Hierarchical data visualization via PCA trees

Miguel Á. Carreira-Perpiñán
Dept. CSE, UC Merced
mcarreira-perpinan@ucmerced.edu

Kuat Gazizov
Dept. CSE, UC Merced
kgazizov@ucmerced.edu

As a form of exploratory data analysis, dimensionality reduction (DR) for visualization seeks to provide a representation of a dataset in dimension at most 3 (we will focus on 2D) that, combined with the human visual ability to pick patterns, is able to find structure of interest in a high-dimensional dataset. Many DR methods have been proposed over the years. Currently, t-SNE [4] and PCA are the most popular choices and have seen widespread application in many areas.

We propose a new model for dimensionality reduction, the PCA tree, which works like a regular autoencoder, having explicit projection and reconstruction mappings. The projection is effected by a sparse oblique tree, having hard, hyperplane splits using few features and linear leaves. The reconstruction mapping is a set of local linear mappings. The PCA tree provides significant, complementary advantages over previous methods: 1) They optimize the reconstruction error, which has a clear meaning. They do not require a neighborhood graph (and perplexity parameter, etc.), which is tricky to estimate so it captures manifold structure, and computationally very costly; 2) PCA trees define nonlinear out-of-sample mappings; 3) PCA trees are highly interpretable and, as demonstrated in fig. 1, they extract a wealth of information from complex datasets. This comes from the fact that both trees and PCA are techniques with well understood interpretability; 4) PCA map shows clusters, these are real—in contrast with t-SNE's tendency to create false clusters [2]; 5) A final advantage is that the loss function is really a self-supervised regression problem. This makes it possible to use cross-validation to determine the hyperparameters.

Definition of the PCA tree as an autoencoder Like any autoencoder, the PCA tree defines an encoder $\mathbf{F}$ and decoder $\mathbf{f}$, but in a peculiar way, as follows. Firstly, consider a fixed rooted directed tree structure with decision nodes and leaves indexed by sets $\mathcal{D}$ and $\mathcal{L}$, respectively, and $\mathcal{N} = \mathcal{D} \cup \mathcal{L}$. Both the encoder and autoencoder use this tree structure. Each decision node $i \in \mathcal{D}$ has a decision function $h_i(\mathbf{x}; \mathbf{w}_i, w_{i0})$: $\mathbb{R}^D \rightarrow \mathcal{C}_i$, where $\mathcal{C}_i = \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$, sending instance $\mathbf{x}$ to the corresponding child of i. We use oblique trees, having hyperplane decision functions "go to right if $\mathbf{w}_i^T \mathbf{x} + w_{i0} \geq 0$", with $\mathbf{w}_i \in \mathbb{R}^D$ and $w_{i0} \in \mathbb{R}$. The tree's prediction for an instance $\mathbf{x}$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and applying this leaf's predictor (defined below). We define the *reduced set (RS)* $\mathcal{R}_i \subset \{1, \dots, N\}$ of a node $i \in \mathcal{N}$ as the training instances that reach i given the current tree parameters, where N is the number of training samples.

Encoder $\mathbf{F}$ This is given by a tree mapping $\mathbf{T}^e(\mathbf{x}; \boldsymbol{\Theta})$: $\mathbb{R}^D \rightarrow \mathcal{L} \times \mathbb{R}^L$ where the predictor for leaf j has the form of a linear mapping $\mathbf{F}_j(\mathbf{x}; \mathbf{U}_j, \boldsymbol{\mu}_j) = \mathbf{U}_j^T(\mathbf{x} - \boldsymbol{\mu}_j)$, where $\mathbf{U}_j \in \mathbb{R}^{D \times L}$ is an orthogonal matrix and $\boldsymbol{\mu}_j \in \mathbb{R}^D$. The encoder parameters are $\boldsymbol{\Theta} = \{\mathbf{w}_i, w_{i0}\}_{i \in \mathcal{D}} \cup \{\mathbf{U}_j, \boldsymbol{\mu}_j\}_{j \in \mathcal{L}}$. Thus, the encoder maps an input instance $\mathbf{x} \in \mathbb{R}^D$ to a leaf index $j \in \mathcal{L}$ and an L-dimensional real vector $\mathbf{z} = \mathbf{U}_j^T(\mathbf{x} - \boldsymbol{\mu}_j)$, which at an optimum will be the PCA projection in that leaf. This means that the PCA tree does not have a common latent space of dimension L where all instances are projected. Instead, it has one separate L-dimensional PCA space per leaf.

Decoder $\mathbf{f}$ This maps a leaf index j and L-dimensional vector $\mathbf{z}$ (in $\mathcal{L} \times \mathbb{R}^L$) to a vector in $\mathbb{R}^D$. It consists of a set of linear mappings of the form $\mathbf{f}_j(\mathbf{z}; \mathbf{U}_j, \boldsymbol{\mu}_j) = \mathbf{U}_j \mathbf{z} + \boldsymbol{\mu}_j$ for $j \in \mathcal{L}$.

Tree autoencoder We can now define the autoencoder as the composition of the decoder and encoder, $\mathbf{T} = \mathbf{f} \circ \mathbf{F}$. Conveniently, we can absorb the leaf index implicitly into each leaf. Thus, $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ is simply a regression tree identical to the tree encoder $\mathbf{T}^e(\mathbf{x}; \boldsymbol{\Theta})$ but where the predictor mapping of leaf j has the form $\mathbf{g}_j(\mathbf{x}; \mathbf{U}_j, \boldsymbol{\mu}_j) = \mathbf{U}_j \mathbf{U}_j^T(\mathbf{x} - \boldsymbol{\mu}_j) + \boldsymbol{\mu}_j$. Note this is a linear mapping of rank $L < D$, and it is the composition $\mathbf{f}_j \circ \mathbf{F}$ of leaf j's decoder with the encoder.

Bay Area Machine Learning Symposium (BayLearn 2024).

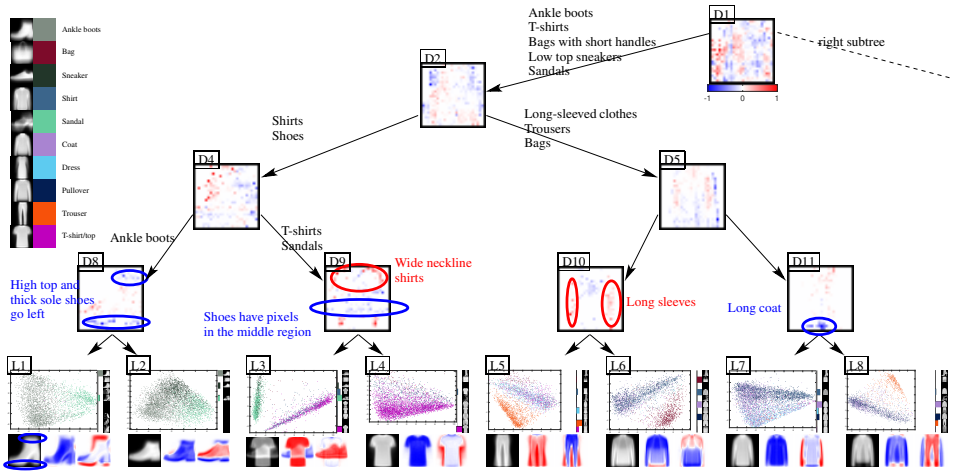

Figure 1: PCA tree trained on the Fashion MNIST dataset. The decision nodes' weight vectors (and the leaves' PCs $\mathbf{U}_j$) are shown as 28×28 images, with negative/zero/positive values colored blue/white/red, respectively. Each leaf shows a 2D PCA scatterplot of its RS (instances reaching it), and below it the mean $\boldsymbol{\mu}_j$ as a grayscale image and the 2 PCs $\mathbf{U}_j$ as color images. To the right of the scatterplot, a bar chart displays class proportions and class means. The legend (top left) shows, for each class, its mean (grayscale image), color (for the scatterplots) and description.

Optimization algorithm and time complexity We borrow the idea of alternating optimization over the tree nodes from [1]. Essentially, it repeatedly updates the nodes in turn: at a leaf it solves a PCA, and at a decision node it solves an ℓ_1-regularized, logistic regression problem. The cost of optimization over decision nodes is as in the sparse oblique classification trees of [1], which is $\mathcal{O}(ND\Delta)$, where Δ is the tree depth. For the leaves, as Δ increases, RS becomes smaller, allowing us to use the Gram matrix instead of the covariance matrix. The overall result is that the cost is at most linear in N and quadratic in D, just like in regular PCA. As a useful summary, *the rough cost is $\Theta(ND^2)$ for shallow trees and $\mathcal{O}(ND)$ for deep trees*—which is asymptotically faster than PCA! (although very deep trees having few instances per leaf are likely not practical). This makes the algorithm highly scalable to large sample sizes without the need for any approximation as shown in Fig. 2. This is a significant advantage over neighbor embedding algorithms such as t-SNE, which have a quadratic cost on N and require some approximation to reduce this [3, 5, 6].

Visualization of PCA tree We show results for the Fashion MNIST dataset (fig. 1). Even at a glance we can see lots of structure in the tree. We do not show the whole tree for readability. Although our algorithm is unsupervised, it has the ability to separate different types of classes. For example, leaves L1 and L2 contain different types of shoes. The coordinate axes in the leaves show the most variance in the data within the leaf's region. The decision nodes' weight vectors select features sparsely, often focusing on parts of the image that can differentiate groups of images and send them selectively to the left or right subtree. The figure is annotated in some places to make this clear. For example, decision node D2 separates clothes "with" and "without" sleeves. In some cases, this results in the tree separating actual classes, as in leaves L1 and L4. Since each leaf applies 2D PCA on its corresponding reduced set, its linear projection scatterplot shows the most variance directions in the leaf region. As expected with PCA, amplified by the fact that it focuses on a region of the whole data, this often shows insights about the intrinsic local structure of the data. In particular, we can usually identify the coordinate axes (PCs) with meaningful quantities or degrees of freedom. For example, leaves L1, L2, L4, L7 contain objects of a specific type, such as L1 containing high-heel and high-ankle boots and sandals. Interestingly, we find all these leaves often detect similar degrees of freedom: usually, PC1 is (overall) pixel intensity in the image and PC2 is object size (either height or width). Again, this is the result of minimizing reconstruction error, which projects along maximum-variance directions.

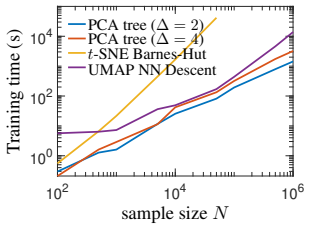

Figure 2: Training time per iteration on Infinite MNIST for PCA trees with different N, Δ, and for t-SNE and UMAP.

References

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

[2] D. Kobak and P. Berens. The art of using t-SNE for single-cell transcriptomics. *Nature Communications*, 10(5416), 2019.

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

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

[5] M. Vladymyrov and M. Á. Carreira-Perpiñán. Linear-time training of nonlinear low-dimensional embeddings. In S. Kaski and J. Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 968–977, Reykjavik, Iceland, Apr. 22–25 2014.

[6] Z. Yang, J. Peltonen, and S. Kaski. Scalable optimization for neighbor embedding for visualization. In S. Dasgupta and D. McAllester, editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 127–135, Atlanta, GA, June 16–21 2013.