Distance-based dimensionality reduction for big data literature review

Adrià Casanova Lloveras

April 3, 2025

Contents

1 Rdimtools: An R Package for Dimension Reduction and Intrinsic Dimension Estimation [25]

1.1 Abstract

Original: Discovering patterns of the complex high-dimensional data is one of the fundamental pillars of modern data science. Dimension reduction and intrinsic dimension estimation are two thematic programs that facilitate geometric characterization of the data. We present `Rdimtools`, an R package that supports 143 dimension reduction and manifold learning methods and 17 dimension estimation algorithms whose unprecedented extent makes multifaceted scrutiny of the data in one place easier. `Rdimtools` is distributed under the MIT license and is accessible from CRAN, GitHub, and its package website, all of which deliver instruction for installation, self-contained examples, and API documentation.

Apple Intelligence summary: `Rdimtools`, an R package, supports 160 dimension reduction and manifold learning methods, making data analysis easier. It is available on CRAN, GitHub, and its package website.

1.2 Key Points

- **R package**: that supports 143 dimension reduction and manifold learning methods and 17 dimension estimation algorithms.
- **Other libraries**: `drtoolbox` in MATLAB, `scikit-learn` in Python, a C++ template library `tapkee` with a known basis of popularity. In R, packages `dimRed`, `dyndimred`, `intrinsicDimension`.
- **Implementation**: mixture of R and C++ that are integrated by `Rcpp`. For numerical operations, `RcppArmadillo` is heavily used to take advantage of `Armadillo` C++ linear algebra library.
- **3 function families**: `do.{algorithm}`, `est.{algorithm}` and `aux.{algorithm}` for DR, IDE, and auxiliary functions.
- Downloaded 1013 times per month on average from CRAN.
- **Future plan**: support for out-of-memory execution in response to the increased needs for big data analysis.

1.3 Example R Code

```r
# Documentation example:
# do.idmap (Interactive Document Map)

library(Rdimtools)

## load iris data
data(iris)
set.seed(100)
```

```r
subid = sample(1:150,50)
X = as.matrix(iris[subid,1:4])
lab = as.factor(iris[subid,5])
## let's compare with other methods
out1 <- do.pca(X, ndim=2)
out2 <- do.lda(X, ndim=2, label=lab)
out3 <- do.idmap(X, ndim=2, engine="NNP")
## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(out1$Y, pch=19, col=lab, main="PCA")
plot(out2$Y, pch=19, col=lab, main="LDA")
plot(out3$Y, pch=19, col=lab, main="IDMAP")
par(opar)
```

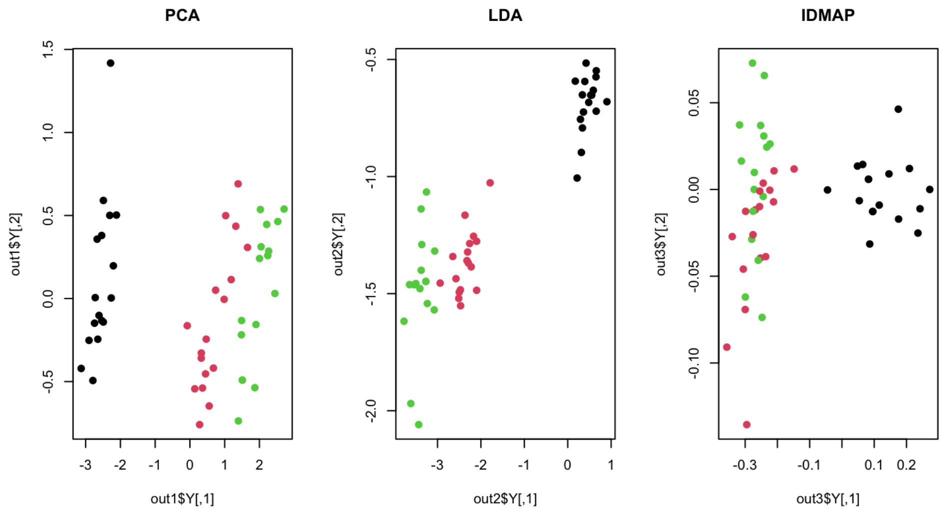


Figure 1: Rdimtools test output.

2 Global versus local methods in nonlinear dimensionality reduction [21]

2.1 Abstract

Original: Recently proposed algorithms for nonlinear dimensionality reduction fall broadly into two categories which have different advantages and disadvantages: global (Isomap), and local (Locally Linear Embedding, Laplacian Eigenmaps). We present two variants of Isomap which combine the advantages of the global approach with what have previously been exclusive advantages of local methods: computational sparsity and the ability to invert conformal maps.

Apple Intelligence summary: Two new Isomap variants are presented, combining global advantages with local computational sparsity and conformal map inversion.

2.2 Key Points

- **Introduction of LMDS** by applying it to Isomap (L-Isomap).
- ***Landmark points*** ($n << N$) reduce the complexity of computing:
  - the distances matrix with Dijkstra's algorithm with Fibonacci heaps (k = neighborhood size) from $\mathcal{O}(kN^2 \log N)$ to $\mathcal{O}(knN \log N)$.
  - MDS from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 N)$.
- If x is a landmark point, then the embedding given by LMDS is consistent with the original MDS embedding.
- If the distance matrix $D_{n,N}$ can be represented exactly by a Euclidean configuration in $\mathbb{R}^l$, and if the landmarks are chosen so that their affine span in that configuration is l-dimensional (i.e. general position), then LMDS will recover the configuration exactly, up to rotation and translation.

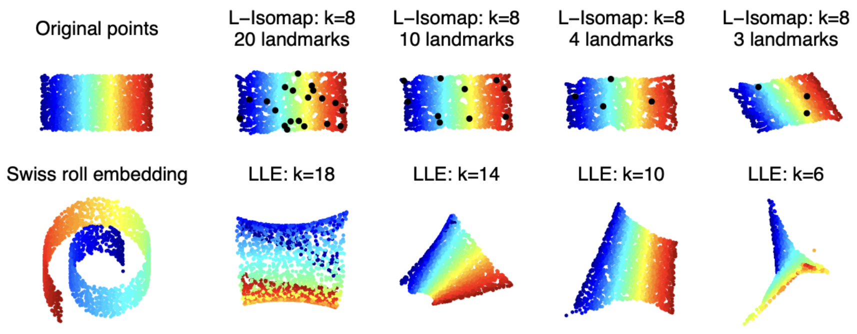


Figure 2: L-Isomap is stable over a wide range of values for the sparseness parameter (the number of landmarks). Results from LLE are shown for comparision[21].

3 `dimRed` and `coRanking` - Unifying Dimensionality Reduction in R [6]

3.1 Abstract

Original: "Dimensionality reduction" (DR) is a widely used approach to find low dimensional and interpretable representations of data that are natively embedded in high-dimensional spaces. DR can be realized by a plethora of methods with different properties, objectives, and, hence, (dis)advantages. The resulting low-dimensional data embeddings are often difficult to compare with objective criteria. Here, we introduce the `dimRed` and `coRanking` packages for the R language. These open source software packages enable users to easily access multiple classical and advanced DR methods using a common interface. The packages also provide quality indicators for the embeddings and easy visualization of high dimensional data. The `coRanking` package provides the functionality for assessing DR methods in the co-ranking matrix framework. In tandem, these packages allow for uncovering complex structures high dimensional data. Currently 15 DR methods are available in the package, some of which were not previously available to R users. Here, we outline the `dimRed` and `coRanking` packages and make the implemented methods understandable to the interested reader.

Apple Intelligence summary: Dimensionality reduction (DR) methods create low dimensional data representations, but comparison is challenging. The `dimRed` (figure 3) and `coRanking` R packages are introduced to address this.

3.2 Key Points

- The difficulty in applying DR is that each DR method is designed to maintain certain aspects of the original data and therefore may be appropriate for one task and inappropriate for another. Most methods also have parameters to tune and follow different assumptions.
- **Software packages for other languages**:
  - Python: `scikit-learn`, which contains a module for DR.
  - Julia: `ManifoldLearning.jl` for nonlinear and `MultivariateStats.jl` for linear DR.
  - Matlab: several toolboxes.
  - C++: `Shogun` toolbox, which offers bindings for many high level languages (including R).
- At the time (2018), no comprehensive package for R.
- None of the former provides means to consistently compare the quality of different methods.
- MDS can be seen as kPCA with kernel $x^T y$, since a distance matrix can be transformed to a matrix of inner products.
- `dimRed` wraps `cmdscale`.

- In contrast to a supervised problem, there is no natural way to directly measure the quality of any output or to compare two methods. Every method optimizes a different error function.

- **Quality criteria implemented in** `coRanking`:

  - **Co-ranking matrix based measures**: the co-ranking matrix Q is the 2d-histogram of the distance ranks. q_{ij} is an integer which counts how many points of distance rank j became rank i. In a perfect DR, this matrix will only have non-zero entries in the diagonal. In R, the co-ranking matrix can be calculated using the the `coRanking::coranking` function. The `dimRed` package contains the functions `Q_local, Q_global, Q_NX, LCMC`, and `R_NX` to calculate the above quality measures in addition to `AUC_lnK_R_NX`. If R_{NX} is high for low values of K, then local neighborhoods are maintained well; if R_{NX} is high for large values of K, then global gradients are maintained well (see figure 4).
  - Cophenetic correlation.
  - **Reconstruction error**: the fairest one when the method provides an inverse mapping. RMSE $= \sqrt{\frac{1}{n}\sum_{i=1}^{n} d\left(x_i', x_i\right)^2}$, with $x_i' = f^{-1}(y_i) = f^{-1}(f(x_i))$.

- **Test datasets**: Common ones being the 3d S-curve and the Swiss roll. Real world examples usually have more dimensions and often are much noisier and we cannot be sure if we can observe all the relevant variables. Can be retrieved with `dimRed::loadDataSet`

- **Main functions:** `embed, quality, plot, plot_R_NX`.

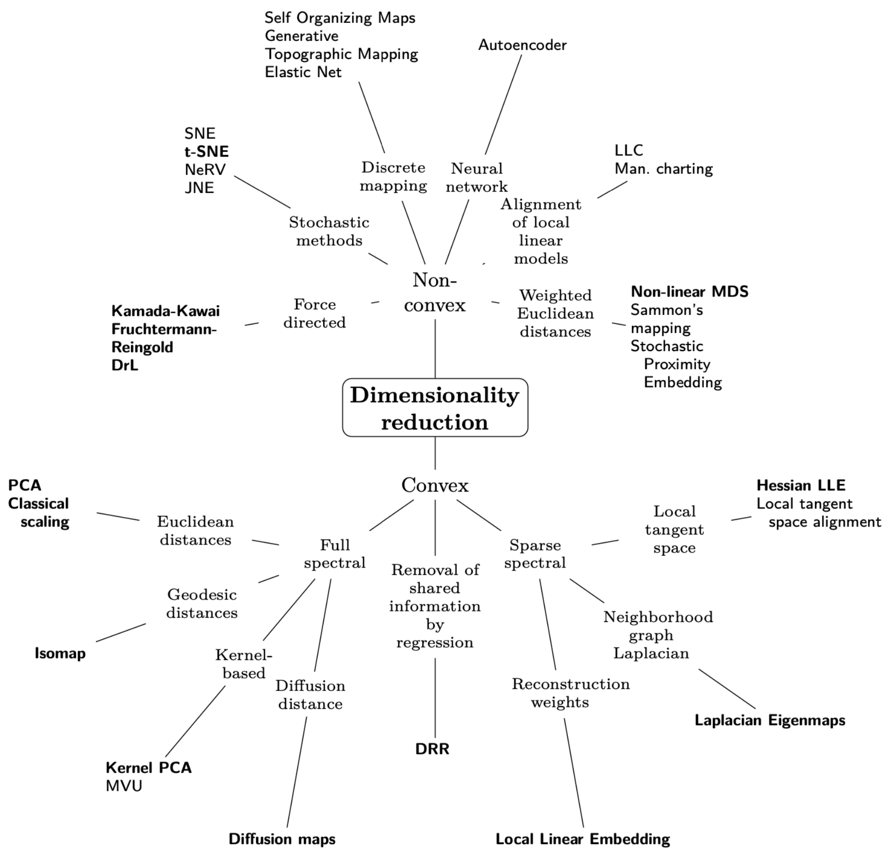


Figure 3: DR methods implemented in `dimRed` [6].

4 Sparse multidimensional scaling using landmark points [3]

4.1 Abstract

Original: In this paper, we discuss a computationally efficient approximation to the classical multidimensional scaling (MDS) algorithm, called Landmark MDS (LMDS), for use when the number of data points is very large. The first step of the algorithm is to run classical MDS to embed a chosen subset of the data, referred to as the 'landmark points', in a low-dimensional space. Each remaining data point can be located within this space given knowledge of its distances to the landmark points. We give an elementary and explicit theoretical analysis of this procedure, and demonstrate with examples that LMDS is effective in practical use.

Apple Intelligence summary: Landmark MDS approximates classical multidimensional scaling for large datasets. It embeds a subset of data points, called "landmark points", in a low-dimensional space, then locates remaining points based on their distances to these landmarks.

4.2 Key Points

- **LMDS**: Landmark MDS.

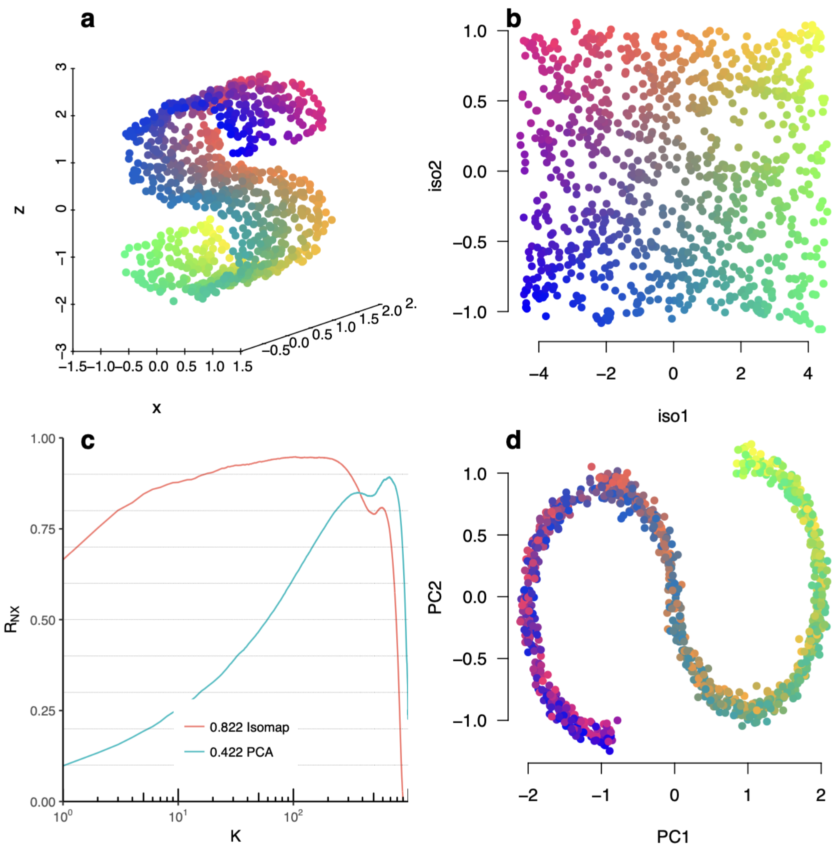


Figure 4: R_{NX} measures the quality of the embedding. [6].

- **Method**:
  1. Select n landmark points from N data points.
  2. Apply MDS to the $n \times n$ distance matrix to obrain L.
  3. Embed remaining points via distance-based triangulation.
- **Complexity**: Classical MDS: $\mathcal{O}(N^2)$ storage, $\mathcal{O}(N^3)$ time. LMDS: $\mathcal{O}(nN)$ storage, lower time complexity.
- It has links with Isomap (L-Isomap), the Nyström method (which finds approximate solutions to a positive semi-definite symmetric eigenvalue problem using just a few of the columns of the matrix) and FastMap.

5 Comparative study for dimensionality reduction techniques for big data [19]

5.1 Abstract

Original: Nowadays, big data represents the solution for different type of users especially enterprises due to its huge amount of information augmented in real time. All these generated data could be described in one of big data characteristics named variety. One of the most challenging issues for big data variety is high dimensionality because, it prevents the analysis process, demands heavy computations and adds noise to our data. The solution for this challenging issue is dimensionality reduction which minimize the dimensions and keeps only the essential information that give accurate results during analysis and decrease the computations complexity.

This paper aims to summarize and compare some of the recent methods that are used to solve the problem of high dimensionality in big data, Thereby, facilitating the process of research in this field.

Apple Intelligence summary: High dimensionality in big data poses challenges, requiring dimensionality reduction to minimize dimensions and retain essential information. This paper compares recent methods for solving this problem, aiding research in the field.

5.2 Key Points

- **Techniques Compared:**
  - Classical/Modified PCA (modified to handle memory limits via row-scanning and MapReduce).
  - Simplicial Nonnegative Matrix Tri-Factorization (SNMTF), an improved variant of NMF.
  - Stacked Autoencoders, which yield lower reconstruction error than PCA.
  - Linguistic Hedges Neuro-Fuzzy Classifiers with Feature Selection (LHNFCSF), a combination of neural networks and fuzzy inference systems (neuro-fuzzy) based on linguistic hedges with feature selection method.
  - Deep Belief Networks (DBNs), which consist of multiple hidden layers where each layer is an RBM (Restricted Boltzmann Machine), which is also a class of neural networks. Each RBM is connected with two layers, a hidden layer and a visible layer, and so on.

6 Out-of-Core Dimensionality Reduction for Large Data via Out-of-Sample Extensions [18]

6.1 Abstract

Original: Dimensionality reduction (DR) is a well-established approach for the visualization of high-dimensional data sets. While DR methods are often applied to typical DR benchmark data sets in the literature, they might suffer from high runtime complexity and memory requirements, making them unsuitable for large data visualization especially in environments outside of high-performance computing. To perform DR on large data sets, we propose the use of out-of-sample extensions. Such extensions allow inserting new data into existing projections, which we leverage to iteratively project data into a reference projection that consists only of a small manageable subset. This process makes it possible to perform DR out-of-core on large data, which would otherwise not be possible due to memory and runtime limitations. For metric multidimensional scaling (MDS), we contribute an implementation with out-of-sample projection capability since typical software libraries do not support it.

We provide an evaluation of the projection quality of five common DR algorithms (MDS, PCA, t-SNE, UMAP, and autoencoders) using quality metrics from the literature and analyze the trade-off between the size of the reference set and projection quality. The run-time behavior of the algorithms is also quantified with respect to reference set size, out-of-sample batch size, and dimensionality of the data sets. Furthermore, we compare the out-of-sample approach to other recently introduced DR methods, such as PaCMAP and TriMAP, which claim to handle larger data sets than traditional approaches. To showcase the usefulness of DR on this large scale, we contribute a use case where we analyze ensembles of streamlines amounting to one billion projected instances.

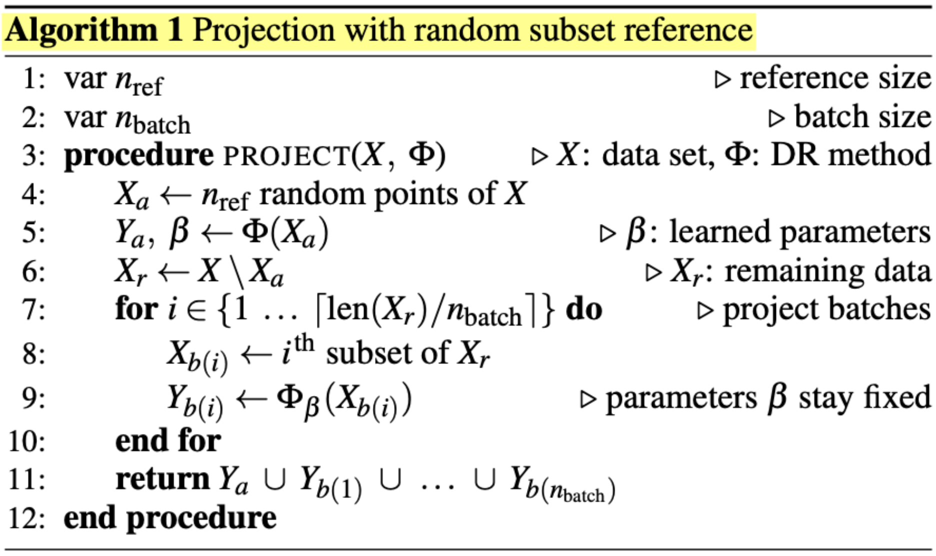
Apple Intelligence summary: The study compares out-of-sample DR methods with PaCMAP and TriMAP on large datasets. A use case demonstrates the usefulness of DR on a dataset of one billion projected streamline instances.

6.2 Key Points

- A framework for OOS extensions of DR methods similar to Interpolation MDS.
- Applied to PCA, MDS, t-SNE, UMAP and autoencoders.
- Compared to TriMap and PaCMAC, which are efficient DR methods suitable for big data.
- **Related work**:
  - **Efficient DR algorithms**: PaCMAP (up to 4 million data points), TriMAP (up to 11 million data points), LargeVis (also millions of data points, aimed to reduce computational cost of t-SNE). Note that t-SNE and UMAP fail at 1 million data points.
  - **Parametric DR methods**: learn explicit maps between the high- and low-dimensional space. Hinterreiter et al. [5] introduced a framework making parametric extensions generally available for DR techniques.

- **GPU porting**: of t-SNE, UMAP and metrics computations such as trustworthiness. Reichmann et al. also parallelize metric MDS.
- **OOS extensions**: Bengio et. al. [1] introduced an OOS framework by learning the corresponding eigenfunctions. Gisbrecht et. al. [4] proposed three kernel-based methods and [26] improved them.

- **Computational framework**: see figure 5. Details and sematics of the β parameters depends on the DR method. As an example, see figure 6.



Algorithm 1 Projection with random subset reference

```
 1: var $n_{\text{ref}}$                                  ▷ reference size
 2: var $n_{\text{batch}}$                                ▷ batch size
 3: procedure PROJECT($X$, $\Phi$)        ▷ $X$: data set, $\Phi$: DR method
 4:     $X_a \leftarrow n_{\text{ref}}$ random points of $X$
 5:     $Y_a$, $\beta \leftarrow \Phi(X_a)$                ▷ $\beta$: learned parameters
 6:     $X_r \leftarrow X \setminus X_a$                   ▷ $X_r$: remaining data
 7:     for $i \in \{1 \ldots \lceil \text{len}(X_r)/n_{\text{batch}} \rceil\}$ do   ▷ project batches
 8:         $X_{b(i)} \leftarrow i^{\text{th}}$ subset of $X_r$
 9:         $Y_{b(i)} \leftarrow \Phi_\beta(X_{b(i)})$     ▷ parameters $\beta$ stay fixed
10:     end for
11:     return $Y_a \cup Y_{b(1)} \cup \ldots \cup Y_{b(n_{\text{batch}})}$
12: end procedure
```

Figure 5: OOS projection framework proposed in [18].

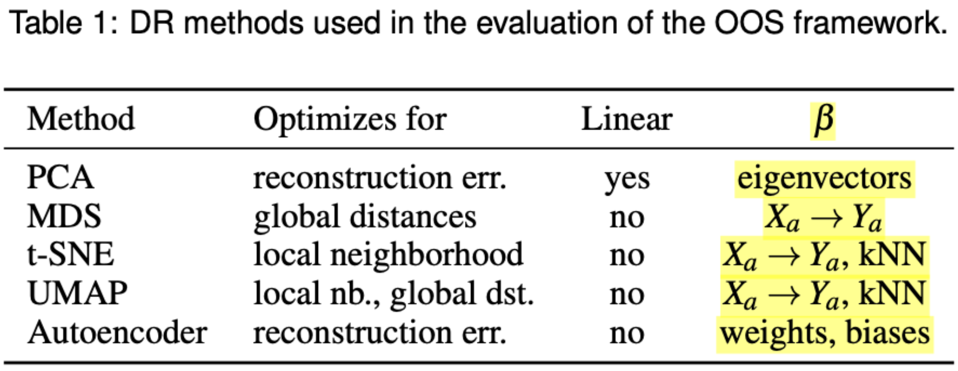


Table 1: DR methods used in the evaluation of the OOS framework.

Method	Optimizes for	Linear	β
PCA	reconstruction err.	yes	eigenvectors
MDS	global distances	no	$X_a \to Y_a$
t-SNE	local neighborhood	no	$X_a \to Y_a$, kNN
UMAP	local nb., global dst.	no	$X_a \to Y_a$, kNN
Autoencoder	reconstruction err.	no	weights, biases

Figure 6: DR methods used in the evaluation of the OOS framework proposed in [18].

- **OOS extensions** used:
  - **PCA** and the **autoencoder** learn a parametric mapping and give an explicit function to map the data.
  - For **metric MDS**, they minimize stress for a single point while keeping all others fixed (*single scaling*) to perform the OOS projection of a single point. The gradient for an OOS point x' with respect to the points x_i of the reference set X_a and corresponding low-dimensional points y_i is given as $\delta =$

$\sum_i \left(1 - d\left(x', x_i\right) / \|y' - y_i\|\right) \cdot (y' - y_i)$, with dissimilarity function $d(\cdot, \cdot)$. We have run the code provided in the article with a Windows system that has an NVIDIA graphics card and obtained the results shown in figure 7. Details can be found on `code/ReichmanHagele2024_MDS/mds_demo.py`.

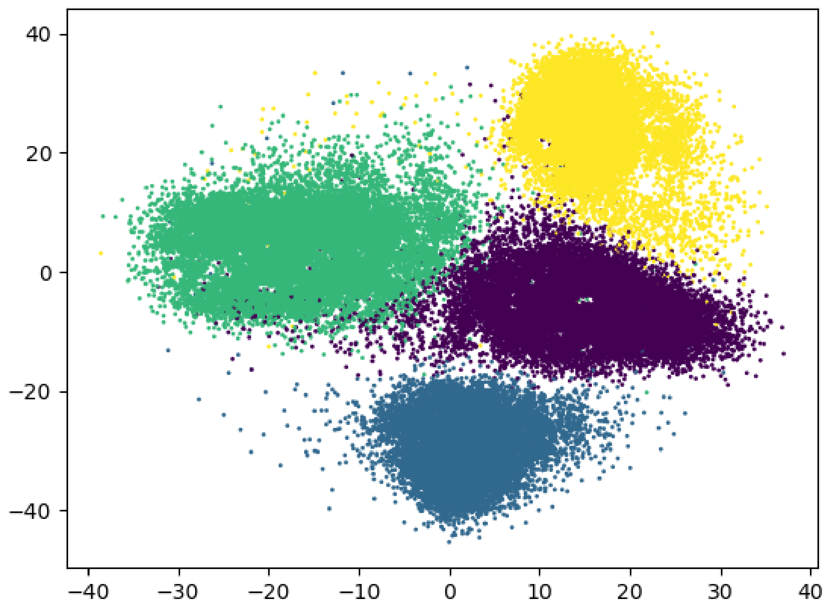


Figure 7: Output of the CUDA imeplementation in [18] of metric MDS for a test dataset.

  - A similar mechanism can be used in **t-SNE** and **UMAP**.

- **Datasets** of up to 50,000,000 data points were used to evaluate the OOS framework, as can be seen in figure 8. That being said, a non-consumer very powerful system was used; it consisted of an AMD Ryzen Threadripper PRO 3995WX with 64 CPU cores, an NVIDIA RTX A6000 GPU with 48 GiB of VRAM, and 251 GiB of RAM.

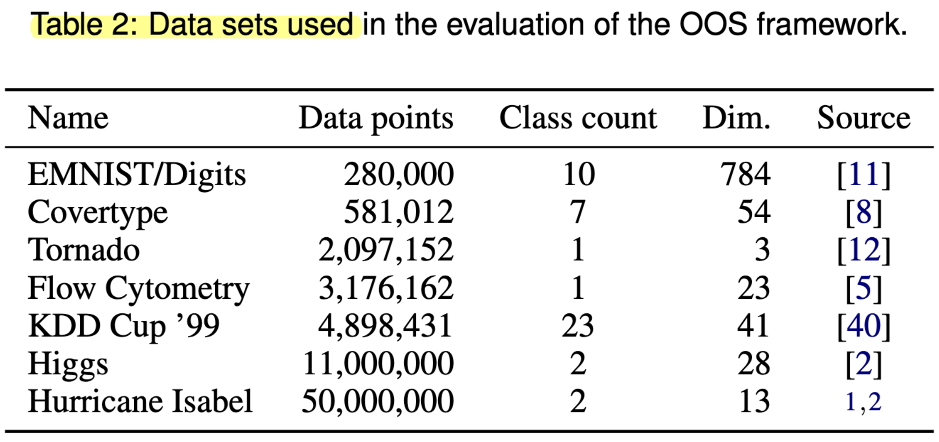


Table 2: Data sets used in the evaluation of the OOS framework.

Name	Data points	Class count	Dim.	Source
EMNIST/Digits	280,000	10	784	[11]
Covertype	581,012	7	54	[8]
Tornado	2,097,152	1	3	[12]
Flow Cytometry	3,176,162	1	23	[5]
KDD Cup ’99	4,898,431	23	41	[40]
Higgs	11,000,000	2	28	[2]
Hurricane Isabel	50,000,000	2	13	1,2

Figure 8: Datasets used in the evaluation of the OOS framework proposed in [18].

- **Metrics** used:

  - **Global**: *stress* (calculation in batches in the GPU), *Pearson correlation coefficient* between the flattened low- and high-dimensional distance vectors.
  - **Local**: *KNN precision*, *trustworthiness* (calculation in batches in the GPU).

- **Computational complexities**:
  - Runtime (RT) complexity: $\mathcal{O}\Big(\Phi^{\mathrm{RT}}\big(n_{\mathrm{ref}}\big)\Big) + \mathcal{O}\Big(\Phi_{\beta}^{\mathrm{RT}}\big(n_{\mathrm{batch}}\big) \cdot \mathrm{batch\_count}\Big)$
  - Memory (M) complexity of a batch projection: $\mathcal{O}\Big(\Phi^{\mathrm{M}}\big(n_{\mathrm{ref}}\big)\Big) + \mathcal{O}\Big(\Phi_{\beta}^{\mathrm{M}}\big(n_{\mathrm{batch}}\big)\Big)$
  - Metric MDS: $\Phi\big(n_{\mathrm{ref}}\big)$ is $\mathcal{O}(n_{\mathrm{ref}}^2)$ in RT and M. $\Phi_\beta\big(n_{\mathrm{batch}}\big)$ has $\mathcal{O}(n_{\mathrm{ref}} \cdot n_{\mathrm{batch}})$ time complexity (and I'd argue it is $\mathcal{O}(n_{\mathrm{batch}}^2)$ in memory).
  - t-SNE: same as metric MDS, except for M of Φ_β.
  - UMAP: $\Phi\big(n_{\mathrm{ref}}\big)$ is $\mathcal{O}(n_{\mathrm{ref}}^{1.14})$ in T and $\Phi_\beta\big(n_{\mathrm{batch}}\big)$ is like in MDS.
  - Autoencoders: Φ has linear TR and its M depends on the architecture. Φ_β is approximately linear.
  - PCA: Φ is linear with respect to n_{ref} and dimensionality. Φ_β is approximately linear.

- **Implementation**: in Python with `Numba`(see section 10) compiled functions for performance-critical sections. They used the UMAP implementation of the umap-learn library, the openTSNE library for t-SNE, scikit-learn for PCA, and keras for the autoencoder. For MDS, they provided their own implementation (see `code/ReichmanHagele2024_MDS/mds_demo.py`).

- **Experiments results**:
  - **Quality**: PCA and MDS are consistent very early on (starting with a ref. set size of 128). t-SNE and UMAP overfit and generate clusters on small ref. sets, although they perform well on local metrics. Autoencoders is the most inconsistent method, but in general when the ref. set size increases, global metrics worsen and local ones improve.
  - **Runtime**: PCA and MDS are consistent with theory. t-SNE shows a below quadratic time complexity of Φ, why? Moreover, time complexity of Φ_β seems constant with respect to n_{ref}, why? UMAP is significantly slower in high-dimensional datasets. The dependance on batch size is inconsistent in most methods, even though the general trend is to be sub-linear.

- **Comparison to large-scale DR methods**: when comparing UMAP with the proposed OOS framework against classical UMAP, PaCMAP and TriMap, PaCMAP is consistently better in terms of quality metrics, although the OOS framework is capable of reducing runtime significantly at the expense of quality. In previous work, UMAP failed to transform datasets of 1 million rows; how did the authors manage to run it on datasets of 2, 5 and 11 million data points (Tornado, KDD Cup '99 and Higgs, respectively)?

- They finally show a use case with 1 billion streamlines generated on the *fluid simulation ensemble for machine learning*.

- **Final discussion**:
  - Even though results depend on the DR technique, runtime can always be substantially reduced by using smaller reference set sizes.
  - The authors recommend testing the reference projection before applying the OOS projection to the whole data set.
  - Tuning the hyperparameters of the DR methods should improve results.
  - An informed selection of the reference set instead of a random one could improve results as well.
  - Another limitation of the proposed framework is that relationships between OOS points are ignored and that Φ_β is not updated with OOS points.

7 Local Multidimensional Scaling for Nonlinear Dimension Reduction, Graph Drawing, and Proximity Analysis [2]

7.1 Abstract

Original: In the past decade there has been a resurgence of interest in nonlinear dimension reduction. Among new proposals are "Local Linear Embedding", "Isomap", and Kernel Principal Components Analysis which all construct global low-dimensional embeddings from local affine or metric information. We introduce a competing method called "Local Multidimensional Scaling" (LMDS). Like LLE, Isomap, and KPCA, LMDS constructs its global embedding from local information, but it uses instead a combination of MDS and "force-directed" graph drawing. We apply the force paradigm to create localized versions of MDS stress functions with a tuning parameter to adjust the strength of nonlocal repulsive forces. We solve the problem of tuning parameter selection with a meta-criterion that measures how well the sets of K-nearest neighbors agree between the data and the embedding. Tuned LMDS seems to be able to outperform MDS, PCA, LLE, Isomap, and KPCA, as illustrated with two well-known image datasets. The meta-criterion can also be used in a pointwise version as a diagnostic tool for measuring the local adequacy of embeddings and thereby detect local problems in dimension reductions.

Apple Intelligence summary: Tuned LMDS outperforms other dimension reduction methods on image datasets. LC meta-criteria measure agreement between data and embedding, aiding tuning and detecting local problems.

7.2 Key Points

- The stability of optimal MDS configurations stems from the large dissimilarities, and localized MDS did not appear to be a viable approach.
- In view of MDS' reliance on large distances, Isomap may be driven mostly by the large but noisy shortest-path imputations, whereas the local distances play only a minor role.
- Isomap suffers from more variance, LLE and LMDS from more bias.
- LC meta-criteria are proposed and used to tune parameters and to produce diagnostic plots. However, they are not smooth and statistically unstable.
- **LMDS**: given a distance matrix $D_{i,j}$, let $\mathcal{N}$ be a symmetrized k-NN graph: $(i,j) \in \mathcal{N}$ if j is among the K nearest neighbors of i or viceversa. LMDS looks for a low-dimensional configuration $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^d$ that minimizes

  $$\text{LMDS}^D_{\mathcal{N}}(\mathbf{x}_1, \dots, \mathbf{x}_N) = \sum_{(i,j)\in N} \left(D_{i,j} - \|\mathbf{x}_i - \mathbf{x}_j\|\right)^2 - t \sum_{(i,j)\notin N} \|\mathbf{x}_i - \mathbf{x}_j\|,$$

  with $t = \frac{|\mathcal{N}|}{|\mathcal{N}^C|} \cdot \text{median}_{\mathcal{N}}(D_{i,j}) \cdot \tau$.

  The first term is the "local stress" and the segond one is the "repulsion".

A good strategy for optimization is to start with a large value such as $\tau = 1$ to obtain a stable configuration, and lower τ successively as low as 0.01, using previous configurations as initializations for smaller values of τ.

- **LCMC (Local Continuity Meta-Criteria)**: For case i we form the index set $\mathcal{N}^D_{K'}(i) = \{j_1, \ldots, j_{K'}\}$ of K'-NNs with regard to $D_{i,j}$, and $\mathcal{N}^X_{K'}(i) = \{k_1, \ldots, k_{K'}\}$ of $K'-$ NNs with regard to $\|\mathbf{x}_i - \mathbf{x}_k\|$ (excluding i). The overlap is measured pointwise and globally by

$$N_{K'}(i) = \left|\mathcal{N}^D_{K'}(i) \cap \mathcal{N}^X_{K'}(i)\right|, \quad N_{K'} = \frac{1}{N}\sum_{i=1}^{N} N_{K'}(i).$$

  Maximum faithfulness is achieved when $N_{K'} = K'$, so we normalize:

$$M_{K'} = \frac{N_{K'}}{K'}.$$

  If there is complete absence of association between the data and the configuration, the local overlap $N_{K'}(i)$ is random and can be modeled by a hypergeometric with $\mathbb{E}[N_{K'}] = \frac{K'^2}{N-1}$. Hence, we define the *adjusted LC meta-criteria* as:

$$M^{\text{adj}}_{K'} = M_{K'} - \frac{K'}{N-1}.$$

- Selection of tuning parameters:
  - τ: Given K for $\text{LMDS}_{K,\tau}$ and K' for $M_{K'}$, we can optimize the repulsion weight τ with a grid search. That is, for a set of τ values, we perform $\text{LMDS}_{K,\tau}$ and compute $M_{K'}$ on the obtained embeddings. Then, we choose the value for τ that maximizes $M_{K'}$.
  - K: there are two strategies:
    - $K' = K$: For fixed K minimize $\text{LMDS}_{K,\tau}$ for various values of τ; pick the τ whose configuration maximizes $M^{\text{adj}}_{K'}$; repeat for various values of $K = K'$ and pick that $K = K'$ which maximizes the trace.
    - K', K decoupled: It is desirable that K is not just $M^{\text{adj}}_{K'}$ -optimal for $K' = K$, but for a range of values K'. To find out, one may plot traces $K' \mapsto M^{\text{adj}}_{K'}$ one for each value of K.

- The LC meta-criteria are doubly "non-metric" in the sense that they only use rank information of both $\{D_{i,j}\}$ and $\{\|x_i - x_j\|\}$.

- The Achilles heel of methods considered here is the complete reliance on distance data or dissimilarities, which holds both for the LMDS fitting criterion and the LC meta-criteria. This means that users might have to choose specific distances/dissimilarities for their problems.

8 Classical MDS benchmarked in R

In this section we aim to test the limits of `Rdimtools::do.mds`. Specifically, we will run Rdimtools's implementation of Classical MDS on datasets of increasing number of observations and dimensionality in order to test its limitations as well as its complexity. We will also compare it with `dimRed::embed(.method="MDS", ...)`.

Code, data and figures of the experiment can be found in `code/MDS_benchmark_R_lib`. As for the system used, tests were performed on a Macbook Pro (14-inc, Nov 2023) with 16 GB of RAM and the Apple M3 chip.

In our first experiment (see figure 9), we run `Rdimtools::do.mds` on datasets consisiting of four isotropic gaussian blobs with random covariance matrices randomly centered in a d-dimensional space with sizes ranging from 100 to 100000. We let d be 8, 32 and 64. However, for all dimensionalities tested, R crashed when the dataset had at least 56234 rows. This result shows the memory limitation of classical MDS, which is reached in datasets with sizes of the order of 10^5 for modern consumer-grade systems.

Regarding time complexity, classical MDS shows quadratic growth with respect to number of individuals and linear growth in dimensionality, although we cannot be absolutely sure of the latter because only three dimensionalities were tested.

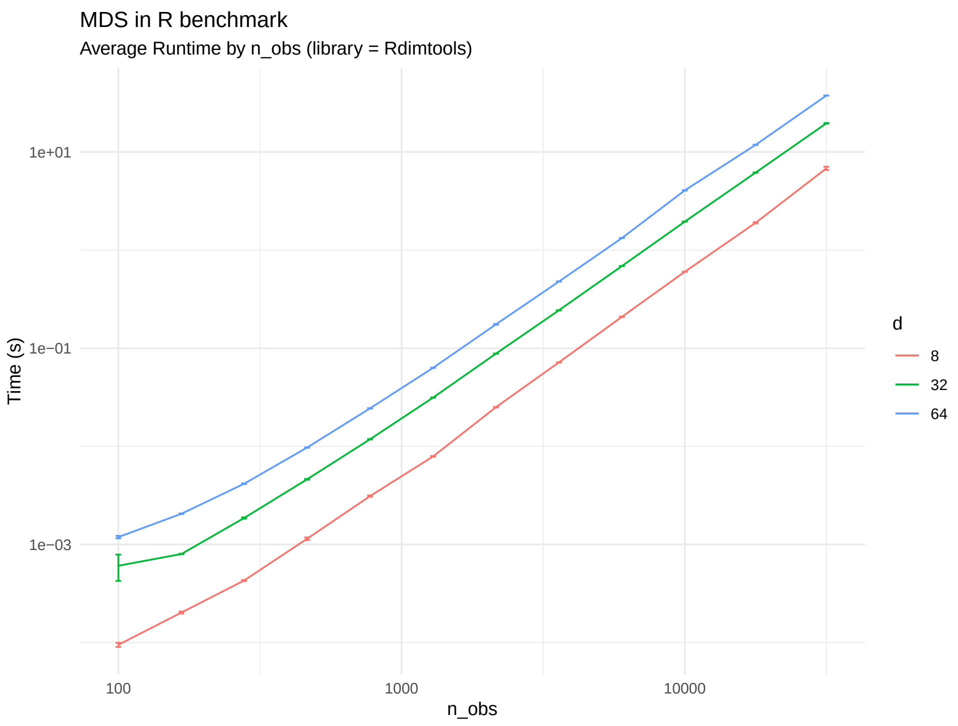


Figure 9: Runtime (s) of `Rdimtools::do.mds` averaged over 20 experiments. The tests were performed on datasets consisiting of four isotropic gaussian blobs with random covariance matrix and randomly centered in a d-dimensional space with sizes ranging from 100 to 31623. We let d be 8, 32 and 64.

Next, we compared the implementations of classical MDS in libraries `Rdimtools` and `dimRed`, since `Rdimtools` is said to be very efficient while `dimRed` just wraps the `cmdscale` method [6]. As can be seen in figure 10, our tests corroborate the results shown in [25], since `Rdimtools::do.mds` has a lower time complexity than `dimRed::embed(.method="MDS", ...)`, resulting in runtime improvements of 10x for datasets with 100 rows and up to 1000x for samples with 10000 observations. Indeed, we did not test `dimRed::embed(.method="MDS", ...)` on larger datasets because, when size equaled 1000 rows, each experiment took around 15 minutes to run on our system.

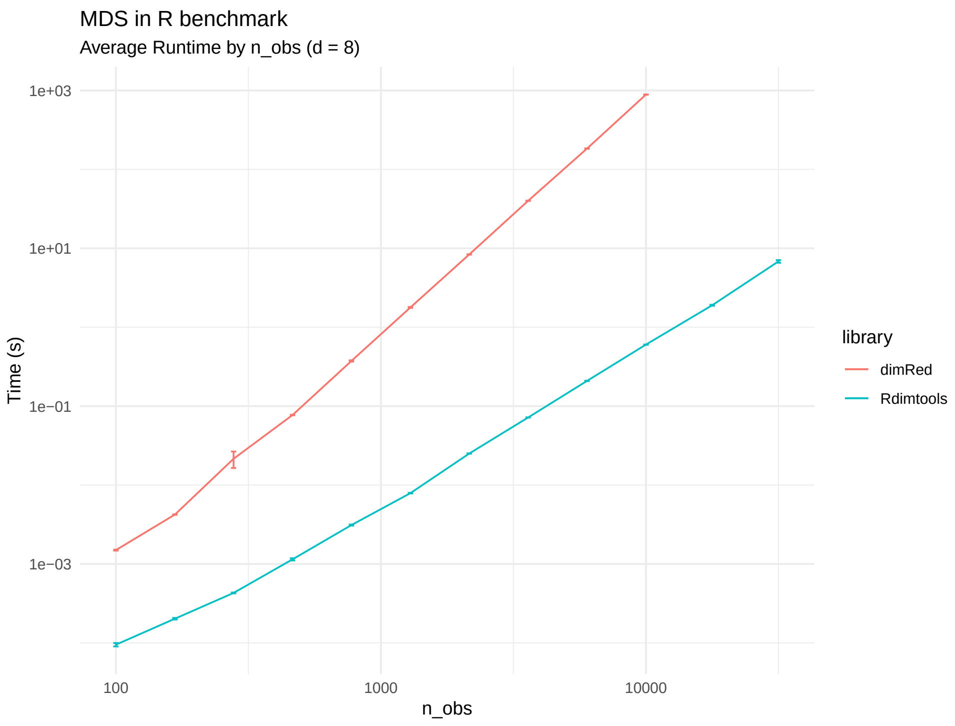


Figure 10: Runtime (s) of `Rdimtools::do.mds` and `dimRed::embed(.method="MDS", ...)` averaged over 20 experiments. Both methods were tested on the same datasets, consisiting of four isotropic gaussian blobs with random covariance matrix and randomly centered in an 8-dimensional space with sizes ranging from 100 to 10000 for `dimRed::embed(.method="MDS", ...)` and from 100 to 31623 for `Rdimtools::do.mds`.

9 DR Python packages

9.1 Key Points

- To our knowledge, there is no comprehensive Python package that implements many DR techniques akin the R library `Rdimtools` (see section 1).
- To show the consequences of this in an example, classical MDS is only implemented in `pyseer` (through `pyseer.cmdscale`), a package for microbial pangenome-wide association studies [8].
- The package that contains the most DR methods is `scikit-learn`, specifically in its `manifold` module [15]. It implements: Isomap, LLE, Laplacian Eigenmaps, t-SNE and non-classical MDS (both `sklearn.manifold.MDS` and `sklearn.manifold.smacof` can perform metric and non-metric MDS). Moreover, `sklearn.manifold.trustworthiness` measures to what extent the local structure is retained when dimensionality is reduced. Hence, the total amount of methods present in `sklearn.manifold` cannot be compared to the 143 of `Rdimtools`.
- Another remarkable, although differently oriented, library is `direpack`, which implements a set of modern statistical dimension reduction techniques including projection pursuit, sufficient dimension reduction, and robust M estimators. It also includes regularized regression estimators, pre-processing utilities, plotting functionality, and cross-validation utilities, all consistent with the scikit-learn API [13]. Nonetheless, these methods are outside the scope of our thesis.
- Other, more specific, libraries that implement DR methods in Python are:
  - `umap-learn`: the library that introduced UMAP [11].
  - `MulticoreTSNE`: a multicore modification of Barnes-Hut t-SNE with Python CFFI-based wrappers [23]. Barnes-Hut is a tree-based algorithm that can be used to accelerate t-SNE up to $\mathcal{O}(N \log N)$ [10].
  - `fitsne`: Fast Fourier Transform-accelerated Interpolation-based t-SNE [9].
  - `OpenTSNE`: incorporates the latest improvements to the t-SNE algorithm, including the ability to add new data points to existing embeddings, massive speed improvements, enabling t-SNE to scale to millions of data points and various tricks to improve global alignment of the resulting visualizations [16]. The team behind `OpenTSNE` benchmarked the here described t-SNE Python implementations and showed that the fastest one in general is `fitsne`, although `OpenTSNE` is equally efficient on multicore systems (see figure 11) [16].
- Moreover, other libraries focused on certain fields like genomics (`phate`) or NLP (`gensim`) implement DR methods for specific tasks and scenarios in their topics [14, 17].

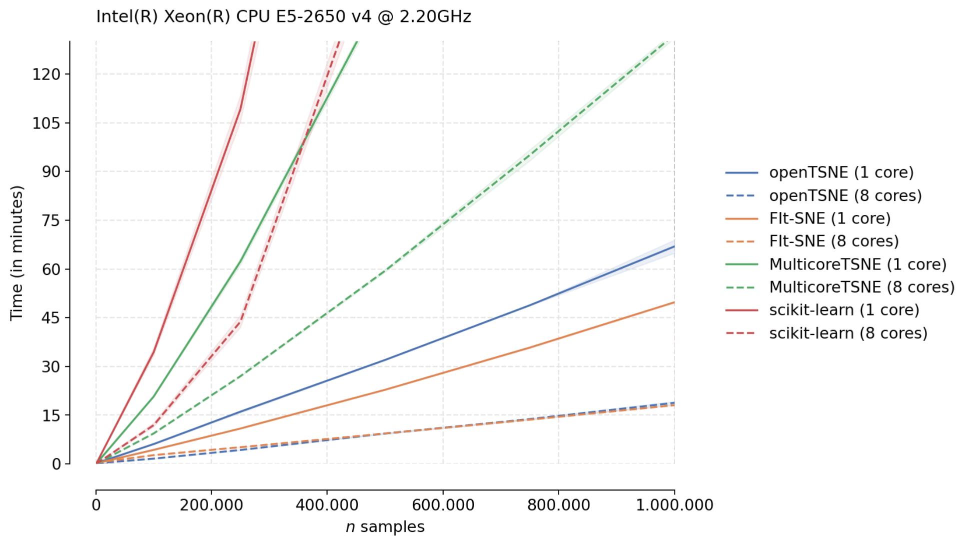


Figure 11: Benchmark of t-SNE Python implementations [16].

10 The `Numba` Python package

`Numba` [7] is an open-source just-in-time (JIT) compiler that translates a subset of Python and `Numpy` code into optimized machine code using LLVM.

It accelerates numerical computations on CPUs and GPUs by enabling parallelization and vectorization with minimal code modifications. To gain the most out of it, computations should use loops, `Numpy` functions and `Numpy` broadcasting.

10.1 How does `Numba` work?

As described in its documentation [7], `Numba` reads the Python bytecode for a decorated function and combines this with information about the types of the input arguments to the function. It analyzes and optimizes your code, and finally uses the LLVM compiler library to generate a machine code version of your function, tailored to your CPU capabilities. This compiled version is then used every time your function is called.

11 Isomap in Python

In this section we will explore the current Isomap Python implementations to use them later in our divide-and-conquer and recursive big data algorithms.

11.1 Package implementations

- `sklearn.manifold.Isomap` [15]: as shown in `code/ReichmanHagele2024_MDS/mds_demo.py`, the constructor of this class accepts the dimensionality of the projection. Then, the method `fit_transform` applies Isomap to the dataset passed in the arguments.
- `pysomap.isodata` [22]: this package requires Python 2.4.3, `numpy` 1.0.1 and `SWIG` 1.3.29. After some attempts to install the package on our macOS system, we have not been succesful, so, given its age, we will reject it.
- Distributed implementation with Apache Spark: [20].
- `megaman` [12]: this package is said to implement manifold learning algorithms for big data, but it is not very trustworthy. Indeed, it has not been published in any magazine nor conference and our attemps to install it have failed. We tried to use the conda installation, but it was not active anymore. Moreover, we sourced its GitHub repository and failed again because it depends on the deprecated module `numpy.distutils.msvccompiler`, which seems to be very complicated to run on macOS.
- Let us test the partitions' projections:
  - With $n = 3000, l = 1000$, projections concentrate in filaments, but are uniform along both axes with $n_neighbors \leq 6$.

12 Code notes and changelog

12.1 March 6th

- Made more clear the calls to `main_divide_conquer_isomap`.
- Replaced `np.array_split` by `np.split` when obtaining the indexes of middle subsets in `get_partitions_for_divide_conquer`. `np.array_split` works like `np.split`, except for when the length of the array (l) is not a multiple of the number of subarrays (n). Then, instead of raising an error, it creates $n\%l$ subarrays of length $n\backslash\backslash l+1$ and $n-n\%l$ subarrays of length $n\backslash\backslash l$. Hence, `np.split` is more appropiate for our code.
- Replaced $p = \lceil 1 + \frac{n-l}{l-c} \rceil$ by $p = \lceil \frac{n-l}{l-c} \rceil$ and adapted the rest of `get_partitions_for_divide_conquer` to the change. In other words, now p is the number of subsets with less than l data points.
- If $l - c \mid n - l$, then

  `last_partition_sample_size = n - (l + (p - 1) * (l - c_points))`

  makes `last_partition_sample_size` negative!! Moreover, I do not understand why

  ```
  if last_partition_sample_size < min_sample_size:
      p = p - 1
      last_partition_sample_size = n - (l + (p - 1) * (l - c_points))
  ```

  ensures that `last_partition_size` $\geq$ `min_sample_size`. Therefore, I propose to get the indexes of all paritItons except for the first one with `np.array_split`. Moreover, `min_sample_size` is controlled by the initial `ValueError` checks, so by changing the third one to `l-c_points` $<$ `r+2` we do not have to worry about `min_sample_size` anymore.
- Improved visualization of whole and partitions projections. Now it is easier to evaluate them.
- Tested Isomap D&C with new visualizations and stored results. Problems seem to come from partitions being too small, since applying Isomap to the Swiss Roll with the same amount of data points as in a partition in the previous experiment provides the same results as the partitions. In particular, this means that Isomap D&C works better with less `n_neighbors` than Isomap.
- Written `d_and_c.py`, a generic function to apply D&C to DR methods.
- Translated `stops`' implementation of Local MDS to Python.

12.2 March 13th

- `d_and_c.divide_conquer` now uses enums to specify the DR method.
- Improved code structure of `d_and_c.py`: separated methods in proper modules; specified private methods; created `__init__.py`.

- Reviewed methods descriptions and added type specifications.
- Written `examples.example_3D_to_2D`, a testing function for 2D emebeddings of 3D datasets. With respect to previous tests, it:
  - Stores relevant results in a text file.
  - Stores figures and other results in subfolders instead of using very long filenames.
  - Adds titles to plots.
  - Allows any picture format for plots.
- Tried to apply `Numba`'s `@jit` decorator to some functions in order to accelerate them by parallelizing numeric computations on the CPU. Performance was tested both on a MacBook Pro with the M3 ARM chip and on a Windows system with an NVIDIA GPU, but it decreased in all cases...:
  - On `d_and_c.divide_conquer`: tried to optimized the final centering and alignment with principal components:

```python
@njit(parallel=True)
def center_and_rotate(combined_projection: np.ndarray) -> np.ndarray:
    """
    Center a matrix and rotate it to align with its principal components.

    Parameters:
        combined_projection (np.ndarray): A 2D array where each row is a
        data point and each column is a feature.

    Returns:
        transformed_matrix (np.ndarray): The transformed data matrix after
        centering and rotation.
    """
    n_samples, n_features = combined_projection.shape

    # Compute column means in parallel.
    mean = np.empty(n_features)
    for j in prange(n_features):
        s = 0.0
        for i in prange(n_samples):
            s += combined_projection[i, j]
        mean[j] = s / n_samples

    # Subtract mean from each row in parallel.
    centered = np.empty_like(combined_projection)
    for i in prange(n_samples):
        for j in prange(n_features):
            centered[i, j] = combined_projection[i, j] - mean[j]

    # Compute covariance matrix in parallel over rows.
    cov = np.empty((n_features, n_features))
    for i in prange(n_features):
        for j in prange(n_features):
            s = 0.0
            for k in prange(n_samples):
                s += centered[k, i] * centered[k, j]
            cov[i, j] = s / (n_samples - 1)

    # Eigen-decomposition (cannot be parallelized by Numba).
    eigvals, eigvecs = np.linalg.eigh(cov)

    # Sort eigenvalues in descending order and reorder eigenvectors.
    idx = np.argsort(eigvals)[::-1]
    sorted_eigvecs = eigvecs[:, idx]

    # Project the centered data onto the eigenvectors.
    result = np.empty_like(centered)
    for i in prange(n_samples):
        for j in prange(n_features):
            s = 0.0
            for k in prange(n_features):
                s += centered[i, k] * sorted_eigvecs[k, j]
            result[i, j] = s

    return result
```

  – `private_d_and_c._get_procrustes_parameters`
  – `private_d_and_c.perform_procrustes`
  – `methods.isomap`

- Simplified `stops`'s Local MDS implementation. Why is $cc = \frac{2|\mathcal{N}|-N}{N^2}$ instead of $cc = \frac{|\mathcal{N}|}{\frac{N(N-1)}{2}-|\mathcal{N}|}$? There are other steps I do not comprehend well. Moreover, `stops` calls `smacofx:lmds` and adds diagnostic metrics. `smacofx` also uses its own optimization framework, which is not straightforward.
- Therefore, I am trying to write my own Local MDS implementation on Python with `cvxpy`. Nonetheless, at the moment it does not work. I am figuring out how to solve an optimization problem with so many variables as stress minimization.

12.3 March 20th

- `Rdimtools` nor `dimRed` have an implementation of Local MDS. Therefore, we keep using the one in `smacofx`.
- Local MDS has been tested on the examples seen in the AMA course. Results in Pyhton and R coincide graphically, although axes' limits may vary slightly. See `code/d_and_c/LMDS/LMDS_examples.ipynb`.
- `t-SNE` has been added as a `DRMethod` through the openTSNE implementation. The Jupyter notebook `code/d_and_c/t-SNE/tSNE_examples.ipynb` includes the following tests:

  - two examples from [24]: two 2D clusters with equal number of points; random Gaussian walk.
  - four 3D clusters with equal number of points centered in the vertices of a tetrahedron. The goal of this experiment was to check if opposite vertices change with different parameters, by chance or modifying the noise in the clusters. We have found out that variations in the noise of each vertex of the tetrahedron may embed a pair of vertices in the high-dimensional space into opposite or adjacent vertices of a square (or even not form a square).
  - the MNIST train dataset with numeric digits. We have splitted it into two partitions and then applied t-SNE on each. Results are inconsistent, since shapes are different and some regions are swapped; specifically, those corresponding to digits 2, 5 and 6.

  From these tests we can conclude that t-SNE probably will not be a suitable DR method for D&C. Indeed, Procrustes transformation (consisting of a rotation and maybe a translation) cannot rectify the observed incoherences in the embeddings of different subsets of MNIST or the tetrahedron.

- Benchmarked D&C Isomap on the swiss roll datasets for different parameter combinations, number of individuals and parallel vs serial. Scripts, notebooks and results can be found in `code/d_and_c/benchmark`. Embedding's plots have been stored, as well as a benchmark log that includes warnings regarding disconnected kNN graphs. Finally, `code/d_and_c/benchmark/plot_benchmark.ipynb` contains a plot summarizing the results obtained.

12.4 March 27th

- `openTSNE` parallelization according to Claude 3.7 Sonnet Thinking: it is applied to computationally intensive operations:
  - Affinity Calculation: When computing nearest neighbors in the `MultiscaleMixture` constructor:

```
affinities = MultiscaleMixture(X, ..., n_jobs=self.n_jobs)
```

  - Gradient Computation: Used in both positive and negative gradient calculations:

```
# In Barnes-Hut approximation
sum_Q = _tsne.estimate_negative_gradient_bh(..., num_threads=n_jobs)

# In positive gradient calculation
sum_P, kl_divergence_ = _tsne.estimate_positive_gradient_nn(...,
    num_threads=n_jobs)
```

  The actual parallel execution happens in the C++ code (`_tsne` module):

  - The Python parameter `n_jobs` is passed as `num_threads` to C++ functions
  - These C++ functions use native threading libraries (likely `OpenMP`) to distribute computation

  - Parallelization focuses on data-parallel operations like:
    * Distance calculations
    * Nearest neighbor searches
    * Gradient computations
    * Tree building (for Barnes-Hut)
    * FFT calculations (for interpolation method)

    Concluding, the key to make parallelization available in Apple Silicon chips seems to be C++ code, which at this moment seems to be out of scope for our project.

- Added a parameter to all DR methods to apply Principal Components to the obtained embedding.

- Tested Principal Components on Local MDS with the swiss roll dataset. Embedding improves, but it is still bad in the swiss roll dataset.

- Tested t-SNE on a solid tetrahedron. To generate data uniformly on a regular tetrahedron, the `runif_in_tetrahedron` function from the `uniformly` R library has been translated to Python. Moreover, this Python implentation has been optimized with `Numba`. As a consequence, generating 1000 points takes about 4 miliseconds. Meanwhile, the same function compiled without the `@numba.njit` decorator takes about 0.2 seconds to complete the same task. That being said, when generating 1000000 points, the speed up gained with `Numba` is only of 10x.

  Actually, we have tested an alternative method, which generates data in the 3-simplex through the flat Dirichlet distribution and then affinely transforms it into points from the specific tetrahedron. This method is more elegant, understandable and equally fast than the `Numba` optimized implementation.

  Let us see why the flat Dirichlet distribution is uniform in the 3-simplex. The density function of Dir($\boldsymbol{\alpha}$) is given by

$$f\left(x_1, \ldots, x_K ; \alpha_1, \ldots, \alpha_K\right)=\frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{i=1}^K x_i^{\alpha_i-1},$$

  where $\{x_k\}_{k=1}^{k=K}$ belong to the standard $(K-1)$-simplex, or in other words:

$$\sum_{i=1}^K x_i=1 \text{ and } x_i \in[0,1] \text{ for all } i \in\{1, \ldots, K\}$$

  The normalizing constant is the multivariate beta function, which can be expressed in terms of the gamma function:

$$\mathrm{B}(\boldsymbol{\alpha})=\frac{\prod_{i=1}^K \Gamma\left(\alpha_i\right)}{\Gamma\left(\sum_{i=1}^K \alpha_i\right)}, \quad \boldsymbol{\alpha}=\left(\alpha_1, \ldots, \alpha_K\right)$$

  Now, the flat Dirichlet distribution is obtained when $\alpha_i=1\,\forall i$ and, in our case, $K=4$. Therefore, when $x_i \neq 0\,\forall i$ (the volume of this set's complement is null):

$$f_{flat}\left(x_1, \ldots, x_4\right)=\frac{1}{\mathrm{B}(1,1,1,1)}=\frac{\Gamma(4)}{\Gamma(1)^4}=\frac{3!}{1}=6$$

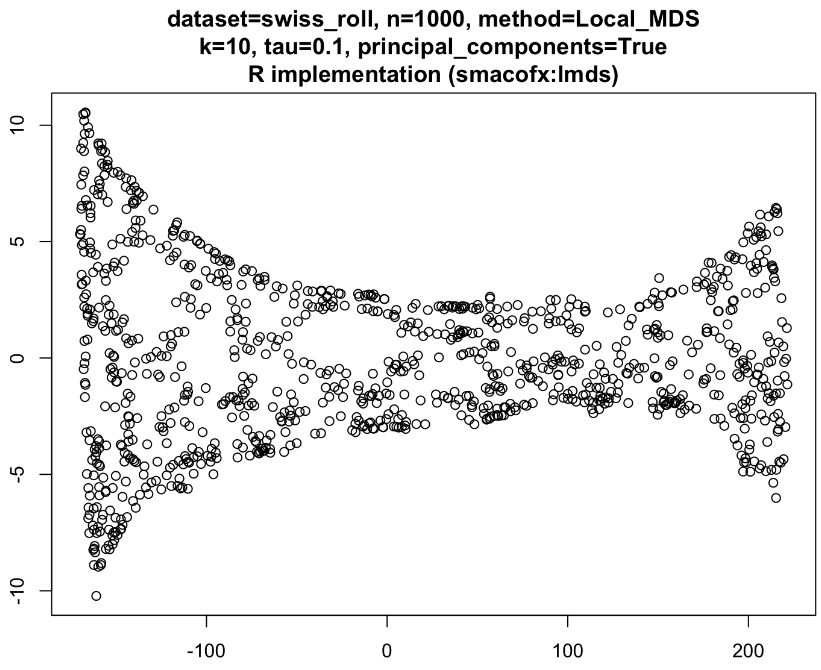
In other words, the flat Dirichlet distribution of order K is uniform over the $(K-1)$-simplex.

12.5 April 3rd

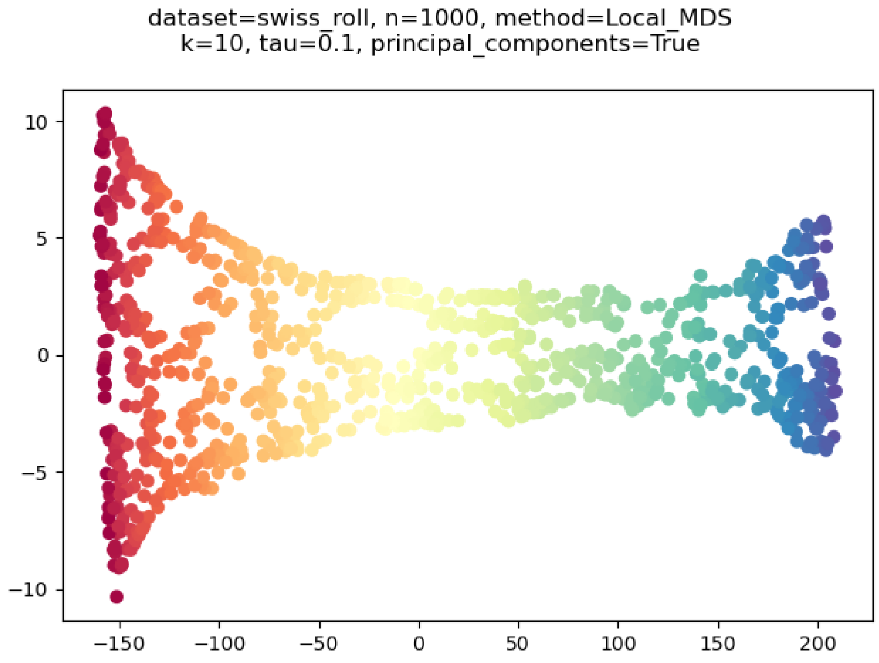
- **Applied t-SNE to the Swiss Roll dataset**: We have fine-tuned the parameters of t-SNE to obtain a resonably good 2D-embedding of the Swiss Roll dataset with 1000 rows. The obtained parameters have been $\{perplexity : 30, n\_iter : 250\}$. Results with $n\_iter = 500$ are similar, but the algorithm is significantly slower. Plots of each embedding can be found at `d_and_c.LMDS.plots`.

- **Measured time complexity of D&C t-SNE**: with the fine-tuned parameters previously obtained, we have performed a series of experiments on the Swiss Roll dataset. Results can be seen in `d_and_c.benchmark.plot_benchmark_ipynb`.

- **Repeated last week's experiments on Local MDS with Principal Components on R**: to ensure once more that the Python translation of `smacofx:lmds` is sound. As seen in `d_and_c.LMDS.run_LMDS.Rmd`, it is:



(a) R implementation



(b) Python implementation

Figure 12: Comparison of Local MDS implementations on the Swiss Roll dataset (n=1000, k=10, τ=0.1) with principal components applied.

- **Optimized Local MDS**: we have preallocated matrices in the main optimization loop and calculated the gradient and stress with Numba compiled independent functions. See `d_and_c.private_lmds.py`.

- **Tested Isomap on a tetrahedron**: both on four clusters centered at the vertices and on a solid tetrahedron. The goal is to observe whether the insconsistences present when using t-SNE also appear with this DR method.

  Results have turned out similar than in t-SNE for both datasets. However, note that in the ”vertices” dataset the kNN graph had to be modified by adding the shortest edges between pairs of clusters to connect the graph. This is a necessary operation to compute geodesics on it. For more details, see `d_and_c.Isomap.Isomap_examples.ipynb`.

13 Title TEXstring

13.1 Abstract

Original:

Apple Intelligence summary:

13.2 Key Points

-

References

[1] Yoshua Bengio et al. “Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering”. In: *Advances in Neural Information Processing Systems.* Ed. by S. Thrun, L. Saul, and B. Schölkopf. Vol. 16. MIT Press, 2003. URL: `https://proceedings.neurips.cc/paper_files/paper/2003/file/cf05968255451bdefe3c5bc64d550517-Paper.pdf`.

[2] Lisha Chen and Andreas Buja. “Local Multidimensional Scaling for Nonlinear Dimension Reduction, Graph Drawing, and Proximity Analysis”. In: *Journal of the American Statistical Association* 104.485 (2009), pp. 209–219. DOI: `10.1198/jasa.2009.0111`. eprint: `https://doi.org/10.1198/jasa.2009.0111`. URL: `https://doi.org/10.1198/jasa.2009.0111`.

[3] Vin De Silva and Joshua B Tenenbaum. *Sparse multidimensional scaling using landmark points.* Tech. rep. Technical Report, Stanford University, 2004.

[4] Andrej Gisbrecht et al. “Out-of-sample kernel extensions for nonparametric dimensionality reduction”. In: *The European Symposium on Artificial Neural Networks.* 2012. URL: `https://api.semanticscholar.org/CorpusID:15613611`.

[5] Andreas Hinterreiter et al. “ParaDime: A Framework for Parametric Dimensionality Reduction”. In: *Computer Graphics Forum* 42.3 (2023), pp. 337–348. DOI: `https://doi.org/10.1111/cgf.14834`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14834`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14834`.

[6] Guido Kraemer, Markus Reichstein, and Miguel D. Mahecha. “dimRed and coRanking - Unifying Dimensionality Reduction in R”. In: *R J.* 10 (2018), p. 342. URL: `https://api.semanticscholar.org/CorpusID:62831555`.

[7] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC.* 2015, pp. 1–6.

[8] John A Lees et al. “pyseer: a comprehensive tool for microbial pangenome-wide association studies”. In: *Bioinformatics* 34.24 (July 2018), pp. 4310–4312. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/bty539`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/34/24/4310/48919461/bioinformatics\_34\_24\_4310.pdf`. URL: `https://doi.org/10.1093/bioinformatics/bty539`.

[9] George C. Linderman et al. “Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data”. In: *Nature Methods* 16.3 (Mar. 1, 2019), pp. 243–245. DOI: `10.1038/s41592-018-0308-4`. URL: `https://doi.org/10.1038/s41592-018-0308-4`.

[10] Laurens van der Maaten. “Accelerating t-SNE using Tree-Based Algorithms”. In: *Journal of Machine Learning Research* 15.93 (2014), pp. 3221–3245. URL: `http://jmlr.org/papers/v15/vandermaaten14a.html`.

[11] L. McInnes, J. Healy, and J. Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *ArXiv e-prints* (Feb. 2018). arXiv: `1802.03426 [stat.ML]`.

[12] James McQueen et al. *megaman: Manifold Learning with Millions of points.* 2016. arXiv: `1603.02763 [cs.LG]`. URL: `https://arxiv.org/abs/1603.02763`.

[13] Emmanuel Jordy Menvouta, Sven Serneels, and Tim Verdonck. "direpack: A Python 3 package for state-of-the-art statistical dimensionality reduction methods". In: *SoftwareX* 21 (2023), p. 101282. ISSN: 2352-7110. DOI: `https://doi.org/10.1016/j.softx.2022.101282`. URL: `https://www.sciencedirect.com/science/article/pii/S235271102200200X`.

[14] Kevin R. Moon et al. "Visualizing structure and transitions in high-dimensional biological data". In: *Nature Biotechnology* 37.12 (Dec. 1, 2019), pp. 1482–1492. DOI: `10.1038/s41587-019-0336-3`. URL: `https://doi.org/10.1038/s41587-019-0336-3`.

[15] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[16] Pavlin G. Poličar, Martin Stražar, and Blaž Zupan. "openTSNE: A Modular Python Library for t-SNE Dimensionality Reduction and Embedding". In: *Journal of Statistical Software* 109.3 (2024), pp. 1–30. DOI: `10.18637/jss.v109.i03`. URL: `https://www.jstatsoft.org/index.php/jss/article/view/v109i03`.

[17] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[18] Luca Reichmann, David Hägele, and Daniel Weiskopf. *Out-of-Core Dimensionality Reduction for Large Data via Out-of-Sample Extensions*. 2024. arXiv: `2408.04129 [cs.LG]`. URL: `https://arxiv.org/abs/2408.04129`.

[19] Henouda Salah Eddine et al. "Comparative study for dimensionality reduction techniques for big data". In: Nov. 2020.

[20] Frank Schoeneman and Jaroslaw Zola. " Scalable Manifold Learning for Big Data with Apache Spark ". In: *2018 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2018, pp. 272–281. DOI: `10.1109/BigData.2018.8622617`. URL: `https://doi.ieeecomputersociety.org/10.1109/BigData.2018.8622617`.

[21] Vin Silva and Joshua Tenenbaum. "Global Versus Local Methods in Nonlinear Dimensionality Reduction". In: *Advances in Neural Information Processing Systems*. Ed. by S. Becker, S. Thrun, and K. Obermayer. Vol. 15. MIT Press, 2002. URL: `https://proceedings.neurips.cc/paper_files/paper/2002/file/5d6646aad9bcc0be55b2c82f69750387-Paper.pdf`.

[22] spiwokv. *Pysomap: Python Library for Isometric Feature Mapping (Isomap)*. `https://web.vscht.cz/~spiwokv/`. Accessed: 26 February 2025. 2007.

[23] Dmitry Ulyanov. *Multicore-TSNE*. `https://github.com/DmitryUlyanov/Multicore-TSNE`. 2016.

[24] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. "How to Use t-SNE Effectively". In: *Distill* (2016). DOI: `10.23915/distill.00002`. URL: `http://distill.pub/2016/misread-tsne`.

[25] Kisung You and Dennis Shung. "Rdimtools: An R Package for Dimension Reduction and Intrinsic Dimension Estimation". In: *Software Impacts* 14 (2022), p. 100414. ISSN: 26659638. DOI: `10.1016/j.simpa.2022.100414`.

[26] Haili Zhang et al. “Out-of-sample data visualization using bi-kernel t-SNE”. In: *Information Visualization* 20.1 (2021), pp. 20–34. DOI: `10.1177/1473871620978209`. eprint: `https://doi.org/10.1177/1473871620978209`. URL: `https://doi.org/10.1177/1473871620978209`.