we learn such a mapping for a given type of NLE? In section 4 we argue that the optimal mapping must be jointly learned during the embedding process, not after the fact. However, this implies a difficult optimization problem, because the tree is not differentiable. We give an algorithm that solves this by alternatingly updating an embedding and a tree, and which works with any type of NLE, such as t-SNE (van der Maaten and Hinton, 2008), the elastic embedding (Carreira-Perpiñán, 2010), multidimensional scaling (Borg and Groenen, 2005), the Sammon mapping (Sammon, 1969) or others. Finally, in section 5 we show that the resulting tree can indeed provide useful insight into the data beyond what the 2D visualization of the embedding itself provides[3].

1 RELATED WORK

Much work exists in achieving interpretability for linear DR models such as PCA and factor analysis. In both cases, the objective function directly depends on the D coefficients of the linear mapping (if the instances are D-dimensional) $\mathbf{z} = \mathbf{F}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$. The most natural way to make this usually large number of coefficients is to make most of them zero, so each projection dimension depends on few input features. An early way to do this is with varimax rotation (Kaiser, 1958), which postprocesses the mapping as $\mathbf{z}' = \mathbf{R}\mathbf{z} = (\mathbf{R}\mathbf{A})\mathbf{x} + \mathbf{R}\mathbf{b}$ with a rotation matrix $\mathbf{R}$ that encourages having many zero elements in $\mathbf{R}\mathbf{A}$. Later work revisited this by making use of other rotations (Jolliffe and Uddin, 2000; Chennubhotla and Jepson, 2001), or of sparse formulations using the ℓ_1 or ℓ_0 norms (d'Aspremont et al., 2008; Journée et al., 2011; Jolliffe et al., 2003; Kuleshov, 2013; Mackey, 2009; Moghaddam et al., 2006; Papailiopoulos et al., 2013; Shen and Huang, 2008; Sriperumbudur et al., 2007; Thiao et al., 2010; Zhang and El Ghaoui, 2011; Zou et al., 2006) and nonnegative coefficients (Zass and Shashua, 2007). Most of these works focus on the algorithmic or theoretical aspects of achieving a fast, possibly approximate solution, which is NP-hard in some formulations.

We are aware of very little work, if any, regarding learning interpretable mappings for nonlinear embeddings such as t-SNE. An obvious approach would be to optimize the embedding and then fit to it a sparse linear or otherwise nonlinear but interpretable mapping, but as we note later this is suboptimal.

Since the form of our projection mapping is a tree of linear mappings, this can also be regarded as a form of local DR. Examples of this are mixtures of factor analyzers (Ghahramani and Hinton, 1996) or of PCAs (Kambhatla and Leen, 1997; Tipping and Bishop, 1999) and related models (Roweis et al., 2002).

2 CHOICE OF PROJECTION MAPPING

Let us call $\mathbf{F}$: $\mathbf{x} \in \mathbb{R}^D \rightarrow \mathbf{z} \in \mathbb{R}^L$ the *projection mapping*, which maps a high-dimensional point $\mathbf{x}$ in D dimensions to a low-dimensional point $\mathbf{z}$ in $L \ll D$ dimensions. This mapping is in general nonlinear and we seek a type of mapping that can be interpreted. Black-box mappings such as a neural net or a random forest, while highly accurate, are very hard to interpret in a robust way, in spite of many efforts in this direction, such as feature importance or saliency maps (Lipton, 2018). We focus here on models that are easier to interpret by construction. One candidate are linear mappings, but they are too restrictive and would distort the NLE considerably. (A linear mapping is also a particular case of our model below.) Another candidate are generalized additive models (Hastie and Tibshirani, 1990), which define $\mathbf{F}$ as a sum of functions each operating on a single feature (or

[3]Note that sections 2–3 describe supervised models (regression trees), while our overall problem is unsupervised. This is because our joint optimization algorithm in section 4 relies on a subproblem involving a regression tree.

perhaps a pair of them). These functions can then be plotted to inspect them. This model is also very restrictive, in that features are generally expected to interact, and it scales poorly with the dimension D. A third candidate are traditional, axis-aligned decision trees[4], where each decision node thresholds a single input feature and has two children. This is also very restrictive for several reasons. First, the axis-aligned structure of the splits is wholly inadequate if features interact or have correlations, and it gives rise to a large number of nodes, which makes the tree hard to interpret. Besides, the maximum number of features a binary tree with K leaves can use is $K-1$ (one per decision node), and each root-leaf path would use even fewer. If D is large, as is expected in DR, this would force the tree to have many nodes, or else it would apply a drastic feature selection on the embedding.

Then, our proposed mapping is a *sparse oblique tree*, where each decision node i uses a sparse linear mapping (a hyperplane split using few features): if $\mathbf{w}_i^T\mathbf{x}+w_{i0}>0$ we send instance $\mathbf{x}$ to the right child, else to the left child, where the $D\times 1$ vector $\mathbf{w}_i$ is sparse. Each leaf i uses a sparse linear mapping $\mathbf{A}_i\mathbf{x}+\mathbf{b}_i$ where $\mathbf{A}_i$ is a sparse $L\times D$ matrix. Such a tree is learned by optimizing the following objective function over a tree $\mathbf{T}$ of fixed structure (say, a complete tree of depth Δ):

$$E(\boldsymbol{\Theta})=\sum_{n=1}^{N}L(\mathbf{y}_n,\mathbf{T}(\mathbf{x}_n;\boldsymbol{\Theta}))+\lambda\sum_{i\in\text{nodes}}\phi_i(\boldsymbol{\theta}_i). \quad (1)$$

Here, $\mathbf{T}$: $\mathbb{R}^D\rightarrow\mathbb{R}^L$ is the tree predictive mapping with parameters $\boldsymbol{\Theta}=\{\boldsymbol{\theta}_i\}_{\text{nodes}}$, where $\boldsymbol{\theta}_i=\{\mathbf{w}_i,w_{i0}\}$ for a decision node and $\boldsymbol{\theta}_i=\{\mathbf{A}_i,\mathbf{b}_i\}$ for a leaf; the loss is the squared error $L(\mathbf{y},\mathbf{y}')=\|\mathbf{y}-\mathbf{y}'\|^2$; and the regularization terms have the form $\|\mathbf{w}_i\|_1$ for decision nodes and $\|\mathbf{A}_i\|_1$ for leaves, i.e., they are ℓ_1 penalties promoting sparsity. Note that if $\mathbf{w}_i=\mathbf{0}$ for a decision node, the node becomes redundant (it sends all instances to the right child if $w_{i0}>0$ and to the left child otherwise) and can be pruned, hence reducing the tree size. This type of tree is ideal for our goal for several reasons. It can model nonlinear mappings using very few nodes compared to an axis-aligned tree. It is especially convenient when clusters exist in the data, which can be captured by the tree leaves. It can make full use of any and all features of an instance, and exactly which features will be used depends on the path the instance follows. And, crucially, we can control the tree complexity in number of nodes, number of features used in each decision node and number of nonzero weights in each leaf mapping via the hyperparameter λ. This offers a convenient way to achieve a range of explanation levels, from detailed and accurate to simple and less accurate. If λ is large enough, the tree will collapse to a single leaf node, i.e., a sparse linear mapping. Indeed, and as seen in our experiments, the sparse oblique tree is much more accurate than an axis-aligned tree while using many fewer nodes, which makes the tree quite interpretable.

[4]We do not consider "soft" trees, where an input instance traverses every path of the tree, because they greatly complicate interpretability. (Soft trees do have the advantage of being differentiable, so learning them jointly with the NLE is straightforward via gradient-based methods.)

3 LEARNING SPARSE OBLIQUE TREES FOR REGRESSION

The desire to learn oblique trees was already expressed in the pioneering CART book (Breiman et al., 1984), but until recently no effective way existed to optimize an arbitrary objective function such as (1) over them. A recent algorithm, *Tree Alternating Optimization (TAO)* (Carreira-Perpiñán and Tavallali, 2018) is able to learn such trees in a scalable way to large dimension and sample size and produces highly accurate trees (Zharmagambetov et al., 2021c). Importantly, TAO can use any initial tree (structure and parameter values) and monotonically decrease (1) at each iteration. This is crucial since TAO appears as a subproblem of our overall algorithm in section 4. TAO was originally proposed for classification but can be modified to handle regression with sparse linear mappings at the leaves (Zharmagambetov and Carreira-Perpiñán, 2020). Furthermore, recent works on TAO have demonstrated a great improvement in training numerous tree-based models: forests of boosted and bagged trees (Carreira-Perpiñán and Zharmagambetov, 2020; Zharmagambetov et al., 2021a), neural trees (Zharmagambetov and Carreira-Perpiñán, 2021), softmax trees (Zharmagambetov et al., 2021b), etc. Here, we provide a brief description of the modified algorithm; see Appendix A for details.

TAO assumes that the initial tree structure is given (for example, a complete binary tree of depth Δ). To optimize eq. (1) over the parameters $\boldsymbol{\theta}_i$ of a tree $\mathbf{T}$, TAO uses alternating optimization over the nodes of the tree. This is possible due to two facts. Firstly, a *separability condition*: optimizing (1) over a set of nodes that are not descendants of each other separates over each individual node, so they can be handled in parallel (Carreira-Perpiñán and Tavallali, 2018). Second, optimizing over a single node can be shown to be equivalent to a simpler, *reduced problem*, over just the training instances that reach the corresponding node under the current tree (the *reduced set* $\mathcal{R}_i$ of node i), as described next.

Reduced problem over a decision node While the TAO algorithm can be applied to an arbitrary

tree structure, we consider binary trees, i.e., a decision node i has two children. If node i sends an input instance $\mathbf{x}$ to the left child, propagating $\mathbf{x}$ down that subtree (whose parameters are fixed) to a leaf results in a certain output (given by that leaf's linear predictor). Similar arguments hold for the right child. Therefore, the loss term $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta}))$ in eq. (1) can take only one of two possible values for each $\mathbf{x}_n \in \mathcal{R}_i$. Then the problem of optimizing (1) over the node's parameters can be equivalently formulated as a weighted 0/1 loss binary classification problem where $\mathbf{x}_n$ has a "pseudolabel" $\overline{y}_n$ which denotes the child achieving the lower loss. It is weighted because the loss of the best child is different for each instance (see details in Appendix A). Moreover, we add a regularization with penalty λ (eq. (1)) to control the model complexity. Training such a classifier is NP-hard but can be approximated efficiently using an ℓ_1-regularized logistic regression using existing tools, such as LIBLINEAR (Fan et al., 2008).

Reduced problem over a leaf Optimizing (1) over a leaf is easier since it has no children. It is equivalent to optimizing the leaf's predictor on its reduced set, as seen by replacing the tree output in (1) replaced by the output of a leaf. With our linear-predictor leaves, this corresponds to fitting a linear regressor with an ℓ_1 penalty, i.e., a Lasso problem (Hastie et al., 2015).

We repeat the described procedure for each node (either leaf or decision node) in our tree and one pass over the entire tree constitutes one TAO iteration. TAO keeps optimizing nodes in this way until some stopping criterion is met (e.g. error tolerance or maximum number of iterations). Fig. 1 gives the pseudocode.

3.1 Choice of the complexity hyperparameter λ and tree depth Δ

We take the initial tree structure as a complete binary tree of depth Δ, which gives the largest tree we can learn. Δ should be set to a large enough value so λ is the actual complexity control. Then, the hyperparameters in our model are Δ and $\lambda \geq 0$, which controls the actual number of nodes and nonzero weights in the tree. In a supervised setting, Δ and λ would be obtained by cross-validation, but in DR (which is an exploratory task) we have no ground truth labels. Indeed, just as the dimension L of the embedding space or the perplexity in t-SNE, these hyperparameters should be left to the user to select depending on the case. By running our algorithm for a range of λ values, from zero to very large, we obtain a range of trees whose accuracy and complexity to explain decrease as λ increases, hence a range of explanation levels.

input training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$; penalty λ;
initial tree $\mathbf{T}(\cdot; \boldsymbol{\Theta})$ of depth Δ with params $\{\boldsymbol{\theta}_i\}$;
$\mathcal{N}_0, \dots, \mathcal{N}_\Delta \leftarrow$ nodes at depth $0, \dots, \Delta$, resp.;
generate $\mathcal{R}_i$ (instances that reach node i) using an initial tree;
repeat
  for $d = \Delta$ **down to** 0
    for $i \in \mathcal{N}_d$ (can be done in parallel)
      if i is a leaf **then**
        $\boldsymbol{\theta}_i \leftarrow$ fit Lasso linear regressor on $\mathcal{R}_i$ with penalty λ
      else
        generate pseudolabels $\overline{y}_n$ and weights $\overline{w}_n$ for each instance $\mathbf{x}_n \in \mathcal{R}_i$
        $\boldsymbol{\theta}_i \leftarrow$ fit ℓ_1 regularized weighted binary classifier on $\{(\mathbf{x}_n, \overline{w}_n, \overline{y}_n)\} \in \mathcal{R}_i$ with penalty λ
    end for
  end for
  update $\mathcal{R}_i$ for each node
until stop
return T

Figure 1: Fitting a regression tree with TAO.

4 JOINTLY LEARNING AN OPTIMAL TREE AND EMBEDDING

A nonlinear embedding (NLE) method defines an objective function $E(\mathbf{Z})$ over the low-dimensional coordinates $\mathbf{Z}_{L\times N} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$ of the high-dimensional training points $\mathbf{X}_{D\times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$. For example, for t-SNE (van der Maaten and Hinton, 2008) this is:

$$E(\mathbf{Z}) = \sum_{n=1}^{N} \mathrm{D}\left(P_n \| Q_n(\mathbf{Z})\right) = \sum_{n,m=1}^{N} p_{nm} \log \frac{p_{nm}}{q_{nm}(\mathbf{Z})} \tag{2}$$

while for the elastic embedding (Carreira-Perpiñán, 2010) it is:

$$E(\mathbf{Z}) = \sum_{n,m=1}^{N} \left(w_{nm} \|\mathbf{z}_n - \mathbf{z}_m\|^2 + \alpha e^{-\|\mathbf{z}_n - \mathbf{z}_m\|^2} \right) \tag{3}$$

where the specific terms are not relevant here, what matters is that E is a function of the low-dimensional projections and that is what must be optimized over. Call the result the *free embedding*, in that it is not constrained to obey any particular mapping $\mathbf{F}$. However, if we want an out-of-sample mapping $\mathbf{F}$ so we can

project new points, then $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n)$ for $n = 1, \dots, N$ by definition and so we have a *parametric embedding* objective function:

$$\min_{\mathbf{F}} P(\mathbf{F}) = E(\mathbf{F}(\mathbf{X})) + \lambda\, \phi(\mathbf{F}) \tag{4}$$

where $\phi(\mathbf{F})$ is a regularization term on the mapping. For example, for the elastic embedding $E(\mathbf{F}(\mathbf{X}))$ would be:

$$\sum_{n,m=1}^{N} \Big(w_{nm} \|\mathbf{F}(\mathbf{x}_n) - \mathbf{F}(\mathbf{x}_m)\|^2 + \alpha e^{-\|\mathbf{F}(\mathbf{x}_n) - \mathbf{F}(\mathbf{x}_m)\|^2} \Big).$$

Note that the process (which we call *direct fit*) of training a mapping directly to predict the free embedding, i.e., $\min_{\mathbf{F}} \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2$ (possibly with regularization), will work poorly unless $\mathbf{F}$ is very flexible, and will result in a new embedding "$\mathbf{F}(\mathbf{X})$" that can considerably distort the optimal embedding. This is particularly true if we limit the complexity of $\mathbf{F}$ to make it more interpretable.

If $\mathbf{F}$ was differentiable, we could easily optimize (4) via gradient-based methods, as done for the Sammon mapping or MDS with a RBF network (Lowe and Tipping, 1996; Webb, 1995), or for t-SNE with a neural net (van der Maaten, 2009). However, this is not possible with a tree, which defines a non-differentiable mapping. Instead, we will reformulate the problem in a way that allows us to derive an iterative algorithm that capitalizes on the fact that we can use TAO to train a regression tree and the original NLE algorithm to train the embedding. To do this, we apply the *method of auxiliary coordinates (MAC)* (Carreira-Perpiñán and Wang, 2012, 2014), which is applicable to problems involving nested functions, such as (4), as follows.

First, consider the following constrained problem, where the nesting is broken:

$$\min_{\mathbf{Z},\mathbf{F}} E(\mathbf{Z}) + \lambda\, \phi(\mathbf{F}) \quad \text{s.t.} \quad \mathbf{Z} = \mathbf{F}(\mathbf{X}) \tag{5}$$

where E is the original, free embedding objective function. This is equivalent to the parametric embedding problem (4), but with "auxiliary coordinates" $\mathbf{Z}$ introduced which capture the output of $\mathbf{F}$. (A similar formulation was presented by Carreira-Perpiñán and Vladymyrov (2015), but here $\mathbf{F}$ is a nondifferentiable decision tree.) We solve (5) using a penalty method. We describe the quadratic-penalty method for simplicity, but in the experiments we use the augmented Lagrangian. This defines a new, unconstrained objective function of $\mathbf{Z}$ and $\mathbf{F}$:

$$\min_{\mathbf{Z},\mathbf{F}} E(\mathbf{Z}) + \lambda\, \phi(\mathbf{F}) + \mu \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2. \tag{6}$$

Optimizing this for fixed $\mu > 0$ produces $(\mathbf{Z}_\mu, \mathbf{F}_\mu)$ which, as $\mu \to \infty$, progressively force the constraints to be satisfied while making the objective as large as possible. Note that, for $\mu \to 0^+$ (and $\lambda = 0$), this produces as $\mathbf{Z}_0$ the free embedding and as $\mathbf{F}_0$ a tree fitted to that embedding; this is what we called direct fit earlier (without regularization). As μ increases, $\mathbf{Z}$ and $\mathbf{F}$ cooperatively reorganize to find a better solution of (5). Finally, we optimize (6) itself by alternating optimization over $\mathbf{Z}$ and $\mathbf{F}$. Over $\mathbf{Z}$, eq. (6) is the original embedding objective E but with a quadratic regularization term on $\mathbf{Z}$:

$$\min_{\mathbf{Z}} E(\mathbf{Z}) + \mu \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2. \tag{7}$$

This can be easily solved by reusing an algorithm to optimize the original embedding (t-SNE, the elastic embedding or whatever), with a minor modification to handle the additional quadratic term. This is very convenient because we can capitalize on the literature of NLE optimization, specifically on algorithms that scale to large datasets (Vladymyrov and Carreira-Perpiñán, 2012; Yang et al., 2013; van der Maaten, 2014; Vladymyrov and Carreira-Perpiñán, 2014). Over $\mathbf{F}$, the problem reduces to a regression fit which we solve using TAO:

$$\min_{\mathbf{F}} \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2 + \frac{\lambda}{\mu} \phi(\mathbf{F}). \tag{8}$$

The ability of TAO to take an initial tree and improve over it is essential here to make sure the step over $\mathbf{F}$ improves over the previous iteration, and to be able to use warm-start to speed up the computation.

In summary, our algorithm alternates between training an embedding with a regularization term that pushes it towards the current tree predictions, and training a tree to fit the current embedding. As the penalty term μ increases, the embedding and the tree coadapt until they agree on an optimal result. Fig. 2 gives the pseudocode for the overall algorithm.

5 EXPERIMENTS

Our experimental findings demonstrate that 1) tree embedding as an out-of-sample mapping is quite accurate and helps to interpret the embeddings; 2) our optimization algorithm generally finds better optima compared to the naive direct fit. Moreover, we illustrate that tree embeddings can provide helpful insights (which are not covered by the original embeddings) about data. To show that, we consider three datasets from different domains: breast cancer (available from UCI ML repository by Lichman (2013)), 20 newsgroups and MNIST. Here, we mainly focus on breast cancer and 20 newsgroups, whereas additional results (including MNIST) can be found in Appendix C.

input training set $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$;
$\mathbf{Z} \leftarrow$ fit the free embedding (eq. (2),(3), etc.);
$\mathbf{F} \leftarrow$ fit a tree to $(\mathbf{X}, \mathbf{Z}_0)$ (algorithm in fig. 1);
$\boldsymbol{\beta} \leftarrow 0$ (initialize Lagrange multipliers);
for $\mu = \mu_0 < \mu_1 < \mu_2 < \cdots < \mu_{\max}$;
  repeat
    $\mathbf{F} \leftarrow \mathbf{F} + \frac{1}{\mu}\boldsymbol{\beta}$;
    optimize over $\mathbf{Z}$ given $\mathbf{F}$ (eq. (7));
    $\mathbf{Z} \leftarrow \mathbf{Z} - \frac{1}{\mu}\boldsymbol{\beta}$;
    optimize over $\mathbf{F}$ given $\mathbf{Z}$ (eq. (8))
      via TAO (algorithm in fig. 1);
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{Z} - \mathbf{F})$ (multipliers step);
  until stop
end for
return F

Figure 2: Joint optimization framework for learning a tree and embedding (augmented Lagrangian version).

5.1 Setup

Although our method generalizes to any type of nonlinear embeddings, we perform experiments on t-SNE and elastic embedding (EE). We use our in-house implementation in Matlab so that we can easily handle $\mathbf{Z}$-step (eq. (7)) of our algorithm. The reduced dimension is set to 2D. The details of the NLE training are as follows. We compute the perplexity using the entropic affinities (Vladymyrov and Carreira-Perpiñán, 2013). We use the spectral direction method (Vladymyrov and Carreira-Perpiñán, 2012) where the gradients of the embedding objective were approximated by the Barnes-Hut method (van der Maaten, 2013) for t-SNE and the fast multipole method (Vladymyrov and Carreira-Perpiñán, 2014) for EE. We apply spectral directions until the relative error of the two recent iterates is less than 10^{-4}. For computing pairwise distances between input instances, we use entropic affinities with varying perplexity depending on dataset (Hinton and Roweis, 2003).

We use TAO to train decision trees in $\mathbf{F}$-step (eq. (8)) which is implemented in Python. All reported trees in this section are sparse oblique trees with linear leaves. To optimize individual nodes, we use Lasso linear regression (Hastie et al., 2015) in leaves and ℓ_1-regularized logistic regression in decision nodes (both implemented in scikit-learn;Pedregosa et al. (2011)); and they use the same sparsity penalty (λ). Throughout this section, *direct fit* means a minimizer of $\min_{\mathbf{F}} \|\mathbf{Z}_0 - \mathbf{F}(\mathbf{X})\|^2 + \lambda\,\phi(\mathbf{F})$, i.e., fitting a tree (with TAO) on the free embedding using the same λ penalty as our method. To train the direct fit, we initialize TAO from a complete tree of depth Δ with random parameters at each node. Maximum number of TAO iterations is set to 15. We denote our proposed method as the *tree embedding* which closely follows the implementation described in fig. 2. Initial value for $\boldsymbol{\beta}$ (estimate of the Lagrange multipliers) is set to zero. The initialization for $\mathbf{Z}$ is the *free embedding* (e.g. by running t-SNE on data). We use the direct fit as the beginning of the path (line 3 in fig 2). Empirically we found out that rescaling the penalty parameter in eq. (8) such that it always equals to λ (instead of $\frac{\lambda}{\mu}$) shows better performance, so we apply it in our experiments. After each AL step, we use warm-start in both steps by starting from the previous iteration's values. Our main running script is in Python which uses Matlab Engine API to run $\mathbf{Z}$-step.

5.2 Results

We first evaluate our method on the EE objective. We use a subset of 6 classes from 20 newsgroups document classification benchmark. Tf-idf statistics on unigrams and bigrams were used to represent each document (1000 features in total). Before running EE, we project data into 20 dimensions using PCA which helps us to eliminate noises. Pairwise distances were calculated using entropic affinities with perplexity $K = 15$. Then we run EE with homotopy parameter $l = 100$ to obtain the free embedding. For our method (tree embedding), we set the number of iterations to 15 and start from small value for $\mu_0 = 1e-6$ which is multiplied by 1.4 after each iteration. Please, refer to Appendix B for description of the preprocessing steps and other details.

Fig. 3 (bottom) shows 2D embeddings obtained by several methods. Directly applying EE on preprocessed data (free embedding) yields the best results since it does not have any constraints. Next, we train a TAO tree to learn out-of-sample mapping either by the direct fit or by using our method (tree embedding). For both methods, we set the sparsity penalty for TAO $\lambda = 0.067$. However, the spectrum of various values for λ was explored in the suppl. mat. (see animations). Tree embedding maps the data into 2D with a slight degradation compared to the free embedding, but noticeably better than the direct fit. The learning curves (bottom-right) align with that: the 1st iteration of the tree embedding is the direct fit and there is a clear improvement over iterations.

Fig. 3 (top) shows the visualization of the tree embedding (our method). Since each decision node is oblique (having hyperplane splits) and input features use tf-idf where each entry is a word, we show up to top-7 words which corresponds to the largest non-zero values in the weight vector. For each leaf, we show the region of its responsibility by using instances falling into that leaf (obtained via convex hull of the 2D mappings) and pro-

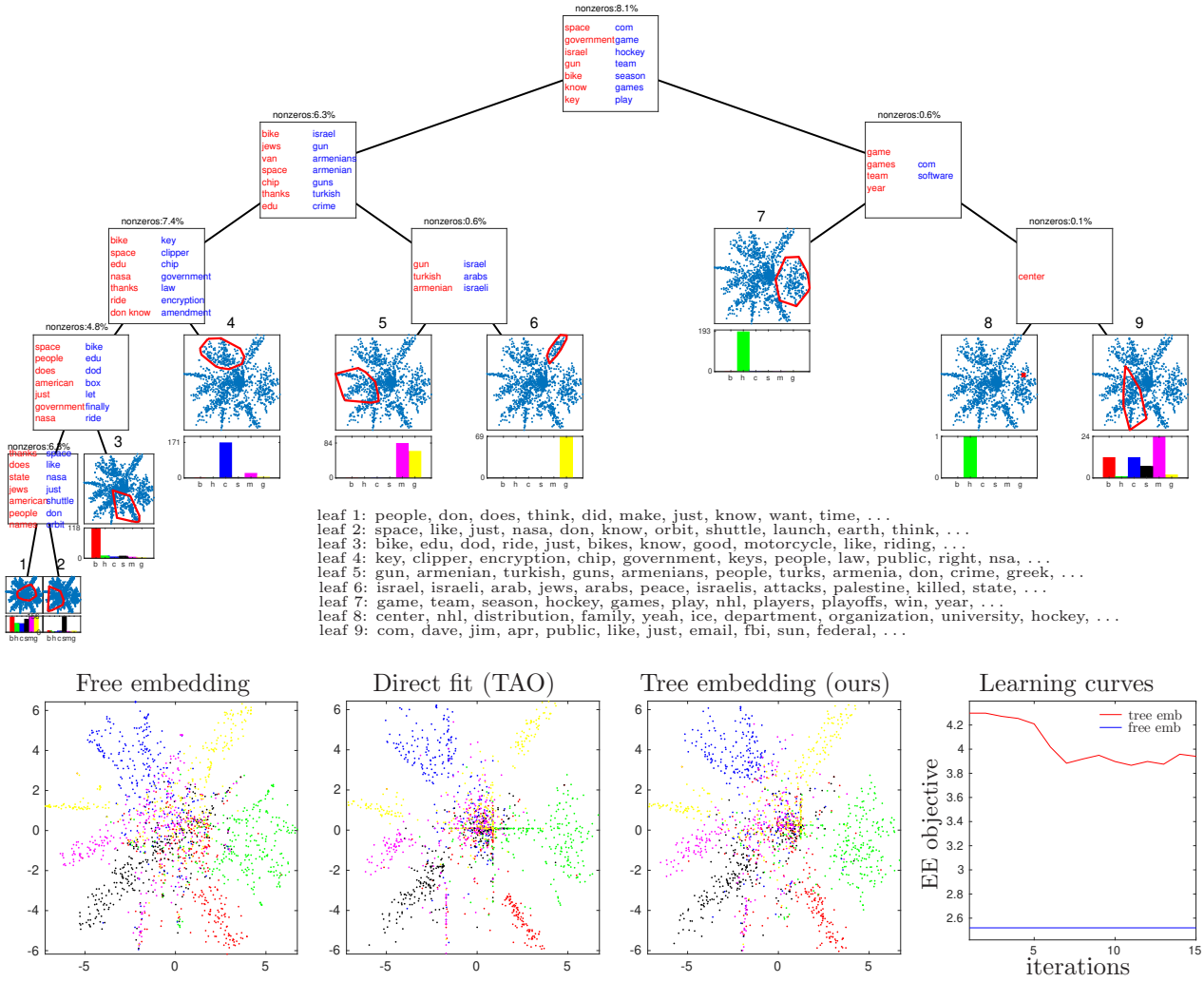

Figure 3: 20 newsgroups using EE objective. *Bottom*: 2D projections using the free embedding, direct fit and tree embedding (ours). Right figure shows the objective value over iterations. *Top*: Final tree embedding. Decision nodes show top-7 words with the largest positive (blue) or negative (red) non-zero values. Each leaf shows the region of its responsibility (top) and the histogram of class distributions (bottom, see section 5.2 for the encoding of classes in x-axis). Text at the bottom of the tree shows the most frequent 10 words in documents falling into the corresponding leaf (leaf 1, leaf 2, etc.). Note: plots at leaves are the same as the 2nd right 2D embeddings (but with uniform color).

vide histogram counts of classes where encodings in x-axis mean: "b"–motorcycles, "h"–hockey, "c"–crypt, "s"–space, "m"–mideast and "g"–guns. According to the leaf regions, there is a clear clustering structure in the tree hierarchy because majority of leaves focus on few classes and this was obtained without explicit ground truth information. Moreover, the hierarchy respects class ontology by merging instances of similar classes under one subtree. For example, gun and mideast (leaf #5 and #6) are located under the same parent, whereas their locations in 2D are not next to each others. Explanation for such separation can be obtained from decision nodes: features (words) with the highest positive/negative values were responsible for sending an instance to a certain child. In some cases, only several words were enough to identify the next node (e.g. 4 words for leaf #7 and only 1 word for leaf #8). Occasionally, the tree provides some *surprising insights*: leaf #8 has only one point which lies within the region of leaf #7 (hockey). However, the tree decided to separate that point from the remaining group and assign it closer to leaf #9 (mixed classes). By closely inspecting that document, we have found that it discusses several topics (hockey team, donation form some person, private company name and a url) which seems to be an outlier. These types of insights

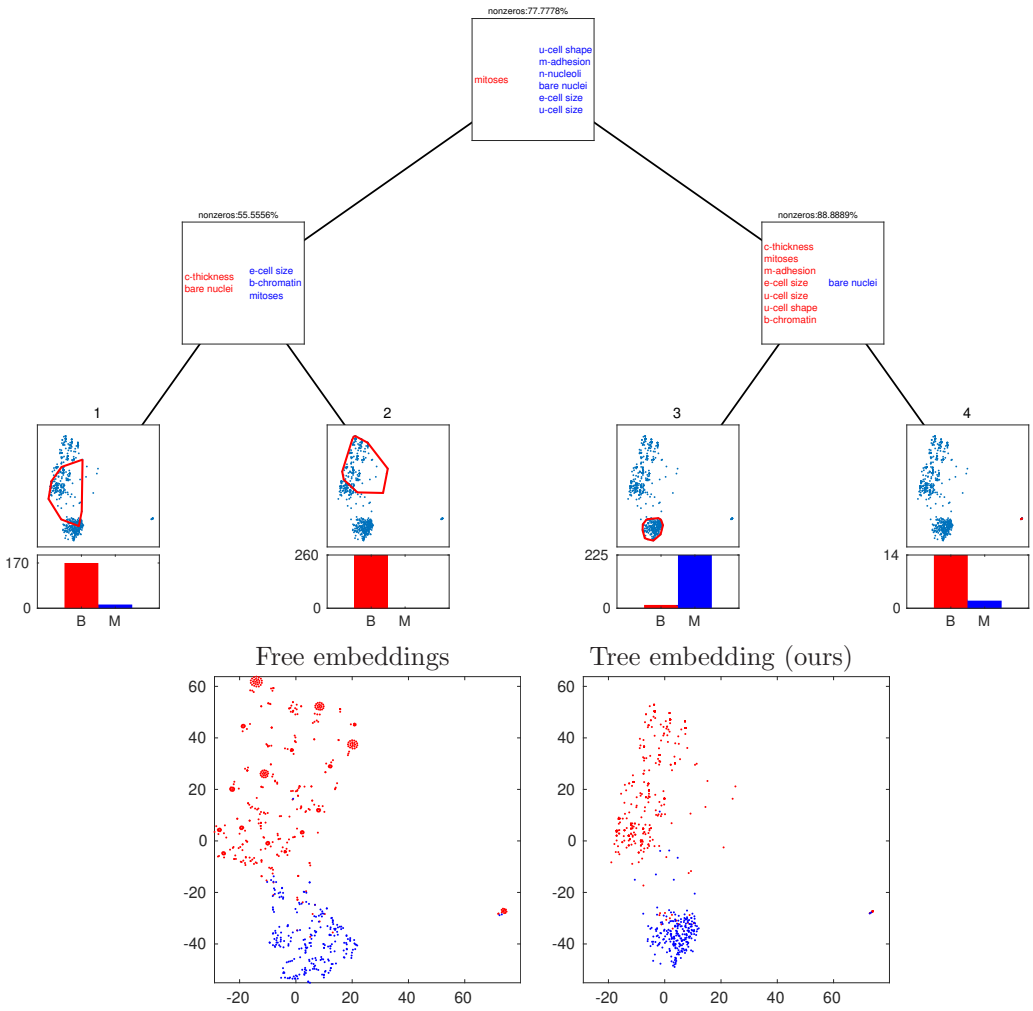

Figure 4: Similar to fig. 3 but for "breast cancer" using t-SNE.

are *not possible to infer using embeddings only*. Finally, for each leaf, we collect all documents reaching that leaf and show the most frequent words (text at the bottom of the tree). Although such information can be extracted from embeddings as well, we argue that this alone is not sufficient for interpretability. For example, top words in leaf #1 are quite generic words (mostly verbs) and it is understandable since documents in that group are from all 6 classes. But they are not insightful to determine why all these documents ended up in that region. On the other hand, the hierarchy allows us to trace the root-to-leaf path and identify region-specific words by observing weights at each decision node.

Similar conclusion can be made for fig. 4 which is for the "breast cancer" dataset. Our approach leads to the parametric embedding with a high quality (bottom panel) and tree allows us to interpret the mapping. Highlighted region at each leaf makes sense and in agreement with data. Surprisingly, leaf #1 covers the part of leaf #3. However, careful inspection shows that the border of leaf #3 contains a lot of patients from benign class and thus, the algorithm decided to assign that region to the first leaf. Again, such observations are not trivial to detect using only embeddings as we see clear separation into two clusters. In this experiment, we run t-SNE to obtain the free embedding. We use the original input features (no PCA) to compute pairwise distances and apply entropic affinities with perplexity $K = 15$. To train the tree embedding, we set $\mu_0 = 1e-5$ and multiply by 1.4 after each step (20 iterations in total). More details of the experimental setup can be found in Appendix B.

5.3 Direct fit using CART

The naive way to get the tree embedding is to first train the free embedding (e.g. via EE, t-SNE, etc.) and then fit the traditional decision tree (e.g. using

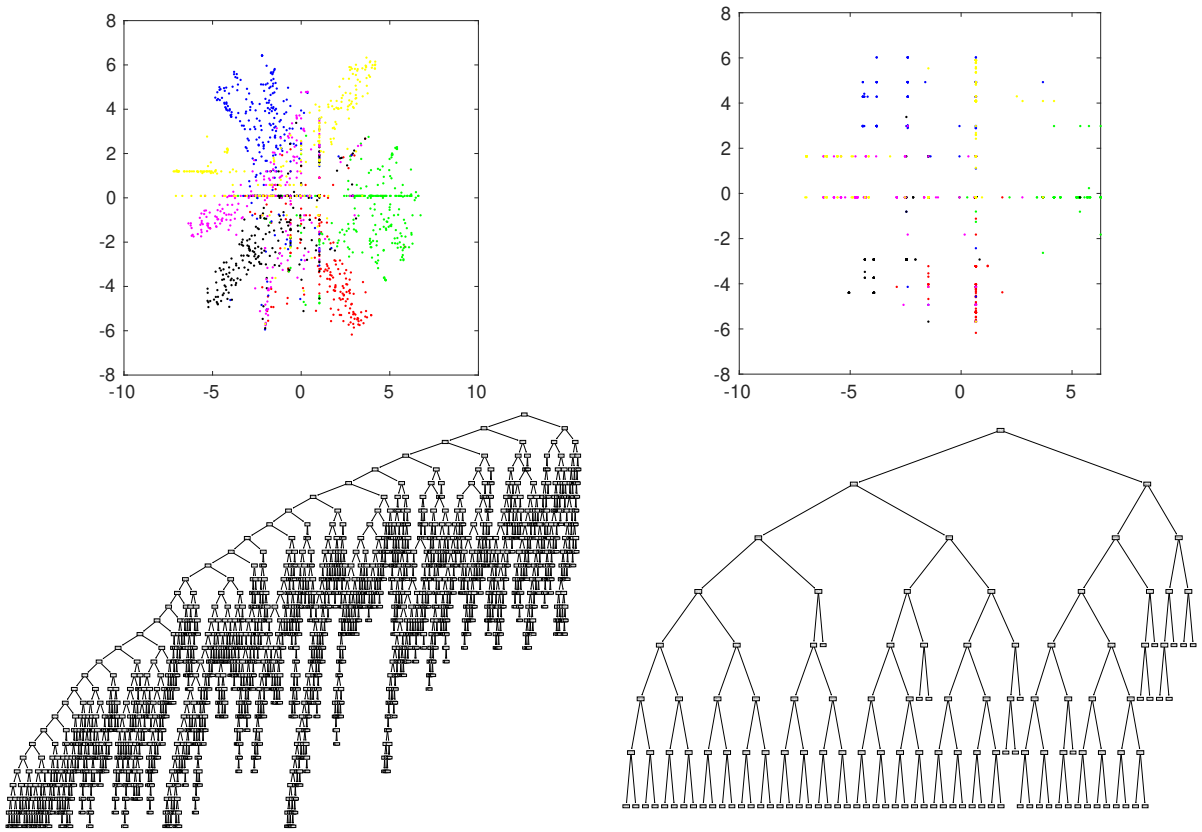

Figure 5: *Top*: 2D embeddings for 20 newsgroups provided by CART trees of depth 30 (left) and 7 (right). Each tree was trained using the free embedding (the same one as in fig. 3) as a ground truth (i.e., direct fit). *Bottom*: corresponding decision trees. It is clear that smaller (although more interpretable) CART trees lead to significant distortion of the embeddings.

CART, C4.5) to learn the mapping. However, apart from being suboptimal, we show that it can be impractical. Potentially, we can fully grow a tree which will perfectly match the free embedding. But it is well known that such model hugely overfits and the final tree could be very deep making it hard to interpret. Instead, it makes sense to apply some pruning. We use scikit-learn's implementation of CART regression trees and apply pre-pruning strategy using 30% of data as validation set. Top plot in fig. 5 shows 2D embeddings obtained by CART for the same problem as in fig. 3. For the top left figure, although the general structure is preserved, we can clearly see artifacts in certain regions due to discrete nature of CART. Substantially reducing the depth causes a significant increase in loss (right plots). More importantly, bottom plot shows the visualization of the trees. Even for relatively simple problem as 20 newsgroups, the final pruned tree in the bottom left is very deep and contains > 2400 nodes. Moreover, for regression problems, scikit-learn fits a separate tree for each output dimension and here we show one tree. It is clear that interpretability becomes non-trivial in such cases which practically limits this approach only to toy problems.

6 CONCLUSION

We have argued for the use of sparse oblique trees as a convenient choice of explanatory dimensionality reduction mapping, and provided an algorithm that learns an optimal tree for an arbitrary choice of the nonlinear embedding objective. Hence, the latter determines what the ideal low-dimensional point projections should be, while the tree determines the projection mapping that will actually produce the embedding. By controlling the tree complexity (number of nodes and nonzero parameters) via an ℓ_1 penalty we achieve a range of solutions that span a tradeoff of accuracy and interpretability. Inspecting the tree we obtain insights about the data and about how high-dimensional instances are projected to the embedding, which go beyond the insights obtained by simply visualizing the embedding in 2D.

Acknowledgements Work partially supported by NSF awards IIS–1423515 and IIS–2007147.

References

Borg, I. and Groenen, P. (2005). *Modern Multidimensional Scaling: Theory and Application*. Springer Series in Statistics. Springer-Verlag, Berlin, second edition.

Breiman, L. J., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, Calif.

Carreira-Perpiñán, M. Á. (2010). The elastic embedding algorithm for dimensionality reduction. In Fürnkranz, J. and Joachims, T., editors, *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, pages 167–174, Haifa, Israel.

Carreira-Perpiñán, M. Á. and Tavallali, P. (2018). Alternating optimization of decision trees, with application to learning sparse oblique trees. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 1211–1221. MIT Press, Cambridge, MA.

Carreira-Perpiñán, M. Á. and Vladymyrov, M. (2015). A fast, universal algorithm to learn parametric nonlinear embeddings. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 253–261. MIT Press, Cambridge, MA.

Carreira-Perpiñán, M. Á. and Wang, W. (2012). Distributed optimization of deeply nested systems. arXiv:1212.5921.

Carreira-Perpiñán, M. Á. and Wang, W. (2014). Distributed optimization of deeply nested systems. In Kaski, S. and Corander, J., editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 10–19, Reykjavik, Iceland.

Carreira-Perpiñán, M. Á. and Zharmagambetov, A. (2020). Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting. In *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, pages 35–46, Seattle, WA.

Chennubhotla, C. and Jepson, A. (2001). Sparse PCA (S-PCA): Extracting multi-scale structure from data. In Little, J. and Lowe, D., editors, *Proc. 8th Int. Conf. Computer Vision (ICCV'01)*, volume 1, pages 641–647, Vancouver, Canada.

d'Aspremont, A., Bach, F., and El Ghaoui, L. (2008). Optimal solutions for sparse principal component analysis. *J. Machine Learning Research*, 9:1269–1294.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874.

Ghahramani, Z. and Hinton, G. E. (1996). The EM algorithm for mixtures of factor analyzers. Technical Report CRG–TR–96–1, University of Toronto.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York.

Hinton, G. and Roweis, S. T. (2003). Stochastic neighbor embedding. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 857–864. MIT Press, Cambridge, MA.

Hoffgen, K.-U., Simon, H. U., and Vanhorn, K. S. (1995). Robust trainability of single neurons. *J. Computer and System Sciences*, 50(1):114–125.

Jolliffe, I. T., Trendafilov, N. T., and Uddin, M. (2003). A modified principal component technique based on the LASSO. *Journal of Computational and Graphical Statistics*, 12(3):531–547.

Jolliffe, I. T. and Uddin, M. (2000). The simplified component technique: An alternative to rotated principal components. *Journal of Computational and Graphical Statistics*, 9(4):689–710.

Journée, M., Nesterov, Y., Richtárik, P., and Sepulchre, R. (2011). Generalized power method for sparse principal component analysis. *J. Machine Learning Research*, 11:517–553.

Kaiser, H. F. (1958). The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, 23(3):187–200.

Kambhatla, N. and Leen, T. K. (1997). Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516.

Kuleshov, V. (2013). Fast algorithms for sparse principal component analysis based on Rayleigh quotient iteration. In Dasgupta, S. and McAllester, D., editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 1418–1425, Atlanta, GA.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Lipton, Z. C. (2018). The mythos of model interpretability. *Comm. ACM*, 81(10):36–43.

Lowe, D. and Tipping, M. E. (1996). Feed-forward neural networks and topographic mappings for exploratory data analysis. *Neural Computing & Applications*, 4(2):83–95.

Mackey, L. (2009). Deflation methods for sparse PCA. In Koller, D., Bengio, Y., Schuurmans, D., Bottou, L., and Culotta, A., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 21, pages 1017–1024. MIT Press, Cambridge, MA.

Moghaddam, B., Weiss, Y., and Avidan, S. (2006). Spectral bounds for sparse PCA: Exact and greedy algorithms. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 18. MIT Press, Cambridge, MA.

Neches, R., Swartout, W. R., and Moore, J. D. (1985). Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Trans. Software Engineering*, 11(11):1337–1351.

Papailiopoulos, D. S., Dimakis, A. G., and Korokythakis, S. (2013). Sparse PCA through low-rank approximations. In Dasgupta, S. and McAllester, D., editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 747–755, Atlanta, GA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830. Available online at `https://scikit-learn.org`.

Pitt, L. and Valiant, L. G. (1988). Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984.

Roweis, S., Saul, L., and Hinton, G. (2002). Global coordination of local linear models. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pages 889–896. MIT Press, Cambridge, MA.

Sammon, Jr., J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Trans. Computers*, C–18(5):401–409.

Shen, H. and Huang, J. Z. (2008). Sparse principal component analysis via regularized low rank matrix approximation. *J. Multivariate Analysis*, 99(6):1015–1034.

Sriperumbudur, B. K., Torres, D. A., and Lanckriet, G. R. G. (2007). Sparse eigen methods by D.C. programming. In Ghahramani, Z., editor, *Proc. of the 24th Int. Conf. Machine Learning (ICML'07)*, Corvallis, Oregon.

Thiao, M., Dinh, T. P., and Thi, H. A. L. (2010). A DC programming approach for sparse eigenvalue problem. In Fürnkranz, J. and Joachims, T., editors, *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, Haifa, Israel.

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482.

van der Maaten, L. J. P. (2009). Learning a parametric embedding by preserving local structure. In van Dyk, D. and Welling, M., editors, *Proc. of the 12th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2009)*, pages 384–391, Clearwater Beach, FL.

van der Maaten, L. J. P. (2013). Barnes-Hut-SNE. In *Proc. of the 1st Int. Conf. Learning Representations (ICLR 2013)*, Scottsdale, AZ.

van der Maaten, L. J. P. (2014). Accelerating t-SNE using tree-based algorithms. *J. Machine Learning Research*, 15:3221–3245.

van der Maaten, L. J. P. and Hinton, G. E. (2008). Visualizing data using t-SNE. *J. Machine Learning Research*, 9:2579–2605.

Vladymyrov, M. and Carreira-Perpiñán, M. Á. (2012). Partial-Hessian strategies for fast learning of nonlinear embeddings. In Langford, J. and Pineau, J., editors, *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, pages 345–352, Edinburgh, Scotland.

Vladymyrov, M. and Carreira-Perpiñán, M. Á. (2013). Entropic affinities: Properties and efficient numerical computation. In Dasgupta, S. and McAllester, D., editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 477–485, Atlanta, GA.

Vladymyrov, M. and Carreira-Perpiñán, M. Á. (2014). Linear-time training of nonlinear low-dimensional embeddings. In Kaski, S. and Corander, J., editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 968–977, Reykjavik, Iceland.

Webb, A. R. (1995). Multidimensional scaling by iterative majorization using radial basis functions. *Pattern Recognition*, 28(5):753–759.

Yang, Z., Peltonen, J., and Kaski, S. (2013). Scalable optimization for neighbor embedding for visualization. In Dasgupta, S. and McAllester, D., editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 127–135, Atlanta, GA.

Zass, R. and Shashua, A. (2007). Nonnegative sparse PCA. In Schölkopf, B., Platt, J., and Hofmann, T., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 1561–1567. MIT Press, Cambridge, MA.

Zhang, Y. and El Ghaoui, L. E. (2011). Large-scale sparse principal component analysis with application to text data. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 24, pages 532–539. MIT Press, Cambridge, MA.

Zharmagambetov, A. and Carreira-Perpiñán, M. Á. (2020). Smaller, more accurate regression forests using tree alternating optimization. In Daumé III, H. and Singh, A., editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 11398–11408, Online.

Zharmagambetov, A. and Carreira-Perpiñán, M. Á. (2021). Learning a tree of neural nets. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21)*, pages 3140–3144, Toronto, Canada.

Zharmagambetov, A., Gabidolla, M., and Carreira-Perpiñán, M. Á. (2021a). Improved boosted regression forests through non-greedy tree optimization. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event.

Zharmagambetov, A., Gabidolla, M., and Carreira-Perpiñán, M. Á. (2021b). Softmax tree: An accurate, fast classifier when the number of classes is large. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP 2021)*, pages 10730–10745, Online.

Zharmagambetov, A., Hada, S. S., Gabidolla, M., and Carreira-Perpiñán, M. Á. (2021c). Non-greedy algorithms for decision tree optimization: An experimental comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event.

Zou, H., Hastie, T., and Tibshirani, R. (2006). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286.

Supplementary Material: Learning Interpretable, Tree-Based Projection Mappings for Nonlinear Embeddings

Supplementary Material provides the following: 1) Extended explanation of the tree learning algorithm with TAO. 2) Experimental setup for datasets and methods. 3) Additional experiments, including: results for CART trees and visualizations; results for MNIST; exploring the effect of sparsity penalty in trees and visualizing the linear mapping at each leaf.

A LEARNING SPARSE OBLIQUE TREES FOR REGRESSION

This section provides a detailed explanation of the tree learning algorithm, Tree Alternating Optimization (TAO), that we use (i.e., extended version of section 4). Originally developed for training classification trees (oblique and axis aligned), Tree Alternating Optimization (TAO) (Carreira-Perpiñán and Tavallali, 2018) can be extended to fit multi-output regression trees, which enables us to use it in **F**–step of our proposed algorithm. TAO assumes that the initial tree structure is given (can be random or generated from other methods, such as CART) and it minimizes a supervised regression objective function:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{i \in \text{nodes}} \phi_i(\boldsymbol{\theta}_i) \tag{9}$$

where $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \mathbb{R}^K$ is the training set of D-dimensional input and K-dimensional outputs. In our problem, the ground truth outputs $\mathbf{Z}$ come from nonlinear embeddings (NLE). We use the mean squared error (MSE) as the loss function $L(\cdot, \cdot)$, but other regression losses can be employed (e.g. huber loss). We apply regularization (ϕ) for parameters at each node i to encourage sparsity (ℓ_1 penalty was used in our experiments). $\mathbf{T}(\cdot; \boldsymbol{\Theta}) \in \mathbb{R}^K$ denotes the tree output parametrized by $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}$, where $\boldsymbol{\theta}_i$ are the weights at node i. Throughout this paper, we consider oblique decision trees with linear leaves. That is, $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$ for a *decision node* which applies a dot product to determine the next child (e.g. $\mathbf{w}_i^T \mathbf{x} + w_{i0} \geq 0$); and $\boldsymbol{\theta}_i = \{\mathbf{A}_i, \mathbf{b}_i\}$ for a *leaf* which performs matrix-vector product and produces the actual output (embedding mapping).

To learn parameters $\boldsymbol{\theta}_i$ of a tree $\mathbf{T}$, TAO uses coordinate descent which fixes one part of the tree and optimizes over the remaining set of nodes. Under a certain assumptions, TAO guarantees the monotonic decrease of the objective function value in eq. 9. To make it work, the loss function must be separable over each instance (most losses in regression satisfy this property) and unfixed nodes must be non-descendants w.r.t. each others. All such nodes can be optimized in parallel (see fig. 6, left). This is called–a **separability condition**. We omit the formal proof of this condition since it is identical to (Carreira-Perpiñán and Tavallali, 2018). Next, optimization of a single node (which we call a *reduced problem*) depends on its type: it is weighted 0/1 loss binary classification problem (with ℓ_1 penalty) for a decision node and fitting ℓ_1 penalized linear regressor for a leaf. For both of them, the optimization utilizes a subset of instances that reach the corresponding node under the current tree. Such subset is called a *reduced set* (denoted as $\mathcal{R}_i$). We discuss each of these optimization steps below.

Reduced problem over a decision node A single decision node optimization can be intuitively explained using fig. 6 (right). Suppose we optimize over the node $i = 2$, which means we fix all parameters of a tree except $\boldsymbol{\theta}_2$ and we need to only consider the reduced set $\mathcal{R}_2$. The given node has two children $\mathcal{C}_2 = \{4, 5\}$ with the corresponding subtrees: $\mathbf{T}_4$ and $\mathbf{T}_5$ for the left and right children of node i, respectively. If node $i = 2$ sends an instance $\mathbf{x}$ to the left child, then we compute $\mathbf{T}_4(\mathbf{x})$ and return its prediction. Note that the parameters of $\mathbf{T}_4$ are fixed which means $\mathbf{x}$ follows a unique path starting from node $i = 4$ down to a certain leaf. Then we apply a predictor at that leaf (linear mapping, in our case) to compute the output. Similar arguments hold for the right child. Therefore, the loss term $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta}))$ in eq. (9) can take only one of two possible values for all $\mathbf{x}_n \in \mathcal{R}_2$: $l_{in,4} = L(\mathbf{y}_n, \mathbf{T}_4(\mathbf{x}_n))$ if the left child is chosen and $l_{in,5} = L_n(\mathbf{y}_n, \mathbf{T}_5(\mathbf{x}_n))$ if the right child is chosen.

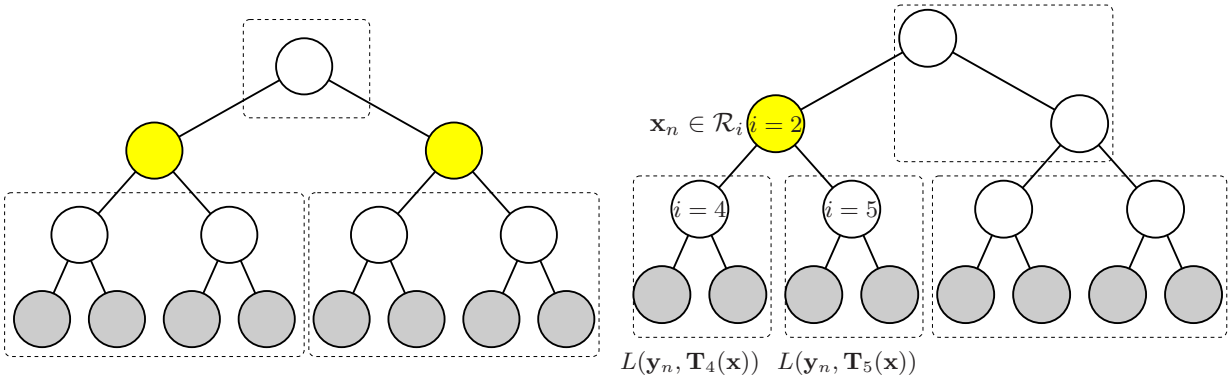

Figure 6: *Left*: Separability condition which states that non-descendant set of nodes (e.g. yellow nodes) can be optimized independently. *Right*: Optimizing a single decision node for $i = 2$ (i.e., reduced problem) which includes: 1) calculate errors induced by left/right subtrees for each instance $\mathbf{x}$ that reaches the given node ($\mathcal{R}_i$); 2) assign pseudolabels and weights; 3) solve the weighted binary classification problem. *Note*: highlighted region (with dashed lines) indicates the fixed part of the tree.

Then, define a cost function l_{in}: $\mathcal{C}_i \rightarrow \mathbb{R}$ as $l_{in,z} = L(\mathbf{y}_n, \mathbf{T}_z(\mathbf{x}_n))$, where $z \in \mathcal{C}_i$ is any child of i and $\mathbf{T}_z(\cdot)$ is the output of subtree z. In other words, the cost function l_{in} gives the loss value incurred by each of the two children of node i. Since we have only two children, our problem reduces to encouraging a decision function at node $i = 2$ (denoted as f_2) to send $\mathbf{x}$ to the "best" child. Suppose our best child is denoted as $\overline{y}_{in}$ and we want to send $\mathbf{x}$ to that child (ideally). In order to achieve this, define a function $\overline{L}_{in}$ that satisfies $\overline{L}_{in}(\overline{y}_{in}, y) = 0$ if $y = \overline{y}_{in}$, and $\overline{L}_{in}(\overline{y}_{in}, y) = \max_z (l_{in,z}) - \min_z (l_{in,z})$ otherwise, hence it is a weighted 0/1 loss function with "ground-truth" label $\overline{y}_{in}$ (the best child). For example, if $\mathcal{C}_i = \{\texttt{left}, \texttt{right}\}$ and $l_{in}(\texttt{left}) > l_{in}(\texttt{right})$ then $\overline{y}_{in} = \texttt{right}$, $\overline{L}_{in}(\texttt{right}, \texttt{right}) = 0$ and $\overline{L}_{in}(\texttt{right}, \texttt{left}) > 0$. This leads us to the following weighted binary classification problem over a single decision node i:

$$\min_{\boldsymbol{\theta}_i} \overline{E}_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} \overline{L}_{in}(\overline{y}_{in}, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{10}$$

Optimizing the given weighted 0/1 loss above (eq. (10)) over a hyperplane is an NP-hard problem even for the unweighted case (Pitt and Valiant, 1988; Hoffgen et al., 1995). However, good approximate solutions can be obtained efficiently by using a convex surrogate loss. In all our experiments, we use the logistic loss with an ℓ_1 regularizer (implemented in LIBLINEAR (Fan et al., 2008)).

Reduced problem over a leaf Optimizing a leaf is much easier since leaves do not have any children. Clearly, leaves are non-descendant w.r.t. each others and we can apply the *separability condition*. By fixing the remaining parameters of a tree, we can train each leaf independently. Note that $\mathbf{x}$ follows root-to-leaf path and actual prediction is given by a leaf. Therefore, tree output from eq. (9) can be replaced by the output of a leaf. But that leaf operates only on a subset of points reaching the given leaf (its reduced set $\mathcal{R}_i$). Therefore, the solution for any leaf i is the minimization of (9) over its parameters $\boldsymbol{\theta}_i$ on a reduced set $\mathcal{R}_i$ which, in our case, corresponds to fitting a linear regressor with ℓ_1 penalty (i.e., Lasso problem (Hastie et al., 2015)).

We repeat the described procedure for each node (either leaf or decision node) in our tree and one pass over the entire tree constitutes one TAO iteration. TAO keeps optimizing nodes in this way until some stopping criterion met (e.g. error tolerance or maximum number of iterations). Fig. 1 in the main paper provides the pseudocode of the algorithm.

B EXPERIMENTAL SETUP

This section provides details of the conducted experiments for replicability purposes. It includes: datasets descriptions and preprocessing steps; implementation details and hyperparameters choice for NLE methods; and implementation details for TAO.

B.1 Datasets

Dataset	N	D	classes
20 newsgroups	3496	1000	6
MNIST	2 000	784	10
Breast cancer	699	9	2

Table 1: Datasets used in our experiments: number of instances (N), number of features D, number of classes K. Note: class information is not used by any of the methods, only for reporting purposes.

Table 1 summarizes the datasets used in this study. Below, we provide the details of the preprocessing steps:

- **20 newsgroups**[5]. We select a subset of 6 classes: 'rec.motorcycles', 'rec.sport.hockey', 'sci.crypt', 'sci.space', 'talk.politics.mideast', 'talk.politics.guns'. The resulting documents are collected and transformed into tf-idf representations with top 1000 unigrams and bigrams. We use scikit-learn's implementation for tf-ifd and set `min_df`= 3. Before that, we apply standard preprocessing steps: removing non-alphanumeric symbols, English stopwords and headers/footers from documents. To compute entropic affinities, we first project the data into 20 dimensions using PCA.

- **MNIST** (LeCun et al., 1998). We randomly select a subset of 2000 points from the training data with equal number of instances per class. We normalize features to have values between [0,1] and subtract mean. To compute entropic affinities, we first project the data into 50 dimensions using PCA.

- **Breast cancer**[6]. We normalize features to have values between [0,1]. Entropic affinities are computed directly on the original input features.

B.2 NLE methods

Although our method generalizes to any type of nonlinear embeddings, we perform experiments on t-SNE and elastic embedding (EE). For both of them, we use our in-house implementation in Matlab so that we can easily handle $\mathbf{Z}$-step of our algorithm. The reduced dimension is set to 2D. The details of the optimization are as follows: we use the spectral direction method (Vladymyrov and Carreira-Perpiñán, 2012) where the gradients of the embedding objective were approximated by the Barnes-Hut method (van der Maaten, 2013) for t-SNE and the fast multipole method (Vladymyrov and Carreira-Perpiñán, 2014) for EE. We apply spectral directions until the relative error of the two recent iterates is less than 10^{-4}. For computing pairwise distances between input instances, we use entropic affinities with varying perplexity (K) depending on dataset (Hinton and Roweis, 2003): $K = 15$ for 20 newsgroups and breast cancer; and $K = 30$ for MNIST. The α (see eq. (3)) parameter for the elastic embedding is set to $l = 100$ in all experiments.

B.3 Decision trees

TAO We use oblique decision trees (having a hyperplane function at each decision node) and each each leaf has a linear mapping which produces continuous output (multidimensional). An initial tree is a complete binary tree of given depth (Δ) with random parameters at each node (each node's weight vector has Gaussian (0,1) entries, and then we normalize the vector to unit length). We use 15 TAO iterations to train trees at each μ (also in direct fit). The following hyperparameters are set individually for each dataset:

- **20 newsgroups**: $\Delta = 6$, the sparsity penalty λ is explored from the following range: $\lambda = [1/200, 1/100, 1/50, 1/30, 1/20, 1/15, 1/10, 1/7, 1/5, 1/3, 1.0]$

- **MNIST**: $\Delta = 6$, the sparsity penalty λ is explored from the following range: $\lambda = [1/30, 1/10, 1/5, 1.0, 1.25, 2, 2.5, 3.33, 5, 10]$

[5] `http://qwone.com/~jason/20Newsgroups/`

[6] `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)`

- **Breast cancer**: $\Delta = 4$, the sparsity penalty (λ) is explored from the following range: $\lambda = [1/200, 1/100, 1/50, 1/30, 1/20, 1/15, 1/10, 1/7, 1/5, 1/3, 1.0]$

We implemented TAO in Python 3.7.6 and do not use any parallel processing. We use ℓ_1 regularized logistic regression (implemented in LIBLINEAR (Fan et al., 2008)) to solve the penalized binary classification problem at each decision node. LIBLINEAR is accessed through scikit-learn interface (Pedregosa et al., 2011). The leaf optimization involves an ℓ_1-regularized linear regression (Lasso) for which we also use scikit-learn (internally it uses coordinate descent solver).

CART We use the Python implementation in scikit-learn Pedregosa et al. (2011). Potentially, we can fully grow a tree which will perfectly match the free embedding on the given dataset. But it is well known that such model hugely overfits (which is undesirable for out-of-sample mapping) and the final tree could be very deep making it hard to interpret. Instead, it makes sense to apply some pruning. We apply pre-pruning strategy using 30% of data as validation set. Once the `max_depth` hyperparameter is identified, we fit the tree with all available data.

C ADDITIONAL EXPERIMENTS

This section provides additional experimental results and findings. Specifically, we explore the *effect of regularization parameter λ in TAO* on interpretability and accuracy of the parametric embedding. Separate gif files further provide a collection of trees with the decreasing value of λ for MNIST and 20 newsgroups. Additionally, we visualize the actual mapping that is happening at each leaf. Finally, we illustrate a subset of the trained CART trees on the same tasks.

C.1 Experiments on 20 newsgroups

Fig. 7–8 present trained tree embeddings for two different values for λ (the sparsity penalty in TAO). It can be clearly seen that smaller sparsity value $\lambda = 0.01$ yields more accurate embedding which has smaller objective value. However, visually there is not much difference compared to $\lambda = 0.067$ which also provides quite accurate 2D visualization. Moreover, the obtained tree is much smaller and this results to more interpretability. Note that in both cases, the tree embedding improves over the direct fit (see learning curves in bottom-right) since the 1st iteration of our algorithm is the direct fit and there is a clear improvement over iterations. This is also noticeable from visualizations. Please, refer to section 5.2 for the discussion regarding interpretation of the resulting trees.

C.2 Experiments on MNIST

We conduct similar experiments on MNIST (fig. 9–10). As in the previous case, the tree embedding improves over the direct fit. This is also noticeable from visualizations (especially in fig. 10). However, in this experiment, we choose a different visualization method for decision nodes: sparse vector $\in \mathbb{R}^{784}$ at each node is reshaped to form the matrix $\in \mathbb{R}^{28\times 28}$ which is then displayed as an image with scaled colors (colored according to the sign). Since MNIST contains the collection of handwritten digits, we can clearly see which part of the image is being considered to select the next child (left or right). For example, the leaf #5 in fig 10 contains mostly digits from class 0. Consider the parent of that leaf. Red strokes indicate that the positive weights are picking left/right side pixels in digit 0 which are not presented in other digits. Moreover, 0 does not contain any strokes in the middle which is clearly shown in that node (blue weights). Similar information can be extracted from fig. 9. However, this is not trivial since the tree is somewhat bigger due to λ, although embeddings look better w.r.t. free embeddings. These figures further confirm that there is a trade-off between accuracy and interpretability which can be controlled using our method.

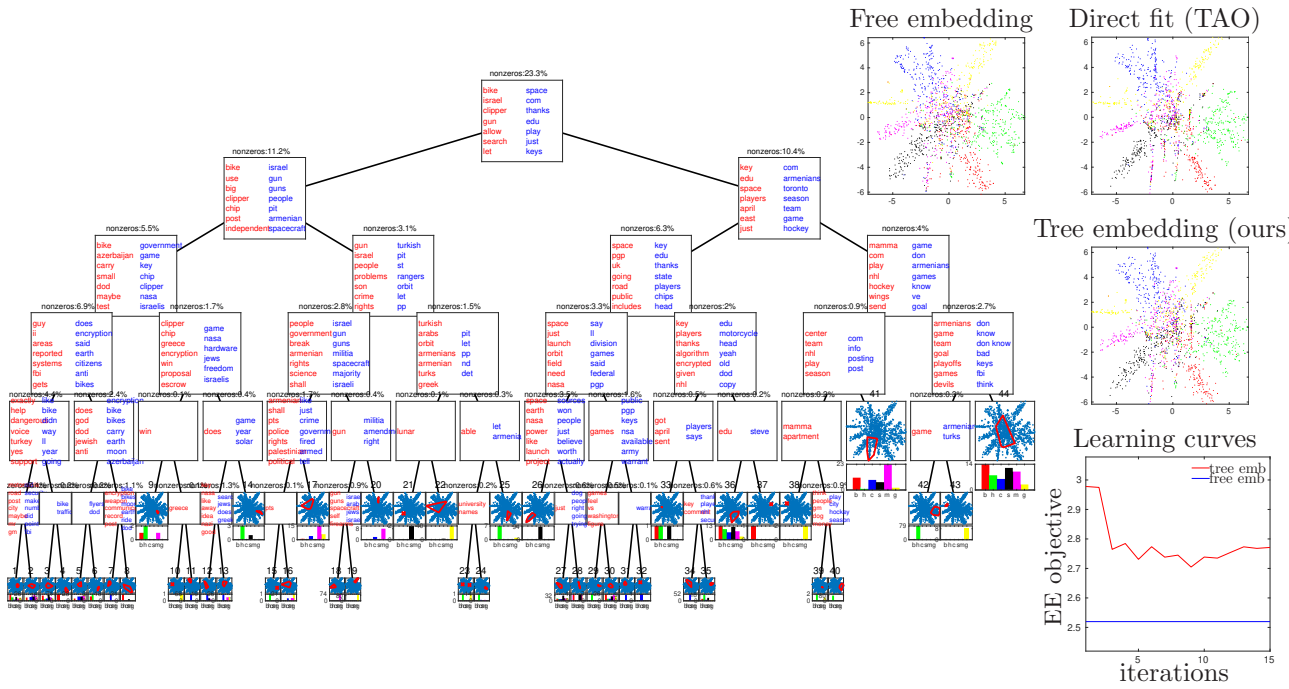

Figure 7: Results for 20 newsgroups. This is similar to fig. 3 in the main paper but uses $\lambda = 0.01$ as the tree sparsity penalty. The purpose of this figure is to show that the output of a tree can better match with the free embedding (i.e., obj. func. approaches the free emb.) at a cost of having more nodes (which is less interpretable).

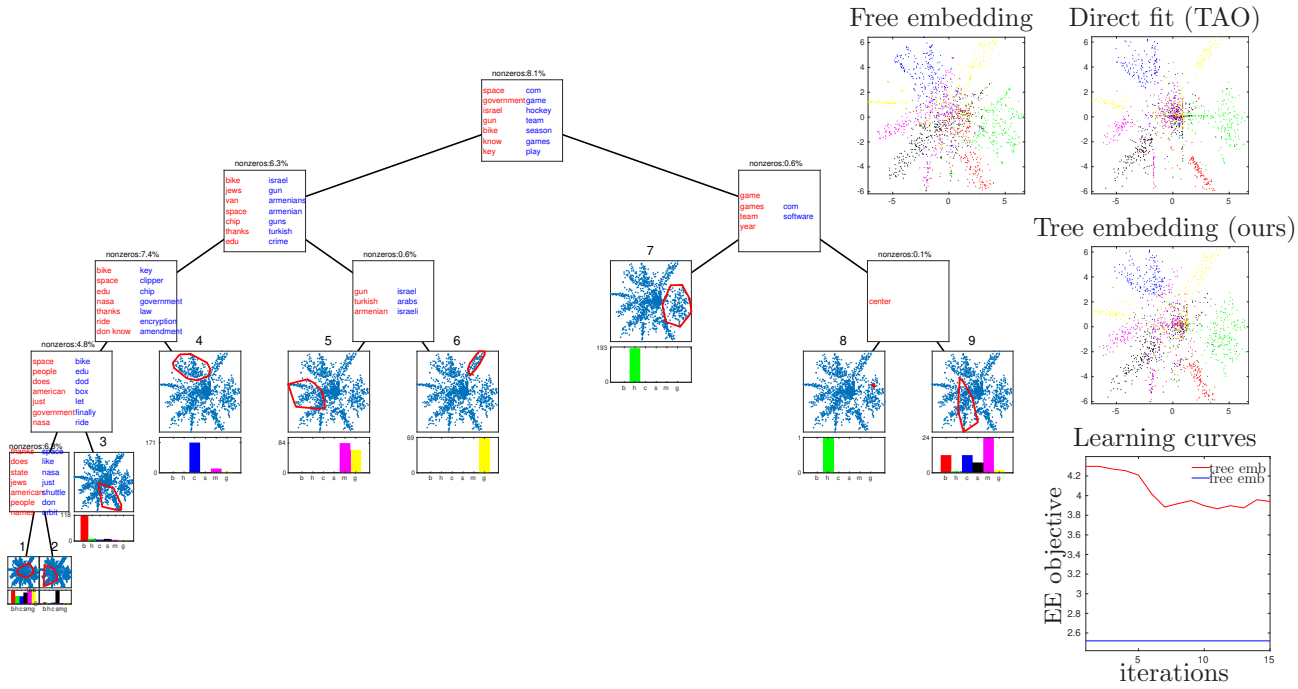

Figure 8: Similar to fig. 7 but for $\lambda = 0.067$ which enforces more sparsity. We obtain more interpretable tree at a cost of having larger objective function value (compared to fig. 7).

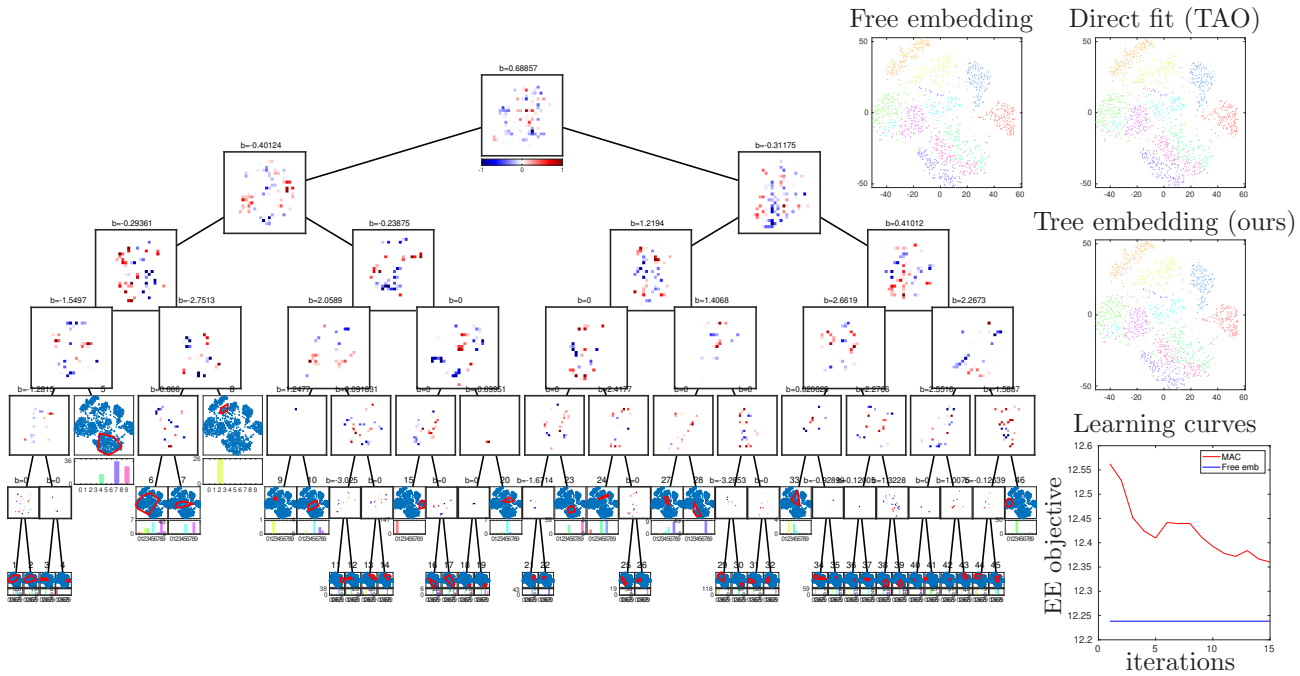

Figure 9: Results for MNIST with $\lambda = 0.03$ as the tree sparsity penalty. Similar to fig. 7, this figure shows that the output of a tree can better match with the free embedding (i.e., obj. func. approaches the free emb.) at a cost of having more nodes (which is less interpretable). Note: title in decision nodes shows a value of the bias term in node parameters.

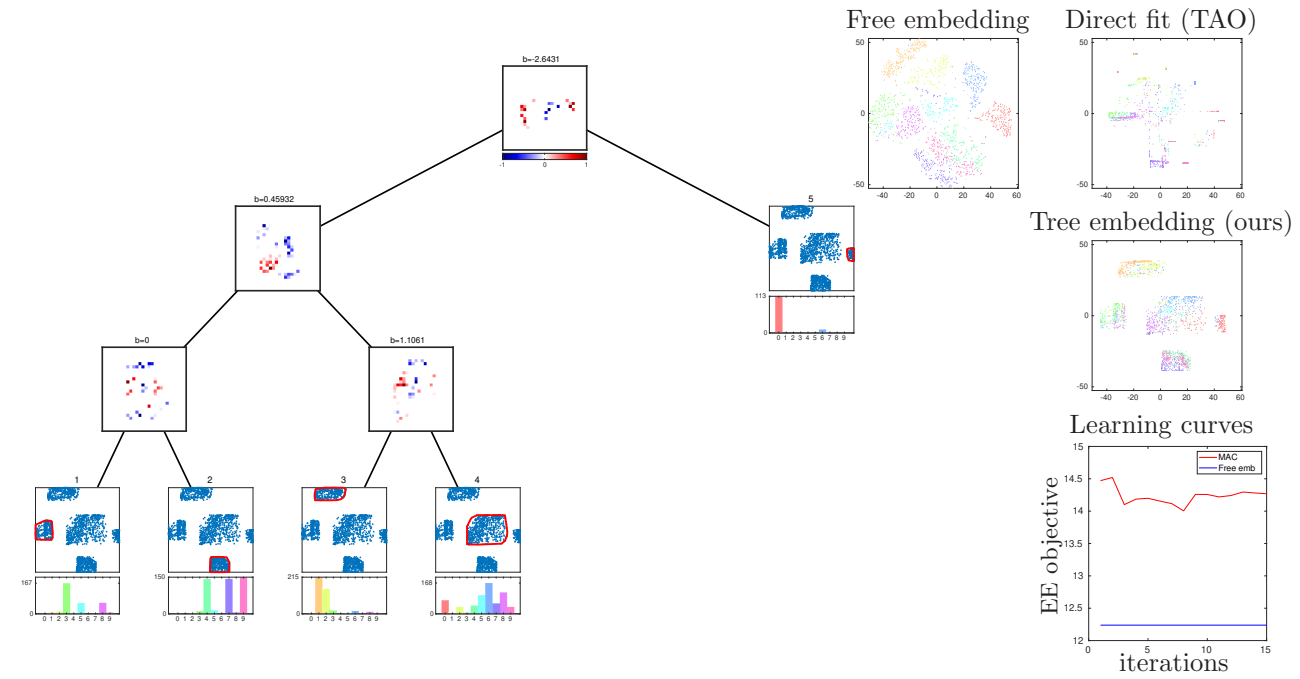

Figure 10: Similar to fig. 9 but for $\lambda = 1.25$ which enforces more sparsity. We obtain more interpretable tree at a cost of having larger objective function value (compared to fig. 9).

Additionally, fig. 11 visualizes the linear mapping at each leaf. For MNIST, each leaf contains matrix $\mathbf{A} \in \mathbb{R}^{784\times 2}$ which maps an input vector into the 2D manifold. Note that $\mathbf{A}$ was obtained by fitting ℓ_1 penalized linear regression (Lasso). Therefore, the resulting matrix is sparse. We perform row-wise sum of that matrix to obtain 784 dimensional vector. Then, similarly to the decision node visualization, we reshape the resulting vector to form the matrix $\in \mathbb{R}^{28\times 28}$ and display it as an image. The resulting images make sense. For example, leaf #2 focuses on classes: 4,7 and 9. Therefore, non-zero elements at that leaf are concentrated where we typically have strokes in these digits (e.g. bottom-center part and in the middle). Similarly, leaf #5 has non-zero weights where digit 0 typically has pixels. Leaf uses only those features to map an instance into the 2D manifold.

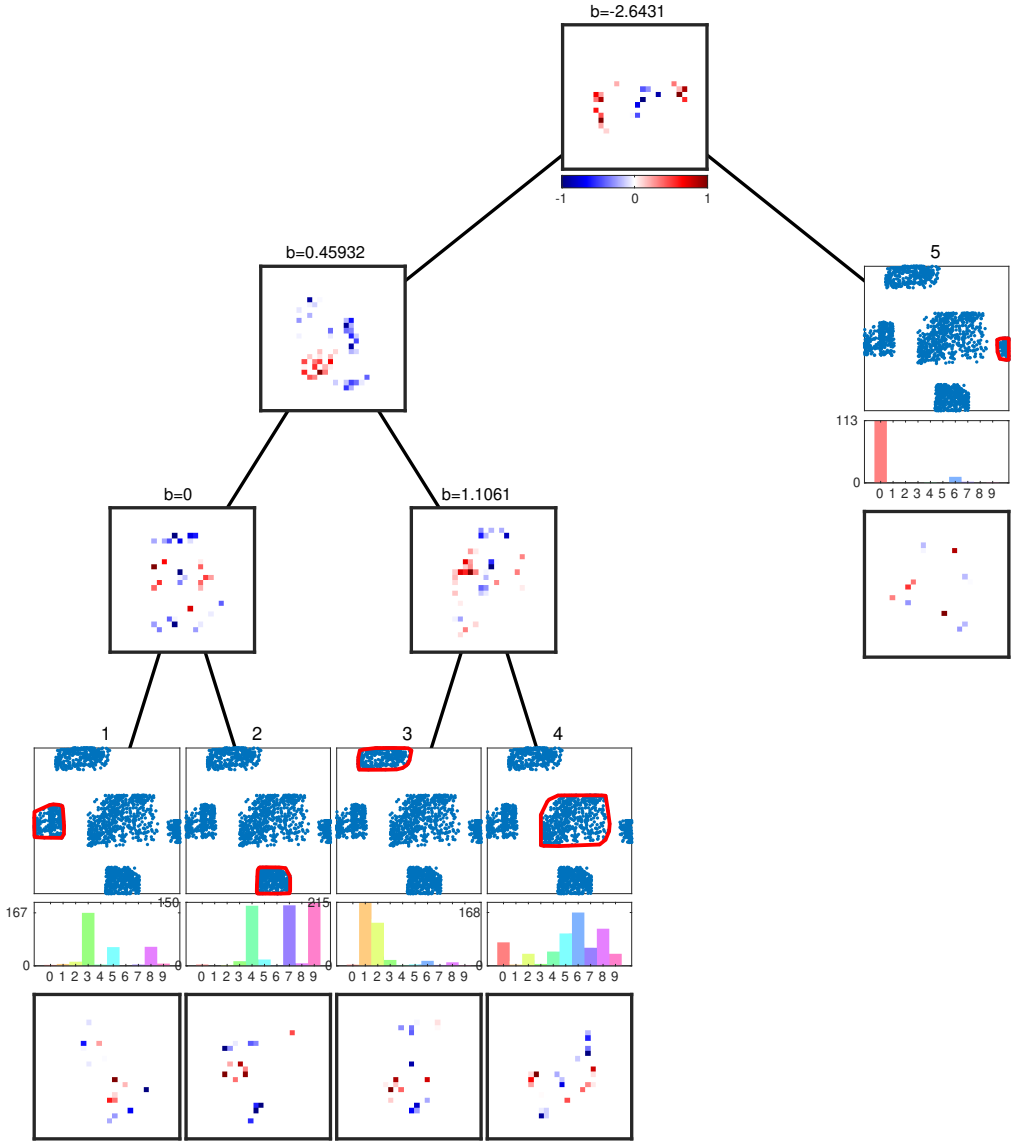

Figure 11: Similar to fig. 10 but visualization of the linear mapping at each leaf is added.

C.3 Experiments with CART trees

Free emb	Tree emb (λ0.01)	Tree emb ($\lambda = 0.067$)	CART
2.52	2.70	3.86	3.82

Table 2: EE objective value (see eq. (3) in the main paper) for the free embedding, tree embedding and CART on 20 newsgroups (lower is better). Although CART provides a similar error as the tree embedding with $\lambda = 0.067$, but the resulting trees are humongous and non-trivial to interpret (see fig. 12–13 below).

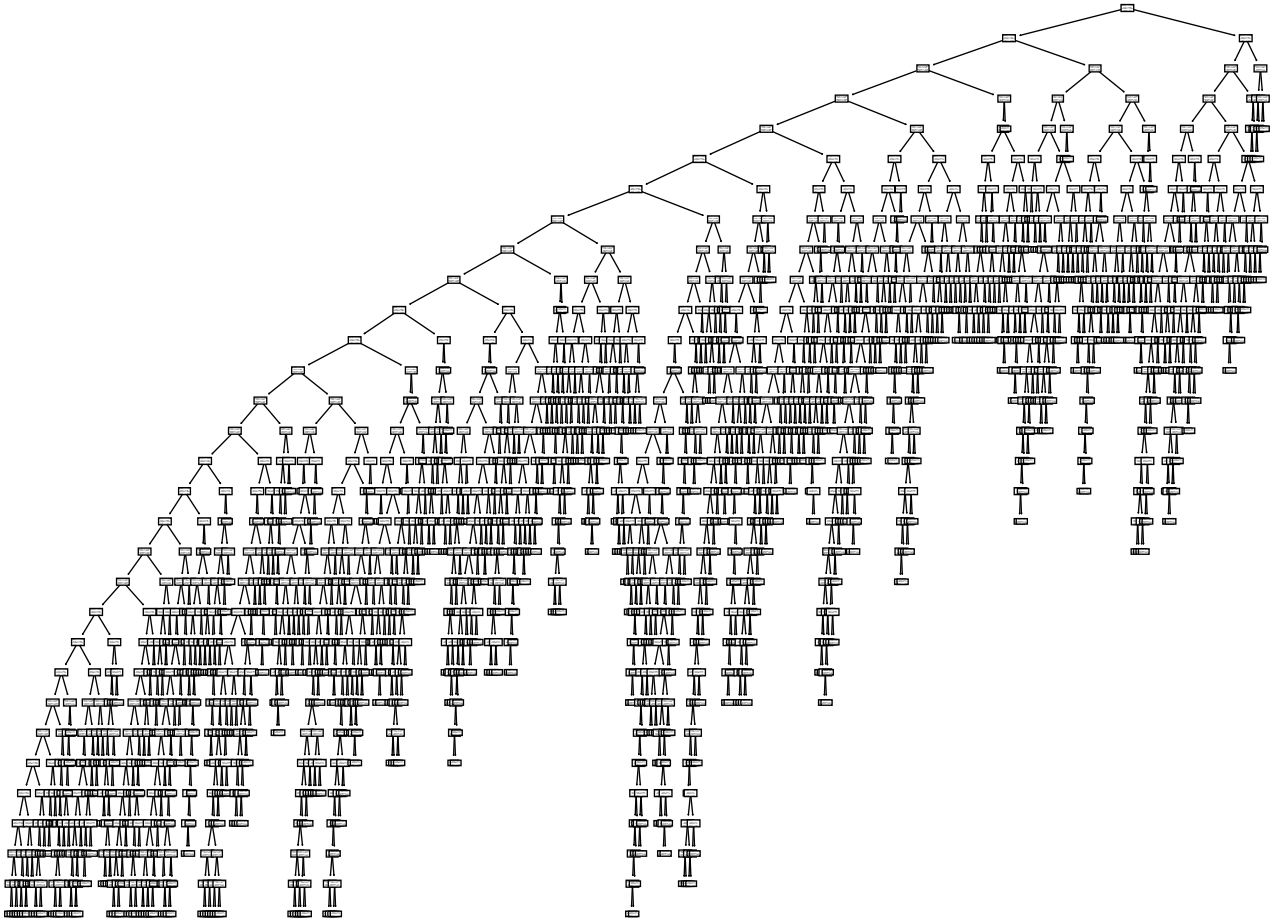

Figure 12: Directly fitted CART tree using the free embedding as a ground truth (on 20 newsgroup). Input is the original tf-idf features. Compare this tree with fig. 7–8.

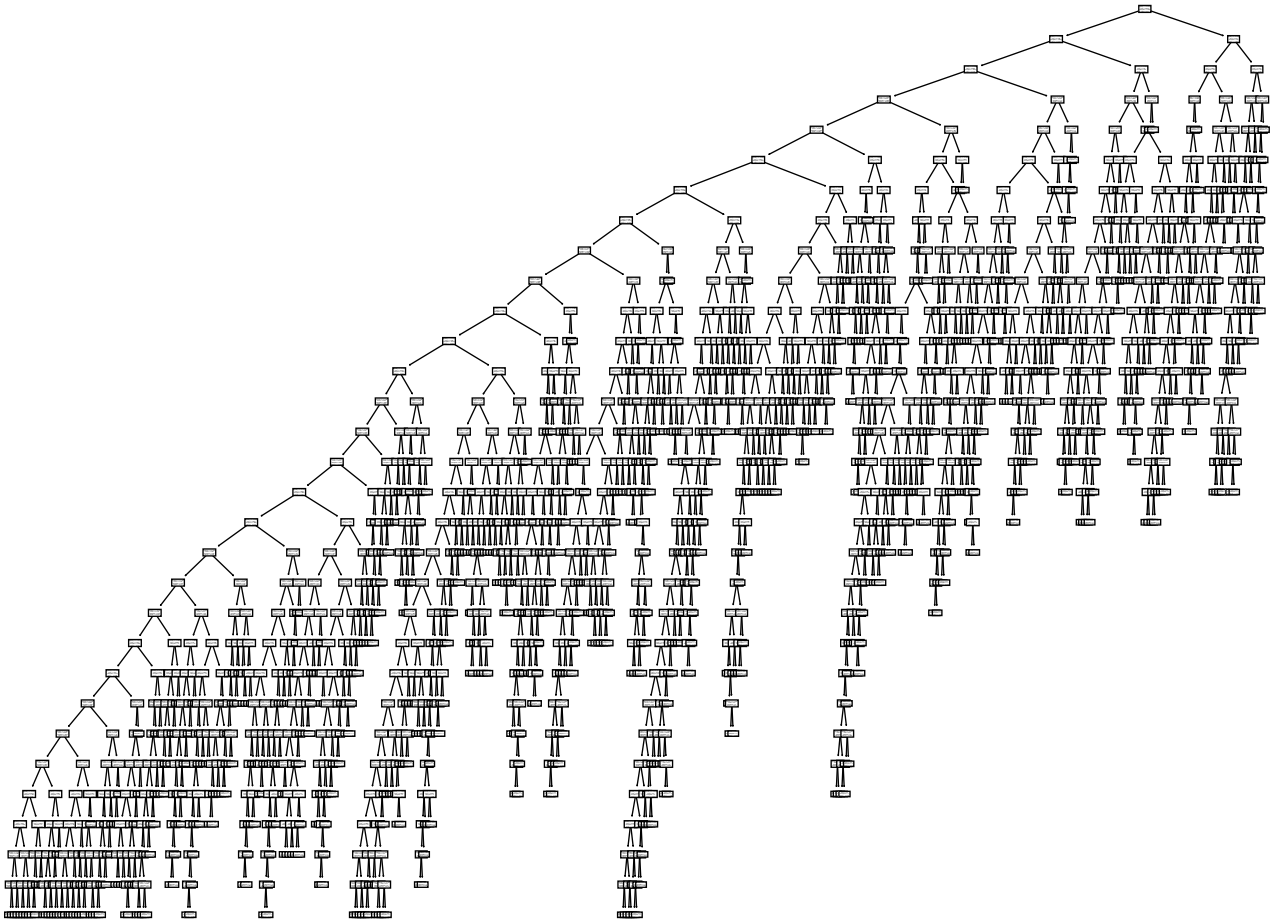

Figure 13: Similar to fig. 12 but for the second dimension. In multi-output regression, scikit-learn generates a separate tree for each output. Since we project points in 2D, we have 2 CART trees (this and in fig. 12).