"Software development for interactive visualization of high-dimensional data"

Lambert, Pierre

ABSTRACT

Dimensionality reduction is the process of mapping high dimensional data to a lower dimensional space, such that some properties are preserved. The process of dimensionality reduction requires the scientist to take decisions and make compromises, as part of the information is typically lost in the process. This paper presents a software environment for dimensionality reduction. The program offers an extensive array of tools for quality assessment, as well as features aiming to develop the user's intuition and help the development of new dimensionality reduction algorithms. This work also introduces a novel approach to multidimensional scaling (the process of preserving the pairwise distances between the original space and lower dimensional space), which enables the user to make embeddings in O(N) time while consuming O(N) memory.

CITE THIS VERSION

Lambert, Pierre. *Software development for interactive visualization of high-dimensional data.* Ecole polytechnique de Louvain, Université catholique de Louvain, 2021. Prom. : Lee, John Aldo ; Verleysen, Michel. http://hdl.handle.net/2078.1/thesis:30698

École polytechnique de Louvain

Software development for interactive visualization of high-dimensional data

Author: **Pierre Lambert**
Supervisors: **John Lee, Michel Verleysen**
Readers: **Cyril De Bodt, Benoit Frénay**

Academic year 2020–2021
Master [120] in Computer Science

Abstract

Dimensionality reduction is the process of mapping high dimensional data to a lower dimensional space, such that some properties are preserved. The process of dimensionality reduction requires the scientist to take decisions and make compromises, as part of the information is typically lost in the process.

This paper presents a software environment for dimensionality reduction. The program offers an extensive array of tools for quality assessment, as well as features aiming to develop the user's intuition and help the development of new dimensionality reduction algorithms. This software is also the object of a paper for the *Journal of Machine Learning Research*, which is in the process of being written.

This work also introduces a novel approach to multidimensional scaling (the process of preserving the pairwise distances between the original space and lower dimensional space), which enables the user to make embeddings in $\mathcal{O}(N)$ time while consuming $\mathcal{O}(N)$ memory.

Some knowledge in the behaviour of high dimensional data and in dimensionality reduction is needed to understand the purpose of the different features of the software. For this reason, this paper is divided into two parts: part one poses a context and some theoretical background, part two is dedicated to explaining the various features of the proposed software. Some recurring words and phrases are abbreviated in this paper, a list of abbreviations is available in *Annex A: Recurring abbreviations.*

Contents

Part I

Aspects of dimensionality reduction for visualization

Chapter 1

Context and problem statement

Curiosity is one of the defining characteristics of mankind : driven by our desire for control, we strive to understand the world in which we live. From the first written alphabet to the latest DNA sequencing machine, every new discovery is a step forward in our quest for knowledge. Theoretical models of the world paired with engineering offer us an increasing capacity to extract and store information from the environment. Lately, this increase in capabilities accelerated greatly : in the span of a couple of generations, we passed from observing a blurry red spot in a telescope to analysing the composition of the surface of mars by sending robots equipped with spectrometers, or from observing a cell in a microscope to reading its gene expression levels for thousands of genes at once.
A challenge arises from such a surge in the volume data available to us: our biology cannot keep up. The human brain is the result of a slow process of evolution in a 3-dimensional environment, therefore it excels at modelling 2- or 3-dimensional objects but cannot comprehend higher-dimensional data.

Data visualization acknowledges this biological limitation and attempts to present data in a way that humans can understand. Up until the digital era, visualization was limited to two dimensional drawings with superimposed information in the form of colors or numbers, or to observing the distribution of variables independently, for instance with adjacent histograms. Observing variables independently can reveal some patterns in the data, but complex structures can remain hidden, and looking at each variable independently is impractical when the dimensionality is high. Recently, by using the power of computers in conjunction with applied statistics, a powerful paradigm gained popularity for data visualization: **dimensionality reduction (DR)**.
Dimensionality reduction is the process of finding a faithful **lower-dimensional (LD)** representation of **higher-dimensional (HD)** data. In the context of visualization, DR typically has a target dimension of three or two. This work focuses

on two dimensional representations as they are the easiest to interact with.

Section one gives a broad description of some key aspects of DR. Section two builds on these notions to formulate a motive for this work. Section three presents a quick review of the existing works. Finally, section four enumerates the contributions brought by this work to the domain of dimensionality reduction for visualization.

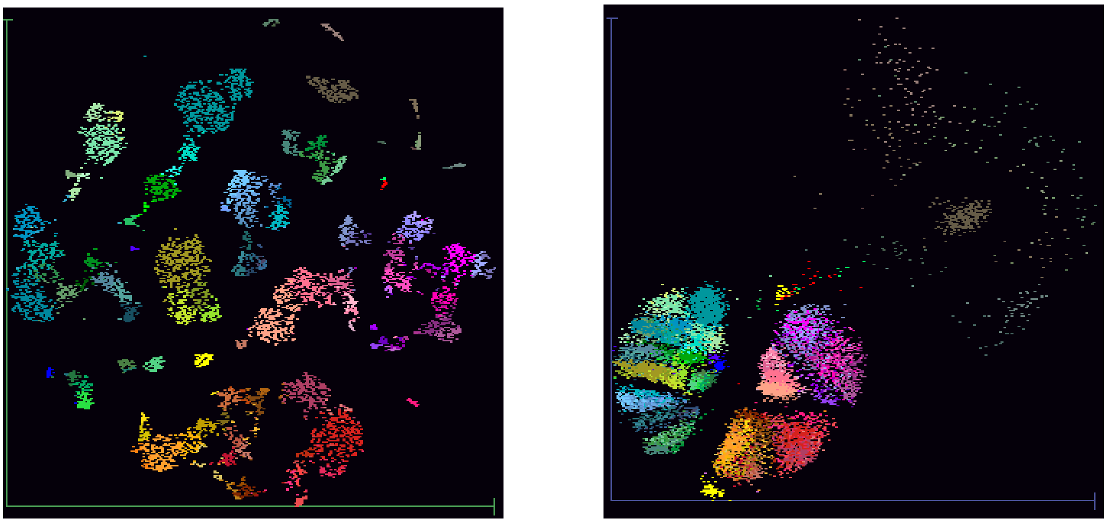
1.1 Introductory notions

In the beginning of his book *Envisioning Information*, Edward Tufte writes [1] : "*The world is complex, dynamic, multidimensional; the paper is static, flat. How are we to represent the rich visual world of experience and measurement on mere flatland?*". This summarises well the difficulties that arise when trying to fit high dimensional data on a less expressive, two-dimensional space. This section introduces some key notions of DR in order to bring intuition on the principal difficulties and choices faced by the data scientist. In this section, formal definitions and mathematical formulations are purposefully avoided when possible, as the main objective here is bring a broad intuition on the domain without going into too much details.

Dimensionality reduction can take many forms; one way to reduce the dimensionality of a dataset is to simply discard some of the variables that are deemed less useful for the task at hand and only keep the most relevant variables. There are many ways to choose which variables to keep, some supervised, for instance by taking the variables most correlated to the target, and some unsupervised, for instance by keeping the variables having a variance above a certain threshold. This process is called **feature selection**.
Another way to do DR is by **feature extraction**, in this process the HD data is projected into a lower dimensional space. This LD space can be understood as an intangible volume detached from the observable world, where the initial information is concentrated as best as possible. From now on, this paper will concentrate on feature extraction, as the ability to combine multiple HD features into few LD variables makes it more popular in the context of visualisation. In practice however, feature selection often precedes feature extraction, it can be considered as an initial cleanup enabling the feature extraction process to concentrate on the most important aspects of the data.

A paramount notion when doing DR is the **notion of scale**. The data typically presents some large structures in which are located smaller-scale structures. Capturing the organisation of the data on multiple scales is one of the major difficulties of DR, as most algorithms need to discard the information from one scale in order to capture the information from another.

The notion of scale is illustrated in Figure 1.1, where a dataset of gene expression levels for various rat cells is embedded in a 2-dimensional space using two algorithms focusing on different scales.
The first algorithm, *t*-SNE (for t-distributed stochastic neighbor embedding) [2], excels at preserving small-scale structures. We can see that the various clusters are well-defined and the relationships of the points within the clusters are easy to read. In Figure 1.1-b the same dataset is subject to another DR method, MDS (for multidimensional scaling) [3]. This method preserves large-scale structures better than smaller scale ones. Here, we see that the relationships within the clusters are not as easy to read, but a new scope of relations appears: three parts can be easily distinguished. These three clusters are in adequation with the underlying domain knowledge, as the warm colours correspond to inhibitory neurons, the colder colours to excitatory neurons, and the brown colours correspond to non-neuron cells.



(a) *t*-SNE : focusing on small-scale structures

(b) MDS : focusing on large-scale structures

Figure 1.1: Embeddings of single-cell transcriptomics. In warm colours: inhibitory neurons, in cold colours: excitatory neurons, in brown: non-neuron cells

Dimensionality reduction is the process of finding a faithful LD representation of the data. Therefore when designing a DR algorithm, one must first define **what**

properties should be preserved in order to consider a projection as a faithful representation of the original data.

The most straightforward definition of faithfulness for DR might be the preservation of distances. In this paradigm, the LD embedding is constructed in order to have pairwise distances that are as close as possible to the pairwise distances in the HD space. When preserving distances, the HD-distances are not necessarily Euclidean, but they are generally mapped into Euclidean LD distances.

At first glace, preserving the distances is an attractive approach, as a perfect preservation would perfectly capture the multi-scale similarities and dissimilarities between the various structures laying in the HD space. However, as shown in [4], an interesting but unfortunate phenomenon arises when dealing with high-dimensional data: as the dimensionality grows, the pairwise Euclidean distances between points tend to become very similar. This phenomenon is sometimes called **the concentration of norms**.

The problem brought by the concentration of norms in the context of DR is that the two-dimensional space lacks the necessary degrees of freedom to place points such that they are all at a similar distance. However, the distance preserving paradigm for DR should not be discarded : in practice, many datasets have a HD distribution of distances that has enough variance to discriminate between the different large-scale structures of the data. This paper will show that using a distance preserving algorithm in conjunction to other paradigms is a powerful way to understand both the global and the local structure of the data.

In order to counter the effects of the concentration of norms, the faithfulness of an embedding can be defined by using pairwise similarities. Distances need to follow four constraints: they have to be positive, they have to be symmetric, a distance of zero means the objects are identical, and the straight line should be the path of smallest distance between two points. Similarities do not need to follow these constraints, this provides the liberty to adapt to the local conditions around one of the two points of interest and overcome the effects of the concentration of norms.

For instance, some algorithms of a family called neighbor embeddings define HD similarities from the i^{th} observation to the j^{th} observation in a dataset of size N as:

$$\sigma_{ij} = \frac{\exp(-\pi_i \delta_{ij}^2/2)}{\sum_{k \in \mathcal{I} \setminus \{i\}} \exp(-\pi_i \delta_{ik}^2/2)}; \sigma_{ii} = 0.$$

with δ_{ij} a distance between the points indexed by i and j, $\mathcal{I} = \{1, ..., N\}$, and π_i a parameter tuned to the local properties of the HD space around the i^{th} point. The parameter π_i is tuned to bring output values that are discernible for observations close to point number i, but close to zero for other points, even if all the distances

are very similar.
The methods based on preserving non-linear similarities tend to bring good small-scale preservation but at the cost of distortions regarding the larger-scale structures.

Another element that should be pointed out is the notion of **manifold**. The idea behind manifolds is that the HD observations of dimension M can actually be laying in some M'-dimensional object, with $M' < M$. For instance, a 3-dimensional crumpled piece of paper could be unrolled into a 2-dimensional plane. Some DR algorithms build directly on this idea in order to unroll the manifold into a LD-space.

1.2 Problem statement

The previous subsection introduced the intuition that the process of DR is full of choices and compromises : on what scale should the LD representation be faithful to the HD structures, and to what extent can we allow ourselves to distort the structures from the other scales? If attempting to preserve distances, what type of distances should be used on the HD-side? If using similarities, how to define them? Should we consider the notion of manifold when looking at the HD data? Should some features be discarded before attempting the projection? As the HD space is not comprehensible by humans, on what basis can an embedding be considered a faithful representation of the HD data? Condensing these interrogations into a broader statement, one could ask:

Given the high number of decisions and compromises that are involved in the process of DR, how can the data-scientist know what the best course of action is for a given dataset and a given objective? Once an embedding is produced, how to know if it indeed corresponds to the expectations? Is there a way to be sure that an important aspect of the HD relationships hasn't been lost in translation?

Two key elements are necessary to answer these questions. The DR practitioner should:

1. Understand and master the range of available methods for DR and develop some intuition in the domain.

2. Be able to objectively evaluate the strengths and weaknesses of the produced embeddings.

One way to both develop one's intuition in DR and offer the possibility to assess the quality of the embeddings is by the use of a software environment. Ideally, such a software should satisfy the following set of properties:

- The software should offer high interactivity between the user, the data, and the projections. The user should be able to easily jump from one dataset to the other and to quickly launch a new projection or re-launch a projection after modifying some parameters. This encourages the user to try different ideas and configurations, and therefore helps building intuition.
- The software should be able to show multiple embeddings side by side. It enables the user to visually compare projections and see different viewpoints on the dataset. Ideally, the user should be able to interact with the embeddings by selecting points and seeing where the corresponding points lay in the other scatterplots.
- The software should offer tools to assess the quality of the embeddings, ideally with multiple quality criteria. The quality can either be a global measure on an embedding, or a local measure where zones of different quality are highlighted. Ideally, the software should also provide a tool to compare the strengths and weaknesses of multiple embeddings.
- Importantly, the user should have the possibility to customize the software, by adding new datasets, new DR algorithms, or new quality criteria.

1.3 Related work

Most DR algorithms have publicly available implementations in the Python and R languages, either though well-known machine-learning packages such as Python's *scikit-learn*, or through public repositories linked in research papers. However, softwares or frameworks that offer en environment for DR are less common.

One framework which was an inspiration for this work is R's *QVisVis* [5]. It focuses on local quality assessment (QA): the user can plot 2-dimensional or 3-dimensional scatterplots with a colour-code on the dots depending on the local quality. Often, some structures of the HD data are trickier to project than others, a colour-code based on local error is a way to identify these zones of difficulty.
Another feature brought by *QVisVis* is comparative local quality assessment, where the colouring depends on the point-wise quality in one embedding versus the quality on another embedding. This is both useful for the veteran DR-practitioner who wishes to get the most out of the embeddings and for the beginner, who can develop intuition by comparing how various algorithms compare in terms quality on specific structures of the data.
To evaluate the point-wise quality, *QVisVis* uses neighbour agreement between the HD and LD spaces. The neighbour agreement, parameterized by K, measures how

well the K-nearest neighbours of each point are preserved in the embedding. It is defined as:

$$AR_K = \frac{1}{Kn}\sum_{i=1}^{n} a_{iK}$$

With a_{iK} being an integer corresponding to the size of the intersection of the K-neighbours set on point i in HD and LD space. The user-defined parameter K provides the freedom to assess the quality on different scales: a large K captures global structures, and a small K captures local ones.
While *QVisVis* offers some useful tools to assess the absolute or relative local quality of embeddings, it's a programming framework. For this reason alone it can be considered as lackluster in terms of ease of interaction : the only input that the user can bring is through lines of code.

Another work, *VisCoDeR*[6] brings a software that follows the opposite approach from *QVisVis* : *VisCoDeR*'s first objective is to develop the intuition of the user by offering the possibility to view multiple embeddings in the same screen, and interact with them by selecting points and changing the hyperparameters. The interface is responsive and encourages the user to try different configurations of hyperparameters. A view is even dedicated to the exploration of a 2-dimensional hyperparameter space by plotting points corresponding to pre-computed embeddings. The weakness of *VisCoDeR* is that it doesn't bring as much tools as *QVisVis* in terms of QA and it is lackluster in terms of customisation, as new datasets and DR algorithms are not supported.

As a general overview, the publicly available software environments or code frameworks for DR often focus on one of the requirements listed in the previous section while neglecting the others. In the works that offer tools for QA, the QA is often restricted to neighbour preservation metrics; there are however other way to evaluate the quality of an embedding, for instance by looking at the pairwise distances or at the distribution of the labels.

1.4 Contributions to the domain

The contributions of the work conducted as part of this master's thesis are listed in this section. The main contribution, which attempts to answer directly to the stated problem, is a software environment.
This work also brings two other contributions to DR, in the form of a DR algorithm

and a study of an existing algorithm. These contributions can be considered as secondary contributions as they do not answer directly to the stated problem.

Software contribution

The first contribution to the domain of DR brought by this work is a software that still lacks a name. For this reason, it shall now be referred to as *the software*. *The software*'s objective is to answer to the four requirements which were introduced when stating the problem. *The software* brings a special focus on QA, as visual assessment alone cannot be completely trusted in the context of DR. Indeed, equally "good looking" scatterplots can preserve very different aspects of the HD structures, therefore any serious data exploration tool should incorporate some objective quality indicators.

For the purpose of clarity, the characteristics of *the software* are presented as a list.

- *The software* presents multiple embeddings in the same screen. The user can select points in an embedding and see where these points are located in the other embeddings.
- The user can choose to open the set of selected points as if they constitute a new dataset. New projections can then be applied to this set of points as a way to zoom in a specific part of the dataset.
- Two screens are dedicated to QA: one assesses the quality of embeddings in terms of absolutes, the other is designed for comparative quality assessment. These screens can show QA-related information of three different nature: the preservation of neighbourhoods, the preservation of distances, and the distribution of the labels.
- New datasets, DR algorithms, and local QA criteria can be added with minimal effort.
- When adding a new DR algorithm to *the software*, the user can choose to display the position of the points during the optimisation. This provides a visual assessment of the optimisation process.
- When adding a new dataset, any missing value is automatically imputed.

An article for *The Journal of Machine Learning Research* introducing *the software* is also in the process of being written.

Algorithmic contribution

This work also brings an algorithmic contribution to DR by introducing a fast and memory-efficient method for metric multidimensional scaling (MDS)[3]. Metric MDS is a statistical process that aims to find an embedding that preserves the distances, the problem with MDS is that the algorithms tend to have high time and memory complexities. By using intuition developed with *the software*, an algorithm allowing MDS in $\mathcal{O}(N)$ time and memory complexity has been envisioned and implemented.

A description of this algorithm will be given alongside other algorithms in the next chapter and a paper detailing this algorithm is attached in Annex B : "Stochastic quartet approach for fast multidimensional scaling". For the remainder of this paper, this fast MDS algorithm will be referred to as SQuaD-MDS, short for "Stochastic Quartet Descent-MDS".

A study of fast multiscale neighbor embedding

The work conducted as part of this master's thesis also brings a third contribution to the domain of DR, with a study of the fast multiscale neighbor embedding algorithm. This algorithm will be described alongside the other DR algorithms in the next chapter.

Fast multiscale neighbor embedding estimates the topology of the observations in the HD space by relying on multiple samples of decreasing size. The purpose of the study is to evaluate the algorithm's sensitivity to randomness, the paper is linked in Annex C : "Impact of data subsamplings in Fast Multi-Scale Neighbor Embedding". The conclusion of the study is that fast multiscale neighbor embedding is already surprisingly robust to the effects of randomness.

Chapter 2

Dimensionality reduction : algorithms and quality assessment

The first section of this chapter brings some theory on the different DR algorithms that come by default with *the software*.
Section two introduces three ways to assess the quality of LD projections.

2.1 Dimensionality reduction algorithms

The software comes with a set of DR algorithms, this section brings some elements of theory for each of these algorithms. As the purpose of this paper isn't to review the existing DR algorithms, most of the algorithms will only be subject to a very brief overview. Neighbour embeddings will be reviewed with more details, as this family of methods is at the center of attention in the DR community; SQuaD-MDS, the algorithmic contribution of this work, will also be explained in detail.

This section uses the following naming convention:
The set of N points in a HD space with M features is noted Ξ, the associated set of points in the LD space is called $\mathbf{X}$; the set of integers $\mathcal{I} = \{1, ..., N\}$ contains the indices of each observation. The HD and LD distances between the i^{th} and j^{th} points are δ_{ij} and d_{ij}, respectively, for $i \in \mathcal{I}$ and $j \in \mathcal{I} \setminus \{i\}$.

Principal component analysis

Principal component analysis, or PCA, is a linear transformation of the data that aims to find the orthogonal coordinate system that explains most of the variance. Calling S the covariance matrix of the dataset, the axis of greatest variance

corresponds to the eigenvector (sometimes called component) of S which has the highest eigenvalue. The second most important axis variance-wise corresponds to the eigenvector with the second largest eigenvalue, and so on. Once the desired number of components is found (in the context of this work, two), the centered dataset is projected onto them by matrix multiplication.
PCA is one of the most widely used DR algorithms, with applications reaching further than visualisation. On many datasets, a high percentage of the total variance can be captured in significantly less components than the original dimension, this makes PCA a good pre-processing step for many machine-learning tasks.
In the context of DR for visualisation, PCA tends to capture the global structure of the data, as most of the variance is used to distinguish between elements from distinct parts of the landscape. PCA usually produces embeddings faster than the other DR algorithms, it is therefore a good way to get a first overview of the data when opening a new dataset.

Single-scale neighbor embedding

Neighbor embeddings (NE) are a class of DR algorithms aiming to preserve the pairwise similarities between the observations in both spaces.
Stochastic neighbor embedding (SNE) [7] respectively defines the HD and LD pairwise similarities as:

$$\sigma_{ij} = \frac{\exp(-\pi_i \delta_{ij}^2/2)}{\sum_{k\in\mathcal{I}\setminus\{i\}} \exp(-\pi_i \delta_{ik}^2/2)}, \;\; s_{ij} = \frac{\exp(-d_{ij}^2/2)}{\sum_{k\in\mathcal{I}\setminus\{i\}} \exp(-d_{ik}^2/2)}, \;\; \sigma_{ii} = s_{ii} = 0 \;\; .$$

Precision π_i is adapted to the local density in order to tune the HD similarities to a target scale. The scale is provided by the user in the form of a perplexity K for the distribution $[\sigma_{ij}; j \in \mathcal{I} \setminus \{i\}]$ such that $\log K = -\sum_{j\in\mathcal{I}\setminus\{i\}} \sigma_{ij} \log \sigma_{ij}$. By interpreting these similarities as neighborhood probabilities around each point, SNE produces a LD embedding that minimises the sum of Kullback-Leibler (KL) divergences $C_{SNE} = \sum_{i\in\mathcal{I}, j\in\mathcal{I}\setminus\{i\}} \sigma_{ij} \log(\sigma_{ij}/s_{ij})$.

SNE is subject to an unfortunate phenomenon called the 'crowding problem'. This problem is best understood by using simple scenario: in a 3-dimensional space, there can be up to 4 equally distant points, while in a 2-dimensional space there can only be 3. This means that, when projecting the data from a 3-dimensional space to a 2-dimensional space, one of the four points will have to be placed too far from the others, this phenomenon is exacerbated when the dimensionality grows. The resulting gradients attempt to bring the points closer by producing a slight attractive force, the combination of all these slight attractive forces in the embedding tends to crush the points towards the center of the map, hence the name 'crowding problem'.

To counter the effects of crowding, SNE has an extension called t-SNE [2]. t-SNE symmetrizes the similarities and considers a Student t function with one degree of freedom in the LD space. Using this distribution rather than a Gaussian distribution brings slight repulsive forces between the points, countering the crowding effect. The HD similarities τ_{ij} and LD similarities t_{ij} are defined as

$$\tau_{ij} = (\sigma_{ij} + \sigma_{ji})/(2N), \;\; t_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \in \mathcal{I}, l \in \mathcal{I} \setminus \{k\}} (1 + d_{kl}^2)^{-1}}, \;\; \tau_{ii} = t_{ii} = 0 \;\; .$$

In its default version, t-SNE has a time complexity of $\mathcal{O}(N^2)$. The typical smallness of the perplexity K with respect to N enables single-scale t-SNE to consider that sufficiently distant points have a HD similarity of zero. Sparse HD similarities can be computed by finding the sets of nearest neighbors for each point using vantage-point trees [8]. This reduces the computation time of HD similarities to $\mathcal{O}(KN \log N)$.
On the LD side, a Barnes-Hut (BH) algorithm [9, 10] approximates efficiently the similarities by relying on tree structures to compound far-away points, dispensing with computing all the pairwise interactions. The BH method reduces the time complexity of computing LD similarities to $\mathcal{O}(N \log N)$.

t-SNE is very popular in the domain of DR for visualization, as it tends to produce well-defined and easy to read clusters, while achieving a good preservation of similarities. As stated previously, using nonlinear similarities that adapt to the local properties of the HD space is an effective way to counter the effect of the concentration of norms; t-SNE is therefore equipped to discriminate between structures in very high-dimensional data.
One important drawback of t-SNE is its propensity to distort large-scale structures. The distortions are caused by the tendency of t-SNE to consider sufficiently distant points as equally distant, as their similarities shrink to values very close to zero. This property is often overlooked by the users that don't come from a DR background, which can lead to erroneous conclusions.

Multi-scale neighbor embedding

The multi-scale SNE method [11] combines multiple SNE similarities which are computed with exponentially increasing perplexities. Single-scale similarities are indexed by a scale counter h for $i \in \mathcal{I}$ and $j \in \mathcal{I} \setminus \{i\}$:

$$\sigma_{ijh} = \frac{\exp(-\pi_{ih}\delta_{ij}^2/2)}{\sum_{k \in \mathcal{I} \setminus \{i\}} \exp(-\pi_{ih}\delta_{ik}^2/2)}, \; s_{ijh} = \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k \in \mathcal{I} \setminus \{i\}} \exp(-p_{ih}d_{ik}^2/2)}, \; \sigma_{iih} = s_{iih} = 0 \;\; .$$

HD precisions π_{ih} are set using perplexities growing as $K_h = 2^{h-1}K_1$, with a small base perplexity $K_1 = 2$ and $1 \leq h \leq H = \lfloor \log_2(N/K_1) \rceil$, where $\lfloor \cdot \rceil$ denotes rounding. LD precisions $p_{ih} = p_h = K_h^{-2/P}$ follow the exponential growth of the HD precisions. Multi-scale similarities average single-scale ones as

$$\sigma_{ij} = H^{-1} \sum\nolimits_{h=1}^{H} \sigma_{ijh}, \;\; s_{ij} = H^{-1} \sum\nolimits_{h=1}^{H} s_{ijh} \;\; .$$

Multi-scale neighbor embedding counters the main problem of t-SNE, which is its tendency to only preserve small-scale relationships. In terms of preservation of neighbors across all scales, this algorithm is generally the best option. However, multi-scale t-SNE has a flaw: it has a time complexity of $\mathcal{O}(N^2 \log N)$, making it difficult to use on large datasets.

Fast multi-scale neighbor embedding

Fast multi-scale neighbor embedding (f-ms-NE)[12] is the accelerated version of multi-scale neighbor embedding (ms-NE). In ms-NE, the largest scale used to compute multi-scale similarities has a perplexity K_H in the $\mathcal{O}(N)$ range. At such a scale, the sparse HD similarity approximation of single-scale t-SNE would require neighbor sets of $\mathcal{O}(N)$ size, defeating the purpose of accelerating the computations. For this reason, reducing the time complexity of ms-NE requires a different approach on the HD side. In [12], the authors tackle this problem by defining small multi-scale neighbor sets $\widetilde{\mathcal{I}}_i$ for $i \in \mathcal{I}$.

To compute $\widetilde{\mathcal{I}}_i$, the authors define a hierarchy of subsampled HD data sets $\{\Xi_h\}_{h=1}^{H}$, where Ξ_h is a random sample of Ξ with $\lfloor 2^{1-h}N \rceil$ elements drawn without replacement. A vantage-point tree is created on Ξ_h for each scale $h \in \{1, ..., H\}$; the trees are generated in $\mathcal{O}(N \log N)$ time. For each scale, the corresponding tree is used to compute the neighborhood $\widetilde{\mathcal{I}}_{ih}$ within the sample Ξ_h for $i \in \{1, ..., N\}$; this takes a total of $\mathcal{O}(N \log^2 N)$ time. By having samples $\{\Xi_h\}_{h=1}^{H}$ that decrease in size when the scale grows, the accounted neighbors $\widetilde{\mathcal{I}}_{ih}$ are more and more dispersed in the data cloud, hence capturing larger scale properties. The multi-scale neighbor sets are then defined as $\widetilde{\mathcal{I}}_i := \cup_{h=1}^{H} \widetilde{\mathcal{I}}_{ih}$.

Like the non-accelerated version, f-ms-NE uses multi-scale similarities averaged over sparse single-scale similarities,

$$\widetilde{\sigma}_{ij} = H^{-1} \sum_{h=1}^{H} \widetilde{\sigma}_{ijh}, \quad \text{with} \quad \widetilde{\sigma}_{ijh} = \begin{cases} \frac{\exp(-\widetilde{\pi}_{ih}\delta_{ij}^2/2)}{\sum_{k \in \widetilde{\mathcal{I}}_i} \exp(-\widetilde{\pi}_{ih}\delta_{ik}^2/2)} & \text{if } \; j \in \widetilde{\mathcal{I}}_i \\ 0 \;\; \text{otherwise} \end{cases} \quad .$$

Precision $\widetilde{\pi}_{ih}$ is fixed such that

$$K_1 = - \sum_{j \in \widetilde{\mathcal{I}}_{ih}} \widetilde{\sigma}^{\pi}_{ijh} \log \widetilde{\sigma}^{\pi}_{ijh} \text{ where } \widetilde{\sigma}^{\pi}_{ijh} = \begin{cases} \frac{\exp(-\widetilde{\pi}_{ih}\delta^2_{ij}/2)}{\sum_{k \in \widetilde{\mathcal{I}}_{ih}} \exp(-\widetilde{\pi}_{ih}\delta^2_{ik}/2)} & \text{if } j \in \widetilde{\mathcal{I}}_{ih} \\ 0 \text{ otherwise} \end{cases} .$$

In the $t-$distributed version, the authors use symmetrised HD similarities $\widetilde{\tau}_{ij} = (\widetilde{\sigma}_{ij} + \widetilde{\sigma}_{ji})/(2N)$ and optimise the cost function $\widetilde{C}_t = - \sum_{i \in \mathcal{I}, j \in \mathcal{I} \setminus \{i\}} \widetilde{\tau}_{ij} \log t_{ij}$.

The use of random samples to approximate the global structure might bring interrogations on the robustness of the algorithm with regards to the effects of randomness.
The study linked in Annex C : "Impact of data subsamplings in Fast Multi-Scale Neighbor Embedding", shows that the benefits of using multiple resamplings for each scale instead of using one sample are limited. The study concludes from this observation that the algorithm is already surprisingly robust to the effects of randomness.

Class-aware t-SNE

CAt-SNE, short for class-aware t-SNE[13], is a supervised version of t-SNE designed for classification datasets. To account for class labels, CAt-SNE defines a condition on the weighted proportion t_i of neighbours sharing the same class as point i:

$$t_i = \sum_{j \in \mathcal{I} \setminus \{i\} | c_j = c_i} \sigma_{ij} > \theta \ .$$

With c_i the class assigned to point i, and θ a user-defined parameter which takes values in the $[0.5, 1[$ range to ensure the majority of class c_i. Instead of using a user-defined perplexity to tune the precision π_i, the precision is minimized under the constraint t_i. When no precision can fulfill the condition t_i, then π_i is set to maximise t_i.

Fast Interpolation-based t-SNE

Abbreviated FIt-SNE[14], this recent extension to t-SNE brings a great increase in speed. The full algorithm will not be described here, the general principle behind the acceleration is that the pairwise interactions are interpolated from interactions between the points and a small number p of interpolation nodes defined in LD. By using a constant number of nodes to mediate between the points, the number of computations at each iteration is reduced to $\mathcal{O}(N)$.

FIt-SNE achieves results that are as accurate as the BH t-SNE algorithm but much faster, making it applicable for datasets of hundreds of thousands of observations. It is often used to visualise biological data, as the field usually has very large datasets.

Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection also called UMAP[15], is an algorithm that represents the HD topology with a fuzzy graph using a mathematical machinery that is outside of the scope of this paper. Similarly to the perplexity of single-scale neighbor embeddings, a user-defined parameter called 'n neighbors' influences the way the graph is connected, greater values bring stronger connections to more observations than lower values. UMAP aims to build a LD embedding that follows the topology of the graph built on the HD data by using a force-based optimisation.
UMAP tends to be faster than t-SNE and the authors claim that the global structure is generally better preserved than with t-SNE. Nuance can be brought to this claim by comparing t-SNE and UMAP embeddings on multiple datasets using *the software* and its quality assessment tools.

Laplacian eigenmaps

Laplacian eigenmaps [16] project the observations into a LD space by preserving the local properties of the HD manifold. This method minimises the distance between the LD points that are connected on a K-neighbours graph computed in the HD space with a user-defined parameter K, in conjunction with another term that prevents all the LD points to collapse into similar values. Laplacian eigenmaps build on spectral graph theory to find the embedding by eigenvalue decomposition of the graph Laplacian.
In terms of visualisation, embeddings produced by Laplacian eigenmaps are usually less easy to read than embeddings from other manifold learning algorithms such as UMAP or isomap.

Local linear embedding

Abbreviated LLE, Local linear embedding [17] focuses on the local properties of the manifold by representing each point as a weighted linear combination of nearby points. First, the weights are set by minimising the error between the real HD observations x_i and their approximation using their set of K neighbours $\mathcal{K}_i$:

$$\min \sum_{i \in \mathcal{I}} \left\| x_i - \sum_{j \in \mathcal{K}_i} w_{ij} x_j \right\| \; .$$

Having fixed the weights, LLE finds the LD representations y_i that minimises:

$$\min \sum_{i \in \mathcal{I}} \left\| y_i - \sum_{j \in \mathcal{K}_i} w_{ij} y_j \right\| .$$

As for Laplacian eigenmaps, LLE produces embeddings that are often harder to read than those produced by other manifold learning methods. This algorithm is also particularly sensitive to the hyperparameter K, which often requires trial and error to tune.

Autoencoders

Autoencoders [18] are an elegant way to use deep-learning for dimensionality reduction. The architecture of autoencoders can be decomposed into two successive parts: the information passes through an encoder and then through a decoder. The last layer of the encoder has a number of nodes corresponding to the desired embedding dimension, each node in this bottleneck layer represents one dimension of the LD space, which is also called 'latent space' in the literature. The cost function optimised by autoencoders is a measurement of reconstruction error, for instance the squared error between the input and the decoded vector.

Metric multidimensional scaling

Metric MDS, short for metric multidimensional scaling is the statistical process of finding a LD representation of HD data such that the pairwise distances are preserved. The metric MDS minimises a cost function which is of the form:

$$\mathcal{C} = \sqrt{\sum_i \sum_{j \neq i} w_{ij} (d_{ij} - \delta_{ij})^2}$$

The weights w_{ij} are 1 by default, they can be modified to orient the optimisation towards a certain desired property. One common way to optimise this cost function is by using the SMACOF algorithm [19]. The SMACOF algorithm involves iterations of $\mathcal{O}(N^2)$ time complexity and uses the full HD distance matrix, consuming $\mathcal{O}(N^2)$ memory.

The poor scaling of MDS with regards to the number of observations restricts its use to datasets of moderate size. One common way to circumvent the problem is to apply MDS to a subset of the dataset, as in [20]. In visualization, MDS is generally a good way to capture the global landscape of the data as it tends to produce richer patterns than PCA.

Isometric mapping

Also called isomap, this algorithm can be seen as a version of MDS that looks at geodesic distances computed on a nearest-neighbours graph. The user defines the number of neighbours K that should be considered when building the graph.

SQuaD-MDS

The name SQuaD-MDS is short for "Stochastic QUArtet Descent - MDS", the algorithmic contribution of this work. This algorithm produces competitive solutions for MDS in $\mathcal{O}(N)$ time and consumes $\mathcal{O}(N)$ memory, a significant improvement over the SMACOF optimisation. Details and empirical quality assessments can be found in annex B : "Stochastic quartet approach for fast multidimensional scaling".

The main idea behind SQuaD-MDS is that the optimisation process can be divided into many smaller and easier problems defined on quartets of points.
For a quartet of points with HD distance matrix δ and a LD Euclidean distance matrix d, SQuaD-MDS defines relative distances as:

$$d^{\mathrm{r}}_{ij} = d_{ij} \Big/ \Big(\sum\nolimits_{a=1}^{3} \ \sum\nolimits_{b=a+1}^{4} d_{ab} \Big) \quad \text{and} \quad \delta^{\mathrm{r}}_{ij} = \delta_{ij} \Big/ \Big(\sum\nolimits_{a=1}^{3} \ \sum\nolimits_{b=a+1}^{4} \delta_{ab} \Big) \quad .$$

A cost function for the preservation of relative distances within this quartet of points can be defined:

$$C_{\text{quartet}} = \sum\nolimits_{a=1}^{3} \ \sum\nolimits_{b=a+1}^{4} (\delta^{\mathrm{r}}_{ab} - d^{\mathrm{r}}_{ab})^2 \quad .$$

SQuaD-MDS optimises this function by gradient-descent; for each element of the sum, the gradients take the form:

$$\frac{\partial(\delta^{\mathrm{r}}_{ij} - d^{\mathrm{r}}_{ij})^2}{\partial x_q} \quad = \quad \frac{2}{S}(d^{\mathrm{r}}_{ij} - \delta^{\mathrm{r}}_{ij}) \Big(\frac{I_{qi}(x_q - x_j) + I_{qj}(x_q - x_i)}{d_{ij}} - d^{\mathrm{r}}_{ij} \sum_{b \in \{1,..,4\} \setminus q} \frac{x_q - x_b}{d_{qb}} \Big)$$

with I the 4x4 identity matrix and $S = \sum_{i<j} d_{ij}$.

SQuaD-MDS applies the cost function to the whole data set by iteratively and randomly splitting the observations in groups of 4, the gradients are then computed within these randomly defined quartets. Using quartets of points gives constant-time gradients and has a natural analogy to triangulation (1 point versus 3 others) when applied to a target dimension of 2. Quartets are also the reason why the full HD distance matrix isn't needed : only 6 HD distances are used at any instant.

Relative distances are used instead of raw distances for two reasons:
First, it's a way to remove the scale constraint which is generally present in distance-preserving algorithms. The scale constraint is deemed unnecessary as it is the relation between the points that is of interest in an projection, not their positions in terms of absolute values.
Second, using relative distances is a way to bring information on the whole geometry of the quartet into each distance, as all the six distances of a quartet are used to compute each relative distance. During the iterative optimisation, each point moves in order to minimise the cost function on the whole quartet instead of only looking a three distances.

One of the strengths of SQuaD-MDS is its usage of a straightforward gradient-descent optimisation, making it open for hybrid approaches by combining the distance-preserving gradients to gradients of other nature. *The software* offers the possibility to add weighted t-SNE gradients to SQuaD-MDS gradients. The hybrid approach produces embeddings that both retain the global structure of the HD data thanks to the distance preservation gradients, and the small-scale structures thanks to t-SNE's gradients.

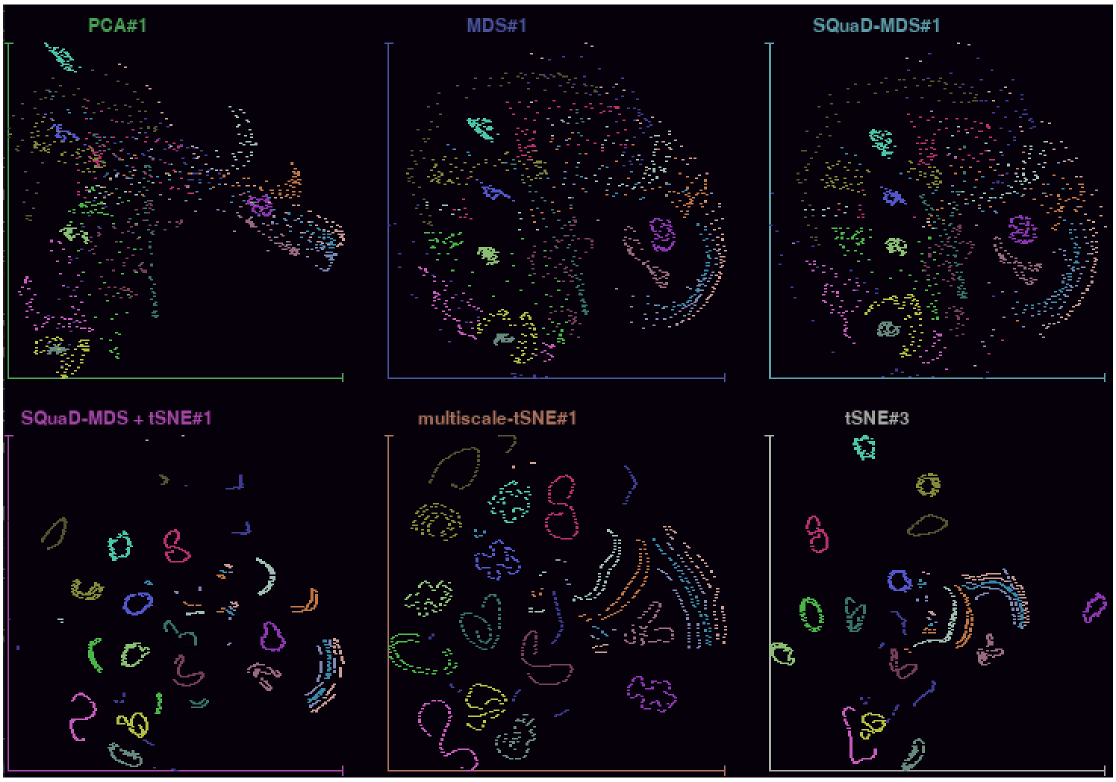


Figure 2.1: Embeddings of the Coil-20 dataset produced by PCA, SMACOF MDS, SQuaD-MDS, SQuadD-MDS+t-SNE, multiscale-t-SNE, and t-SNE

Figure 2.1 shows a visual comparison of embeddings produced by multiple DR algorithms on the Coil-20 dataset. This dataset consists of 32-by-32 images of objects rotating around one axis, this rotation ensures that each picture has at least two close neighbours. The scatterplot labelled "MDS" is achieved using the standard SMACOF algorithm.

The obvious similarity between the scatterplot "MDS" and the scatterplot "SQuad-MDS" shows that the stochastic quartet approach of SQuaD-MDS does indeed result in a global distance scaling of the whole dataset.

According to the distance preserving models, the t-SNE embedding is visually the one with the most large-scale distortions, for instance the pink cluster on the right of the embedding should not be so far out. However, the t-SNE embedding shows a global organisation that isn't completely random, this is mostly due to its PCA initialisation. The propensity of t-SNE to break out of the initial PCA configuration depends on its hyperparameters and on the dataset.

The embedding "SQuaD-MDS + tSNE" shows that adding t-SNE gradients gives sharper clusters and keeps most of general topology; for instance, the relative positions of the clusters of the right-hand side of the embedding are particularly well preserved according to the MDS scatterplots.

2.2 Quality assessment for dimensionality reduction

As the high-dimensional space is not directly comprehensible by the human mind, evaluating the quality of an embedding is a process that requires dedicated tools. This work brings tools to evaluate the quality of the embedding based on information of three different nature:

- The rank : how well the embedding preserves the order of neighbours between the HD and LD dimension. This work sometimes refers to it as "neighbour preservation".
- The distances : the strength of the (positive) linear relation between distances in HD and LD.
- The labels : comparing the distribution of the labels in the embedding with the distribution in HD.

Another possible way to consider the quality of an embedding would be by evaluating the ease and quality of reconstruction from LD back to the HD space. However, as it is not a property that is particularly sought-after in the field of visualisation, the ease of reconstruction is ignored in this work.

This section introduces ways to evaluate the quality of embeddings by looking at the ranks, the distances, and the labels.

2.2.1 Evaluation on the ranks

In order to evaluate the preservation of ranks between both spaces, *QVisVis*[5] uses neighbour agreement on a user-defined scale K. Let ν_i^K and ρ_i^K be the K-ary neighbourhoods for the i^{th} observation in HD and LR respectively, the K-ary neighbour agreement can be written as:

$$Q_{NX}(K) = \frac{1}{KN}\sum_{i=1}^{N} |\nu_i^K \cap \rho_i^K|$$

This definition of neighbour agreement brings a score between zero and one. Randomly generated LD coordinates (not to be confused with random projections, which produce embeddings by linear transformation of the HD data using

a randomly-generated matrix) have an expected value for $Q_{NX}(K)$ equal to : $\mathbb{E}[Q_{NX}(K)] = K/(N-1)$. This becomes problematic when K becomes a non-negligible portion of N, as the effects of randomness bloat the measurement.
To account for that, the authors of [21] introduce another agreement measure which takes into account the effects of randomness:

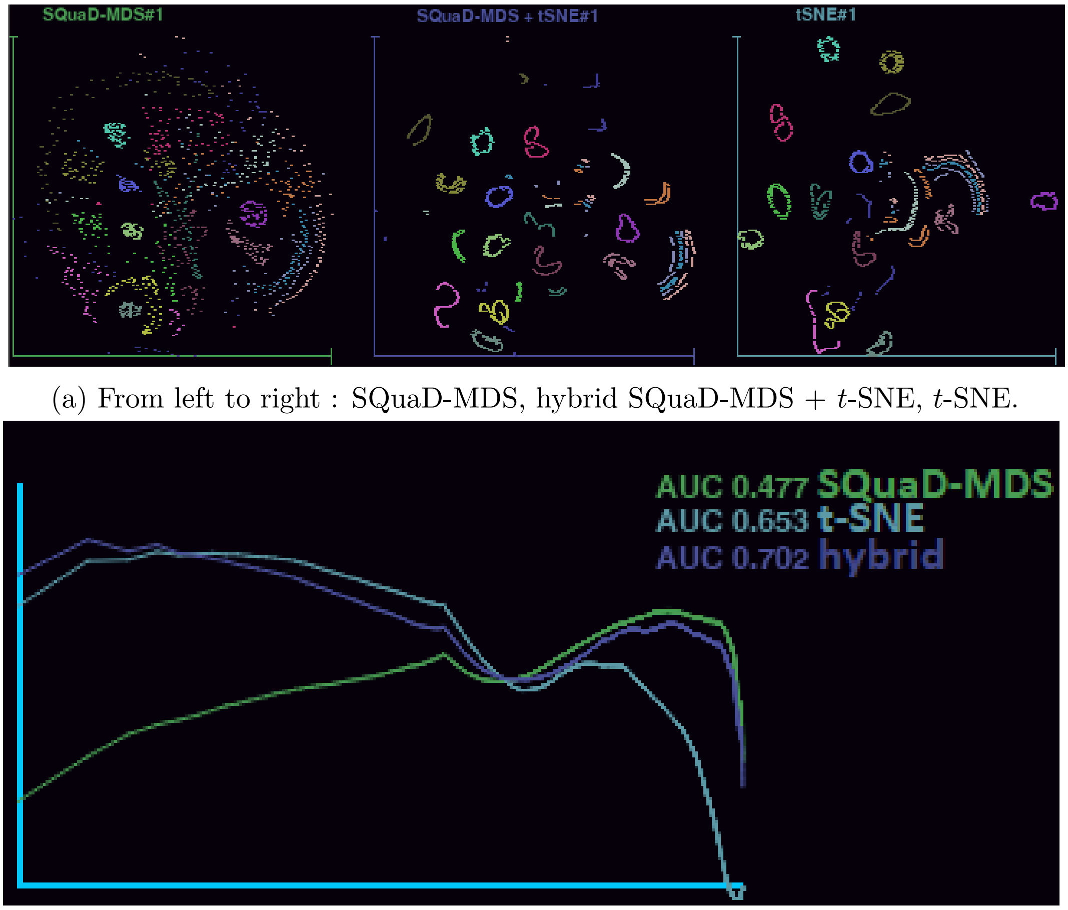
$$R_{NX}(K) = \frac{(N-1)Q_{NX}(K) - K}{N-1-K},$$

for $1 \leq K \leq N-2$. $R_{NX}(K)$ measures the improvement of the embedding over a randomly generated projection.

Asking the user to set K to desired values can be problematic, as it might encourage him or her to take shortcuts and only evaluate the quality on a handful of arbitrary values of K. For this reason, when assessing the global quality of an embedding, this work plots the whole $R_{NX}(K)$ curve, using a logarithmic axis for K. Using a logarithmic scale for K is a way to reflect the idea that an error in a small neighbourhood is more costly than in a large one. To get a global scalar score, [21] uses the area under curve (AUC) of $R_{\mathrm{NX}}(K)$ (also with a logarithmic scale).

$$AUC(R_{NX}(K)) = \frac{\sum_{K=1}^{N-2} R_{NX}(K)/K}{\sum_{K=1}^{N-2} 1/K}.$$

The AUC takes values in $[-1, 1]$, positive values indicate an improvement over randomly placed points.



(a) From left to right : SQuaD-MDS, hybrid SQuaD-MDS + *t*-SNE, *t*-SNE.

(b) The corresponding color-coded R_{NX}(K) curves and their AUC.

Figure 2.2: Embeddings of the Coil-20 dataset and their R_{NX}(K) curves.

Figure 2.2.a shows three embeddings of the Coil-20 dataset using SQuaD-MDS, the hybrid between SQuaD-MDS and *t*-SNE, and *t*-SNE. Figure 2.2.b shows the corresponding $R_{\mathrm{NX}}(K)$ curves.

According to the AUC of the curves, the SQuaD-MDS embedding is the worst in terms of neighbour preservation. When looking at its curve (green line), we see that this projection performs badly in terms of neighbour preservation on small values of K. This observation is in line with the expectation, as distance preserving algorithms tend to capture the large-scale structures better than the small-scale ones.

The hybrid approach brings the best AUC. When following its $R_{\mathrm{NX}}(K)$ line (in purple), we see that it takes the best of both worlds: the quality is similar to *t*-SNE on small values of K, but drops later, indicating a better preservation of the global structure.

The observant reader might notice a sharp angle in the curves at about three fifths of the K-axis, the value for K at this level is 71. The Coil-20 dataset has 72 pictures for each label, so the drop behind $K = 71$ indicates that, for some parts of the dataset, it is harder to preserve the neighbourhoods when they start to encompass points that are assigned another label.

2.2.2 Evaluation on the distances

In our 3-dimensional world, we expect the shortest path between two points to be the straight line, this is because what we could call the "density of space" appears to be homogeneous. This assumption isn't valid when looking at an embedding representing HD data : the LD space can represent some nonlinear transformation of the HD space, resulting in local dilations and compressions.

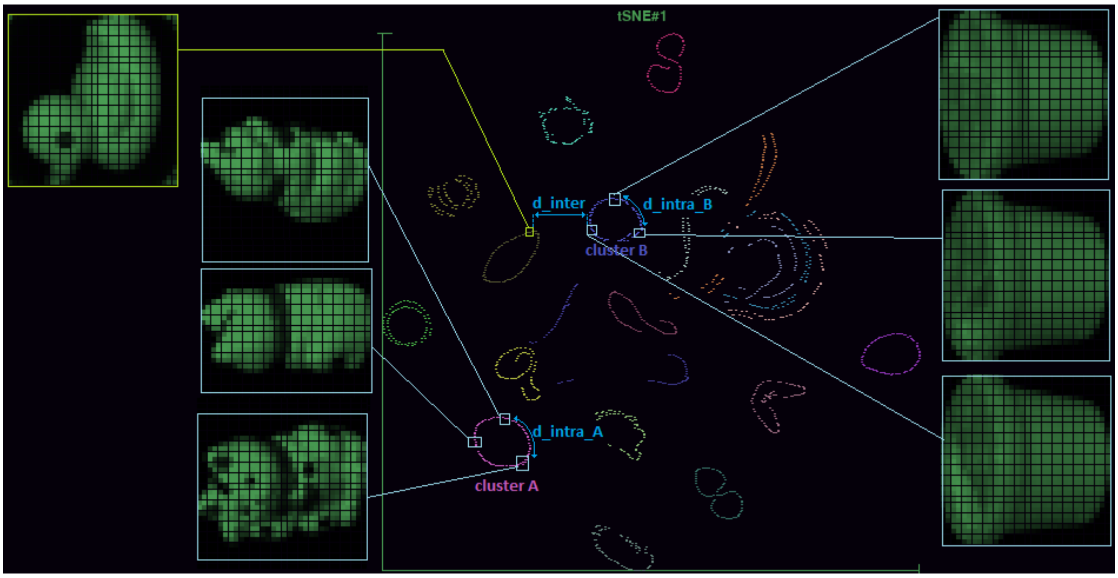


Figure 2.3: Visual illustration of the distance distortions in a t-SNE embedding of Coil-20.

Figure 2.3 shows a annotated t-SNE embedding which illustrates how unreliable the distances can get. The two rings called cluster A and B in the figure have roughly the same diameter, we see however that the images within cluster B barely change when traveling along the ring, which is not the case in cluster A even though the traveled distances are comparable.
This figure shows that the inter-cluster distances are also unreliable: 'd_inter' separates two very distinct images and 'd_intra_B' separates almost identical

pictures, yet both of these distances are roughly equal.

To evaluate the distortion of distances with a scalar, this work uses the Pearson correlation coefficient (PCC) between the distances in the HD space and those in the LD space.
The PCC takes values in $[-1, 1]$: values close to zero mean that there is very little linear relationship between the distances, and values close to one mean a strong correlation. Negative values are rarely encountered, they indicate a linear relationship with a negative coefficient. The embedding produced by t-SNE in Figure 2.3 has a PCC of 0.475, an MDS embedding on the same dataset can be expected to have a PCC above 0.830 .

The PCC gives a global overview on the distortion of distances, however if we were to ask ourselves what parts of the distribution of distances are receiving the most distortions, a simple scalar would be insufficient. One way to visually compare the distribution of distances is by using a Shepard diagram[22]. Shepard diagrams plot the sorted scaled LD distances on one axis, and the corresponding scaled HD distances on the other axis.
An embedding with a perfect preservation of distances would have a Shepard diagram with points along the diagonal, representing the line $f(x) = y$. Looking at the Shepard diagram of an embedding can show where most of the distortions occur : the further the points from the diagonal, the more distortion there is.

The Shepard diagram corresponding to the t-SNE embedding from Figure 2.3 is shown in Figure 2.4, using a random sample of 5000 distances. In this work, the Shepard diagrams use the Y-axis for HD distances and the X-axis for LD distances; the distances are first shifted to get values starting at zero, they are then scaled in order for the interval $[0, 1]$ to encompass five standard deviations. The shifted and scaled distances with a value greater than one in either the LD or the HD distributions are not plotted.
The bright yellow line corresponds to the mean value in 42 buckets taken on the sorted LD distances. This diagram shows that the first quarter of the LD distances are associated to a wide range of values on the HD side, whereas the last half of the LD distances represent HD distances with a smaller range of values. The wide range of values on the HD distances associated to the small LD distances is in line with the observations that were done in Figure 2.3, where small distances can correspond to very similar HD observations as well as very dissimilar ones.

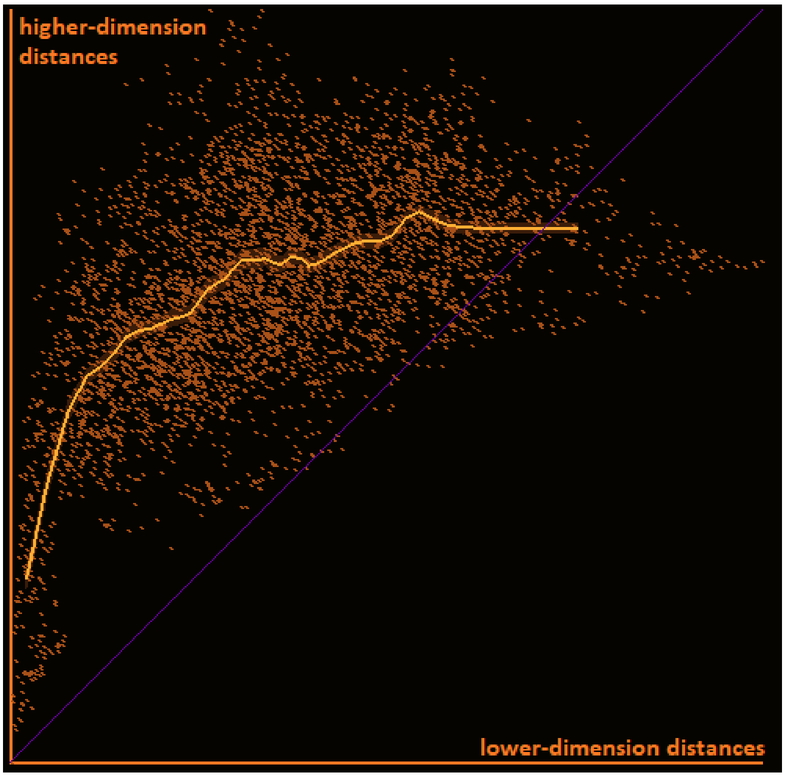


Figure 2.4: Visual illustration of the distance distortions in a t-SNE embedding of Coil-20.

2.2.3 Evaluation on the labels

Assessing the quality in terms of neighbour preservation or distance distortions is straightforward in the sense that the objective is easy to define: a good embedding is an embedding that brings the least distortions. When taking into account the labels however, the line separating a good embedding from a bad one is not as easy to trace.
On one side, one could favour an embedding where the classes are well separated. On the other side, one could prefer an embedding where the distribution of labels among the neighbours are as faithful as possible to the distribution of labels in HD, without aiming for an improvement in terms of KNN accuracy.

In the case of unsupervised DR algorithms, whether an embedding will bring a good class separation or not depends a lot on the relationship between the labels and the true structure of the HD data. Some datasets have labels that are only loosely related to the properties of the data, in that case, one cannot expect a good embedding to produce a good class separation. In other cases, the label is

an obvious consequence of the true structure of the data, in which case a good embedding would inadvertently bring a good class separation.
One dataset with labels that are loosely related to the true nature of the data is "winequality", where the variables represent various chemical and physical properties of different wines, and the label is given by a wine expert. An opinion on the quality of a wine cannot be perfectly mapped to objective measurements, therefore preserving the HD properties in an embedding does not consequently bring a good label separation. The opposite would be Coil-20, where each label corresponds to an object with a distinct HD characteristics: a good preservation of the HD properties brings a good separation of classes.

This work uses KNN gain [13] to bring a global evaluation of the class separation in the LD space compared to the HD space. KNN gain is defined as

$$G_{NN\ classification}(K) = \frac{1}{N}\sum_{i=1}^{N}\frac{|j \in n_i^K\ s.t.\ c_i = c_j| - |j \in \nu_i^K\ s.t.\ c_i = c_j|}{K} \in [-1, 1],$$

with n_i^K and ν_i^K the sets of K-nearest neighbours for observation i in LD and HD respectively, and c_i the label of point i. KNN gain takes positive values if the embedding brings an improvement over the HD data for a certain scale K, and negative values otherwise. This work also brings an adaptation of KNN gain for regression datasets, by using the LD and HD prediction errors:

$$G_{NN\ regression}(K) = \frac{1}{N}\sum_{i=1}^{N}\frac{(\varepsilon_i^K - e_i^K)/\sigma}{K},$$

with ε_i^K and e_i^K the absolute error between the value obtained by KNN regression on observation i and the ground truth, in HD and LD respectively. σ is the standard deviation of the target on the whole dataset.

Similarly to the $R_{NX}(K)$ curves, the AUC of the KNN gain can be computed to get an overall score; using a logarithmic scale for K:

$$AUC[G_{NN}(K)] = \frac{\sum_{K=1}^{N-2} G_{NN}(K)/K}{\sum_{K=1}^{N-2} K^{-1}}.$$

Figure 2.5 shows the profile of the KNN gain curves of embeddings of Coil-20 produced with t-SNE, $SQuaD-MDS$, and the hybrid between $SQuaD-MDS$ and t-SNE. The middle of the Y-axis corresponds to a KNN gain of zero.
The curve corresponding to $SQuaD-MDS$ is the one in bright orange with values

smaller than zero, the one with a sharp summit corresponds to *t*-SNE. In general (but not always), distance-preserving algorithms have lower values in their KNN gain curves than similarity-based algorithms.
The summit of the KNN-gain curve of the *t*-SNE embedding occurs at a value of $K = 71$. The Coil-20 dataset having 72 observations for each class, this spike is in line with the appearance of the embedding, where most classes are grouped into well defined and isolated clusters. This increase in class separation in the *t*-SNE embedding brings some information on the HD structure of the data; we can conclude that the "rings" of neighbours in high-dimension are less well defined and possibly interlocking.

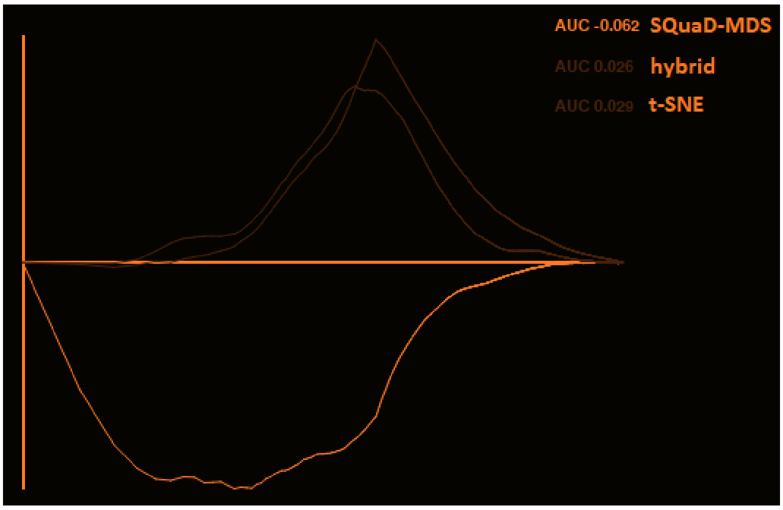


Figure 2.5: KNN gain curves for the three embeddings of the Coil-20 dataset shown in Figure 2.2

Part II

The software

The first objective of this work is to provide a software environment in which the user is able to fully indulge in the art of DR-based visualisation. This is done by providing tools to easily evaluate and compare the results while still offering the possibility to customise the experience by changing modules within the code. This second part of the paper goes into the details of the features of *the software*. Most of the explanations are accompanied by one or several use-cases, as a way to root *the software* into a context.

The next chapter brings a broad description of *the software*, without going into the details of the features. Chapter 4 explains the different elements of the main screen, chapter 5 shows the features of the QA screens.

Chapter 3

General overview

This chapter provides a broad description of *the software* without detailing its features.

Target users

To fully understand the purpose of this product and its design choices, one must first understand who this software is built for. Three profiles of users are targeted by this program:

- The domain expert who wishes to visualise his data on multiple embeddings. This user can take decisions based on the scatterplots and the various quality measures provided by *the software*.
  The domain expert should have beginner-level knowledge in programming, as loading a new dataset requires writing a function which outputs the data into a predefined format.
- The researcher in DR who wishes to implement new ideas and evaluate them.
- Machine-learning practitioners who wish to build their intuition in the domain of DR. Projecting different datasets using various algorithms is a way to understand how the true essence of information can be hidden inside the high-dimensional observed space. This kind of intuition can be useful in areas of machine-learning that aren't directly related to DR.

File architecture

The file structure of *the software* is schematised in Figure 3.1 . The files and folders framed with a green rectangle are those where a modification from the user is expected. The expected modifications are of four nature :

- Changing a general parameter of the software in the config.txt file (for instance, changing the resolution for the windowed mode or changing some shortcut keys).

- Loading a new dataset.

- Adding a new DR algorithm, or modifying an existing one.

- Changing the criteria for local QA.

To keep the details of the code outside of the bulk of this paper, the instructions on how to do these modifications are given in annexes D to F.

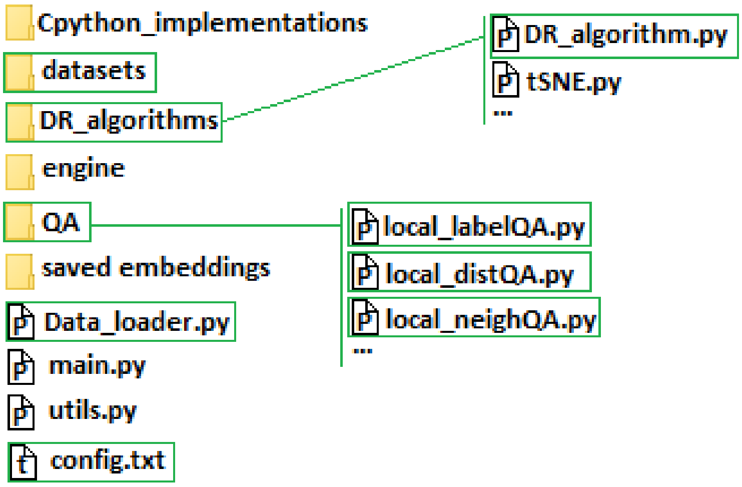


Figure 3.1: File organisation of the software, green frames indicate modifiable files

The program allows the user to save embeddings in a file, allowing them to be exported. These are located in the *saved_embeddings* folder. The saved embeddings are in the "*.npy*" format, in the form of a numpy matrix of shape (N, 3), N being the number of observations. For each entry in the matrix, the first two dimensions correspond to the embedding, and the third is the label.

Required libraries

This software is written in Python 3.8 and the usual machine-learning libraries are required (*numpy*, *scikit-learn* and *scipy*). Some DR algorithms require some specific libraries (for instance UMAP and FIt-NSE). For these algorithms however, the import is done only once the user attempts to launch a projection, meaning

that the program can still be executed if the libraries aren't installed. Attempting to launch a projection with a failing import statement aborts the projection but doesn't bring any further disruptions in the program.
The implementations of multi-scale *t*-SNE and its accelerated variant rely on some pre-compiled Cpython code, running these algorithms therefore requires the user to compile these files. To compile them, the user needs to go inside the folder "*Cpython_implementations*" and enter the command "*python setup.py install*".

To draw things on the screen and capture the user inputs, this software builds on top of the library *pygame* (which is available with the *pip* package management system, more information on www.pygame.org). Designed to code games in python, *pygame* offers all the necessary tools to build a graphical user interface (GUI), with no constraints in terms of visual design or execution pipeline.

In order to integrate the *pygame* library into a GUI context, a custom application programming interface (API) was built for this work. The API is designed to be easy to use and not too demanding in terms of computations when running.
To spare some computations, this API organises the screen in a tree structure, the mouse events are propagated through the tree instead of going through all the possible elements of the screen. On the same line, when something needs to be drawn, the update of the pixels are only done on the parts of the screen requiring a change. Additionally, the logic of this program is event-driven, meaning that the GUI doesn't use a lot of CPU resources when the user is not generating inputs.

Organisation into screens

This program has three screens: the main screen from which datasets can be loaded and projections can be launched, and two QA screens, one for absolute QA and one for comparative QA. The user can change between the screens by using keyboard shortcuts defined in *config.txt*.

Achilles's heel

In programming, there is often a balance between memory and speed: going faster means using more memory and vice-versa. In the case of this software the aim is to be ergonomic and responsive, for this reason, speed is favored to the detriment of memory usage. For instance, many QA assessment tools require a full HD distance matrix or ranking of neighbours, these are stored in memory in order to compute them only once.

As a consequence, the memory usage of *the software* can grow quickly with the dataset size. As a general rule of thumb, dataset sizes smaller than 5000 shouldn't cause trouble if using reasonably modern computers, but going above can bring problems depending on the computer. The program hasn't been tested with datasets bigger than 10^4 observations.
As an indicator, opening a dataset of size 5000 with four different scatterplots consumes 2.5GB memory.

The memory limitations of *the software* does not completely preclude the use of datasets made of tens of thousands of observations. For instance, the user can use a random sample of a large dataset and evaluate various embeddings on it. From there, the user can use his or her observations to project the full dataset from outside the software.
Another option is to save the best embedding of the sample by using *the software*'s save functionality, and then embed the remaining points inside the topology determined by the sampled points. There are multiple ways to place new points into an embedding, one of them is detailed in [20].

Chapter 4

The main screen

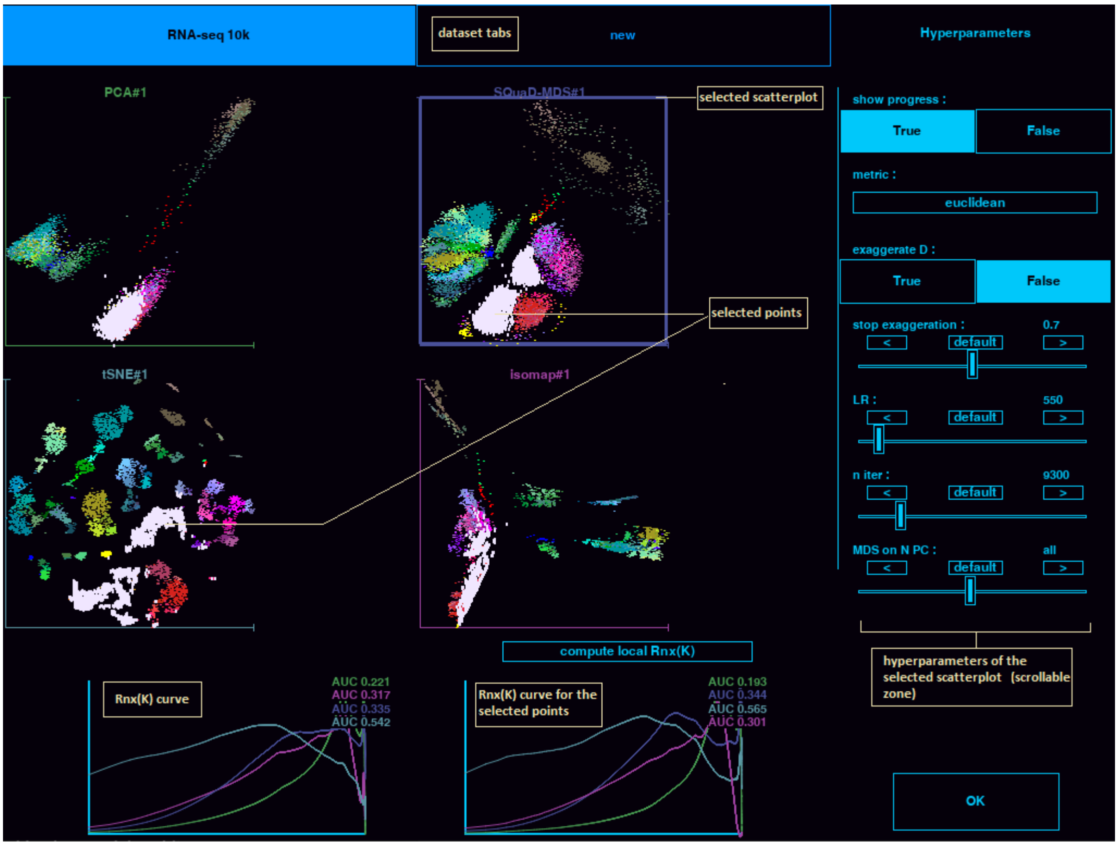
The main screen is where the user inputs most of his decisions: it is from there that new datasets can be loaded and new projections can be opened. It's also from the main screen that the user can select points in the scatterplots and open them as subsets of the datasets. This chapter details thee features of the main screen, and brings use-cases to justify their presence.

A recurring use-case in this chapter involves a dataset of single-cell transcriptomics, which are measurements of the gene expression levels on individual cells [23]. One observation in a single-cell transcriptomics dataset can be seen as a snapshot of the activity of the cell, a powerful tool to differentiate between cells or to study the dynamics of gene expression.

This single-cell transcriptomics dataset is referred to as *RNAseq* in this paper, it is a subset of 10^4, 50-dimensional observations from the dataset used in the paper "*The art of using t-sne for single-cell transcriptomics*" by D. Kobak and P. Berens [20]. The 50 dimensions correspond to the 50 principal components of a collection of genes selected as in [20].

4.1 Introductory overview of the main screen

Figure 4.1 shows a typical main-screen, with four scatterplots of the *RNAseq* dataset.
The top bar is used to jump between datasets, clicking on the "new" tab opens the option to load a new dataset. In this example, only one dataset is open. Right-clicking on the tab corresponding to a dataset closes it, freeing memory.



show progress : True False

metric : euclidean

exaggerate D : True False

stop exaggeration : 0.7

LR : 550

n iter : 9300

MDS on N PC : all

Figure 4.1: The elements of the main screen

The majority of the surface is occupied by the scatterplots field. This field can show up to eight scatterplots at once. Clicking on a scatterplot opens its hyperparameters in the right side bar, they can be modified and the scatterplot can be re-launched by clicking the "ok" button at the bottom of this bar.
The scatterplots field can be right-clicked, this opens a window which gives the option to launch a new projection.
Clicking inside the selected scatterplot selects the K-nearest points to the click location, the parameter K can be changed by right-clicking inside a scatterplot. Another way to select points is by drawing a boundary while pressing the "control" key.

At the bottom left of the screen, a general $R_{NX}(K)$ plot is shown, with a color code corresponding to the open scatterplots. At the right of this, a local $R_{NX}(K)$ plot is shown, where the quality is only evaluated on the selected points.

4.2 Imputation of missing values

Some datasets have missing values, this is problematic when doing DR as most algorithms are designed to use a filled data-matrix. *The software* enables the use of datasets with missing values by automatically imputing the missing values when detecting entries of the type "*numpy.nan*".

A naive method for missing value imputation is simply replacing the missing values by the mean of the variable. The limitation of this simplistic method is that it doesn't take into account the information present in the non-missing values of the target data point.

An improvement on the naive method is by finding the K-nearest neighbours (KNN) of the observation of interest according to its variables that have a value. The missing value can then be imputed from these neighbours, for instance by taking their mean value on the variable of interest.
This method tends to bring good results when applicable but it has a limitation: when the dimension gets high or when the percentage of missing values grows, the number of valid candidates on which to find the KNN quickly shrink to zero.
For instance, on the *Coil*-20 dataset (1440 observations of dimensionality 1024), having 7% of missing values makes each point unique in their profile of non-missing values, making a straightforward search of the KNN impossible.

This work attempts to counter the weakness of the default KNN-based algorithm by taking inspiration from random forests: the neighbours are found according to random samples of a small number M' of features. The idea motivating this approach is that having a small amount of features makes the probabilities of having valid candidates for the KNN search much higher than using the full set of features. The algorithm is described in Algorithm 1:

Algorithm 1: Imputation of missing values by using multiple feature samples.

```
Data: X_missing, Nb_resamples
Result: X without missing values
N, M $\leftarrow$ X_missing.shape() // number of observations and of features
N_features $\leftarrow$ binary_search_N_features(X_missing)
V $\leftarrow$ zeros(N,M) // V for ’votes’
C $\leftarrow$ zeros(N,M) // C for ’counts’
$\sigma \leftarrow$ mean_std(X_missing)
while passes $\leq$ Nb_resamples $\wedge$ elapsed() $\leq$ 30s do
    $R_f \leftarrow$ random_features(N_features, M)
    eligibles $\leftarrow$ nonmissing_indices(X_missing[:, $R_f$])
    $\mathcal{N}^1 \leftarrow$ first_neighbours(X_missing[eligibles, $R_f$])
    foreach i$_e \in \{1, ..., |eligibles|\}$ do
        i $\leftarrow$ eligibles[i$_e$]
        j $\leftarrow$ eligibles$\Big[\mathcal{N}^1[\mathrm{i}_e]\Big]$ // the first neighbour according to features $R_f$
        $s_{\mathrm{ij}} \leftarrow$ nonmissing_similarity(X_missing, i, j, $\sigma$)
        for $m \in \{1, ..., M\}$ do
            if is_missing(X_missing[i, m]) then
                if $\neg$ is_missing(X_missing[j, m]) then
                    V[i, m] $\leftarrow$ V[i, m] + $s_{\mathrm{ij}}$*X_missing[j, m]
                    C[i, m] $\leftarrow$ C[i, m] + $s_{\mathrm{ij}}$
                end
            end
        end
    end
end
for i $\in \{1, ..., N\}$ do
    for m $\in \{1, ..., M\}$ do
        if is_missing(X_missing[i,m]) $\wedge$ $C$[i,m] $> 0$ then
            X_missing[i, m] $\leftarrow$ V[i, m] / C[i, m]
        end
    end
end
return mean_imputation(X_missing)
```

The function $binary\_search\_N\_features$ finds a value $N\_features$ for the number of features to sample at each iteration such that approximately 6% of the observations have non-missing values when only looking at $N\_features$. The number $N\_features$ is constrained to be greater than 2 and smaller than $max(2, M-3)$

with M the dimensionality of the data. Using an upper bound for $N\_features$ is a way to ensure that there is some diversity in the different combinations of features that can be sampled. The target of 6% of the dataset is arbitrarily chosen, this value brings good results on the tested datasets.

The function *mean_std* computes the mean of the standard deviations of each variable, ignoring missing values. This implies that the data is expected to have variables that are normalised in terms of scale, which is usually the case in DR.

The function *nonmissing_indices* finds the indices of the observations where all the values in the sampled features are present.

The function $first\_neighbours$ finds the nearest neighbour for each point of the dataset given in argument. This dataset is the set of points in *eligibles* with only the sampled features, it is therefore a dataset with no missing values. In practice, the neighbour search is done by constructing a KD-tree [24].

The function *nonmissing_similarity* computes a similarity s_{ij} between two points indexed i and j: $s_{ij} = exp(-d^f_{ij}/(0.31\sigma^2))$. With d^f_{ij} a measure of distance between the i^{th} point and the j^{th} point, this distance corresponds to the mean squared difference in the variables where both points have a value. The value 0.31 in the denominator is arbitrary, this value brings good results on the tested datasets.

The function *mean_imputation* at the end of the algorithm performs a naive imputation of values by taking the mean value of the target variables. Calling this function is a way to ensure that every missing value receives a value. This is necessary as there is no guarantee that the KNN-based algorithm finds a value for each missing value. In practice, if there are any values that were missed by the algorithm, they constitute a small percentage of the values that were originally missing.

The time complexity of this algorithm with regards to the number of points N is determined by the computation of the KD-tree, which is used to compute the nearest neighbours. This makes the time scale with $\mathcal{O}(N \log N)$. This value should be nuanced though, as in practice, a high number of random samplings are done on the features. When the dimensionality M is large with respect to N, the time complexity can also be expressed as $\mathcal{O}(NM)$
In practice, the time to do the imputation of missing values with this algorithm in the context of *the software* is in the range of seconds or tens of seconds, which is deemed reasonable by this author. In case the imputation takes too long on a

certain dataset, the implementation of this algorithm in *the software* stops when the elapsed time is greater than 30 seconds.

A quantitative quality assessment of the proposed algorithm is presented in Table 4.1. This table shows the mean classification accuracy of KNN classifiers applied on datasets that were subject to missing value imputations. Calling N the number of observations and M the dimensionality of the data, the evaluated datasets have the following characteristics : *Coil*-20: $(N, M) = (1440, 1024)$; *blob* : $(N, M) = (2000, 25)$; *RNAseq* : $(N, M) = (10^4, 50)$; *winequality* : $(N, M) = (1599, 11)$; *anuran* : $(N, M) = (7195, 22)$.
The missing values were simulated as these datasets do not naturally come with missing values, the rows of the table correspond to a certain percentage of missing values.

The shown accuracies are the mean values obtained by training 5-nearest neighbours classifiers on 20 random splits of the data. Each split determines a training set and a testing set of size 65% and 35% of the dataset size respectively.
The values displayed in parenthesis are the accuracies obtained by using the same protocol on datasets whose missing values were imputed using the naive method.

amount missing	Coil-20	blob	RNAseq	winequality	anuran
0 %	0.95	0.81	0.90	0.28	0.98
7 %	0.95 (0.95)	0.78 (0.77)	0.90 (0.86)	0.26 (0.26)	0.98 (0.98)
17 %	0.95 (0.93)	0.74 (0.73)	0.90 (0.79)	0.23 (0.21)	0.97 (0.96)
27 %	0.95 (0.92)	0.70 (0.65)	0.89 (0.73)	0.20 (0.19)	0.97 (0.94)
37 %	0.95 (0.90)	0.65 (0.60)	0.87 (0.66)	0.13 (0.12)	0.97 (0.91)
47 %	0.95 (0.82)	0.58 (0.52)	0.82 (0.56)	0.10 (0.10)	0.96 (0.89)

Table 4.1: Accuracies obtained on datasets subject to missing value imputation.

Table 4.1 shows that this algorithm has a performance that varies a lot from one dataset to the other, but it tends to bring better results than the naive imputation of values. The reader might also notice that the higher the percentage of missing values, the better the performance with regards to the naive method.

A more qualitative evaluation of the algorithm can be done by looking at Figures 4.2 and 4.3. The first figure shows t-SNE plots of the *Coil*-20 dataset which was subject to missing values imputation, with a growing number of missing values. The top scatterplots correspond to the described algorithm, and the bottom ones to the naive method, which uses the mean values of the missing variables.
In the scatterplots associated with the naive method, the clusters quickly become

blurry as *t*-SNE has difficulties to distinguish between similar data points. The KNN-based method keeps sharp clusters, even when 67% of the values are missing.

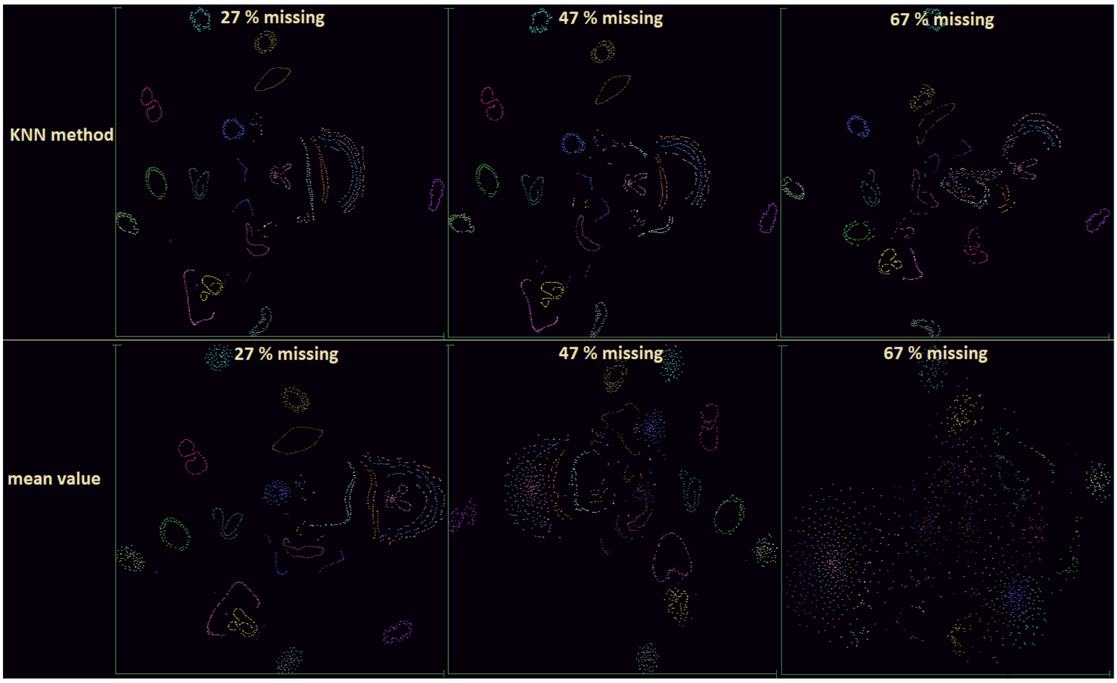


Figure 4.2: *t*-SNE plots of *Coil*-20 with missing values imputed by using the KNN method (top) and the naive method (bottom).

Figure 4.3 shows how some pictures from the *Coil*-20 dataset look like after performing the missing value imputation. The images from the naive method tend to become blurry and less distinguishable. This is in line with what can be seen in the plots of Figure 4.2, where the blurry and circular clusters indicate observations that are hard to distinguish from one another.

The qualitative results of Figures 4.2 and 4.3 should however be taken with reservation, because the *Coil*-20 dataset is a particularly easy dataset for value imputation, as hinted by the quantitative results of Table 4.1 . The easiness of performing values imputation on *Coil*-20 could come from the fact that each observation has at least two very similar neighbours, which makes the imputation quite accurate. A HD space with a more homogeneous distribution of points would probably bring less impressive results.

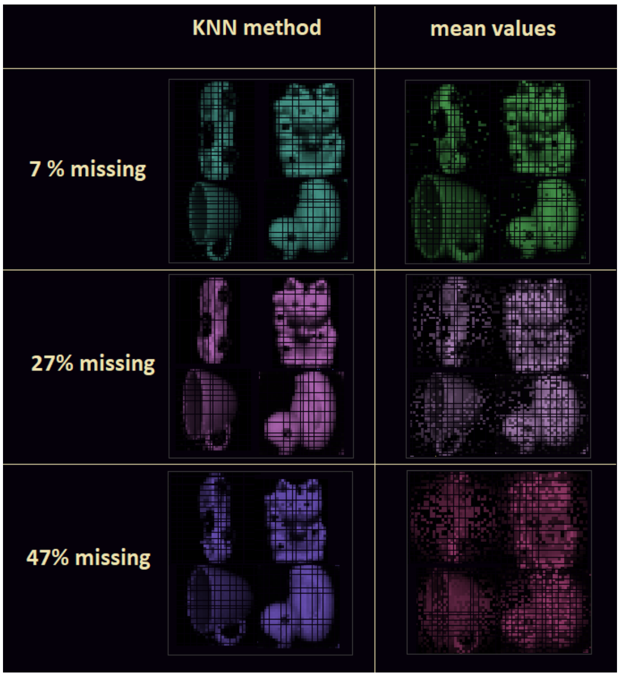


Figure 4.3: Objects of *Coil*-20 after missing values imputation by both methods

4.3 The scatterplots

The scatterplots field occupies most of the main screen's surface. The option to open a new scatterplot is given to the user when right-clicking in this field. When the *control* key is pressed, two buttons appear on each scatterplot, these buttons allow the user to close a scatterplot or to save it.

Four important features of the scatterplots field can be brought to attention:

- The ability to see multiple scatterplots at once.
- The ability to see the state of the embeddings while they are being optimised.
- The ability to inspect the HD values of the points.
- The ability to select points inside the scatterplots.

Multiple scatterplots at once

Having multiple scatterplots on the same screen is one of the most important features of this program, as it enables the user to look at the data from multiple points of view.

To put this feature in a context, in [20], the authors start their visualisation process by showing a PCA embedding of the *RNAseq* dataset alongside with multiple t-SNE embeddings and an MDS embedding on points corresponding to the class means of the dataset. The reason why they use class means for MDS is because the common MDS algorithms have high computational complexities, making them unreasonable for datasets larger than a couple of thousands of points. In the case of this software, using class means is not necessary, as the time and space complexities of SQuaD-MDS with regards to N are $\mathcal{O}(N)$.

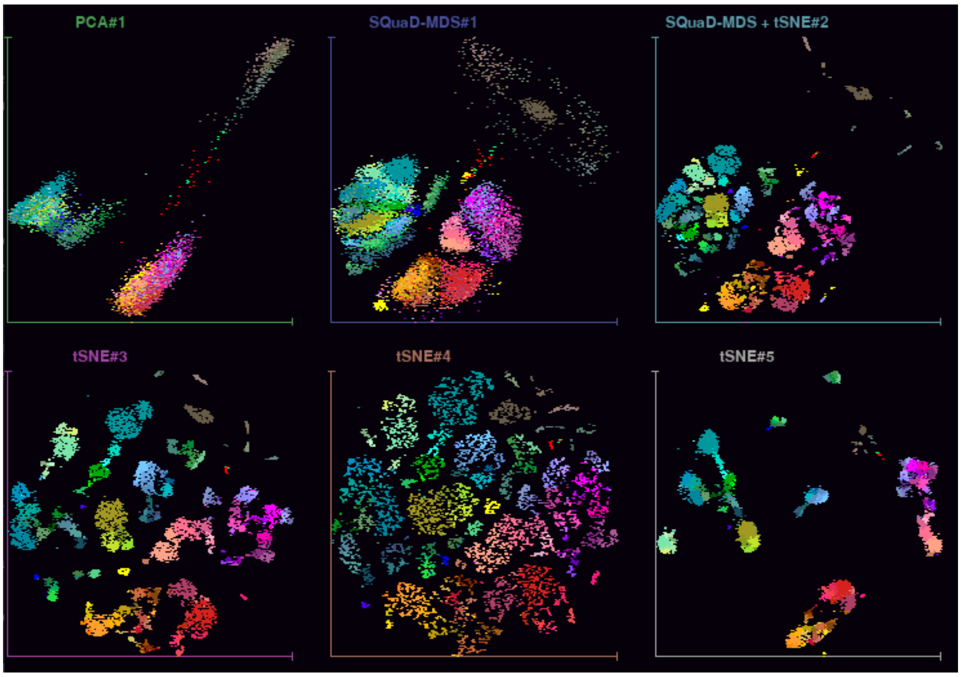


Figure 4.4: Multiple points of view on *RNAseq*. From left to right, the t-SNE embeddings have perplexities : 30, 7, and 390

Figure 4.4 recreates this scenario using *the software*. The various points of view capture a landscape on multiple scales. The fact that these embeddings can be launched in just a couple of clicks isn't a luxury, it's a way to encourage the user to try different things. It is by trying different algorithms and combinations of hyperparameters that one can develop intuition in DR, or find the best configuration to analyse the data.

Seeing multiple embeddings at once is also useful when a DR researcher is

implementing an algorithm, as it is a way to visually evaluate the impact of certain hyperparameter values on the algorithm in construction.
This has greatly facilitated the development of SQuaD-MDS during its implementation, when deciding between different approaches, and after the implementation, when searching for good hyperparameter values.

Visible optimisation

Some implementations of the DR algorithms in *the software* can show the state of the embedding during the optimisation process. The algorithms that have this functionality are: *t*-SNE, CA*t*-SNE, SQuaD-MDS and it hybrid variant, as well as the autoencoder.
When implementing an algorithm, the user can send the current state of an embedding to the screen in one line of code (more details in : *Annex E: Adding a new DR algorithm to the software*).

Being able to see the optimisation in real time is a good way to develop intuition on the process.
This feature can also be used to tune hyperparameters, for instance if the end of an optimisation still involves considerable movement, the number of iterations could be too low.
It is also a useful diagnostics tool when implementing an algorithm, as some errors can be hard to understand when only looking at the end result. For instance, a normal looking optimisation that suddenly "explodes" could be a sign of numerical instability, such as an underflow.
Some optimisations can be appealing look at, watching the process can also be a way to occupy the user when waiting for an optimisation to finish.

Inspecting the high-dimensional values

When hovering above points in a scatterplot, the user can press *control* to see an image corresponding to the HD values of the nearest point, this is shown in a small window next to the mouse. Keeping control pressed and moving the mouse brings an instant update on the shown image, allowing the user to scan through the embedding.

The image is a square divided into smaller squares, each small square corresponds to a HD variable, its brightness reflects the HD value. An example is shown when evaluating the quality of missing values imputation in Figure 4.3.

Seeing the HD values corresponding to points within a certain structure can help to understand why an embedding is organised as it is. For instance, inspecting the HD values inside a cluster might reveal that a certain direction corresponds to changes in a particular variable.

Selecting points

Clicking inside a scatterplot selects the K points that are closest to the mouse location, the parameter K can be changed by right-clicking inside the scatterplot. The selected points appear on all the open scatterplots simultaneously.

Moving the mouse while maintaining the click makes the selection follow the movement of the mouse. Movement brings another dimension to the visual comparison of the scatterplots. For instance, the user can scan a MDS embedding along a certain direction while simultaneously looking at the corresponding points in a t-SNE embedding. This allows the user to bring context to the t-SNE projection with regards to the large-scale organisation of the data represented in the MDS embedding.

Another way to select points in a scatterplot is by drawing a boundary, the points inside the boundary are selected. To draw a boundary, the user needs to keep *control* pressed while drawing with the mouse. This is mostly useful for recursive exploration, the subject of the next section.

4.4 Recursive exploration

One way to explore the structures of a dataset on multiple scales is to explore it recursively: the user can first look at the global structures on the whole dataset, and then zoom in by only considering a subset of the dataset.
This approach is especially interesting when the data presents a hierarchical organisation, for instance, *RNAseq* contains a first level of organisation separating neurons from non-neurons, a second level organises the neurons into two distinct groups: inhibitory and excitatory, and a third level separates these into cell classes.

The software allows the user to recursively explore the datasets by pressing the 'o' key when points are selected in a scatterplots. This opens a new dataset tab containing the selected observations.

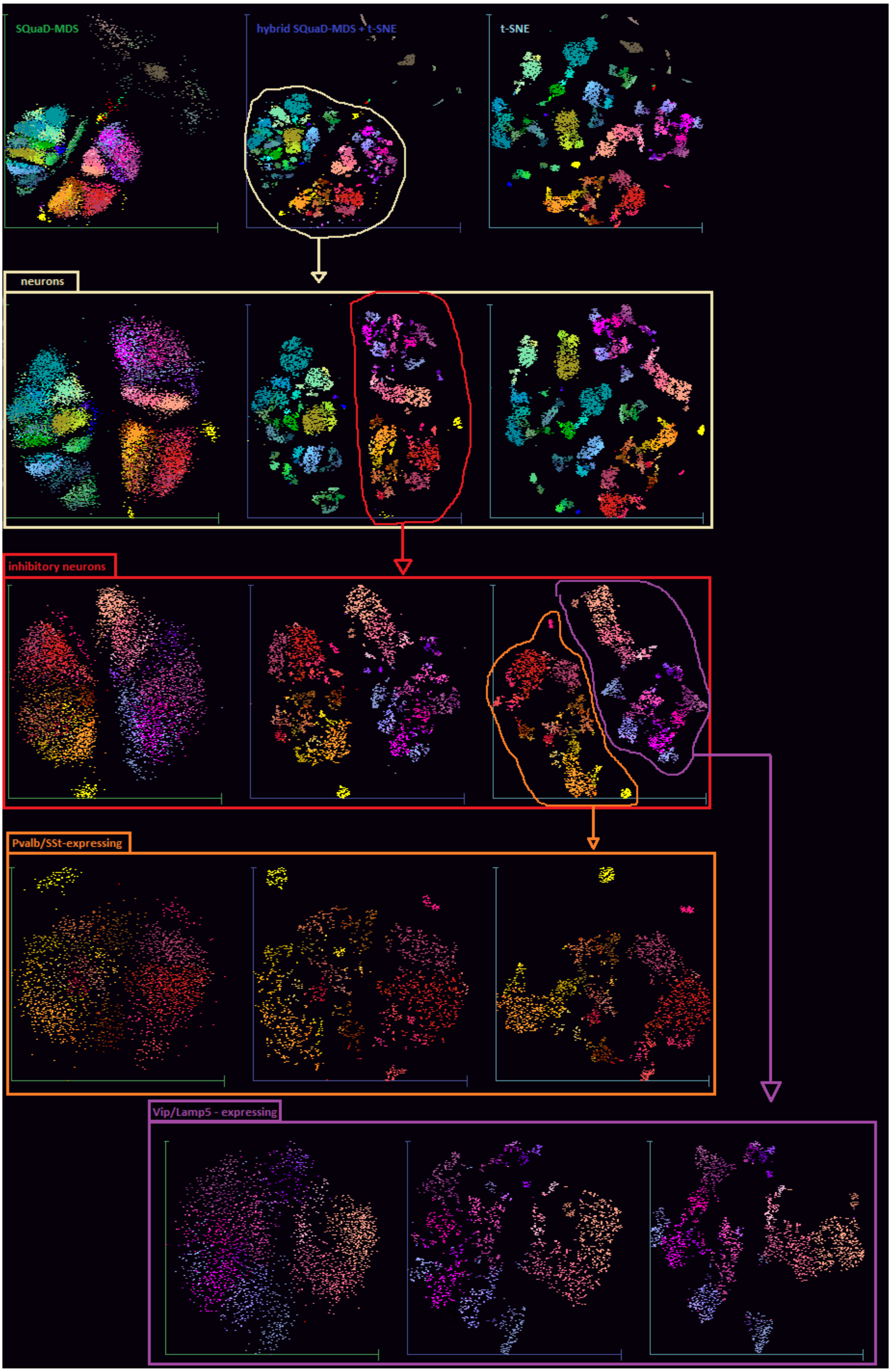


Figure 4.5: Recursive exploration of the $RNAseq$ dataset. From left to right, the embeddings were generated by $SQuaD - MDS$, the $SQuaD - MDS + t - SNE$ hybrid, and $t - SNE$.

The biologist looking at single-cell data can thus have multiple tabs opens, each corresponding to a level of 'zoom' in a certain dataset, jumping from one to the other by simply clicking on the appropriate tabs. Figure 4.5 illustrates this approach with a hierarchical exploration of $RNAseq$, various subsets corresponding to types of neurons cited in [20] are shown.

It should be brought to attention that the clusters tend to organise differently when seen from different levels of zoom. An explanation for this observation is that, when eliminating certain elements of the data by zooming in, the local structures have more freedom to organise themselves without external influence.

4.5 $R_{nx}(K)$ plots

In its main screen, *the software* also plots two $R_{nx}(K)$ curves for each open embedding. The curves are colour-coded and all share the same X- and Y-axis, facilitating their comparison.

The first $R_{nx}(K)$ curve corresponds to the values of $R_{nx}(K)$ computed on the whole dataset. This curve and its AUC are a good way to have a quick estimation of the overall quality of the embeddings. The profile of the $R_{nx}(K)$ curve also brings information on the scale at which the structures are best preserved. This plot is located on the left of the bottom bar.
The second $R_{nx}(K)$ curve is applied to the selected points only, this allows the user to quantify the neighbourhood preservation quality in specific zones of the embeddings. This plot is located on the right of the bottom bar.

Figure 4.6 illustrates how using the global and the local $R_{nx}(K)$ plots together can help identify zones of poorer quality in embeddings. This figure shows a t-SNE and a PCA embedding of the *blob-25* dataset, which contains 2000, 25-dimensional observations separated into normally distributed clusters of points.
In the t-SNE embedding, the center of the clusters only contain a single class, but the labels tend to mix in the boundaries between the clusters. By observing this, one might wonder if the quality in terms of neighbour preservation is lessened in the zones between the clusters.
Figure 4.6 shows that the AUC of the t-SNE's $R_{nx}(K)$ curve applied to the points located between the clusters is indeed worse than the AUC of global $R_{nx}(K)$ curve. Interestingly, the percentage of the decrease in quality between the AUC of the global and the local $R_{nx}(K)$ is roughly the same between the t-SNE and the PCA plots. This could indicate that the poor placement of these points comes from them being generally hard to place, and not from a weakness proper to t-SNE.

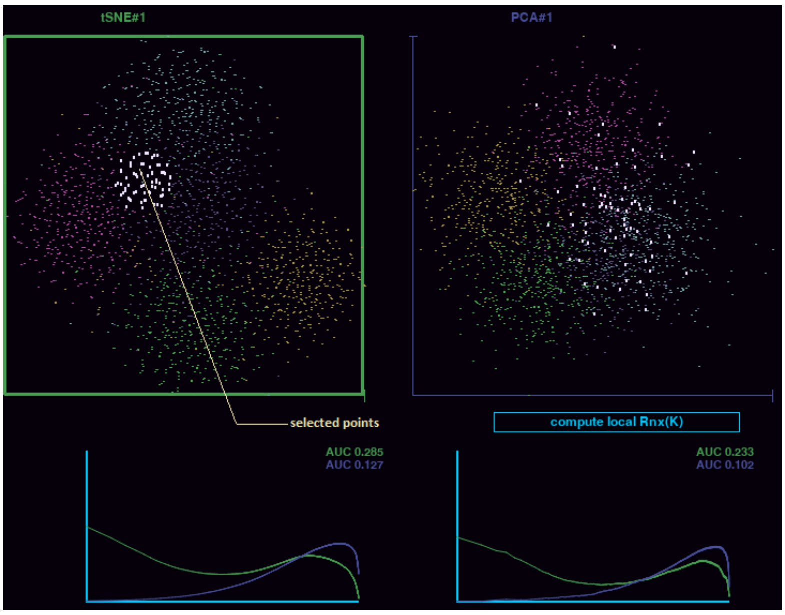


Figure 4.6: t-SNE and PCA embeddings of 25-dimensional observations with their associated global $R_{nx}(K)$ curve. The local $R_{nx}(K)$ curve is applied to the selected points.

Chapter 5

The quality assessment screens

The software comes with two QA screens: one for absolute QA, and the other for relative QA, where embeddings can be compared. The user can jump between the screens by pressing keyboard shortcuts, which are determined in *config.txt*. By default, the main screen is associated with the key 'x', the absolute QA screen to the key 'v', and the relative QA screen to the key 'c'.
This program uses three types of information for QA : information based on the distances, the neighbours, or the distribution of labels. Both QA screens have three different views: one for each type of QA, these views are called "distances", "neighbours" and "labels" in *the software*.

Section 1 brings an overview of the organisation of the absolute QA and relative QA screens. Section 2 is separated into three parts, one for each view.

5.1 Organisation of the quality assessment screens

This section explains the general organisation of both QA screens, without entering into the details of the information displayed on these screens.

Organisation of the absolute QA screen

Figure 5.1 shows the absolute QA screen, the embedding being assessed is a projection of a subset of *RNAseq* produced with the SQuaD-MDS + t-SNE hybrid method.

At the left of the screen, a column shows the open embeddings of the current dataset. Clicking on one of these embeddings selects it as the embedding being assessed, the selected embedding is highlighted with a coloured frame. In this column, each embedding is accompanied by four scalars labeled "Dcorr", "Rnx", and "KNN", these correspond to the PCC between HD and LD distances, the AUC of the $R_{nx}(K)$ curve, and the AUC of the KNN-gain curve, respectively.

The buttons "distances", "neighbours", and "labels" visible at the top of Figure 5.1 can be clicked to jump between the views mentioned in this chapter's introductory paragraph.

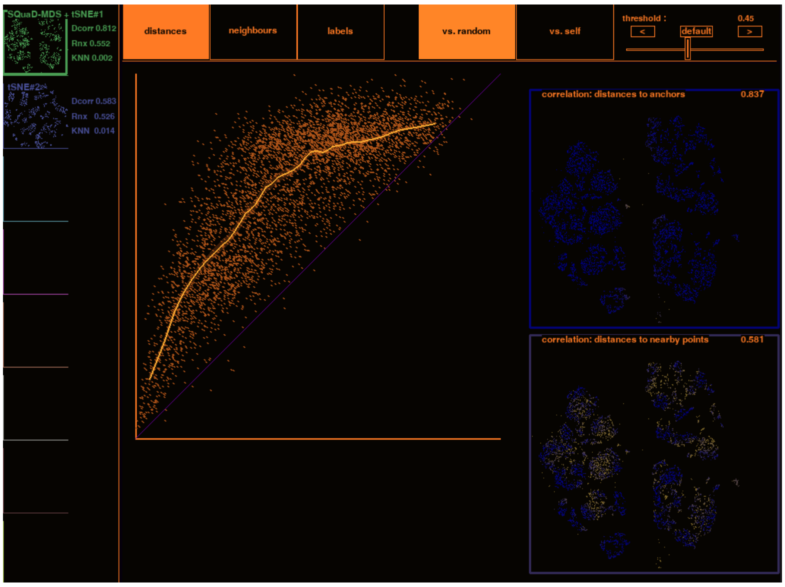


Figure 5.1: Absolute quality assessment screen, showing the "distances" tab

By default, the scores used to display the local qualities of the embeddings quantify the improvement over an embedding with random coordinates. When the local score's values are very similar throughout the embedding, the score can show too little nuance between the different zones of the scatterplot. As a way to inflate the differences in scores throughout the embedding, the user can choose to see values that are scaled by using the maximum and minimum values or the scores. The self-scaled score is defined as $s_i^{self} = (s_i - min(s))/(max(s) - min(s))$, with s the vector containing the scores of the points with regards to a randomly generated embedding. The user can go from one display mode to the other by pressing the buttons "vs. random" and "vs. self", visible at the top of Figure 5.1.

Each scatterplot displaying point-wise scores also shows a scalar number, representing the mean score. This scalar uses the score computed with regards to randomness, it can therefore be used for a fair comparison between the embeddings. This scalar is displayed at the top-right of each scatterplot, as shown in Figure 5.1.

When showing the point-wise quality in a scatterplot, the absolute QA screen transforms a score with values in $[0, 1]$ into shades of blue or yellow. The yellow colour corresponds to low values (bad quality), and the blue colour to points of good quality.

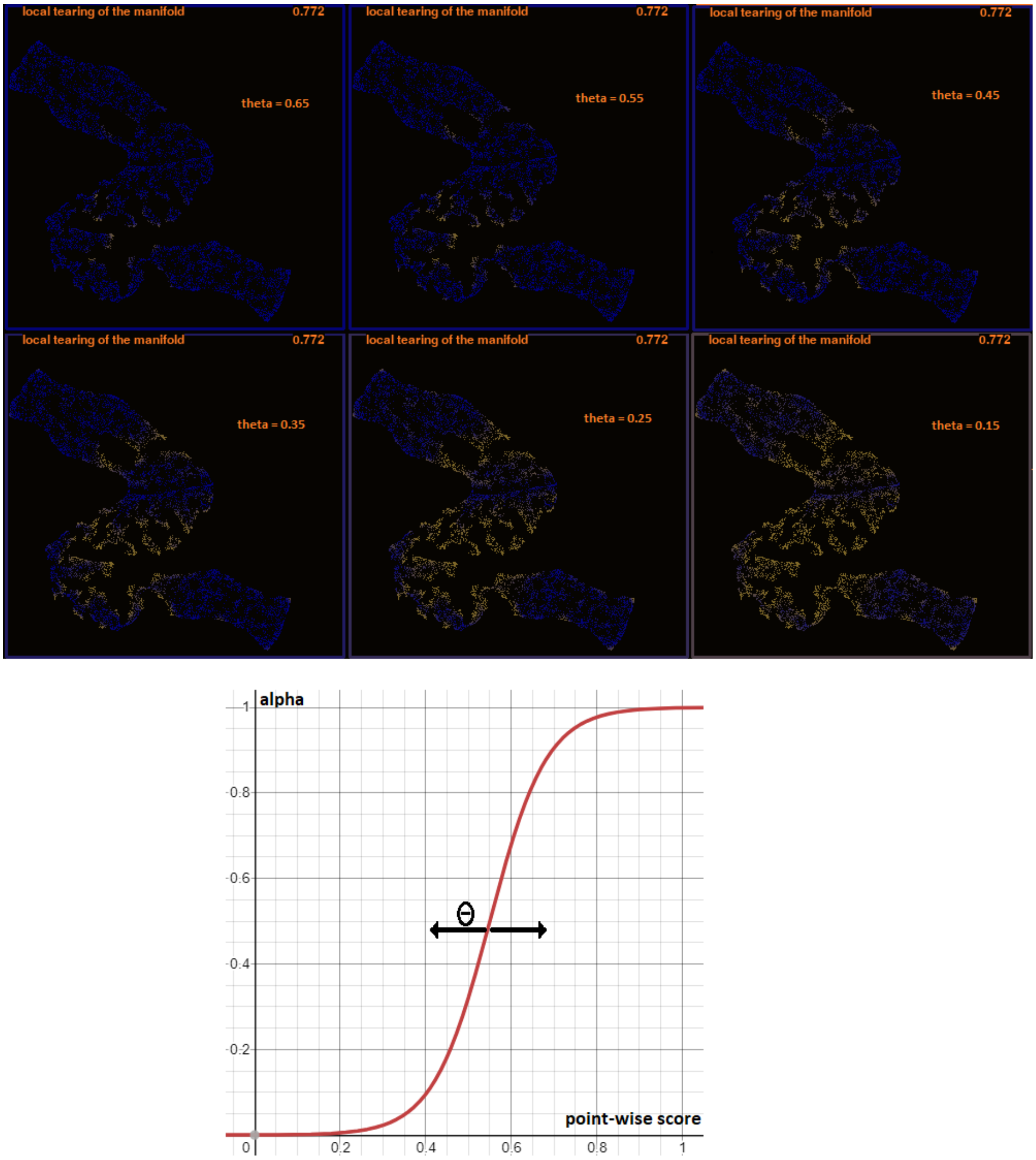


Figure 5.2: Influence of the parameter θ in the mapping between pointwise scores and colours.

In order to give the user control on the value of the score at which the colours should turn blue or yellow, the *the software* maps the score of the i^{th} point to the displayed colour with the following function : $colour_i = \alpha_i B + (1 - \alpha_i)Y$, with B the 3-dimensional vector corresponding to the red-green-blue code for the colour blue, and Y the vector for the colour yellow. α_i is computed with the relation $\alpha_i = \Big(1 + \exp\left(-15(s_i + \theta - 1)\right)\Big)^{-1} \in [0, 1]$, with s_i the score given for the i^{th} point, and θ a user-defined parameter. θ can be changed with the slider named "threshold" shown at the top-right of Figure 5.1.

The bottom part of Figure 5.2 shows how alpha changes with regards to the point-wise score. This plot also illustrates how the user-defined parameter can shift the cutoff along the X-axis.
The top part of Figure 5.2 shows the influence of the parameter θ on the scatterplots: the smaller θ, the higher the cutoff for α, resulting in more yellow points. The criterion being assessed here is the preservation of the manifold, more details on this will be brought in section 2.

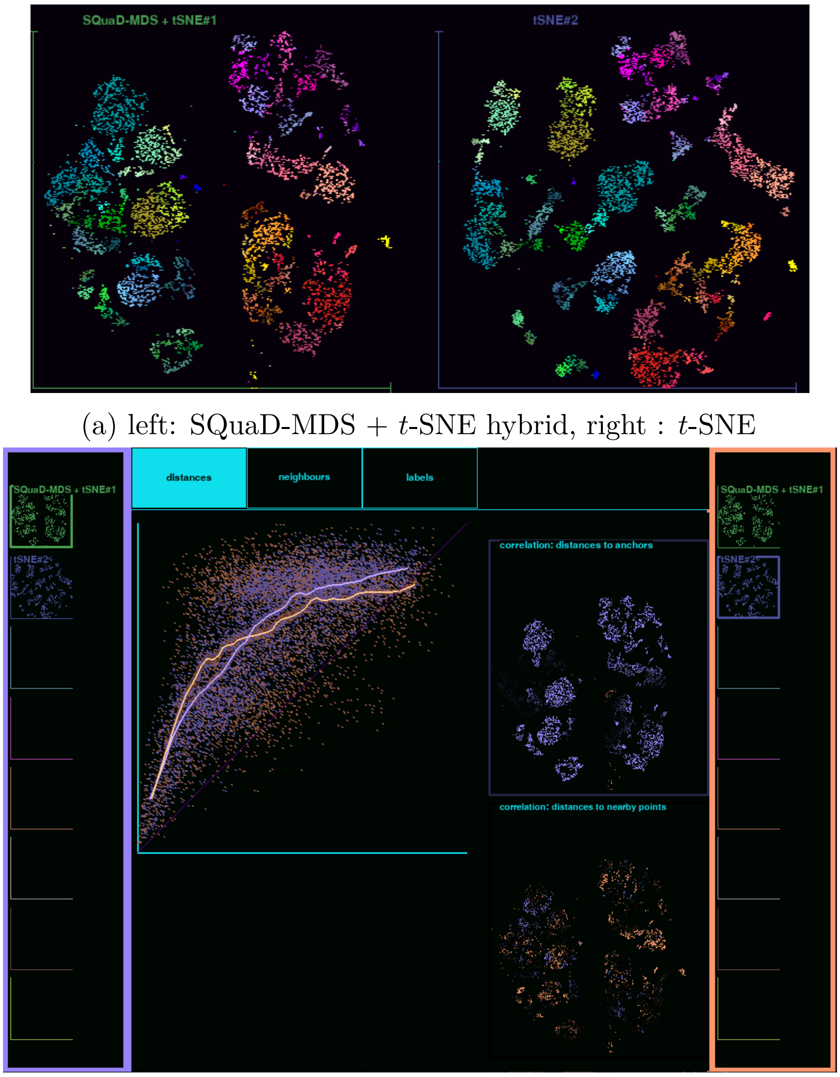
Organisation of the relative QA screen

The second QA screen is designed to allow a comparative assessment of the embeddings. Figure 5.3 shows the organisation of the screen.

This screen contains two clickable columns: one on the left (lavender-coloured) and one on the right of the screen (in orange). The columns each contain a list of the open embeddings, selecting a scatterplot from the list assigns it the colour of the column. The QA plots and scatterplots are coloured to show a comparison between the two selected scatterplots.

Non-scatterplot QA plots, such as the Shepard diagram of Figure 5.3.b, show a superposition of the values taken by the selected embeddings. The QA scatterplots colour the points with the colour of the embedding that performs best, with darker colours when the difference in quality is small. The topology on which the colouring takes place is the topology drawn by the embedding selected in the left column.

In Figure 5.3.b, the top-right scatterplot shows that, with regards to the evaluated criterion, most of the right-hand side cluster is coloured in lavender, meaning that the embedding selected in the left column performs better than the one selected on the right. Some parts of the left-hand side cluster are associated to a dark colour, indicating zones of similar quality between both embeddings. The bottom scatterplot shows more orange than lavender, indicating that it uses a criterion that favors the embedding selected in the right column.



(a) left: SQuaD-MDS + *t*-SNE hybrid, right : *t*-SNE

(b) The "distances" tab of the relative quality assessment screen, comparing the embeddings of Figure 5.1.a.
The colours in the scatterplots correspond to the colour of the embedding which performs best.

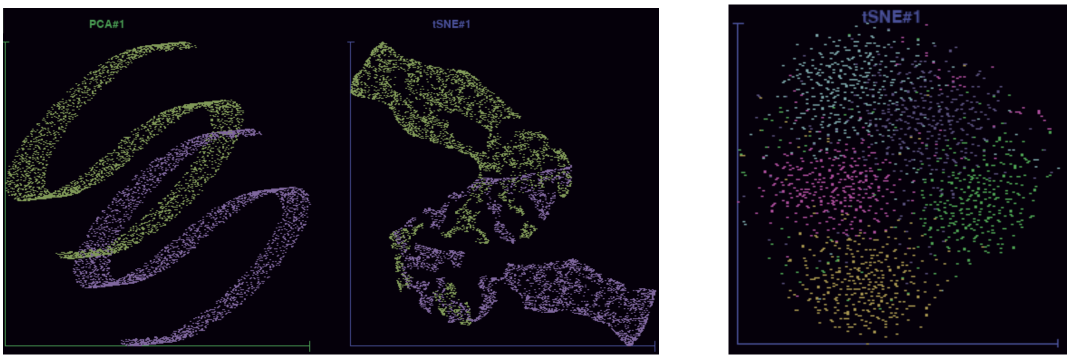
Figure 5.3: Embeddings of a subset of *RNAseq* and relative quality assessment screen

In Figure 5.3.b, the embedding selected on the left is produced with the hybrid SQuaD-MDS + *t*-SNE method, and the embedding on the right is produced with *t*-SNE. The criterion used by the top-right scatterplot measures the preservation of distances between the large-scale structures, and the criterion used by the bottom scatterplot measures the preservation of local distances. The comparative evaluation of both embeddings shows that t-SNE is better at preserving the small-scale structure, and the hybrid is better at preserving the large-scale relations.

5.2 The content of each view

This section brings an overview for each of the three views available in the QA screens, these views are named "distances", "neighbours", and "labels".

Using *RNAseq* as an example to demonstrate the QA tools would not be practical, as the dataset doesn't have a HD distribution that is easily understandable for most readers. For this reason, this section uses two easier datasets: *blob* and *double-S*.
Already featured in chapter 4, *blob* contains 2000, 25-dimensional observations separated into five normally distributed clusters of points, each cluster is assigned a different label.
The second dataset, *double-S*, is a 3-dimensional set of 10^4 observations. The observations are uniformly distributed along two 2-dimensional manifolds folded in the "S" shape, each "S" is assigned a label. The manifolds intersect, as shown in the PCA embedding of Figure 5.4.a.



(a) PCA and t-SNE embeddings of *double-S*

(b) t-SNE embedding of *blob*

Figure 5.4: Embeddings of the two datasets used in this section

5.2.1 Evaluation using the distances

The *distances* view shows three QA plots:

- A Shepard diagram allows the user to evaluate how the distribution of LD distances compares to the distribution of the HD distances.

- A local QA scatterplot titled "*correlation: distances to anchors*". This scatterplot shows the PCC of the HD and LD distances between each point and some points designated as *anchors* points.
  Anchor points are designated when a dataset is opened by performing a recursive clustering of the HD data, and assigning the points nearest to the cluster centers as *anchors*. The idea behind this clustering is to capture the different structures of the data on multiple scales. Usually, around 100 points are designated as *anchors* points.
  The score displayed by this scatterplot reflects the quality of the preservation of distances between large or medium-scale structures.

- A local QA scatterplot titled "*correlation: distances to nearby points*". This scatterplot shows the PCC of the HD and LD distances between each point and its closest neighbours in the HD space.
  The number of considered neighbours is arbitrarily chosen to be equal to five percent of the total number of observations. The score displayed by this scatterplot reflects the quality of the preservation of distances within the small-scale structures.

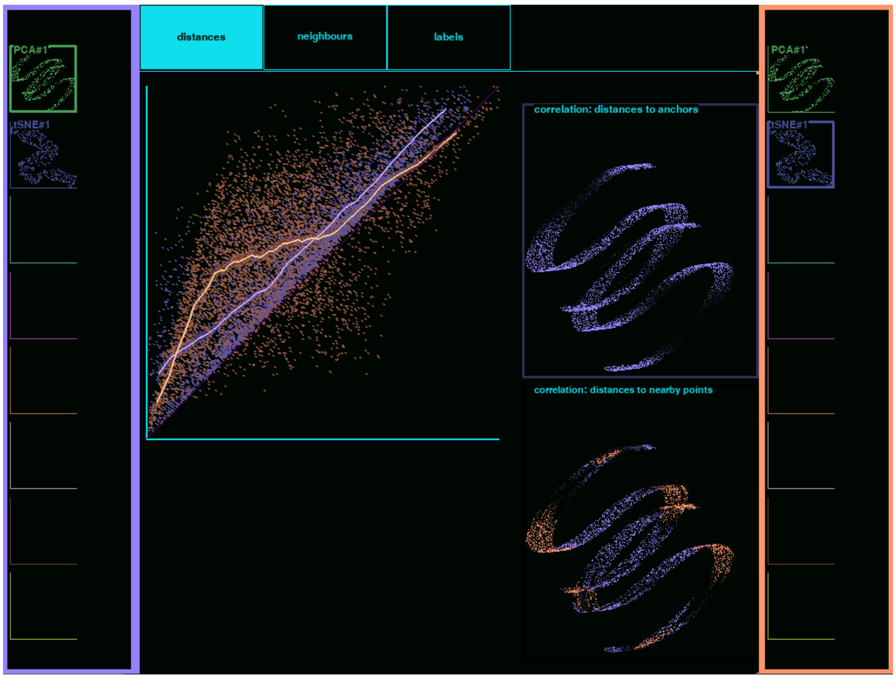

distances
neighbours
labels
PCA#1
tSNE#1
PCA#1'
tSNE#1'
correlation: distances to anchors
correlation: distances to nearby points

Figure 5.5: "Distances" view of the relative QA screen. The PCA embedding (lavender) of *double-S* is compared to the *t*-SNE embedding (orange).

Figure 5.5 shows the *distances* view of the relative QA screen, with a PCA embedding of *double-S* selected on the left (lavender), and a t-SNE embedding selected on the right (orange). These are the same embeddings that are shown in Figure 5.3. The Shepard diagram shows that PCA does a better job at preserving a linear relationship between the LD and HD distances than t-SNE. This observation is in line with the information shown in the *correlation: distances to anchors* scatterplot, which is mostly lavender-coloured.

When looking at the preservation of the small-scale distances however, some zones are of better quality in the t-SNE embedding, these zones tend to be located where the manifold is most bent. This can be surprising considering that t-SNE performs a non-linear projection of the HD data. This could lead towards the postulate that the relative bad quality of the PCA embedding in these zones is not due to a particularly good performance of t-SNE, but due to a bad performance of PCA. To bring substance to this argument, the user can change to the absolute QA screen version of this scatterplot and observe that, indeed, the orange zones in Figure 5.5 correspond to the weakest zones of the PCA embedding, shown in Figure 5.6.

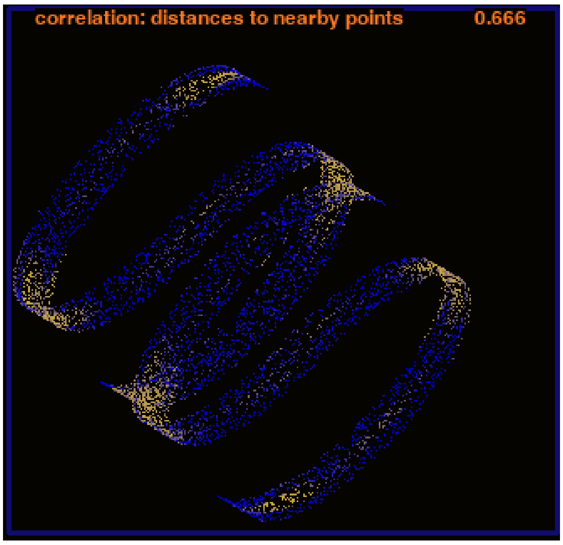


Figure 5.6: *correlation: distances to nearby points* scatterplot for the PCA embedding of *double-S* in the absolute QA screen.

5.2.2 Evaluation based on the ranking

This view evaluates the embeddings in terms of preservation of rank, either on a small scale or on a more global scale. Four plots are shown:

- The $R_{nx}(K)$ plot, with its AUC. This shows the same information as in the main screen. The selected embedding is highlighted with a brighter colour, this can be useful when dealing with many similar-looking curves.

- A local QA scatterplot titled "*preservation of ranking with anchor points*". For each point, considering the set of anchor points A of size N_A, a raw score for the i^{th} point is computed as:

$$s_i^{raw} = \sum_{a \in A} exp\left(\frac{-abs(r_{ia}^{HD} - r_{ia}^{LD})}{0.1N_A} \right),$$

  with r_{ia}^{HD} and r_{ia}^{LD} the rank of the anchor point a among the other anchors with relation to the i^{th} point, in HD and LD respectively. A perfect preservation of the ranking of the anchor points brings a raw score equal to N_A. The value 0.1 in the denominator is chosen arbitrarily, this value is considered by this author as a good compromise between allowing small differences in ranks and punishing large differences.
  To transform the raw score to a score $s_i \in [0, 1]$ which measures the improvement over randomly generated coordinates, s_i^{raw} is transformed with: $s_i = max\Big[0,\ s_i^{raw} - \hat{E}[\mathfrak{s}^{raw}]\Big]/(N_A - \hat{E}[\mathfrak{s}^{raw}])$, with $\hat{E}[\mathfrak{s}^{raw}]$ the estimated expected raw score for points taken from a randomly generated embedding, the estimation is done by using a simulation.
  This scatterplot measures how well each point is placed with regards to the global structure of the data.

- A local QA scatterplot titled "*preservation of furthest neighbours*". For each point, this scatterplot shows how well the K-furthest points are preserved between the HD and LD space. The focus in DR is often on the nearest neighbours, but having an embedding which keeps dissimilar points well apart can be a way to reduce the probability that the user underestimates the dissimilarity between two clusters of points.
  A raw score is first computed as $s_i^{raw} = \Big(|F_i^{HD} \cap F_i^{LD}|/N\Big)$, with F_i^{HD} and F_i^{LD} the sets of the K-furthest neighbours of the i^{th} point, with K equal to twenty percent of the number of observations N. The value of twenty percent is chosen arbitrarily.
  The raw score is then transformed to reflect an improvement over randomly generated embeddings with the formula:
  $s_i = max\Big[0,\ \Big(s_i^{raw} - E[\mathfrak{s}^{raw}]\Big)\Big]/\Big(0.2 - E[\mathfrak{s}^{raw}]\Big)$, with $E[\mathfrak{s}^{raw}] = 0.2 \times 0.2$ the expected raw score per point for a randomly generated embedding, 0.2 corresponds to the perfect raw score.

- A local QA scatterplot titled "*local tearing of the manifold*". This shows how well the KNN-graph is preserved between the HD and LD space, with

$K = 0.03N$. The value of 3% is chosen as it is generally small enough to capture the local properties of the manifold, while still representing a big enough number of points to enable a wide range of values in the score. The score is computed in the same way as for the furthest neighbours, but this time using the sets of 3%-nearest neighbours.

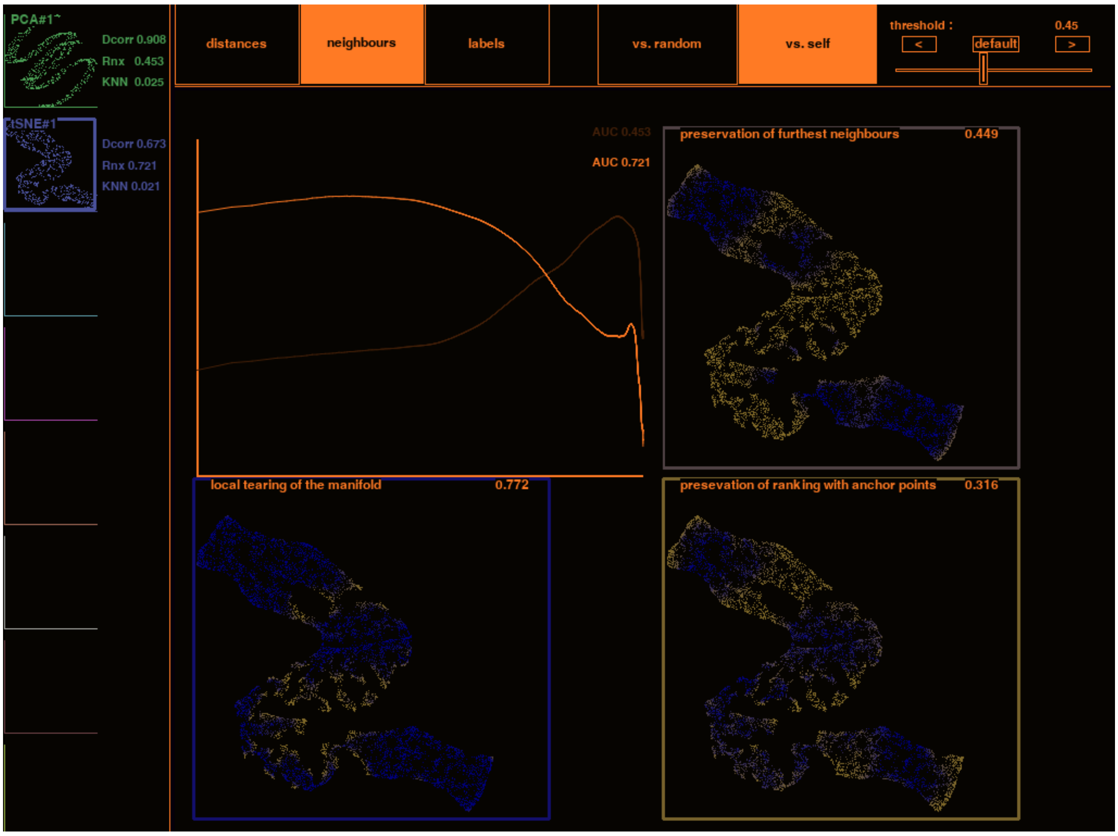


Figure 5.7: *neighbours* view of the absolute QA screen, looking at a t-SNE embedding of *double-S*.

Figure 5.7 shows the *neighbours* view of the absolute QA screen, with the t-SNE embedding of *double-S* selected. According to the three scatterplots, the middle part of the embedding is the zone posing most of the difficulty to t-SNE. This zone corresponds to the intersection of both manifolds, forming a closed loop. The presence of a loop might explain why the quality is lower in this region, as embedding loops into smaller dimensions is often difficult [25].

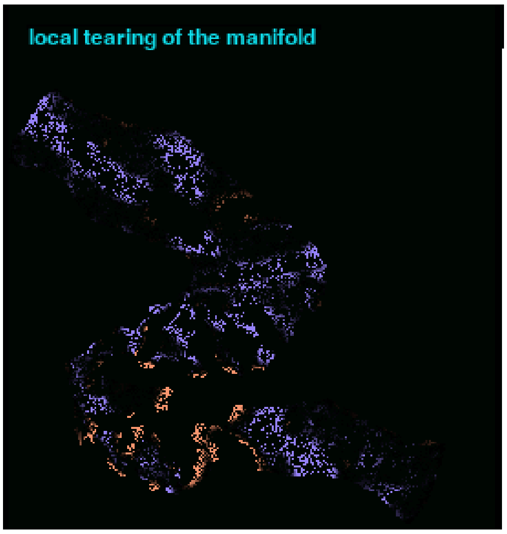


Figure 5.8: Comparison of the manifold tearing between a t-SNE of perplexity 94 (lavender) and another of perplexity 30 (orange), used on a dataset of size 10^4.

Figure 5.8 shows the *local tearing of the manifold* scatterplot in the relative QA screen, the lavender colour corresponds to the t-SNE embedding of *double-S* shown previously, it has a perplexity of 94. The orange colour is associated to another t-SNE embedding with a perplexity of 30. As a reminder, this dataset has a size $N = 10^4$.
The embedding with the higher perplexity shows an overall better preservation of the manifold, this illustrates the importance of adapting the perplexity to the dataset size when using t-SNE. The usual implementations of t-SNE have default perplexities around 40, however, when the dataset size grows, it is often beneficial to adapt the perplexity.

5.2.3 Evaluation with regards to the labels

The third view available with the QA screens does QA by using the labels. Four elements are shown in this view:

- The KNN-gain curve along with its AUC. This shows the improvement of the LD embedding over the HD data in terms of classification accuracy or regression error.
- A local QA scatterplot titled "*local uncertainty conservation*". This scatterplot measures how well the local uncertainty of the labels is preserved.
  To measure the local uncertainty on a scale K around the i^{th} point in a classification dataset, the probability for each label is computed by looking

at the labels of the K-nearest neighbours of the i^{th} point, this is done in LD and in HD. Let $\mathcal{H}_i$ and H_i be the entropies of these local label probabilities around the i^{th} point in HD and in LD respectively. A score reflecting their preservation is computed with: $s_i^{raw} = \left(2\mathcal{H}_i H_i\right) / \left(\mathcal{H}_i^2 + H_i^2 + \epsilon\right)$.
Other measures of similarity between the entropies were considered and tested, the formula detailed above gave the most readable results. It can be pointed out that a similarity computed from a divergence between the label probability distributions would not be adequate in this scatterplot, as the objective is to measure the preservation of uncertainty, not the preservation of the distribution.
For regression datasets, the standard deviation of the K-dimensional vector containing the values of the target for the K-nearest neighbours of the i^{th} point is used instead of the entropies.

In order to reduce the bias resulting from an arbitrary choice of K, this process is computed on multiple scales, with K starting at 3 and doubling until it becomes greater than 10% of the dataset size. The average score is computed and scaled to represent a measure of improvement over randomly placed points, as in the scatterplots of the *neighbours* view.

- A local QA scatterplot titled "*small scale label agreement*". This scatterplot measures how well the local probabilities of the labels is preserved.
  The label probabilities around the i^{th} point are computed as in the *local uncertainty conservation* scatterplot. The label agreement score is computed by subtracting the Jensen–Shannon divergence between the probabilities in HD and in LD to the value 1.
  Two scales are used in this scatterplot : $K = 1$ and $K = 3$, their scores are averaged into a final score.

- A local QA scatterplot titled "*multi-scale label agreement*". This scatterplot uses the same process, but this time with scales starting at 3 and doubling until reaching 50% of the dataset size. The scale-wise scores are averaged to obtain a multi-scale score.

Figure 5.9 shows the *labels* view in the absolute QA screen, showing the t-SNE embedding of *blob* introduced in the beginning of this section. The label uncertainty conservation is low inside the clusters. Since there is very little variety in the labels within each cluster in LD, this observation indicates that the classes aren't as well separated in the HD space. This is in line with the observation displayed in the KNN-gain curve, which is mostly positive.

As explained in the first part of this paper, having a good class separation is neither an inherently good or bad property of the embedding. It is merely an indication that the best way that the DR algorithm found to optimise its cost function brought a good class separation. This could indicate that the true nature of the data is best described in a way that also happens to separate the labels.

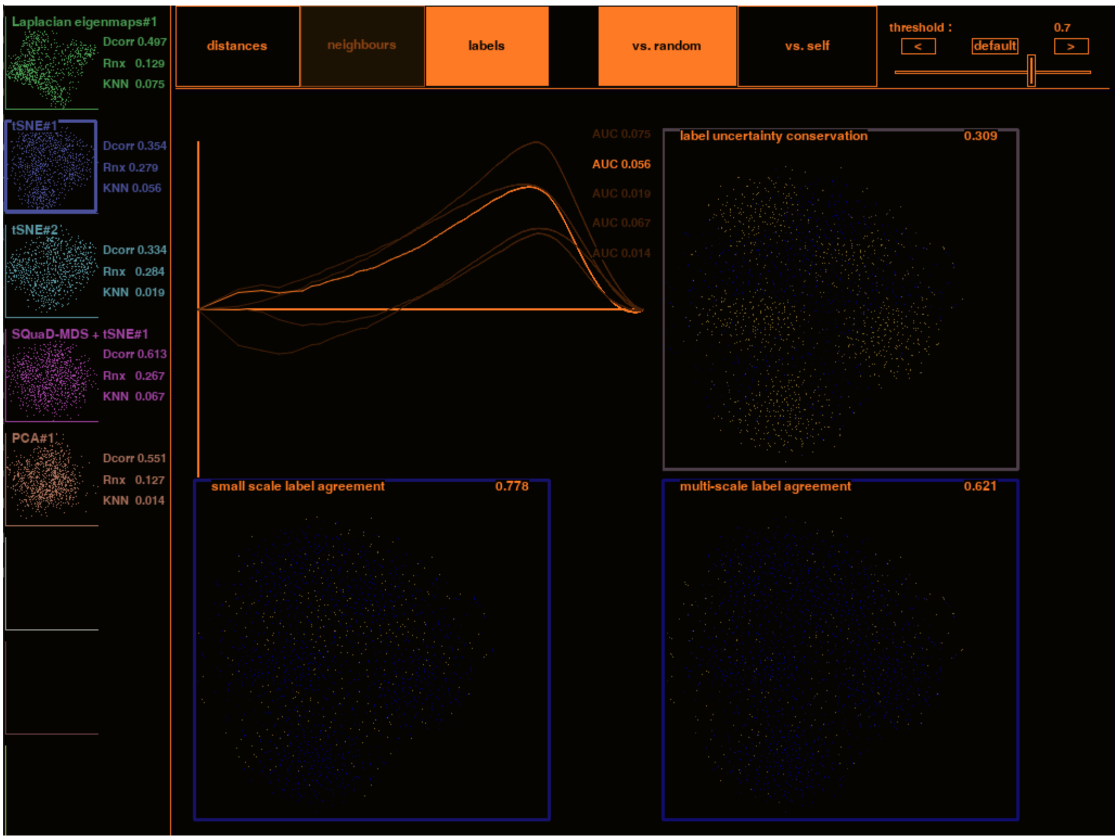


Figure 5.9: *labels* view of the absolute QA screen on a t-SNE embedding of *blob*.

Chapter 6

Conclusion

The software offers an environment where both beginners and experts in the field of DR can explore data and develop their intuition in the domain. Because visualisation has many pitfalls which can lead the user to erroneous conclusions, one of the major selling points of this program is its an extensive collection of tools for QA.

This work also introduced a novel approach to MDS with time and space complexities of $\mathcal{O}(N)$, allowing the usage of MDS to evaluate the large-scale structures of the data without having to rely on sub-samplings. This method also brings the possibility to produce embeddings that preserve both similarities and distances, an approach to DR that is still mostly unexplored.

Another contribution to the domain of DR brought by this master's thesis comes in the form of a study of the impact of randomness on *fast-multiscale neighbor embedding*.

Further improvements could target different aspects of *the software*. For instance, a possible improvement would be to produce a second version of the software with less focus on interactivity and speed, but more focus on memory efficiency. Such a software could go hand-in-hand with the current product, as the scientist might do a first exploration on a sample of the data using the interactive software, and then switch to the less interactive environment to produce the final embedding on the full dataset.

Being able to interpret the structures in an embedding is an important aspect of DR. Some methods can explain the organisation of the embeddings, by showing which HD variables or combination of variables have most influence on the local positions of the points [26]. The lack of such methods in *the software* is an important drawback, however, the flexibility of the custom GUI API should make their

integration to the code-base an easy process.
The integration of graph embeddings is also in the list of possible improvements, as these methods have a wide range of applications and are the target of a lot of researchers.

The SQuaD-MDS method is still new, it could be the target of many improvements. For instance, a version allowing the preservation of weighted distances could be envisioned, this would allow the user to guide the optimisation towards a certain property. Another exciting perspective is to further study the hybrid version, which, in its current state, produces promising embeddings by adding *t*-SNE gradients to the quartet-based distance preserving gradients. A different approach on the way the gradients are mixed, or a different optimiser strategy might further improve the results.

Part III

Annex

Annex A : Recurring abbreviations

HD : High-dimensional or higher-dimensional

LD : Low-dimensional or lower-dimensional

QA : Quality assessment

PCA : Principal component analysis

MDS : Multidimensional scaling

SQuaD-MDS : "Stochastic Quartet Descent"-MDS

NE : Neighbor embeddings

SNE / *t*-SNE : (t-distributed) Stochastic neighbor embedding

ms-*t*-SNE : Multi-scale *t*-SNE

f-ms-*t*-SNE : Fast multi-scale *t*-SNE

CAt-SNE : Class-Aware *t*-SNE

FIt-SNE : Fast interpolation-based *t*-SNE

UMAP : Uniform Manifold Approximation and Projection

KL divergence : Kullback-Leibler divergence

AUC : area under curve

PCC : Pearson correlation coefficient

GUI : graphical user interface

API : application programming interface

KNN : K-nearest neighbours

Annex B : Stochastic quartet approach for fast multidimensional scaling

Stochastic quartet approach for fast multidimensional scaling

Pierre Lambert[1], Cyril de Bodt[1], Michel Verleysen[1], John A. Lee[1,2]

1- UCLouvain.be - ICTEAM/ELEN
Place du Levant 3 L5.03.02, 1348 Louvain-la-Neuve - Belgium

2- UCLouvain.be - IREC/MIRO
Avenue Hippocrate 55 B1.54.07, 1200 Brussels - Belgium

Abstract. Multidimensional scaling is a statistical process that aims to embed high-dimensional data into a lower-dimensional, more manageable space. Common MDS algorithms tend to have some limitations when facing large data sets due to their high time and spatial complexities. This paper attempts to tackle the problem by using a stochastic approach to MDS which uses gradient descent to optimise a loss function defined on randomly designated quartets of points. This method mitigates the quadratic memory usage by computing distances on the fly, and has iterations in $\mathcal{O}(N)$ time complexity, with N samples. Experiments show that the proposed method provides competitive results in reasonable time. Public codes are available at `https://github.com/PierreLambert3/SQuaD-MDS.git`.

1 Multidimensional scaling and its limitations

Dimensionality reduction (DR) is the process of mapping high-dimensional (HD) observations into a lower-dimensional (LD) space such that the LD embedding is a faithful representation of the HD data. The main DR uses are in machine learning, to curb the curse of dimensionality, and in visualisation. Mapped data can reveal structures that would lay hidden from the human perception if left in HD. Typically, some information is lost by the DR and, therefore, each DR method has a take on what kind of information should be preserved and what can be lost. Used frequently in visualisation, *t*-SNE [1] aims at retaining the neighbourhood of each point according to a distance metric and a perplexity, which reflects the size of the neighbourhood to preserve. While *t*-SNE excels at retaining local structures, sufficiently remote points tend to be considered equally distant by the algorithm and, therefore, the larger-scale structures can be distorted. Such distortions can lead to erroneous conclusions by the human user, who might overestimate the dissimilarity between two clusters that are distant in the LD embedding. For this reason, using multiple DR paradigms in conjunction is a good practice in visualisation: another embedding that preserves distances instead of neighbourhoods would have prevented this erroneous conclusion.

This paper considers metric multidimensional scaling (MDS): a DR technique that produces a LD embedding such that the pairwise distances in LD reflect those in HD. MDS minimises a cost function which, in its simplest form, is the sum of the squared differences between distances in HD and the Euclidean distances in LD. A common strategy to optimize this cost function is based on the SMACOF algorithm [2], involving iterations of $\mathcal{O}(N^2)$ time complexity with N samples. This quickly becomes problematic when N grows. SMACOF also requires the full matrix of pairwise HD distances, consuming $\mathcal{O}(N^2)$ memory.

Some alternatives with lower time or memory complexities exist, such as the iterative spring-based model [3], where each point is subject to forces computed according to two constant-sized sets of points. This enables iterations with $\mathcal{O}(N)$ time complexity. However, the sets are subject to iterative refinements, making the total number of required iterations increase with the size of the data set. A different divide-and-conquer approach [4] applies MDS on sub-matrices of the HD distance matrix and then stitches together the results. Another take on the problem is proposed in landmark MDS [5], which first embeds a small subset of the data in LD and then arranges the rest of the points with respect to these landmarks. Alternatively, the effects of high computational complexities can be reduced by focusing on certain parts of the LD embedding thanks to the input of a user [6].

While these accelerated MDS methods provide interesting results, they often involve multi-step approaches that complicate the algorithm and tend to make it monolithic in the sense that the optimisation process becomes an intractable block once the optimisation started. This paper proposes an algorithm that has $\mathcal{O}(N)$ time complexity and does not require a full distance matrix, making it more memory-efficient. In addition to showing that the algorithm can yield good distance correlation between the HD and LD spaces, this paper shows that by using a simple gradient descent optimisation, the algorithm is open for hybrid approaches, where gradients of different types are mixed.

Section 2 presents the algorithm, while Section 3 is dedicated to an empirical assessment of its quality and speed performance. Section 4 sketches some conclusions and perspectives.

2 Proposed method

The proposed algorithm performs MDS by using gradient descent on a cost function defined on quartets of points. Before defining the gradients, this section starts by explaining how to articulate a distance scaling problem around the use of quartets.

2.1 Quartet-based optimisation

Let us consider a quartet formed by four points that has a HD distance matrix δ and a LD Euclidean distance matrix d. If these points have a LD embedding that perfectly preserves their HD distances, then the relative distance between the i^{th} and the j^{th} points in LD, d^{r}_{ij}, also perfectly matches their relative distance in HD, δ^{r}_{ij}. These relative distances are defined as

$$d^{\text{r}}_{ij} = d_{ij} \Big/ \Big(\sum_{a=1}^{3} \sum_{b=a+1}^{4} d_{ab} \Big) \quad \text{and} \quad \delta^{\text{r}}_{ij} = \delta_{ij} \Big/ \Big(\sum_{a=1}^{3} \sum_{b=a+1}^{4} \delta_{ab} \Big) \ .$$

A cost function for the preservation of relative distances within this quartet of points can be defined:

$$C_{\text{quartet}} = \sum\nolimits_{a=1}^{3} \sum\nolimits_{b=a+1}^{4} (\delta^{\text{r}}_{ab} - d^{\text{r}}_{ab})^2 \quad . \tag{1}$$

To apply this quartet approach to the whole data set, the algorithm iteratively and randomly splits the observations in groups of 4, and the gradient for each point is computed according to (1). This use of quartets should not be confused with triplet methods such as stochastic triplet embedding [7], as they work with data of different natures. Triplet embeddings use partial similarity rankings of the form "A is more similar to B than to C" often resulting from human judgement, whereas the proposed fast MDS method uses HD observations.

Using quartets of points gives constant-time gradients and has a natural analogy to triangulation (1 point versus 3 others) when applied to a target dimension of 2. Quartets are also the reason why the full HD distance matrix isn't needed : only 6 HD distances are used at any instant. It is noteworthy that using relative distances precludes any preservation of scale: the LD distances tend to correlate to those in HD, but the scale in LD will stay close to the scale at which the embedding is initialised.

2.2 The gradients

Considering a quartet of points with a LD Euclidean distance matrix d and a HD distance matrix δ, I is the 4x4 identity matrix and x_q is the value taken for some LD dimension by the q^{th} point in the quartet. For the sake of concision, only the gradients for one term in the sum of (1) is shown here, as the other terms in the sum share similar forms:

$$\frac{\partial(\delta^{\text{r}}_{ij} - d^{\text{r}}_{ij})^2}{\partial x_q} \quad = \quad \frac{2}{S}(d^{\text{r}}_{ij} - \delta^{\text{r}}_{ij})\left(\frac{I_{qi}(x_q - x_j) + I_{qj}(x_q - x_i)}{d_{ij}} - d^{\text{r}}_{ij} \sum_{b \in \{1,..,4\} \setminus q} \frac{x_q - x_b}{d_{qb}}\right)$$

with $S = \sum_{i<j} d_{ij}$. The motivation for using relative distances is that each of the six LD distances in the quartet contributes to the update for each LD point, as opposed to direct distance preservation where only three distances would contribute. This results in the behaviour illustrated in Fig. 1, where each point moves to minimise the error on the quartet as a whole.

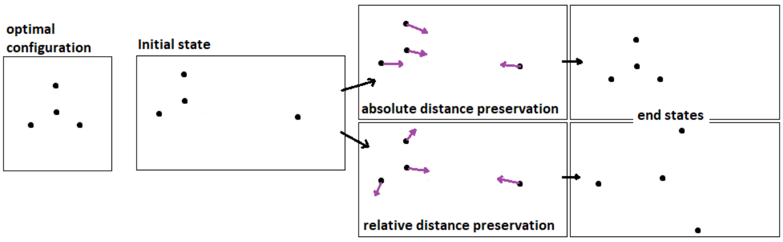

optimal configuration
Initial state
absolute distance preservation
end states
relative distance preservation

Fig. 1: Difference in behaviour when using relative distances and raw distances.

3 Experiments and discussion

To assess the performance of the proposed algorithm in terms of speed and distance preservation, it is compared to the SMACOF-based MDS [2] on various data sets. Section 3.1 details our implementation, while Section 3.2 describes the results.

3.1 Implementation

Our implementation uses a gradient descent optimizer in its simplest form: the gradients are scaled by a learning rate and then directly subtracted to the positions of the LD points. During the first part of the optimisation, the HD distances are squared before being transformed to relative distances. This helps the optimisation by exaggerating the differences between the distances within a quartet. Because of the stochastic nature of the optimisation, the updates on the position of the points tend to be erratic; this isn't a problem during the first part of the optimisation, but it renders the fine-tuning of the final positions difficult. To counter this behaviour, it is important that the learning rate decays to small values. We determined experimentally that, for embeddings initialised with a standard deviation around 10, a learning rate starting in the hundreds that decays to 10^{-3} is adequate. Having it decay to greater values tends to reduce the quality of the results. All the following embeddings using the proposed fast MDS are given the same hyper-parameters. They result from a single run of 1000 iterations, with a target dimension of 2 and a PCA initialisation scaled to have a standard deviation of 10. Non-PCA initialisations can produce similar results, but they typically require a larger number of iterations before convergence. The code can be found at: `https://github.com/PierreLambert3/SQuaD-MDS.git`.

3.2 Empirical assessment

Seven data sets are featured in this paper, with size N and dimensionality M. Abalone: $(N, M) = (4177, 8)$; coil20: $(N, M) = (1440, 1024)$; a subset of MNIST digits: $(N, M) = (3400, 784)$; a subset of MNIST fashion: $(N, M) = (4000, 784)$; satellite: $(N, M) = (4434, 36)$; a subset of Gisette: $(N, M) = (3700, 4900)$; RNAseq: $(N, M) = (10^4, 50)$. Most sets are available from the UCI machine learning repository. RNAseq contains single-cell data used in [8]; the same 50 principal components from the selected features are used, as in [8].

Table 1 shows the Pearson correlation between the pairwise distances in HD and LD for various models on the data sets. The 1^{st} row shows results obtained with PCA. The 2^{nd} row shows the best result of 10 different initialisations for the SMACOF MDS algorithm as implemented by scikit-learn. The 3^{rd} row is also the result of a MDS using SMACOF but this time initialised with PCA. The 4^{th} row is our fast MDS algorithm in its default version, and the last row is an experiment where the HD distances for the quartets are first transformed non-linearly before being used by the cost function. This transformation is of the form $\delta'_{ij} = 1 - \exp(-(\delta_{ij} - \delta_{\min})/(2\sigma))$ with σ being the standard deviation of the HD distances in the quartet and $\delta_{\min}$ the smallest HD distance in the quartet. This transformed distance is then divided by the sum of the transformed distances to become relative. The purpose of this transformation is to increase the contrast between the HD distances within the quartet in the hope of reducing the effect of the concentration of norms [9] when the input dimension is high. In terms of correlation of distances, Table 1 shows that the proposed method can provide competitive results, and that its results tend to be particularly good when the data dimension is high.

	abalone	satellite	RNAseq	digits	fashion	coil20	gisette
PCA	0.82	0.98	0.93	0.50	0.69	0.76	0.45
10x MDS	0.82	0.93	0.93	0.63	0.82	0.80	0.84
PCA-MDS	0.99	0.99	0.90	0.66	0.89	0.84	0.67
fastMDS	0.98	0.97	0.95	0.73	0.94	0.86	0.85
fastMDS-rbf	0.98	0.96	0.96	0.75	0.92	0.87	0.89

Table 1: Correlation of the distances in LD and HD; the data sets are of increasing dimensionality from left to right.

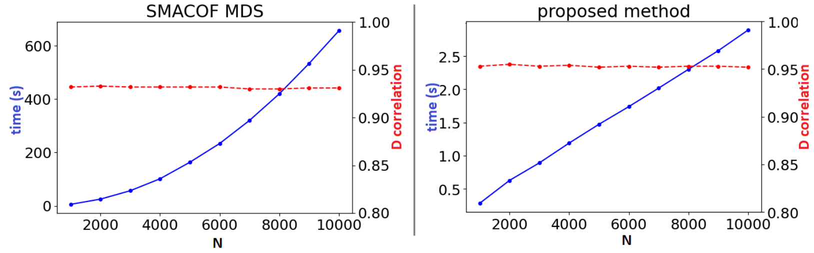


Fig. 2: Evolution of computation time and distance correlation for samples of RNAseq of size varying between 10^3 and 10^4 with steps of 10^3.

Figure 2 shows that the fast MDS algorithm runs in linear time with N, as opposed to the quadratic scaling of SMACOF MDS; the distance correlation is stable with different sizes N, which can be surprising considering that the proportion of HD distances that are used during the optimisation decreases when N increases. Actually, the number of HD distances used by the proposed algorithm is $\mathit{number\_of\_iterations} * \mathit{6\_distances\_per\_quartet} * \mathit{number\_of\_quartets}$, in this case $10^3 * 6 * \frac{N}{4}$, but the total number of HD distances is $\frac{N*(N-1)}{2}$. This supports the intuition that when N is high, using only a subset of the distances can be sufficient for a good embedding, which is a principle of landmark-based approaches.

Figure 3 shows some 2-D embeddings of a subset of RNAseq; 'D correlation' is the Pearson correlation between HD and LD distances and 'R_{NX} AUC' is the AUC of the $R_{\text{NX}}(K)$ curves [10], which reaches 1 when the multi-scale neighbourhoods of the data are well preserved. Since the proposed method relies on a simple gradient descent, it is possible to combine it with another gradient-based algorithm. In the 3$^{\text{rd}}$ scatter plot, we added the quartet based distance gradients to those of t-SNE to obtain a hybrid embedding. The figure shows that

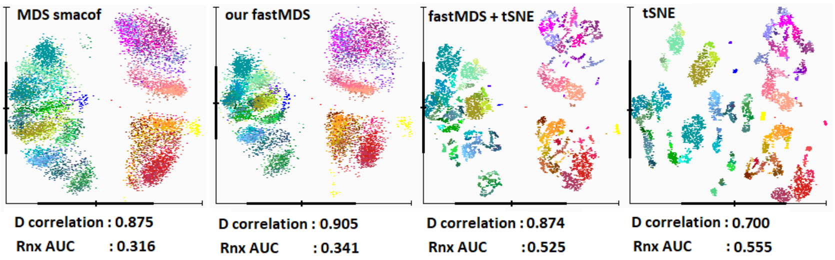


Fig. 3: 2-D embeddings of a subset of RNAseq of size 9300.

this hybrid approach can preserve neighbourhoods quite well while keeping the distances meaningful.

4 Conclusion and perspectives

Experiments show that the proposed fast MDS method produces competitive results while keeping the algorithm simple and the computational complexities low. We may have only scratched the surface of the possibilities that a stochastic quartet-based approach can offer; we can imagine an extension that would enable the use of weighted distances, the use of different quartet-based metrics, or an adaptation for ordinal MDS. Another perspective is to further study the use of quartet-based distance gradients in a hybrid approach to DR, by mixing the distance preservation and the neighbourhood preservation paradigms.

References

[1] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[2] J. de Leeuw and P. Mair. Multidimensional scaling using majorization: Smacof in r. *Journal of Statistical Software, Articles*, 31(3):1–30, 2009.

[3] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. *Proceedings of the 7th conference on Visualization*, pages 127 – 131, 12 1996.

[4] Tzeng, Henry Lu, and Wen-Hsiung Li. Multidimensional scaling for large genomic data sets. *BMC bioinformatics*, 9:179, 02 2008.

[5] V. Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. *Technology*, 01 2004.

[6] M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *IEEE Symposium on Information Visualization*, pages 57–64, 2004.

[7] L.J.P. van der Maaten and K.Q. Weinberger. Stochastic triplet embedding. *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, 2012.

[8] D. Kobak and P. Berens. The art of using t-sne for single-cell transcriptomics. *Nat Commun 10, 5416*, 2019.

[9] D. Francois, V. Wertz, and M. Verleysen. The concentration of fractional distances. *IEEE Trans. Knowl. Data Eng.*, 19(7):873–886, 2007.

[10] J. A. Lee, D. H. Peluffo-Ordóñez, and M. Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261, 2015.

Annex C : Impact of data subsamplings in Fast Multi-Scale Neighbor Embedding

Impact of data subsamplings in Fast Multi-Scale Neighbor Embedding

Pierre Lambert[1], John A. Lee[1,2], Michel Verleysen[1], Cyril de Bodt[1]

1- UCLouvain.be - ICTEAM/ELEN
Place du Levant 3 L5.03.02, 1348 Louvain-la-Neuve - Belgium

2- UCLouvain.be - IREC/MIRO
Avenue Hippocrate 55 B1.54.07, 1200 Brussels - Belgium

Abstract. Fast multi-scale neighbor embedding (f-ms-NE) is an algorithm that maps high-dimensional data to a low-dimensional space by preserving the multi-scale data neighborhoods. To lower its time complexity, f-ms-NE uses random subsamplings to estimate the data properties at multiple scales. To improve this estimation and study the f-ms-NE sensitivity to randomness, this paper generalizes the f-ms-NE cost function by averaging several subsamplings. Experiments reveal that this can slightly improve DR quality while maintaining reasonable computation times. Codes are available at `https://github.com/cdebodt/Fast_Multi-scale_NE`.

1 Introduction

Dimension reduction (DR) maps high-dimensional (HD) data to a low-dimensional (LD) space such that the LD embedding faithfully represents the HD data. Some information is typically lost in the DR process and, therefore, the faithfulness of a projection is considered with respect to some criterion. The main uses of DR are in machine learning, to curb the curse of dimensionality, and in visualisation.

When HD dimension is high, the Euclidean distances between points tend to concentrate towards similar values [1]. Because of this phenomenon, using distance preservation as a direct criterion for DR can become problematic. Neighbor embedding (NE) techniques such as stochastic neighbor embedding (SNE) [2] alleviate the effects of distance concentration by defining neighbor probability distributions in both spaces to embed points in LD [3]. These methods are frequently used in visualization as they tend to produce embeddings that show well separated, readable clusters and nicely preserve local neighborhoods [4].

Originally, NE algorithms required the user to specify a scale for the neighborhood preservation. By capturing the structures on a single scale, embeddings produced by these methods can lead the user to erroneous interpretations. More recently, multi-scale NE approaches were introduced by combining neighborhood probabilities tuned across various scales [5, 6]. Multi-scale approaches often yield better results by preserving the distant structures, while still ensuring good local neighborhood preservation. However, multi-scale NE methods have a time complexity of $\mathcal{O}(N^2 \log N)$ with N data points, restricting their use to data sets of moderate size. A fast version of multi-scale NE (f-ms-NE) [7] has a time complexity of $\mathcal{O}(N \log^2 N)$, with a slight decrease in neighbor preservation as a trade-off. To achieve such an acceleration, f-ms-NE relies on random data subsamplings to capture both the local and global structures in the HD data. However, the stability of the f-ms-NE performances with respect to the number of these subsamplings has not been characterized previously.

This paper hence generalizes f-ms-NE by defining cost functions averaged over multiple random subsamplings, enabling to study the effect of their number on the LD embeddings. In addition to aiming at increasing DR quality while preserving manageable computation times, this work further assesses the sensitivity of f-ms-NE to randomness. Public codes are freely available at `https://github.com/cdebodt/Fast_Multi-scale_NE`.

This paper is structured as follows: Section 2 summarizes NE algorithms. Section 3 explains how f-ms-NE is adapted to consider multiple random subsamplings. Section 4 details experimental results by comparing the performances of the f-ms-NE algorithm across an increasing number of subsamplings, in both terms of DR quality and speed. Final conclusions are drawn in Section 5.

2 Neighbor embedding algorithms

The f-ms-NE algorithm accelerates multi-scale NE, which is based on single-scale NE. Section 2.1 first sketches single-scale NE, Section 2.2 then describes its multi-scale extension, and Section 2.3 finally details the f-ms-NE acceleration.

2.1 Single-scale neighbor embedding

Let $\mathbf{\Xi} = [\xi_i]_{i=1}^N$ denote a set of N points in a HD space with M features. Let $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N$ represent these data in a P-dimensional LD space, $P \leq M$. The HD and LD distances between the i^{th} and j^{th} points are δ_{ij} and d_{ij}, respectively, for $i \in \mathcal{I} = \{1, ..., N\}$ and $j \in \mathcal{I} \setminus \{i\}$. SNE [2] aims at preserving pairwise similarities from HD to LD, which are respectively defined as

$$\sigma_{ij} = \frac{\exp(-\pi_i \delta_{ij}^2/2)}{\sum_{k \in \mathcal{I}\setminus\{i\}} \exp(-\pi_i \delta_{ik}^2/2)},\ s_{ij} = \frac{\exp(-d_{ij}^2/2)}{\sum_{k \in \mathcal{I}\setminus\{i\}} \exp(-d_{ik}^2/2)},\ \sigma_{ii} = s_{ii} = 0\ . \quad (1)$$

Precision π_i is adapted to the local density in order to tune the HD similarities to a target scale. The scale is provided by the user in the form of a perplexity K for the distribution $[\sigma_{ij}; j \in \mathcal{I} \setminus \{i\}]$ such that $\log K = -\sum_{j \in \mathcal{I}\setminus\{i\}} \sigma_{ij} \log \sigma_{ij}$.

SNE interprets these normalized similarities as neighborhood probabilities around each point, in order to produce a LD embedding that minimises the sum of Kullback-Leibler (KL) divergences $C_{SNE} = \sum_{i \in \mathcal{I}, j \in \mathcal{I}\setminus\{i\}} \sigma_{ij} \log(\sigma_{ij}/s_{ij})$.

The t-SNE extension [4] symmetrizes the similarities and considers a Student t function with one degree of freedom in LD space, to cope with 'crowding' problems in LD. The HD similarities τ_{ij} and LD ones t_{ij} are now defined as

$$\tau_{ij} = (\sigma_{ij} + \sigma_{ji})/(2N),\ t_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \in \mathcal{I}, l \in \mathcal{I}\setminus\{k\}} (1 + d_{kl}^2)^{-1}},\ \tau_{ii} = t_{ii} = 0\ . \quad (2)$$

2.2 Multi-scale neighbor embedding

The multi-scale SNE method [5] combines SNE similarities computed with exponentially increasing perplexities. Single-scale similarities are indexed by a scale counter h for $i \in \mathcal{I}$ and $j \in \mathcal{I} \setminus \{i\}$:

$$\sigma_{ijh} = \frac{\exp(-\pi_{ih}\delta_{ij}^2/2)}{\sum_{k\in\mathcal{I}\setminus\{i\}} \exp(-\pi_{ih}\delta_{ik}^2/2)},\ s_{ijh} = \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k\in\mathcal{I}\setminus\{i\}} \exp(-p_{ih}d_{ik}^2/2)},\ \sigma_{iih} = s_{iih} = 0\ .$$

HD precisions π_{ih} are set using perplexities growing as $K_h = 2^{h-1}K_1$, with a small base perplexity $K_1 = 2$ and $1 \leq h \leq H = \lfloor \log_2(N/K_1) \rceil$, where $\lfloor \cdot \rceil$ denotes rounding. LD precisions $p_{ih} = p_h = K_h^{-2/P}$ follow the exponential growth of the HD precisions. Multi-scale similarities average single-scale ones as

$$\sigma_{ij} = H^{-1}\sum\nolimits_{h=1}^{H} \sigma_{ijh},\ \ s_{ij} = H^{-1}\sum\nolimits_{h=1}^{H} s_{ijh}\ \ . \tag{3}$$

The authors recommend using L-BFGS [8] to optimise the cost function.

2.3 Fast multi-scale neighbor embedding

Single-scale and multi-scale NE methods have time complexities of $\mathcal{O}(N^2)$ and $\mathcal{O}(N^2 \log N)$, respectively. The typical smallness of perplexity K with respect to N enables single-scale methods to consider that sufficiently distant points have a null HD similarity. Sparse HD similarities can be computed by finding the sets of nearest neighbors of each point using vantage-point trees [9]. This reduces their computation time to $\mathcal{O}(KN \log N)$. On the LD side, a Barnes-Hut (BH) algorithm [10, 11] approximates efficiently the similarities by relying on tree structures to compound far-away points, dispensing with computing all pairwise interactions, reducing the time complexity to $\mathcal{O}(N \log N)$. The BH method uses a threshold parameter $\theta \in [0, 1]$ that determines whether to use the approximation or not, with higher values meaning rougher but faster estimations.

The largest scale used to compute multi-scale similarities has a perplexity K_H in the $\mathcal{O}(N)$ range. At such a scale, the above sparse HD similarity approximation would require neighbor sets of $\mathcal{O}(N)$ size, defeating the purpose of accelerating the computations. For this reason, reducing the time complexity of multi-scale NE requires a different approach for the HD side. In [7], the authors tackle this problem by defining small multi-scale neighbor sets $\widetilde{\mathcal{I}}_i$ for $i \in \mathcal{I}$.

To compute $\widetilde{\mathcal{I}}_i$, the authors define a hierarchy of subsampled HD data sets $\{\Xi_h\}_{h=1}^{H}$, where Ξ_h is a random sample of Ξ with $\lfloor 2^{1-h}N \rceil$ elements drawn without replacement. A vantage-point tree is created on Ξ_h for each scale $h \in \{1, ..., H\}$; the trees are generated in $\mathcal{O}(N \log N)$ time. For each scale, the corresponding tree is used to compute the neighborhood $\widetilde{\mathcal{I}}_{ih}$ within the sample Ξ_h for $i \in \{1, ..., N\}$; this takes a total of $\mathcal{O}(N \log^2 N)$ time. By having samples $\{\Xi_h\}_{h=1}^{H}$ that decrease in size when the scale grows, the accounted neighbors $\widetilde{\mathcal{I}}_{ih}$ are more and more dispersed in the data cloud, hence capturing larger scale properties. The multi-scale neighbor sets are then defined as $\widetilde{\mathcal{I}}_i := \cup_{h=1}^{H} \widetilde{\mathcal{I}}_{ih}$.

Like the non-accelerated version, f-ms-NE uses multi-scale similarities averaged over sparse single-scale similarities,

$$\widetilde{\sigma}_{ij} = H^{-1}\sum_{h=1}^{H}\widetilde{\sigma}_{ijh}, \quad \text{with} \quad \widetilde{\sigma}_{ijh} = \begin{cases} \frac{\exp(-\widetilde{\pi}_{ih}\delta_{ij}^2/2)}{\sum_{k\in\widetilde{\mathcal{I}}_i}\exp(-\widetilde{\pi}_{ih}\delta_{ik}^2/2)} & \text{if } j\in\widetilde{\mathcal{I}}_i \\ 0 \text{ otherwise} \end{cases} .$$

Precision $\widetilde{\pi}_{ih}$ is fixed such that

$$K_1 = -\sum_{j\in\widetilde{\mathcal{I}}_{ih}}\widetilde{\sigma}_{ijh}^{\pi}\log\widetilde{\sigma}_{ijh}^{\pi} \text{ where } \widetilde{\sigma}_{ijh}^{\pi} = \begin{cases} \frac{\exp(-\widetilde{\pi}_{ih}\delta_{ij}^2/2)}{\sum_{k\in\widetilde{\mathcal{I}}_{ih}}\exp(-\widetilde{\pi}_{ih}\delta_{ik}^2/2)} & \text{if } j\in\widetilde{\mathcal{I}}_{ih} \\ 0 \text{ otherwise} \end{cases} .$$

In the $t-$distributed version, the authors use symmetrised HD similarities $\widetilde{\tau}_{ij} = (\widetilde{\sigma}_{ij}+\widetilde{\sigma}_{ji})/(2N)$ and optimise the cost function $\widetilde{C}_t = -\sum_{i\in\mathcal{I},j\in\mathcal{I}\backslash\{i\}}\widetilde{\tau}_{ij}\log t_{ij}$.

3 Multiple subsamplings in fast multi-scale NE

In f-ms-NE, the larger-scale structures of the HD data are captured by computing neighborhoods on samples of decreasing sizes. Therefore, f-ms-NE estimates the larger-scale data properties with some sensitivity to randomness, as an unlucky sampling could bypass some important structures within the data. We hence propose to adapt f-ms-NE by using multiple subsamplings of the HD data, aiming at reducing the impact of randomness when modeling large-scale structures.

The proposed adaptation is to perform the whole sampling process R times, to compute sets $\widetilde{\mathcal{I}}_{ir}$ for $i\in\{1,...,N\}$ and $r\in\{1,...,R\}$. The sparse similarities are computed independently for each batch and the cost function becomes

$$\widetilde{C}_t = R^{-1}\sum\nolimits_{r=1}^{R}\left(-\sum\nolimits_{i\in\mathcal{I},j\in\mathcal{I}\backslash\{i\}}\widetilde{\tau}_{ijr}\log t_{ij}\right) . \tag{4}$$

As R is set to a small value independent of N, the time complexity with regards to N remains unchanged. But as the original cost function is computed R times, a computation time increase is expected. However, each repetition being independent, a parallel implementation with R threads can easily be done.

4 Experimental results and discussion

The method is tested on seven data sets of size N and dimensionality M [12]. Anuran: (N,M) = (7195, 22); Plant: (N,M) = (9568, 4); Gestures: (N,M) = (9901, 18); Satellite: (N,M) = (6434, 36); Waveform: (N,M) = (5000, 40); Theorem: (N,M) = (6118, 51); and Musk: (N,M) = (6598, 166). On each set and for each $R\in\{1,...,5\}$, f-ms-NE with multiple subsamplings is applied 30 times using a random initialisation, and 30 times with a PCA initialisation. The t-distributed version of the algorithm is used and the target dimension P is 2 for all experiments. Each optimization consists of 30 iterations of the L-BFGS algorithm and uses a BH threshold $\theta = 0.75$, as these values tend to produce good DR quality in reasonable time for the original f-ms-NE [7].

Figure 1 shows the quality of the resulting embeddings. As in [7], their quality is measured by the area under the curve (AUC) of the relative neighborhood preservation R_{NX} [13], with a logarithmic scale for the neighborhood size. The AUC of R_{NX} reaches 1 when the preservation is perfect for all data scales.

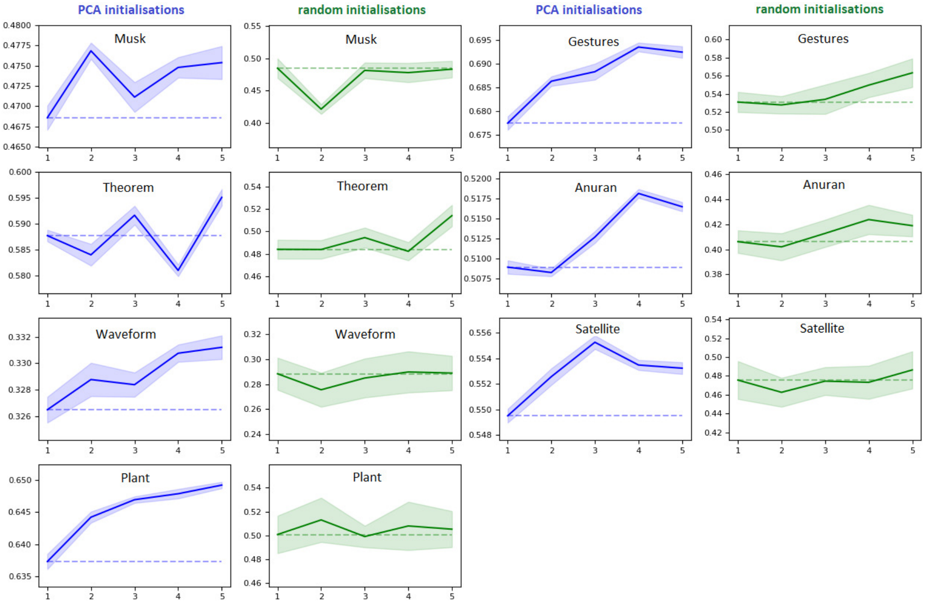


Fig. 1: Evolution of the AUC of the R_{NX} curves with the number of subsamplings. The y-axis is the AUC, and the x-axis is the number R of subsamplings. The solid lines are the mean values over 30 trials, the shaded areas correspond to one standard deviation on each side. The dashed line corresponds to the mean AUC for the original version of f-ms-NE, which uses one sampling.

We observe that subsampling multiple times has an effect on the DR quality of the embeddings that f-ms-NE yields. However, the change is often slight when compared to embeddings produced from a single sampling and, surprisingly, the change is not always positive. Using multiple subsamplings in conjunction with a PCA initialisation tends to produce a more consistent increase in quality, but the magnitude of the increase depends on the data set. The modest benefits of using multiple subsamplings can suggest that the construction of $\widetilde{\mathcal{I}}_i$ in the original f-ms-NE algorithm is already quite robust to randomness.

In terms of computation time, we obtain a consistent linear increase in time with respect to R. With a single thread implementation on the tested computer, using $R = 5$ requires approximately twice the time that the algorithm would take in its original version.

5 Conclusion

Experimental results indicate that using multiple subsamplings in f-ms-NE can provide improvements to the DR quality. The moderate nature of these improvements hints that the sampling process used by f-ms-NE enables to model the multi-scale structures of the HD data with a remarkable robustness with respect to the randomness.

To further study the influence of randomness on f-ms-NE, alternate subsampling strategies will be envisioned. For instance, using a biased sampling to increase the diversity of the sampled points in the hierarchy of subsampled HD data sets $\{\Xi_h\}_{h=1}^{H}$ is likely to improve the estimation of the global structure of the HD data.

References

[1] D. François, V. Wertz, and M. Verleysen. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering*, 19:873–886, 2007.

[2] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840, 2003.

[3] John A. Lee and Michel Verleysen. Shift-invariant similarities circumvent distance concentration in stochastic neighbor embedding and variants. *Procedia Computer Science*, 4:538–547, 2011.

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

[5] John A. Lee, Diego H. Peluffo-Ordóñez, and Michel Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261, 2015.

[6] C. de Bodt, D. Mulders, M. Verleysen, and J. A. Lee. Perplexity-free t-SNE and twice student tt-SNE. *ESANN*, pages 123–128, 2018.

[7] Cyril de Bodt, Dounia Mulders, Michel Verleysen, and John Aldo Lee. Fast multiscale neighbor embedding. *IEEE T NEUR NET LEAR*, pages 1–15, 2020.

[8] R. Byrd, Peihuang Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16:1190–1208, 1995.

[9] Peter Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. volume 93, 01 1993.

[10] Laurens van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

[11] Cyril de Bodt, Dounia Mulders, Michel Verleysen, and John A. Lee. Extensive assessment of Barnes-Hut t-SNE. In *ESANN*, pages 135–140, 2018.

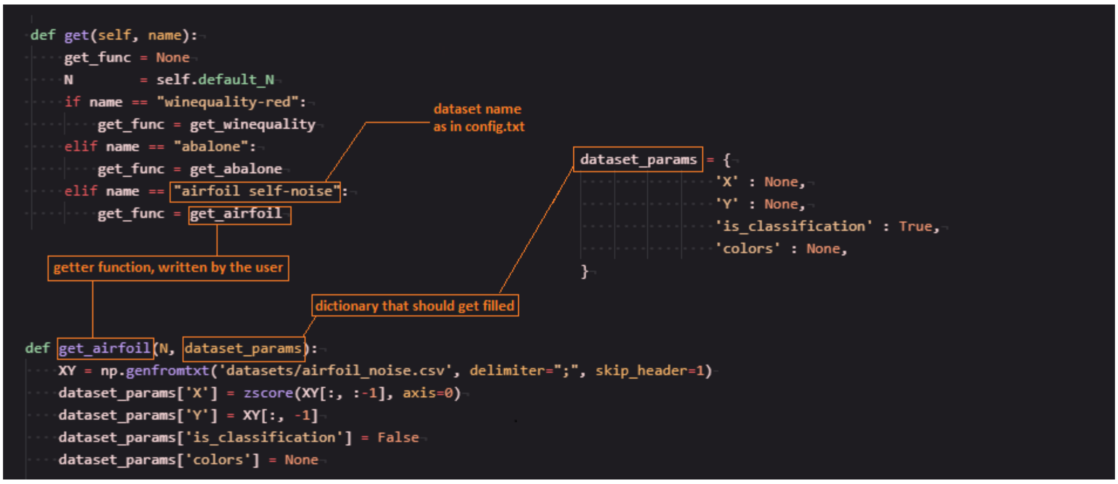
[12] Moshe Lichman. UCI Machine Learning repository, 2013.

[13] John Lee and Michel Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72:1431–1443, 03 2009.

Annex D: Adding a new dataset to the software

Three steps are required to add a new dataset, steps 2 and 3 are illustrated in the following figure:

1. Inside *config.txt*, add the name of the new dataset in the appropriate field.
2. Inside the *get()* function in *Data_loader.py*, add a new case in the big *if/else* statement, the new case should correspond to the name added in *config.txt*. Link the dataset name to the function of step 3, which loads the dataset.
3. Write the function added in step 2, it should fill the *dataset_params* dictionary. The N argument is only present as an indicator for the default dataset size, which can be modified in *config.txt*.



```
def get(self, name):
    get_func = None
    N        = self.default_N
    if name == "winequality-red":
        get_func = get_winequality
    elif name == "abalone":
        get_func = get_abalone
    elif name == "airfoil self-noise":
        get_func = get_airfoil
```

```
dataset_params = {
                  'X' : None,
                  'Y' : None,
                  'is_classification' : True,
                  'colors' : None,
}
```

```
def get_airfoil(N, dataset_params):
    XY = np.genfromtxt('datasets/airfoil_noise.csv', delimiter=";", skip_header=1)
    dataset_params['X'] = zscore(XY[:, :-1], axis=0)
    dataset_params['Y'] = XY[:, -1]
    dataset_params['is_classification'] = False
    dataset_params['colors'] = None
```

dataset name
as in config.txt
getter function, written by the user
dictionary that should get filled

Figure 6.1: Adding a new dataset by modifying Data_loader.py

The dictionary *dataset_params*'s "X" field should be filled with a numpy array of size (N, M) for a dataset of size N and dimensionality M.

The "Y" field should contain the labels; for classification, the labels should be integers. The "Y" field can be left to *None* for non-labeled data.

The "is_classification" field is a boolean indicating if the dataset should be interpreted as a classification dataset or as a regression dataset. This field is ignored if the "Y" field is set to *None*.

The "colors" field is only relevant to classification datasets, it allows the user to link the classes to a specific user-chosen color. If not set to *None*, this field should contain a 2-dimensional numpy array, the first dimension should correspond to the number of different classes, and the second dimension should contain the three red-green-blue values for the color, with values in $[0, 255]$.
If this field's value is set to *None*, the colors are generated automatically by using a K-means clustering algorithm on a randomly populated 3-dimensional space. Each cluster center is corresponds to a color, the first dimension corresponds to a red value, the second to a green value, and the third to a blue value.

Annex E: Adding a new DR algorithm to the software

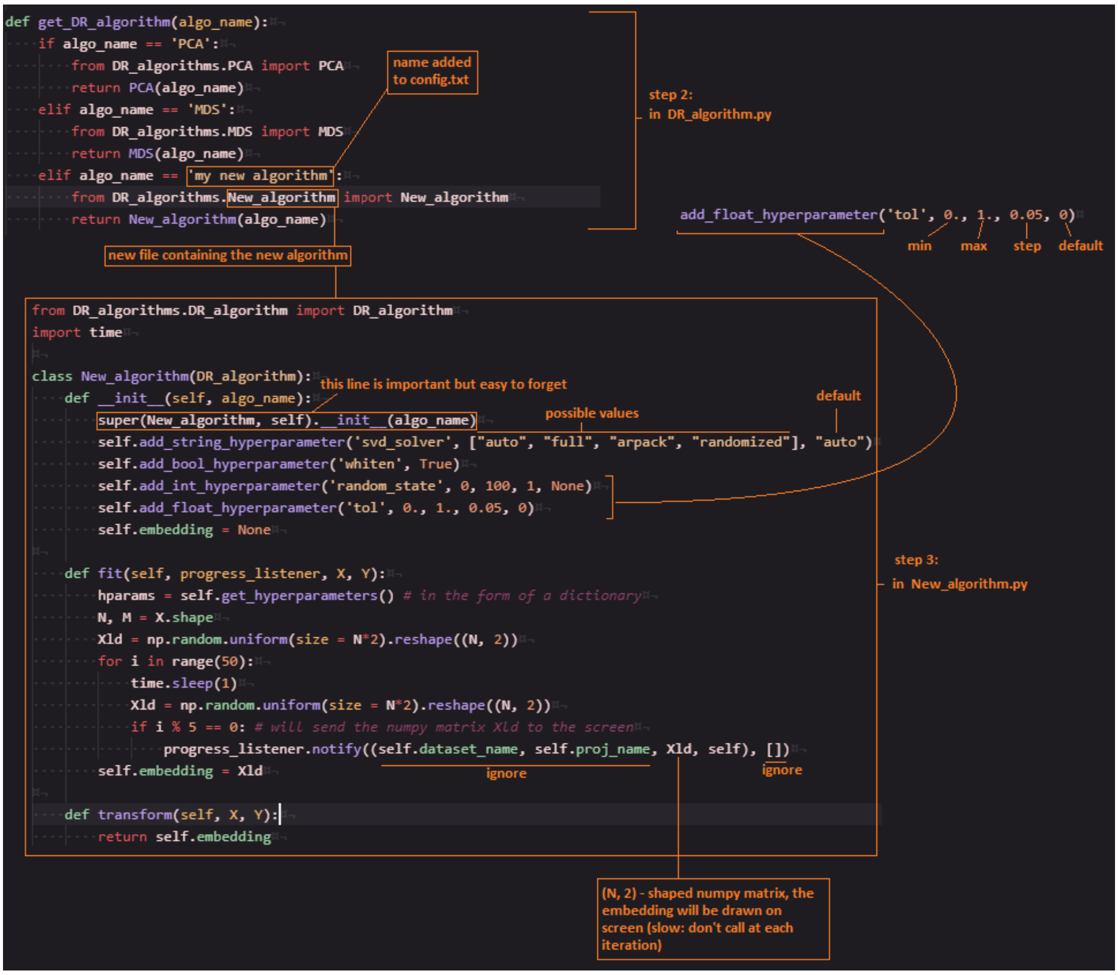
Three steps are necessary to add a new DR algorithm to *the software*:

1. Inside *config.txt*, add the name of the new algorithm in the appropriate field.

2. Inside *DR_algorithms/DR_algorithm.py*, add a case to the big *if/else* statement, with the name that was added to *config.txt* in step 1. The class which will be created in step 3 should be imported inside this case, and a class instance generated and returned (see Figure 5.2).

3. Create a file which will contain the new algorithm in the directory *DR_algorithms/*. This file should contain a class which inherits the class *DR_algorithm.py* (see Figure 5.2), this class should have at least three methods: *__init__*, *fit*, and *transform*.
   An example for these methods is given in Figure 5.2. Hyperparameters can be defined in the *__init__* function, and the *progress_listener* can be used to send the embedding to the screen during the optimisation.

By taking example from the existing implementations and from the following figure, a user with some programming knowledge shouldn't have any difficulties to add a new algorithm.



```python
def get_DR_algorithm(algo_name):
    if algo_name == 'PCA':
        from DR_algorithms.PCA import PCA
        return PCA(algo_name)
    elif algo_name == 'MDS':
        from DR_algorithms.MDS import MDS
        return MDS(algo_name)
    elif algo_name == 'my new algorithm':
        from DR_algorithms.New_algorithm import New_algorithm
        return New_algorithm(algo_name)
```

```python
add_float_hyperparameter('tol', 0., 1., 0.05, 0)
```

```python
from DR_algorithms.DR_algorithm import DR_algorithm
import time

class New_algorithm(DR_algorithm):
    def __init__(self, algo_name):
        super(New_algorithm, self).__init__(algo_name)
        self.add_string_hyperparameter('svd_solver', ["auto", "full", "arpack", "randomized"], "auto")
        self.add_bool_hyperparameter('whiten', True)
        self.add_int_hyperparameter('random_state', 0, 100, 1, None)
        self.add_float_hyperparameter('tol', 0., 1., 0.05, 0)
        self.embedding = None

    def fit(self, progress_listener, X, Y):
        hparams = self.get_hyperparameters() # in the form of a dictionary
        N, M = X.shape
        Xld = np.random.uniform(size = N*2).reshape((N, 2))
        for i in range(50):
            time.sleep(1)
            Xld = np.random.uniform(size = N*2).reshape((N, 2))
            if i % 5 == 0: # will send the numpy matrix Xld to the screen
                progress_listener.notify((self.dataset_name, self.proj_name, Xld, self), [])
        self.embedding = Xld

    def transform(self, X, Y):
        return self.embedding
```

(N, 2) - shaped numpy matrix, the embedding will be drawn on screen (slow: don't call at each iteration)

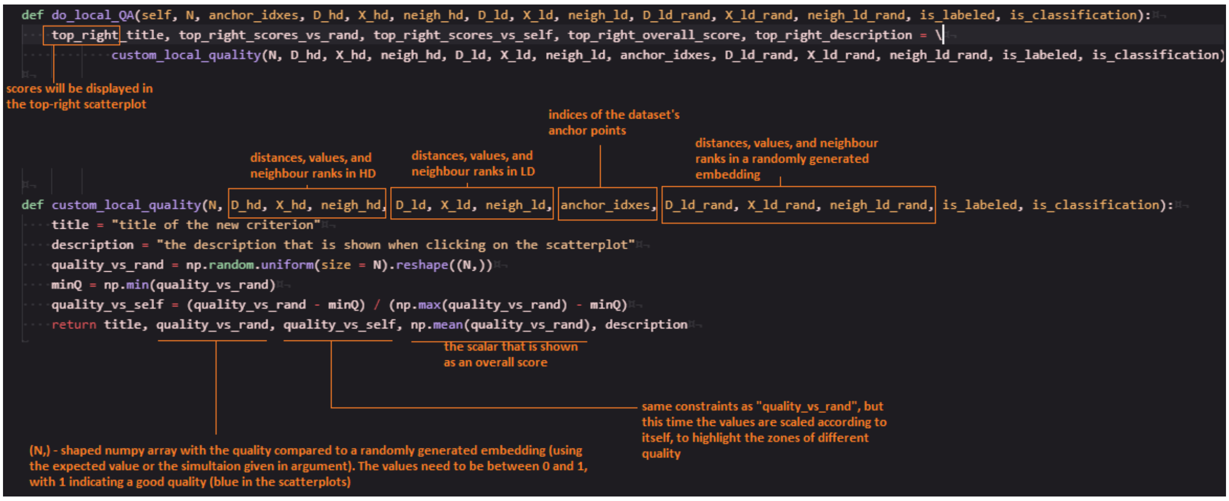
Figure 6.2: Adding a new DR algorithm to the program

Annex F: Adding a new local quality criterion the software

Two steps are necessary to add a new local quality criterion to *the software*:

1. Inside *local_distQA.py*,*local_labelQA.py*, or *local_neighQA.py*, find which variables correspond to the scatterplot that should show the new criterion by looking at their names ("top_right", "bottom_left" or "bottom_right"). Once these variables are identified, replace the call to the function that sets their value by the function written in step 2.

2. Write the function that computes the local scores, as described in the following figure. It should be pointed out that the randomly generated embedding mentioned in the figure is shared between all scatterplots (it's generated when opening the dataset).



```
def do_local_QA(self, N, anchor_idxes, D_hd, X_hd, neigh_hd, D_ld, X_ld, neigh_ld, D_ld_rand, X_ld_rand, neigh_ld_rand, is_labeled, is_classification):
    top_right_title, top_right_scores_vs_rand, top_right_scores_vs_self, top_right_overall_score, top_right_description = \
            custom_local_quality(N, D_hd, X_hd, neigh_hd, D_ld, X_ld, neigh_ld, anchor_idxes, D_ld_rand, X_ld_rand, neigh_ld_rand, is_labeled, is_classification)


def custom_local_quality(N, D_hd, X_hd, neigh_hd, D_ld, X_ld, neigh_ld, anchor_idxes, D_ld_rand, X_ld_rand, neigh_ld_rand, is_labeled, is_classification):
    title = "title of the new criterion"
    description = "the description that is shown when clicking on the scatterplot"
    quality_vs_rand = np.random.uniform(size = N).reshape((N,))
    minQ = np.min(quality_vs_rand)
    quality_vs_self = (quality_vs_rand - minQ) / (np.max(quality_vs_rand) - minQ)
    return title, quality_vs_rand, quality_vs_self, np.mean(quality_vs_rand), description
```

scores will be displayed in the top-right scatterplot

indices of the dataset's anchor points

distances, values, and neighbour ranks in HD

distances, values, and neighbour ranks in LD

distances, values, and neighbour ranks in a randomly generated embedding

the scalar that is shown as an overall score

same constraints as "quality_vs_rand", but this time the values are scaled according to itself, to highlight the zones of different quality

(N,) - shaped numpy array with the quality compared to a randomly generated embedding (using the expected value or the simultaion given in argument). The values need to be between 0 and 1, with 1 indicating a good quality (blue in the scatterplots)

Figure 6.3: Adding a new local QA criterion to the program

Bibliography

[1] Edward R. Tufte. *Envisioning Information.* Graphics Press, 1990.

[2] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[3] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling Theory and Applications.* Springer, New York, 2005.

[4] D. François, V. Wertz, and M. Verleysen. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering*, 19:873–886, 2007.

[5] Stephen L. France and Ulas Akkucuk. A review, framework, and r toolkit for exploring, evaluating, and comparing visualization methods. *The Visual Computer*, 03 2021.

[6] René Cutura, Stefan Holzer, M. Aupetit, and M. Sedlmair. Viscoder: A tool for visually comparing dimensionality reduction algorithms. In *ESANN*, 2018.

[7] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840, 2003.

[8] Peter Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. volume 93, 01 1993.

[9] Laurens van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

[10] Cyril de Bodt, Dounia Mulders, Michel Verleysen, and John A. Lee. Extensive assessment of Barnes-Hut t-SNE. In *ESANN*, pages 135–140, 2018.

[11] John A. Lee, Diego H. Peluffo-Ordóñez, and Michel Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261, 2015.

[12] Cyril de Bodt, Dounia Mulders, Michel Verleysen, and John Aldo Lee. Fast multiscale neighbor embedding. *IEEE T NEUR NET LEAR*, pages 1–15, 2020.

[13] Cyril de Bodt, Dounia Mulders, Daniel López-Sánchez, Michel Verleysen, and John Lee. Class-aware t-sne: cat-sne. 04 2019.

[14] George Linderman, Manas Rachh, Jeremy Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature Methods*, 16:1, 03 2019.

[15] Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. 02 2018.

[16] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 585–591, Cambridge, MA, USA, 2001. MIT Press.

[17] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[18] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[19] J. de Leeuw and P. Mair. Multidimensional scaling using majorization: Smacof in r. *Journal of Statistical Software, Articles*, 31(3):1–30, 2009.

[20] D. Kobak and P. Berens. The art of using t-sne for single-cell transcriptomics. *Nat Commun 10, 5416*, 2019.

[21] J. A. Lee, D. H. Peluffo-Ordóñez, and M. Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261, 2015.

[22] Jan De Leeuw and Patrick Mair. *Shepard Diagram.* 06 2015.

[23] Geng Chen, Baitang Ning, and Tieliu Shi. Single-cell rna-seq technologies and related computational data analysis. *Frontiers in Genetics*, 10:317, 2019.

[24] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[25] John Aldo Lee and Michel Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005. Geometrical Methods in Neural Networks and Learning.

[26] Adrien Bibal, Viet Vu, Géraldin Nanfack, and Benoît Frénay. Explaining t-sne embeddings locally by adapting lime. In *ESANN*, 2020.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**