UCLouvain

École polytechnique de Louvain

# Geodesic distances in high-dimensional spaces

## The Sneodesic distance overcomes the curse of dimensionality

Author: **Jérémy Lebrun**
Supervisor: **John Lee**
Readers: **Marco Saerens, Cyril De Bodt, Pierre Lambert**
Academic year 2023-2024
Master [120] in Data Sciences Engineering

# Acknowledgements

I would like to extend my deepest gratitude to my supervisor, Mr. John Lee, who initially proposed a subject that deeply nourished my interest and motivation throughout this year. His guidance and insight have been invaluable to the progress and completion of this thesis.

I would also like to express my sincere thanks to Mr. Lee and his assistants, Mr. Cyril De Bodt and Mr. Pierre Lambert, for their availability, precious help, and motivation during our meetings. The discussions I had the chance to share with them were not only instructive but also enjoyable, significantly enriching my research experience.

Furthermore, I am grateful to Mr. Marco Saerens for kindly agreeing to be a part of my jury. His expertise and time are greatly appreciated.

To all of you, thank you for your support and contributions, which have been instrumental in the completion of this work.

# Abstract

In data analysis, the choice of distance metrics is paramount, especially in high-dimensional spaces where traditional measures like Euclidean distance become less effective due to the curse of dimensionality. This thesis introduces the sneodesic distance, a novel metric designed to address this challenge. The term 'sneodesic' is a contraction of 'SNE' (Stochastic Neighbor Embedding) and 'geodesic', reflecting its foundation. By incorporating SNE similarities with a small perplexity into geodesic distances, the sneodesic distance maintains meaningful relationships between data points and accounts for both the local and global structure of the dataset.

This new metric proves beneficial in several machine learning applications, and shows situational improvements in dimensionality reduction for better data visualization.

Theoretical explorations and empirical evaluations demonstrate the robustness and broad applicability of the sneodesic distance, making it a valuable tool for high-dimensional data analysis.

# Contents

**Introduction** **1**

**1 State of the Art** **3**
1.1 Distance Metrics . . . 3
1.1.1 Minkowski Distance . . . 3
1.1.2 Mahalanobis Distance . . . 4
1.1.3 k-NN Distance . . . 5
1.1.4 Commute-Time Distance . . . 8
1.2 Dimensionality Reduction . . . 9
1.2.1 Principal Component Analysis . . . 10
1.2.2 Multidimensional Scaling . . . 10
1.2.3 Stochastic Neighbor Embedding . . . 11
1.2.4 Isomap . . . 12
1.2.5 Autoencoder . . . 13
1.2.6 Parametric Nature and Invertibility . . . 13

**2 Definition of the Distance** **15**
2.1 Perplexity . . . 16
2.2 Shortest Path Algorithm . . . 19
2.2.1 Simple Addition . . . 20
2.2.2 Shortcut Path . . . 20
2.3 Distance properties . . . 23
2.3.1 Positivity . . . 23
2.3.2 Nullity . . . 23
2.3.3 Symmetry . . . 24
2.3.4 Triangle Inequality . . . 25

**3 Methodology** **26**
3.1 Performance Assessment Methods . . . 26
3.1.1 Neighborhood Study . . . 26
3.1.2 2D-Plot . . . 28
3.1.3 $Q_{NX}(K)$ . . . 28
3.1.4 Strengths . . . 30
3.2 Datasets . . . 31
3.3 Distance metrics . . . 33
3.4 Dimensionality Reduction Methods . . . 35

**4 Experiments and Results** **36**
4.1 Introductory Example . . . . . 36
4.2 Angle Distribution . . . . . 37
4.3 Neighborhood Study . . . . . 40
4.4 2D Plots . . . . . 43
4.5 $Q_{NX}$ . . . . . 46

**5 Discussion** **50**
5.1 Summary of Results . . . . . 50
5.2 Limitations and Future Work . . . . . 51

**Conclusion** **53**

**A Neighborhood Study Plots** **54**

**B 2D Plots** **56**
B.1 MDS . . . . . 56
B.2 t-SNE . . . . . 61

**C $Q_{NX}(K)$ Plots** **65**

# Introduction

In the realm of data analysis, the ability to effectively measure distances between data points is of critical importance. Distance metrics form the base of many analytical methods, influencing their accuracy and efficiency. Traditional metrics such as the Euclidean distance, established by Greek mathematicians over two millennia ago, have long been the standard for distance measurement. However, as data complexity and dimensionality have increased, these traditional metrics have shown limitations, particularly due to the phenomenon known as the 'curse of dimensionality'. This term, appearing in the 1960s, describes the way in which high-dimensional spaces cause distances to lose their discriminative power, thereby complicating data analysis.

The curse of dimensionality presents a significant challenge in the analysis of high-dimensional datasets. As the number of dimensions increases, the volume of the space increases exponentially, leading to sparsity of data points and making traditional distance metrics less effective. This inefficiency necessitates the development of novel distance measures that can provide more robust and meaningful assessments in high-dimensional contexts. One recent attempt to address these challenges has been the development of geodesic distances. However, while geodesic distances have shown some promise, they too have limitations in very high-dimensional spaces. One of the major drawbacks of the geodesic distance is the presence of the hyperparameter $k$, which makes the distance highly sensitive to its value. Determining the appropriate value of $k$ is challenging because it requires prior knowledge of the dataset, which is often unavailable in high-dimensional settings.

To address these challenges, this thesis proposes the sneodesic distance, a new type of distance metric designed to mitigate the difficulties posed by high-dimensional spaces. The sneodesic distance is formulated to retain meaningful relationships between data points even as dimensionality increases, thus offering a potential solution to the curse of dimensionality. Unlike the geodesic distance, the sneodesic distance eliminates the need for the hyperparameter $k$. This novel distance is a modification of the geodesic distance, by using SNE (Stochastic Neighbor Embedding) similarities instead of Euclidean distances to weight the edges in the geodesic distance calculation. The name 'sneodesic' is a contraction of 'SNE' and 'geodesic', reflecting this combination.

The sneodesic distance proves utility in the context of machine learning, particularly in the k-nearest neighbors (k-NN) classification algorithm, which is one of the most pure and widely used classification methods. The effectiveness of k-NN heavily relies only on the choice of distance metric, making the development of robust metrics like the sneodesic distance crucial for improving classification performance.

The relevance of the sneodesic distance extends beyond its direct application in supervised machine learning models. It can also be used to perform clustering or as an input to dimensionality reduction algorithms, enabling the visualization of high-dimensional datasets in two-dimensional spaces. These applications are particularly valuable in fields such as bioinformatics, image processing, and machine learning, where high-dimensional data is prevalent. By attempting to preserve the intrinsic geometric properties of data, the sneodesic distance facilitates more accurate and insightful visualizations, which are crucial for data interpretation and hypothesis generation.

The contributions of this thesis are multiple. First, I have developed and implemented the sneodesic distance metric in Python, providing a practical tool for researchers and practitioners to use in their analyses. I have also conducted a study of its mathematical properties to ensure robustness and reliability. Concurrently, I performed empirical evaluations of the sneodesic distance on various datasets, comparing its performance to existing distance metrics such as Euclidean or geodesic distances. I also developed several ways of computing this distance and compared them. This dual approach of theoretical exploration and practical application highlights the potential benefits and broad applicability of the sneodesic distance.

This thesis is structured as follows. We begin with a review of existing distance metrics and dimensionality reduction methods, providing the necessary background for understanding the need for sneodesic distances. Following this, we introduce and formally define the sneodesic distance, discussing its properties in detail. We then describe the tools and metrics used to assess the performance of the sneodesic distance. Our experimental results, which compare types of sneodesic distance between themselves and to traditional metrics across multiple datasets, are presented next. Finally, we conclude with a summary of our findings and suggestions for future research directions.

In summary, this thesis aims to introduce, define, and validate the sneodesic distance as a robust alternative to traditional distance metrics. By addressing the curse of dimensionality and facilitating more meaningful data visualizations, the sneodesic distance enhances the potential of data analysis methods and offers a valuable tool for high-dimensional data exploration.

*Note : I would like to acknowledge the use of ChatGPT-3*[1] *for assisting in correcting some English mistakes and rephrasing some sentences in this thesis. Additionally, all figures not specifically referenced have been created by myself using* `matplotlib.pyplot`[2] *or GeoGebra*[3].

[1] https://chatgpt.com/?oai-dm=1
[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
[3] https://www.geogebra.org/geometry

# Chapter 1

# State of the Art

## 1.1 Distance Metrics

The development of distance metrics has been a pivotal aspect of data analysis, with each metric offering unique properties that make it suitable for specific contexts. Some metrics perform exceptionally well in low-dimensional spaces, while others are designed to handle particular structural characteristics within the data.
In this context, a distance metric, or a distance measure, is defined as a method for quantifying the dissimilarity between two points in a space. The choice of distance metric can significantly impact the outcome of various analytical methods, highlighting the importance of understanding and selecting the appropriate metric for a given task.

### 1.1.1 Minkowski Distance

The Minkowski distance [1] is a generalized metric that encompasses several specific distance measures depending on the parameter $p$. Two commonly used instances of the Minkowski distance are the Euclidean distance and the Manhattan distance.

The Euclidean distance is the most widely used and well-known distance measure. It is given by the formula

$$d(x, y) = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2} \ .$$

Intuitively, it measures the straight-line distance between two points in an Euclidean space, which is, in a nutshell, a finite-dimensional space where points have coordinates, and distances and angles follow the rules of Euclidean geometry.

The Manhattan distance has a similar formula to the Euclidean distance, being

$$d(x, y) = \sum_{i=1}^{D} |x_i - y_i| \ .$$

Instead of measuring the straight-line distance, it follows a grid-like path, summing the absolute differences of their coordinates.

The general form of the Minkowski distance is

$$d(x, y) = \left( \sum_{i=1}^{D} |x_i - y_i|^p \right)^{\frac{1}{p}} .$$

By setting $p = 2$, we obtain the Euclidean distance while we have the Manhattan distance with $p = 1$. This flexibility makes the Minkowski distance a powerful tool for measuring distances in various contexts.

### 1.1.2 Mahalanobis Distance

The Mahalanobis distance [2] is a statistical measure that yields favorable outcomes when dealing with correlated variables, as it incorporates the covariance structure of the data. Utilizing the covariance between variables also renders it scale-invariant, a significant advantage. This distance standardizes the dataset and projects it into a space where variables are uncorrelated and have unit variances. The distance is computed with

$$d(x, y) = \sqrt{(x - y)^{\mathrm{T}} \Sigma^{-1} (x - y)} .$$

Here, $\Sigma$ represents the covariance matrix, with each entry $\Sigma_{ij}$ defined as

$$\Sigma_{ij} = \mathrm{cov}(\mathbf{X_i}, \mathbf{X_j}) = \mathrm{E}\left[ (\mathbf{X_i} - \mathrm{E}[\mathbf{X_i}])(\mathbf{X_j} - \mathrm{E}[\mathbf{X_j}])^{\mathrm{T}} \right]$$

where $\mathbf{X_i}$ denotes the random vector defined by the $i^{th}$ dimension of each element of the dataset. It is important to note that having a variable with null variance leads to $\exists i : \mathrm{cov}(\mathbf{X_i}, \mathbf{X_i}) = \Sigma_{ii} = 0$. This makes $\Sigma$ singular and thus non-invertible, making the computation of $d(x, y)$ impossible with the formula presented above. This scenario is not uncommon; for instance, in image datasets like *MNIST* or *Coil-20* where borders often remain uni-color. One can observe in Figure 1.1 that corner pixels of both these datasets never change values, remaining black.

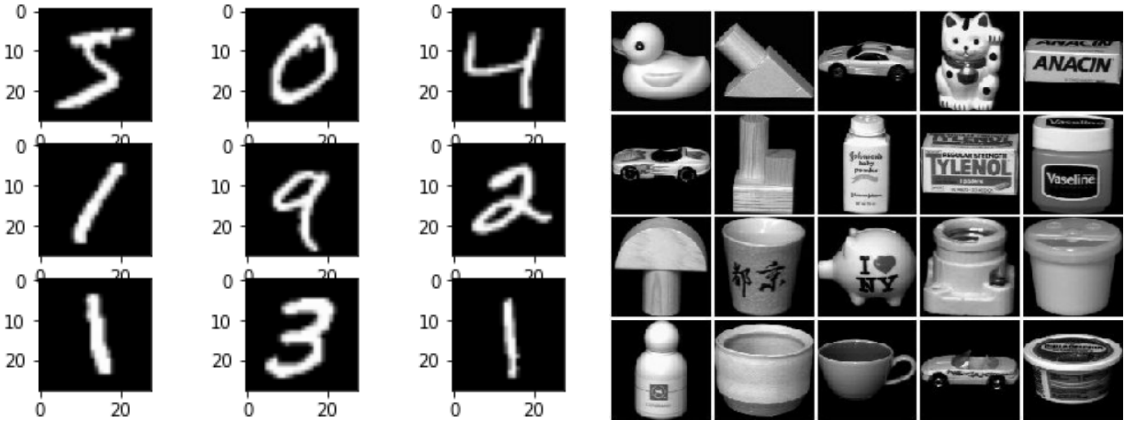



Figure 1.1: Samples of *MNIST* [3] (left) and *Coil-20* [4] (right) datasets

For images, coordinates are created by representing each pixel by one (e.g., grayscale) or more values (e.g., RGB) before flattening the matrix to form a vector. A pixel at a constant value, here black, in every image results in zero variance. Another situation leading to a singular $\Sigma$ is when two variables are linearly dependent, which is also not rare.

To address these issues and significantly reduce computation time by avoiding the inversion of a matrix as large as the number of dimensions, a practical estimation approach can be employed. Here is the method, with $X$ representing the data with observations in rows and variables in columns:

1. Transpose the data: $X = X^T$
2. Center the data: Subtract the mean of each row $i$ from each element $X_{ij}$, resulting in a new centered matrix $X_c$
3. Perform Singular Value Decomposition (SVD): $X_c = USV^T$
4. Retain $V'$: $V'$ comprises the first $n$ columns of $V$
5. Final step: The Mahalanobis distance matrix is the Euclidean distance matrix while taking $V'$ as the dataset

In fact, $V'$ represents the PCA decomposition in $n$ dimensions, where $n$ is a parameter to be defined. [5]

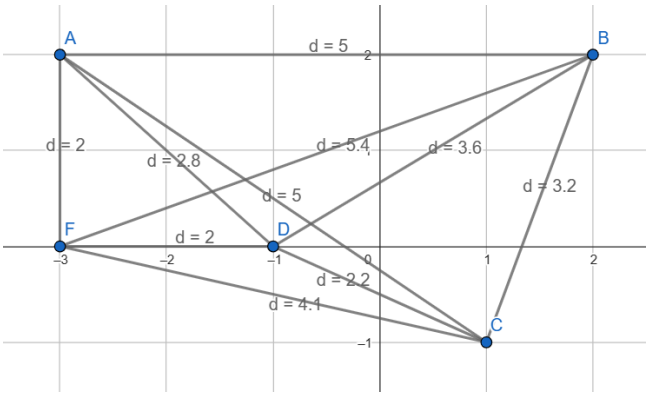
### 1.1.3 k-NN Distance

The k-NN distance stands for 'k-Nearest Neighbors' distance. It is often referred to as 'geodesic distance' [6], but using k-NN highlights the parameter $k$. This notation directly indicates the value of $k$, making the writing more concise and emphasizing that distances such as 5-NN and 50-NN are fundamentally different due to the change in $k$, as explained below. Moreover, in a sense we could say that k-NN distances are empirical approximations of the more strict geodesic definition. They operate on sampled manifolds.

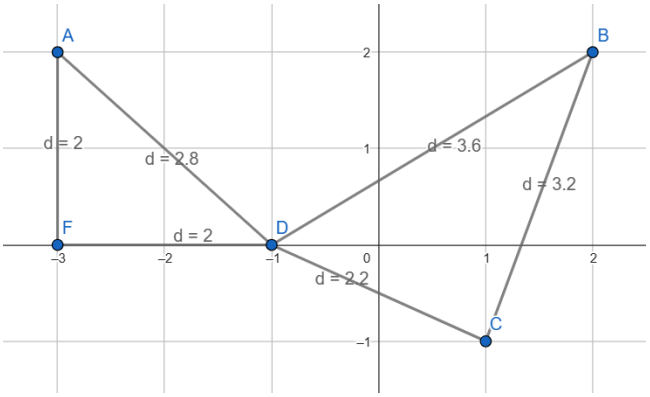
Like many other distance metrics, it is based on the Euclidean distance. Therefore, it may retain some of its limitations. Despite this, k-NN can yield better results in certain scenarios and can even mitigate the curse of dimensionality.

This distance is mentioned frequently in this thesis. Thus, in order to make it as clear as possible, let us inspect a simple example in Table 1.1.

1. Compute the Euclidean distance between each pair of points, which will be used when talking about 'nearest points'




2. Build a graph $G$ consisting of linking each point with its $k$ nearest neighbors, giving each edge the Euclidean distance between points it links as weight[1][2]




3. Compute a shortest path algorithm on the graph $G$ between each pair of points; the lengths of these paths will be our k-NN distances

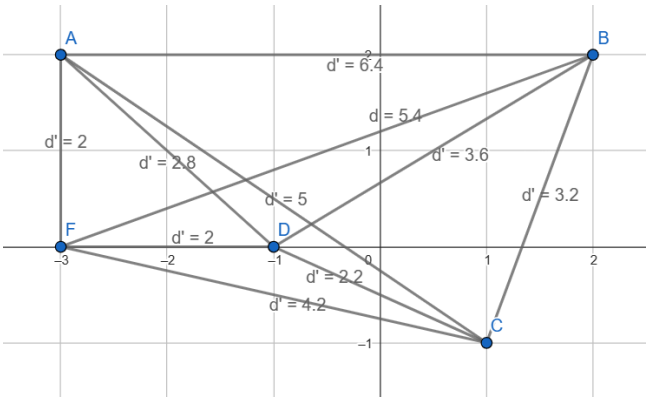



Table 1.1: Steps to compute 2-NN distances on a 2-dimensional example dataset

However, we could encounter a problem here. What should we do if the k-NN graph is not connected? The simple solution would be to assign an infinite distance between two points belonging to different connected components. This solution, being not the most practical for future applications used in this thesis, leads us to propose the following approach.

[1]Giving a weight of 1 to each edge is also a possibility but not the one used in this thesis. It is a way to completely eliminate the influence of Euclidean distances in the final values.

[2]When building the edges, we make them undirected here; directed edges are also possible.

We treat each connected component of $G$, the initial k-NN graph, as a vertex in a graph $G'$. The distance between each of these vertices is the shortest Euclidean distance between any two points belonging to each of the components in $G$. Therefore, this distance serves as the weight of the edges linking all pairs of points in the graph $G'$. Subsequently, we apply the Kruskal's Minimum Spanning Tree Algorithm [7] on this graph.

In a nutshell, this algorithm proceeds as follows :

1. Remove all the edges of the graph
2. Sort the edges in non-decreasing order based on their weights
3. Take the first edge: if it does not create a cycle (meaning there is a path strictly longer than 0 to go from one point to itself), add it to the minimum spanning tree; if not, discard it
4. Repeat the third step until the graph is connected

The output of this algorithm is a minimum spanning tree with $|V| - 1$ edges. These edges connect the vertices of $G'$, recalling they represent the connected components of $G$. We add then these edges to $G$. An edge connecting components $C_1$ and $C_2$ links the closest pair of points in $C_1$ and $C_2$, with the weight being the Euclidean distance between them.

Overall, we employ the k-NN distance when we aim to measure the distance following the curvature of the dataset. Each segment whose length is computed lies between neighboring points, ensuring that we only calculate distances following the shape of the dataset as strictly as possible. Let us consider a simple example, the Swiss roll. In a 2-dimensional space, it appears as a wrapped line; in 3 dimensions, a wrapped plane; in 4 dimensions, a wrapped volume, and so on.

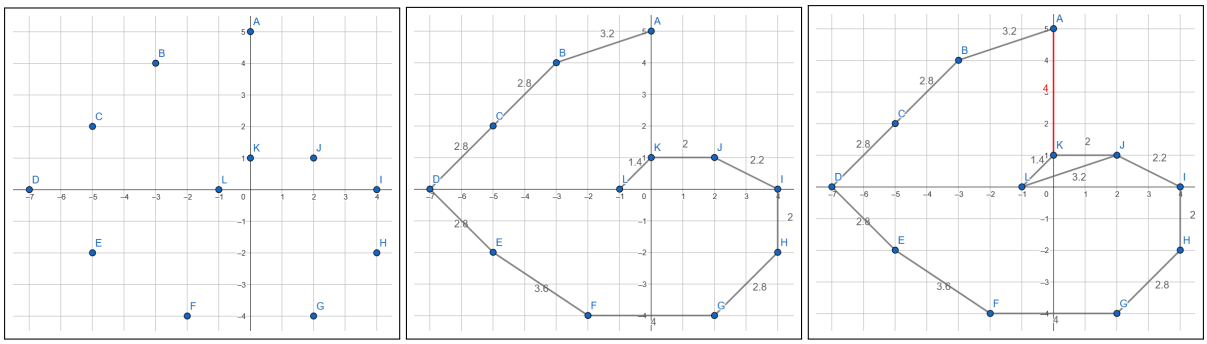



Figure 1.2: An example Swiss roll dataset in 2 dimensions (left) on which we build *1-NN* (center) and *2-NN* (right) graphs

The example in Figure 1.2 highlights the significance of the parameter $k$. Some results are presented in Table 1.2.

| | Euclidean distance | 1-NN distance | 2-NN distance |
|---|---|---|---|
| $d(A, B)$ | 3.2 | 3.2 | 3.2 |
| $d(A, K)$ | 4 | 28.2 | 4 |
| $d(A, G)$ | 9.1 | 19.2 | 13 |

Table 1.2: Some distances computed from Figure 1.2

When two points are direct neighbors (like A and B), there is no difference between Euclidean and k-NN distance, regardless of the value of $k$. However, for A and K, the distance exhibits considerable variation across these metrics. While the 1-NN and 2-NN graphs are identical, except for two additional edges between A and K and L and J, these edges significantly impact several distances. This makes the k-NN distance highly sensitive to the $k$ parameter. While some applications may favor the third figure, others may prefer the second one, considering then this variation as a problem. This phenomenon, highlighted in red in the third plot of Figure 1.2, is known as the 'shortcut problem', where an unwanted shortcut is created in the path. However, with numerous dimensions and points, visualizing the dataset and anticipating the distances becomes challenging, making the choice of $k$ difficult and the requirement to select one considerably burdensome.

The k-NN distance provides an area for improvement against the distance concentration in high-dimensional spaces, as long as $k$ is not too large. Indeed, it will dissociate this huge concentration by putting a coefficient to these distances. Essentially, if each edge has approximately the same weight (caused by distance concentration), the k-NN distance between two points is just equal to the number of edges composing the path between them multiplied by this 'mean' weight. If $k$ is too large, however, these paths remain too short to observe significant differences.

### 1.1.4 Commute-Time Distance

The Commute-Time Distance [8] is designed to compute the similarity between nodes of a weighted and undirected graph. This implies that we can apply it to the k-NN graph discussed earlier. However, there is an important difference to take into account; we assign the inverse of distances as weights to the edges.

The principle involves conducting a random walk through the graph, where each edge has a transition probability $p_{ij} = \frac{a_{ij}}{a_{i.}}$, with $a$ being the adjacency matrix of the graph and $a_{i.} = \sum_{j=1}^{n} a_{ij}$. Our aim is to compute the average first-passage time $m(k|i)$, defined as the average number of steps a random walker takes to reach point $k$ from $i$ for the first time. Here, 'first time' means we are concerned solely with the time it takes for the random walker to arrive at this point once; any subsequent visits are disregarded.

Now, let us discuss how to compute these values $m(k|i)$. They follow the recurrence equations

$$\begin{cases} m(k|k) = 0 \\ m(k|i) = 1 + \sum_{j=1}^{n} p_{ij} m(k|j) & \text{for } i \neq k \end{cases}.$$

There are various ways to derive the final $m$ values. We can easily iterate the recurrence equation. Alternatively, more efficient algorithms exist to solve such equations, representing first-order Markov chains, which are defined by transition probabilities depending solely on the current state and not on past ones. An option is to use the pseudoinverse of the Laplacian matrix of the graph.

Building a distance matrix $M$ with these $m$ values would make it not symmetric. To prevent this, our final distances $n$, named 'average commute-time distances', are computed as

$$n(i, j) = m(j|i) + m(i|j) \ .$$

The ingenious aspect of the commute-time distance lies in the fact that strongly connected nodes are closer than weakly connected ones. It is an innovative method to take into account a structural property, which is not provided by the k-NN distances discussed earlier. Additionally, if we take $\sqrt{N}$ as our final distance matrix, it exhibits the wonderful property of being Euclidean and is called the Euclidean Commute-Time Distance (ECTD).

## 1.2 Dimensionality Reduction

With the worldwide explosion of machine learning, we have quickly encountered the following important dilemma: "If my inputs lie in a highly dimensional space, should I embed them in a space with fewer dimensions or should I leave them untouched?". This dilemma remains relevant today, but there are two significant advantages to embedding them; saving precious computational time during machine learning operations and avoiding the curse of dimensionality [9].

In high-dimensional spaces, the volume increases exponentially with the dimensionality, causing the density of data points to decrease as one can observe in Figure 1.3. As a result, the nearest points become relatively farther apart compared to what happens in lower-dimensional spaces. This leads to a phenomenon known as the curse of dimensionality, where distances between points tend to concentrate, and the concept of proximity becomes less meaningful. Consequently, this results in complications when exploring high-dimensional datasets and decreased performance for many machine learning models.

Now that we understand the potential benefits of reducing the number of dimensions, let us explore how to accomplish this. There exist various dimensionality reduction methods, each with different features or purposes. Some aim to preserve similarity, others focus on distance, and still others consider shape. There are even methods based on neural networks.

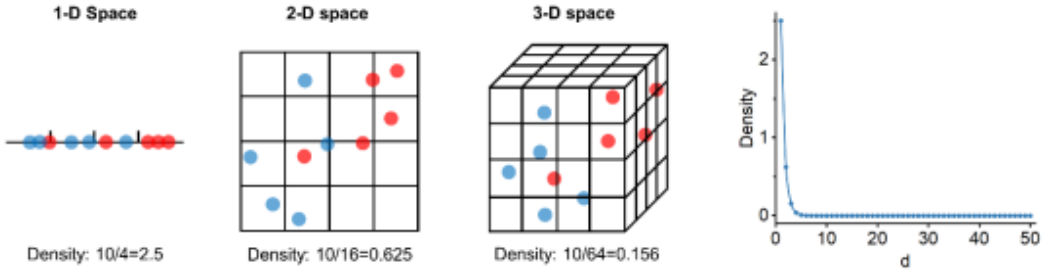



Figure 1.3: Effect of dimensionality on points density [9]

Let us examine some of the existing methods. Please note that $d$ and $\hat{d}$ represent the distances in the higher and lower dimensional spaces, respectively.

### 1.2.1 Principal Component Analysis

Let us begin by making a brief introduction to the well-known Principal Component Analysis [10] (PCA). This method works by geometrically projecting data onto lower dimensions called principal components (PCs). The first PC is chosen to minimize the total distance between the data points and their projection onto the PC, maximizing the variance of the projected points. Subsequent PCs are selected to be uncorrelated with all previous PCs and to capture as much remaining variance as possible. The PCs are defined as linear combinations of the original variables, and this process can be visualized as rotating the data to align with the directions of maximum variance. This method is unsupervised, meaning it finds patterns without reference to prior knowledge about the data.

Despite its usefulness, PCA has limitations. It assumes linear relationships among variables and may miss nonlinear patterns. Additionally, PCA maximizes variance, not necessarily identifying clusters within the data. It is also sensitive to the scaling of variables, which can influence the results. Therefore, proper standardization of variables is crucial, especially when they are on different scales.

### 1.2.2 Multidimensional Scaling

Multidimensional Scaling (MDS) [11] is a set of methods used to embed high-dimensional data into a lower-dimensional space while preserving the distances between points as much as possible. Two of the main types of MDS are *Metric MDS* and *Non-metric MDS*. Metric MDS aims to directly preserve the original distances $d_{ij}$ by minimizing the utility function

$$S_M(x_1, x_2, ..., x_n) = \sqrt{\sum_{i \neq j = 1,...,n} \left(d_{ij} - \hat{d}_{ij}\right)^2} \ .$$

Non-metric MDS, in contrast, seeks to preserve the rank order of distances rather than their exact values and minimizes the function

$$S_{NM}(x_1, ..., x_n; f) = \sqrt{\frac{\sum_{i<j} \left(f(d_{ij}) - \hat{d}_{ij}\right)^2}{\sum_{i<j} \hat{d}_{ij}^2}} \ .$$

For this thesis, Metric MDS is particularly suitable because our goal is to preserve the actual distances from the high-dimensional space in a lower-dimensional representation. This allows us to use any distance metric to compute the distance matrix as input, and the objective is to maintain these distances as accurately as possible during the dimensionality reduction process. It can give an idea of the 'realness' of the distance in high dimension by projecting its implication in a more observable low-dimensional space.

To achieve this algorithm, we use the Metric MDS implementation from the `scikit-learn` library, which employs the SMACOF [12] (Scaling by MAjorizing a COmplicated Function) algorithm. SMACOF is an iterative optimization algorithm that minimizes a stress function, measuring the goodness-of-fit of the configuration in the reduced-dimensional space. It guarantees a series of non-decresaing stress values with a linear convergence, making it an ideal choice for preserving distances but not the most computationally efficient.

This approach is perfect for our needs because it provides flexibility in choosing the distance metric for the initial distance matrix, and it effectively approximates the high-dimensional distances in a lower-dimensional space. This capability is crucial for understanding how well the distances are conserved.

### 1.2.3 Stochastic Neighbor Embedding

Stochastic Neighbor Embedding [13] (SNE) aims to preserve the local structure of data by modeling pairwise similarities between data points in both the original high-dimensional space and the low-dimensional embedding space. It is based on two quantities, $p_{ij}$ and $q_{ij}$, which represent the probabilities that data point $i$ would pick data point $j$ as its neighbor, respectively, in the original high-dimensional space and the low-dimensional space. They are computed by the formulas

$$p_{ij} = \frac{e^{-\frac{d_{ij}^2}{2\sigma^2}}}{\sum\limits_{l \neq i} e^{-\frac{d_{ik}^2}{2\sigma^2}}} \qquad \text{and} \qquad q_{ij} = \frac{e^{-\hat{d}_{ij}^2}}{\sum\limits_{l \neq i} e^{-\hat{d}_{ik}^2}} \ ,$$

where $\sigma^2$ is a parameter that determines the variance of the distribution formed by $p_{ij}$, which is Gaussian. This parameter, as well as the chosen distribution, is crucial. The distribution attributes levels of importance to points according to their distance while the variance determines up to which distance value is still considered non-negligible. $\sigma$ is strongly linked to the *perplexity*, a term of the utmost importance in this thesis and described in detail later in section 2.1.

The objective of SNE is to minimize the sum of Kullback-Leibler divergences between $p_{ij}$ and $q_{ij}$ for each object, given by the equation

$$C = \sum_i \sum_j p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) \ .$$

t-Distributed Stochastic Neighbor Embedding (t-SNE), a variant of SNE, replaces the Gaussian distribution for $q_{ij}$ with a Student's t-distribution, thereby preserving more of the global structure of the data and putting less importance on close neighborhoods. Specifically, the expression for $q_{ij}$ under t-SNE becomes

$$q_{ij} = \frac{(1 + \hat{d}_{ij}^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \hat{d}_{kl}^2)^{-1}} \quad .$$

The algorithm iteratively attempts to minimize the cost function using gradient descent. It starts by randomly initializing the low-dimensional embeddings and then updates them in each iteration to minimize the cost function.

### 1.2.4 Isomap

Isomap [6] is a method that builds upon classical MDS, another member of the family discussed in Section 1.2.2, but with a unique approach to distance calculation. Instead of using conventional distances, Isomap employs the k-nearest neighbors (k-NN) distance presented in Subsection 1.1.3, which is a distance metric which can adapt to complex structures of high-dimensional manifolds.

While various distance metrics can be used within MDS, employing the k-NN distance is particularly effective for preserving local neighborhood structure while also capturing global data characteristics. Additionally, using a small value for $k$ helps mitigate the issue of distance concentration in high-dimensional spaces as explained in Section 1.1.3.

Consider the most common application scenario of Isomap, illustrated by the 'Swiss roll' dataset in Figure 1.4. This dataset was also previously discussed in the heart of Section 1.1.3. One can observe in Figure 1.4.C its impressive ability to flatten this wrapped plane into 2 dimensions. However, as explained in Section 1.1.3, isomap is very sensitive to the parameter $k$. The 'shortcut problem' can appear, completely deforming the plane if $k$ is too large, while a too small $k$ makes the distances loose a lot of information about the neighborhood, forgetting points which can also be very close. That makes Isomap very performing in specific cases but difficult to apply to datasets whose the intrisic shape remains unknown.

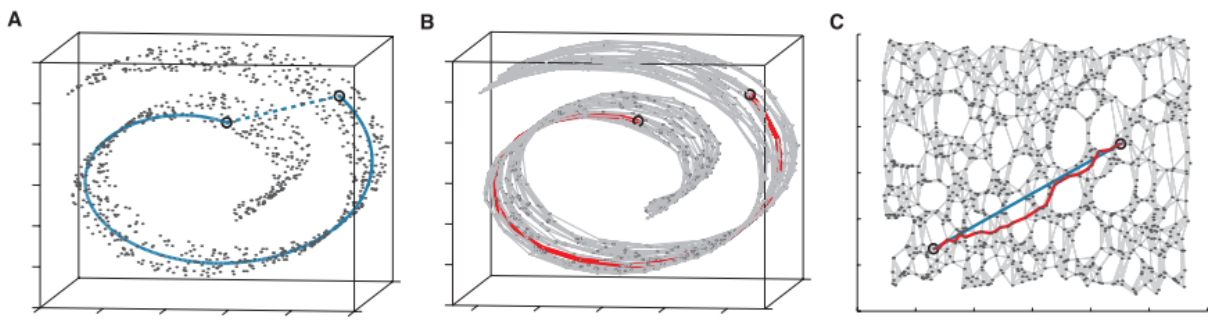



Figure 1.4: Application of Isomap reduction on a 3-dimensional Swiss roll [6]

### 1.2.5 Autoencoder

An autoencoder [14] is a type of artificial neural network used to learn efficient codings of input data in an unsupervised manner. It consists of two main parts: an encoder that maps the input into a lower-dimensional representation, and a decoder that reconstructs the input from this representation, as illustrated in Figure 1.5.

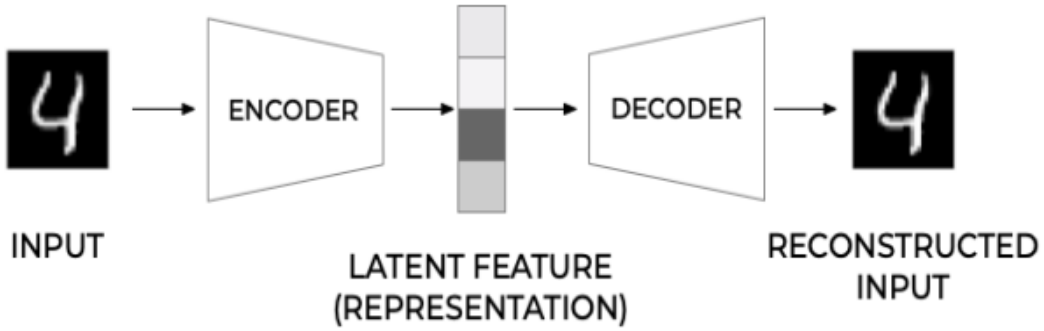



Figure 1.5: Illustration of an Autoencoder Architecture [14]

The primary objective of an autoencoder is to minimize the reconstruction error, ensuring that the compressed representation retains as much information as possible about the original data. This makes autoencoders highly effective for dimensionality reduction, especially when dealing with complex, high-dimensional data that more traditional dimensionality reductions methods can not handle.

The rise of deep learning has further enhanced the capabilities of autoencoders. Deep autoencoders, which utilize multiple hidden layers, can capture intrinsic patterns and structures of the data, leading to more robust and meaningful low-dimensional representations.

Given the increasing importance of deep learning in various domains, it is essential to consider autoencoders among the existing dimensionality reduction methods. They offer a powerful alternative to traditional techniques like PCA and MDS, providing the flexibility to learn non-linear relationships and adapt to complex data distributions. However, they cannot be used to compare metric performances as they do not take distances but coordinates as an input, as PCA.

### 1.2.6 Parametric Nature and Invertibility

In the realm of dimensionality reduction techniques, methods like PCA (Principal Component Analysis) and autoencoders are considered parametric and invertible. PCA is parametric because it decomposes data into principal components based on a covariance matrix, determined by eigenvectors and eigenvalues. Once these parameters are established, they can be directly applied to new data. Similarly, once the parameters of the autoencoders are trained, they can directly encode and decode new data using the same weights and biases.

On the other hand, methods such as t-SNE, MDS, and Isomap are non-parametric. These methods do not assume a predefined form for data distribution and require recalculations when new data is introduced. t-SNE minimizes the Kullback-Leibler divergence between pairwise points in high and low dimensions, requiring recalculation for new data. MDS positions points in low-dimensional space while preserving high-dimensional distances, also necessitating recalculations for new data. Isomap being built on MDS, meets the same shortcoming.

Regarding invertibility, PCA and autoencoders are notable because they allow for the reconstruction of original data from reduced representations. PCA can reconstruct original data using principal components and projection scores, while autoencoders use their decoding process for this purpose. Conversely, t-SNE, MDS, and Isomap are not inherently invertible, as they primarily focus on preserving relational structure in low-dimensional space without a direct mechanism to recover the original high-dimensional data.

Therefore, PCA and autoencoders are parametric and invertible due to their use of fixed parameters and ability to reconstruct original data, whereas t-SNE, MDS, and Isomap are non-parametric and non-invertible but can accept new distance metrics as input, which make them suitable for further experiments in this thesis.

# Chapter 2

# Definition of the Distance

Let us begin by introducing the formulas defining the sneodesic distance, which is the subject of this thesis.

We start with the SNE similarity

$$\sigma_{ij} = \begin{cases} 0 & \text{if } i = j \\ \frac{e^{-\pi_i \delta_{ij}}}{\sum_{k \neq i} e^{-\pi_i \delta_{ik}}} & \text{if } i \neq j \end{cases}$$

where $\delta_{ij}$ represents the Euclidean distance between points $i$ and $j$.
The initial formula takes squared Euclidean distances to have a Normal distribution instead of a Laplacian distribution presented here and will be also be discussed. However, they do not outperform results with simple Euclidean distances so it is interesting to study both, taking the simplest formula as base example.
Note that this value $\sigma_{ij}$ also refers to the probability $p_{ij}$ that point $i$ takes $j$ as its neighbor, as discussed in Section 1.2.3. This ensures that $0 \leq \sigma_{ij} \leq 1$, and consequently $-\log(\sigma_{ij}) \geq 0$. We take advantage of this positivity as well as the non-increasing nature of the function $-\log(x)$ to obtain a positive distance measure that increases as the similarity decreases. Additionally, we include a factor $\frac{1}{\pi_i}$, which will be discussed later in Section 2.1. It leaves us with the values $\overline{\delta}_{ij}$ computed as

$$\overline{\delta}_{ij} = \begin{cases} 0 & \text{if } i = j \\ -\frac{1}{\pi_i}\log(\sigma_{ij}) = \delta_{ij} + \frac{1}{\pi_i}\log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}}\right) & \text{if } i \neq j \end{cases}$$

Once we have our matrix $\overline{\delta}$, we symmetrize it by computing $\frac{\overline{\delta}^T + \overline{\delta}}{2}$. It is important to have a symmetric distance matrix, as mentioned in Section 2.3. Finally, we apply a shortest path algorithm on $\frac{\overline{\delta}^T + \overline{\delta}}{2}$ using the Floyd-Warshall algorithm explained in Section 2.2. The output it provides is the final sneodesic distance matrix.

When it comes to coding, another form is necessary due to precision issues in programming languages. For instance, in a simple Python code without any additional functions, `numpy.exp(-i)` becomes equal to 0 when $i$ is greater than or equal to 745. This means we would lose all information about $\delta$ or $\delta^2$ greater than 745, which is unacceptable. Using $\min_{l \neq i}(\delta_{il}) - \delta_{ij}$ instead of $\delta_{ij}$ solves this problem without changing any of the resulting values, as the factor $\exp(\min_{l \neq i}(\delta_{il}))$ appears in both the numerator and the denominator of the equation, canceling each other out.

The formulas used are then the following :

$$\sigma_{ij} = \begin{cases} 0 & \\ \dfrac{e^{\pi_i \left( \min_{l \neq i}(\delta_{il}) - \delta_{ij} \right)}}{\sum_{k \neq i} e^{\pi_i \left( \min_{l \neq i}(\delta_{il}) - \delta_{ik} \right)}} & \end{cases} \qquad \overline{\delta}_{ij} = \begin{cases} 0 & (\text{if } i = j) \\ \delta_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{\pi_i \left( \min_{l \neq i}(\delta_{il}) - \delta_{ik} \right)} \right) & (\text{if } i \neq j) \end{cases}$$

Using the squared Euclidean distances inside the formulas is more loyal to the initial SNE similarity formula and implies a Normal distribution. This version will be denoted as $\sigma^*_{ij}$ and $\overline{\delta}^*_{ij}$ :

$$\sigma^*_{ij} = \begin{cases} 0 & \\ \dfrac{e^{\pi_i \left( \min_{l \neq i}(\delta^2_{il}) - \delta^2_{ij} \right)}}{\sum_{k \neq i} e^{\pi_i \left( \min_{l \neq i}(\delta^2_{il}) - \delta^2_{ik} \right)}} & \end{cases} \qquad \overline{\delta}^*_{ij} = \begin{cases} 0 & (\text{if } i = j) \\ \delta^2_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{\pi_i \left( \min_{l \neq i}(\delta^2_{il}) - \delta^2_{ik} \right)} \right) & (\text{if } i \neq j) \end{cases}$$

From these formulas, one can observe that the final distance $\overline{\delta}$ and $\overline{\delta}^*$ are, respectively, the simple and squared Euclidean distances, imputed with a value, specific to each point.

## 2.1 Perplexity

To determine the values of $\pi_i$, we first need to explain the concept of perplexity ($\mathcal{P}$). Perplexity can be interpreted as a measure of the effective number of neighboring points. It is often referred to as 'bandwidth' and has a significant impact on the final results.

The similarity $\sigma_{ij}$ can be considered a probability following a Laplacian or Normal distribution as explained previously. These variables require then a variance, which is inferred by the $\pi_i$ values, as one can observe on the formulas of $\sigma_{ij}$ and $\sigma^*_{ij}$ presented above. Their variances are respectively $\frac{2}{\pi_i^2}$ and $\frac{2}{\pi_i}$.

Figure 2.1 illustrates the importance of $\pi_i$ on the probabilities $\sigma^*_{ij}$. For instance, purple and orange points are negligible with $\pi_i = 2$, while they are not with $\pi_i = \frac{2}{3}$. Consequently, purple and red points are considered equidistant, essentially at an infinite distance, in the first case but not in the second.
For our new distance metric, we aim to achieve the smallest possible perplexity. The objective is to consider close local relations as strictly as possible and then determine the optimal global relations using a shortest path algorithm.

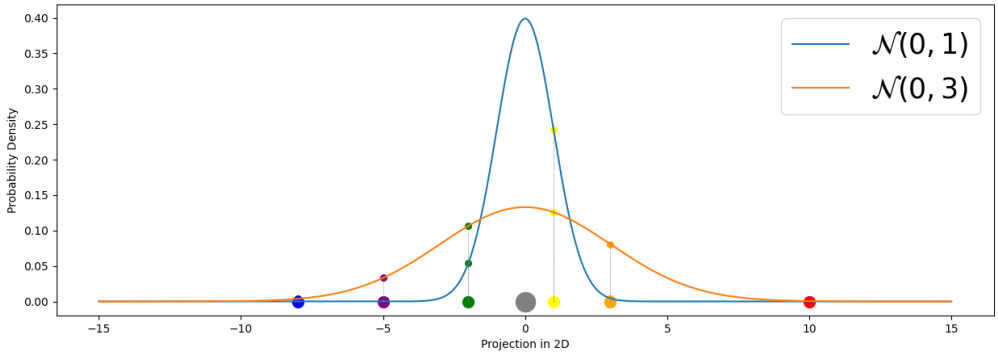



Figure 2.1: Example of SNE probability $\sigma^*$; the observed point, in gray, is the reference point ($i$), while the other points ($j$'s) are placed on the x-axis at a high-dimensional distance $d_{ij}$ from $i$, $\sigma^*_{ij}$ values are computed based on their corresponding normal density with a variance $\frac{2}{\pi_i}$

Now that we understand the desired perplexity and the role of the $\pi_i$ values, we need a method to set the $\pi_i$ values to achieve the target perplexity. This can be done by examining the formula involving perplexity $\mathcal{P}$;

$$\log(\mathcal{P}) = -\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) \ .$$

In essence, this equation describes the entropy, or the amount of information, contained in $\sigma_{ij}$'s for each point $i$, which can also be interpreted as probabilities $p_{ij}$'s as previously mentioned. The concept of entropy and perplexity are intuitively linked: entropy refers to the 'amount of information', while perplexity refers to the 'number of neighbors taken into account'.

Since $\pi_i$ appears in $\sigma_{ij}$, given a value for $\mathcal{P}$, we are left with an equation to solve. This equation being challenging to solve analytically, we use a simple binary search to find the $\pi_i$ values. Alternatively, the Newton-Raphson method could be employed.

Next, let us consider the possible range of values for perplexity. One way to approach this is through analytical analysis. In this part, $\sigma$ and $\sigma^*$ are interchangeable.

First, let us revisit the $\sigma_{ij}$ values. It is important to recall that they also represent the probability that point $i$ selects $j$ as its neighbor, $i$ being prohibited to take $i$. This implies two primary rules and three secondary rules for the following analysis:

1. $0 \leq \sigma_{ij} \leq 1$ $\quad \forall i \neq j$
2. $\sum_{i \neq j} \sigma_{ij} = 1$ $\quad \forall i$
3. $\log(\sigma_{ij}) \leq 0$ $\quad \forall i \neq j \quad (1 \implies 3)$
4. $\sigma_{ij} \log(\sigma_{ij}) \leq 0$ $\quad \forall i \neq j \quad (1, 3 \implies 4)$
5. $\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) \leq 0$ $\quad \forall i \quad (4 \implies 5)$

Let us explore each possible range of values perplexity can take.

### $\mathcal{P} \leq 0$

$$\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) = -\log(\mathcal{P} \leq 0)$$

Here, $\log(x)$ is only defined for $x > 0$, making this equality unattainable.

### $0 < \mathcal{P} < 1$

$$\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) = -\log(0 < \mathcal{P} < 1)$$

Knowing $-\log(0 < \mathcal{P} < 1) > 0$ and Rule 5, we can conclude that this equality is impossible.

### $\mathcal{P} = 1$

$$\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) = -\log(\mathcal{P} = 1) = 0$$

With Rule 4, we know that $\sigma_{ij} \log(\sigma_{ij})$ will have the same sign for all $i \neq j$. Consequently, the only way to achieve $\sum_{i \neq j} \sigma_{ij} \log(\sigma_{ij}) = 0$ is if $\sigma_{ij} \log(\sigma_{ij}) = 0 \ \forall i, j$.
There is only one way to achieve this: $(\sigma_{ij} = 0$ or $\sigma_{ij} = 1)$[1] $\forall i \neq j$.
With Rule 2, we know that the only way to have this is to have $\sigma_{ij} = 1$ for one and only one $j$, and $\sigma_{ij} = 0$ for the other $j$'s different from $i$. Beyond the fact that it would give us a distance matrix very poor in terms of information, it would result in, for each point, a second point $j$ at a distance 0 from $i$, in addition to $i$ itself. Indeed, a similarity of 1 gives a distance of 0. This is noticeable with the formula $\overline{\delta}_{ij} = -\frac{1}{\pi_i} \log(\sigma_{ij} = 1) = 0$. Having two different points at a distance 0 would superimpose them and violate one of the properties a metric must satisfy, positivity, as explained in Section 2.3.1.

### $\mathcal{P} > 1$

$$\sum \sigma_{ij} \log(\sigma_{ij}) = -\log(\mathcal{P} > 1) < 0$$

With Rule 5, we see that this is now possible. None of the other rules pose a problem. Moreover, we must have at least one $0 < \sigma_{ij} < 1$ to have a strictly negative value for $\sum \sigma_{ij} \log(\sigma_{ij}) = -\log(\mathcal{P} > 1) < 0$, as shown in the previous part. That, along with Rule 2, prevents us from having a $\sigma_{ij} = 1$ and then two null distances for the same point.

[1] $\lim_{x \to 0} x \log(x) = 0$

**Conclusion**

As mentioned earlier, we want a perplexity as small as possible, while respecting conditions just discussed. That is why $\mathcal{P} = 1 + \epsilon$ could be a good choice and is the principal $\mathcal{P}$ value emphasized in this thesis.

## 2.2 Shortest Path Algorithm

There are many different algorithms [15] for computing the shortest paths in a graph. To choose the best one for our case, let us examine the complexity of some.
First, it is important to note that we are working on complete graphs, where our sneodesic distance between each pair of points is computed and considered. That means if we have $n$ vertices, representing the $n$ points of the dataset, we will have $\frac{n(n-1)}{2}$ edges.

| Algorithm | Complexity | Complexity (complete graph) |
|---|---|---|
| Dijkstra's Algorithm | $O(\|V\|(\|V\| + \|E\|) \log \|V\|)$ | $O(\|V\|^3 \log \|V\|)$ |
| Bellman-Ford Algorithm | $O(\|V\|^2\|E\|)$ | $O(\|V\|^4)$ |
| Floyd-Warshall Algorithm | $O(\|V\|^3)$ | $O(\|V\|^3)$ |
| Johnson's Algorithm | $O(\|V\|^2 \log \|V\| + \|V\|\|E\|)$ | $O(\|V\|^3 + \|V\|^2 \log \|V\|)$ |

Table 2.1: Some of the most common shortest path algorithms

From Table 2.1, it is clear that the Floyd-Warshall Algorithm is the best choice when we have a dense (here, complete) graph. Moreover, it is an algorithm designed to compute the distance between each pair of points at once and most applications necessitate the whole distance matrix. Additionally, it is easy to use, code, and understand.

The pseudocode of this algorithm is as follows:

```
1: procedure FloydWarshall(dist(i, j, 0))
2:     for k ← 1 to n do
3:         for i ← 1 to n do
4:             for j ← 1 to n do
5:                 dist(i, j, k) = min(dist(i, j, k − 1), dist(i, k, k − 1) + dist(k, j, k − 1))
6:             end for
7:         end for
8:     end for
9: end procedure
```

The input is the initial distance matrix, which is modified during the computation of the algorithm, having then also the great property to be in-place. The final distance $d(i, j)$, after the Floyd-Warshall algorithm, is then $\text{dist}(i, j, n)$.

Now we need to decide how to combine the distance segments. Several different methods have been tested, so let us explain the most successful ones.

*Recall: $\delta_{ij}$ is the Euclidean distance between $i$ and $j$*

### 2.2.1 Simple Addition

This is the simplest way to compute the length of the shortest distance. It involves adding up all the $\overline{\delta}$ values assigned to each edge that is part of the path. This way strictly respects the pseudocode, accurately its line `5`. Most of the distance metrics further presented, as *sneoNS* and *sneoS*, are in this category.

#### sneoNS

*Stands for: sneodesic distance, Non-Squared*

$$\overline{\delta}_{ij} = \delta_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{-\pi_i \delta_{ik}} \right) \qquad (\text{if } i \neq j)$$

$$sneoNS(x, y) = \sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$$

#### sneoS

*Stands for: sneodesic distance, Squared*

$$\overline{\delta}^*_{ij} = \delta^2_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{-\pi_i \delta^2_{ik}} \right) \qquad (\text{if } i \neq j)$$

$$sneoS(x, y) = \sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}^*_{ij}$$

### 2.2.2 Shortcut Path

We can also use squared lengths of the path. The advantage is that it permits us to apply the well-known Pythagorean theorem. However, to use it correctly, we need the angles to be right. If we assume all the angles being part of the paths are right, we can add up the squared distances. It will create a shortcut through the path, as shown in Figure 2.2.

#### sneoSQ

*Stands for: sneodesic distance, Squared, Quadratic addition*

$$\overline{\delta}^*_{ij} = \delta^2_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{-\pi_i \delta^2_{ik}} \right) \qquad (\text{if } i \neq j)$$

$$sneoS(x, y) = \sqrt{\sum_{ij \in \text{ShortestPath}(x,y)} (\overline{\delta}^*_{ij})^2}$$

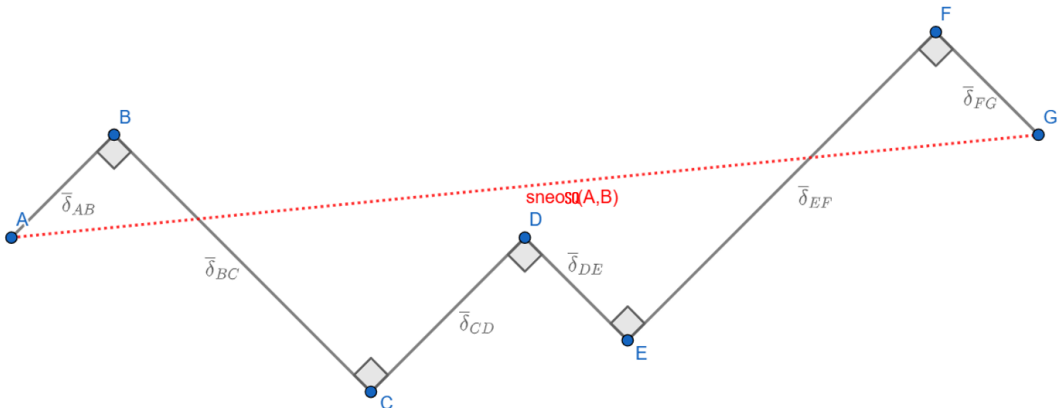



Figure 2.2: Shortcut (in red) of the path (in black) computed using quadratic sum, supposing points lie in dimension $d \geq n-1$

This formula is correct if the intrinsic dimension is larger than the number of segments in the path. Indeed, we can see the total distance as the diagonal of the rectangular parallelepiped in $d$ dimensions.
In a nutshell, the intrinsic dimension of a dataset refers to the minimum number of variables needed to accurately describe the underlying structure of the data. It represents the true dimensionality of the manifold on which the data lies, which can be much lower and only lower than the dimensionality of the original feature space.

Furthermore, if the intrinsic dimension is not large enough, or the angles are not mostly right, we can use the generalized Pythagorean theorem in *sneoNSCos*. It does not make a straight shortcut as *sneoSQ* does, but it remains more straightforward than the zigzags taken by simple additions.

### sneoNSCos

*Stands for: sneodesic distance, Non-Squared Cosine sums*

$$\overline{\delta}^*_{ij} = \delta^2_{ij} + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta^2_{ik}}\right) \qquad (\text{if } i \neq j)$$

$$sneoNCos(x,y) = \sum_{ij,jk \in \text{ShortestPath}(x,y)} \sqrt{\overline{\delta}^*_{ij} + \overline{\delta}^*_{jk} - 2\sqrt{\overline{\delta}^*_{ij}}\sqrt{\overline{\delta}^*_{jk}}\cos(\widehat{ijk})}$$

Shortcuts, shown in Figure 2.3, are much less significant than for *sneoSQ*. They do not take advantage of the other dimensions; they are all confined to 2 dimensions. In contrast, for *sneoSQ*, they benefit from them because it considers the "hyper-hypotenuse".
Note that if the number of points is odd, we simply add the length of the last segment as it is impossible to build another triangle.

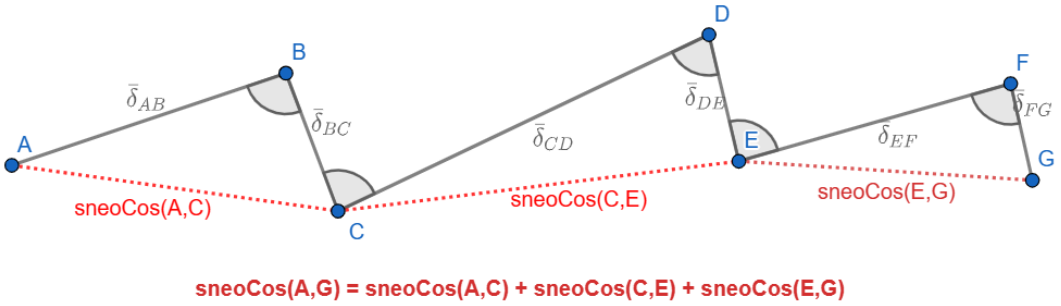



Figure 2.3: Shortcuts taken using cosine sum, points lie in $d$ dimensions

One can observe in Figure 2.4 that the distance computed by *sneoSQ* is much more direct than *sneoNSCos*.

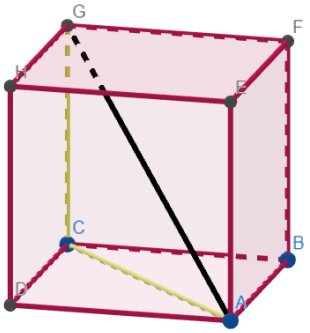



Figure 2.4: Difference between $sneoNSCos(A, G)$ (yellow) and $sneoSQ(A, G)$ (black)

Distance metrics using $\overline{\delta}^*$ with non-quadratic sums, such as *sneoS*, can be considered as hybrid. The fact that these sneodesic distances are the squared euclidean distances imputed with a value keeps somehow the quadratic nature of these. They could then also take advantage of angles concentrated around 90°.

## 2.3 Distance properties

Since the beginning, we consider we manipulate a distance, but is it strictly one ? Basically, a distance indicates a relationship between two data points.
The most restrictive class of distance is defined by the four followings conditions. If it respects all of them, it is considered as a *metric* [16].

### 2.3.1 Positivity

$\mathbf{d(x, y) > 0} \quad \forall \mathbf{x \neq y}$

---

*Proof.* We prove by induction on $k$.
By definition, $dist(i, i, 0) = \overline{\delta}_{ii} = 0 \ \forall i$ and $dist(i, j, 0) = \overline{\delta}_{ij} > 0 \ \forall i \neq j$

**Base case:** $k = 1$

$$\begin{aligned} \forall i \neq j : dist(i, j, 1) &= \min\left(dist(i, j, 0), dist(i, 1, 0) + dist(1, j, 0)\right) \\ &= \min\left(\overline{\delta}_{ij}, \overline{\delta}_{i1} + \overline{\delta}_{1j}\right) \\ &> 0 \qquad (\text{as } \ i \neq 1 \ \text{ or } \ j \neq 1) \end{aligned}$$

by application of Floyd-Warshall
**Inductive Step:**

$$\begin{aligned} \forall i \neq j : \ & i \neq k \ \text{ or } \ j \neq k \\ & \implies dist(i, k, k-1) + dist(k, j, k-1) > 0 \\ & \implies dist(i, j, k) = \min\left(dist(i, j, k-1), dist(i, k, k-1) + dist(k, j, k-1)\right) > 0 \end{aligned}$$

Therefore,

$$dist(i, j, k) > 0 \ \ \forall i \neq j, k \implies dist(i, j, n) > 0 \ \ \forall i \neq j \iff d(x, y) > 0 \ \ \forall x \neq y$$

□

---

### 2.3.2 Nullity

$\mathbf{d(x, x) = 0} \quad \forall \mathbf{x}$

---

*Proof.* We prove by induction on $k$.
By definition, $dist(i, i, 0) = \overline{\delta}_{ii} = 0 \ \forall i$ and $dist(i, j, 0) = \overline{\delta}_{ij} > 0 \ \forall i \neq j$

**Base Case :** $k = 1$

$$\begin{aligned} \forall i : dist(i, i, 1) &= \min\left(dist(i, i, 0), dist(i, 1, 0) + dist(1, i, 0)\right) \\ &= \min\left(\overline{\delta}_{ii}, \overline{\delta}_{i1} + \overline{\delta}_{1i}\right) \\ &= 0 \end{aligned}$$

by application of Floyd-Warshall
**Inductive steps :**

$$\begin{aligned} & dist(i,i,k) = \min\left(dist(i,i,k-1), dist(i,k,k-1) + dist(k,i,k-1)\right) \quad \forall i \\ \implies & dist(i,i,k) = \min\left(0, \alpha \geq 0 + \beta \geq 0\right) \quad \forall i \qquad \text{(by 2.3.1)} \\ \iff & dist(i,i,k) = 0 \quad \forall i \end{aligned}$$

Therefore,

$$dist(i,i,k) = 0 \;\; \forall i,k \implies dist(i,i,n) = 0 \;\; \forall i \iff d(x,x) = 0 \;\; \forall x$$

□

---

### 2.3.3 Symmetry

$\mathbf{d(x,y) = d(y,x)} \quad \mathbf{\forall x, y}$

---

*Proof.* We prove by induction on $k$.
By definition, $dist(i,j,0) = \overline{\delta}_{ij} = \overline{\delta}_{ji} \;\; \forall i,j$ as we symmetrize it by doing $\frac{\overline{\delta}^T + \overline{\delta}}{2}$ before giving it as input to the Floyd-Warshall algorithm
**Base Case :** $k = 1$

$$\begin{aligned} \forall i,j : dist(i,j,1) &= \min\left(dist(i,j,0), dist(i,1,0) + dist(1,j,0)\right) \\ &= \min\left(\overline{\delta}_{ij}, \overline{\delta}_{i1} + \overline{\delta}_{1j}\right) \\ &= \min\left(\overline{\delta}_{ji}, \overline{\delta}_{1i} + \overline{\delta}_{j1}\right) \\ &= \min\left(dist(j,i,0), dist(1,i,0) + dist(j,1,0)\right) \\ &= \min\left(dist(j,i,0), dist(j,1,0) + dist(1,j,0)\right) \\ &= dist(j,i,1) \end{aligned}$$

by application of Floyd-Warshall
**Inductive steps :**

$$\begin{aligned} \forall i,j : dist(i,j,k) &= \min\left(dist(i,j,k-1), dist(i,k,k-1) + dist(k,j,k-1)\right) \\ &= \min\left(dist(j,i,k-1), dist(k,i,k-1) + dist(j,k,k-1)\right) \\ &= \min\left(dist(j,i,k-1), dist(j,k,k-1) + dist(k,i,k-1)\right) \\ &= dist(j,i,k) \end{aligned}$$

Therefore,

$$\begin{aligned} dist(i,j,k) = dist(j,i,k) \;\; \forall i,j,k \implies & dist(i,j,n) = dist(j,i,n) \;\; \forall i,j \\ \iff & d(x,y) = d(y,x) \;\; \forall x,y \end{aligned}$$

□

---

### 2.3.4 Triangle Inequality

$\mathbf{d(x,y) \leq d(x,z) + d(z,y) \quad \forall x, y, z}$

---

*Proof.* We prove by induction on $k$.
**Base Case :** $k = 1$

$$\begin{aligned} \forall i, j : dist(i, j, 1) &= \min\left(dist(i, j, 0), dist(i, 1, 0) + dist(1, j, 0)\right) \\ &\leq dist(i, 1, 0) + dist(1, j, 0) \end{aligned}$$

by application of Floyd-Warshall
**Inductive steps :**

$$\begin{aligned} dist(i, j, k) &= \min\left(dist(i, j, k-1), dist(i, k, k-1) + dist(k, j, k-1)\right) \quad \forall i, j \\ &\leq dist(i, k, k-1) + dist(k, j, k-1) \end{aligned}$$

Therefore,

$$\begin{aligned} dist(i, j, n) &\leq dist(i, q, n) + dist(q, j, n) \quad \forall i, j, q \leq n \\ \iff d(x, y) &\leq d(x, z) + d(z, y) \quad \forall x, y, z \end{aligned}$$

□

---

Consequently, our sneodesic distance metric can well be considered as *metrics*, making it consistent and meaningful.
However, note that this does not hold for *sneoNSCos* and *sneoSQ* as the *Floyd-Warshall* step does not verify exactly the same equation. Nevertheless, it holds for any other types of sneodesic distance metrics presented in this thesis, whether it is based on $\sigma$ or $\sigma^*$.

# Chapter 3

# Methodology

Evaluating the performance of distance measures presents a significant challenge because the exact structure of a dataset is often unknown, as the theoretical relations between points. In high-dimensional spaces, it becomes particularly difficult to understand how distances should ideally function, as visualization is not possible. Different distance metrics might capture different aspects of the data's structure, making it hard to determine a universally effective metric. Moreover, the effectiveness of a distance metric can vary depending on the specific characteristics of the dataset and the methods used to assess performance.

To address these challenges, we will explore three performance assessment methods, each offering a unique perspective on evaluating distance metrics. Additionally, we will apply these methods to eight diverse datasets, ranging from low to high dimensions, to understand how different metrics perform across various data. Furthermore, we will examine the interaction between distance metrics and dimensionality reduction techniques by utilizing two distinct dimensionality reduction methods. This approach aims to provide a thorough evaluation framework for distance metrics, highlighting their strengths and weaknesses in different contexts.

## 3.1 Performance Assessment Methods

### 3.1.1 Neighborhood Study

The first method consists in neighborhood class study. More precisely, we will study the k-NN classification algorithm. For each point in the dataset, we analyze the class of its $k$ nearest neighbors. If the distance metric is effective, neighboring points are expected to belong to the same class as the point itself. We calculate the percentage of points belonging to the same class as the point itself among its $k$ nearest neighbors and compute the mean of this value over all points in the dataset. This gives us a graph showing the mean percentage as a function of $k$.

Consider a simple example assessing the performance of the Euclidean distance with the dataset presented in Figure 3.1.

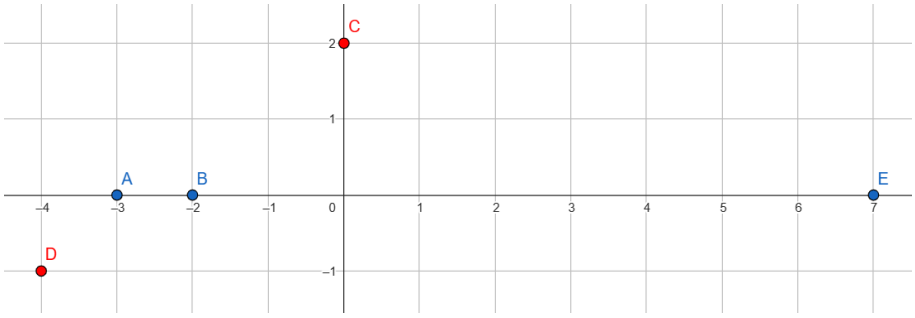



Figure 3.1: Example of a dataset with 5 points in 2 dimensions, belonging to 2 classes, represented by color

First, when looking at just one point, the point $A$, the results evolve as shown in Table 3.1.

| $k$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Point Encountered** | B | D | C | E |
| **Same Class** | yes | no | no | yes |
| **Fraction** | 1 | 0.5 | 0.33 | 0.5 |

Table 3.1: Evolution of the fraction of points belonging to the same class as $A$, as a function of $k$

To get a global idea of the whole dataset, we compute the average of each total, as shown in Figure 3.2. This provides insight into performance in the neighborhood as well as in the global structure. Here, the average value is 0.375, or, as written on the graph provided, 37.5%. The larger this value, the better our distance metric works globally. This total value will always be between 0 and 100. The graph allows us to observe how performance evolves as a function of $k$.

| k | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **A** | 1 | 0.5 | 0.33 | 0.5 |
| **B** | 1 | 0.5 | 0.33 | 0.5 |
| **C** | 0 | 0 | 0.33 | 0.25 |
| **D** | 0 | 0 | 0.33 | 0.25 |
| **E** | 0 | 0.5 | 0.66 | 0.5 |
| **total** | 0.4 | 0.3 | 0.4 | 0.4 |

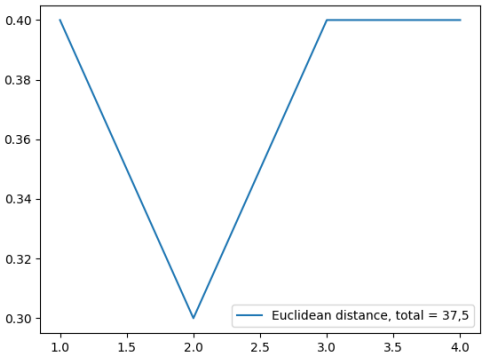



Figure 3.2: Complete performance table and graph showing the evolution of the percentage as a function of $k$, with the total performance value

### 3.1.2 2D-Plot

Another method to evaluate the performance of a distance metric is to apply a dimensionality reduction technique on the distance matrix and project the coordinates into 2 dimensions. Points are colorized based on their class. Although this method does not provide a numerical value, we can expect that the better the distance metric, the more distinguishable and clearly separated the categories should appear.

In Figure 3.3, two examples of t-SNE reduction applied with different distance metrics on the *MNIST* dataset are shown.

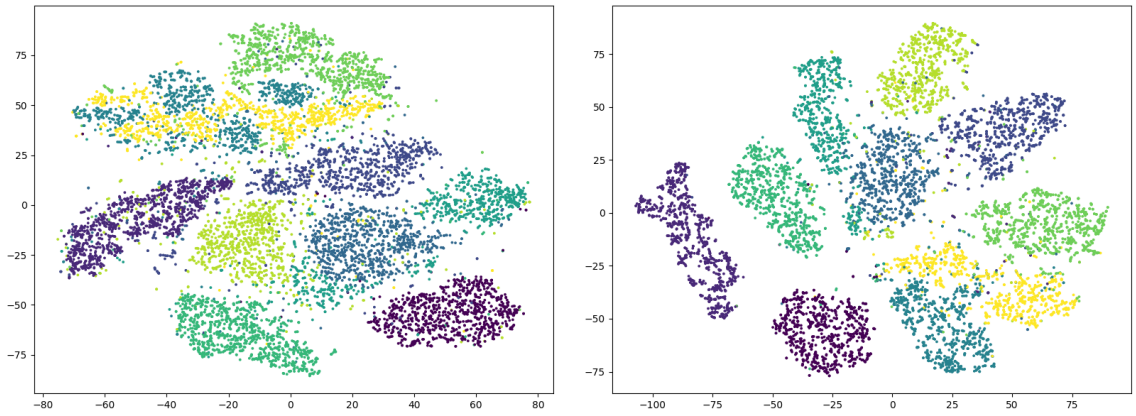



Figure 3.3: t-SNE reduction of 10'000 samples of the *MNIST* dataset, based on their Mahalanobis[1] (left) and *sneoNS* (right) distance matrices

From these graphs, it is evident that the dimensionality reduction performs better in the second example. Different colors are more separated, and clusters do not mix as much as in the first one. From these visual representations, we can infer that the *sneoNS* distance is more effective than the Mahalanobis distance for this specific application.

However, it is important to note that we cannot conclusively determine that the second distance metric is significantly better than the first one. The 2D representation is also influenced by the dimensionality reduction method and the chosen dataset.

### 3.1.3 $Q_{NX}(K)$

Here, we plot the values of $Q_{NX}(K)$ [17]. To do so, we also need to go through a dimensionality reduction as the formula to compute them is

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

where $\nu_i^K$ and $n_i^K$ represent the $K$ nearest neighbors to point $i$ in high and low dimensions (after dimensionality reduction), respectively, and $N$ is the total number of points in the dataset.

[1] Here, we use the estimation approach, involving PCA decomposition into 50 dimensions.

An important metric is the AUC (Area Under the Curve), computed using the formulas

$$R_{NX}(K) = \frac{(N-1)Q_{NX}(K) - K}{N-1-K} \quad \text{and} \quad \text{AUC} = 100\frac{\sum_{K=1}^{N-2} R_{NX}(K)/K}{\sum_{K=1}^{N-2} 1/K} \ .$$

The AUC provides a final metric, ranging from 0 to 100, for the entire dataset, while $Q_{NX}(K)$ allows comparison of the performance at different neighborhood scales. The effectiveness of our distance metric is indicated by the magnitude of $Q_{NX}(K)$ and AUC.

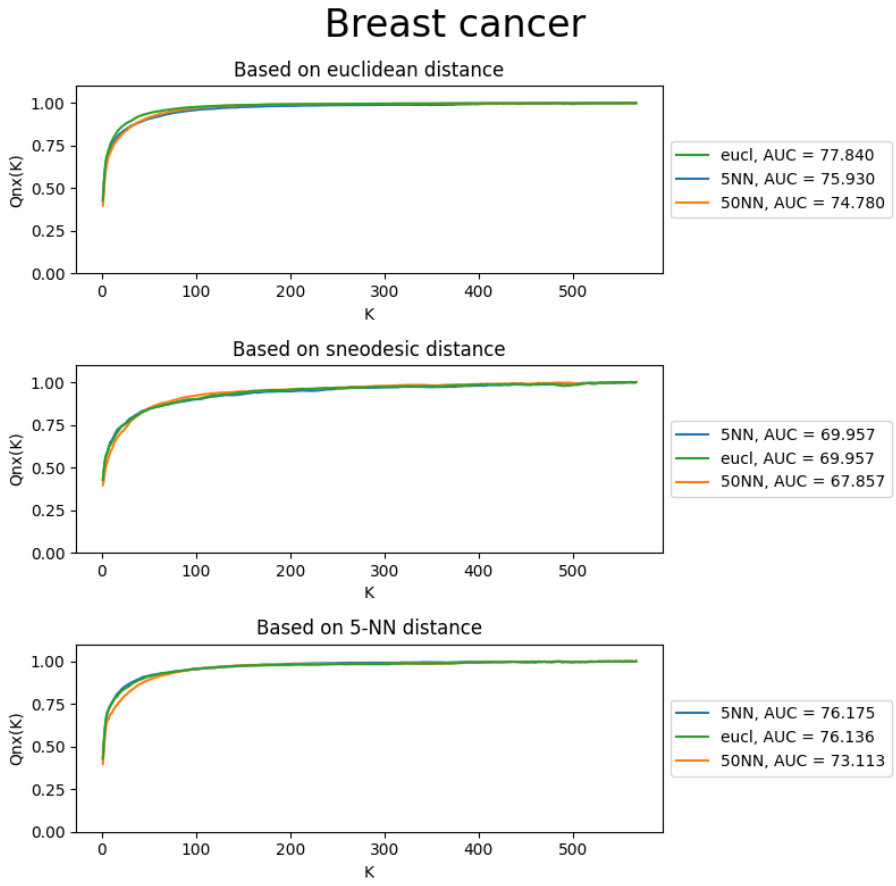



Figure 3.4: Example of the $Q_{NX}(K)$ curves, on the dataset *Breast Cancer*, comparing 3 distance metrics : Euclidean, 5-NN, and 50-NN distances

An example is shown in Figure 3.4. There are three different graphs. The reason behind this is that, to compute the performance between high and low dimensions, we also need a distance metric to determine the K nearest neighbors in those dimensions. If we inspect only one distance, say the Euclidean distance, it is more likely that dimensionality reduction using this same Euclidean distance will perform better, as the objective of the dimensionality reduction technique is to preserve this distance matrix as best as possible.

Let us clarify with an example: the orange line in the third graph of Figure 3.4 represents $Q_{NX}(K)$ with neighboring points in high and low dimensions computed with the *5-NN* distance, with a dimensionality reduction method applied on the *50-NN* distance matrix. The x-axis ranges from 0 to 569 because 569 samples are available in the *Breast Cancer* dataset, and the y-axis ranges from 0 to 1 because it represents the fraction of correct neighbors. $Q_{NX}(N) = 1$, with $N$ being the total number of points, because the entire dataset becomes part of the neighborhood, thus the neighborhood in high dimension is exactly the same as the one in low dimension. The legend is ordered by AUC to have a visual ranking of distance performances.

### 3.1.4 Strengths

Now that we have presented three methods to compare the performances of our different distance metrics, let us examine the strengths we would like them to possess. Table 3.2 summarizes the strengths of each method. In this context, the absence of a strength can be considered a weakness.

#### Class independence

The ability to compare the performance of a distance metric with others, without considering the classes the points belong to is a strength. We do not necessarily know the intrinsic properties of the data related to their classes. It is then risky to assess the performance of a distance metric by looking solely at classes.

#### Dimension preservation

Methods that avoid dimensionality reduction are preferable to assess distance measure performances to prevent introducing biases from the dimensionality reduction algorithms.

#### Global structure

It is important to compare the performance on both neighboring and distant points.

#### Total value

A floating-point total value makes the comparison of different results easier.

| | Class independence | Dimension preservation | Global structure | Total value |
|---|---|---|---|---|
| Neighborhood study | | ✓ | ✓ | ✓ |
| 2D-Plot | | | ✓ | |
| $Q_{NX}(K)$ | ✓ | | ✓ | ✓ |

Table 3.2: Comparison of the different performance assessment methods through their strengths

## 3.2 Datasets

Now that we have methods to evaluate a new distance metric, we need datasets to produce empirical results. Followings are all the datasets inspected within the experiments, with their characteristics and a brief description. They are sorted by increasing number of dimensions. Datasets with more than 10,000 samples available have been shuffled and reduced to this number due to the high computational cost of methods used (distance matrix computation, dimensionality reduction, etc.). This set of datasets was chosen to explore a wide variety of dimensions, structures, and numbers of classes.

### Breast Cancer

Number of dimensions : 30
Number of points : 569
Number of classes : 2
Diminutive : *breastcancer*

This dataset [18] comprises 30 continuous features extracted from digitized images of breast mass fine needle aspirate (FNA), primarily describing cell nucleus characteristics. Features describe, for instance, radius, texture, perimeter, etc. Cells are classified in two categories ; benign or malignant.

### Mouse

Number of dimensions : 50
Number of points : >10,000
Number of classes : 133
Diminutive : *mouse*

This [19] comprises transcriptomic profiles of 23,822 cells sampled from two distinct regions of the mouse neocortex: the primary visual cortex and the anterior lateral motor cortex. Through deep, single-cell RNA sequencing, researchers identified 133 transcriptomic cell types,

### Zebrafish

Number of dimensions : 60
Number of points : >10,000
Number of classes : 7
Diminutive : *zebrafish*

This dataset [20] contains over 90,000 zebrafish embryos within the first day of development classified into 7 categories based on their 60-long RNA sequences.

### MNIST

Number of dimensions : 784
Number of points : >10,000
Number of classes : 10
Diminutive : *mnist*

This database [3] consists of a collection of handwritten digit images one can observe in Figure 3.5a. It includes a training set of 60,000 examples and a test set of 10,000 examples. Each digit image has been size-normalized to fit within a 20x20 pixel box while preserving its aspect ratio, resulting in 28x28 pixel images with grey levels. Additionally, the images are centered by computing the center of mass of the pixels and translating the image to position this point at the center of the 28x28 field.

### Cifar-10

Number of dimensions : 3072
Number of points : >10,000
Number of classes : 10
Diminutive : *cifar10*

This dataset [21] consists of 60,000 32x32 pixels colored (RGB) images showing 10 different classes being airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Classes are not overlapping. Images are not normalized, not centered, have different backgrounds, etc. Images belonging to the same category can then have completely different representations, making this dataset difficult to classify with basic methods. One can observe examples in Figure 3.5b where each line represents one and only one class.

### Cifar-100

Number of dimensions : 3072
Number of points : >10,000
Number of classes : 20
Diminutive : *cifar100*

It is basically the same description as *Cifar-10*. The only difference is that there are 20 different classes.

### Coil-20

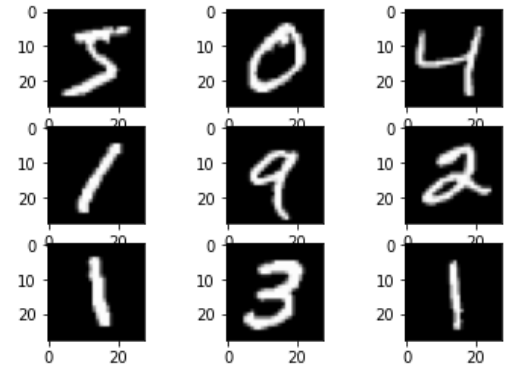
Number of dimensions : 16,384
Number of points : 1440
Number of classes : 20
Diminutive : *coil20*

This dataset [4] consists of 128x128 black and white images representing objects rotating 360°. There are 20 different objects, each pictured every 5°, resulting in 72 pictures per object and a total of 1440 elements, with $128 \times 128 = 16,384$ dimensions. The background has been discarded and the frame is placed so that it builds the smallest square containing the object, as one can observe in Figure 3.5c.

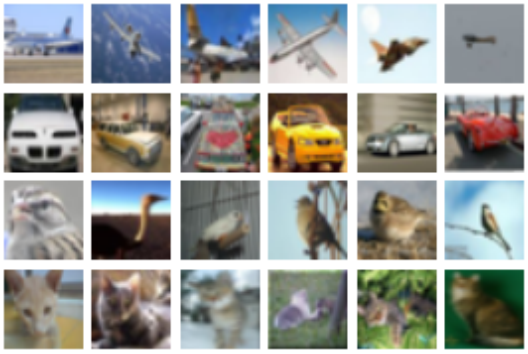
### Coil-100

Number of dimensions : 49,152
Number of points : 7200
Number of classes : 100
Diminutive : *coil100*

It [22] is very similar to Coil-20 but with 100 objects and color images (implying three times more dimensions), resulting in 7200 points in 49,152 dimensions. Some samples can be inspected in Figure 3.5d.



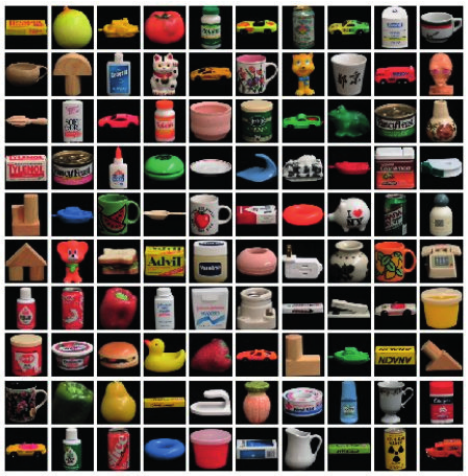
(a) *MNIST* [3]



(b) *Cifar* [21]




(c) *Coil-20* [4]



(d) *Coil-100* [23]

Figure 3.5: Samples of the image datasets

## 3.3 Distance metrics

In Table 3.3, one can see details about all distance metrics used in the experiments. If no perplexity is given for a sneodesic distance metric, it is equal to $1 + \epsilon$ . To compute the results, a value of 1.001 has been chosen.

Other distance metrics have been tried as the *Mahalanobis* distance explained in Section 1.1.2, cosine dissimilarities [24], 2-dimensional Pythagorean theorem to compute path lengths, etc. Here are only the attempts producing results competing with the ones of best sneodesic performances.

| **Short Name** | **Complete Name** | $\overline{\boldsymbol{\delta}}_{\mathbf{ij}} \quad \forall \mathbf{i} \neq \mathbf{j}$ | $\mathbf{d(x,y)} \quad \forall \mathbf{x} \neq \mathbf{y}$ |
|---|---|---|---|
| eucl | Euclidean distance | | $\delta_{ij}$ |
| kNN | k-NN (or geodesic) distance | $\delta_{ij}$ | $\sum_{ij \in \text{ShortestPath}(x,y) \text{ in k-NN graph}} \overline{\delta}_{ij}$ |
| sneoNSS | Sneodesic distance, Non-Squared, Square-rooted | $\delta_{ij}^2 + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}^2}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \sqrt{\overline{\delta}_{ij}}$ |
| sneoNS | Sneodesic distance, Non-Squared | $\delta_{ij} + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoNS$p$ | Sneodesic distance, Non-Squared, perplexity = $p$ | $\delta_{ij} + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoNSCst | Sneodesic distance, Non-Squared, Constant | $\delta_{ij} + 10^{10} \log\left(\sum_{k \neq i} e^{-10^{-10} \delta_{ik}}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoNSCos | Sneodesic distance, Non-Squared, Cosinus | $\delta_{ij}^2 + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}^2}\right)$ | $\sum_{ij,jk \in \text{ShortestPath}(x,y)} \sqrt{\overline{\delta}_{ij}^2 + \overline{\delta}_{jk}^2 - 2\sqrt{\overline{\delta}_{ij}^2}\sqrt{\overline{\delta}_{jk}^2}\cos(\widehat{ijk})}$ |
| sneoS | Sneodesic distance, Squared | $\delta_{ij}^2 + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}^2}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoS$p$ | Sneodesic distance, Squared, perplexity = $p$ | $\delta_{ij}^2 + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}^2}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoSCst | Sneodesic distance, Squared, Constant | $\delta_{ij}^2 + \frac{1}{10^{10}} \log\left(\sum_{k \neq i} e^{-10^{-10} \delta_{ik}^2}\right)$ | $\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}$ |
| sneoSQ | Sneodesic distance, Squared, Quadratic addition | $\delta_{ij}^2 + \frac{1}{\pi_i} \log\left(\sum_{k \neq i} e^{-\pi_i \delta_{ik}^2}\right)$ | $\sqrt{\sum_{ij \in \text{ShortestPath}(x,y)} \overline{\delta}_{ij}^2}$ |

Table 3.3: Summary of all distance metrics used within empirical experiments, with $\delta_{ij}$ being the Euclidean distance between point $i$ and $j$

## 3.4 Dimensionality Reduction Methods

### MDS

In our experiments, we execute the metric MDS algorithm provided by `sklearn` and explained in Section 1.2.2 using this exact line of code:

```
MDS(n_components=2, dissimilarity='precomputed',normalized_stress=False,
max_iter=150, eps=0.01)
```

Here, `n_components` specifies the dimensionality of the output space (set to 2 for visualization purposes), `dissimilarity` indicates that a precomputed dissimilarity matrix is used as input, and `normalized_stress` determines whether stress values are normalized. The `max_iter` parameter limits the maximum number of iterations to 150, and `eps` sets the tolerance for the convergence criterion. These last parameter choices are made to balance computational efficiency with convergence accuracy, particularly considering the potential computational complexity of the algorithm when dealing with datasets containing up to 10,000 points.

This method has been chosen for future experiments to represent the 'simple' dimensionality reduction approach. It is a good way to compare different distance matrices as it does not make any pre-assumption on the kind of distances given as input and it simply attempts to conserve the initial distance matrix as intact as possible.

### t-SNE

*t-SNE* has been chosen as the second method of dimensionality reduction. It is explained at the end of Section 1.2.3. The exact line of code used is:

```
TSNE(metric="precomputed", init='random', n_components=2)
```

Once again, we indicate that we use `metric="precomputed"` to have the freedom to provide any distance matrix we want. The `n_components` argument sets the dimensionality of the embedded space. Additionally, the `init` parameter determines the initialization strategy; `random` initializes the embeddings randomly.

This method has been chosen due to its performance, notably producing some impressive results in 2D representations, which can be observed later in this thesis.

# Chapter 4

# Experiments and Results

## 4.1 Introductory Example

Let us consider a simple example before discussing results on bigger datasets. Figure 4.1 shows the *sneoNS* distances between pairs of points before applying the Floyd-Warshall algorithm. The edges represent the graph on which this algorithm will operate. *sneoNS* has been chosen as it is based on the simple Euclidean distance, facilitating comprehension.

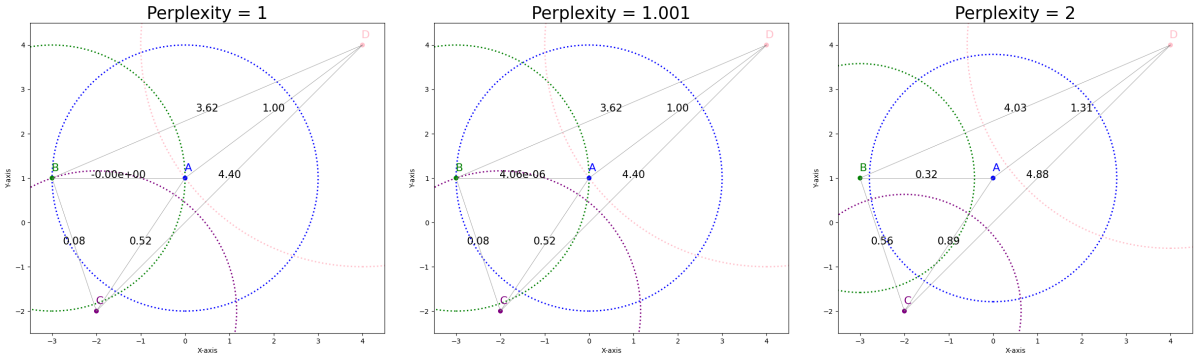



Figure 4.1: Examples of the *sneoNS* distance before applying the shortest path algorithm, in 2D with 4 points and various perplexities: 1 (left), 1 + $\epsilon$ (center), and 2 (right)

In the figures with perplexities 1 and 1.001, the circles appear similar but are not identical. The radii of these circles represent the value subtracted to the Euclidean distance. It is the second term of the sneodesic equation presented earlier;

$$\overline{\delta}_{ij} = \delta_{ij} + \frac{1}{\pi_i} \log \left( \sum_{k \neq i} e^{\pi_i \left( \min_{l \neq i}(\delta_{il}) - \delta_{ik} \right)} \right) \quad \forall i \neq j \ .$$

Let us recall this second term is negative.

Their radii are affected by the perplexity, appearing in $\pi_i$. The importance of choosing a perplexity strictly greater than 1 is demonstrated by this example. For instance, the distance between points $A$ and $B$ is 0 in the first case, but close to 0 (not exactly 0) in the second case, thus avoiding the problem of having zero distance between distinct points. Actually, the blue and green circles touch point A and B in the first case, with $\mathcal{P} = 1$, while their radii are just smaller than the Euclidean distance between those points in the second case, with $\mathcal{P} = 1 + \epsilon$.
The smaller circles in the third figure confirm that the radius decreases as the perplexity increases, reducing the emphasis on the closest neighbor. With a perplexity of 2, the distances are also less discriminant, supporting this assumption.

Now, consider the center sub-plot of Figure 4.1, which illustrates distances with $\mathcal{P} = 1 + \epsilon$, the perplexity value used in this thesis. As previously mentioned, the distance between $A$ and $B$ is close to 0 because they are each other's closest neighbors. The nearest neighbor of $C$ is $B$, but since this is not reciprocal and we symmetrize the final distance matrix with $\overline{\delta} = \frac{\overline{\delta}+\overline{\delta}^T}{2}$, the distance is not as close to 0.

The key difference between the traditional k-NN distance and our sneodesic distance lies in these circles, which shorten the paths based on the SNE similarities, being then influenced by the perplexity value. This emphasizes the significance of sneodesic distances, a term derived from 'SNE' and 'geodesic'.

## 4.2 Angle Distribution

As mentioned previously, some of our distances can vary depending on the distribution of angles in the dataset. Specifically, *sneoSQ* is designed to achieve better performance when angles taken by the shortest path algorithm are close to 90°. Distance metrics such as *sneoS* or *sneoSCst* can also take advantage of squared angles because their final formulas take into account a reduction on the squared Euclidean distances before summing them, although it is expected to be less flagrant than *sneoSQ*. Conversely, other distance metrics such as *sneoNS* and *sneoNSS* do not particularly take advantage of the angular properties of the dataset as they compute basic additions between segments forming the shortest paths.

One can observe in Figure 4.2 that the distribution mostly concentrates around 0.5, representing angles of 60°. When taking three random points, they form a triangle whose sum of angles must be 180°. Hence, the mean of the angles has to be 60°, being simply $\frac{180°}{3}$. Moreover, as the dimension increases, the distribution of distances concentrates, indicating that triangles tend to be equilateral, a type of triangle having only 60° angles. They have then, of course, the same mean but a higher concentration at this value.
The legend of Figure 4.2 is ordered by increasing number of dimensions of the dataset. One can clearly observe the correlation between the number of dimensions and the density at cosine = 0.5, with *MNIST* as the only dataset not strictly respecting the order, *cifar10* and *cifar100* lying in the same number dimensions. This empirical result reinforces the idea of distance concentration in high dimensions, leading to equilateral triangles.

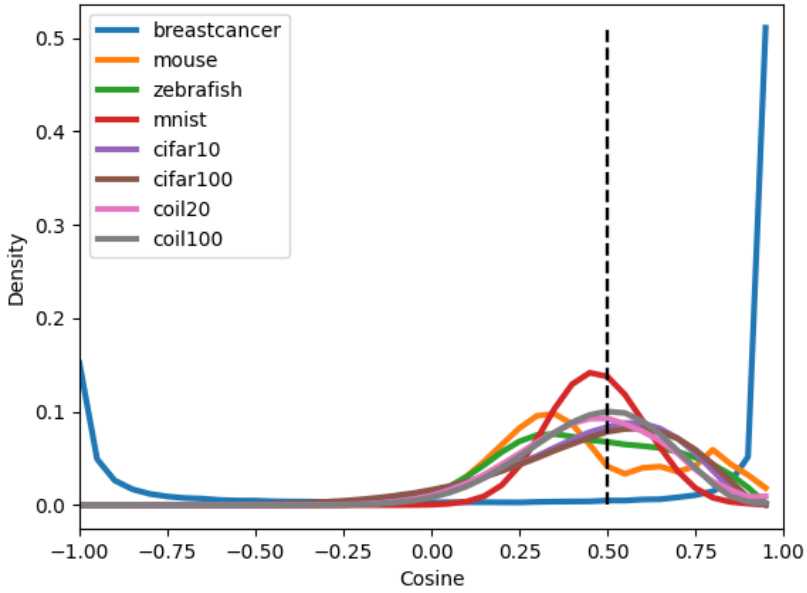



Figure 4.2: Distribution of cosine of angles formed by three points, by dataset

The *Breast Cancer* dataset behaves differently. Its shape seems peculiar, which can be seen with its 2D representation in Figure 4.3, built using the simple *MDS* taking the Euclidean distance matrix as input.

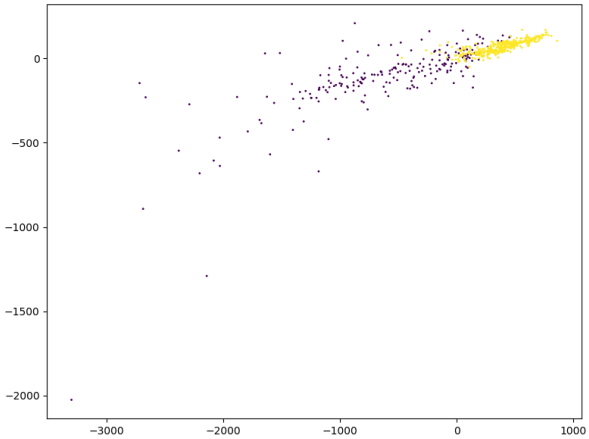



Figure 4.3: Representation of the *Breast Cancer* dataset in 2D, using MDS with Euclidean distance

It is easy to understand why the distribution of very open or very closed angles is so significant with this representation. The dataset nearly follows a straight line. Additionally, it has the fewest dimensions, which limits the degrees of freedom available to form an angle.

Now, let us examine in Figure 4.4 two more graphs that are particularly interesting for our specific case: most open angles possible to build among nearest points and angles being part of the shortest paths used to compute our sneodesic distances.

Although the two graphs are not perfectly identical, they are quite similar. Several factors could explain the presence of differences. For instance, the graphs built for the sneodesic distance are complete, meaning that a jump from one point to another does not necessarily involve its nearest neighbors. Additionally, when approaching the target point, the shortest path does not necessarily select the largest angle; it simply progresses towards the target. Another reason is that the shortest path algorithm can not always choose the largest angle; such a choice would eliminate any decision-making by the algorithm about attempting to select small distances. For instance, it might be more efficient to zigzag among short distances than to take a direct but longer route to the final destination. This would make no sense with the traditional Euclidean distance, but it does with sneodesic distances.

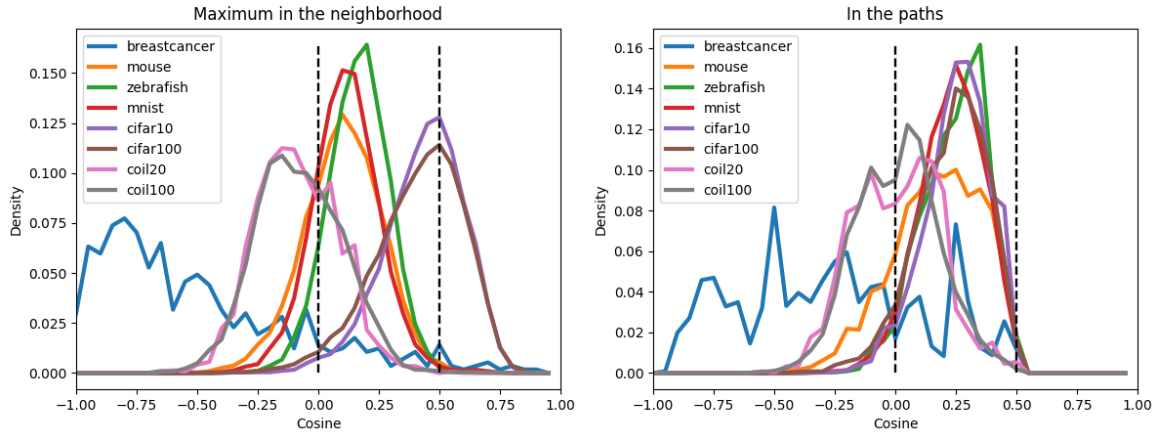



Figure 4.4: Distribution of cosines of the largest angle possible using 2 of the 5 nearest neighbors, in terms of Euclidean distance (left) and of angles within shortest paths between each pair of points when computing sneodesic distances (right)

It is noteworthy that the second graph does not maintain the distribution around 0.5, a cosine value indicating relatively closed angles. Compared to Figure 4.2, the sneodesic distance algorithm tends to take more direct paths with a distribution of cosines generally shifted to the left, although it globally stays around closed angles.

Looking again at the right graph of Figure 4.4, we could infer that the *Coil* datasets benefit most from quadratic sums when measuring path lengths, as their distribution centers around a cosine of 0, which corresponds to angles of 90°.

## 4.3 Neighborhood Study

| | breast cancer | mouse | zebrafish | mnist | cifar10 | cifar100 | coil20 | coil100 |
|---|---|---|---|---|---|---|---|---|
| eucl | 91.059 | 86.319 | 94.491 | 92.532 | 25.488 | 8.029 | 94.994 | 96.539 |
| 5NN | 91.093 | 86.117 | 94.388 | 92.292 | 25.172 | 8.050 | 95.354 | 97.611 |
| 50NN | 91.059 | 86.319 | 94.491 | 92.532 | 25.488 | 8.029 | 94.994 | 96.539 |
| sneoNSS | 90.511 | 85.186 | 94.323 | 93.931 | 28.000 | 9.108 | 99.475 | 99.506 |
| sneoNS | 90.427 | 84.293 | 94.134 | 94.132 | 30.303 | 9.839 | 99.868 | 99.675 |
| sneoNS2 | 90.486 | 84.824 | 94.146 | 93.509 | 30.365 | 9.855 | 99.834 | 99.649 |
| sneoNS10 | 90.540 | 85.284 | 94.278 | 92.963 | 30.671 | 10.016 | 98.832 | 99.278 |
| sneoNSCst | 91.012 | 85.230 | 94.243 | 93.634 | 30.590 | 10.413 | 99.868 | 99.689 |
| sneoNSCos | 90.382 | 84.378 | 93.820 | 93.214 | 28.639 | 8.974 | 99.827 | 99.655 |
| sneoS | 90.590 | 84.453 | 94.179 | 94.442 | 28.475 | 9.045 | 99.891 | 99.717 |
| sneoS2 | 90.605 | 84.644 | 94.202 | 94.378 | 28.532 | 9.855 | 99.827 | 99.683 |
| sneoS10 | 90.761 | 85.513 | 94.326 | 93.240 | 29.371 | 9.466 | 98.393 | 99.105 |
| sneoSCst | 91.236 | 85.230 | 94.287 | 94.149 | 30.405 | 10.413 | 99.891 | 99.730 |
| sneoSQ | 90.590 | 84.912 | 94.179 | 93.795 | 29.016 | 9.325 | 99.891 | 99.717 |

Table 4.1: Performance comparison of different distance metrics across various datasets, showing totals with $k$ going up to 10, as explained in Section 3.1.1, green indicates highest values by column while red indicates lowest ones

| | breast cancer | rat | zebrafish | mnist | cifar10 | cifar100 | coil20 | coil100 |
|---|---|---|---|---|---|---|---|---|
| eucl | 88.402 | 74.008 | 90.946 | 83.311 | 21.285 | 4.859 | 69.407 | 67.836 |
| 5NN | 88.194 | 74.167 | 90.851 | 86.356 | 20.053 | 4.529 | 75.152 | 78.322 |
| 50NN | 88.385 | 74.047 | 90.835 | 83.365 | 21.071 | 4.875 | 70.086 | 68.312 |
| sneoNSS | 88.094 | 73.777 | 91.191 | 86.932 | 23.073 | 5.402 | 74.117 | 76.060 |
| sneoNS | 87.977 | 72.133 | 90.706 | 90.048 | 24.326 | 5.627 | 89.074 | 88.629 |
| sneoNS2 | 88.032 | 72.737 | 90.747 | 88.369 | 24.345 | 5.644 | 84.289 | 84.001 |
| sneoNS10 | 88.238 | 73.602 | 90.953 | 85.619 | 24.683 | 5.778 | 71.564 | 72.105 |
| sneoNSCst | 88.079 | 72.612 | 90.726 | 88.673 | 24.342 | 5.729 | 89.074 | 88.632 |
| sneoNSCos | 88.003 | 71.733 | 90.281 | 87.630 | 22.763 | 5.073 | 89.504 | 89.139 |
| sneoS | 87.739 | 72.144 | 90.758 | 91.023 | 22.047 | 4.965 | 90.432 | 90.342 |
| sneoS2 | 87.799 | 72.467 | 90.810 | 90.771 | 22.122 | 5.644 | 87.628 | 87.299 |
| sneoS10 | 88.249 | 73.835 | 91.061 | 86.991 | 23.036 | 5.234 | 71.569 | 71.421 |
| sneoSCst | 87.849 | 72.612 | 90.778 | 90.051 | 24.344 | 5.729 | 90.432 | 90.345 |
| sneoSQ | 87.739 | 72.554 | 90.758 | 89.529 | 22.355 | 5.041 | 90.432 | 90.342 |

Table 4.2: Performance comparison of different distance metrics across various datasets, showing totals with $k$ going up to 100, as explained in Section 3.1.1, green indicates highest values by column while red indicates lowest ones

Let us now focus on Tables 4.1 and 4.2. The datasets (columns) are sorted by increasing number of dimensions and are divided into three parts. The first three datasets lie in less than 60 dimensions. There is a huge gap between *Zebrafish* and *MNIST*, as this is where we begin to have images, going from 60 to 784 dimensions. The third part includes the *Coil* datasets. Referring back to the distribution of cosines in Figure 4.4, they are the datasets with the highest number of null cosines (90°) in their paths. These datasets should then benefit the most from quadratic additions. Additionally, they lie in the highest dimensional spaces.

The distance metrics are divided into three parts: non-sneodesic distances, non-squared sneodesic distances, and sneodesic distances considered as 'squared' as they imply reduction of squared Euclidean distances without square root in the additions, or a square root of the quadratic sum.

Firstly, let us observe the general values. There is absolutely no k-NN classification better with $k = 100$ than $k = 10$, which indicates that our distances perform better on the neighborhood level. It is very common. This may also be a consequence of the fact that the number of points per class does not influence the neighborhood as much as further points. For instance, when we extend to 100 neighbors, some datasets get lower scores, as *Coil-20* which comports only 72 samples by class, being then prevented from obtaining a score of 100% with 100 neighbors.
The overall best results were achieved with the squared sneodesic distances on the *Coil* datasets and are pretty impressive ; achieving up to 99.891% of precision. As explained in Section 2.2.2, the advantage of using quadratic sums is significant due to the shortcuts in the path it can create. The *Coil* datasets benefit the most from this approach due to their concentration of angles around 90°.
Results are notably low on the *Cifar* datasets. One reason is that this dataset comprises images that vary significantly within the same class. For example, in the "Bird" class, two images representing birds may have different backgrounds (unlike *MNIST* and *Coil*), different species of birds, different camera angles, etc. This means that two images representing the same class can be completely different, resulting in poor neighborhood classification performance on those datasets. If we apply random classification on *Cifar-10* and *Cifar-100*, we should get results of 10 and 5, respectively, as they have 10 and 20 balanced classes. We can then see their poor results, especially with 100-NN on *Cifar-100* where it stays between 4.5 and 5.8, not outperforming a random classifier. Therefore, we must be cautious when analyzing the performance of our distance metrics in these situations. These datasets are not the most reliable for drawing conclusions.

Secondly, let us try to explain the noticeable descending diagonal in green observed in the tables. On the left side, we have datasets lying in the fewest dimensions, specifically the first three datasets, which have 30, 50, and 60 dimensions. These can be considered a gray zone between low and high-dimensional spaces. It can be assumed that the curse of dimensionality does not apply here, as evidenced by the basic Euclidean distance achieving satisfactory results although no distance metric clearly outperforms the others. Indeed, there is no huge difference between performances within the three first columns.
The pertinence of distance metrics using squared sneodesic distances is highlighted on the

right side. Indeed, the three best results for both *Coil-20* and *Coil-100* use either squared distances, or quadratic additions, or even both, while having the cosine distribution closest to 0. For *MNIST*, *Cifar-10*, and *Cifar-100*, it is clear that sneodesic distances perform better, which is encouraging. However, no conclusive statement can be made about the best sneodesic method among them.

Thirdly, let us discuss the perplexity. The three datasets where the desired result, i.e., the total score of neighborhood study, decreases monotonically as the perplexity increases are *MNIST*, *Coil-20*, and *Coil-100*, as shown in Figure 4.5. These are the three datasets confirmed to be in high dimension, with interesting k-NN performances. Therefore, their reliability confirms our assumption that $Perpl = 1.001$ is a good choice.

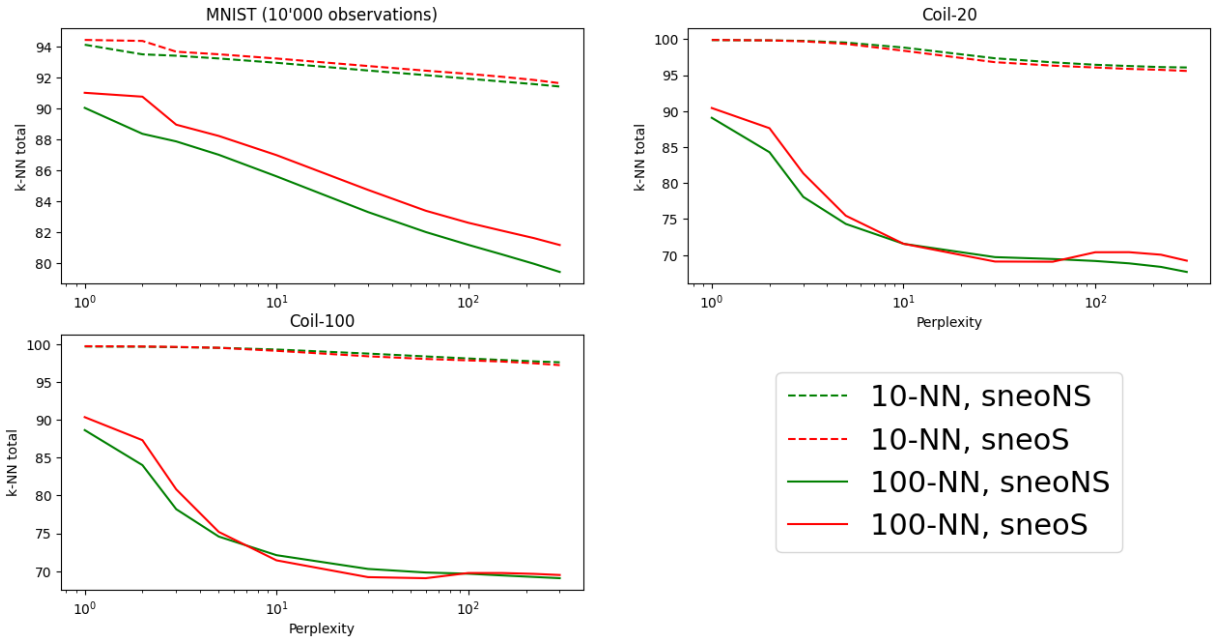



Figure 4.5: Performance of k-NN classifier as a function of perplexity, by dataset

For most of the other datasets, whose results are presented in Figure 4.6, performance decreases with relatively high perplexities, but $1 + \epsilon$ is not the optimal value. This low perplexity is meant to counteract the distance concentration caused by the curse of dimensionality, which means it is normal to see weak performances when the intrinsic dimensionality of the dataset is not high enough. Note also that the real intrinsic dimensionality of these datasets remains unknown.

Lastly, the fact that almost every sneodesic distance metric outperforms all non-sneodesic distances on the high-dimensional datasets is reassuring about our assumptions. The sneodesic distances are based on the shortest path algorithm on graphs, and while they could have results similar to those of k-NN distances, they most frequently outperform them. Moreover, when we look again at Tables 4.1 and 4.2, we can say that they outperform traditional distance metrics in high dimensions much more frequently and in a sharper way than they get outperformed in low-dimensional spaces.

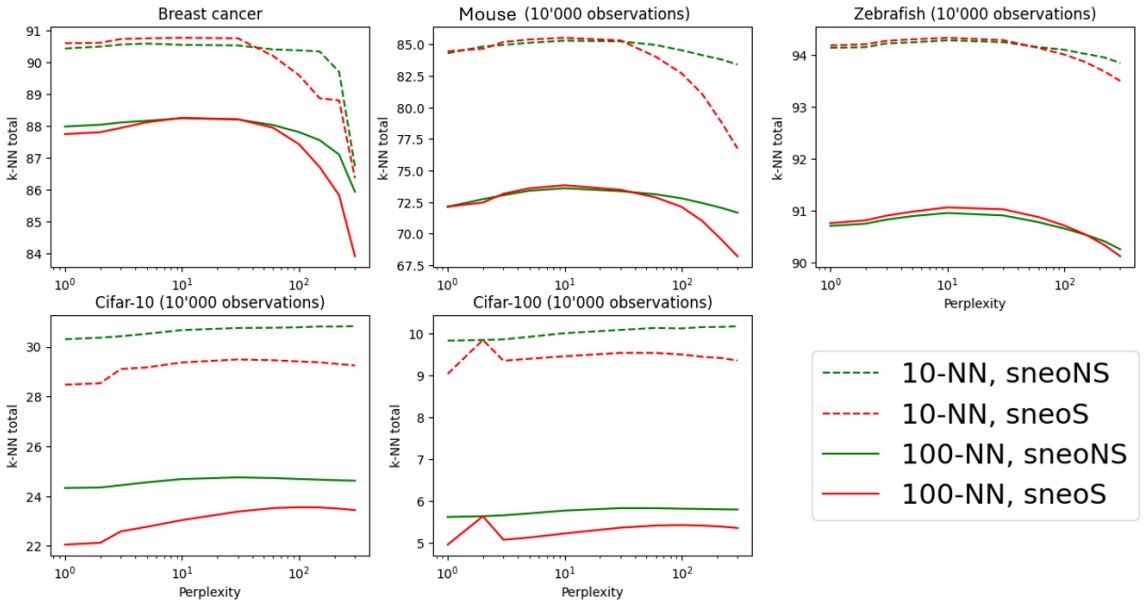



Figure 4.6: Performance of k-NN classifier as a function of perplexity, by dataset

However, *sneoSCst* shows particularly great performance, especially on the *Coil* datasets where it achieves the best results. This distance metric applies a transformation on the Euclidean distance without leveraging the basic principles of SNE similarities, as perplexity does not play any role here. This indicates that the least sneodesic distance among the sneodesic distances sometimes provides the best results.

## 4.4 2D Plots

In this section, we will highlight some of the most interesting results from our dimensionality reduction experiments, given the impracticality of discussing all the plots in detail.

The most impressive results come from the t-SNE dimensionality reduction applied to the *Coil* datasets. These datasets consist of images representing rotating objects. Intuitively, the distance between two images of the same object can be thought of as the rotational angle between them. As the objects complete a full rotation, each of them can be visualized as a closed circle. Ideally, the distance between these circles should be greater than the diameter of any single circle, ensuring that images of the same object (at different angles) are closer to each other than to images representing different objects.
In an optimal scenario, these images would be represented as distinct, closed circles, each corresponding to a single class. As shown in Table 4.3, *sneoNS* and *sneoSQ* outperform basic distances like *eucl* and *5NN* by impressively forming these closed, separate circles for most objects. This clustering capability also explains the superior k-NN classification performance for the *Coil* datasets using sneodesic distances.

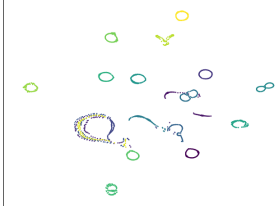
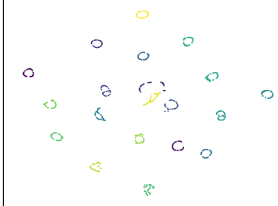
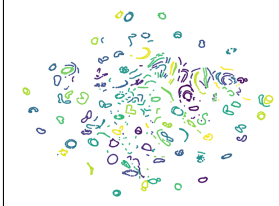
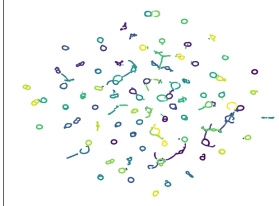
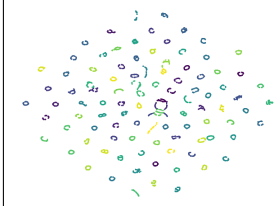
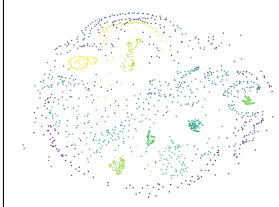
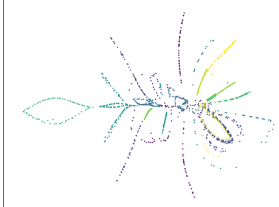
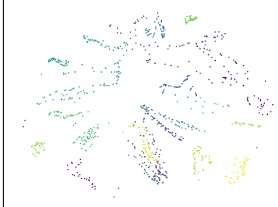
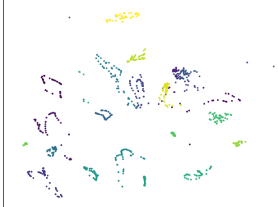
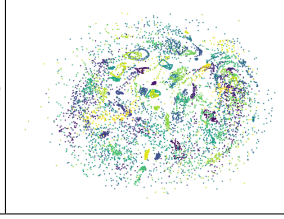
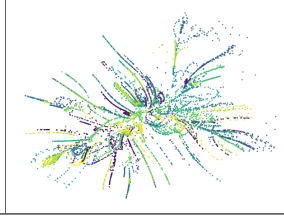
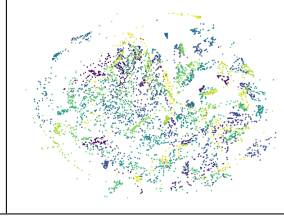
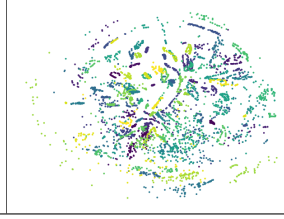
| | eucl | 5NN | sneoNS | sneoSQ |
|---|---|---|---|---|
| Coil-20 t-SNE |  |  |  |  |
| Coil-100 t-SNE |  |  |  |  |
| Coil-20 MDS |  |  |  |  |
| Coil-100 MDS |  |  |  |  |

Table 4.3: 2D representation of the *Coil* datasets, by dimensionality reduction method and distance metric

The best representations among the sneodesic methods are more influenced by the perplexity parameter than by the method itself, meaning that other sneodesic distance metrics provide representations as good as *sneoNS* and *sneoSQ*. Lower perplexities yield better results in almost each case. One can observe an example in Table 4.4. The reason behind this is that a low perplexity emphasizes local neighborhoods, which is crucial for visualizing clusters in 2 dimensions.

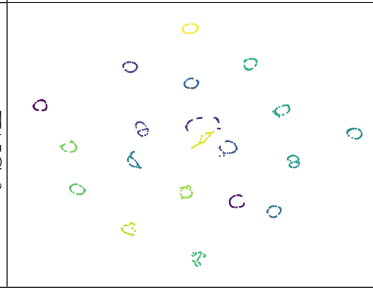
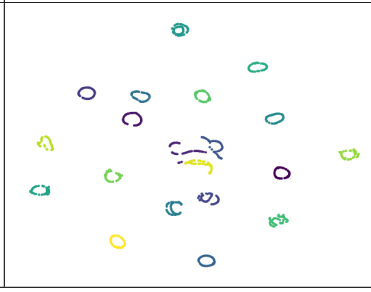
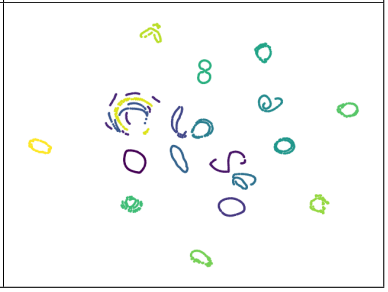
| | sneoNS | sneoNS2 | sneoNS5 |
|---|---|---|---|
| Coil-20 t-SNE |  |  |  |

Table 4.4: 2D representation of the *Coil-20* dataset, using *t-SNE*, by perplexity inside *sneoNS*

We also observe encouraging results on the *Coil-20* dataset using *MDS* as dimensionality reduction algorithm. Classes appear more distinct in the third and fourth columns of Table 4.3 compared to the first two columns. For the *Coil-100* dataset with *MDS*, it is difficult to draw clear conclusions due to the presence of 100 different classes and the lack of discernible shapes.

Other notable results are observable in Figure 4.5, on the *MNIST* and *Zebrafish* datasets. The classes seem globally more mixed and less densely packed within themselves using Euclidean than several different sneodesic distances.

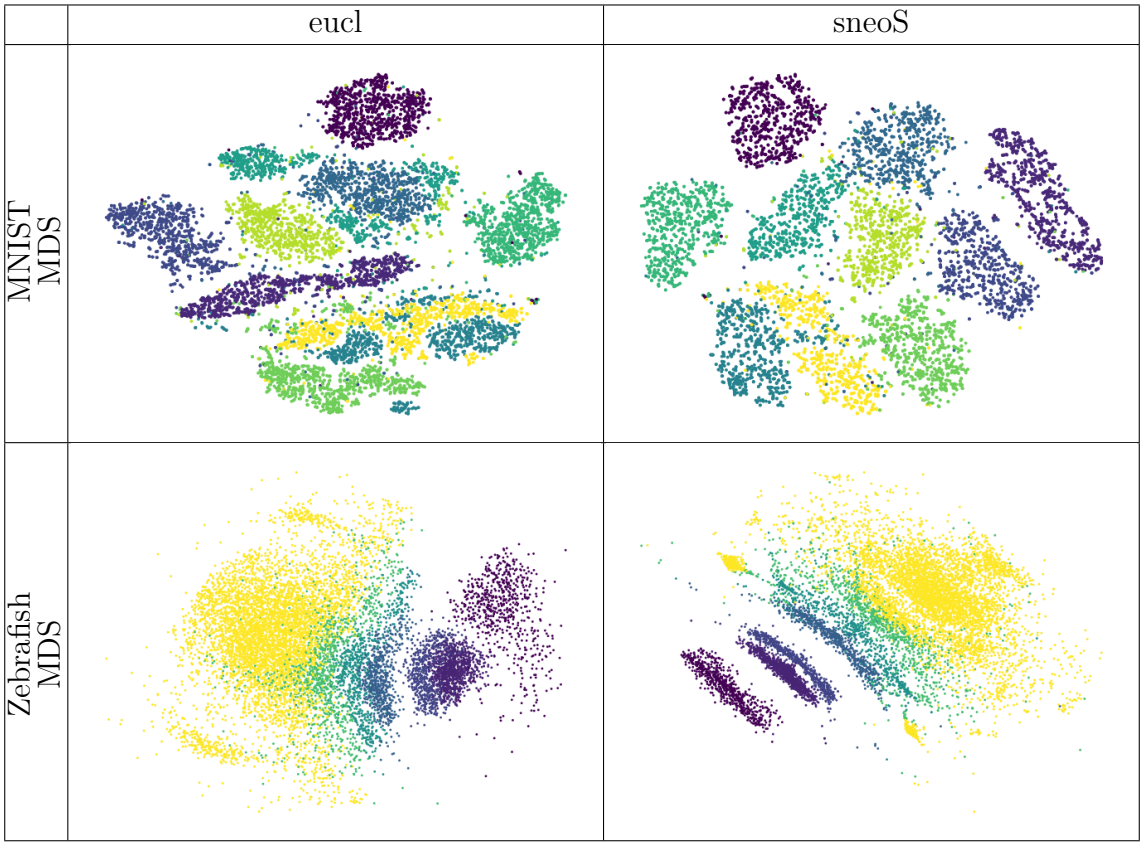



Table 4.5: 2D representation of *MNIST* and *Zebrafish* datasets, using *MDS* as dimensionality reduction algorithm, by distance metric

*sneoSQ* also proves real efficiency when given as input to *MDS* on the *MNIST* dataset, as one can observe in Table 4.6.
For other datasets, no distance metric consistently outperforms the others visually, except for k-NN distances, which seem to perform slightly better on the *Mouse* dataset. As observed in Table 4.7, k-NN distances seem to produce slightly better class grouping. However, *sneoS* still demonstrates better class separation than *eucl* and results in less distortion compared to the 5-NN distance. This distortion is frequently observed in our empirical experiments, being also visible in Table 4.3, and can be considered a significant drawback of these k-NN distances for visualization purposes.

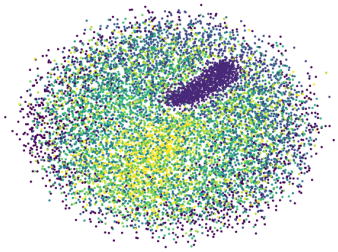
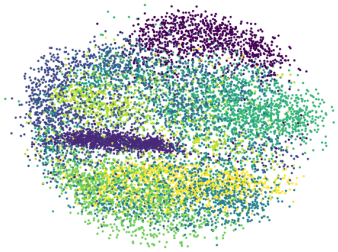
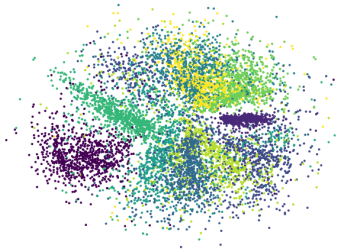
| | eucl | 5-NN | sneoSQ |
|---|---|---|---|
| MNIST MDS |  |  |  |

Table 4.6: 2D representation of the *MNIST* dataset, using *MDS*, by distance metric

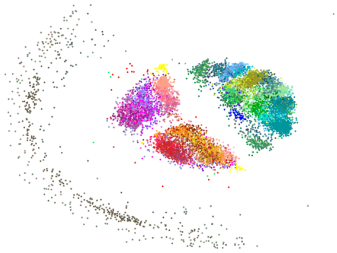
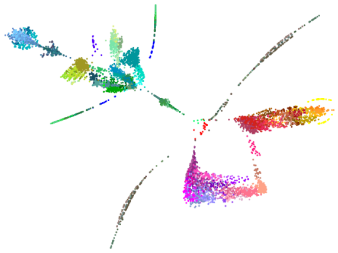
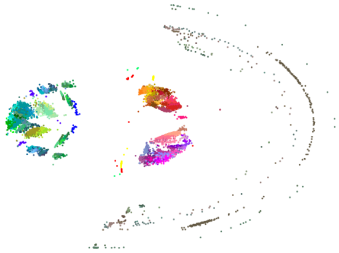
| | eucl | 5-NN | sneoS |
|---|---|---|---|
| mouse MDS |  |  |  |

Table 4.7: 2D representation of the *Mouse* dataset, using *MDS*, by distance metric

Overall, the 2D plots demonstrate sneodesic distances can provide significant improvements in visual separability and clustering quality compared to traditional distances, particularly for datasets with complex structures lying in high-dimensional spaces.
The improved separation of classes in some scenarios using sneodesic distances explains the better classification performance seen in k-NN classifiers. The use of a lower perplexity parameter again proves beneficial, emphasizing local neighborhood relationships which are crucial for high-dimensional data visualization.

## 4.5 $Q_{NX}$

Tables 4.8 and 4.9 present datasets in the columns, while the rows correspond to the distances used in the dimensionality reduction algorithms. Essentially, they represent the distance matrices the methods attempt to preserve from high to low dimensions. The three sub-tables are named based on which distance is used to compute the neighborhood in high and low dimensions for the $Q_{NX}$ formula. Recall it is

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

where $\nu_i^K$ and $n_i^K$ represent the $K$ nearest neighbors to point $i$ in high and low dimensions (after dimensionality reduction), respectively, and $N$ is the total number of points in the dataset.

In Table 4.8, which compares high-dimensional points to points given by *t-SNE* reduction, it is clear that sneodesic distances are the ones that are the easiest to preserve from high to low dimensions. This is a very interesting property that could highlight the 'realness' of this distance. However, one must consider that the dimensionality reduction method used, t-SNE, employs SNE similarities as well, thereby benefiting from this. This is confirmed when looking at the *MDS* table where results are clearly less unequivocal.
When inspecting the sub-tables more profoundly, there are very few instances where the most effective distance metric as input to the dimensionality reduction method is not the distance metric used to calculate the closest neighbors, considering different types of sneodesic distances as a unique distance metric. That is not surprising, as explained in Section 3.1.3. *sneoNS* on the *breastcancer* dataset is one of the very few exceptions. However, it is challenging to draw more conclusions from this. *sneoNS* globally provides satisfying performances on the several datasets in the three sub-tables, appearing to be more adaptive to the various datasets than other sneodesic distance metrics. Squared distances mostly seem to be less performing in this table, getting weaker performance even on the *Coil* datasets. It emphasizes the importance to study both Laplacian and Normal distribution at the base of the SNE similarities and shows less adaptability to the distribution of angles centered around 90°.

As previously mentioned, the performance of sneodesic distances in the *MDS* table, Table 4.9, is less pronounced, although most of the best results are in the second sub-table. Interesting results appear within the sub-table 'Euclidean distance' on the *Coil* datasets. Indeed, one can observe that the sneodesic distances outperform Euclidean and k-NN distances even when neighbors are computed with this same Euclidean distance, supporting the statement about their ability to be preserved. They also achieve better results in the same sub-table on the *Mouse* and *Zebrafish* datasets. This demonstrates their robustness, even in lower-dimensional spaces.
Another interesting performance is the one of the *sneoSQ* distance which globally outperforms every other sneodesic distances on 4 different datasets: *Mouse*, *MNIST*, *Coil-20* and *Coil-100*. It is contradictory to what said in previous paragraph and makes the task to draw more conclusions with those tables difficult.

Nevertheless, it is important to note that the *MDS* table tends to be more trustful to draw any conclusion. Indeed, as explained earlier, it is the method that attempts to conserve distance matrices from high to low dimension in the most strict and simplest way. Moreover, there is no influence by the fact that *t-SNE* and sneodesic distances are both based on a similar concept, the SNE similarities, making them more efficient when working together. However, when comparing the two tables, it is clear that the *t-SNE* reduction is much more performing.

| | breast cancer | mouse | zebrafish | mnist | cifar10 | cifar100 | coil20 | coil100 |
|---|---|---|---|---|---|---|---|---|
| Computed with Euclidean distance | | | | | | | | |
| eucl | 73.978 | 52.240 | 42.323 | 41.110 | 26.800 | 28.037 | 63.452 | 57.727 |
| 5NN | 72.508 | 48.023 | 38.633 | 38.186 | 25.373 | 26.799 | 52.635 | 48.906 |
| 50NN | 73.642 | 52.949 | 42.930 | 40.792 | 25.610 | 27.283 | 63.428 | 56.273 |
| sneoNSS | 70.366 | 45.287 | 37.589 | 33.083 | 23.492 | 24.999 | 48.871 | 44.783 |
| sneoNS | 75.070 | 48.623 | 40.199 | 37.572 | 25.235 | 26.887 | 58.574 | 51.668 |
| sneoNS2 | 71.510 | 46.979 | 36.770 | 36.561 | 23.887 | 25.116 | 52.673 | 48.880 |
| sneoNSCst | 65.557 | 46.314 | 37.887 | 34.400 | 23.452 | 24.571 | 45.069 | 40.806 |
| sneoNSCos | 71.837 | 44.066 | 36.817 | 34.768 | 22.500 | 23.361 | 47.584 | 43.613 |
| sneoS | 71.025 | 44.917 | 38.061 | 34.623 | 22.963 | 24.441 | 48.246 | 44.527 |
| sneoS2 | 71.458 | 44.752 | 38.151 | 34.489 | 23.611 | 25.165 | 51.569 | 48.638 |
| sneoSCst | 65.277 | 47.177 | 37.109 | 33.921 | 23.328 | 25.146 | 43.976 | 40.927 |
| sneoSQ | 69.264 | 49.053 | 40.521 | 37.210 | 24.678 | 25.921 | 48.165 | 44.415 |
| Computed with *sneoNS* distance | | | | | | | | |
| eucl | 71.681 | 49.353 | 45.730 | 42.813 | 24.182 | 26.216 | 64.641 | 60.378 |
| 5NN | 69.247 | 45.899 | 39.301 | 40.957 | 22.688 | 24.957 | 55.638 | 52.324 |
| 50NN | 71.544 | 50.454 | 44.586 | 44.014 | 23.418 | 25.629 | 65.782 | 58.559 |
| sneoNSS | 74.997 | 54.777 | 50.882 | 49.728 | 31.228 | 34.129 | 67.774 | 60.712 |
| sneoNS | 75.285 | 53.723 | 50.004 | 49.647 | 27.484 | 29.695 | 73.113 | 64.526 |
| sneoNS2 | 75.972 | 47.867 | 49.719 | 45.662 | 28.925 | 34.115 | 69.579 | 63.534 |
| sneoNSCst | 73.580 | 48.671 | 51.462 | 44.924 | 28.798 | 33.671 | 66.061 | 60.075 |
| sneoNSCos | 76.542 | 45.886 | 50.436 | 44.374 | 27.558 | 32.980 | 63.975 | 58.479 |
| sneoS | 76.677 | 54.516 | 51.340 | 50.983 | 29.744 | 28.311 | 66.932 | 60.321 |
| sneoS2 | 76.606 | 54.321 | 50.953 | 50.984 | 29.852 | 34.041 | 68.566 | 63.052 |
| sneoSCst | 74.850 | 49.199 | 50.946 | 51.320 | 31.430 | 34.214 | 66.751 | 60.161 |
| sneoSQ | 73.942 | 50.092 | 54.069 | 46.387 | 29.467 | 34.695 | 66.456 | 59.533 |
| Computed with 5-NN distance | | | | | | | | |
| eucl | 76.495 | 53.844 | 45.264 | 46.134 | 29.973 | 30.922 | 62.554 | 58.793 |
| 5NN | 75.621 | 50.128 | 41.934 | 44.686 | 30.698 | 31.806 | 63.484 | 59.722 |
| 50NN | 76.498 | 55.182 | 46.215 | 45.812 | 28.547 | 30.226 | 65.485 | 57.187 |
| sneoNSS | 73.992 | 45.726 | 38.851 | 36.427 | 23.572 | 24.926 | 53.804 | 49.949 |
| sneoNS | 77.082 | 49.329 | 41.777 | 40.698 | 27.216 | 29.755 | 63.673 | 58.366 |
| sneoNS2 | 74.814 | 50.288 | 39.830 | 38.845 | 24.147 | 24.954 | 57.835 | 56.173 |
| sneoNSCst | 69.603 | 46.183 | 40.292 | 38.343 | 23.696 | 24.629 | 49.031 | 46.221 |
| sneoNSCos | 71.073 | 47.593 | 39.286 | 37.925 | 22.446 | 22.963 | 50.015 | 49.850 |
| sneoS | 75.327 | 46.820 | 40.319 | 38.855 | 24.070 | 25.003 | 53.803 | 53.136 |
| sneoS2 | 75.779 | 45.301 | 41.390 | 38.200 | 24.766 | 25.107 | 55.990 | 56.294 |
| sneoSCst | 70.595 | 49.242 | 39.825 | 38.020 | 23.521 | 25.168 | 47.526 | 47.082 |
| sneoSQ | 73.705 | 50.505 | 41.292 | 41.783 | 26.340 | 27.112 | 53.212 | 51.309 |

Table 4.8: Performance comparison of different distance metrics given to *t-SNE*, across datasets, showing totals AUC of $Q_{NX}$, as explained in subsection 3.1.3, neighboring points are computed using three different distances, making three sub tables, green indicates highest values by column while red indicates lowest ones

| | breast cancer | mouse | zebrafish | mnist | cifar10 | cifar100 | coil20 | coil100 |
|---|---|---|---|---|---|---|---|---|
| Computed with Euclidean distance | | | | | | | | |
| eucl | 77.840 | 30.343 | 25.233 | 12.467 | 20.502 | 19.335 | 43.396 | 36.211 |
| 5NN | 75.930 | 37.171 | 27.011 | 21.518 | 18.961 | 19.104 | 35.850 | 27.725 |
| 50NN | 74.780 | 37.837 | 27.070 | 20.738 | 20.172 | 20.314 | 43.615 | 35.134 |
| sneoNSS | 72.544 | 38.010 | 29.096 | 19.045 | 17.444 | 17.961 | 48.090 | 44.918 |
| sneoNS | 74.899 | 34.651 | 30.306 | 16.688 | 21.082 | 20.080 | 47.240 | 42.160 |
| sneoNS2 | 72.198 | 38.497 | 28.397 | 19.550 | 17.633 | 17.572 | 46.141 | 33.389 |
| sneoNSCst | 70.787 | 40.052 | 28.624 | 21.546 | 17.489 | 17.498 | 50.446 | 43.797 |
| sneoNSCos | 65.406 | 34.056 | 26.975 | 19.441 | 15.091 | 16.418 | 41.191 | 40.489 |
| sneoS | 63.717 | 39.454 | 24.476 | 16.533 | 13.884 | 14.691 | 53.094 | 38.585 |
| sneoS2 | 60.242 | 39.530 | 24.283 | 17.363 | 14.070 | 14.470 | 53.260 | 35.521 |
| sneoSCst | 60.408 | 38.064 | 24.578 | 13.408 | 16.395 | 14.181 | 48.610 | 36.795 |
| sneoSQ | 69.306 | 43.620 | 28.967 | 25.751 | 13.873 | 15.366 | 53.686 | 49.646 |
| Computed with *sneoNS* distance | | | | | | | | |
| eucl | 69.957 | 28.568 | 22.642 | 9.209 | 17.629 | 16.623 | 35.061 | 31.657 |
| 5NN | 69.957 | 37.663 | 27.480 | 21.123 | 15.979 | 16.262 | 37.393 | 30.621 |
| 50NN | 67.857 | 37.164 | 25.751 | 18.276 | 17.218 | 17.711 | 38.605 | 33.089 |
| sneoNSS | 75.582 | 41.385 | 33.378 | 25.328 | 19.632 | 19.854 | 57.829 | 47.591 |
| sneoNS | 71.547 | 34.143 | 28.801 | 15.128 | 18.719 | 18.305 | 42.430 | 38.353 |
| sneoNS2 | 74.017 | 40.478 | 32.448 | 23.104 | 18.326 | 19.317 | 49.645 | 33.978 |
| sneoNSCst | 72.509 | 41.624 | 33.044 | 26.076 | 18.219 | 19.317 | 60.573 | 46.103 |
| sneoNSCos | 68.161 | 36.118 | 31.255 | 24.293 | 16.265 | 18.684 | 51.878 | 43.479 |
| sneoS | 66.902 | 43.266 | 28.468 | 18.819 | 16.463 | 15.422 | 65.618 | 41.865 |
| sneoS2 | 63.005 | 43.001 | 28.521 | 20.477 | 16.095 | 15.367 | 62.067 | 37.432 |
| sneoSCst | 62.668 | 40.140 | 28.841 | 18.149 | 18.302 | 15.503 | 62.390 | 40.568 |
| sneoSQ | 72.409 | 45.414 | 33.550 | 28.012 | 15.287 | 17.627 | 66.852 | 56.701 |
| Computed with 5-NN distance | | | | | | | | |
| eucl | 76.136 | 25.847 | 23.554 | 10.254 | 19.457 | 17.892 | 32.843 | 24.697 |
| 5NN | 76.175 | 43.526 | 28.390 | 24.974 | 22.657 | 22.115 | 53.046 | 45.592 |
| 50NN | 73.113 | 37.521 | 26.676 | 20.596 | 19.969 | 19.842 | 37.889 | 28.412 |
| sneoNSS | 68.322 | 36.915 | 29.597 | 20.464 | 15.819 | 16.128 | 48.247 | 36.429 |
| sneoNS | 74.172 | 32.101 | 29.117 | 14.821 | 20.465 | 19.425 | 39.604 | 30.001 |
| sneoNS2 | 69.324 | 36.262 | 29.107 | 20.692 | 16.065 | 15.556 | 40.708 | 25.769 |
| sneoNSCst | 67.944 | 38.019 | 29.215 | 23.241 | 15.938 | 15.549 | 48.507 | 34.423 |
| sneoNSCos | 62.486 | 32.992 | 27.529 | 21.369 | 13.519 | 14.539 | 40.020 | 32.699 |
| sneoS | 55.488 | 39.155 | 24.892 | 17.573 | 12.467 | 12.222 | 53.703 | 31.784 |
| sneoS2 | 58.726 | 39.703 | 24.866 | 18.752 | 12.652 | 13.015 | 52.137 | 29.809 |
| sneoSCst | 54.725 | 38.647 | 25.272 | 14.723 | 14.972 | 12.586 | 50.406 | 32.402 |
| sneoSQ | 65.398 | 41.307 | 27.896 | 27.640 | 12.390 | 13.880 | 56.945 | 46.958 |

Table 4.9: Performance comparison of different distance metrics given to *MDS*, across datasets, showing totals AUC of $Q_{NX}$, as explained in subsection 3.1.3, neighboring points are computed using three different distances, making three sub tables, green indicates highest values by column while red indicates lowest ones

# Chapter 5

# Discussion

## 5.1 Summary of Results

In the previous chapter, we thoroughly examined the performance of various distance metrics across different datasets. The primary focus was on evaluating our proposed sneodesic metrics and comparing their performance with traditional metrics. The experimental results provided several key insights.

First, we observed that the distribution of angles formed by three random points in the datasets typically clustered around 60°. This finding is consistent with geometric intuition, suggesting that in higher dimensions, random triangles tend to become equilateral due to the distance concentration, where each angle is close to 60°. This concentration of angles has significant implications for the performance of distance metrics. Our evaluation showed that the performance of distance metrics varies depending on the angular distribution within the dataset. The *sneoSQ* metric, which is designed to perform optimally when angles are close to 90°, often demonstrated superior performance in datasets where this condition was present. On the other hand, non-quadratic metrics like *sneoNS* and *sneoNSS* also showed good performance but seemed less impacted by this angle distribution. The difference between quadratic and non-quadratic results highlights some importance in considering angular distribution when selecting a distance metric for a specific dataset.

Moreover, while k-NN (geodesic) distances proved to be a good initial solution to the curse of dimensionality, sneodesic distances seem to be even more robust and efficient. Taking a low perplexity proves to be beneficial in most cases, being in agreement with the initial idea of k-'nearest'-neighbors. However, the innovation to not take a discrete number of neighbors into account but to give to all neighbors a statistical similarity value in the k-NN graph seems to give more flexibility while keeping some advantages on the structural aspect the geodesic distance provides. Moreover, the sneodesic distances eliminate the need for the hyperparameter $k$ and create less distortion when visualizing the 2D representations using these distances as input to the *MDS* dimensionality reduction method.

Furthermore, it is challenging to draw definitive conclusions regarding the superiority of any specific sneodesic distance on the others, as each metric exhibits both strengths and weaknesses. The effectiveness of a particular distance metric can vary depending on numerous characteristics of the dataset, such as its angular distribution and dimensionality. Consequently, it is interesting to consider all sneodesic distance metrics, which differ in the distribution of the SNE similarities and the method used to sum the segments during shortest path computations.

Additionally, distance metrics such as *sneoNSCst* or *sneoSCst*, which do not take into account the perplexity intrinsic to the concept of SNE—the basis of the idea behind sneodesic distances—show surprisingly good results. This suggests that some results may benefit more from a well-chosen monotonic transformation of Euclidean distances than from the core principle behind sneodesic distances. However, the step of shortest path calculation, as well as the idea of giving less and less importance to further points, remain present and are still inherent to the concept of sneodesic distance.

Last but not least, some empirical performances showed impressive results, especially on the *Coil* datasets. They particularly benefited from these new metrics as evidenced by their k-NN classification scores and their encouraging 2D representations. For instance, the 2D representation with t-SNE stood out as exceptionally interesting and performing, showing clear, well-formed, and distinct clusters that enhance the interpretability and quality of the visualized data. Other results demonstrate that this distance is very efficient in high-dimensional spaces while not being inefficient in lower dimensions. Overall, sneodesic distances appear to be more adaptive than the Euclidean or k-NN distances, while still showing highs and lows in term of efficiency as every distance metrics.

## 5.2 Limitations and Future Work

While our study presents promising results, there are certain limitations that need to be addressed. Below, we outline these limitations and propose directions for future research.

#### Dataset Variety

The variety of datasets used in our experiments, although diverse, could be expanded to include more complex and varied datasets to further validate our findings.

#### Effects of Dimensionality

The effects of dimensionality on the concentration of angles and, consequently, on metric performance, warrant further investigation, among which studies about the intrinsic dimensionality.

### Dimensionality Reduction Comparisons

Exploring advanced dimensionality reduction techniques, such as well-parameterized autoencoders, could provide valuable insights. Comparing these techniques with our sneodesic distance metric-based approaches would offer a deeper understanding of their effectiveness in high-dimensional data analysis and their ability to provide performing high-dimensional datasets visualization tools.

### Adaptive Metrics

Future research could explore adaptive metrics that dynamically adjust based on real-time analysis of the dataset's properties. This would involve developing algorithms that can automatically select or tune metrics according to the data's characteristics, potentially leading to more robust and versatile models.

### Computational Efficiency

The current implementation takes several hours to compute a sneodesic distance matrix for a dataset of 20,000 elements. Future work should focus on optimizing the code to enhance computational efficiency.

### Integration with Other Algorithms

Sneodesic distances could also be combined with other algorithms, such as the Commute Time Distance presented in Section 1.1.4 which shows a different understanding of the whole data structure. Before the shortest path calculation, we have a new complete distance matrix that could be used with other transformations beyond simple shortest path algorithms.

### Initial Distance Metrics

Currently, we use Euclidean distances, or their squares, as the initial input for sneodesic distance calculations, implying a Laplacian or a Normal distribution, respectively. However, other distance metrics such as Manhattan distance, cosine distance, etc., could also be tested. In fact, any positive dissimilarity value could be used as the initial input for the different sneodesic distances.

# Conclusion

The stakes of this research extend far beyond the immediate focus on distance metrics analysis. By introducing and evaluating innovative sneodesic distance metrics, this study opens new pathways for understanding and leveraging complex data structures. These metrics have the ability to take advantage of specific angular distributions and to make a good compromise between local and global data structures in high dimensions. Indeed, the local aspect is addressed through SNE similarities with a perplexity of $1 + \epsilon$ while the global aspect is handled by using the shortest path algorithm. This combination presents significant opportunities for enhancing the accuracy and relevance of data analysis across diverse fields.

The broader implications are profound. Enhanced techniques for dimensionality reduction and data visualization can greatly improve the interpretability and usability of high-dimensional data, which is crucial in areas such as bioinformatics, image processing, and machine learning. The insights and methodologies developed here could lead to advancements in clustering and classification, thereby impacting various real-world applications.
The 2D representations of the *Coil* datasets using our new metrics and *t-SNE* shown in Table 4.3 in Section 4.4 appear to be the most promising results, among other really great performances. Considering it was not possible to explore all existing datasets in one thesis, it is clear that further explorations can raise new incredible discoveries.

Moreover, this research highlights the potential for combining sneodesic distances with other distances or algorithms for further improvement. Indeed, the formulas at its foundation take a dissimilarity as an input and outputs a new distance matrix. That means that they could be applied to a wide range of initial distance inputs, including Manhattan or cosine distances, broadening their utility and could be given to several algorithms also exploiting different properties of the dataset, such as Commute time distance computations. Globally, there are a lot of improvements to explore, to take even more information about the data structure, without forgetting to make the computation faster as it is one of its principal shortcomings. As a matter of fact, computing the sneodesic distance matrix for more than 20,000 data points is a burden, making the application of these distance metrics to all datasets in their entirety of the utmost difficulty.

Ultimately, the advancements made in this work contribute to the foundational understanding of data structures and provide versatile tools for future research and practical applications in data science, machine learning and beyond.

# Appendix A

# Neighborhood Study Plots

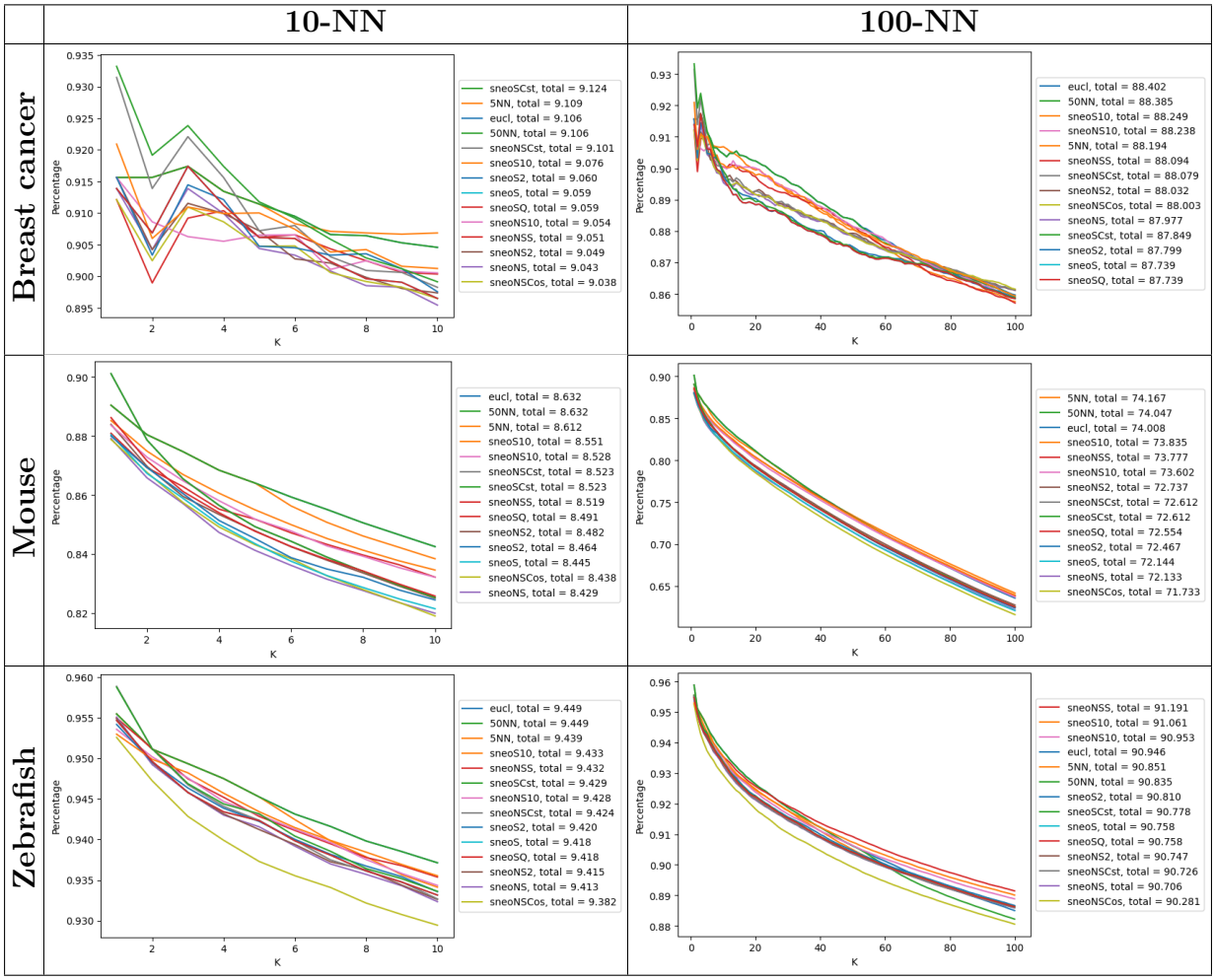



Table A.1: Neighborhood analyze with various distances on different datasets, lines represent the percentage of neighboring points belonging to the same class as points themselves in function of $k$, the legend is ordered by total, which is the mean along $k$, for $k$ going up to 10 (left column) and 100 (right column)

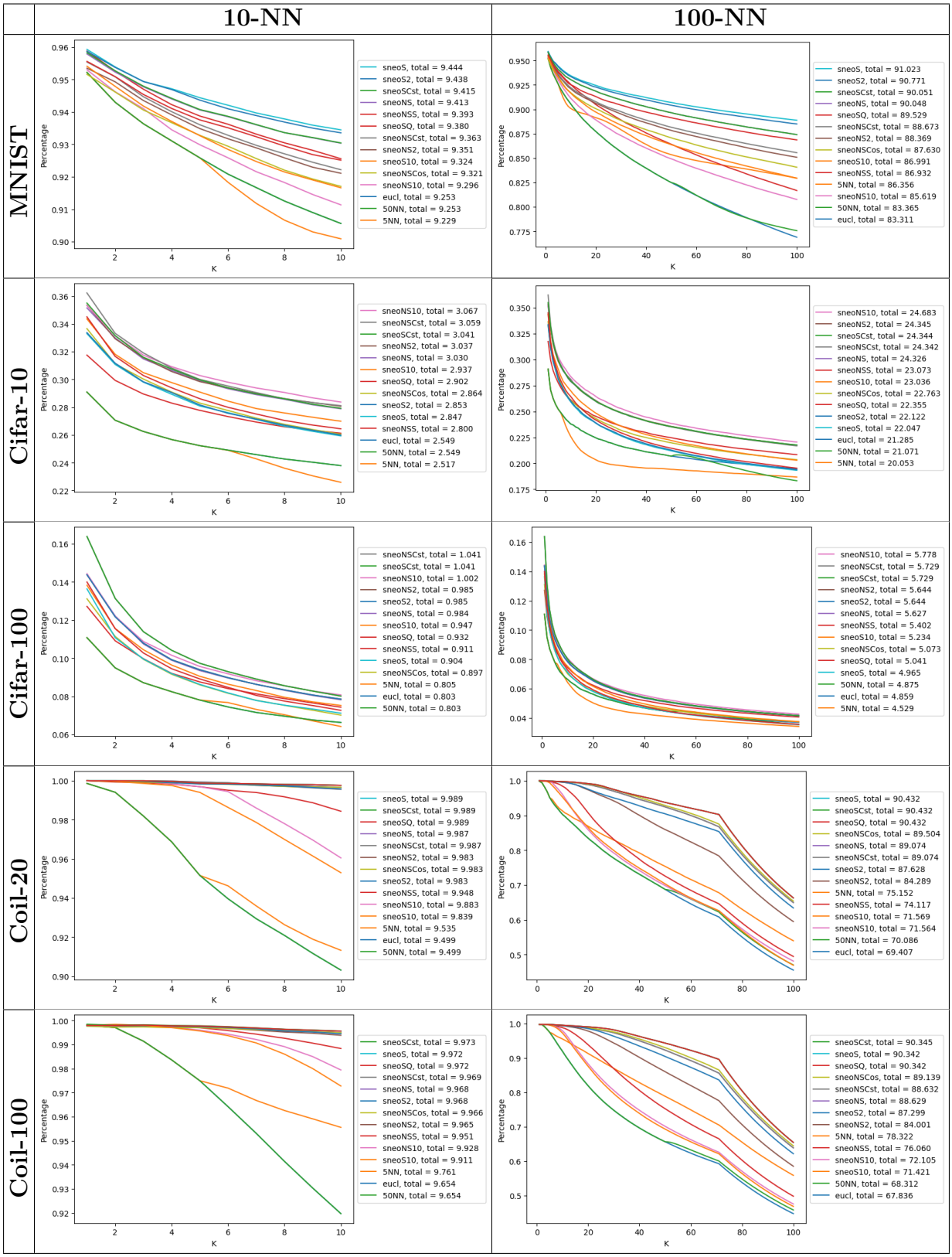



Table A.2: Follows Table A.1

# Appendix B

# 2D Plots

## B.1 MDS

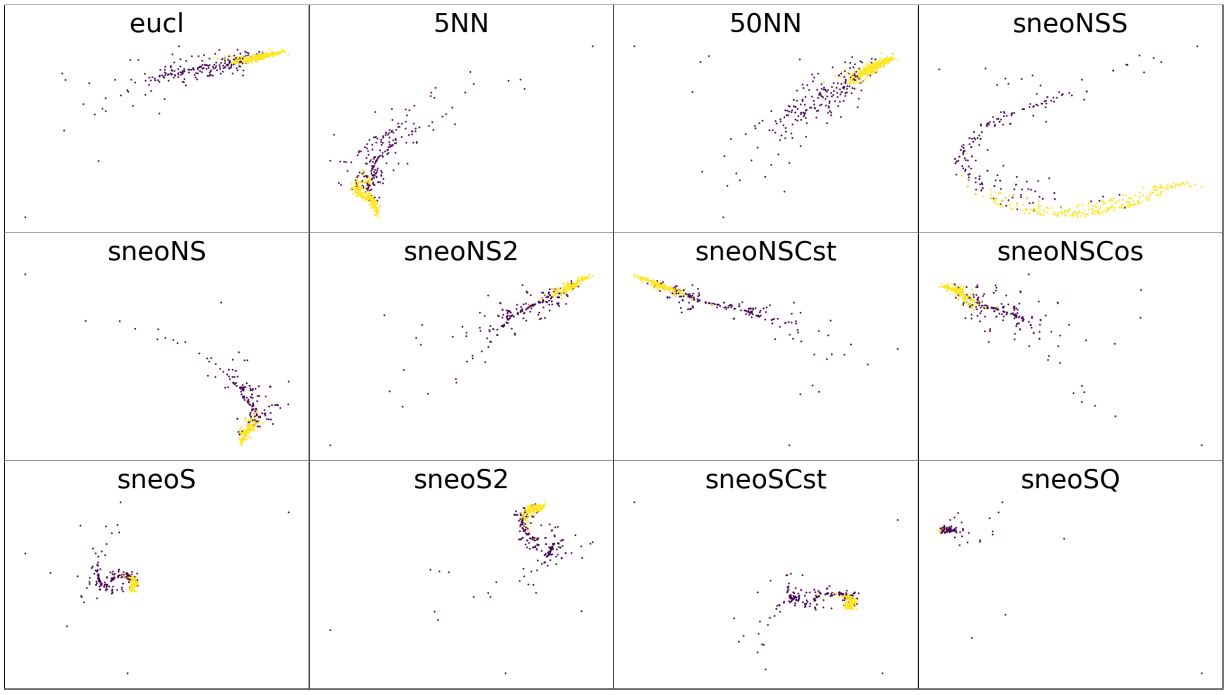



Table B.1: *Breast Cancer* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

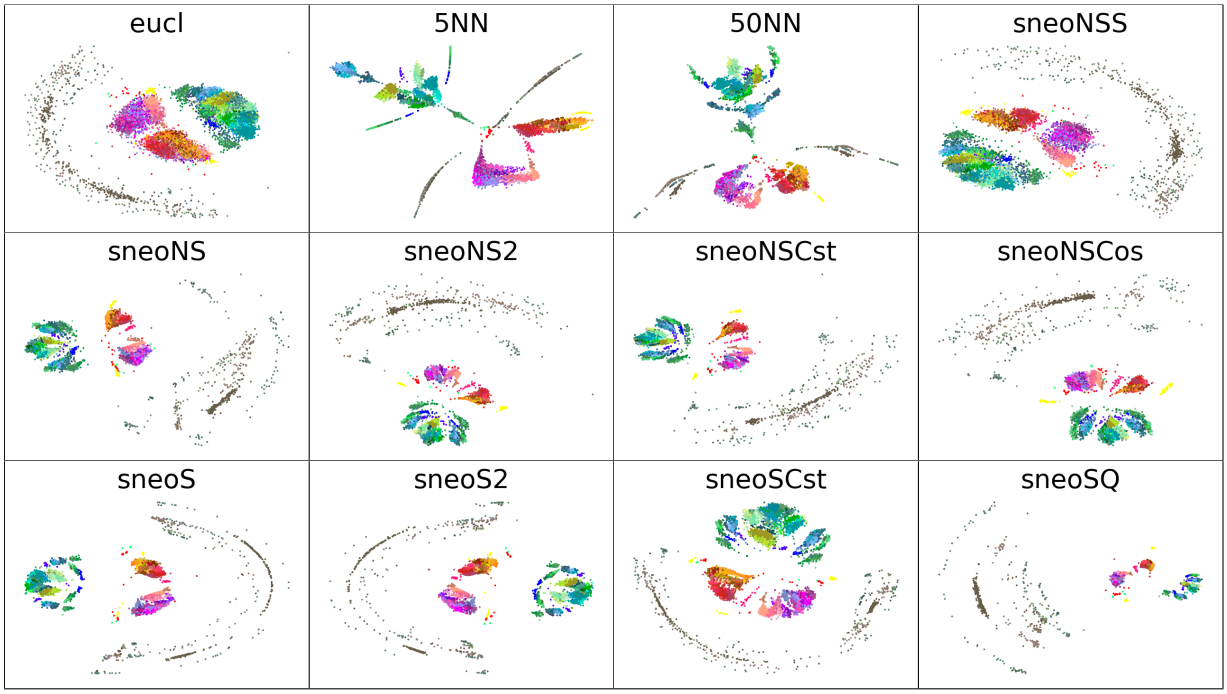



Table B.2: 10'000 observations of the *Mouse* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

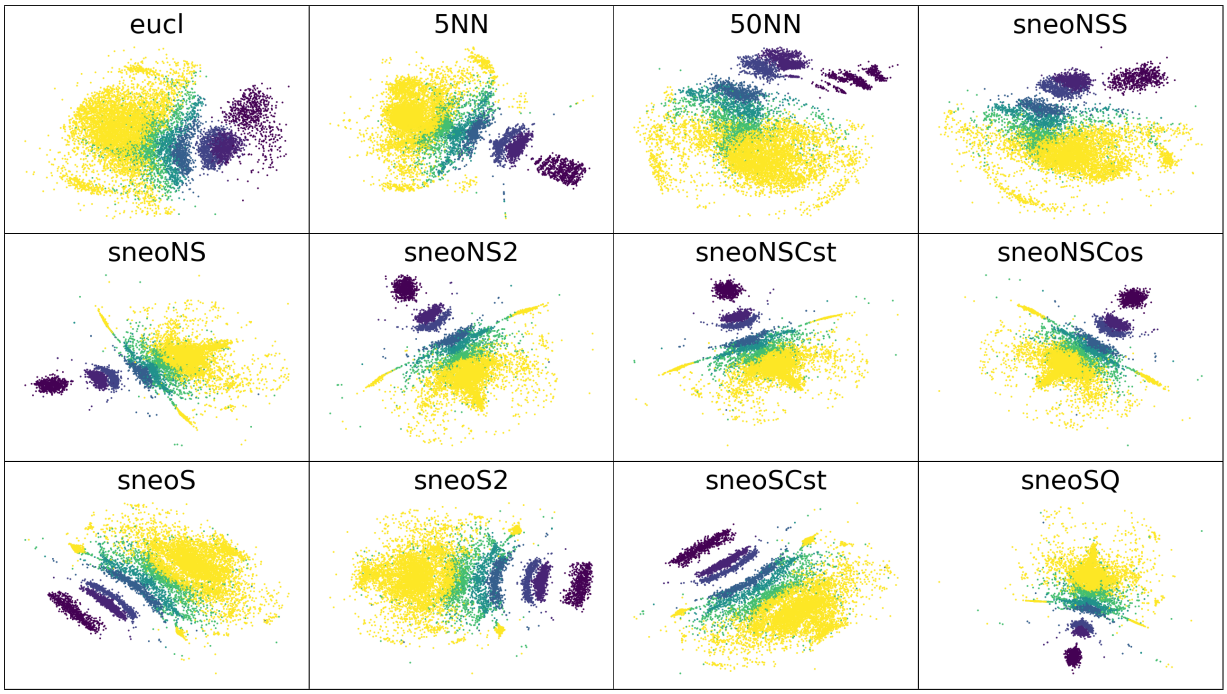



Table B.3: 10'000 observations of the *Zebrafish* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

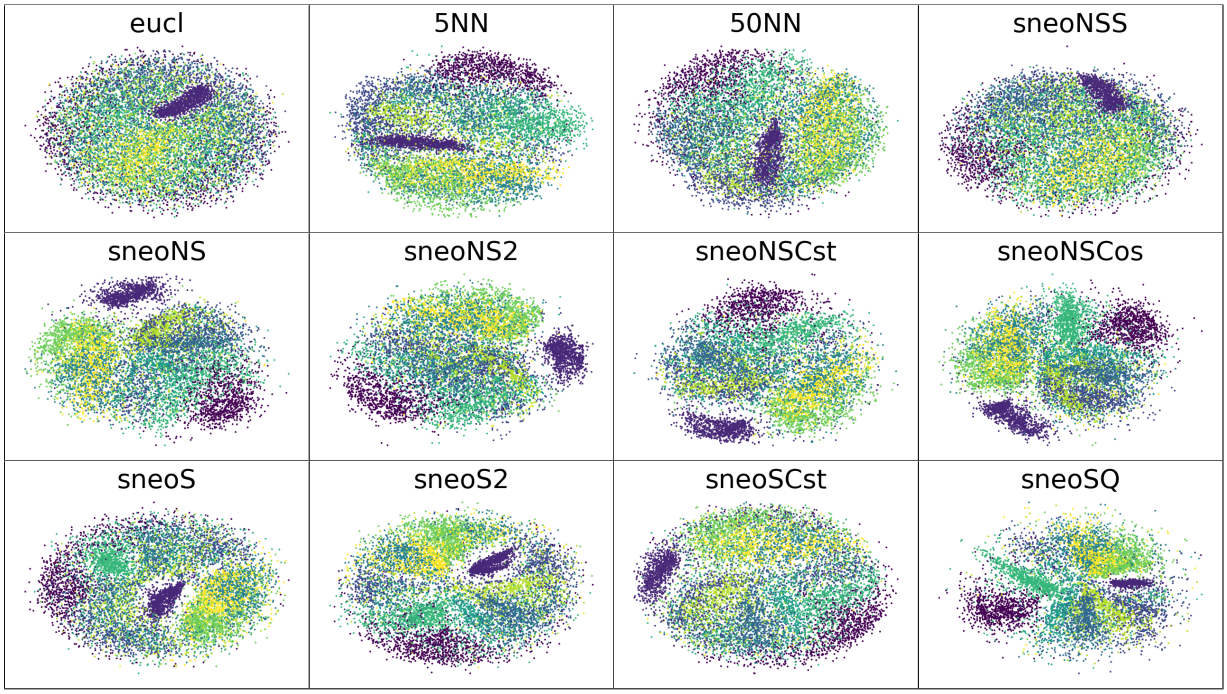



Table B.4: 10'000 observations of the *MNIST* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

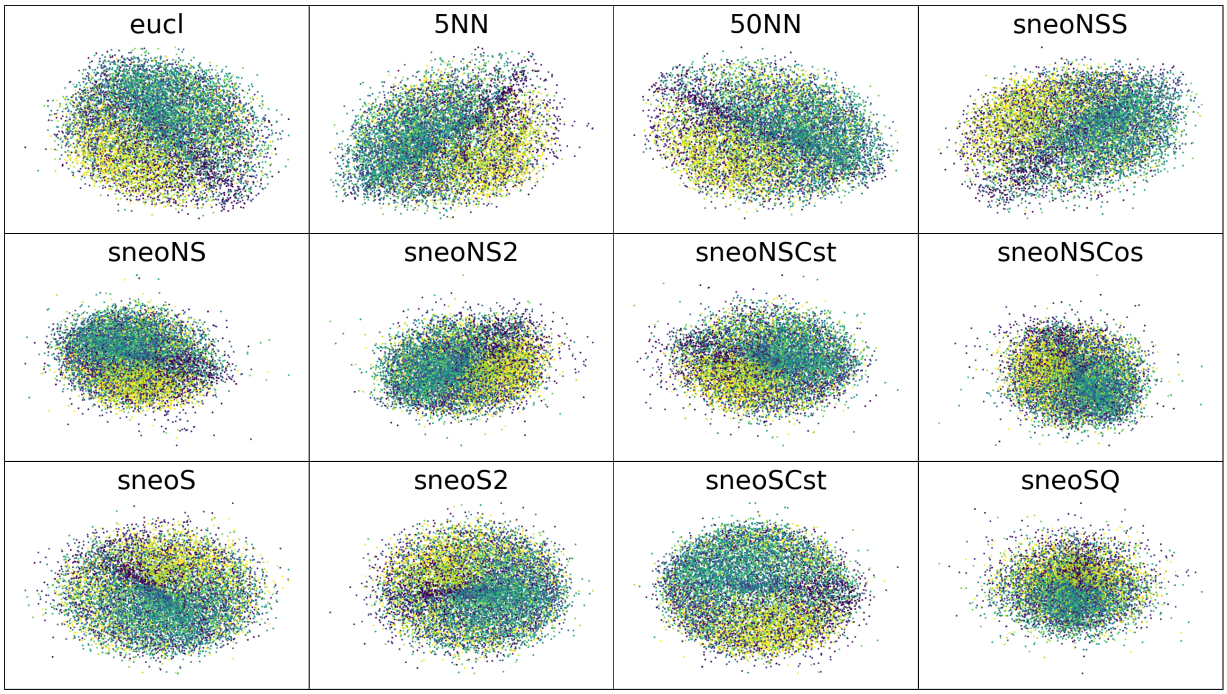



Table B.5: 10'000 observations of the *Cifar-10* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

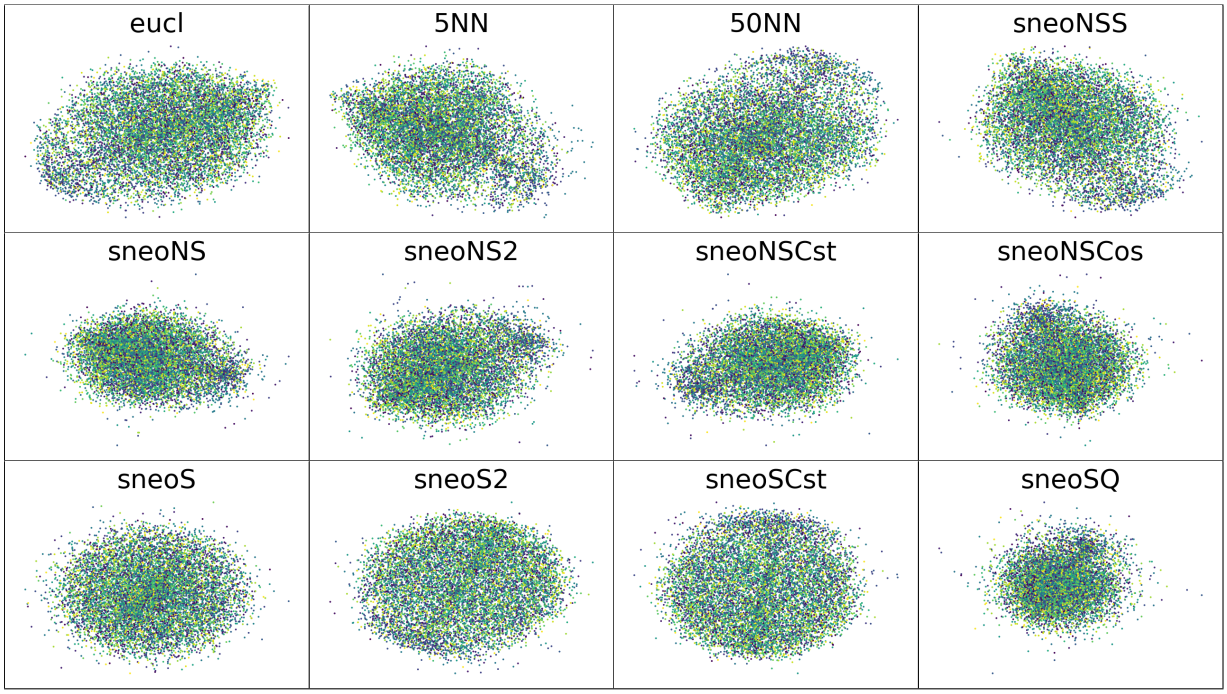



Table B.6: 10'000 observations of the *Cifar-100* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

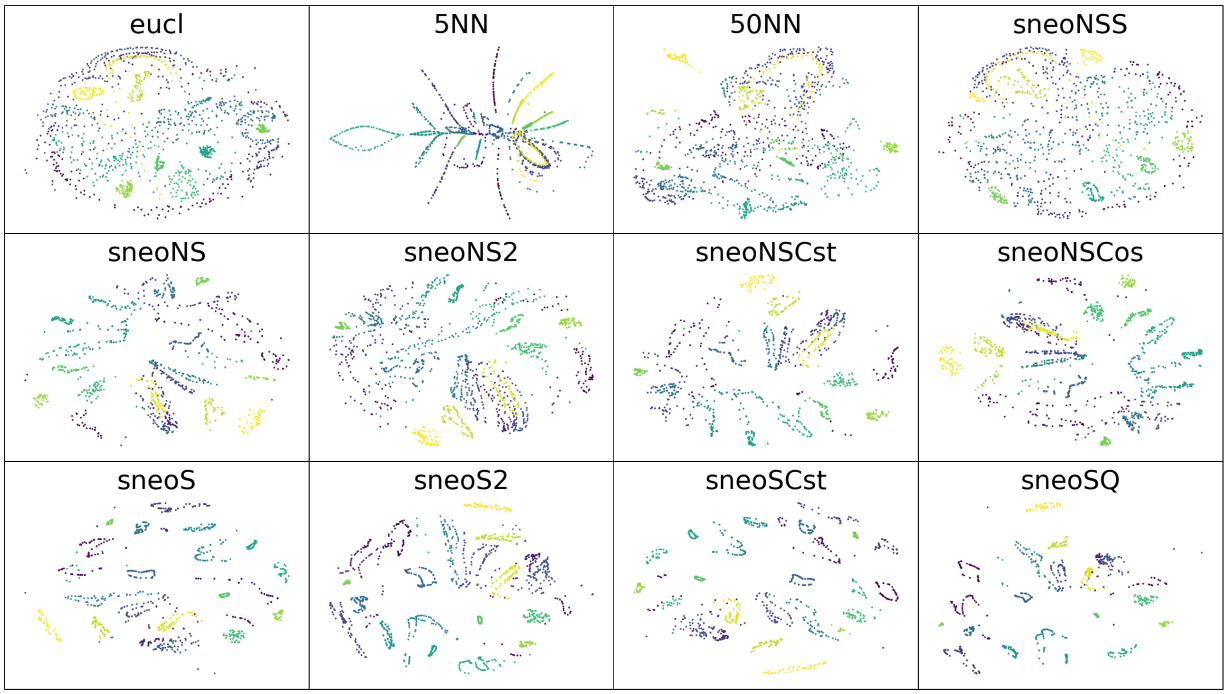



Table B.7: *Coil-20* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

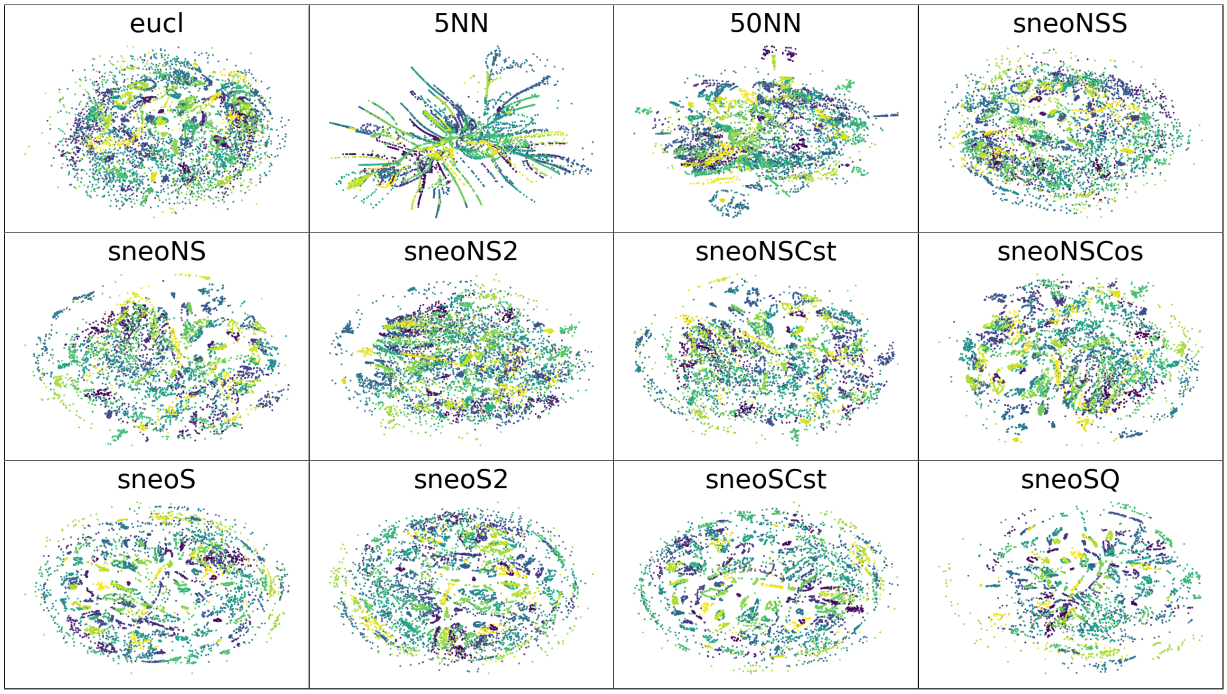



Table B.8: *Coil-100* dataset represented in 2 dimensions using *MDS* as dimensionality reduction method

## B.2 t-SNE

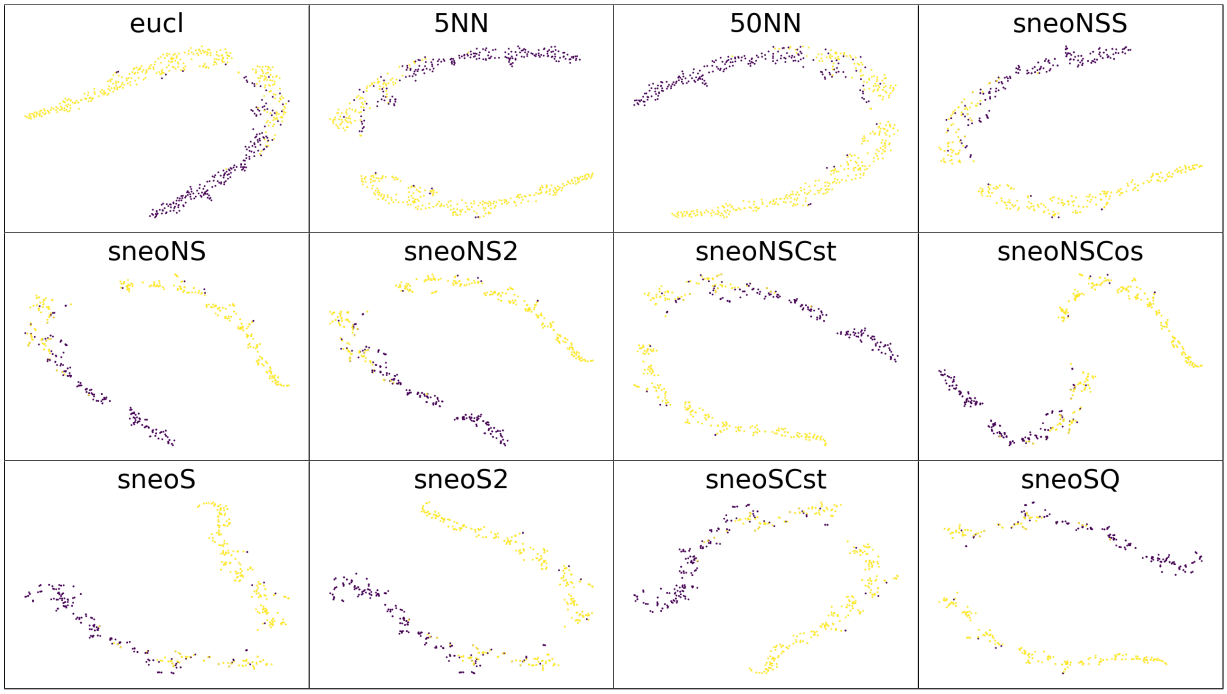



Table B.9: *Breast Cancer* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

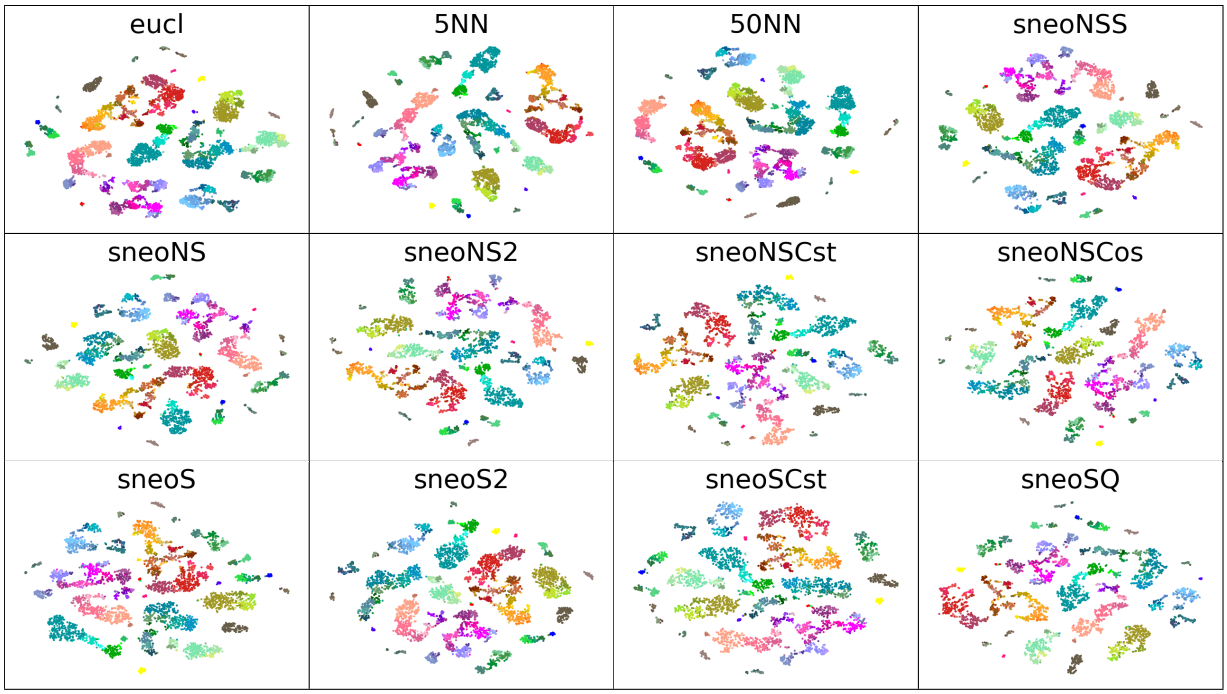



Table B.10: 10'000 observations of *Mouse* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

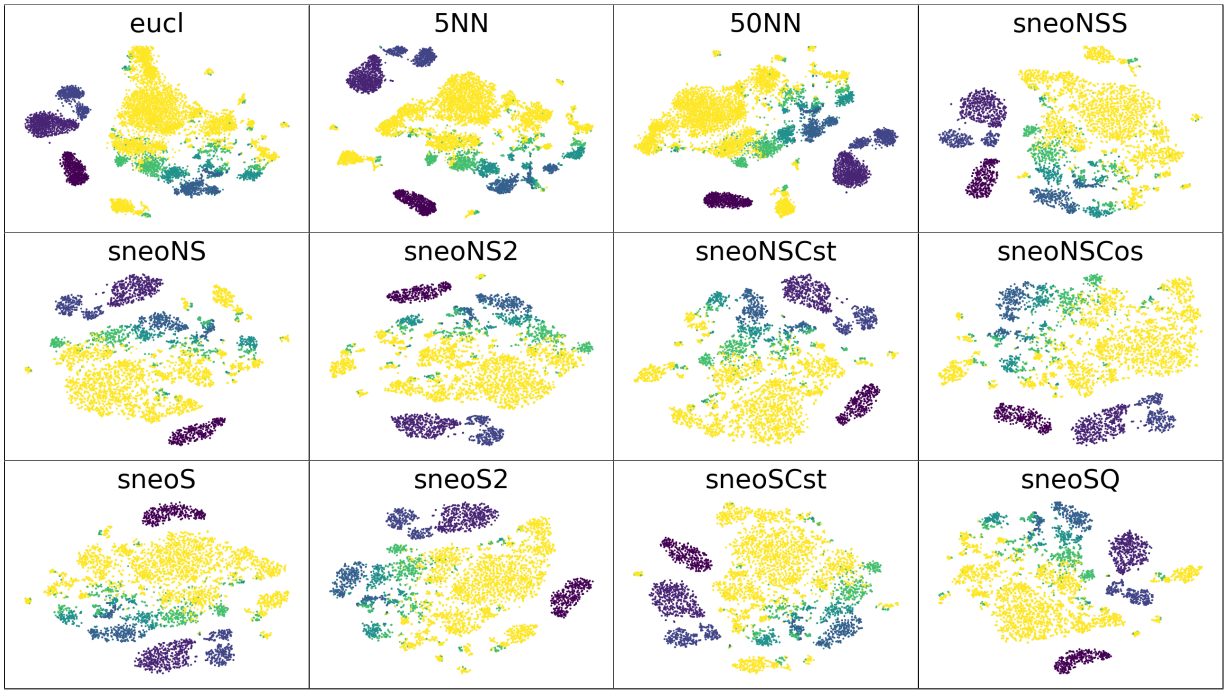



Table B.11: 10'000 observations of the *Zebrafish* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

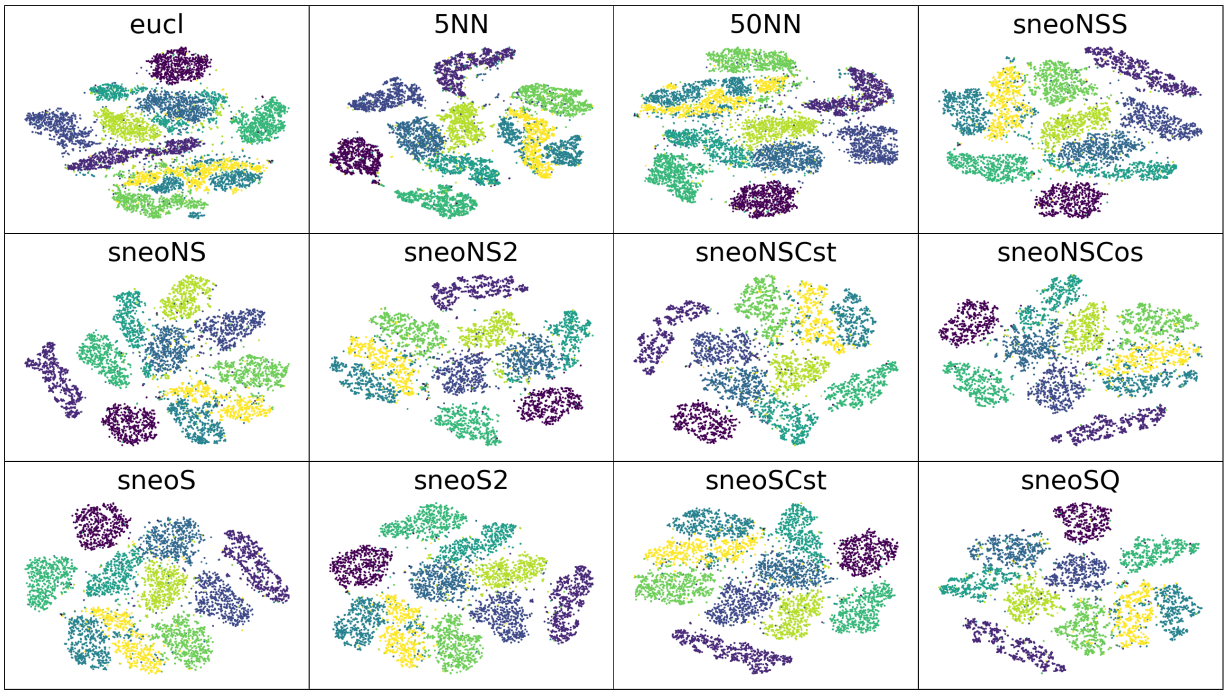



Table B.12: 10'000 observations of the *MNIST* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

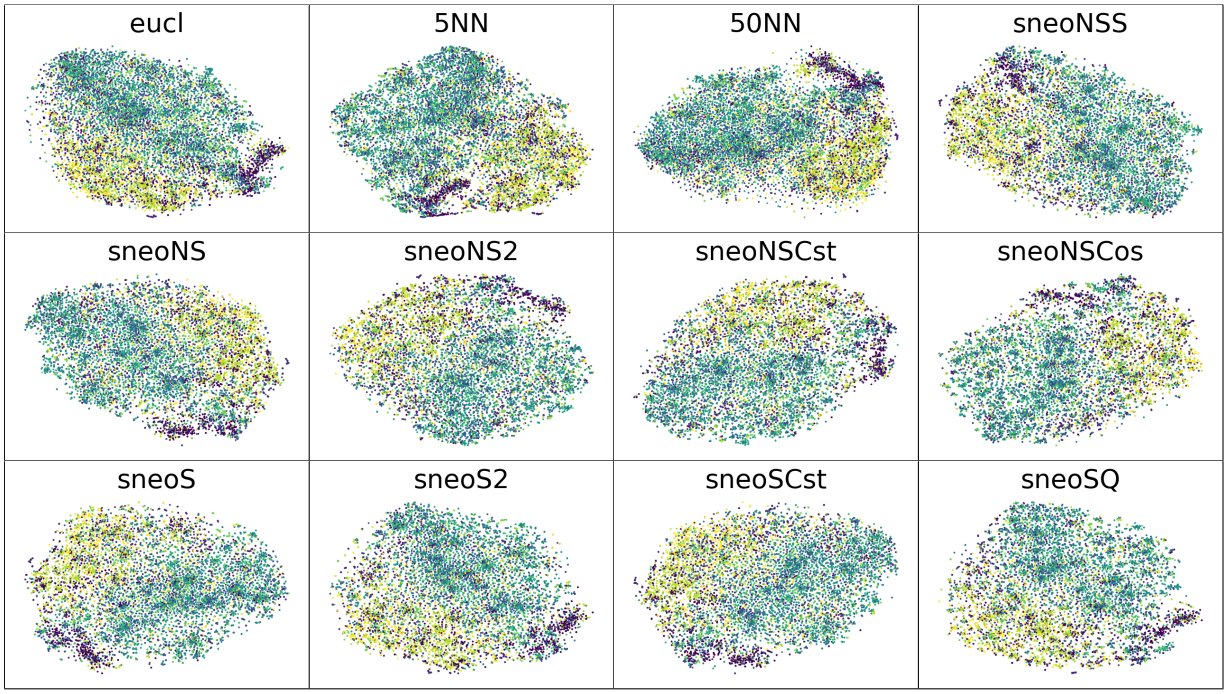



Table B.13: 10'000 observations of the *Cifar-10* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

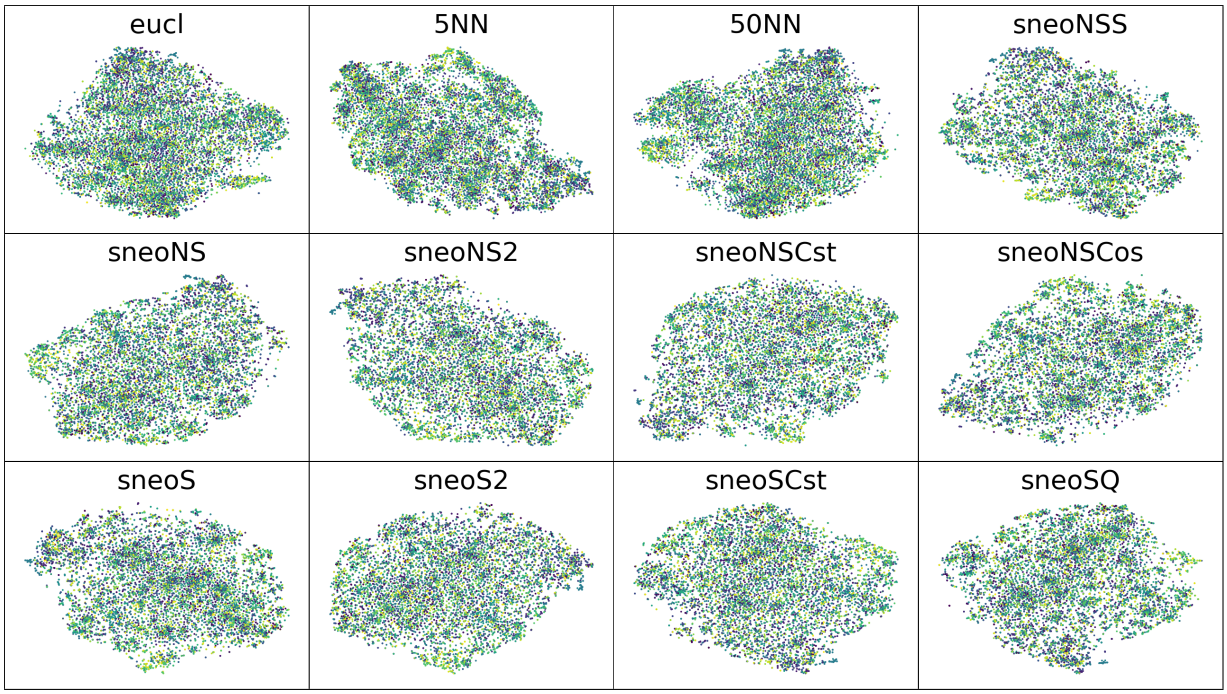



Table B.14: 10'000 observations of the *Cifar-100* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

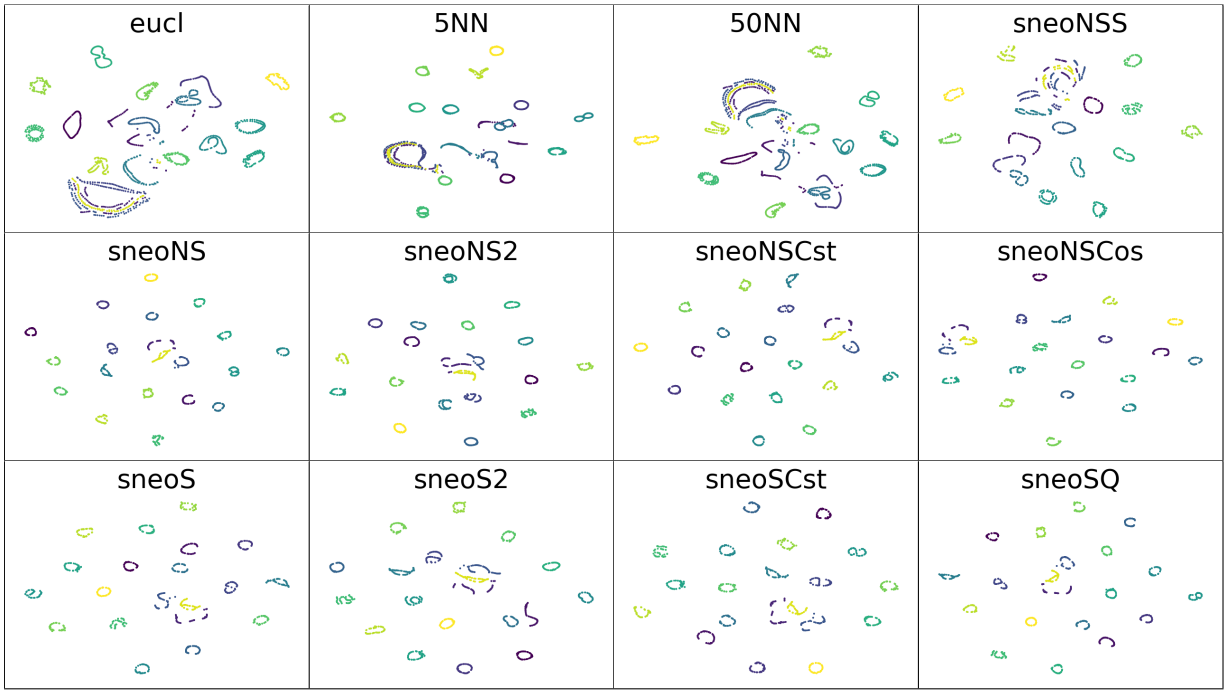



Table B.15: *Coil-20* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

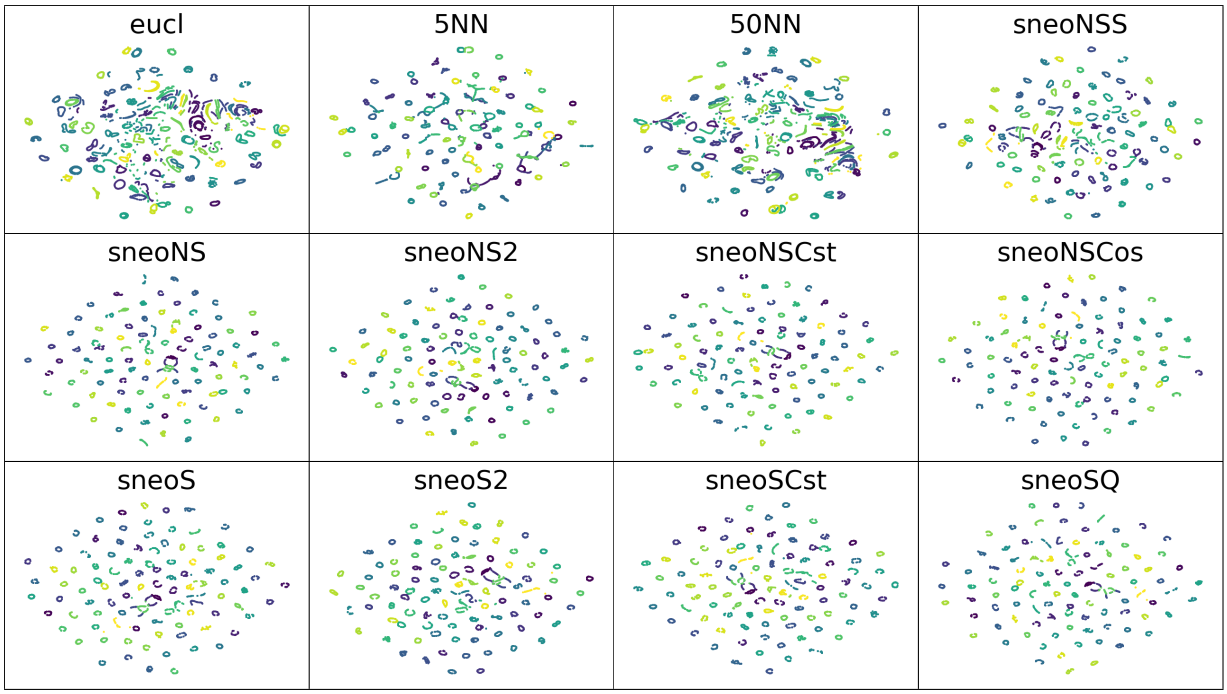



Table B.16: *Coil-100* dataset represented in 2 dimensions using *t-SNE* as dimensionality reduction method

# Appendix C

# $Q_{NX}(K)$ Plots

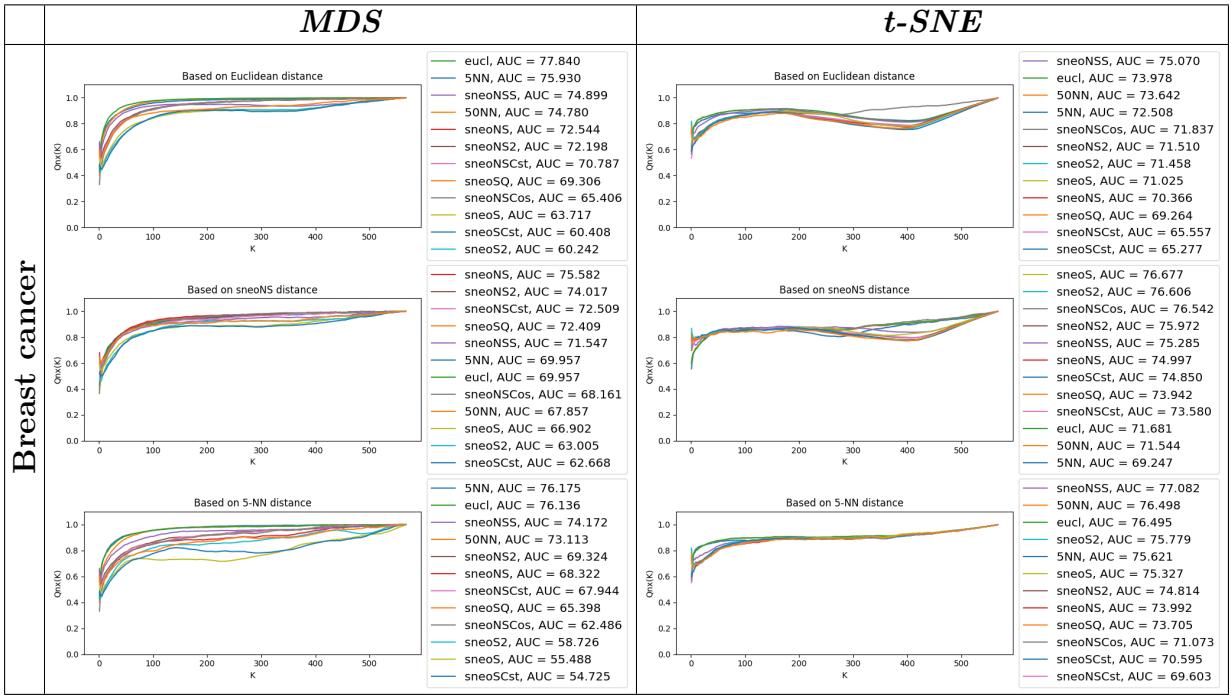



Table C.1: $Q_{NX}(K)$ through all datasets, neighbors in low and high dimensions are collected following three distances, making three sub-graphs, the legend is ordered by performance (AUC) of the distances, given to *MDS* (left column) and *t-SNE* (right column)

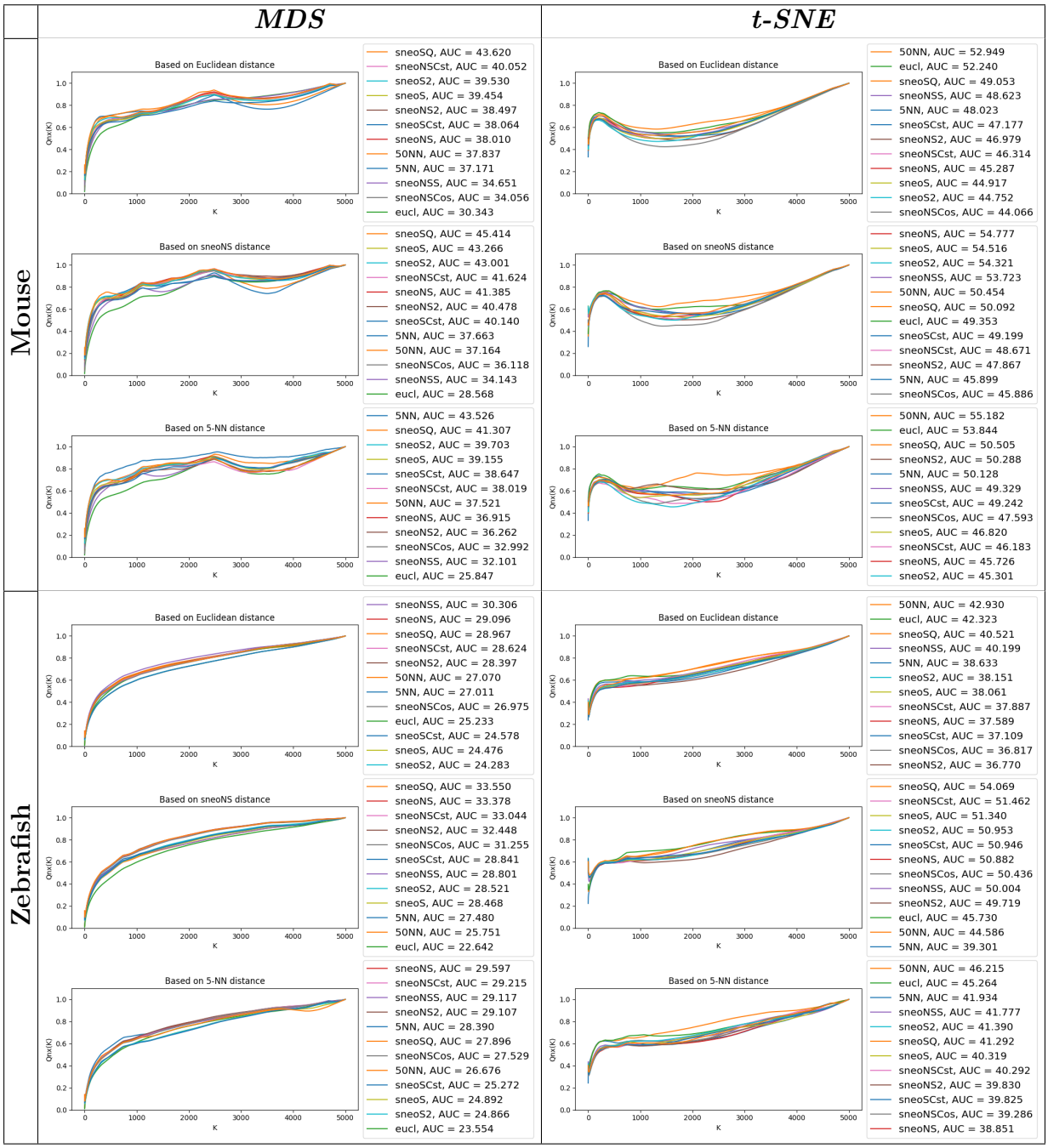



Table C.2: Follows Table C.1

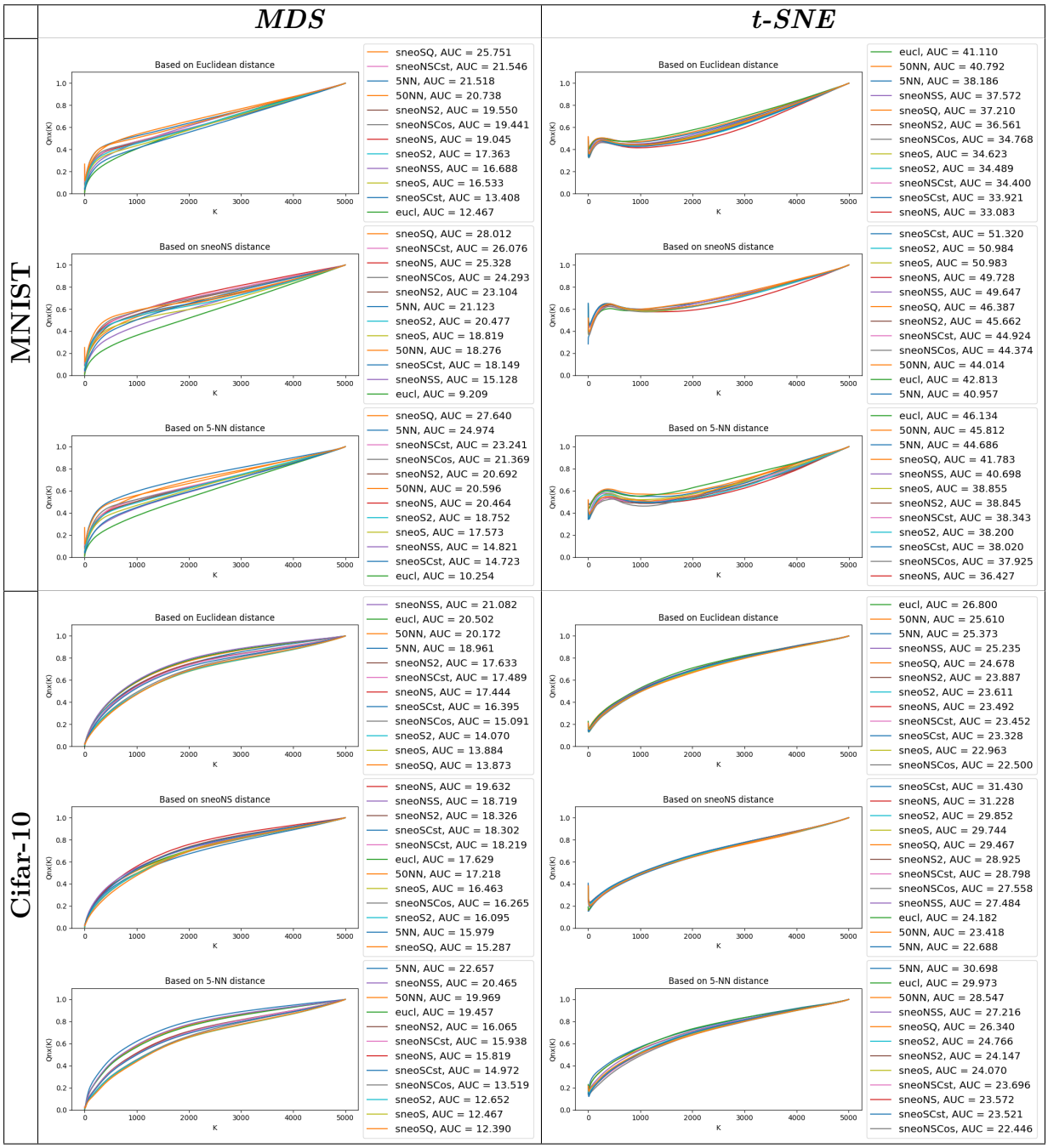



Table C.3: Follows Table C.2

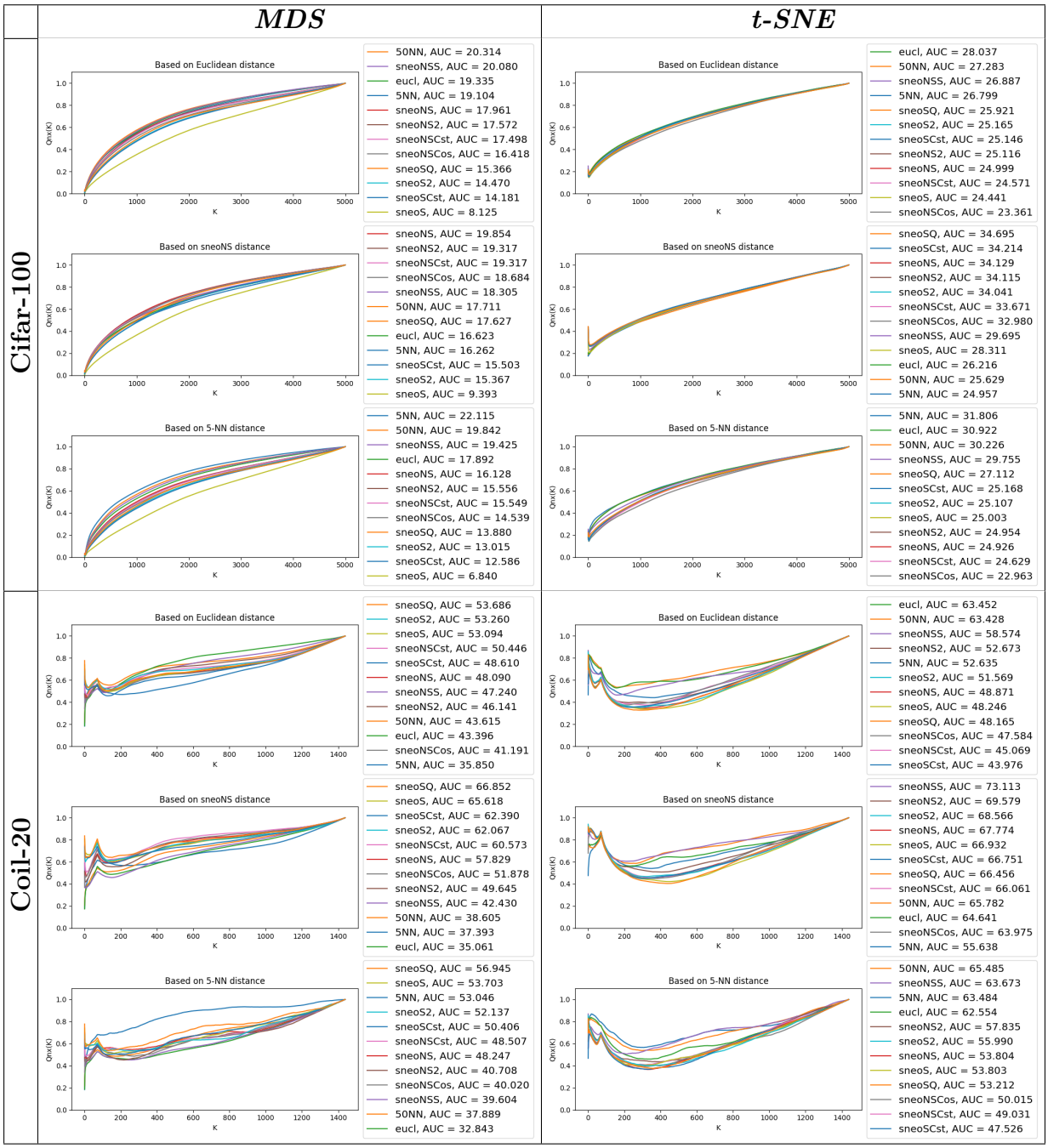



Table C.4: Follows Table C.3

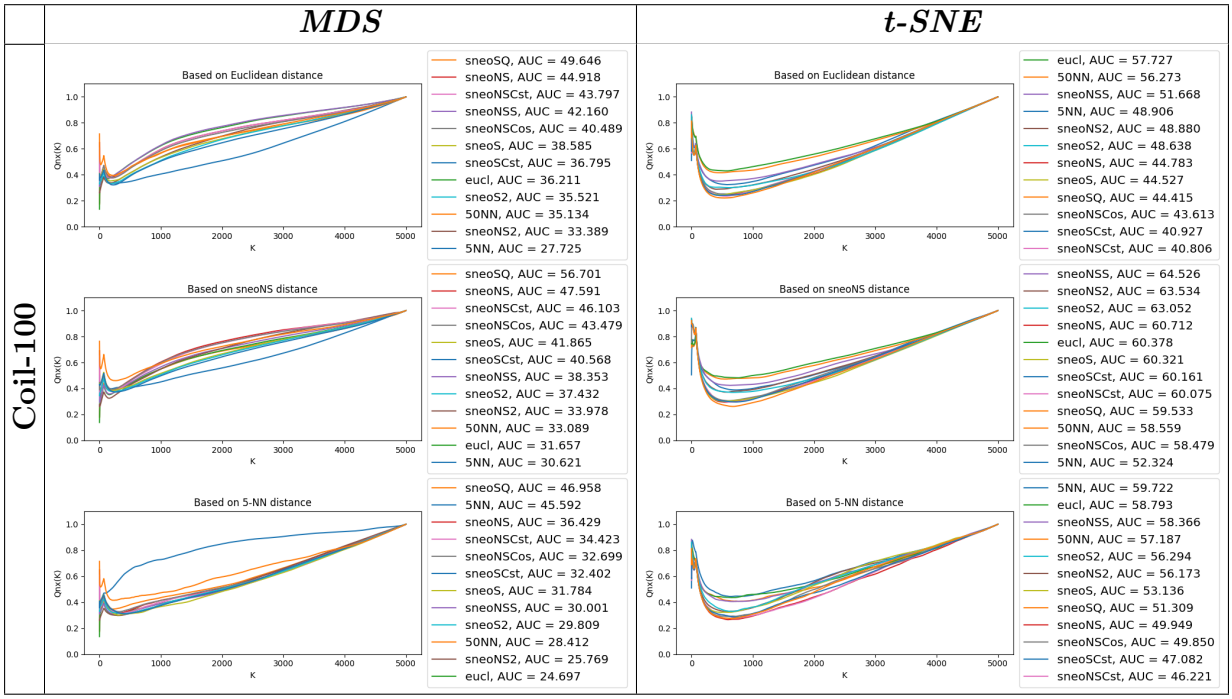



Table C.5: Follows Table C.4

# Bibliography


[1] Wikipedia contributors. *Minkowski distance — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-May-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Minkowski_distance&oldid=1222083881.

[2] Rahul Kumar. *Mahalanobis Distance*. Wall Street Mojo. 2024. URL: https://www.wallstreetmojo.com/mahalanobis-distance/.

[3] C. J.C. Burger Y. LeCun C. Cortes. *THE MNIST DATABASE of handwritten digits*. 2013. URL: http://yann.lecun.com/exdb/mnist/.

[4] S. A. Nene, S. K. Nayar, and H. Murase. *Columbia Object Image Library (COIL-20)*. Technical Report CUCS-005-96. Columbia University, Feb. 1996. URL: https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php.

[5] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. 2014. arXiv: 1404.1100 [cs.LG].

[6] Vin de Silva Joshua B. Tenenbaum and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science Mag* 290 (2000), pp. 2319–2323. URL: https://www.robots.ox.ac.uk/~az/lectures/ml/tenenbaum-isomap-Science2000.pdf.

[7] GeeksforGeeks. *Kruskal's Minimum Spanning Tree (MST) Algorithm*. 2023. URL: https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/.

[8] Francois Fouss et al. "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* 19.3 (2007), pp. 355–369. DOI: 10.1109/TKDE.2007.46.

[9] Dehua Peng, Zhipeng Gui, and Huayi Wu. *Interpreting the Curse of Dimensionality from Distance Concentration and Manifold Effect*. Tech. rep. mailto:pengdh@whu.edu.cn, mailto:zhipeng.gui@whu.edu.cn, mailto:wuhuayi@whu.edu.cn. URL: https://arxiv.org/pdf/2401.00422v1.

[10] Jake Lever, Martin Krzywinski, and Naomi Altman. "Principal component analysis". In: *Nature Methods* 14.7 (2017), pp. 641–642. ISSN: 1548-7105. DOI: 10.1038/nmeth.4346. URL: https://doi.org/10.1038/nmeth.4346.

[11] Wikipedia contributors. *Multidimensional scaling — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Multidimensional_scaling&oldid=1200389318.

[12] Jan de Leeuw and Patrick Mair. "Multidimensional Scaling Using Majorization: SMACOF in R". In: *Journal of Statistical Software* 31.3 (2009), pp. 1–30. DOI: 10.18637/jss.v031.i03. URL: https://www.jstatsoft.org/index.php/jss/article/view/v031i03.

[13] Geoffrey E. Hinton and Sam T. Roweis. "Stochastic Neighbor Embedding". In: *Neural Information Processing Systems*. 2002. URL: https://api.semanticscholar.org/CorpusID:20240.

[14] Umberto Michelucci. "An Introduction to Autoencoders". In: *CoRR* abs/2201.03898 (2022). arXiv: 2201.03898. URL: https://arxiv.org/abs/2201.03898.

[15] GeeksforGeeks. *Shortest Path Algorithm Tutorial with Problems*. 2023. URL: https://www.geeksforgeeks.org/shortest-path-algorithms-a-complete-guide/.

[16] M. Roytman A. Jaroszewicz. "Statistical Measures of Distance". In: *ACM Transactions on Applied Perception* 2.3 (2010). URL: http://www.stat.ucla.edu/~jingyi.li/statsm254-s14/survey_papers/distance_measures.pdf.

[17] John A. Lee and Michel Verleysen. "Quality assessment of dimensionality reduction: Rank-based criteria". In: *Neurocomputing* 72.7 (2009). Advances in Machine Learning and Computational Intelligence, pp. 1431–1443. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2008.12.017. URL: https://www.sciencedirect.com/science/article/pii/S0925231209000101.

[18] William Wolberg et al. *Breast Cancer Wisconsin (Diagnostic)*. UCI Machine Learning Repository. 1995. DOI: 10.24432/C5DW2B.

[19] Bosiljka Tasic et al. "Shared and distinct transcriptomic cell types across neocortical areas". In: *Nature* 563.7729 (Nov. 2018), pp. 72–78. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0654-5. URL: https://doi.org/10.1038/s41586-018-0654-5.

[20] Daniel E. Wagner et al. "Single-cell mapping of gene expression landscapes and lineage in the zebrafish embryo". In: *Science* 360.6392 (2018), pp. 981–987. DOI: 10.1126/science.aar4362. eprint: https://www.science.org/doi/pdf/10.1126/science.aar4362. URL: https://www.science.org/doi/abs/10.1126/science.aar4362.

[21] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, Apr. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[22] S. A. Nene, S. K. Nayar, and H. Murase. *Columbia Object Image Library (COIL-100)*. Technical Report CUCS-006-96. Columbia University, Feb. 1996. URL: https://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php.

[23] Ghazal Ghazaei et al. "An exploratory study on the use of convolutional neural networks for object grasp classification". In: Dec. 2015. DOI: 10.1049/cp.2015.1760.

[24] Wikipedia contributors. *Cosine similarity — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-May-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1224774490.

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