DRUID$_{JS}$ — A JavaScript Library for Dimensionality Reduction

Rene Cutura* TU Wien, University of Stuttgart
Christoph Kralj† University of Vienna
Michael Sedlmair‡ University of Stuttgart

ABSTRACT

Dimensionality reduction (DR) is a widely used technique for visualization. Nowadays, many of these visualizations are developed for the web, most commonly using JavaScript as the underlying programming language. So far, only few DR methods have a JavaScript implementation though, necessitating developers to write wrappers around implementations in other languages. In addition, those DR methods that exist in JavaScript libraries, such as *PCA*, *t-SNE*, and *UMAP*, do not offer consistent programming interfaces, hampering the quick integration of different methods. Toward a coherent and comprehensive DR programming framework, we developed an open source JavaScript library named DRUID$_{JS}$. Our library contains implementations of ten different DR algorithms, as well as the required linear algebra techniques, tools, and utilities.

Index Terms: Software and its engineering—Software notations and tools—Software libraries and repositories; Human-centered computing—Visualization—Visualization systems and tools—Visualization toolkits;

1 INTRODUCTION

DIMENSIONALITY REDUCTION (DR) is a technique used to reduce the total amount of dimensions in a given dataset in order to visualize the dataset or reduce the effects of the curse of dimensionality in machine learning pipelines. DR methods are important tools to project high-dimensional datasets into two or three dimensions in order to visualize them. Python, R, and Matlab provide extensive libraries for DR methods, while no extensive library exists for JavaScript yet. With the increasing popularity of web applications using powerful visualization tools such as D3 [5] and Vega [34], also the use of DR techniques has become popular. At the moment, a dedicated library for DR methods is still missing though, which is particularly problematic as different types of data require different DR algorithms [28]. It is not possible to use the most appropriate algorithm in every case as many DR methods have no JavaScript implementation.

At the moment, DR methods often require either a server-client structure or a precomputed embedding which can be loaded into the browser. The latter approach is easy to implement, but no interaction with the DR method is possible. It thus impedes the user from loading their own data or to interactively change DR methods and their parameterizations. Tools based on a server-client structure have privacy issues as the data needs to be sent to a server, preventing users from using those tools on sensitive data. To address these concerns, users can create custom implementations, but this is difficult, time-consuming, and error-prone. Additionally, many DR techniques are complex and need tools for unsupervised learning, linear algebra, stochastic, etc. which makes efficient implementation a difficult endeavour.

*e-mail: rene.cutura@tuwien.ac.at
†e-mail: christoph.kralj@univie.ac.at
‡e-mail: michael.sedlmair@visus.uni-stuttgart.de

To fill this gap, we contribute DRUID$_{JS}$[1], a JavaScript library for DR methods, targeted at researchers and developers. The main goal is to support them by providing access to the most frequently used DR techniques. Based on our experience working with DR for many years and the frequency of how they are used in the VIS literature at the moment [33], we equipped DRUID$_{JS}$ with an implementation of ten DR algorithms — some of which have no existing JavaScript implementations so far. The programming interface is designed based on best practices used in other libraries such as Scikit-learn [30] and D3 [5]. The DRUID$_{JS}$ library is dependency-free and therefore easily integrated into any project, by properly bundling all necessary tools required to use DR techniques. These necessary tools include matrix multiplications, LU-, QR-, and eigen-decompositions. In the current version, we focused on code readability and ease-of-use from a programmer's perspective.

2 RELATED WORK

The main component of our framework is a set of dimensionality reduction methods which are bundled into a JavaScript library.

There has been a plethora of work on DR methods and their usage. Recent surveys [22, 28, 33] characterized the most important DR algorithms and how they are used to interactively visualize high-dimensional data. Generally, the literature distinguished between two main groups of DR methods: linear and non-linear [39]. Linear techniques, such as *PCA* [29] allow for easy interpretation of the resulting space. In fact, this property makes *PCA* one of the most commonly used DR methods [35]. The results of non-linear methods, such as *t-SNE* [38] and *UMAP* [24], can uncover more complex high-dimensional structures. To do so, they most commonly try to preserve the neighborhoods as well as possible. However, projections might look very different with different starting conditions and might be harder to interpret by humans [40].

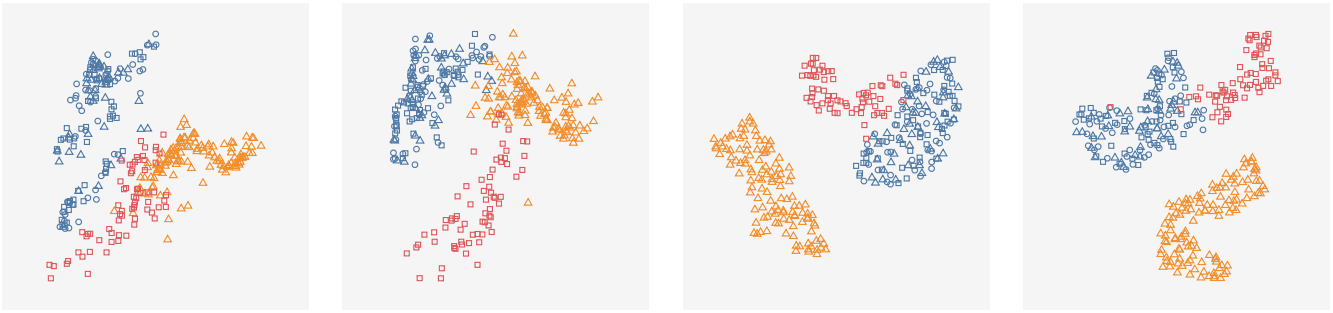
To the best of our knowledge, there is currently no JavaScript framework that implements more than two DR algorithm and works in the browser. For some implementations different libraries exist. MachineLearn.js [17] implements the *PCA* algorithm, which does *not* allow users to set the desired dimensionality and requires additional steps to create the reduced dataset. tSNEJS [16] implements the *t-SNE* algorithm and UMAP-js [19] implements the *UMAP* algorithm. Both use different programming interfaces (Ex. 7) and are therefore complicated to use together. HDSP [15] and ml-pca [18] are TypeScript libraries available for NodeJS only. Like MachineLearn.js, the older pca-js [14] uses the *SVD* [11] algorithm from numeric.js [23]. The library mdsjs [6] implements *PCA* and *LandmarkMDS* [1], but has no documentation and we thus do not include it in our comparative evaluation.

The alternative to JavaScript libraries are libraries in other programming languages like Scikit-learn [30] for Python, dimRed [20] for R, or drtoolbox [37] for Matlab. Our library does the projection directly in the browser and does *not* require a server to do the computations in a different language.

3 DRUID

We now explain the features of DRUID$_{JS}$. We start by showing the basic functions, before we depict some more advanced features.

[1] https://github.com/saehm/DruidJS



(a) *Isomap* projection with $k = 34$. Visualized result of Ex. 1.

(b) *Isomap* projection with $k = 50$. Result of Ex. 2.

(c) *UMAP* projection with default parameters. (First `transform` Ex. 3).

(d) *UMAP* projection with individual parameters. (Second `transform` Ex. 3).

Figure 1: DRUID$_{JS}$ projections of the 4D *Palmer Penguin* dataset, which consists of 342 different specimens of penguins of the species ● Adelie, ● Gentoo, and ● Chinstrap, living on one of the islands ○ Torgersen, △ Biscoe, or □ Dream of the Antarctica.

We use the *Palmer Penguin* [12] dataset for illustration. The dataset consists of 4 dimensions and 342 different specimens of penguins of the species ● Adelie, ● Gentoo, and ● Chinstrap, living on either one of the islands ○ Torgersen, △ Biscoe, or □ Dream of the Antarctica. The dataset contains features for the length and the depth of the culmen/beak, the flipper lengths, and the body mass. We normalized the values of each dimension and filtered data rows with missing values. Fig. 1 shows examples of projections with DRUID$_{JS}$.

Dimensionality Reduction We implemented the linear DR techniquees *PCA* [29], *LDA* [4], and *FASTMAP* [9], and the non-linear DR techniques *Isomap* [36], *LLE* [32], *LTSA* [41], *MDS* [21], *TriMap* [2], *t-SNE* [38], and *UMAP* [24]. All algorithms have the same interface, therefore we use only *Isomap*, *UMAP*, and *t-SNE* to showcase our design decisions regarding the programming interface (referred to as `druid.DR`). The values of the *Palmer Penguin* dataset are stored in the 2D array `penguins`. One line of code (see Ex. 1) is enough to project the data with *Isomap* in this case (Fig. 1a).

```
1   let projection = new druid.ISOMAP(penguins).transform();
```

Example 1: A new DR object `isomap` gets created with `new druid.ISOMAP`. It takes as argument the data. The data can be a 2D array or an object of the `druid.Matrix` class. The method `transform` does the actual projection, and returns the result in the input type.

Transform There are different ways to create a projection. The simplest one is to just call the function `transform` or the async variant `transform_async` on the `druid.DR` object. We use the method name "transform" to stick with the well established naming convention used in Scikit-learn, and as it describes best what happens to the input data: it is transformed into a space of different dimensionality.

Parameterization Parameters influence the results of a DR algorithm [8]. Therefore, it is often necessary to rerun a projection, because the optimization process gets stuck in a local minimum and the result is of bad quality. To tweak the parameters, each of them can be changed by using the method `parameter`.

Fig. 1a (the result of Ex. 1) shows superimposed ● Gentoos and ● Chinstraps. One reason for that could be a bad parameter value for *Isomap*'s k: the number of neighbors per point taken into account during the projection process of *Isomap*. The parameters depend on the respective DR method. In our *Palmer Penguin* example a different value for k changes the neighborhood connections and results in better visual separability of the ● Gentoos and ● Chinstraps in the final plot (Ex. 2 & Fig. 1b).

For users it is often unclear how to properly select these parameters. We thus offer reasonable defaults, but tweaking parameters might still often be necessary.

```
1   let isomap = new druid.ISOMAP(penguins)
2   isomap.parameter('k', 50);
3   projection = isomap.transform();
```

Example 2: Changing the parameter k of `isomap` to 50, followed by `transform` projecting the already defined *Palmer Penguin* dataset in Ex. 1. Fig. 1b shows the result.

Generally, similar to the method `attr` of D3 for setting and getting attributes of HTML elements, we used a method `parameter` (Ex. 3) which takes as arguments the name of the parameter (set and get) and the value of the respective parameter (set only). The method returns the `druid.DR` object itself when setting a parameter-value to allow method chaining (see Ex. 3 lines 4–8). Otherwise the value of the parameter gets returned.

```
1   let umap = new druid.UMAP(penguins);
2   umap.transform(); // first run with defaults
3
4   umap.parameter('local_connectivity', 1)
5       .parameter('min_distance', 2)
6       .parameter('metric', druid.manhattan)
7       .parameter('dimensionality', 2)
8       .transform(); // second run with new parameters
9   let seed = umap.parameter('seed');
```

Example 3: Changing the parameters of `umap`, a `druid.UMAP` object. The statement `umap.transform()` returns a projection with those parameters (see Fig. 1c & 1d). The variable `seed` gets the seed value from the `druid.UMAP` object.

Constructor The code used in Ex. 3 can be shortened by using the constructor of the respective DR class. The constructor takes as arguments the input data (see Ex. 1), then DR-dependent parameters, and then the hyper-parameters. The input data can be either a `druid.Matrix` object or a 2D array which gets internally converted to a `druid.Matrix` object.

```
1   new druid.UMAP(penguins, 1, 2, 2, druid.manhattan,
    ↪  seed).transform();
```

Example 4: Using the constructor shortens the code (from 5 lines to 1 line), but outputs the same as Ex. 3.

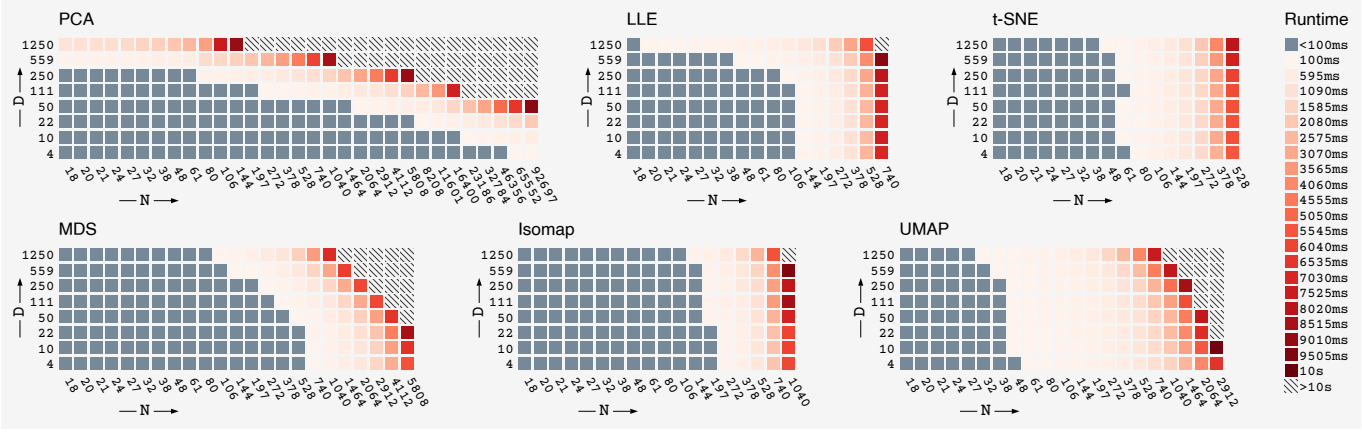


Figure 2: DRUIDJS runtimes of *PCA* [29], *LLE* [32], *t-SNE* [38], *MDS* [21], *Isomap* [36], and *UMAP* [24]. Each cell shows the mean time of 5 runs of the respective DR method using a generated random dataset according N points (x-axis), and D dimensions (y-axis). The gray cells ■ indicate a runtime under 100ms, which allows for on-the-fly user interactions [7, 26].

Generator Some DR methods like *t-SNE* (see Fig. 2) are computationally complex and require more time. To integrate such methods into an interactive visualization tool, the intermediate results can be plotted. Such intermediate results also provide insight into the DR algorithm and allow for progressive approaches [27]. We use a generator function to provide the user access to the intermediate results of the projection after each optimization step (Ex. 5).

```
let umap = new druid.UMAP(penguins);
for (let result of umap.generator()) {
    do_something(result); // i.e. draw
}
```

Example 5: A generator yielding the preliminary `result`'s until the optimization process of the DR algorithm stops.

Data Structure DRUIDJS's matrix class `druid.Matrix` stores the data values. The function `from` takes a 2D array and checks for proper shape of the array, and stores it efficiently in the memory as JavaScript typed arrays. The type of the return value of a DR algorithm depends on the input type. Internally,d DRUIDJS converts a 2D array to an object of the `druid.Matrix` class. When more than one projection of a dataset is planned, using a `druid.Matrix` object avoids redoing this step each time. To use a `druid.Matrix` object, for instance, with D3, calling the function `to2dArray` converts the object to a JavaScript 2D array.

Initialization Some DR algorithms comprise computational steps before the actual projection. For example, *MDS* and *t-SNE* require a distance matrix before projection. Reusing such previous computations reduces the runtime. These initialization steps are DR technique-specific and require a lookup by the user, but can provide precomputations to the DR algorithm (see Ex. 6).

```
let Δ = distance_matrix(penguins);
let mds = new druid.MDS().init(Δ).transform();
let tsne = new druid.TSNE().init(Δ).transform();
```

Example 6: The function `init` provides the precomputed distance matrix Δ to the `MDS`- and `TSNE` object.

4 EVALUATION

We evaluated DRUIDJS in three steps. First, we did an individual runtime analysis of DRUIDJS under different scales of data. Second, we compared these runtimes to common Python and some existing JavaScript DR implementations. Finally, we conducted a case study comparing code readability and runtime of DRUIDJS to existing JavaScript DR libraries.

4.1 Runtime analysis

To analyze the runtime of DRUIDJS, we generated random data for each pair of datasize $N \in \{\lfloor 16+2^{n/2} \rfloor, n \geq 3\}$ and dimensionality $D \in \{\lfloor 2 \cdot 5^{d/2} \rfloor, d \in \{1,\ldots,8\}\}$. Then, we projected each dataset five times with six DR methods, measured the runtime for each, and averaged the results. We picked the six most common ones due to page limit; the full set of ten DR methods and more detailed data can be found in the supplemental materials (also for the results in Sec. 4.2). We stopped further projections for a specific D if a projection with length N needed more than ten seconds — a common time limit for keeping a user's attention [7, 26]. The computations were run locally on a notebook with an Intel Core i7-8705G processor with 16GB memory using the Chrome browser 83.0.4103.116.

Looking at the results (see Fig. 2), we observe that most techniques are strongly affected by the number of points. *PCA* is less affected by N, but much more by a high D, while *t-SNE* is less affected by D, but much more by N. Besides *PCA* and *t-SNE*, all DRUIDJS methods have a reasonable runtime (under 1 second) for $N < 500$. For $N > 500$, visualizing intermediate results with the `generator` method can be a viable option.

We created an online demo[2], so that users/readers can get a qualitative "feeling" on the duration of these runtimes. The demo allows to select an $N \times D$ dataset and then computes the respective DRUIDJS projection on the fly.

4.2 Runtime comparison to other libraries

We now compare these runtimes to other frequently used implementations. In Python, we use Scikit-learn [30] and UMAP-learn [25] (Fig. 3 ● `sklearn*`). In terms of JavaScript, we use MachineLearn.js [17], tSNEJS [16], and UMAP-js [19] (Fig. 3 ● `js`).

The results (Fig. 3) show that DRUIDJS performs better than Scikit-learn for datasets with small N and small D for all methods. At some point Scikit-learn starts to be faster than DRUIDJS. This result is not surprising, as Scikit-learn has been heavily optimized for runtime, while we have not engaged in such optimizations yet. Scikit-learn implemented an iterative version of *MDS*, which is slower than the direct version of DRUIDJS.

[2] `https://renecutura.eu/druid_demo/`

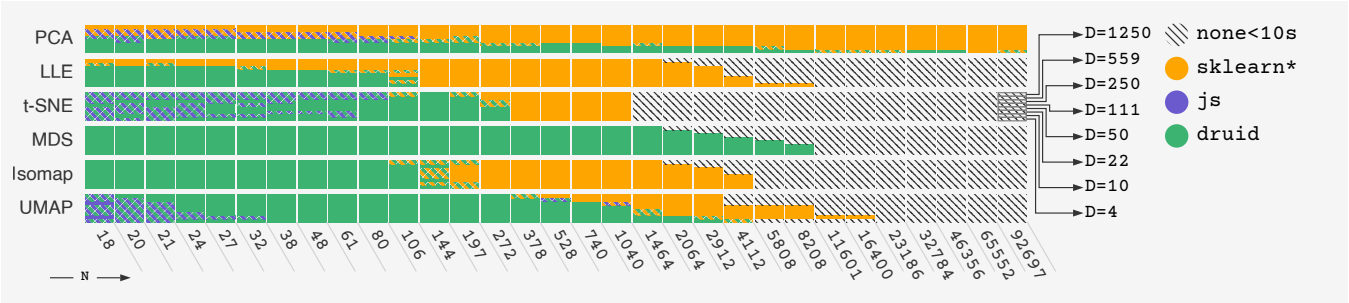


Figure 3: Each cell represents a dataset of N points and D dimensions. The rows represent the six most common DR techniques. For each dataset cell, we let ● `sklearn*`, ● `js`, and our ● DRUID$_{JS}$ implementations run 5 times each, and measured the runtimes. We selected the two fastest ones and color-coded them. This results in a single color cell if the two fastest were from the same implementation, and in stripes if they were from two different libraries. We can see, that for small N and small D, ● DRUID$_{JS}$ is consistently faster than ● `sklearn*` (green cells bottom left, yellow cells top right). For $N < 106$ ● DRUID$_{JS}$ and the existing ● `js` implementations lead to similar runtimes (striped green/blue cells).

```
1  import { tSNE } from 'tsnejs';
2  import { PCA } from 'machinelearn/decomposition';
3  import { UMAP } from 'umap-js';
4  import iris from 'IRIS_DATASET';
5  // compute t-SNE embedding
6  const tsne_dr = new tSNE();
7  tsne_dr.initDataRaw(iris);
8  for (let k = 0; k < 350; ++k) {
9      tsne_dr.step();
10 }
11 const tsne = tsne_dr.getSolution();
12 // compute PCA embedding
13 const pca_dr = PCA();
14 pca_dr.fit(iris);
15 const pca = iris.map(specimen => {
16     return [
17         specimen.map((col, i) => col *
       ↪  pca_dr.components[0][i])
18             .reduce((a, b) => a + b),
19         specimen.map((col, i) => col *
       ↪  pca_dr.components[1][i])
20             .reduce((a, b) => a + b)
21     ]
22 });
23 // compute UMAP embedding
24 const umap_dr = new UMAP().fit(iris);
```

Example 7: Projection of the *IRIS* data [10], without DRUID$_{JS}$. The library for *PCA* computes the principal components only, therefore a manual computation of the projection is needed.

When looking at ● `js` implementations all of them performed worse than DRUID$_{JS}$, except PCA on high dimensions.

4.3 Case Study

We use a common use case — the projection of a dataset to two dimensions for visualization purposes using different DR methods — to illustrate the code readability of DRUID$_{JS}$. We chose the well known *IRIS* dataset [10], consisting of 150 points with 4 dimensions for our example. We compare three existing JavaScript libraries with DRUID$_{JS}$ based on the runtime and the number of lines of code. To compare the runtime we executed the scripts Ex. 7 and Ex. 8 1000 times with equal parameterization. We performed 100 warm-up runs before taking measurements to ensure similar JIT compiler states and therefore improve the comparability. One run with DRUID$_{JS}$ (Ex. 8) took on average $922 \pm 109.82ms$, and with existing implementations (Ex. 7) on average $1051 \pm 70.64ms$. DRUID$_{JS}$ is, thus, roughly 10% *faster* and has just 5 (Ex. 8) lines of code instead of 21 (Ex. 7). In addition, tSNEjs requires to do the iterations manually (Ex. 7, lines 8–10) which requires good knowledge about the method to choose the right iteration count. MachineLearn.js's *PCA* computes only the principal components and requires the user to create the projection manually, worsening the code readability (Ex. 7, lines 15–22). The UMAP-js implementation has a state-of-the-art interface and requires only one line. With DRUID$_{JS}$ every DR method requires only one line and is likely easier to use (Ex. 8).

```
1  import druid;
2  import iris from 'IRIS_DATASET';
3  // compute t-SNE embedding
4  const tsne = new druid.TSNE(iris).transform();
5  // compute PCA embedding
6  const pca = new druid.PCA(iris).transform();
7  // compute UMAP embedding
8  const umap = new druid.UMAP(iris).transform();
```

Example 8: Same as Ex. 7, but with DRUID$_{JS}$. Same results, but with just one line of code for each DR method.

5 Limitations & Future Work

During the creation of DRUID$_{JS}$, our focus was the support of inexperienced users by offering reasonable default parameterizations. This brought us some feature limitations we plan to fix in future releases of DRUID$_{JS}$. One of these limitations regards the addition of more points, so called "out-of-sample extensions". Some DR algorithms allow the addition of more points to an existing projection. We do not support this so far as it is not possible for all algorithms and we wanted to keep the interface consistent over all algorithms. We implemented a few important DR techniques, but others such as *SVD* [11] are still missing. Of course, there is also still plenty of room for implementations that scale JavaScript DR methods to larger datasets, e.g., with GPU acceleration like tfjs-tsne [31].

6 Conclusion

In this paper, we present the JavaScript library DRUID$_{JS}$, which allows to consistently use different DR methods directly in the browser and integrate them into interactive tools. DRUID$_{JS}$ is based on a comprehensive programming interface using tried-and-tested coding conventions. As most VIS-tools are online nowadays [3, 13], we hope that DRUID$_{JS}$ will help to lower the entry barrier for developers who want to use DR, and will support applications where data privacy is of concern.

Acknowledgments

This work was supported by the FFG ICT of the Future program via the ViSciPub project (no. 867378).

REFERENCES

[1] Sparse Multidimensional Scaling using Landmark Points. Technical report.

[2] E. Amid and M. K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets, 2019.

[3] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated Extraction and Interpretation of Visualizations from the Web. In *ACM Conf. on Human Factors in Computing Systems (SIGCHI)*, pp. 1–8, 2018. doi: 10.1145/3173574.3174168

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research (JMLR)*, 3:993–1022, 2003.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[6] U. Brandes and C. Pich. Eigensolver Methods for Progressive Multidimensional Scaling of Large Data. In M. Kaufmann and D. Wagner, eds., *Graph Drawing*, vol. 4372 of *Lecture Notes in Computer Science*, pp. 42–53. Springer, 2007. doi: 10.1007/978-3-540-70904-6_6

[7] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *ACM Conf. on Human Factors in Computing Systems (SIGCHI)*, pp. 181–186, 1991. doi: 10.1145/108844.108874

[8] R. Cutura, S. Holzer, M. Aupetit, and M. Sedlmair. VisCoDeR: A Tool for Visually Comparing Dimensionality Reduction Algorithms. In *Euro. Symp. on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 641–646.

[9] C. Faloutsos and K.-I. D. Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *ACM Conf. on Management of Data (SIGMOD)*, pp. 163–174, 1995. doi: 10.1145/223784.223812

[10] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188, 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x

[11] G. H. Golub and C. Reinsch. Singular Value Decomposition and Least Squares Solutions. In *Linear Algebra*, pp. 134–151. Springer, 1971. doi: 10.1007/978-3-662-39778-7_10

[12] K. B. Gorman, T. D. Williams, and W. R. Fraser. Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins (Genus Pygoscelis). *PloS one*, 9(3):e90081, 2014. doi: 10.1371/journal.pone.0090081

[13] E. Hoque and M. Agrawala. Searching the Visual Style and Structure of D3 Visualizations. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 26(1):1236–1245, 2019. doi: 10.1109/TVCG.2019.2934431

[14] https://github.com/bitanath/pca. pca-js.

[15] https://github.com/IBM/projections. HDSP.

[16] https://github.com/karpathy/tsnejs. tSNEjs.

[17] https://github.com/machinelearnjs/machinelearnjs. MachineLearn.js.

[18] https://github.com/mljs/pca. ml-pca.

[19] https://github.com/PAIR-code/umap js. UMAP-js.

[20] G. Kraemer, M. Reichstein, and M. D. Mahecha. dimRed and coRanking—Unifying Dimensionality Reduction in R. *The R Journal*, 10(1):342–358, 2018. doi: 10.32614/RJ-2018-039

[21] J. B. Kruskal. Multidimensional Scaling by Optimizing Goodness of Fit to A Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, 1964. doi: 10.1007/BF02289565

[22] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing High-Dimensional Data: Advances in the Past Decade. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 23(3):1249–1268, 2016. doi: 10.1109/TVCG.2016.2640960

[23] S. Loisel. numeric.js.

[24] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, 2018.

[25] L. McInnes, J. Healy, N. Saul, and L. Grossberger. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software (JOSS)*, 3(29):861, 2018. doi: 10.21105/joss.00861

[26] R. B. Miller. Response time in man-computer conversational transactions. In *Fall Joint Computing Conference (AFIPS)*, pp. 267–277, 1968. doi: 10.1145/1476589.1476628

[27] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the Black Box: Strategies for Increased User Involvement in Existing Algorithm Implementations. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 20(12):1643–1652, 2014. doi: 10.1109/TVCG.2014.2346578

[28] L. G. Nonato and M. Aupetit. Multidimensional Projection for Visual Analytics: Linking Techniques with Distortions, Tasks, and Layout Enrichment. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 25(8):2650–2673, 2018. doi: 10.1109/TVCG.2018.2846735

[29] K. Pearson. LIII. 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(11):559–572, 1901. doi: 10.1080/14786440109462720

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011.

[31] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Höllt, B. Van Lew, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. GPGPU Linear Complexity t-SNE Optimization. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 26(1):1172–1181, 2019. doi: 10.1109/TVCG.2019.2934307

[32] S. T. Roweis and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000. doi: 10.1126/science.290.5500.2323

[33] D. Sacha, L. Zhang, M. Sedlmair, J. A. Lee, J. Peltonen, D. Weiskopf, S. C. North, and D. A. Keim. Visual Interaction with Dimensionality Reduction: A Structured Literature Analysis. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 23(1):241–250, 2016. doi: 10.1109/TVCG.2016.2598495

[34] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Trans. Visualization & Computer Graphics (TVCG)*, 22(1):659–668, 2015. doi: 10.1109/TVCG.2015.2467091

[35] M. Sedlmair, M. Brehmer, S. Ingram, and T. Munzner. Dimensionality Reduction in the Wild: Gaps and Guidance. Technical Report TR-2012-03, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada.

[36] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000. doi: 10.1126/science.290.5500.2319

[37] L. J. P. Van der Maaten. An Introduction to Dimensionality Reduction Using Matlab. Technical Report MICC 07-07, Universiteit Maastricht, Faculty of Humanities Sciences, MICC/IKAT, Maastricht, The Netherlands.

[38] L. J. P. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9:2579–2605, 2008.

[39] L. J. P. Van Der Maaten, E. Postma, and J. Van den Herik. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research (JMLR)*, 10(66–71):13, 2009.

[40] M. Wattenberg, F. Viégas, and I. Johnson. How to use t-SNE effectively. *Distill*, 1(10):e2, 2016. doi: 10.23915/distill.00002

[41] Z. Zhang and H. Zha. Principal Manifolds and Nonlinear Dimensionality Reduction via Tangent Space Alignment. *SIAM Journal on Scientific Computing*, 26(1):313–338, 2004. doi: 10.1137/S1064827502419154