The Gap between Visualization Research and Visualization Software (VisGap) (2020)
C. Gillmann, M. Krone, G. Reina, T. Wischgoll (Editors)

Selecting and Sharing Multidimensional Projection Algorithms: A Practical View

M. Espadoto [1†], E. F. Vernier[2] and A. C. Telea[3]

[1]Institute of Mathematics and Statistics, University of São Paulo, Brazil
[2]Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
[3]Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands

Abstract

Multidimensional Projection techniques are often used by data analysts for exploring multivariate datasets, but the task of selecting the best technique for the job is not trivial, as there are many candidates and the reasons for picking one over another are usually unclear. On the other hand, researchers developing new techniques can have a hard time comparing their new technique to existing ones and sharing their code in a way that makes it readily available for the public. In this paper, we try to address those issues systematically by analyzing recent surveys in the area, identifying the methods and tools used, and discussing challenges, limitations, and ideas for further work.

CCS Concepts

• ***Visualization*** → *Visual analytics;* • ***Computing methodologies*** → ***Dimensionality reduction and manifold learning;***

1. Introduction

Multidimensional projections, also called Dimensionality Reduction (DR) techniques, have a key place in the toolset of data scientists. They support many tasks dealing with high-dimensional data, such as *analysis* (finding interesting patterns in the data, *e.g.*, clusters, subspaces, or outliers); *simplification* (reducing the number of dimensions needed to capture the data structure); *prediction* (classification or regression tasks executed on the data samples); and *visualization* (presenting the data for exploration or communication via 2D or 3D scatterplots) [HG02, LMW*15, KH13, TLZM16].

Different audiences have different requirements for DR algorithms. *Practitioners* want to apply DR to solve their specific analysis, simplification, prediction, or visualization tasks on their data. Their main question relates to how to find the DR technique *implementation* that optimally covers their requirements, including adding such algorithms to their data-processing pipeline. *Researchers* that develop novel DR techniques need to compare their (new) technique with existing ones to demonstrate its added value, and next want to share it with an as wide as possible public.

Having worked in both practitioner and researcher roles in various teams for over 10 years, we have observed several challenges, which we summarize by two key questions, as follows:

- Q_P **(Practitioners):** How to choose the best DR algorithm implementation for my context from the wide set of options available in the public arena?
- Q_R **(Researchers):** How to compare my new DR algorithm against the existing ones, and next share it with as many other practitioners and/or researchers as possible?

Both questions can be answered in many ways, and using many instruments, *e.g.*, surveys, benchmarks, and open-source repositories. At a meta level, we ask ourselves: How to answer these questions *efficiently* and *effectively*? For practitioners, the search space (of existing DR algorithm implementations) is huge. How to approach the search process, starting from one's context-specific requirements, to find as quickly as possible the best DR techniques that fit these requirements? For researchers, the effort of developing new DR techniques is already large. How to ensure, with minimal effort, that the developed techniques are indeed better (and if so, how to measure this) than existing ones?

In this paper, we examine both above questions in a systematic way. We identify the *workflows* that typical practitioners and researchers follow when answering these questions. Next, we identify available *instruments* to complete each step of these workflows, and discuss the challenges and limitations we observed when applying these instruments. In particular, we propose an architecture for a benchmark for DR evaluation that is generic and extensible in terms of datasets, DR algorithms, quality metrics, and visualizations. Finally, we discuss ways forward for the community of practitioners and researchers interested in applying, respectively developing, DR algorithms.

2. Background

We first introduce a few definitions. A projection *technique* is a function $P : \mathbb{R}^n \rightarrow \mathbb{R}^m$, which maps a set $D \subset \mathbb{R}^n$ of n-dimensional points to a same-size m-dimensional scatterplot $P(D) \subset \mathbb{R}^m$, where typically $m \in \{2,3\}$. P aims to capture the structure of an input

† This study was financed in part by FAPESP (2017/25835-9) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

dataset D, by preserving various aspects thereof in $P(D)$, *e.g.*, inter-point distances or point neighborhoods. To quantify how well P preserves such data aspects, quality *metrics* [BTK11] are used. An *implementation* of a projection technique P is a reusable software component, *e.g.* executable or library, that computes the function P.

Two types of requirements come into play when answering Q_P and Q_R, as usual when engineering software systems:

- *functional requirements* describe properties of the DR *technique* itself. These include the type of data the projection accepts (nD samples or a distance matrix), whether P is deterministic or stochastic, linear or non-linear, global or local, quality (measured by quality metrics), computational and memory scalability, and out-of-sample ability. Such aspects can be usually inferred from the technique's description;
- *non-functional requirements* describe properties of *implementations* of DR techniques, *e.g.* ease of use, documentation, portability, third-party software components and programming language needed, and interoperability with other toolkits. Finding these typically requires one to study specific implementations of P.

The questions Q_P and Q_R listed in Sec. 1 have preoccupied both practitioners and researchers, increasingly more in the last decade, when many DR techniques have emerged in the literature. Several sources of information are typically used to answer them. We rank these sources based on how strongly they support answering Q_R and Q_P, on an ordinal scale ranging from '- -' (least) to '++' (most), as follows.

Papers ($Q_R: ++; Q_P: --$): Technical papers describing DR techniques are the prime information source for researchers aiming to understand and/or extend such techniques. Papers discuss functional aspects of the presented techniques well, but comparisons with other techniques are in general limited to a few. Non-functional aspects and *implementation* details are less thoroughly touched upon in papers, which leaves Q_P largely unanswered.

Surveys ($Q_R: +; Q_P: +$): Surveys compare (tens of) projections and consider more functional aspects (*e.g.*, metrics) than technical papers (see *e.g.* [NA18, vP09, EMK*19a, EHH12, SVM14]. Surveys offer a very good way to choose *techniques* based on their functional properties. Yet, surveys rarely discuss how to choose specific implementations of these techniques, and also discuss less non-functional aspects.

Benchmarks ($Q_R: -; Q_P: ++$): Benchmarks gather concrete datasets and projection/metric implementations to help both practitioners and researchers to compare *practically* DR techniques against each other. They also help replicability and are very common in other fields of computer science, *e.g.* machine learning [MCC*19] or medical imaging [RKHH09, MFM*13]. However, benchmarks are rare in the DR community. Three notable recent benchmarks are Espadoto *et al.* [EMK*19a] (18 datasets, 44 projection techniques, and 7 metrics); Vernier *et al.* [VGdS*20] (focus in dynamic DR – 10 datasets, 11 techniques, 12 metrics); and SmallVis [Mel] (focus on t-SNE, UMAP, and LargeVis [TLZM16]).

Frameworks ($Q_R: --; Q_P: ++$): Frameworks are collections of DR technique implementations designed, documented, and coded for reusability. They come as libraries, *e.g. scikit-learn* [PVG*11], Tapkee [LWG13]; and turnkey systems, *e.g.* MATLAB (which provides out-of-the-box implementations of PCA, Factor Analysis [Jol86], NMF [LS01], MDS [Tor58, Kru64] and t-SNE [vH08]), Projection-Explorer [JCC*11], and VisPipeline [vis], which provide end-to-end tools for interactive exploration of projections. An extreme model of frameworks are code bases that implement a single technique, *e.g.* Van der Maaten's t-SNE [vH08], dt-SNE [RFT16], and Espadoto's deep-learning DR technique [EHT19]. While frameworks best support Q_P, finding which implementations (in which frameworks) best match a practitioner's set of requirements still requires significant manual trial-and-error testing of the DR implementations they provide.

From the above, we see that the search space for Q_P and Q_R is large and heterogeneously structured. This affects both *practitioners* – it is not evident where to start searching, and how to systematically search; and *researchers* – there's no unanimously accepted answer to what to compare against, what to compare on, how to compare, and how to report the results; also, researchers face the question on how to best share their results with others, *e.g.*, simply release their code or take the time to integrate it with some framework. Hence, we find a salient gap between DR research and practice. Our aim next is to provide insights on how to fill this gap with a low effort, and where the largest unanswered challenges reside.

3. Operational Workflows

We identify two related, but not identical, workflows that practitioners, respectively researchers, follow to answer their respective questions Q_P and Q_R stated in Sec. 1. We inferred these workflows both from studies of existing papers and surveys in the DR literature, and from our own experience with answering both Q_P and Q_R in practice. Concerning our experience, we have studied 46 actual DR implementations. These implementations, and their main functional and non-functional aspects, are listed in Tab. 1. Additional functional aspects of these techniques, such as complexity and quality are provided in recent surveys [EMK*19a, VGdS*20].

Figure 1 depicts the steps of both these workflows, which we detail next. Colored dots in the middle table show which information sources (surveys, papers, benchmarks, or our own analysis in Tab. 1) are mainly used to support every workflow step. For each step, we next discuss the main questions asked by practitioners and researchers, outline how these can be answered, and which are the challenges we observed in doing this.

3.1. Practitioner workflow

A. Search techniques. *Where do I start searching?* Starting points for this search are, obviously, technical papers on DR techniques and, more broadly, surveys thereof [EMK*19a, NA18]. Additional search sources are conference presentations, blogs, and peer input. This search typically delivers a (large) subset of potentially suitable DR *technique* candidates (but usually no specific *implementations*).

B. Search implementations. *Where do I find implementations?* Table 1 provides our survey of available implementations, with frameworks providing these in Tab. 2. Related to the search process, we observed several points. Since many data science projects use Python, practitioners would likely favor Python-based DR implementations. We found out that scikit-learn [PVG*11] provides high-quality implementations of many well-known DR techniques. However, other techniques come in different languages (Tapkee [LWG13]: C++; Van Der Maaten's DR Toolbox [VdMPvdH07]: MATLAB; and Vispipeline [vis]: Java). These require manual Python wrapping, which is not easy for the average user, and may hamper adoption of less known DR techniques, which are not available anywhere else.

C. Select implementations. *How to do this selection?* This involves considering both functional and non-functional requirements. For the

Table 1: *Studied DR technique implementations with their functional and non-functional aspects listed.*

Projection	Linearity	Input	Neighborhood	Parameters	Out-of-sample	Deterministic	Implementation
Autoencoder [HS06]	NL	S	G	network size	yes	no	Keras
Diffusion Maps [LL06]	NL	S	L	2	no	yes	Tapkee
Factor Analysis [Jol86]	LIN	S	G	1	yes	yes	scikit-learn
Fastmap [FL95]	NL	D	G	0	no	yes	Vispipeline
GDA [BA00]	NL	D	G	1	no	yes	DR Toolbox
GPLVM [Law04]	NL	D	G	1	no	no	DR Toolbox
Fast ICA [Hyv99]	LIN	S	G	2	yes	yes	scikit-learn
IDMAP [MPL06]	NL	S	L	3	no	yes	Vispipeline
Isomap [TDL00]	NL	S	L	1	yes	yes	scikit-learn
Landmark Isomap [CCG06]	NL	S	L	1	no	no	Vispipeline
LAMP [JCC*11]	NL	S	L	3	yes	no	Vispipeline
Laplacian Eigenmaps [BN02]	NL	D	L	0	no	no	scikit-learn
LLC [TR02]	NL	S	L	3	no	yes	DR Toolbox
LLE [RS00]	NL	S	L	3	yes	no	scikit-learn
Hessian LLE [DG03]	NL	S	L	3	yes	no	scikit-learn
Modified LLE [ZW07]	NL	S	L	3	yes	no	scikit-learn
LMNN [WBS06]	LIN	S	L	3	no	yes	DR Toolbox
LPP [HN04]	LIN	S	G	1	yes	yes	Tapkee
LSP [PNML08]	NL	S	L	4	no	yes	Vispipeline
LTSA [ZZ04]	NL	S	L	3	yes	no	scikit-learn
Linear LTSA [ZYZG07]	LIN	S	L	1	no	no	Tapkee
Manifold Charting [Bra02]	NL	S	L	2	no	yes	DR Toolbox
MCML [GR06]	NL	S	L	0	no	no	DR Toolbox
MDS [Tor58]	NL	D	G	2	no	no	scikit-learn
Landmark MDS [DT04]	NL	D	G	1	no	no	Tapkee
Nonmetric MDS [Kru64]	NL	S	G	2	no	no	scikit-learn
Landmark MVU [WPS05]	NL	S	G	2	no	no	DR Toolbox
NMF [LS01]	LIN	S	G	4	yes	no	scikit-learn
PBC [PM06]	NL	S	L	4	no	yes	Vispipeline
PCA [Jol86]	LIN	S	G	0	yes	yes	scikit-learn
Incremental PCA [RLLY08]	LIN	S	G	0	yes	no	scikit-learn
Kernel PCA (polynomial) [SSM98]	NL	S	G	1	yes	no	scikit-learn
Kernel PCA (RBF) [SSM98]	NL	S	G	1	yes	no	scikit-learn
Kernel PCA (Sigmoid) [SSM98]	NL	S	G	1	yes	no	scikit-learn
Probabilistic PCA [TB99]	LIN	S	G	1	yes	yes	DR Toolbox
Sparse PCA [ZHT06]	LIN	S	G	3	yes	yes	scikit-learn
PLSP [PEP*11]	NL	S	G	0	no	yes	Vispipeline
Random Projection (Gaussian) [Das00]	NL	S	G	0	yes	no	scikit-learn
Random Projection (Sparse) [Das00]	NL	S	G	0	yes	no	scikit-learn
Rapid Sammon [PdRDK99]	NL	S	G	0	yes	no	Vispipeline
t-SNE [vH08]	NL	D	L	3	no	no	Multicore TSNE, t-SNE archive, TFJS-t-SNE
SPE [Agr03]	NL	S	G	2	no	no	Tapkee
Truncated SVD [HMT09]	LIN	S	G	1	yes	no	scikit-learn
UMAP [MHM18]	NL	D	L	3	yes	no	umap-learn
dt-SNE [RFT16]	NL	D	L	4	no	no	dt-SNE repository
NNproj [EHT19]	NL	S	L	network size	yes	yes	NNProj code

Table 2: *DR frameworks used in the study.*

Framework name	Available at	Programming Language(s)	Documentation Quality
DR Toolbox	lvdmaaten.github.io/drtoolbox	MATLAB	−−
Multicore TSNE	github.com/DmitryUlyanov/Multicore-TSNE	Python and C++	−−
scikit-learn	scikit-learn.org	Python	++
Tapkee	tapkee.lisitsyn.me	C++	−−
umap-learn	github.com/lmcinnes/umap	Python	++
Vispipeline	vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex	Java	−−
Keras	keras.io	Python	++
t-SNE archive	https://lvdmaaten.github.io/tsne	MATLAB, Python, C++, JavaScript, CUDA, R, Java	+
dt-SNE repository	https://github.com/paulorauber/thesne	Python	−−
TFJS-t-SNE	https://nicola17.github.io/tfjs-tsne-demo	Python + WebGL	+
NNproj repository	https://github.com/mespadoto/dlmp	Python	+

former, one can easily screen DR techniques based on their properties listed in recent surveys [EMK*19a,NA18]. For the latter, we point to our own survey in Tabs. 1 and 2. Based on both requirement types, a ranking is made and a subset of candidates are selected. We observe several points concerning non-functional requirements. Regarding *documentation*, outside of scikit-learn and UMAP, which are well documented, most libraries we found had little to no documentation at all (Tab. 2), making adoption hard. In some cases, reading the source code is the only option, like in the case of many techniques found in Van Der Maaten's DR Toolbox. Even in the cases where the library is well documented, we often found not enough details on the role of each parameter in the final quality of the projection, and even less on the interaction between parameters. Separately, the *API design* promoted by DR libraries can vary enormously. Each library has its own conventions on data format and parameters, which makes the problem of interfacing hard. Take, for example, the examples below, which run t-SNE [vH08] on some dataset X using scikit-learn and Tapkee (Listings 1 and 2). We believe the two-step API of scikit-learn (create object with parameters, call fit_transform()) to be much simpler to understand for the average user than Tapkee's method of chaining with globally-namespaced, non-specific keywords.

Listing 1: *t-SNE example with scikit-learn*

```
from sklearn.manifold import TSNE
tsne = TSNE(perplexity=30)
output = tsne.fit_transform(X)
```

Listing 2: *t-SNE example with Tapkee*

```
using tapkee;
TapkeeOutput tsne = initialize()
  .withParameters((
    method=tDistributedStochasticNeighborEmbedding,
    target_dimension=2,
    sne_perplexity=30))
```

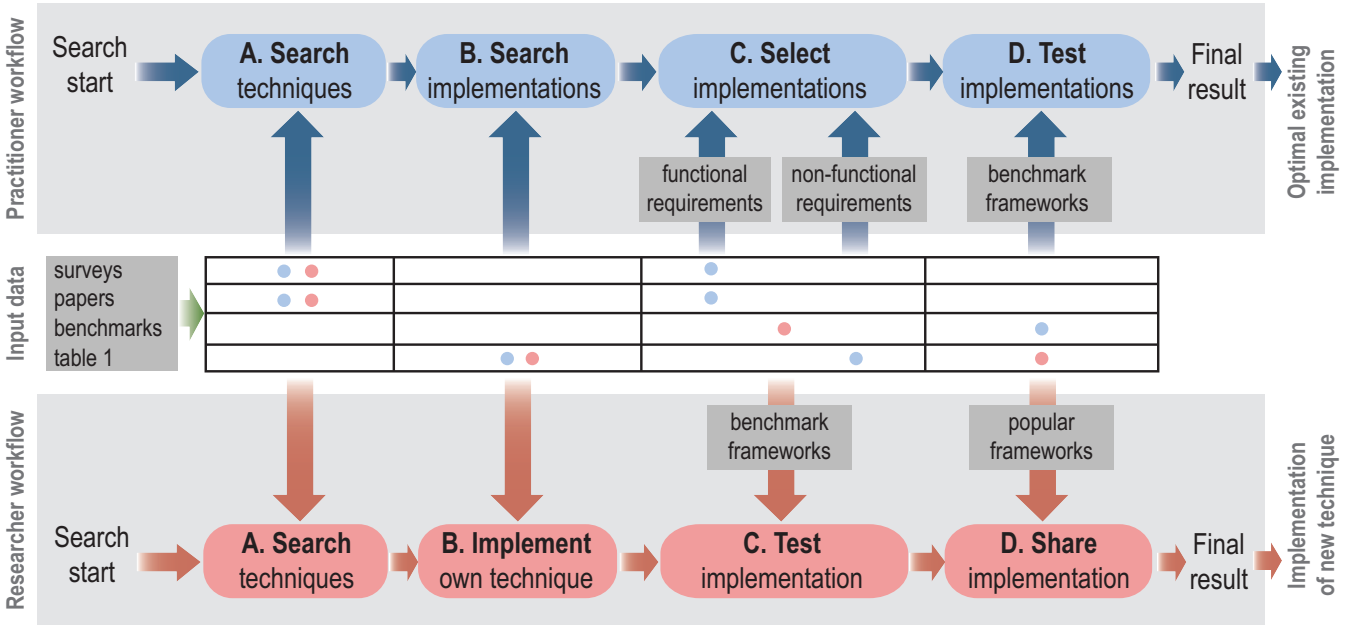

Figure 1: *Workflows for selecting DR implementations for practitioners (top, see Sec. 3.1) and researchers (bottom, see Sec. 3.2). Colored dots indicate which input data (surveys, papers, benchmarks, Tab. 1) are used by the two roles in each step.*

```
    .embedUsing(X);
auto output = tsne.embedding.transpose();
```

D. Test implementations. *How to test the selected implementations?* Surveys and code inspection cannot tell everything about a specific DR implementation. One needs to actually test an implementation on given datasets and against a set of metrics. For this, *benchmarks* are needed. To our knowledge, only three such benchmarks exist in DR landscape [EMK*19a, VGdS*20, Mel]. Yet, such benchmarks are never complete, so they need to be extended with additional DR implementations, datasets, and metrics. We discuss how such benchmarks can be designed for extensibility next in Sec. 4.

3.2. Researcher workflow

A. Search techniques. This step is largely similar to step A for practitioners (Sec. 3.1). The focus is though different for researchers, who are mainly interested in finding *functional* limitations of existing DR techniques that they want to improve upon, rather than the non-functional ones that are more relevant to practitioners.

B. Implement own technique. This step can proceed, for the most part, independently from existing DR implementations. However, some researchers may choose to follow coding standards and API conventions of existing (successful) DR implementations to already maximize exposure at this stage.

C. Test implementation. This step typically uses the same benchmarks as in step D for practitioners. An important part of this step is *presenting* the test results. While less important for practitioners, researchers need compelling ways to show that their (novel) techniques perform well *vs* many other techniques on many metrics to convince their peer researchers. For this, metric *visualizations* are used, the most prominent being bar and boxplot charts [vP09], tables [NA18], and space-filling charts [EMK*19a]. A challenging aspect here is that the space to visualize is at least four-dimensional (datasets, metrics, DR techniques, parameter settings). Ideally, creating such visualizations, and adding new visualizations, should be supported by the benchmarks.

D. Share implementation. Once a DR implementation has been successfully tested, its further impact critically depends on how easily it is *shared* with practitioners. Doing this follows two approaches. *Standalone code* is the easiest way – the researcher just makes her DR implementation available via a website or software repository. Examples hereof are dt-SNE, NNproj, and DR Toolbox. One issue with this approach is that integrating DR code having non-standard APIs with other components of a data science pipeline can be hard. Also, standalone code is less visible to practitioners than code shared via well-known frameworks (discussed next). Finally, adaptive maintenance is less often done on standalone code, which can easily break it upon the evolution of third-party components it uses. An example hereof is dt-SNE, which depends on the unmaintained Theano [the] library. However, good examples of standalone code sharing exist, such as t-SNE archive and UMAP (Tab. 2), discussed below. *Framework integration*, the second approach, adds the DR implementation to a mainstream data science or similar framework. Examples hereof are scikit-learn, Tapkee, MATLAB, and Vispipeline. This takes considerably more effort than sharing standalone code, as the code to integrate must comply with framework APIs and documentation constraints, but favors (far) larger exposure. Framework integration can be hard for DR techniques which need more than data input-output communication with the framework. Examples hereof are projections which advertise landmarks interactively placed by the user, such as LAMP [JCC*11] or for techniques which use GPU acceleration, *e.g.* TFJS-t-SNE [PTM*19]. To integrate these, a framework should provide APIs for interaction, respectively for GPU computing.

Standalone code sharing has limitations, but can still be very successful. Two good examples are UMAP [MHM18] and t-SNE

[vH08]. UMAP's author did an excellent job of providing a Python library which is API-compatible with scikit-learn, easy to install and use, and well documented. We believe the availability of the UMAP library via the standard Python package manager and its scikit-learn compatibility are key factors (apart from UMAP's quality) of its growing popularity. In contrast, the t-SNE archive (maintained by Van der Maaten) chose to provide a 'portal' for implementations in several languages (C++, Python, JavaScript, CUDA, R, Java, MATLAB), thereby simplifying integration with third-party code in all these languages.

4. Architecting an Evaluation Benchmark

Having a *benchmark* to quantitatively evaluate DR algorithms is essential for both the practitioner and researcher workflows (see Sec. 3). As mentioned there, not many such benchmarks exist. Importantly, by a benchmark, we do not understand here just a 'passive' collection of high-dimensional datasets to battle-test DR techniques, but rather a *runnable* software system that lets one select datasets, DR technique implementations (and their parameter values), metrics, execute them, and visually inspect the results in an easy and highly automated way. The only two benchmarks that approach this definition are [EMK*19a] (for static DR techniques) and [VGdS*20] (for dynamic DR techniques). However, these benchmarks have their own limitations. Extending them involves, at points, manually reading and reverse-engineering their code, which is hard. Creating an even better (broader, easier to use) benchmark is a high-effort task involving many decisions.

We aim to support the interested users in either the extension task or the design-from-scratch task by proposing a generic architectural template for such a benchmark (see Fig. 2. We created this architecture by studying the two benchmarks [EMK*19a, VGdS*20], including the implementation [EMK*19b] of the former, and next unifying their design and implementation, aiming to generalize and simplify. We believe that our proposal meets well the genericity and extensibility requirements, as detailed next.

Overall design: The benchmark follows a dataflow execution model (see *execute* function in Fig. 2). Datasets d from a database $\mathcal{D}$ are projected in turn by several DR technique implementations p from a DR technique collection $\mathcal{P}$, using several parameter values *params*, yielding corresponding 2D scatterplots $d_{2D} = p(d, params)$. For each such scatterplot, a set of projection quality metrics $m(d, d_{2D})$ is computed. The results d_{2D} and m are stored in a result database $\mathcal{R}$, implemented using the Python "pickle" binary format files for efficiency. These results can be next visually explored by visualizations selected by the analyst from a given set $\mathcal{V}$. We next detail each of the main components outlined above.

Dataset collection $\mathcal{D}$: This is the set of datasets to be considered in the evaluation, stored as a name-value dictionary (d_{name}, d_{data}). The values d_{data} are URLs pointing to actual files that store the data samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$, each having n dimensions $\mathbf{x}^1, \ldots, \mathbf{x}^n$, in a table format, using either CSV following the 'tidy data' [Wic14] standard or binary NumPy [WCV11].

Projection techniques $\mathcal{P}$: This is the set of DR technique implementations to be evaluated, as well as their parameters to be used during evaluation, stored as a set of tuples $(p_{name}, p_{code}, p_{params}$. Here, p_{code} points to the Python implementation of a DR technique, which is a function that expects a dataset *dataset* and parameter-set *params* , and returns the computed 2D scatterplot d_{2D}. p_{params} stores a so-called *parameter grid*, *i.e.* a table having as many columns as parameters p expects, and one row per parameter-set to be used during the evaluation. For instance, if we want to evaluate a t-SNE implementation of p that expects two parameters *perplexity* and number of *iterations* (see [vH08] for details on these parameters), which range in $perplexity \in \{20, 40\}$ and $iterations \in \{100, 150, 200\}$, we provide a parameter-grid p_{grid} table equal to the Cartesian product $\{20, 40\} \times \{100, 150, 200\}$. This design allows flexibly evaluating DR techniques having different numbers and types of parameters over user-supplied parameter grids.

Metrics $\mathcal{M}$: This is the set of quality metrics to be used to assess the benchmarked projections, stored as a dictionary $(m_{name}, m_{code}$. Here, m_{code} points to the Python implementation of a metric, which is a function that expects a dataset $d \in \mathcal{D}$ and its 2D projection d_{2D} computed by one of the techniques $p \in \mathcal{P}$, and returns a metric value (typically a scalar). As for projections, this design allows easily incorporating any of the projection quality metrics known in the literature.

Visualizations $\mathcal{V}$: This is the set of visualization tools offered to the analyst to explore an evaluated benchmark. Each visualization *Vis* is a Python function receiving a tuple $(d_{name}, p_{name}, params, d_{2D}, m)$. If a parameter is set to the predefined value *each*, then the visualization will generate separate small-multiples for each of the values of that parameter in $\mathcal{R}$. If a parameter is set to the predefined value *aggregate*, then the visualization will generate a single plot for all values of that parameter. These two options are conceptually analogous to the SQL operations *SELECT **, respectively *SELECT SUM*. This allows one to easily specify visualizations having different levels of data aggregation. For example, if we want to display quality metrics, setting $d_{name} = each, p_{name} = each$ creates one separate metric plot for each different pair of dataset and DR technique in $\mathcal{R}$. Setting $d_{name} = each, p_{name} = aggregate$ creates one metric plot that shows, for each dataset, the aggregate (*e.g.*, average, depending on the actual *Vis* implementation) values of metrics over all DR techniques. Finally, setting a parameter to a specific value, *e.g.* $p_{name} = t\text{-}SNE$, creates a visualization only for the respective DR technique entries present in $\mathcal{R}$. The dictionary keys $d_{name}, p_{name}, m_{name}$ show now their purpose: They are used both for the user to select specific entries in $\mathcal{R}$ to visualize and to create labels in the generated visualizations. This design allows specifying a quite large range of visualizations, see the examples in [EMK*19a, VGdS*20].

In terms of implementation, actual visualizations can be coded as Python scripts calling Matplotlib [Hun07] (as in [EMK*19a]). A more interactive and flexible development workflow can be achieved by using Jupyter notebooks that allow for independent and interactive execution of their code cells and rich presentation of their output (visualizations, narrative text, mathematical equations, tables). Execution automation is supported by Papermill [pap] which allows the parameterization, instantiation, and execution of Jupyter notebooks. For example, to generate for each dataset d a separate video showing a small-multiple display of all its time-dependent projections $p(d)|d \in \mathcal{D}$, one can write a template video generation notebook and use Papermill to instantiate a new notebook for each dataset d.

Extensibility and genericity: The above architecture is easily extensible: Adding new datasets, metrics, DR techniques, or parameter grids implies simply adding entries to the respective dictionaries. Dictionaries can be implemented either in Python or, even simpler, as folders having filenames as keys and the respective file contents as actual values. For large benchmarks, implementations using relational databases (*e.g.*, SQL) could also be done relatively easily following the same template architecture. The architecture is also generic, since the formats of datasets, respectively signatures of functions implementing DR techniques, metrics, and visualizations, can

accommodate most, if not all, concrete instantiations thereof that we know of from the DR literature and practice. For instance, all DR technique implementations in Tab. 1 fit this architecture. Parallelization can also be easily incorporated, *e.g.*, by simple running of multiple processes at the level of the for-loops in *execute* (a design used in [EMK*19a]). Finally, supporting time-dependent DR techniques implies only a small change to the architecture, namely having d_{data} point to a *sequence* of tables rather than a single one (as done in [VGdS*20]).

5. Discussion

Summarizing, we see a number of open challenges to the selection and sharing of DR algorithms that, jointly, create a gap between the state-of-the art work in DR literature and practical realities in the field, as follows.

Implementation: For *selection* (adoption), we note the lack of analysis of DR *implementations* with respect to non-functional requirements such as programming language, documentation quality, and ease of use. These make practitioners stay away from techniques whose implementations score poorly along these requirements and, conversely, favor techniques that have good-scoring implementations. For sharing, we see that there is no single framework that provides implementations of most DR techniques known in the literature – the closest to this is scikit-learn which implements roughly half of the DR techniques for which we found a mainstream implementation. For *sharing*, we do not see yet a momentum for researchers to develop DR algorithms within mainstream data science frameworks – the dominant sharing form is still standalone code.

Benchmarks – Datasets: Selecting a representative collection of datasets to gauge DR techniques is hard. Typically, papers, surveys, and benchmarks use datasets that are known in DR literature (for historical reasons) or in a given application domain. However, gauging the quality of a DR technique *at large* should use datasets that ideally represent well any problem. Espadoto *et al.* [EMK*19a] do a first attempt in this direction by characterizing datasets by traits (*e.g.*, dimensionality, intrinsic dimensionality, sparsity, provenance) and create a benchmark by sampling these dimensions. Vernier *et al.* [VGdS*20] use the same idea and aim to also cover dataset dynamics. However, both these surveys admit to only very sparsely sample the space of all possible datasets. Coming up with a good such sampling is an open, and important problem, in DR practice.

Benchmarks – Techniques: To our knowledge, most DR surveys and benchmarks focus on techniques that handle quantitative data, and static projections (except [VGdS*20]). Adding DR techniques that handle other attribute types such as categorical is another open direction towards creating comprehensive benchmarks.

Benchmarks – Metrics: DR literature knows tens of different quality metrics [BTK11]. However, existing benchmarks implement only very few – the most being, to our knowledge, the 6 metrics in [EMK*19a]. A benchmark with a wide set of readily-implemented metrics would be of high value to both practitioners and researchers.

Benchmarks – Extensibility: Some DR benchmarks [EMK*19a, EMK*19a] provide code that allows the experiments to be reproduced and allow for some extensibility, in terms of adding new projection techniques, metrics and datasets. In terms of storing results and creating visualizations, users would benefit from a more structured approach, with data saved in portable formats, and with some form of integration with popular data visualization and exploration tools, such as Tableau [CSH03].

6. Conclusions

We presented an overview of the challenges that exist in the process of selecting and sharing DR techniques from the point of view of different audiences, practitioners and researchers, where we described the typical workflows used in their processes. We listed and ranked the most common sources of information about DR techniques, namely, papers, surveys, benchmarks, and frameworks, and compiled a list with popular frameworks and techniques. We described the architecture used in two recent benchmarks and showed how these can be extended to consider more datasets, techniques, and metrics.

We believe the visualization community would benefit from a more integrated, well-documented benchmark framework, where new techniques, datasets, and metrics could be easily added by the user with minimal programming, and with the capability of integration with existing visualization tools.

References

[Agr03] AGRAFIOTIS D. K.: Stochastic proximity embedding. *J Comput Chem 24*, 10 (2003), 1215–1221. 3

[BA00] BAUDAT G., ANOUAR F.: Generalized discriminant analysis using a kernel approach. *Neural computation 12*, 10 (2000), 2385–2404. 3

[BN02] BELKIN M., NIYOGI P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proc. NIPS* (2002), pp. 585–591. 3

[Bra02] BRAND M.: Charting a manifold. In *Proc. NIPS* (2002), pp. 985–992. 3

[BTK11] BERTINI E., TATU A., KEIM D.: Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE TVCG 17*, 12 (2011), 2203–2212. 2, 6

[CCG06] CHEN Y., CRAWFORD M., GHOSH J.: Improved nonlinear manifold learning for land cover classification via intelligent landmark selection. In *Proc. IEEE IGARSS* (2006), pp. 545–548. 3

[CSH03] CHABOT C., STOLTE C., HANRAHAN P.: Tableau software. *Tableau Software* (2003). 6

[Das00] DASGUPTA S.: Experiments with random projection. In *Proc. UAI* (2000), Morgan Kaufmann, pp. 143–151. 3

[DG03] DONOHO D. L., GRIMES C.: Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proc. of the National Academy of Sciences 100*, 10 (2003), 5591–5596. 3

[DT04] DE SILVA V., TENENBAUM J. B.: *Sparse multidimensional scaling using landmark points*. Tech. rep., Stanford University, 2004. 3

[EHH12] ENGEL D., HÜTTENBERGER L., HAMANN B.: A survey of dimension reduction methods for high-dimensional data analysis and visualization. In *Proc. Dagstuhl IRTG Workshop* (2012), pp. 135–149. 2

[EHT19] ESPADOTO M., HIRATA N. S., TELEA A. C.: Deep learning multidimensional projections, 2019. arXiv:1902.07958. 2, 3

[EMK*19a] ESPADOTO M., MARTINS R. M., KERREN A., HIRATA N. S., TELEA A. C.: Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG* (2019). 2, 3, 4, 5, 6

[EMK*19b] ESPADOTO M., MARTINS R. M., KERREN A., HIRATA N. S., TELEA A. C.: Towards a quantitative survey of dimension reduction techniques - companion site. mespadoto.github.io/proj-quant-eval/, 2019. 5

[FL95] FALOUTSOS C., LIN K.: FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *ACM SIGMOD 24*, 2 (1995), 163–174. 3

[GR06] GLOBERSON A., ROWEIS S. T.: Metric learning by collapsing classes. In *Proc. NIPS* (2006), pp. 451–458. 3

[HG02] HOFFMAN P., GRINSTEIN G.: A survey of visualizations for high-dimensional data mining. *Information Visualization in Data Mining and Knowledge Discovery 104* (2002), 47–82. 1

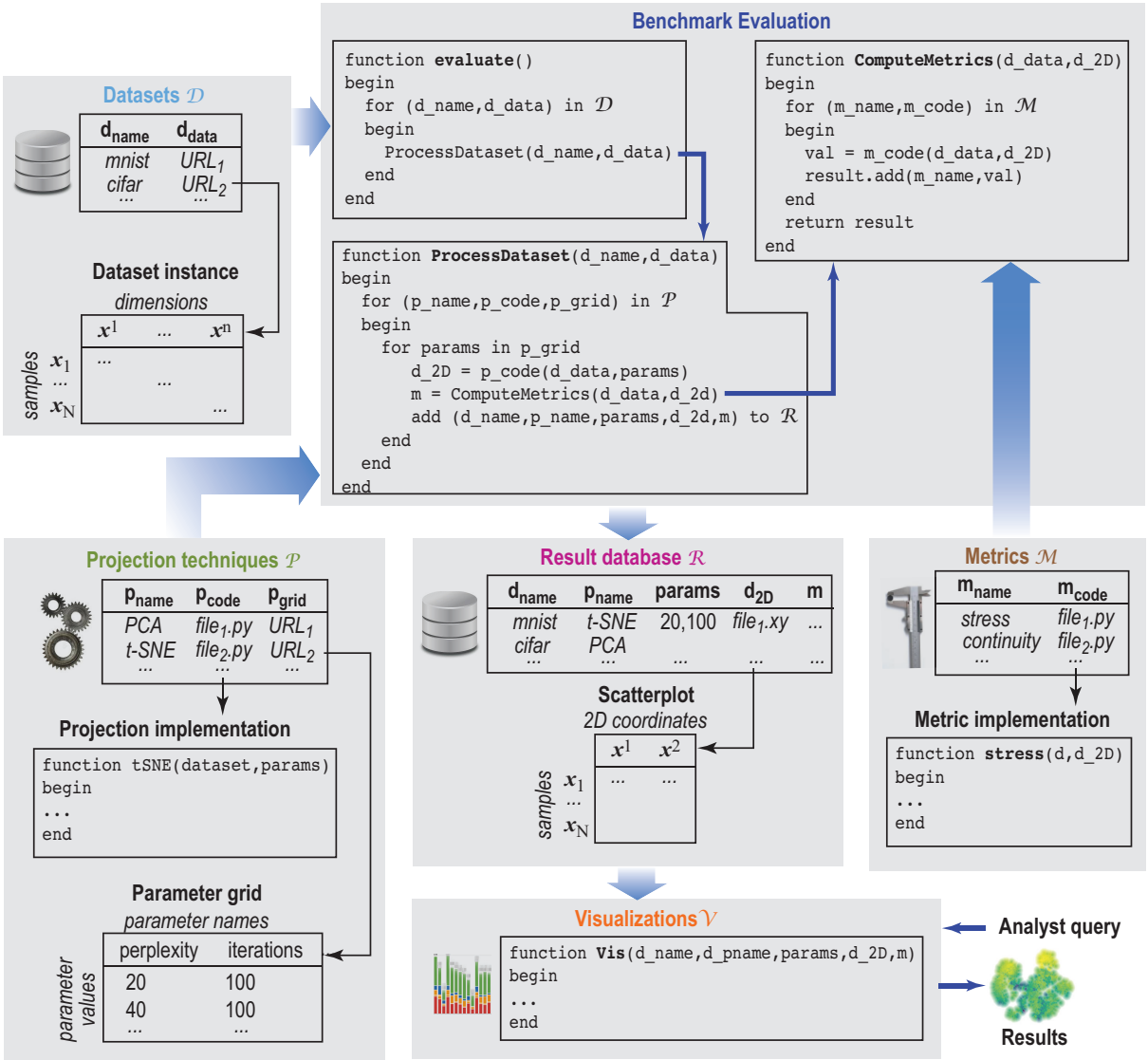

Figure 2: *Proposed benchmark architecture and its dataflow execution workflow (Sec. 4).*

[HMT09] HALKO N., MARTINSSON P.-G., TROPP J. A.: Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions, 2009. arXiv:0909.4061 [math.NA]. 3

[HN04] HE X., NIYOGI P.: Locality preserving projections. In *Proc. NIPS* (2004), pp. 153–160. 3

[HS06] HINTON G., SALAKHUTDINOV R.: Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507. 3

[Hun07] HUNTER J. D.: Matplotlib: A 2d graphics environment. *Computing in science & engineering 9*, 3 (2007), 90–95. 5

[Hyv99] HYVARINEN A.: Fast ICA for noisy data using Gaussian moments. In *Proc. IEEE ISCAS* (1999), vol. 5, pp. 57–61. 3

[JCC*11] JOIA P., COIMBRA D., CUMINATO J. A., PAULOVICH F. V., NONATO L. G.: Local affine multidimensional projection. *IEEE TVCG 17*, 12 (2011), 2563–2571. 2, 3, 4

[Jol86] JOLLIFFE I. T.: Principal component analysis and factor analysis. In *Principal Component Analysis*. Springer, 1986, pp. 115–128. 2, 3

[KH13] KEHRER J., HAUSER H.: Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE TVCG 19*, 3 (2013), 495–513. 1

[Kru64] KRUSKAL J.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika 29*, 1 (1964), 1–27. 2, 3

[Law04] LAWRENCE N.: Gaussian process latent variable models for visualisation of high dimensional data. In *Proc. NIPS* (2004), pp. 329–336. 3

[LL06] LAFON S., LEE A. B.: Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and dataset parameterization. *IEEE TVCG 28*, 9 (2006), 1393–1403. 3

[LMW*15] LIU S., MALJOVEC D., WANG B., BREMER P.-T., PASCUCCI V.: Visualizing high-dimensional data: Advances in the past decade. *IEEE TVCG 23*, 3 (2015), 1249–1268. 1

[LS01] LEE D. D., SEUNG H. S.: Algorithms for non-negative matrix factorization. In *Proc. NIPS* (2001), pp. 556–562. 2, 3

[LWG13] LISITSYN S., WIDMER C., GARCIA F. J. I.: Tapkee: An efficient dimension reduction library. *JMLR 14*, 1 (2013), 2355–2359. 2

[MCC*19] MATTSON P., CHENG C., COLEMAN C., DIAMOS G., MI-

CIKEVICIUS P., PATTERSON D., TANG H., WEI G.-Y., BAILIS P., BITTORF V., ET AL.: MLPerf benchmark. *arXiv:1910.01500* (2019). 2

[Mel] MELVILLE J.: SmallVis benchmark. github.com/jlmelville/smallvis. 2, 4

[MFM*13] MENDONÇA T., FERREIRA P. M., MARQUES J. S., MARCAL A. R., ROZEIRA J.: Ph 2-a dermoscopic image database for research and benchmarking. In *2013 35th annual international conference of the IEEE engineering in medicine and biology society (EMBC)* (2013), IEEE, pp. 5437–5440. 2

[MHM18] MCINNES L., HEALY J., MELVILLE J.: UMAP: Uniform manifold approximation and projection for dimension reduction, 2018. arXiv:1802.03426v2 [stat.ML]. 3, 4

[MPL06] MINGHIM R., PAULOVICH F. V., LOPES A. A.: Content-based text mapping using multi-dimensional projections for exploration of document collections. In *Proc. SPIE* (2006). 3

[NA18] NONATO L., AUPETIT M.: Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG* (2018). 2, 3, 4

[pap] Papermill. papermill.readthedocs.io. 5

[PdRDK99] PEKALSKA E., DE RIDDER D., DUIN R. P. W., KRAAIJVELD M. A.: A new method of generalizing Sammon mapping with application to algorithm speed-up. In *Proc. ASCI* (1999), vol. 99, pp. 221–228. 3

[PEP*11] PAULOVICH F., ELER D., POCO J., , BOTHA C., MINGHIM R., NONATO L.: Piecewise laplacian-based projection for interactive data exploration and organization. *CGF 30*, 3 (2011), 1091–1100. 3

[PM06] PAULOVICH F., MINGHIM R.: Text map explorer: a tool to create and explore document maps. In *Proc. IEEE IV* (2006), pp. 245–251. 3

[PNML08] PAULOVICH F. V., NONATO L. G., MINGHIM R., LEVKOWITZ H.: Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG 14*, 3 (2008), 564–575. 3

[PTM*19] PEZZOTTI N., THIJSSEN J., MORDVINTSEV A., HOLLT T., VAN LEW B., LELIEVELDT B., EISEMANN E., VILANOVA A.: GPGPU linear complexity t-SNE optimization, 2019. arXiv:1805.10817 [cs.LG]. 4

[PVG*11] PEDREGOSA F., VAROQUAUX G., GRAMFORT A., MICHEL V., THIRION B., GRISEL O., BLONDEL M., ET AL.: Scikit-learn: Machine learning in Python. *JMLR 12*, Oct (2011), 2825–2830. 2

[RFT16] RAUBER P., FALCÃO A., TELEA A.: Visualizing time-dependent data using dynamic t-SNE. In *EuroVis short papers* (2016), pp. 73–77. 2, 3

[RKHH09] ROHKOHL C., KECK B., HOFMANN H., HORNEGGER J.: RabbitctâĂŤan open platform for benchmarking 3d cone-beam reconstruction algorithms a. *Medical Physics 36*, 9Part1 (2009), 3940–3944. 2

[RLLY08] ROSS D. A., LIM J., LIN R.-S., YANG M.-H.: Incremental learning for robust visual tracking. *IJCV 77*, 1-3 (2008), 125–141. 3

[RS00] ROWEIS S. T., SAUL L. K.: Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 5500 (2000), 2323–2326. 3

[SSM98] SCHÖLKOPF B., SMOLA A. J., MÜLLER K.-R.: Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation 10*, 5 (1998), 1299–1319. 3

[SVM14] SORZANO C., VARGAS J., MONTANO A.: A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877 [stat.ML]. 2

[TB99] TIPPING M. E., BISHOP C. M.: Probabilistic principal component analysis. *Journal of the Royal Statistical Society 61*, 3 (1999), 611–622. 3

[TDL00] TENENBAUM J. B., DE SILVA V., LANGFORD J. C.: A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323. 3

[the] Theano library. deeplearning.net/software/theano. 4

[TLZM16] TANG J., LIU J., ZHANG M., MEI Q.: Visualizing large-scale and high-dimensional data. In *Proc. WWW* (2016), pp. 287–297. 1, 2

[Tor58] TORGERSON W.: *Theory and methods of scaling*. Wiley, 1958. 2, 3

[TR02] TEH Y. W., ROWEIS S. T.: Automatic alignment of hidden representations. In *Proc. NIPS* (2002), pp. 841–848. 3

[VdMPvdH07] VAN DER MAATEN L., POSTMA E., VAN DEN HERIK H.: Matlab toolbox for dimensionality reduction. *Maastricht Univ.* (2007). 2

[VGdS*20] VERNIER E., GARCIA R., DA SILVA I., COMBA J., TELEA A.: Quantitative evaluation of time-dependent multidimensional projection techniques, 2020. arXiv:2002.07481 [cs.LG]. 2, 4, 5, 6

[vH08] VAN DER MAATEN L., HINTON G. E.: Visualizing data using t-sne. *JMLR 9* (2008), 2579–2605. 2, 3, 5

[vis] Vispipeline. vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex. 2

[vP09] VAN DER MAATEN L., POSTMA E.: *Dimensionality Reduction: A Comparative Review*. Tech. rep., Tilburg University, Netherlands, 2009. Tech. report TiCC TR 2009-005. 2, 4

[WBS06] WEINBERGER K., BLITZER J., SAUL L.: Distance metric learning for large margin nearest neighbor classification. In *Proc. NIPS* (2006), pp. 1473–1480. 3

[WCV11] WALT S., COLBERT S. C., VAROQUAUX G.: The numpy array: a structure for efficient numerical computation. *Comp Sci Eng 13*, 2 (2011), 22–30. 5

[Wic14] WICKHAM H.: Tidy data. *Journal of Statistical Software 59* (2014). 5

[WPS05] WEINBERGER K. Q., PACKER B., SAUL L. K.: Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *AISTATS* (2005). 3

[ZHT06] ZOU H., HASTIE T., TIBSHIRANI R.: Sparse principal component analysis. *J Comput Graph Stat 15*, 2 (2006), 265–286. 3

[ZW07] ZHANG Z., WANG J.: MLLE: Modified locally linear embedding using multiple weights. In *Proc. NIPS* (2007), pp. 1593–1600. 3

[ZYZG07] ZHANG T., YANG J., ZHAO D., GE X.: Linear local tangent space alignment and application to face recognition. *Neurocomputing 70*, 7-9 (2007), 1547–1553. 3

[ZZ04] ZHANG Z., ZHA H.: Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM journal on scientific computing 26*, 1 (2004), 313–338. 3