Journal of Machine Learning Research 18 (2018) 1-6 Submitted 7/17; Revised 10/17; Published 4/18

HyperTools: a Python Toolbox for Gaining Geometric Insights into High-Dimensional Data

Andrew C. Heusser† andrew.c.heusser@dartmouth.edu
Kirsten Ziman† kirsten.k.ziman.gr@dartmouth.edu
Lucy L. W. Owen lucy.w.owen.gr@dartmouth.edu
Jeremy R. Manning jeremy.r.manning@dartmouth.edu
Department of Psychological and Brain Sciences, Dartmouth College, Hanover, NH 03755, USA
†*Denotes equal contribution*

Editor: Alexandre Gramfort

Abstract

Dimensionality reduction algorithms have played a foundational role in facilitating the deep understanding of complex high-dimensional data. One particularly useful application of dimensionality reduction techniques is in data visualization. Low-dimensional visualizations can help practitioners understand where machine learning algorithms might leverage the geometric properties of a dataset to improve performance. Another challenge is to generalize insights across datasets [e.g. data from multiple modalities describing the same system (Haxby et al., 2011), artwork or photographs of similar content in different styles (Zhu et al., 2017), etc.]. Several recently developed techniques (e.g. Haxby et al., 2011; Chen et al., 2015) use the procrustean transformation (Schönemann, 1966) to align the geometries of two or more spaces so that data with different axes may be plotted in a common space. We propose that each of these techniques (dimensionality reduction, alignment, and visualization) applied in sequence should be cast as a single conceptual *hyperplot* operation for gaining geometric insights into high-dimensional data. Our Python toolbox enables this operation in a single (highly flexible) function call.

Keywords: visualization, high-dimensional, dimensionality reduction, procrustes, time-series data

1. Introduction

A major focus of machine learning research is finding patterns in, and making predictions from, complex data that might be otherwise inscrutable to a human viewer. Paradoxically, designing effective algorithms for discovering, characterizing, and leveraging structure in many complex datasets often requires the practitioner to have some initial intuitions about the structure of the data. One commonly used approach for discovering structure in high-dimensional data is to use dimensionality reduction algorithms (e.g. Pearson, 1901; Tipping and Bishop, 1999; Jutten and Herault, 1991; Comon et al., 1991; Torgerson, 1958; van der Maaten and Hinton, 2008) to visualize high-dimensional data in two or three dimensions, thereby providing insights into geometric patterns that pervade the data (Uddenberg et al., 2016). Despite the so-called "curse of dimensionality," whereby high-dimensional data can behave differently from low-dimensional data, visualizing data via low-dimensional

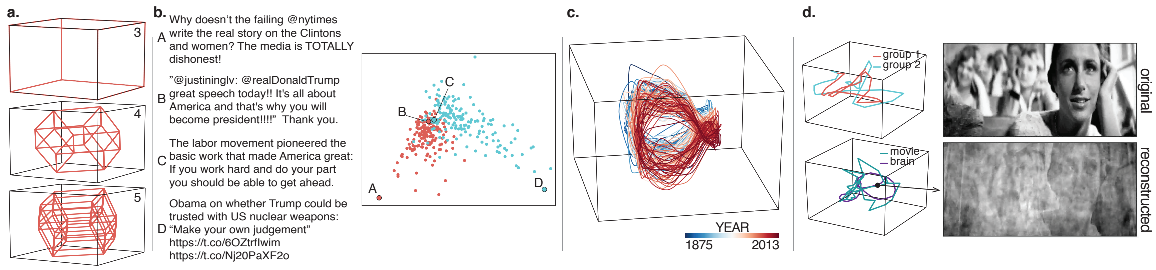

Figure 1: **Data visualization examples. a.** Hypercubes of increasing dimensionality (3, 4 and 5 dimensions). The cubes provide some insights into which aspects of high-dimensional data are preserved (or distorted) when projected onto 3 dimensions. **b.** Topic modeling (Blei et al., 2003) of political Twitter data: 2D projections of topic vectors of Hillary Clinton's (blue) and Donald Trump's (red) tweets during their 2016 presidential campaigns (link to data). The 'V' shape highlights that Trump and Clinton tweets are largely about fundamentally different topics. The highlighted examples include (A) a Trump tweet that is especially "Trump-like," (B) a Trump tweet that is Clinton-like, (C) a Clinton tweet that is Trump-like, and (D) a Clinton tweet that is especially Clinton-like. **c.** Changing temperatures across the Earth's surface from 1875—2013 (link to data). The visualization highlights the cyclical (seasonal) nature of global temperatures that occurs alongside a gradual increase in global temperatures over time. **d.** Brain/movie trajectories during movie viewing (link to data). (Top left) Group-averaged trajectories of brain activity from the ventral visual cortex, split into two randomly-selected groups of subjects (group 1: $n = 6$, group 2: $n = 5$) watching *Raiders of the Lost Ark* (Haxby et al., 2011). (Bottom left) Group-averaged trajectory of brain activity from ventral visual cortex and trajectory of the movie frames (pixel intensities over time), hyperaligned to a common space. (Right) Movie frame reconstructed from the group-averaged brain activity that was aligned to movie space. The example illustrates geometric commonalities across brains, and between the movie frames and the brain responses to those frames.

projections can provide a glimpse (though imperfect) into what the dataset "looks like." By leveraging different dimensionality reduction algorithms, or by viewing the data through different projections (or as an animation), one can begin to form intuitions that generalize beyond the specific imperfections of a given dimensionality reduction tool or projection. We highlight several examples of how low-dimensional projections may be used to understand the geometry of high-dimensional data in Figure 1. Complete details may be found here, and sample IPython notebooks may be found here.

Although dimensionality reduction algorithms provide a useful means of visualizing high-dimensional data, they cannot (by themselves) solve the problem of identifying geometric similarities across different types of data. For example, as highlighted in Figure 1d, different people's brains might exhibit similar (but not identical) activity patterns while watching a movie, and those brain patterns might exhibit a similar temporal covariance structure to the movie itself. However, if one were to project brain data and movie data onto a three dimensional space, although the overall "shapes" of those datasets might be similar, there would be no inherent reason for the datasets to align. Algorithms inspired by the procrustean transformation (Schönemann, 1966), including Hyperalignment (Haxby et al., 2011) and the Shared Response Model (Chen et al., 2015) compute the affine transformations of two or more datasets that bring the data into a common alignment. This can reveal similarities (or differences) between the underlying geometries of otherwise incompatible data. Further, because the transformation is invertible, one can map between any point in that common space (e.g. a shared movie-brain space) and the original data spaces (e.g. the movie frames; the lower right panel of Fig. 1d shows a movie frame corresponding to a brain pattern recorded during one time in the movie, and the upper panel shows the original movie frame viewed at that moment).

The `HyperTools` toolbox (current version: 0.4.2) provides a powerful set of Python functions for projecting high dimensional data onto lower-dimensional spaces, aligning data of different types, and visualizing the results in publication-quality figures and movies. A central goal of the toolbox is to cast dimensionality reduction, alignment, and visualization as a single highly flexible "hyperplot" command:

```
import hypertools as hyp                                                    (1)
hyp.plot(list_of_arrays, reduce='TSNE', align='hyper', ndims=3)             (2)
```

This example function call projects the high-dimensional data onto 3 dimensions using the t-SNE algorithm, aligns the data matrices in the given list of arrays into a common space using hyperalignment, and produces a 3D plot analogous to those shown in Figure 1. These hyperplot visualizations provide an intuitive means of understanding how different observations relate to each other or how those observations change over time. These insights can guide algorithm design decisions, or help practitioners to understand which aspects of the data may be easy (or difficult) to model or measure.

2. Overview of the Toolbox

`HyperTools` is open-source, installable from GitHub or pip (`pip install hypertools`), and is distributed with the MIT License. The toolbox depends on the following open-source software packages: `Matplotlib` (Hunter, 2007) for plotting functionality, `Seaborn` (Waskom et al., 2016) for plot styling, `Scikit-learn` (Pedregosa et al., 2011) for data analysis (dimensionality reduction, clustering, etc.), and `PPCA` for inferring missing data (Tipping and Bishop, 1999). The toolbox also includes a port of the hyperalignment algorithm (Haxby et al., 2011) from the `PyMVPA` library, as well as the shared response model from the `BrainIAK` toolbox (Capota et al., 2017), as an alternative data alignment technique. At the time of writing this manuscript, the toolbox incorporates two general classes of algorithms: *dimensionality reduction* (PCA, incremental PCA, sparse PCA, kernel PCA, probabilistic PCA, t-SNE, MDS, ICA, factor analysis, truncated SVD, dictionary learning, mini-batch dictionary learning, isomap, spectral embedding, and local linear embedding) and *alignment* (hyperalignment, shared response model, and procrustean transformation). `HyperTools` provides a simple interface to these functions, with support for a variety of convenient data formats including `NumPy` arrays (van der Walt et al., 2011), `Pandas` dataframes (McKinney, 2010), or mixed lists of arrays and dataframes. [Each to-be-analyzed (or to-be visualized) dataset must be formated as a number-of-observations (S) by number-of-features (F) array or dataframe.] `HyperTools` also adds a number of custom arguments to facilitate data visualization and manipulation of high-dimensional data. Nearly all of the `HyperTools` functions may be accessed through the `plot` function (Ex. 2). This design enables complex data analysis and plotting to be carried out in a single function call. The same function call also returns the analyzed data (and the transformations applied to it) for potential follow-up analyses. There are two general types of plots supported by the toolbox: *static plots* and *animated plots*.

2.1 Static Plots

By default, the `plot` function will perform dimensionality reduction (using incremental PCA), converting the $S \times F$ data matrix (or matrices) into an $S \times 3$ matrix (or matrices), and then create an interactive 3D line plot that can be explored by using the mouse to rotate the plot. If there are `NaNs` present in the dataset, these missing values will be automatically interpolated using PPCA (Tipping and Bishop, 1999) (preserving the dimensionality of the original data) prior to reducing the dimensionality of the data using the user-specified algorithm. The `plot` function also accepts format strings to specify line styling (following `Matplotlib` conventions):

```
hyp.plot(data)                                  (3)
hyp.plot([data1, data2, data3], '.')            (4)
```

2.2 Animated Trajectory Plots

Animated 3D plots (created using the `animate` flag) are useful for visualizing high-dimensional timeseries data:

```
hyp.plot(data, animate=True).                   (5)
```

This function call creates a 3D trajectory animated over the rows of the `data` matrix. Each frame of the animation displays a portion of the total data trajectory enclosed in a cube. In successive frames, the displayed portion of the data trajectory incrementally advances, and the camera angle rotates around the cube, providing visual access to different aspects of the data as the animation progresses.

2.3 Align

Two or more datasets may share geometric structure, but reside in different coordinate systems. The `align` function accepts a list of arrays as input and returns a hyperaligned list of arrays in a common geometric space:

```
hyp.plot([array1, array2, array3], align='hyper')             (6)
hyperaligned_list = hyp.align([array1, array2, array3])       (7)
```

2.4 Reduce

Users can access a variety of dimensionality reduction algorithms by passing the `reduce` keyword argument to the `plot` function. We also provide API access to the `reduce` function that underlies these transformations. At its core, the `reduce` function wraps many of `scikit-learn`'s dimensionality reduction algorithms, along with the `PPCA` library. `HyperTools` extends the functionality of these tools by providing a streamlined syntax that accepts data matrices in a larger variety of formats (e.g. `NumPy` arrays and `Pandas` dataframes, or lists of arrays and dataframes, may all be analyzed using the same syntax). The function may be used as follows:

```
hyp.plot(data, reduce='MDS')                                    (8)
reduced_data = hyp.reduce(data, reduce='TSNE', ndims=5)         (9)
```

3. Concluding Remarks

The `HyperTools` toolbox is designed to provide geometric insights into high-dimensional datasets with a single function call. We also provide a convenient syntax to access a variety of dimensionality reduction and data alignment algorithms. The above overview of the toolbox highlights its primary features; additional functionality and a detailed description of the API may be found here.

Acknowledgments

We are grateful to Luke J. Chang and Matthijs van der Meer for useful discussions, and to J. Swaroop Guntapalli for help implementing our `align` function. We also thank Ryan Arredondo, Aly Sivji and Steph Wright for their code contributions during the Mozilla Global Sprint in June 2017, and we thank Mozilla for organizing the Global Sprint. We also thank Chase Williams for contributing code. Our work was supported in part by NSF EPSCoR Award Number 1632738. The content is solely the responsibility of the authors and does not necessarily represent the official views of our supporting organizations.

References

D M Blei, A Y Ng, and M I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993 – 1022, 2003.

M Capota, J Turek, P-H Chen, X Zhu, J R Manning, N Sundaram, B Keller, Y Wang, and Y S Shin. Brain imaging analysis kit, 2017. URL `https://doi.org/10.5281/zenodo.59780`.

P-H Chen, J Chen, Y Yeshurun, U Hasson, J Haxby, and P J Ramadge. A Reduced-Dimension fMRI Shared Response Model. In C. Cortes and N. D. Lawrence and D. D. Lee and M. Sugiyama and R. Garnett, editor, *Advances in Neural Information Processing Systems 28*, pages 460–468. Curran Associates, Inc., 2015.

P Comon, C Jutten, and J Herault. Blind separation of sources, part II: Problems statement. *Signal Processing*, 24(1):11 – 20, 1991.

J V Haxby, J S Guntupalli, A C Connolly, Y O Halchenko, B R Conroy, M I Gobbini, M Hanke, and P J Ramadge. A common, high-dimensional model of the representational space in human ventral temporal cortex. *Neuron*, 72:404–416, 2011.

J D Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

C Jutten and J Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991.

W McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

K Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2:559–572, 1901.

F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P Schönemann. A generalized solution of the orthogonal Procrustes problem. *Psychometrika*, 31:1–10, 1966.

M E Tipping and C M Bishop. Probabilistic principal component analysis. *Journal of Royal Statistical Society, Series B*, 61(3):611–622, 1999.

W S Torgerson. *Theory and methods of scaling*. Wiley, New York, 1958.

S Uddenberg, G Newman, and B Scholl. Perceptual averaging of scientific data: Implications of ensemble representations for the perception of patterns in graphs. *Journal of Vision*, 16(12):1081, 2016.

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

S van der Walt, S C Colbert, and G Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13:22–30, 2011.

M Waskom, O Botvinnik, D Okane, P Hobson, David, Y Halchenko, S Lukauskas, J B Cole, J Warmenhoven, J de Ruiter, S Hoyer, J Vanderplas, S Villalba, G Kunter, E Quintero, M Martin, A Miles, K Meyer, T Augspurger, T Yarkoni, P Bachant, M Williams, C Evans, C Fitzgerald, Brian, D Wehner, G Hitz, E Ziegler, A Qalieh, and A Lee. Seaborn: v0.7.1, 2016. URL `https://doi.org/10.5281/zenodo.54844`.

J-Y Zhu, T Park, P Isola, and A A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv*, 1703.10593, 2017.