Majorizing the Laplacian SNE cost function

Master Thesis Business Administration and Quantitive Marketing

Author: Safouane Tazrouti (585981)*

Supervisor: Prof. Dr. P.J.F. Groenen

Second assessor: D.J.W. Touw

February 2022

ERASMUS SCHOOL OF ECONOMICS

*The content of this thesis is the sole responsibility of the author and does not reflect the view of the supervisor, second assessor, Erasmus School of Economics or Erasmus University.

Contents

1 Introduction

As big data continues to permeate numerous industries, many dimensionality reduction algorithms have been developed over the past decades. These algorithms attempt to transform high-dimensional data into lower dimensions while preserving the structure and properties of it. This enables visualization of real-world data, such as voice signals and pictures, which is necessary to handle and explain it adequately.

Initially only linear techniques such as Principal Components Analysis (PCA) and classical scaling were used for dimensionality reduction (Van Der Maaten et al., 2009). PCA was introduced by Pearson (1901) and independently modernized by Hotelling (1933). It uses eigenvectors and eigenvalues of a covariance or correlation matrix to deduce similarities between variables (Alexander, 2008). Classical scaling is one of the first versions of Multi Dimensional Scaling (MDS). It was introduced by Young and Householder (1938) and developed by Torgerson (1958). Later, Gower (1966) showed that classical scaling is closely related to PCA and named it principal co-ordinate analysis. Kruskal (1964a,b) came up with an approach that has become the most widely used version of MDS. He defined the cost function, which he called stress, as the residual sum of squares which allows to fit Euclidean distances to potential dissimilarities directly.

While MDS has been studied for over half a century, modern techniques such as Stochastic Neighbor Embedding (SNE) and t-distributed SNE (t-SNE), introduced to handle nonlinear data, receive much attention lately (Poličar et al., 2019). Since real-world data often contain nonlinear structures and patterns, these nonlinear techniques outperform the linear dimensionality reduction algorithms on complex visualization tasks (Van Der Maaten et al., 2009). SNE had been introduced by Geoffrey Hinton and Sam Roweis back in 2002 as an algorithm to present high-dimensional objects in a low dimensional space preserving neighbor identities (Hinton and Roweis, 2002). Although SNE led to much better solutions than alternative nonlinear embedding techniques, it was mainly criticized for the so called crowding problem which will be discussed in Section 3.1.1 and for its cost function that is difficult to optimize. Cook et al. (2007) for example, addresses the fact that SNE minimizes the divergence between conditional distributions which brings inconvenient properties. As a solution, the paper introduced Symmetric SNE which uses a single joint distribution. This produced a simpler cost function which leads to simpler derivatives. Hinton later used this cost function to improve

his SNE algorithm in collaboration with Laurens van der Maaten to the so called t-distributed SNE.

This version became very popular especially for the visualization of so called gene expression data that are obtained from the RNA sequencing of cells (Kobak and Berens, 2019). According to Kobak and Berens (2019), the popularity in this field is partly due to the lack of serious competitors. Therefore some of the well known downsides of SNE and t-SNE are frequently overlooked. One of these downsides which has been addressed by Wattenberg et al. (2016) among others, is that different runs with the same parameters yield different results for certain data sets. This was also highlighted by Van der Maaten and Hinton (2008). The reason for this shortcoming is the non-convexity of the t-SNE cost function. The authors, however, think that this weakness is insufficient reason to reject t-SNE in favor of methods that require convex optimization. Although this statement may be plausible, in some situations a correct visualization of the global geometry is essential as indicated by Kobak and Berens (2019).

With the progress of machine learning, researchers have managed to improve non-convex optimization algorithms in recent years (Arora et al., 2018). However, most improvements speed up the optimization process only and do not remove the stochastic nature of gradient descent optimization. A way of dealing with this problem is by iteratively constructing a so called majorizing function which is convex and thus easier to optimize. Due to the paper of Lange et al. (2000), this concept is known as minimization by majorization or maximization by minorization (MM). The general principle behind the MM algorithm was introduced by Ortega and Rheinboldt (1970) in the context of line search. De Leeuw and Heiser (1977) developed the MM algorithm for MDS and Jan de Leeuw has made many theoretical contributions to this method (e.g. De Leeuw (1977) and De Leeuw (1988) among other works).

In this thesis we propose a procedure that can be used to remove the stochastic characteristic of SNE techniques. To do so we introduce a version of SNE, which we call Laplacian SNE, that solves the crowding problem of SNE. We then perform several majorizations to the Laplacian SNE cost function. We will assess the performance of this majorized version, which we call Majorized SNE, to see whether majorization can be applied on SNE cost functions. Accordingly, the research question of this thesis is formulated as *Can we develop a procedure which improves the optimizing process of SNE cost functions with majorization, such that different runs on the same data yield*

the same solution?

The outline of this thesis is as follows. Sections 2 and 3 discuss MDS and SNE briefly. In Section 3.1.1 we discuss the crowding problem and introduce Laplacian SNE. Section 4 explains the MM algorithm and gives an illustration of it. In Section 5 we introduce the proposed procedure of Majorized SNE including all derivations of the majorization. The section concludes with a clear algorithm for Majorized SNE. In Section 6 we will implement the proposed algorithm on data from the MNIST[1] data base and analyze the results. Finally, in Section 6 the research question will be answered and future research suggestions will be made.

[1] A data base of 60 000 gray scale images of handwritten digits.

Notation

To keep this thesis organized, Table 1 provides an overview of the notation used. In general, a lowercase italic character denotes a scalar, a lowercase bold character denotes a vector and an uppercase bold character denotes a matrix.

Notation	Definition
$\mathbf{x}_i$	The coordinate of data point i in the high dimensional space.
$\mathbf{y}_i$	The coordinate of data point i in the low dimensional space.
$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$	The high dimensional data with N observations.
$\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\}$	The low dimensional representation of $\mathbf{X}$.
$\mathbf{Y}^{(0)}$	The supporting point and current estimate of $\mathbf{Y}$.
σ_i	The variance of the Gaussian centered at point i.
p_{ij}	The pairwise similarity between points i and j in the high dimensional space.
$\mathbf{P}$	An $n \times n$ matrix filled with the high dimensional pairwise similarities.
$\mathbf{p}$	The vectorization of the upper triangle elements of matrix $\mathbf{P}$.
q_{ij}	The pairwise similarity between points i and j in the low dimensional space.
$\mathbf{Q}$	An $n \times n$ matrix filled with the low dimensional pairwise similarities.
$\mathbf{q}$	The vectorization of the upper triangle elements of matrix $\mathbf{Q}$.
$\delta_{ij} = \|\|\mathbf{x}_i - \mathbf{x}_j\|\|$	The Euclidean distance between points i and j in the high dimensional space.
$\mathbf{\Delta}$	An $n \times n$ matrix filled with Euclidean distances between the points in the high dimensional space.
$d_{ij} = \|\|\mathbf{y}_i - \mathbf{y}_j\|\|$	The Euclidean distance between points i and j in the low dimensional space.
$\mathbf{D}$	An $n \times n$ matrix filled with Euclidean distances between the points in the low dimensional space.
$\mathbf{d}$	The vectorization of the upper triangle elements of matrix $\mathbf{D}$.
$\mathbf{d}^{(0)}$	The supporting point of $\mathbf{d}$ which is a function of the current estimate $\mathbf{Y}^{(0)}$.
$\mathbf{E}_{ij}$	An $n \times n$ matrix with zeros everywhere except $e_{ii} = e_{jj} = 1$ and $e_{ij} = e_{ji} = -1$.
$C(\mathbf{d})$	The SNE cost function or Kullback-Leibler divergence between $\mathbf{P}$ and $\mathbf{Q}$.
$f(\mathbf{d}, \mathbf{d}^{(0)})$	The majorizing function of C.
$\omega_{stress}(\mathbf{Y}, \mathbf{\Delta})$	The MDS cost function.
$\mathcal{F}(\delta_{ij})$	The representation function, a function of the high dimensional distances δ_{ij}.

Table 1: An overview of important variables and functions with their definitions.

2 Multidimensional Scaling

Multidimensional scaling (MDS) aims to visualize high dimensional data in a low dimensional space such that the (Euclidean) distances in the lower dimension, optimally represent the high dimensional dissimilarities between pairs of data points (Groenen and van de Velden, 2016).

There are numerous variants of MDS. They use different types of geometry preferences, treat statistical error in the models differently and the algorithms to find an optimal data visualization differ per technique (Borg and Groenen, 2005). Regardless of these differences, most versions of MDS use a loss function which can be written as

$$\omega_{stress}(\mathbf{Y}, \mathbf{\Delta}) = \sum_{i<j} w_{ij}[\mathcal{F}(\delta_{ij}) - d_{ij}(\mathbf{Y})]^2, \tag{1}$$

where δ_{ij} represents the Euclidean distance between the high dimensional data points i and j and $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\}$ is the low dimensional representation of the high dimensional data. This loss function, often called the raw stress function, represents the errors between the representation function $\mathcal{F}(\delta_{ij})$ from the high dimensional data and the low-dimensional distances $d_{ij}(\mathbf{Y})$ over half of the data (Borg and Groenen, 2005). The goal in MDS is to minimize $\omega_{stress}(\mathbf{Y}, \mathbf{\Delta})$ over $\mathbf{Y}$.

2.1 The SMACOF algorithm for MDS

The acronym SMACOF stands for Scaling by Majorizing a Complicated Function. It is used in MDS to obtain an optimal solution for the low dimensional representation $\mathbf{Y}$. To do so, SMACOF majorizes the stress function and minimizes the obtained majorization function. This optimization method is called Minimization by Majorization and will be discussed in Section 4. The solution obtained from this procedure tends, due to the so called sandwich inequality, to be the optimal solution for the stress function as well. A clear and comprehensive explanation of how SMACOF is derived can be found in Borg and Groenen (2005). The main idea and parts of this technique will be used in Section 5 where we construct our proposed algorithm.

3 Stochastic Neighbor Embedding

Stochastic Neighbor Embedding (SNE) tries to find a low-dimensional representation $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\}$ for a high-dimensional data set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ while preserving neighborhood identity. It uses similarities which are obtained by converting high-dimensional Euclidean distances between the data points into probabilities. The goal is then to find a low-dimensional representation which minimizes the sum of the divergence between these probabilities and probabilities of the representation of the points in the lower dimension (Van Der Maaten, 2014). This way the clusters on the low-dimensional output correspond as much as possible to the same clusters in the high-dimensional input objects.

3.1 Symmetric Stochastic Neighbor Embedding

Symmetric SNE was introduced by Cook et al. (2007) as a simpler version of the original SNE algorithm. It uses joint probability distributions instead of conditional ones. This causes the probabilities to be symmetric. Due to this property the authors named this version Symmetric SNE. The pairwise similarities q_{ij} in the low-dimensional map are given by

$$q_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k<l} \exp(-d_{kl}^2)}, \tag{2}$$

where d_{ij} is the Euclidean distance between points i and j in the lower dimension. The pairwise similarities in the high-dimensional space p_{ij} are given by

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}, \tag{3}$$

with

$$p_{j|i} = \frac{\exp(-\delta_{ij}^2/2\sigma_i^2)}{\sum_{k<i} \exp(-\delta_{ik}^2/2\sigma_i^2)}, \tag{4}$$

which corresponds to the conditional probability that point i would pick j as its neighbor if neighbors were picked in proportion to their Gaussian centered at i, with variance σ_i. To find values for σ_i, SNE performs a binary search while taking into account a fixed perplexity. This perplexity can be interpreted as the effective number of neighbors of an observation and typically takes values between 5 and 50. δ_{ij} again represents the Euclidean distance between points i and j in the high-dimensional space. Finally, the cost function is defined as the Kullback-Leibler divergence between the joint distributions $\mathbf{P}$ and $\mathbf{Q}$

$$C(\mathbf{d}) = KL(\mathbf{P}||\mathbf{Q}) = \sum_{i<j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{5}$$

3.1.1 Laplacian Stochastic Neighbor Embedding

As mentioned earlier, one of the issues of SNE is the so called crowding problem. This problem occurs when the available space to accommodate data points that lie far away from each other, is not as large as the space that is available to accommodate data points close to each other. This causes data points to be placed too far away in the low-dimensional map. Van der Maaten and Hinton (2008) deals deals with this problem by using the kernel of a normalized t-distribution instead of a Gaussian kernel to convert distances into probabilities q_{ij}. The heavy tails of this distribution create more space in the low-dimensional embedding which allows t-SNE to model small pairwise distances more accurately than when a Gaussian kernel is used. Based on this idea we propose to use the kernel of the Laplace distribution which also has fatter tails than the Gaussian distribution (see Figure 1). However, the Laplacian kernel has simpler derivatives and is therefore more convenient to use. In Section 5 we will need this property because it allows us to use Theorem 1 of Groenen and Josse (2016). This Theorem introduces a majorizing function that guarantees monotone convergence. Using the Laplacian kernel we define the pairwise similarities q_{ij} in the low-diensional map as

$$q_{ij} = \frac{\exp(-d_{ij})}{\sum_{k \neq l} \exp(-d_{ij})}. \tag{6}$$

The pairwise similarities in the high-dimensional space p_{ij} and the cost function C remain the same as the ones used for Symmetric SNE.

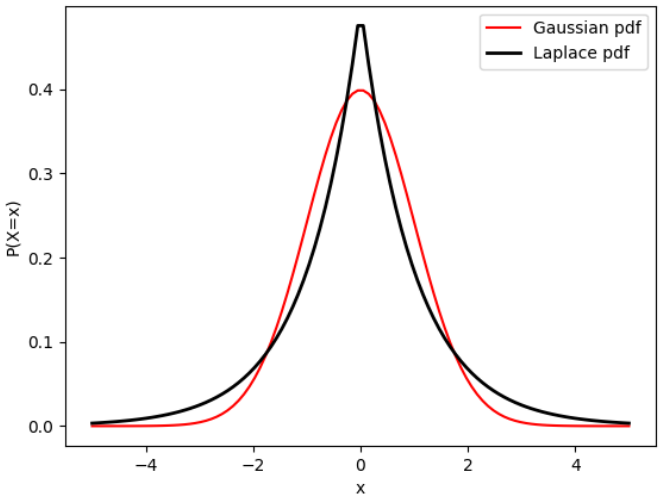


Figure 1: The pdf's of a Gaussian and Laplace distribution both with location 0 and variance 1. From this example it becomes very clear that the Laplace distributions has fatter tails than the Gaussian.

4 Minimization by majorization

In SNE, a gradient descent method is used to minimize its non-convex cost function. A simpler and more powerful way of dealing with non-convex optimization is to use a linear or quadratic majorizing function (Groenen and van de Velden, 2016). This technique is called minimization by majorization (MM). It guarantees global convergence and gives similar results for different runs when initialized with the same starting point. Therefore we will apply MM on the Laplacian SNE cost function. First we will explain in this section how this method works. In Section 5 we will construct a majorizing function for the Laplacian SNE cost function.

Let $\mathbf{d}$ be the vectorization of the upper triangle elements of matrix $\mathbf{D}$ filled with Euclidean distances between every pair of points in the low dimensional space. Then the idea of MM is to construct a surrogate function $f(\mathbf{d}, \mathbf{d}^{(0)})$ that touches the objective function at the supporting point $\mathbf{d}^{(0)}$ and equals or lies above the objective function $C(\mathbf{d})$ anywhere else. By minimizing $f(\mathbf{d}, \mathbf{d}^{(0)})$, the next supporting point $\mathbf{d}^{(1)}$ is obtained. This supporting point is then used to construct the majorizing function for the next iteration and so forth. This procedure guarantees descent because the surrogate functions are convex and thus their minimums are lower than or equal the supporting point in each iteration. Figure 2 two illustrates the majorization procedure for two data points.

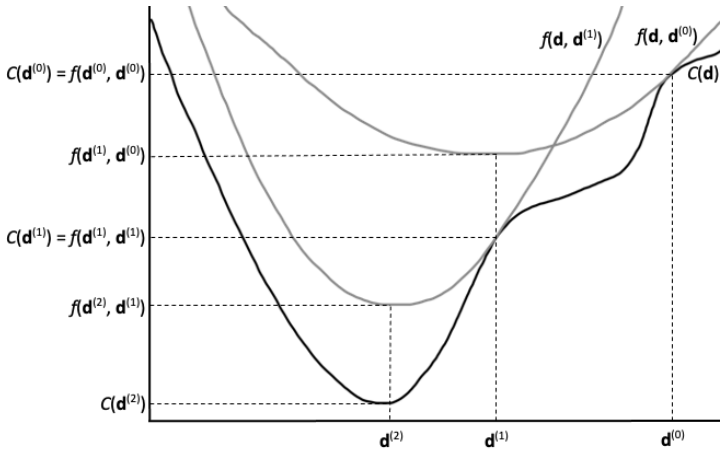


Figure 2: The idea of Minimization by Majorization. (Based on Figure 8.4 from Borg and Groenen (2005))

The algorithm starts at $\mathbf{d}^{(0)}$. The first majorization function $f(\mathbf{d}, \mathbf{d}^{(0)})$ is fitted by matching the value and first derivative of $C(\mathbf{d})$ at $\mathbf{d}^{(0)}$. In the next step $f(\mathbf{d}, \mathbf{d}^{(0)})$ is minimized to find the next supporting point $\mathbf{d}^{(1)}$. The same steps are then executed to find point $\mathbf{d}^{(2)}$, etc.

5 Majorized SNE

To construct a majorizing function for the Kullback-Leiber divergence in equation (5), we use the one proposed by Theorem 1 of Groenen and Josse (2016). This theorem states that the cost function $C(\mathbf{d})$ can be majorized such that

$$C(\mathbf{d}) \leq C(\mathbf{d}^{(0)}) + (\mathbf{d} - \mathbf{d}^{(0)})'\nabla C(\mathbf{d}^{(0)}) + 1/4||\mathbf{d} - \mathbf{d}^{(0)}||^2 = f(\mathbf{d}, \mathbf{d}^{(0)}). \tag{7}$$

For this theorem to be true, the Hessian of the objective function $C(\mathbf{d})$, with respect to the squared elements of $\mathbf{d}$, must have eigenvalues which are smaller than $1/2$. We will show that this is true. First we derive the gradient and Hessian of $C(\mathbf{d})$. For convenience, we define $a_{ij} = \exp(-d_{ij})$ and $b = \sum_{k<l} \exp(-d_{kl})$ and rewrite

$$C(\mathbf{d}) = \sum_{i<j} \left[p_{ij} \log p_{ij} - p_{ij} \log \left(\frac{\exp(-d_{ij})}{\sum_{k<l} \exp(-d_{kl})} \right) \right] = \sum_{i<j} \left[p_{ij} \log p_{ij} + p_{ij} d_{ij} + p_{ij} \log(b) \right]. \tag{8}$$

Then, for $\forall g \in \{1, 2, ..., n-1\}$ and $h \in \{1, 2, ..., n\}$

$$\frac{\partial C}{\partial d_{gh}} = p_{gh} - \sum_{i<j} p_{ij} \frac{a_{gh}}{b} = p_{gh} - q_{gh} \sum_{i<j} p_{ij} = p_{gh} - q_{gh}, \tag{9}$$

because $\sum_{i<j} p_{ij} = 1$. Using this expression, we can write the gradient of the cost function as

$$\nabla C(\mathbf{d}) = \begin{pmatrix} \frac{\partial C}{\partial d_{12}} \\ \frac{\partial C}{\partial d_{13}} \\ \vdots \\ \frac{\partial C}{\partial d_{gh}} \\ \vdots \\ \frac{\partial C}{\partial d_{(n-1),n}} \end{pmatrix} = \begin{pmatrix} p_{12} - q_{12} \\ p_{13} - q_{13} \\ \vdots \\ p_{gh} - q_{gh} \\ \vdots \\ p_{(n-1),n} - q_{(n-1),n} \end{pmatrix} = \mathbf{p} - \mathbf{q}, \tag{10}$$

where $\mathbf{p}$ and $\mathbf{q}$ are vectorizations of the upper triangle elements of matrices $\mathbf{P}$ and $\mathbf{Q}$ respectively.

Next we have

$$\frac{\partial^2 C}{\partial d_{gh} \partial d_{gh}} = -\frac{\partial q_{gh}}{\partial d_{gh}} = -\frac{-b a_{gh} + a_{gh} a_{gh}}{b^2} = q_{gh} - q_{gh}^2, \tag{11}$$

and for $r, s \neq g, h$

$$\frac{\partial^2 C}{\partial d_{gh} \partial d_{rs}} = -\frac{\partial q_{gh}}{\partial d_{rs}} = -\frac{a_{gh}}{b^2} a_{rs} = -q_{gh} q_{rs}. \tag{12}$$

By combining these expression we can write the Hessian of the cost function as

$$\nabla^2 C(\mathbf{d}) = \begin{pmatrix} q_{12} - q_{12}^2 & -q_{12}q_{13} & \cdots & -q_{12}q_{(n-1),n} \\ -q_{13}q_{12} & q_{13} - q_{13}^2 & & \vdots \\ \vdots & & \ddots & \vdots \\ -q_{(n-1),n}q_{13} & \cdots & \cdots & q_{(n-1),n} - q_{(n-1),n}^2 \end{pmatrix} = \text{Diag}(\mathbf{q}) - \mathbf{q}\mathbf{q}'. \tag{13}$$

Finally we use Gerschgorin disks, which say that the eigenvalue ϕ is always smaller than a diagonal element plus the sum of its absolute off-diagonal row (or column) values, to obtain an upper bound for the eigenvalues of this Hessian. Doing so we find that

$$\begin{aligned} \phi &\leq q_{gh} - q_{gh}^2 + q_{gh} \sum_{ij \neq gh} q_{ij} \\ &= q_{gh} - q_{gh}^2 - q_{gh}^2 + q_{gh} \sum_{i<j} q_{ij} \\ &= q_{gh} - 2q_{gh}^2 + q_{gh} \quad (\text{Because} \sum_{i<j} q_{ij} = 1) \\ &= 2(q_{gh} - q_{gh}^2) = 2q_{gh}(1 - q_{gh}). \end{aligned} \tag{14}$$

It can be verified that $2q_{gh}(1 - q_{gh})$ reaches its maximum of $1/2$ at $q_{gh} = 1/2$. So, the maximum eigenvalue of $\nabla^2 C(\mathbf{d})$ is always smaller than or equal to $1/2$.

We can now use expression (7) to derive the Laplacian SNE majorizing function

$$\begin{aligned} f(\mathbf{d}, \mathbf{d}^{(0)}) &= C(\mathbf{d}^{(0)}) + (\mathbf{d} - \mathbf{d}^{(0)})'(\mathbf{p} - \mathbf{q}^{(0)}) + 1/4||\mathbf{d} - \mathbf{d}^{(0)}||^2 \\ &= \frac{1}{4} \sum_{i<j} \left(d_{ij} - d_{ij}^{(0)} \right)^2 + \mathbf{d}'(\mathbf{p} - \mathbf{q}^{(0)}) + C(\mathbf{d}^{(0)}) - \mathbf{d}^{(0)\prime}(\mathbf{p} - \mathbf{q}^{(0)}) \\ &= \frac{1}{4}\mathbf{d}'\mathbf{d} + \mathbf{d}'\mathbf{p} - \mathbf{d}'(\frac{1}{2}\mathbf{d}^{(0)} + \mathbf{q}^{(0)}) + C(\mathbf{d}^{(0)}) - \mathbf{d}^{(0)\prime}(\mathbf{p} - \mathbf{q}^{(0)}) + \frac{1}{4}\mathbf{d}^{(0)\prime}\mathbf{d}^{(0)}. \end{aligned} \tag{15}$$

Before analyzing this majorizing function, we introduce some convenient notation for the squared Euclidean distance

$$\begin{aligned} d_{ij}^2 = d_{ij}^2(\mathbf{Y}) &= \sum_{s=1}^{S} (y_{is} - y_{js})^2 \\ &= \sum_{s=1}^{S} \mathbf{y}_s'(\mathbf{i}_i - \mathbf{i}_j)(\mathbf{i}_i - \mathbf{i}_j)'\mathbf{y}_s \\ &= \text{tr}\mathbf{Y}'(\mathbf{i}_i - \mathbf{i}_j)(\mathbf{i}_i - \mathbf{i}_j)'\mathbf{Y} \\ &= \text{tr}\mathbf{Y}'\mathbf{E}_{ij}\mathbf{Y}, \end{aligned} \tag{16}$$

with $\mathbf{y}_s$ column s of $\mathbf{Y}$ and $\mathbf{i}_i$ the i-th column of the identity matrix $\mathbf{I}$ so that the matrix $\mathbf{E}_{ij}$ has zeros everywhere except $e_{ii} = e_{jj} = 1$ and $e_{ij} = e_{ji} = -1$. Using this notation, we can write the first term of equation (15) as $\text{tr}\mathbf{Y}'(\frac{1}{4}\sum_{i<j}\mathbf{E}_{ij})\mathbf{Y}$. The second part of equation (15) needs to be further majorized because it is linear in $\mathbf{d}$. A majorization for this part can be found through

$$\begin{aligned}(d_{ij} - d_{ij}^{(0)})^2 &\geq 0 \\ 2d_{ij}d_{ij}^{(0)} &\leq d_{ij}^2 + d_{ij}^{(0)2} \\ d_{ij} &\leq \frac{d_{ij}^2}{2d_{ij}^{(0)}} + \frac{1}{2}d_{ij}^{(0)}. \end{aligned} \tag{17}$$

Using expressions (16) and (17) we can majorize $\mathbf{d}'\mathbf{p}$ by $\text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{p_{ij}}{2d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y} + \frac{1}{2}\mathbf{d}^{(0)\prime}\mathbf{p}$. Finally, the negative part of equation (15) can be majorized using the Cauchy-Schwartz inequality. Through this inequality we have

$$\begin{aligned} d_{ij}d_{ij}^{(0)} &\geq tr\mathbf{Y}'\mathbf{E}_{ij}\mathbf{Y}^{(0)} \\ -d_{ij}d_{ij}^{(0)} &\leq -tr\mathbf{Y}'\mathbf{E}_{ij}\mathbf{Y}^{(0)} \\ -d_{ij} &\leq -tr\mathbf{Y}'(d_{ij}^{(0)-1}\mathbf{E}_{ij})\mathbf{Y}^{(0)}, \end{aligned} \tag{18}$$

where $\mathbf{Y}^{(0)}$ is the supporting point of $\mathbf{Y}$. Using this result we can majorize $-\mathbf{d}'(\frac{1}{2}\mathbf{d}^{(0)} + \mathbf{q}^{(0)})$ by $-\text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{d_{ij}^{(0)}+2q_{ij}^{(0)}}{2d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y}^{(0)}$. Combining these results gives the majorizing inequality

$$\begin{aligned} C(\mathbf{d}) &\leq f(\mathbf{d}, \mathbf{d}^{(0)}) \\ &\leq \text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{1}{4}\mathbf{E}_{ij}\right)\mathbf{Y} + \text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{p_{ij}}{2d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y} - \text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{d_{ij}^{(0)}+2q_{ij}^{(0)}}{2d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y}^{(0)} \\ &\quad + \frac{1}{2}\mathbf{d}^{(0)\prime}\mathbf{p} + C(\mathbf{d}^{(0)}) - \mathbf{d}^{(0)\prime}(\mathbf{p} - \mathbf{q}^{(0)}) + \frac{1}{4}\mathbf{d}^{(0)\prime}\mathbf{d}^{(0)} \\ &= \text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{d_{ij}^{(0)}+2p_{ij}}{4d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y} - 2\text{tr}\mathbf{Y}'\left(\sum_{i<j}\frac{d_{ij}^{(0)}+2q_{ij}^{(0)}}{4d_{ij}^{(0)}}\mathbf{E}_{ij}\right)\mathbf{Y}^{(0)} + \zeta(\mathbf{d}^{(0)}) \\ &= \text{tr}\mathbf{Y}'\mathbf{A}(\mathbf{Y}^{(0)})\mathbf{Y} - 2\text{tr}\mathbf{Y}'\mathbf{B}(\mathbf{Y}^{(0)})\mathbf{Y}^{(0)} + \zeta(\mathbf{d}^{(0)}), \end{aligned} \tag{19}$$

with $\zeta(\mathbf{d}^{(0)}) = \frac{1}{2}\mathbf{d}^{(0)\prime}\mathbf{p} + C(\mathbf{d}^{(0)}) - \mathbf{d}^{(0)\prime}(\mathbf{p} - \mathbf{q}^{(0)}) + \frac{1}{4}\mathbf{d}^{(0)\prime}\mathbf{d}^{(0)}$. Since this function is quadratic in $\mathbf{Y}$, and thus convex, we can easily equate its first derivative to zero and solve for $\mathbf{Y}$. Updates for

$\mathbf{Y}$ can then be obtained from

$$\begin{aligned} 2\mathbf{A}(\mathbf{Y}^{(0)})\mathbf{Y} - 2\mathbf{B}(\mathbf{Y}^{(0)})\mathbf{Y}^{(0)} &= \mathbf{0} \\ \mathbf{A}(\mathbf{Y}^{(0)})\mathbf{Y} &= \mathbf{B}(\mathbf{Y}^{(0)})\mathbf{Y}^{(0)} \\ \mathbf{Y}^{+} &= \mathbf{A}^{-}(\mathbf{Y}^{(0)})\mathbf{B}(\mathbf{Y}^{(0)})\mathbf{Y}^{(0)}, \end{aligned} \tag{20}$$

where $\mathbf{A}^{-}(\mathbf{Y}^{(0)})$ is the Moore-Penrose inverse of $\mathbf{A}(\mathbf{Y}^{(0)})$.

To conclude this section, Algorithm 1 provides a summary of the majorization algorithm for Laplacian SNE. It takes a high dimensional data set $\mathbf{X}$ as input and outputs a low dimensional representation $\mathbf{Y}$ of the data.

Algorithm 1: Majorized SNE

Pre-process the data using PCA ;
Initialize $\mathbf{Y}^{(0)}$ and choose a value for the perplexity;
Compute $\mathbf{P}$ from the pre-processed data $\mathbf{X}$;
Set $C(\mathbf{d}^{(0)}) = \texttt{inf}$;
$t = 0$;
while $C(\mathbf{d}^{(t)}) - C(\mathbf{d}^{(t+1)}) > 10^{-5}$ **do**
    $t = t + 1$;
    Compute $\mathbf{D}$ and $\mathbf{Q}$ from $\mathbf{Y}^{(t)}$;
    Compute matrix $\mathbf{A}$ with elements $a_{ij} = \frac{d_{ij}+2p_{ij}}{4d_{ij}}$ if $i \neq j$;
    Fill the diagonal of $\mathbf{A}$ with $a_{ii} = \sum_j -a_{ij}$;
    Compute matrix $\mathbf{B}$ with elements $b_{ij} = \frac{d_{ij}+2q_{ij}}{4d_{ij}}$ if $i \neq j$;
    Fill the diagonal of $\mathbf{B}$ with $b_{ii} = \sum_j -b_{ij}$;
    Update $\mathbf{Y}$: $\mathbf{Y}^{(t+1)} = \mathbf{A}^{-}\mathbf{B}\mathbf{Y}^{(t)}$;
    Compute $C(\mathbf{d}^{(t+1)})$ through (5);
return $\mathbf{Y}$

6 Results

To evaluate our method, we apply it to data from the MNIST database. First we will shortly introduce this data and discuss some remarks on it. Then we will discuss the implementation of the MDS, Symmetric SNE and Majorized SNE techniques. After that, we will analyze what initialization suits our data the best. Then we will discuss how to select a stopping criterion for the Majorized SNE by analyzing the development of the KL-cost and the visualizations after several number of iterations. Finally we will compare our method with MDS and Symmetric SNE visualizations.

6.1 Data

The data used for the experiments in this Results section comes from the the Modified National Institute of Standards and Technology (MNIST) database. It contains 60,000 images of handwritten digits which were size normalized and centered in a fixed-size image. The images consist of 28 pixels by 28 which brings the dimensionality of the data set to 784. Figure 3 shows 100 samples from the MNIST database.

Figure 3: A sample of 100 handwritten digits from the MNIST data base. Each digit is captured in a 784 pixel image. (Figure adapted from Baldominos et al. (2019))

From this figure we notice that some digits are difficult to distinguish from each other. For example the tenth '6' looks very similar to the tenth '0' while the first one is meant to be a '6' and the other one a '0'. In addition, some handwritten digits, the ninth '2' and or the sixth '8' for example, are difficult to recognize. As stated by Hinton and Roweis (2002), when working with data from handwritten digits, SNE does not as cleanly separate these pairs. Therefore we will run our algorithm on the classes 0,1,3 and 4 only. We tried to run the algorithm on data sets with more classes, but this gave very bad visualizations in which points from many classes were put together almost randomly. All the experiments in this Results section are performed using the same set of 500 randomly selected handwritten digits from the previously mentioned classes. The perplexity was set to 15 following Hinton and Roweis (2002) who used this value for similar data.

6.2 Implementation

As mentioned at the beginning of this Results section, we will make use of MDS, Symmetric SNE and Majorized SNE results. In this section we will explain how these results were obtained. The MDS output was obtained using the *sklearn.manifold.MDS* package[2] in Python. Here we changed the stop criterion by replacing line 130 with `dis = n_samples*((X**2) .sum(axis=1)).sum()`. The package uses the SMACOF algorithm as explained in Borg and Groenen (2005). We initialized the MDS code with the Classical Scaling solution and used an epsilon of 10^{-6} at which convergence of the stress function is declared.

The output for Symmetric SNE was obtained by adjusting the t-SNE Python code that was published by Laurens van der Maaten on his website[3]. Specifically, we changed line 148, in which the numerator of the elements in matrix $\mathbf{Q}$ is being computed, by `num = np.exp(-1.* np.add(np.add(num, sum_Y).T, sum_Y))`. The second change was made in line 156 which computes the gradient of the KL-cost. It was replaced by `dY[i, :]  = np.dot(PQ[i,:], Y[i,:]-Y)`. We used the default settings of the t-SNE code regarding the optimization process and used a perplexity of 15. Finally, the Python code for the implementation of Majorized SNE can be found in Appendix A. In the following sections we will discuss the settings for Majorized SNE.

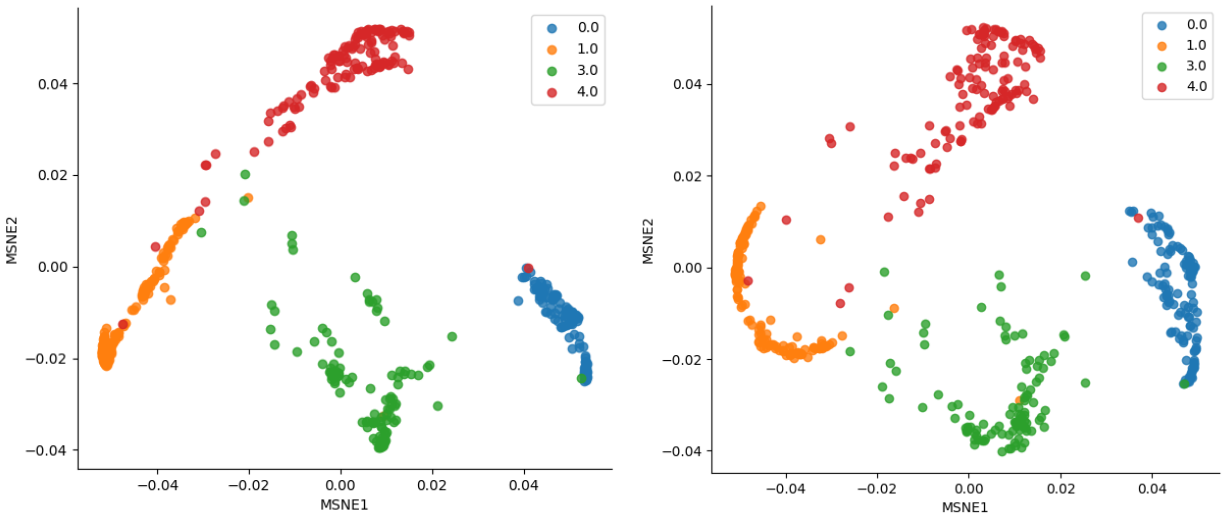
[2]https://github.com/scikit-learn/scikit-learn/blob/7e1e6d09b/sklearn/manifold/_mds.pyL310

[3]https://lvdmaaten.github.io/tsne/

6.3 Initialization

The Laplacian SNE cost function is a non convex function which contains several local optima. When applied on non convex functions, both gradient descent and MM usually end up in such a local optimum. This is not a major issue since for visualization techniques a local optimum that captures important aspects of the data is often preferable to a global optimum which over fits the data for example (Van der Maaten and Hinton, 2008). The initialization of the searching process has a significant impact on the final minimum. Therefore, in this section, we will compare two initializations. To do so we will run the Majorized SNE with the same settings but initialized differently. The resulting visualizations will be Procrustes rotated such that the similarity between them is maximal. This enables us to compare the results fairly. Procrustes rotation was introduced by Hurley and Cattell (1962) and generalized by Gower (1975). We will analyze the visualizations visually first. Then we will run k-means clustering on the resulted embeddings to see how this method performs on separating the classes in both cases. Finally we will select the initialization that suits our data the best.

While SNE algorithms are usually initialized by a random configuration, most MDS programs use a rational starting configuration such as the Classical Scaling (CS) solution of Torgerson (1958) and Gower (1966). However, this initialization is not necessarily the best one as stated by Borg and Groenen (2005). Therefore we compare the CS intialization to a method that we call dimension dropping (dd), which amounts to the following. First we ran Algorithm 1 on our data set which reduced the dimensionality to 5. Next we rotated the solution using principal axis rotation and dropped the dimension that accounted for the least variance. Then we repeated the same steps until we were left with a two dimensional embedding $\mathbf{Y}$. Following Hinton and Roweis (2002) we center the initialization very close to the origin , which means that we scale the initialization with 10^{-5}. Figure 4 shows the visualizations produced by Majorized SNE, initialized with the Classical Scaling solution in Figure 4a and dimension dropping in Figure 4b. The visualizations are rotated using Procrustes rotation.



(a) (b)

Figure 4: The Majorized SNE visualizations initialized with the Classical Scaling solution in Figure 4a and dimension dropping in Figure 4b. The visualizations are Procrustes rotated such that the similarity between them is maximized. For both experiments we used the same data sample with 500 handwritten digits and a perplexity of 15.

When initialized with Classical Scaling, Majorized SNE converged[4] after 8.3 minutes at 10,000 iterations. The KL-cost went from 3.372 to 3.229. Dimension dropping caused our algorithm to converge after almost 6 minutes at 7,000 iterations. Here the KL-cost went from 3.372 to 3.297, which is a smaller reduction than in the first case. While the classes are separated quite cleanly in Figure 4a, the visualization in Figure 4b looks a little bit messier. Especially the handwritten digits '1' and '3' are separated less cleanly.

In order to compare the resulted visualizations properly, we perform k-means clustering on both low dimensional embeddings. To do so we used the *sklearn.cluster.KMeans* Python package, which performs the algorithm 10 times with different random initializations. Logically we set the number of clusters $k = 4$. To assess how good the points were clustered we compare the confusion matrices and the weighted F1 scores for both cases. Table 2 shows the confusion matrices for both initializations.

[4]What we mean by convergence will be explained in Section 6.4.

		True classes							
		CS initialization				dd initialization			
		'0'	'1'	'3'	'4'	'0'	'1'	'3'	'4'
Assigned classes	'0'	117	0	1	1	117	0	2	1
	'1'	0	142	3	6	0	142	3	5
	'3'	0	1	106	0	0	1	106	0
	'4'	0	0	1	122	0	0	0	123

Table 2: The confusion matrices of the k-means allocation on the low dimensional embeddings obtained from the Majorized SNE algorithm initialized with the Classical Scaling solution and dimension dropping.

We observe that both visualizations enable k-means clustering to perform very well. Out of 500 handwritten digits, the majorized SNE with Classical Scaling initialization allocated 13 digits wrongly, whereas the approach of dimension dropping generated 12 inaccurate designations. The weighted F1 scores are 0.974 for the first setting and 0.976 for the second one. Given the similar performance of both settings, we conclude that for this data set it does not matter which of the two initializations is used. For the remainder of the experiments we will use the CS initialization.

Next to the CS and dd initializations, we also tried to initialize the Majorized SNE algorithm with the MDS and Symmetric SNE solutions that will be discussed in Section 6.5. However, our algorithm did not manage to improve these solutions. Finally we want to mention that we also tried to scale $\mathbf{Y}$ in each iteration. That is, we performed a line search using the so called golden section method to select the the correct scale. Each iteration we minimized the cost function given the new scaled embedding $a\mathbf{Y}^{(t)}$ over a. This resulted in $a = 10^{-5}$ for the initialization and $a = 1$ for the rest of the iteration. Therefore we only scale the initialization.

6.4 Selecting the number of iterations

There are several stop criteria for an optimization process. The most common one is convergence. As instructed in step 6 of Algorithm 1, the process is then stopped if the difference between the cost in the current and previous iteration is smaller than a small value (for example 10^{-5}). Figure 5 shows that even after 250,000 iterations of the Majorized SNE algorithm, there is no sign of convergence of the KL-cost function. However, if we look at the development of the Majorized SNE visualizations in Figure 6 on page 21, we see that the process converges to a visualization

with cleanly separated clusters after 10,000 iterations already. We think that the KL-cost keeps decreasing, despite the convergence of the visualization, because the distances within clusters become smaller and smaller. This produces higher values for q_{ij}, which causes the cost function in equation 5 to decrease.

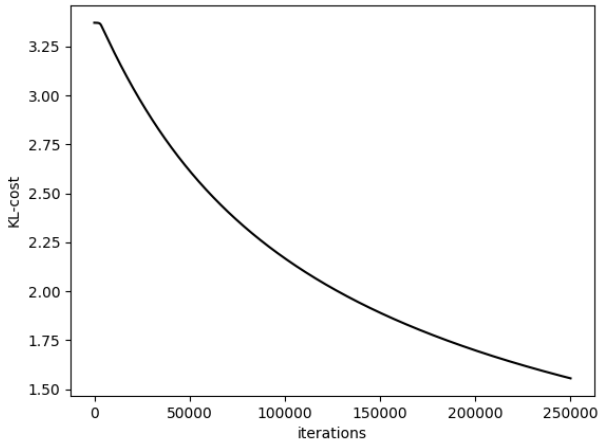
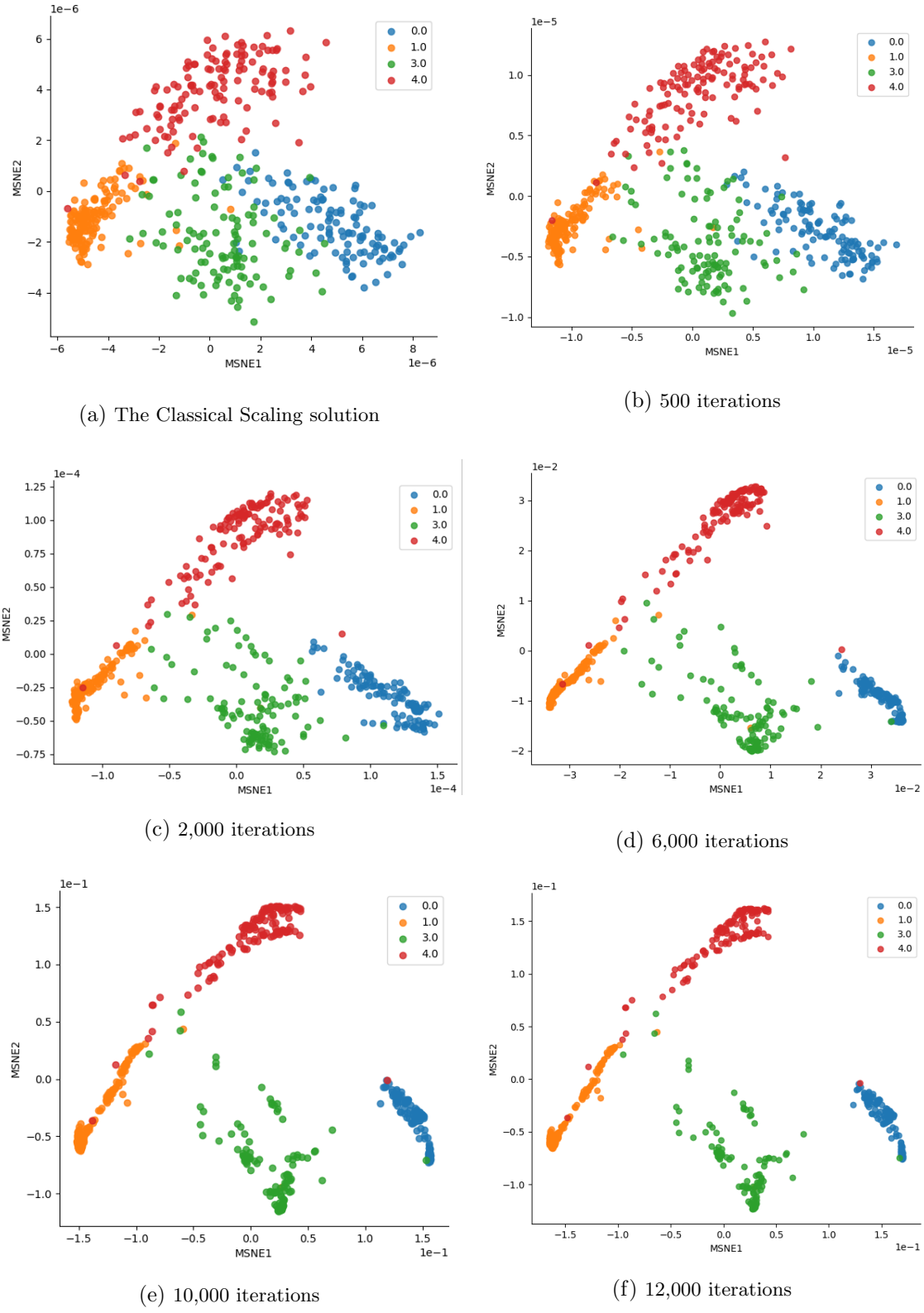


Figure 5: The course of the KL-cost function over 250,000 iterations of the Majorized SNE method with a perplexity of 15 and initialized with the Classical Scaling solution. The KL-cost starts at 3.372 and is reduced to 1.556 after 250,000 iterations.

This being the case, we stopped Algorihm 1 based on the visualizations in Figure 6. In general, we do not want to stop the process of too early and end up with a visualization that can not be interpreted because similar points did not get the chance to find each other yet (Wattenberg et al., 2016). On the other hand, we do not want to run the algorithm too long, because this may cause similar points to overfit and form several sub clusters. As stated earlier, the process starts with the CS in solution (Figure 6a). After 500 iterations (Figure 6b), the distances within the classes got smaller. After 2,000 and especially after 6,000 iterations (Figures 6c and 6d) the classes are already separated from each other. However, after 10,000 iterations the classes '1', '3' and '4' are separated more cleanly. Since the difference between the visualization after 12,000 iterations (Figure 6f) does not significantly differ from Figure 6e, we select 10,000 iterations to fit the data the best.



(a) The Classical Scaling solution

(b) 500 iterations

(c) 2,000 iterations

(d) 6,000 iterations

(e) 10,000 iterations

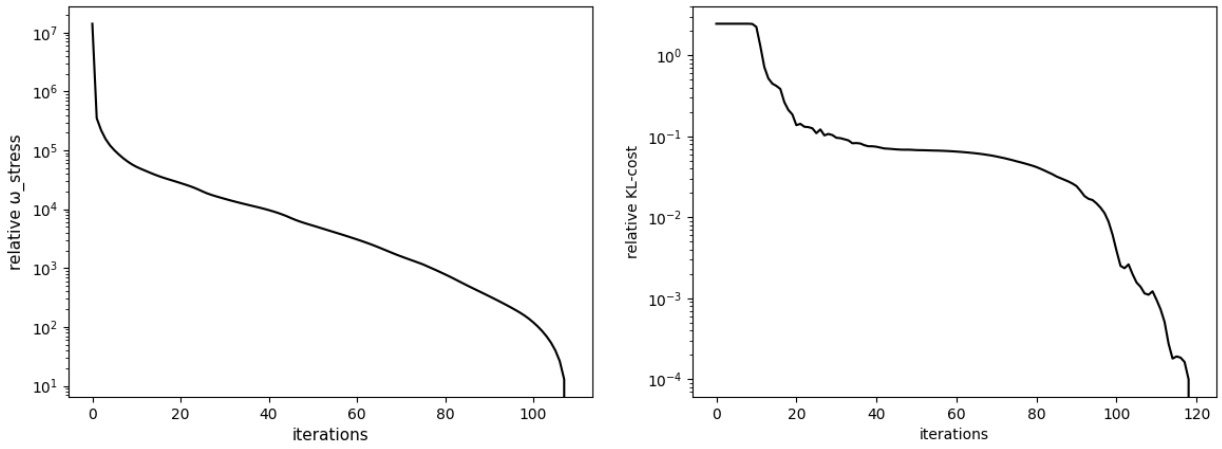
(f) 12,000 iterations

Figure 6: The realization of the Majorized SNE visualization starting from the Classical Scaling Solution with a perplexity of 15. The optimization converges after 10,000 iterations.

6.5 Comparing Majorized SNE with MDS and Symmetric SNE

To properly assess our method, we run the MDS and Symmetric SNE algorithms with the same data and initialization in this section. First we will discuss the parameter settings and explain how the results were obtained. After Procrustes rotating the MDS and Symmetric SNE outputs with respect to the Majorized SNE output, we will compare these results visually first. Finally, we will run k-means clustering on the embeddings of each technique to see how this method performs on separating classes in each embedding.

To capture the progress of the MDS stress and KL-cost correctly, Figure 7 plots them on a logarithmic scale corrected by the value at which the cost functions converge (i.e. we subtract the final stress or cost value from the stress or cost value in each iteration). From these plots we conclude that the MDS and Symmetric SNE cost functions decrease very fast and converge after 109 and 117 iterations respectively, both at a convergence rate of 10^{-6}. MDS needed only 3 seconds to get the job done. The Symmetric SNE algorithm needed 8.0 seconds to bring the KL-cost from 3.372 to 0.9121. This is incredibly fast in comparison to our method which takes over 8 minutes to finish.



(a) (b)

Figure 7: The progress of the MDS stress function in figure 7a and the KL-cost function optimized by the Symmetric SNE technique in figure 7b corrected by the value at which they converge. The vertical axis uses a logarithmic scale and the horizontal axis uses a linear scale for the number of iterations. Both techniques were initialized with the Classical Scaling solution and performed on the same data sample of 500 handwritten digits.

The left column in Figure 8 on page 24 shows the resulting visualizations for MDS, Symmetric SNE and Majorized SNE. The visualizations are Procrustes rotated with respect to the Majorized SNE result. Starting with the MDS visualization in Figure 8a we see that the classes are not separated cleanly. Especially the digits from class '3' seem to mess up the MDS solution a little bit. Although the majority of this class has been put together, several green dots are misplaced. The Symmetric SNE solution, showed in Figure 8c performs better in terms of placing digits from the same class together. However, we will see that without any prior information about the class each digit belongs to, the classes can not be separated correctly. Finally, our Majorized version of Symmetric SNE separates the classes very cleanly. Especially the classes '0', '1' and '4' form clear clusters. The digits '3' are a bit more spread out but only three or four of them are misplaced. We note that all techniques present the classes '0' and '1' far from each other. This means that the techniques recognize the big dissimilarity between these digits, which is true since they are not to be confused in any way. This shows that our method performs equally to MDS and Symmetric SNE in terms of global geometry.

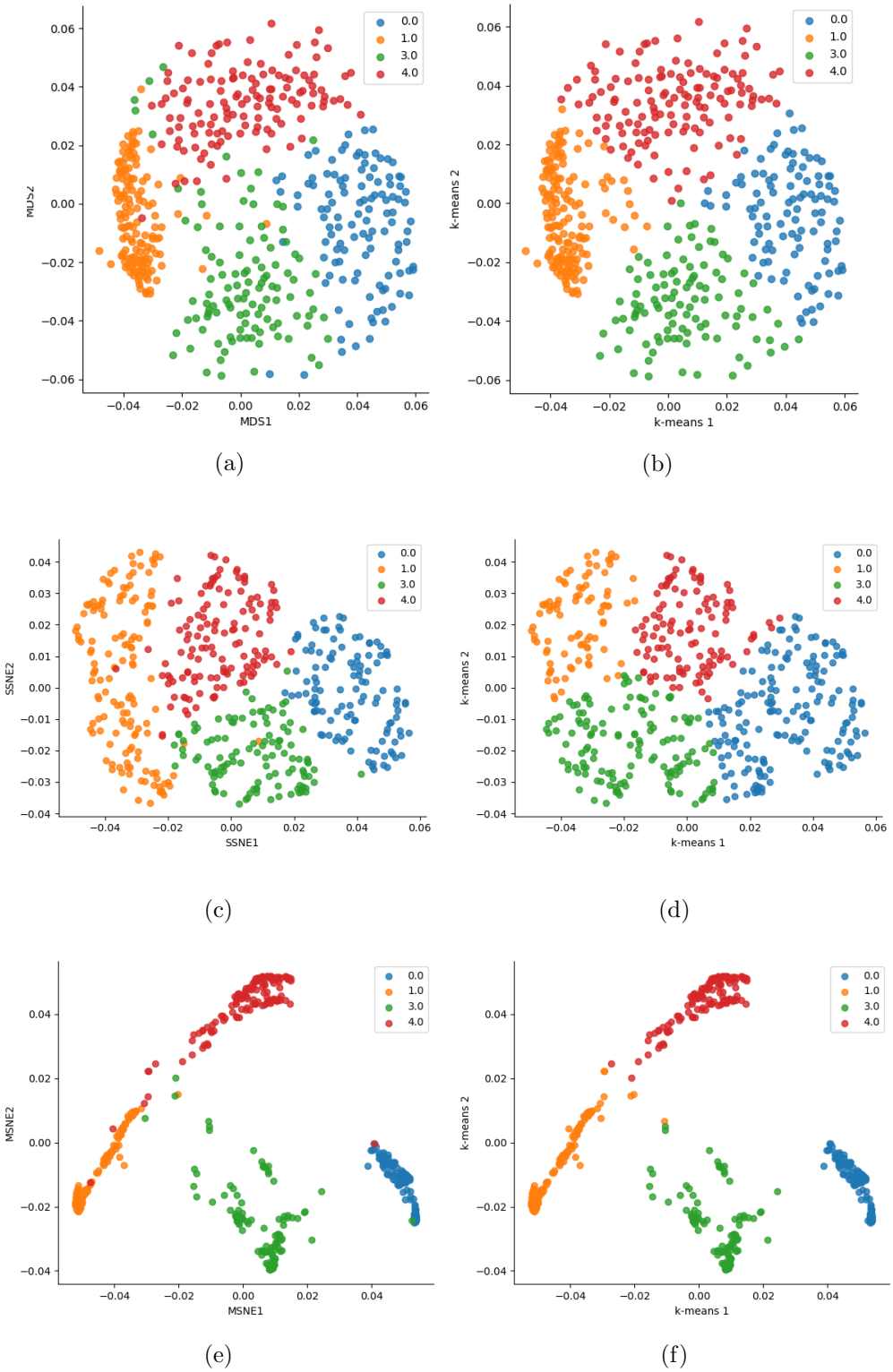


Figure 8: The resulting visualizations of the MDS (Figure 8a), Symmetric SNE (Figure 8c) and the Majorized SNE (Figure 8e) methods on the same data sample of 500 handwritten digits. On the right side of each of these figures the results of the k-means algorithm on the resulting low dimensional embedding are shown.

Next we will compare the performance of k-means clustering on the low dimensional embedding of each of the three methods. The output of MDS enables k-means to form acceptable clusters (Figure 8b). The class '0', in particular, is sufficiently separated for the k-means algorithm to detect it as a cluster. On the other hand, we see that many handwritten digits '3' are not classified as such. The first part of Table 3 confirms this. In comparison to other digits, many 3's have been assigned to the wrong class. This backs up what we said about this class in the preceding paragraph. The F1 score for k-means on the MDS embedding is 0.911. In the previous paragraph we also mentioned that the Symmetric SNE visualization appears to be good at first glance, but that the classes can not be separated correctly when an algorithm without prior information about the true labels is used. Figure 8d and the second part of Table 3 show that many digits are indeed labeled wrongly. For example, a big number of handwritten digits '1' are labeled as '3' and less than half of digits '3' are classified as such. Class '0' looks good and has the highest amount of true positives, but many 3's are wrongly assigned to this class. An F1 score of 0.701 confirms this bad performance. This is much lower than the F1 score for the MDS case. The third part of Table 3, which corresponds to the performance of k-means on the result of Majorized SNE, shows the highest number of true positives. Only the classes '3' and '4' have a few misplaced handwritten digits. This confirms our findings in the previous paragraph on the Majorized SNE output. In Section 6.3 we saw that F1 score for k-means on the Majorized SNE output is 0.974. This suggests that our algorithm outperforms both MDS and Symmetric SNE. However, MDS and Symmetric SNE can handle more complicated data sets while Symmetric SNE already fails when the class '2' would be added to the data. On top of that, Symmetric SNE requires much more iterations and longer computation time.

		True classes											
		MDS				Symmetric SNE				Majorized SNE			
		'0'	'1'	'3'	'4'	'0'	'1'	'3'	'4'	'0'	'1'	'3'	'4'
Assigned classes	'0'	108	0	4	1	111	1	56	0	117	0	1	1
	'1'	0	140	8	6	0	81	0	7	0	142	3	6
	'3'	8	2	86	0	0	61	53	19	0	1	106	0
	'4'	1	1	13	122	6	0	2	103	0	0	1	122

Table 3: The confusion matrices of the k-means allocation on the low dimensional embeddings obtained from the Majorized SNE algorithm initialized with the Classical Scaling solution and dimension dropping.

7 Conclusion and discussion

In this thesis we presented an alternative way of optimizing the Kullback-Leibler cost function. After discussing literature on dimensionality reduction and pointing out the main weakness of SNE, we formulated the research question of this thesis as *Can we develop a procedure which improves the optimizing process of SNE cost functions with majorization, such that different runs on the same data yield the same solution?* Based on Section 5, we might answer this research question positively for Laplacian SNE. There, we derived an updating equation for the low dimensional embedding $\mathbf{Y}$ which optimizes the KL-cost. By definition, this approach causes each run on the same data to yield the same solution. In Section 6 we have seen that, for a simplified data sample, the classes in the visualization resulted by our method are separated cleanly. However, it takes more time and requires more iterations than the MDS and Symmetric SNE algorithms. Moreover, we have seen that the Majorized SNE solution lies at a higher KL-cost value than the Symmetric SNE solution (3.229 and 0.9119 respectively). Finally, when applied to more complicated data sets, Majorized SNE does not manage to give correct visualizations of the data while Symmetric SNE, for example, does. Therefore we do not recommend using this version of Majorized SNE directly on complex data sets.

However, we think that the main idea behind Majorized SNE can be useful for further research on removing the stochastic characteristic of SNE techniques. Specifically, majorizing the t-SNE cost functions might give better results. In t-SNE a Student-t distribution, rather than a Laplacian distribution, is used to compute the similarities q_{ij} between two points in the low-dimensional space. This causes the density of a point to be evaluated much faster because the Student-t distribution does not involve an exponential. This might also simplify the process of majorizing the t-SNE cost function. However, the required conditions for Theorem 1 of Groenen and Josse (2016) used in Section 5 to majorize the Laplacian SNE function do not hold for the t-SNE cost function. As a result, we have restricted ourselves to the majorization of Laplacian SNE in this thesis.

In this thesis we did also not discuss the importance of perplexity in SNE techniques. For our experiments we set the perplexity to 15 following Hinton and Roweis (2002) who used this value for a similar data sample. When running Majorized SNE on a different data set, we recommend studying Wattenberg et al. (2016) and the FAQ on the website of Laurens van der Maaten on

t-SNE (https://lvdmaaten.github.io/tsne/) before setting the perplexity parameter. It might also be interesting to experiment with majorization of an SNE technique while omitting the perplexity or replacing it with k-nearest neighbours for example. For the data we used, this provided no additional value.

Finally, we would like to note that we believe SNE techniques are more cluster algorithms than dimension reduction techniques. There are similarities with spectral clustering where no strong assumptions are made on the shapes of clusters. Moreover, the whole optimization method of t-SNE for example focuses on creating clusters. As explained in Section 3.4 of Van der Maaten and Hinton (2008), "*the optimization is encouraged to focus on modeling the large* p_{ij}*'s by fairly large* q_{ij}*'s*". This causes the formation of widely separated clusters in the map.

References

Alexander, C. (2008). *Market risk analysis, practical financial econometrics*, volume 2. John Wiley & Sons.

Arora, S., Hu, W., and Kothari, P. K. (2018). An analysis of the t-sne algorithm for data visualization. In *Conference on Learning Theory*, pages 1455–1462. PMLR.

Baldominos, A., Saez, Y., and Isasi, P. (2019). A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, 9(15):3169.

Borg, I. and Groenen, P. J. (2005). *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media.

Cook, J., Sutskever, I., Mnih, A., and Hinton, G. (2007). Visualizing similarity data with a mixture of maps. In *Artificial Intelligence and Statistics*, pages 67–74. PMLR.

De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. *Recent developments in statistics*, pages 133–145.

De Leeuw, J. (1988). Convergence of the majorization method for multidimensional scaling. *Journal of classification*, 5(2):163–180.

De Leeuw, J. and Heiser, W. J. (1977). Convergence of correction matrix algorithms for multidimensional scaling. *Geometric representations of relational data*, 36:735–752.

Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338.

Gower, J. C. (1975). Generalized procrustes analysis. *Psychometrika*, 40(1):33–51.

Groenen, P. J. and Josse, J. (2016). Multinomial multiple correspondence analysis. *arXiv preprint arXiv:1603.03174*.

Groenen, P. J. and van de Velden, M. (2016). Multidimensional scaling by majorization: A review. *Journal of Statistical Software*, 73(1):1–26.

Hinton, G. and Roweis, S. T. (2002). Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.

Hurley, J. R. and Cattell, R. B. (1962). The procrustes program: Producing direct rotation to test a hypothesized factor structure. *Behavioral science*, 7(2):258.

Kobak, D. and Berens, P. (2019). The art of using t-sne for single-cell transcriptomics. *Nature communications*, 10(1):1–14.

Kruskal, J. B. (1964a). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27.

Kruskal, J. B. (1964b). Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129.

Lange, K., Hunter, D. R., and Yang, I. (2000). Optimization transfer using surrogate objective functions. *Journal of computational and graphical statistics*, 9(1):1–20.

Ortega, J. and Rheinboldt, W. (1970). Iterative solution of nonlinear equations in several variables. new york us: Academic press. *Iterative Solution of Nonlinear Equations in Several Variables. New York, US: Academic Press.*

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. london, edinburgh. dublin philos mag j sci 2: 559–572.

Poličar, P. G., Stražar, M., and Zupan, B. (2019). opentsne: a modular python library for t-sne dimensionality reduction and embedding. *BioRxiv*, page 731877.

Torgerson, W. S. (1958). *Theory and methods of scaling.* Wiley.

Van Der Maaten, L. (2014). Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Van Der Maaten, L., Postma, E., and Van den Herik, J. (2009). Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13.

Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-sne effectively. *Distill.*

Young, G. and Householder, A. S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3(1):19–22.

A Appendix

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import euclidean_distances
import time
from sklearn.manifold import MDS
from scipy.linalg import orthogonal_procrustes

def procrustes(Y, Y_2):

    """
    This method comes from the scipy.spatial.procrustes package:
    https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/
    scipy.spatial.procrustes.html
    """

    mtx1 = np.array(Y, dtype=np.double, copy=True)
    mtx2 = np.array(Y_2, dtype=np.double, copy=True)

    # translate all the data to the origin
    mtx1 -= np.mean(mtx1, 0)
    mtx2 -= np.mean(mtx2, 0)

    norm1 = np.linalg.norm(mtx1)
    norm2 = np.linalg.norm(mtx2)

    # change scaling of data (in rows) such that trace(mtx*mtx') = 1
    mtx1 /= norm1
    mtx2 /= norm2

    # transform mtx2 to minimize disparity
    R, s = orthogonal_procrustes(mtx1, mtx2)
    mtx2 = np.dot(mtx2, R.T) * s

    # measure the dissimilarity between the two datasets
    disparity = np.sum(np.square(mtx1 - mtx2))

    return mtx1, mtx2, disparity

def Hbeta(D=np.array([]), beta=1.0):
    """
        This method comes from the t-SNE package:
        Compute the perplexity and the P-row for a specific value of the
        precision of a Gaussian distribution.
    """

    # Compute P-row and corresponding perplexity
    P = np.exp(-D.copy() * beta)
    sumP = sum(P)
    H = np.log(sumP) + beta * np.sum(D * P) / sumP
    P = P / sumP
    return H, P

def x2p(X=np.array([]), tol=1e-5, perplexity=30.0):
    """
```

```
        This method comes from the t-SNE package:
        Performs a binary search to get P-values in such a way that each
        conditional Gaussian has the same perplexity.
    """

    ####### Initialize some variables
    print("Computing pairwise distances...")
    (n, d) = X.shape
    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)
    P = np.zeros((n, n))
    beta = np.ones((n, 1))
    logU = np.log(perplexity)

    ####### Loop over all datapoints
    for i in range(n):

        ####### Print progress
        if i % 500 == 0:
            print("Computing P-values for point %d of %d..." % (i, n))

        ####### Compute the Gaussian kernel and entropy for the current precision
        betamin = -np.inf
        betamax = np.inf
        Di = D[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))]
        (H, thisP) = Hbeta(Di, beta[i])

        ####### Evaluate whether the perplexity is within tolerance
        Hdiff = H - logU
        tries = 0
        while np.abs(Hdiff) > tol and tries < 50:

            ####### If not, increase or decrease precision
            if Hdiff > 0:
                betamin = beta[i].copy()
                if betamax == np.inf or betamax == -np.inf:
                    beta[i] = beta[i] * 2.
                else:
                    beta[i] = (beta[i] + betamax) / 2.
            else:
                betamax = beta[i].copy()
                if betamin == np.inf or betamin == -np.inf:
                    beta[i] = beta[i] / 2.
                else:
                    beta[i] = (beta[i] + betamin) / 2.

            ####### Recompute the values
            (H, thisP) = Hbeta(Di, beta[i])
            Hdiff = H - logU
            tries += 1

        ####### Set the final row of P
        P[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))] = thisP

    ####### Return final P-matrix
    print("Mean value of sigma: %f" % np.mean(np.sqrt(1 / beta)))
    return P


```

```
def MNE():

    Y = a*X[:, :low_dim]

    # data = pd.DataFrame(np.column_stack((Y, labels[:subset])),
    # columns=['MSNE1', 'MSNE2', 'label'])
    # sns.lmplot(data=data, x='MSNE1', y='MSNE2', hue='label',
    # fit_reg=False, legend=False, legend_out=True)
    # plt.ticklabel_format(axis="both", style="sci", scilimits=(0,0))
    # plt.gca().set_aspect('equal', adjustable='box')
    # plt.legend()
    # plt.show()


    ####### Compute matrix P
    upperInds = np.triu_indices(n, 1)
    P = x2p(X, 1e-15, perplexity)
    P = P + np.transpose(P)
    P = P / sum(P[upperInds])
    P = np.maximum(P, 1e-12)

    C_old = np.inf
    iter = 0
    while iter < iterations:

        ####### Compute high dimensional Euclidean distances
        D = euclidean_distances(Y, Y)

        ####### Compute matrix Q
        Q = np.exp(-D**2)
        Q = Q / sum(Q[upperInds])
        Q = np.maximum(Q, 1e-12)

        ####### Prepare A and B for the updating equation
        D = np.maximum(D, 2e-2)
        A = (D + 2*P) / (4*D)
        B = (D + 2*Q) / (4*D)
        np.fill_diagonal(A, 0)
        np.fill_diagonal(A, -sum(A))
        np.fill_diagonal(B, 0)
        np.fill_diagonal(B, -sum(B))
        np.fill_diagonal(D, 0)

        Y_new = np.linalg.solve(A+1, B@Y)
        Y = 2*Y_new - Y

        C = np.sum(P * np.log(P / Q))/2

        delta = C_old - C
        C_old = C

        if delta < 0:
            print(f"###### WARNING ######: Iteration {iter + 1} --> delta = {delta
    }")
        if (iter + 1) % 500 == 0:
            print(f"Iteration {iter +1}: KL-Cost = {C}")
        #     vC.append(C)
        # if (iter + 1) in [500, 2000, 6000, 10000, 12000]:
        #     print(iter + 1)
```

```
        #     data = pd.DataFrame(np.column_stack((Y, labels[:subset])),
        #     columns=['MSNE1', 'MSNE2', 'label'])
        #     sns.lmplot(data=data, x='MSNE1', y='MSNE2', hue='label',
        #     fit_reg=False, legend=False, legend_out=True)
        #     plt.ticklabel_format(axis="both", style="sci", scilimits=(0,0))
        #     plt.gca().set_aspect('equal', adjustable='box')
        #     plt.legend()
        #     plt.show()

        iter += 1

    return (Y, C)


def main():

    (Y, C) = MNE()

    print(f"Computation time of your algorithm: {time.time() - start_time} ")

    data = pd.DataFrame(np.column_stack((Y, labels[:subset])),
    columns=['MSNE1', 'MSNE2', 'label'])
    sns.lmplot(data=data, x='MSNE1', y='MSNE2', hue='label',
    fit_reg=False, legend=False, legend_out=True)
    plt.ticklabel_format(axis="y", style="sci", scilimits=(0,0))
    plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
    plt.gca().set_aspect('equal', adjustable='box')
    plt.legend()
    plt.show()


def pca(X=np.array([]), no_dims=50):
    """
        This method comes from the t-SNE package:
        Runs PCA on the NxD array X in order to reduce its dimensionality to
        no_dims dimensions.
    """

    print("Preprocessing the data using PCA...")
    (n, d) = X.shape
    X = X - np.tile(np.mean(X, 0), (n, 1))
    (l, M) = np.linalg.eig(np.dot(X.T, X))
    Y = np.dot(X, M[:, 0:no_dims])
    print("PCA finished")
    return Y

def readData():

    X = np.loadtxt("mnist2500_X.txt")
    labels = np.loadtxt("mnist2500_labels.txt")

    indexes = np.where((labels < 2.0) | (labels == 3.0) | (labels == 4.0) )
    labels = labels[np.isin(labels, [0.0, 1.0, 3.0, 4.0])]
    X = X[indexes]

    return (X[:subset], labels[:subset])


if __name__ == "__main__":
```

```
    import os
    os.chdir("/Users/safouane/Downloads/tsne_python")
    start_time = time.time()

    iterations = 10000
    initial_dims = 50
    perplexity = 15.0
    subset = 500
    a = .000001
    vC = []

    (data, labels) = readData()

    ####### Prepare data for large data sets
    X = pca(data, initial_dims).real

    ####### Set variables and initial coordinates Y
    (n, dims) = X.shape
    low_dim = 2

    main()
```