uc3m | Universidad Carlos III de Madrid

BACHELOR'S DEGREE IN TELECOMMUNICATION TECHNOLOGIES ENGINEERING (2018-2019)

*Bachelor thesis*

# "AN IN-DEPTH REVIEW OF T-SNE WITH APPLICATIONS"

Alfonso Albacete Zapata

Tutor: Eduardo García Portugués

Leganés, julio 2019

# ABSTRACT

In this project we have reviewed a dimension-reduction technique named *t*-Stochastic Neighbour Embedding (*t*-SNE). This technique has produced a huge impact in the machine learning community due to its capabilities and its flexibility to reduce the dimensionality of a dataset. It has become a direct competitor of classical, linear, dimension-reduction techniques such as principal component analysis and multidimensional scaling. Following the original paper of Van der Maaten and Hinton (2008) in *Journal of Machine Learning Research* we have implemented a pedagogic version of *t*-SNE algorithm that allowed us to explain the technique in detail. Our version gives outcomes that are comparable to the state-of-the-art implementation. We have studied *t*-SNE in synthetic data with different patterns and have applied *t*-SNE to a variety of real datasets. In this process, we have analysed the main benefits and drawbacks of *t*-SNE.

All the code developed in this project is openly available in the following GitHub repository:

https://github.com/100346868/TSNE

**Keywords:** ***t*-SNE, PCA, MDS, dimension-reduction, algorithm.**

## ACKNOWLEDGMENT

To my family and the people who have supported me along these years.

# CONTENTS

# 1. INTRODUCTION

## 1.1. Dimension-reduction techniques

Nowadays everything is related to data, it have become really important in our lives, they contain a lot of information about us. Even all the companies are storing lots of information about processes they carry out and the information they gather about their clients among others. Therefore, this is producing that the quantity of data is growing exponentially and we are using computers to store these huge quantities of information. Since we have this information gathered, there appear the idea of analyse it in order to obtain patterns and look for solutions to the problems may be originated. Computers are prepared to process and interpret data that depends on several variables but we can not. Then, to use this data for our purpose, we need to visualize it and understand it. Sometimes we need to obtain conclusions from a huge data set that depends on lots of variables that we will call dimensions. When this situation occurs, we can not visualize our data set since a sample that depends on several variables needs several dimensions to represent the points, in order to visualize it there appears new dimensional objects we can not think of. To solve this problem, dimension-reduction techniques, such as *t*-SNE, have been developed. This algorithm allows to reduce dimensions, which means that we are going to simplify our data into two or three dimensions, so we can perfectly visualize the same original data as the expense of some information, but preserving enough important information being faithful to the original data.

Eyesight is very important for human, since it is going to give us the very first impression of anything. Supposing we have data, we are trying to simplify it so we can use it to understand how is it behaving or to obtain some relations among the different samples. To take some advantage of data, we will need to represent it on a way we can visualize it. Firstly we may wonder, what is the shape of this dataset? Supposing we have one dimension we can represent it in a straight line, if we have two dimensions, we are able to visualize them using a plane, even if we have three components we can also represent it in a cube. However, as the number of variables increases it is more difficult to picture it mentally. Although we have some tricks to represent them, for example the pairwise plot, there is some loss of information. We are looking for an algorithm that really help to the human to reach this goal. In this project we are going to deepen in *t*-SNE, which will help us to reduce dimensions so we can obtain an image of the data we can visualize, it will have some loss of information.

Moreover, these techniques will help us to reduce our data and this implies that our computational cost will be lower without losing relevant information. This fact is very important when we manage huge data that takes too long to process it, because we are able to use less resources to obtain nearly the same results. Sometimes we may think that the more information we have the better, nevertheless, when we have more data we have to process it, then we are interested in reducing this computational cost because the time to process the data can take very long. By reducing the dimensions we will reduce considerably this time, besides we will understand better the data we are processing and will obtain better conclusions when we picture it. Even we may use this output as an input of an algorithm that help us to classify our data.

As in real life, when we find a complex situation, we always try to simplify the problem so we can solve it in a feasible way. Once we have simplified it, we can go backwards to give a solution to each individual problem and finally solve the big one. The same situation can be found when we manage big data. Thanks to this algorithm, simplifying a problem will be possible. Using this clustering technique will allow us to visualize the multidimensional data in fewer dimensions. Clustering means to divide the individual data in groups that are more similar to each other than to those in other groups or clusters. Otherwise, if we do not apply this dimension-reduction techniques it would not be possible to have an idea of how the data is.

Explaining what is reduction of dimensions in a naive/nontechnical way, people use the well-known example of the television. When we watch a program on the screen, we are seeing it in two dimensions, however, it was recorded in real life, what means in 3D. Therefore, we are visualizing the event without losing relevant information. Although we have lost a dimension we preserve the original. So a dimension has been reduced, obviously, this algorithm will allow to visualize bigger and more complex data.

There are several techniques to reduce dimensions nowadays, the most noteworthy of these are: MultiDimensional Scaling (MDS) and principal component analysis (PCA). Depending on the situation it will be more suitable to use one or the other. A brief analysis of each technique is given next for these two.

### 1.1.1. Multidimensional scaling

MDS seeks to express a distance matrix $n \times n$ as the Euclidean distance matrix associated to a $p$-dimension data.

Multidimensional scaling is a multivariate analysis technique that starting from a matrix of similarities between observations in a dataset produces a representation of the samples in an Euclidean space so the distances are the more approximated to the distances of the original dataset.

So thanks to the eigenvalues and eigenvectors obtained from that matrix, MDS construct some variables, typically two or three in order to plot them. So these new distances represent faithfully the the distances of the original dataset. These new variables are going to be called principal coordinates.

MultiDimensional Scaling (MDS) can be considered as a generalization of PCA, to dive in this concept we are basing on Peña (2002). MDS is a dimension-reduction technique based on the *similarity*, this concept will be used indistinctly as the distance between the points of the dataset. MDS is going to obtain the similarity matrix **D** which is squared $n \times n$ , being $n$ the number of elements, by calculating how similar are the variables among them so we will be able to represent them in a reduced dimension.

MDS is going to represent this similarity matrix thanks to orthogonal variables $\mathbf{x}_1, ..., \mathbf{x}_p$ where $p < n$, then, we have to ensure that Euclidean distances among the coordinates of the elements with respect to these variables is equal or at least approximately equal to the distances among the original matrix. Summarising, from a matrix **D**, we are obtaining a matrix **X** of dimensions $n \times p$ that can be interpreted as the matrix where the Euclidean distances of the original matrix are reproduced faithfully.

This technique is going to describe and interpret the data as the other dimension-reduction techniques we are seeing. In order to construct the similarity matrix we have to produce the variables in function of the distances, thanks to the equation

$$\tilde{\mathbf{X}} = (\mathbf{I} - \frac{1}{n}\mathbf{11}')\mathbf{X} = \mathbf{PX} \quad \mathbf{P} = (\mathbf{I} - \frac{1}{n}\mathbf{11}'), \tag{1.1}$$

where we can see that the **I** is the identity matrix and **1** represents a vector of ones and the apostrophe denotes transposition. This is subtracting the mean of each value giving to us variables with zero mean. Thanks to this matrix we are able to compute other matrices such as the covariance matrix, $\mathbf{S} = \tilde{\mathbf{X}}'\tilde{\mathbf{X}}/n$ or the cross product matrix that we are interpreting as the similarity matrix among the $n$ elements, $\mathbf{Q} = \tilde{\mathbf{X}}'\tilde{\mathbf{X}}$. The terms of this matrix are composed by the scalar product of each pair of elements:

$$q_{ij} = \sum_{s=1}^{p} x_{is}x_{js} = \mathbf{x}_i'\mathbf{x}_j, \tag{1.2}$$

where $\mathbf{x}_i'$ is the $i$-th row of the matrix $\tilde{\mathbf{X}}$, finally we can interpret $\tilde{\mathbf{X}}'\tilde{\mathbf{X}}$ as the similarity matrix among the elements. The distances among the observations can be obtained from this matrix of similarity. Square Euclidean distance can be computed using

$$d_{ij}^2 = \sum_{s=1}^{p}(x_{is} - x_{js})^2 = ||\mathbf{x_i} - \mathbf{x}_j||^2, \tag{1.3}$$

that can be calculated in terms of the **Q** by the expression

$$d_{ij}^2 = q_{ii} + q_{jj} - 2q_{ij}. \tag{1.4}$$

In conclusion, given a matrix $\tilde{\mathbf{X}}$ we are able to construct the matrix of similarities $\mathbf{Q} = \tilde{\mathbf{X}}'\tilde{\mathbf{X}}$ and finally, thank to this one we are computing the matrix of **D** distances that will given by

$$\mathbf{D} = \mathrm{Diag}(\mathbf{Q})\mathbf{1}' + \mathbf{1}\mathrm{Diag}(\mathbf{Q})' - 2\mathbf{Q}, \tag{1.5}$$

where Diag is going to extract the associated diagonal of the matrix.

Once we have computed this **D** matrix of $n \times n$ elements we have to go backward in the process by constructing the **Q** matrix.

Since distances between two points $d_{ij}^2$ will not vary if we express the variables in deviations to the mean, we can assume that our variables will have zero mean.

We have that:

$$d_{ij}^2 = \sum_{s=1}^{p}(x_{is} - x_{js}) = \sum_{s=1}^{p}[(x_{is} - \tilde{x}_s) - (x_{js} - \tilde{x}_s)]^2. \tag{1.6}$$

To solve this indetermination we are looking for a matrix $\tilde{\mathbf{X}}$ composed by variables with zero mean. By adding the rows in the Equation (1.4):

$$\sum_{i=1}^{n} d_{ij}^2 = \sum_{i=1}^{n} q_{ii} + nq_{jj} = t + nq_{jj}, \tag{1.7}$$

where $t = \sum_{i=1}^{n} q_{ii}$. If we sum the columns we obtain:

$$\sum_{j=1}^{n} d_{ij}^2 = t + nq_{ii}, \tag{1.8}$$

and if we add the rows again we obtain:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 = 2nt, \tag{1.9}$$

and substituting in the previous equations and solving for $q_{ij}$ as we can see in Peña (2002), we finally obtain that:

$$q_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i}^2 - d_{j}^2 + d_{...}^2). \tag{1.10}$$

Once we obtain this matrix, we must calculate the eigenvalues and eigenvectors and the matrix $\mathbf{X}$ given $\mathbf{Q}$. Assuming that our distances matrix is defined positive it can be represented by:

$$\mathbf{Q} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}', \tag{1.11}$$

where $\mathbf{V}$ is $n \times p$ dimensions and it contains the eigenvectors corresponding to the non null eigenvalues of $\mathbf{Q}$, we can see that $\mathbf{\Lambda}$ is of $p \times p$ dimensions and it contains the eigenvalues and obviously $\mathbf{V}$ is the transpose with dimensions $p \times n$. This can be expressed as

$$\mathbf{Q} = \sum_{i=1}^{p} \lambda_i \mathbf{v}_i \mathbf{v}_i', \tag{1.12}$$

where we can see that $\lambda_i$ are the eigenvalues and $\mathbf{v}_i$ are the corresponding eigenvectors. Therefore, defining $\mathbf{y}_i = \sqrt{\lambda_i}\mathbf{v}_i$ then we can finally state that

$$\mathbf{Y}_r = \mathbf{V}_r \sqrt{\mathbf{\Lambda}}_r. \tag{1.13}$$

Obtaining then the principal coordinates associated, giving to us a result matrix with uncorrelated variables and the similar dimensions as the original. We notice that we are not going to obtain the exact values as the ones as when we started the technique, but its principal components, we are going to obtain conclusions from these components by plotting them. Finally what we have reached is to reduce the original dimension by preserving the original data losing irrelevant information.

MultiDimensional Scaling is very related to principal component analysis. Reducing the dimension of the data is the main objective of both techniques, as we will see in next section. However, a difference between MDS and PCA is that MDS works with the matrix of similarities.

### 1.1.2. Principal component analysis

One of the most commonly used technique is PCA. There is a lot of studies explaining this technique. We are going to explaining basing on Jolliffe (1986). It has become popular with the development of the computers, because they can perform lots of computations and represent the results. PCA is a linear transformation that establish a new coordinate system. This technique will describe the information given by a big data set $n \times p$ with a small set of variables (principal components) which are linear combinations of the former variables. These components are uncorrelated and the first one will be the most important. PCA will look for the projection in a line where the variance is maximum (1st component). Secondly, it will look for a perpendicular line that will keep the rest of the variability. We can have as much components as variables we have, assuming that $n > p$, being $p$ the dimension of the data and $n$ the number of samples. However, we expect that the first two components contain as much information as possible. PCA will give us a scatter diagram where we can see the samples with respect the principal components. PCA is commonly used to discover which are the important variables of a large data set. It helps us to reveal relationship that we might not have seen on a first sight. PCA is mainly used as an intermediate step in data analysis, to solve the problem that we have a large number of variables that make difficult our analysis.

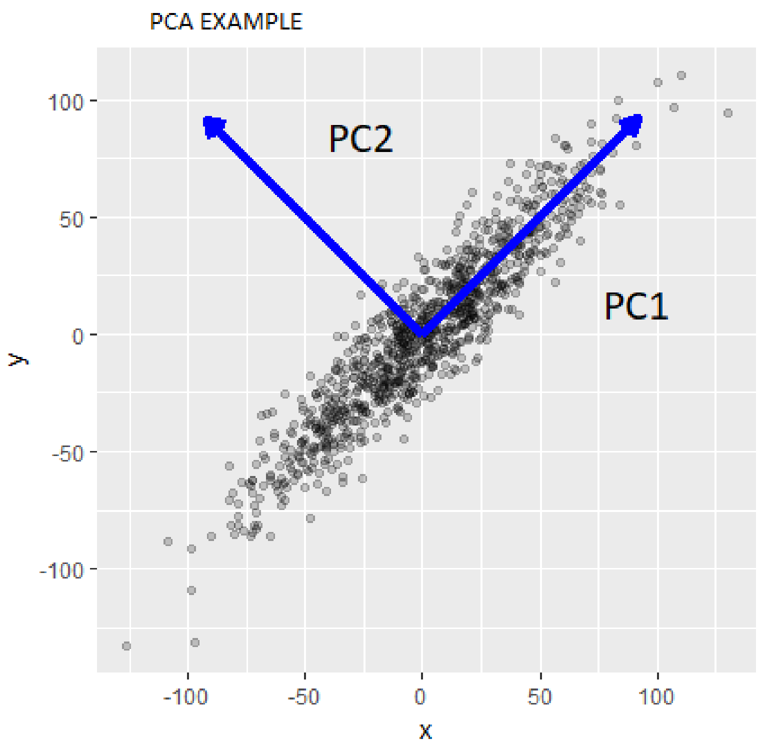


Fig. 1.1. Principal component analysis: In a large data set two components are assigned to the vectors that points to the maximum variance (Datascience, 2017).

The main objective of PCA is going to represent huge data in a dimension that human beings can obtain conclusions. This algorithm is assigning the data to a matrix taking into account that each column of the matrix is going to be a dimension. The first thing we have to obtain is the mean of the data matrix,

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & \cdots & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix}, \tag{1.14}$$

$$\bar{\mathbf{x}} = \begin{pmatrix} \bar{x}_{11} & \bar{x}_{12} & \dots & \bar{x}_{1m} \end{pmatrix}. \tag{1.15}$$

Once we have obtained the mean, we need to obtain the covariance matrix of the whole dataset.

$$\mathbf{S} = \frac{\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{y})}{n-1}. \tag{1.16}$$

Using the above formula, we can find the covariance matrix of **X**. Also, the result would be a square matrix of $d \times d$ dimensions. We have to take into account that this matrix is symmetric. This covariance matrix is composed by the variance and covariance among the different variables. The diagonal is formed by the variances of the variables, whereas the outer elements are the covariance among all the pair of variables.

The next step in PCA is to obtain the eigenvalues and the eigenvectors of this covariance matrix. The eigenvalues of **X** are roots of the characteristic equation. Disclaimer, we are using **X** to design the covariance matrix.

$$\det(\mathbf{X} - \lambda\mathbf{I}) = 0 \tag{1.17}$$

Which can be equivalently expressed as

$$\det\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & ... & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} - \lambda\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \tag{1.18}$$

So the equation we have to solve to obtain the eigenvalues is:

$$\det\begin{pmatrix} x_{11} - \lambda & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} - \lambda & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} - \lambda \end{pmatrix} \tag{1.19}$$

Once we have obtained the eigenvalues, we go back to the original expression and obtain the vector that solves the equation for each eigenvalue:

$$(\mathbf{X} - \lambda\mathbf{I})\mathbf{v} = 0 \tag{1.20}$$

The next step we should follow is to sort the eigenvectors by decreasing values and choose the number of eigenvectors with the largest eigenvalues that we are using to reduce our dimensions forming a matrix. The main idea of PCA is to project the space onto a smaller subspace, eigenvectors are going to form the axes of this new subspace. But we have to notice that these vectors only define the direction of the new axis, we have to remember their length is 1. Therefore, we need to decide which eigenvectors we are dropping for our low-dimensional space. We are going to drop the eigenvectors associated to the lowest eigenvalues since they have less information about the distribution of the data. Imagine we have used the two highest eigenvalues and these are its associated eigenvectors.

$$\mathbf{W} = \begin{pmatrix} w_1 & w_2 \end{pmatrix}. \tag{1.21}$$

Once we have kept the number of eigenvectors(associated to the higher eigenvalues) that we want to reduce our dimension to, we need to transform the samples onto the new space. We use this new matrix to transform our data to. To do so, we use the equation $Y = WX$ Peña (2002) .

Finally, recall that This procedure can specially be done thanks to the Spectral Decomposition Theorem, which says that a symmetric real matrix can be expressed as:

$$A = \lambda_1 v_1 v_1' + ... + \lambda_p v_p v_p', \tag{1.22}$$

where $\lambda$ are the *real* eigenvalues of $\mathbf{A}$ and $\mathbf{v}$ are the normalized eigenvectors associated to the eigenvalues of $\mathbf{A}$. This guarantees the existence of $\lambda_1, ..., \lambda_p \in \mathbb{R}$, whose non-negativity is ensured since $\mathbf{S}$ is semi-positive definite.

Therefore, we have reduced the dimension using the PCA technique, preserving the variation we had at the original dataset. If we plot the result, we could extract conclusions that otherwise we could have not.

## 1.2. What is *t*-SNE?

These techniques we have introduced (MDS and PCA) are linear, since they are based on linear algebra. However, there exist non-linear techniques. In this project we are focusing on one of them which is called: *t*-Stochastic Neighbour Embedding. All of these techniques have brought several benefits to the human beings. They can be used for different purposes as: Visualization/Interpretation of the data, clustering or Discriminant Analysis, discovering interesting characteristics of the data (patterns) or even to create several variables in order to perform a linear regression. So this data is not reduced only for human beings but also for different algorithms that will be faster thanks to this simplification.

*t*-SNE is a non-linear technique that is going to reduce the dimension following a different procedure than the previous techniques we described. As we explained before, PCA obtains the covariance matrix and computes the eigenvalues and eigenvectors so we can preserve the direction of the data. However, *t*-SNE works on a different way: imagine we have a cloud of points in a high dimension we want to move this cloud to a lower dimension that is also simpler, trying to preserving the structure of this points, specially the relation among the neighbours.

To obtain the probability of all the points, it will center a *t*-Student distribution to each value of the input, so it will use its density to obtain it. Therefore, it will convert the Euclidean distance between data points into conditional probabilities. *t*-SNE will minimize a cost function using a gradient descent method to retain the structure of the data in the low dimensional map. Using some tuneable parameters and optimising the objective function it will obtain a reasonable visualization of the data. In this, we see the a considerable reduction of dimensions without losing information.

There was a previous algorithm that was called simply SNE by Hinton and Roweis (2003). The main difference between these two algorithms is that SNE was employing a Gaussian distribution instead of *t*-Student when computing the similarity between two points in low-dimensional space. There are other differences that we will dive in section 2.2, such as the cost function will also vary from one technique to other. However, this differences made *t*-SNE stronger and easier to use than SNE (Van der Maaten and Hinton, 2008).

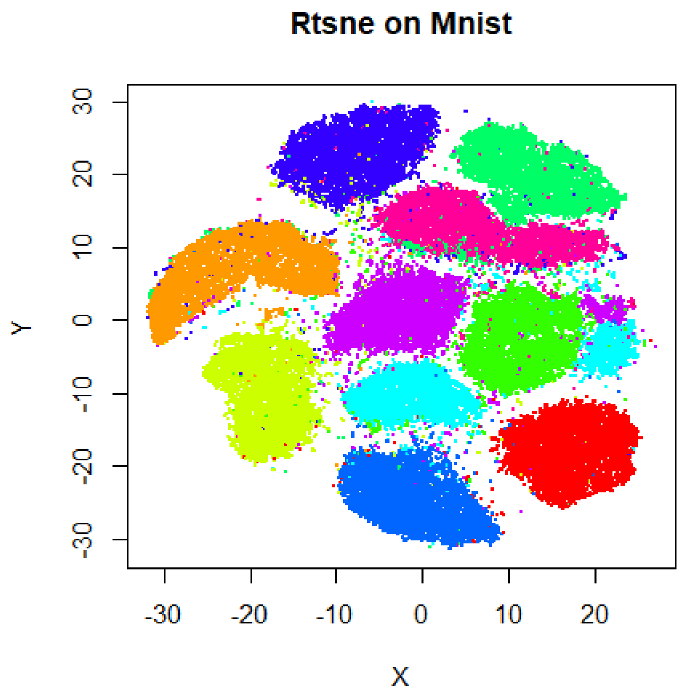


Fig. 1.2. Final stage of *t*-SNE applied to MNIST dataset. It can be seen that the numbers has been separated in clusters.

This figure shows a final stage of *t*-SNE applied to the well-known dataset MNIST. We can see a visual separation among the different digits that the set contains. *t*-SNE has reduced the dimensions of the dataset by creating these clusters that represent in two dimensions the original data, helping us to identify the different numbers at a glance.

In 2012, t-Distributed Stochastic Neighbour Embedding was chosen to win Merck Viz Challenge due to its incredible characteristics and its innovation.

## 1.3. Project objectives

The main objective of this project is to understand what are dimension-reduction techniques, why is it necessary to reduce the dimensions, to understand *t*-SNE technique perfectly, what is *t*-SNE applied to, knowing what is every parameter used for, how they behave when we tune them and implement our own version of the code.

The outline of the project will be as follows, in Chapter 2 we are deepening into the algorithm, we have learnt about the behaviour of the technique. Besides, we will follow the steps on the paper in order to learn and understand how is the complete process of the algorithm. Moreover, we will get in touch with the parameters that the algorithm will use and we will learn how to use them so we can see what are the consequences of modifying them, such as `perplexity` and `learning rate`. In this chapter we will also compare *t*-SNE with other techniques we mentioned in the introduction such as SNE, PCA and MDS. In Chapter 4, we are using all the knowledge we have acquired about this algorithm in order to show it in practical experiments. We are going to compare the difference between the function we have created for this project and the one that is provided by the authors in their GitHub Laurens Van der Maaten (2008). With the help of synthetic data (multivariate Gaussian mixtures), we test how the algorithm behaves in particular situations including more than two dimensions. We have also run the algorithm with the iris dataset. In Chapter 3 we are explaining the implementation of the algorithm we have developed for this project. In Chapter 5 we have applied our knowledge in *t*-SNE to a real dataset. We have used the well-known MNIST dataset so we can extract conclusions from the result, besides, we are using a a dataset that contains lots of people photograph, in order to see if *t*-SNE is able to reduce it dimension. We have analysed the regulatory framework in Chapter 6 in order to see how are we affected by law when studying and applying this technique. We have also prepared a planning of this project in Chapter 7 and we have included a Gantt Diagram that shows all the activities we are going to develop in order to reach our main goal. Finally we have studied an economical budget in Chapter 8 that evaluates how much is the cost of carrying this project out.

# 2. $T$-SNE EXPLAINED IN DETAIL

Nowadays, the amount of data has increased considerably, every application we download, every page we visit on the internet is storing data. There are computers storing all the information related to human, what do we buy or sell, which places do we visit most. Our smartphones are gathering lots of information that we can not see, because it depends on too many variables. So, the question is, how can we visualize this data, since it can be very useful to see patterns in the society, it can help to determinate which factors leads to a disease. Well, technology is evolving very fast, there exist algorithms that allows us to see this patterns created. It perform clusters in the data so we can see different groups that before applying this we may have not seen.

In this project, we are going to focus on $t$-student stochastic neighbour embedding ($t$-SNE). This algorithm takes this high dimensional data and creates a low dimensional space that preserves the neighbourhood among points. It will take into account several parameters to perform the algorithm. We have developed a code that imitates the behaviour of the real algorithm. The language we have selected to develop this algorithm is R (R Development Core Team, 2019). This programming language is very flexible calculating statistical analysis, is open-source and we can manage big quantities of data. The code is attached at the next chapter.

## 2.1. Learning process

```
   Data: Data set $\mathbf{X} = (x_1 + x_2, ...x_n) \in \mathbb{R}^{n \times p}$
1  cost function parameters: perplexity Equation (2.2)
2  optimization parameters: number of iterations $\mathcal{T}$, learning rate $\eta$, momentum $\alpha(t)$
   Result: low dimensional data representation $\mathcal{Y} = (y_1, y_2, ..., y_n) \in \mathbb{R}^{n*q}$
3  begin
4      Compute pairwise affinities $p_{j|i}$ with perplexity.
5      Set $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$ Equation (2.10)
6      Initial solution $\mathcal{Y}^{(0)}$ from $\mathcal{N}(0, 10^{-4}I)$
7      for $t = 1$ to $\mathcal{T}$: do
8          Compute low dimensional affinities $q_{ij}$ Equation (2.5)
9          Compute gradient $\frac{\partial C}{\partial \mathcal{Y}}$ Equation (2.13)
10         Set $\mathcal{Y}^t = \mathcal{Y}^{t-1} + \eta\frac{\partial C}{\partial \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{t-1} - \mathcal{Y}^{t-2})$ equation (2.14)
11     end
12     end
13     end
14 end
```

In Van der Maaten and Hinton (2008), the authors suggested this pseudocode we are following to explain the algorithm in detail. It contains all the parameters we have to compute in order to obtain a correct development.

In this pseudocode we can see that the first thing we are computing is the high dimensional matrix $\mathbf{P}_{ij}$ and after that we are computing the low dimensional matrix $\mathbf{Q}_{ij}$ in a loop that takes into account three versions of the low dimensional matrix in three different instants of time. As we can see it is going to depend on several parameters such as a cost function that has been minimized $\frac{\delta C}{\delta \mathcal{Y}}$ that is also affected by a learning rate $\eta$, reaching at the end a solution $\mathcal{Y}^t$ for the low dimensional matrix that preserves the neighbourhood that had the high dimensional data , besides it depends on a momentum $\alpha$ that will also vary depending on the iteration as we will see.

### 2.1.1. Starting with the high-dimensional data: $\mathbf{P}_{ij}$ matrix

Imagine we have a large dataset that depends on several variables, using $t$-SNE, we are going to obtain a visualization in a low dimension, to do it, the algorithm starts by computing the conditional probabilities $\mathbf{P}_{i|j}$ of $\mathbf{x}_i$ "being a good neighbour" of $\mathbf{x}_j$, these are all the points from the original dataset $\mathbf{x}_1, ..., \mathbf{x}_n$ in $\mathbb{R}^n$. For data points that remain close, $\mathbf{P}_{i|j}$ is going to be high, whereas for points that are very separated, $\mathbf{P}_{i|j}$ will be almost zero. The mathematical quantification of this idea is the probability:

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma^2\right)}, \tag{2.1}$$

Where $\sigma^2$ is the variance that represents the variability of the data with respect to its mean. The method we have followed to determine this $\sigma^2$ will be explained in section 2.1.2 when perplexity is introduced and as we will see they are very related. Notice that this matrix is going to have the same size as the data matrix.

What the algorithm is really applying is a softmax function which is being used to convert distances into probabilities, since the softmax function (Gao and Pavel, 2017) helps to reduce the data in the real space into values in the range [0,1]. When we developed the code, we had some troubles to compute the result, due to a number instability, which we will dive in later in section 2.3. However, we manage to fix it by using a function that takes this instability into account so we can perform the operations.

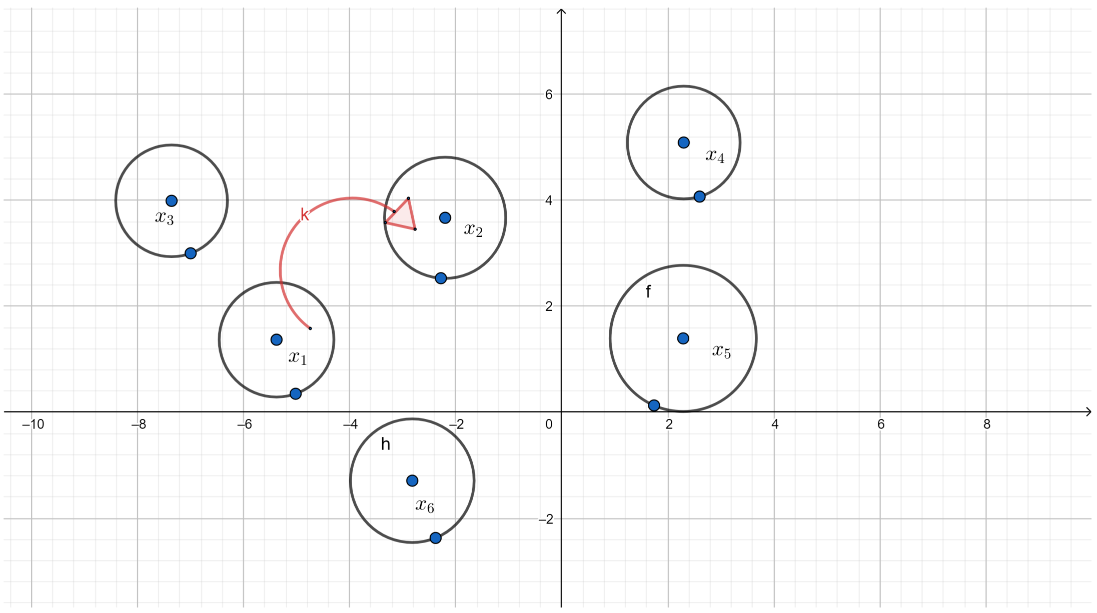


Fig. 2.1. Representation of the probability that $\mathbf{x}_1$ chooses $\mathbf{x}_2$ of being a neighbour.

As the author says in the paper: "The similarity of datapoint $\mathbf{x}_1$ to datapoint $\mathbf{x}_2$ is the conditional probability, $p_{2|1}$, that $\mathbf{x}_1$ would pick $\mathbf{x}_2$ as its neighbour if were picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_1$" (Van der Maaten and Hinton, 2008). In this figure generated with the Geogebra software Hohenwarter et al. (2018), we can see that the points are the center of the Gaussian probability density and we see the arrow as the probability that $\mathbf{x}_1$ could choose $\mathbf{x}_2$ as a "good neighbour", understanding this concept as there is a high probability of being chosen as a neighbour.

The conditional probabilities we are calculating along this algorithm can be calculated thanks to Bayes Theorem:

$P(a|b) = \frac{P(a \cap b)}{P(b)} = \frac{P(b|a)P(a)}{P(b)}$.

As we saw in the Equation (2.1), we will be able to compute the $\mathbf{P}_{j|i}$ matrix.

The $\sigma^2$ of this matrix is behaving as a tuning parameter that will be affected by the perplexity, this will be explained in the next section. Before continuing with the algorithm, Perplexity parameter must be introduced to understand the correct computation of the $P_{i|j}$ matrix. As we will see later, we will use $P_{ij} = \frac{p_{i|j}+p_{j|i}}{2n}$.

### 2.1.2. Introducing perplexity and its relation to $\sigma^2$

Perplexity is a fixed parameter introduced by the user by argument of the function. As we said before, this parameter is very important, since it is going to make modifications in the matrix previously introduced. This tuneable parameter is going to increase or decrease depending on our data. *Perplexity* is calculated by using:

$$P_i = 2^{H(P_i)}, \tag{2.2}$$

where $(H(P_i))$ is :

$$H(P_i) = -\sum_{j=1}^{n} p_{j|i} \log_2 p_{j|i} \tag{2.3}$$

With the help of the Shannon entropy we are able to compute the perplexity. This entropy will be non negative and this is obvious since the logarithm of a probability between 0 and 1 will be negative and corrected by the minus sign. The basic concept of this entropy is related to the uncertainty. It is the sum of the quantity of information that is given by the probabilities. The measure of information must be linear, if we change any probability a little, the entropy should also experience a small change. If all of the elements of our data are equally-likely then the entropy is going to be maximum.

What is perplexity going to be used for? It will be used to calculate an optimum $\sigma^2$ to calculate the values of $P_{ij}$ matrix. The procedure we have followed is to define a function $\text{Perp}_{\text{fixed}} - \text{Perp}_{\text{calculated}} = 0$ and we use an optimiser to minimize the equation. Recall we saw in Equation (2.1) that $P_{ij}$ values depend on $\sigma^2$ and in Equation (2.2) that *perplexity* depends on these values. We are calculating the value of perplexity that this $\sigma^2$ would produce and we want to make it equivalent to the perplexity that the user set by argument. With the help of a function, we are obtaining the $\sigma_{\text{opt}}$ that will produce that the $P_{ij}$ matrix has the perplexity fixed. This suppose that the $\sigma^2$ depends directly on the perplexity fixed by the user and therefore the matrix of $\mathbf{P}_{ij}$ is depending on it also. This is the reason why choosing a correct perplexity is really important.

Therefore, with some iterations we are obtaining the complete matrix of $\sigma^2$ that leads to obtain the perplexity user fixed as an argument. Using the perplexity calculated in each iteration and the one given as a parameter we can continue the algorithm.

This perplexity parameter indicates a guess about the number of close that each point has. As we will see, this perplexity will have a complex impact on the visualizations. By reading the original paper, we know that best performance will be produced when values are between 5 and 50. Otherwise, the behaviour will not be the one expected (Van der Maaten and Hinton, 2008). However, the value of perplexity depends on the number of samples we have, in order to obtain a good performing, we need that perplexity is less than the number of samples.

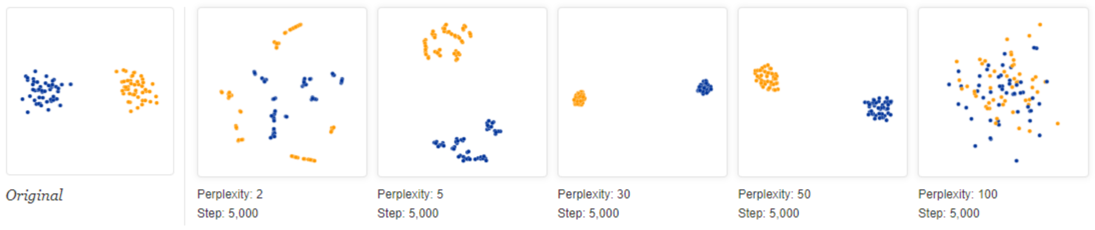


Fig. 2.2. For different values of perplexity we can see different behaviours of $t$-SNE, if we do not satisfy the condition that Perplexity $< n$ the algorithm will not converge (Wattenberg et al., 2016).

When using the algorithm in R, the function does not let us to use a value of perplexity higher than the number of points, we had to use an online tool that did allow us to do some experiments tuning the perplexity (Wattenberg et al., 2016). As we can see in the figure, for two Gaussian mixtures of 30 points each, for the different values of the perplexity we are obtaining different results. The first capture indicates how is distributed the data before applying the algorithm. Once we have applied the algorithm with a value of perplexity inside the operable range we can see that the points are taking into account the Euclidean distances. However, when the perplexity is above 50 and also bigger than the number of points, we can see that the algorithm can not converge the points leaving giving a poor quality result.

Once we have computed these parameters we are able to compute the matrix of $P_{i|j}$. We have to take into account that since we are interested in modelling pairwise similarities we will set the values of $P_{i|i}$ and $P_{j|j}$ to zero. However, this conditional matrix is not going to be the final one in $t$-SNE. Since $t$-SNE uses a symmetrized version of the conditional probabilities, we use $P_{ij} = \frac{p_{i|j}+p_{j|i}}{2n}$ that in fact, this solve a problem that occurred in SNE.

### 2.1.3. Obtaining the low-dimensional data: $q_{ij}$

Now we have obtained the high-dimensional matrix, we have to compute the low-dimensional matrix that will vary in the algorithm to reach the final result as we explained in the pseudocode. The main question to solve is how this matrix is computed. This low dimensional matrix preserves the neighbourhood of the high dimensional matrix, so it will be a faithful representation of the high one thanks to maintain this neighbourhood untouched.

When SNE was developed, the formula was used to compute the low dimensional data was the one as follows:

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i}\exp\left(-\|y_i - y_k\|^2\right)}. \tag{2.4}$$

The low dimensional part implemented in SNE started centering in a Gaussian distribution with variance $\sigma^2$ equal to $\frac{1}{\sqrt{2}}$ to model the similarity of map point $y_j$ to map $y_i$. If the map correctly model the similarities between the high dimensional points, then, the conditional probabilities $q_{i|j}$ and $p_{i|j}$ will be equal. Then, the main objective of SNE is to minimize the mismatch between those two.

This two matrices, low and high dimensional, will be used to minimize the cost function that we will deepen later.

When the Gaussian was used in SNE, there appeared a problem that is called the 'crowding problem'. When we were trying to calculate the equidistant probabilities in a high-dimension data, for example ten dimensions, we may have a huge number of points that there are equally distant in that dimension, however, there is no feasible way to model it in two dimensions. When we try to do this, the algorithm tends to 'crowd' the data in one point, since is the only way the algorithm calculates the minimum distance among them.

However, in the paper the authors came up with some ideas to solve the problems of SNE, firstly, he uses a symmetrized version of the cost function with simpler gradients and it will be used a Student-t distribution rather than a Gaussian when we want to calculate the similarity between two points in the low-dimensional space.

The employment of this $t$-distribution will modify the low-dimensional matrix $q_{ij}$:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}. \tag{2.5}$$

Notice that in $t$-SNE we are using joint probabilities instead of using conditional probabilities. This is due to a concept of symmetry we will explain in next section.

The author says in the pseudocode that we can implement the matrix of the low dimension, initialized in an Normal distribution with zero mean and variance equal to $10^{-4}$. In $t$-SNE, we employ a Student t-distribution with one degree of freedom as the heavy-tailed distribution in the low-dimensional mapping. It has a property that approaches an inverse square law for large pairwise distances in low dimension. This will solve the problem we mentioned before, because large clusters that are far apart interact the same way as individual points. This can be theoretically proved because Student t-distribution is an infinite mixture of Gaussians. Since it does not involve an exponential is much faster to evaluate the density in the new one.

### 2.1.4. Cost function: Kullback–Leiber divergence

In SNE, the algorithm uses the Kullback–Leiber divergence to measure the distance between the conditional probabilities of the two matrices. This divergence is commonly used to indicate the similarity between two functions. In information theory, it is known as the divergence of the information or relative entropy. We have to take into account that is a divergence and not a metric because it has no symmetry. Since we want to make them as similar as possible, we are trying to minimize this cost function. The Kullback–Leiber divergence has some interesting properties:

- it is always positive.
- it is null only if $P_{i|j} = Q_{i|j}$

Therefore, the cost function C is given by:

$$C = \sum_i \sum_j p_{i|j} \log \frac{p_{i|j}}{q_{i|j}}. \tag{2.6}$$

Interpreting this divergence we can see that if $P_i$ increases, then this means that the object is closer. Besides, if $Q_i$ increases whereas $P_i$ is decreasing, this means that we are approaching two objects that were far. This approximation is not good, however, the error we are obtaining is inappreciable, since our Q-ith element is weighted by $P_{j|i}$. Then, *t*-SNE preserves the structure of the neighbourhood without taking into account the global structure (Otterbach, 2016).

However, *t*-SNE is using a symmetric version of SNE, then the algorithm is using the joint probabilities instead of the conditional probabilities. It is also possible to minimize a Kullback–Leiber between joint probability distribution:

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.7}$$

Moreover, we have some interesting properties at this point, because $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$. Besides we experiment some changes in the calculation of high and low dimensional matrices:

$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/\sigma^2)}{\sum_{k \neq l} \exp(-||x_k - x_l||^2/\sigma^2)}, \tag{2.8}$$

$$q_{ij} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq l} \exp(-||y_k - y_l||^2)}. \tag{2.9}$$

But this can occasion problems when a high-dimensional datapoint $x_i$ is an outlier. When this happens, the value of $p_{ij}$ is very small for all $j$. Therefore, the new location on the low dimension have a small effect on the cost function. The authors of the paper define the joint probability of high dimension:

$$P_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}, \tag{2.10}$$

this will ensure that makes a big contribution to the cost function. We can see that this can be reached by adding the conditional probability matrix and its transposed.

With this solution, we have computed the high dimensional matrix of the data so we can minimize the cost function. To perform this operation, we are using a gradient descent method, which broadly consist on seek for the minimal point by going in the direction of the slope of the function we are minimizing by doing several iterations. Since we have used a symmetric version of SNE we will experience a simplification of the gradient. The gradient in SNE has this form:

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{2.11}$$

It can be seen that the Kullback–Leiber divergence is used over the conditional probabilities. We can experience a long time to converge, since we are using exponentials for all the probabilities for point $i$ and $j$. We may experience a big trouble to optimize this gradient.

By using a symmetric version of SNE, in $t$-SNE we see a simplification in the gradient:

$$\frac{\partial C}{\partial y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j). \tag{2.12}$$

Since we are using a $t$-distribution, the formula of the gradient we are using will introduce another term:

$$\frac{\partial C}{\partial y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}. \tag{2.13}$$

This equation for the gradient will be the one in charge to minimize the cost function.

### 2.1.5. Gradient update

Once we have obtained all the parameters needed to compute the gradient, the algorithm will perform a gradient update. This will take into account some parameters as the momentum or the learning rate. For some iterations the gradient will be updating its value and so the low dimensional matrix. Using the previous calculation the algorithm will make that the difference between the matrix **P** and **Q** gets smaller. Mathematically, the procedure we must follow is given by

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\partial C}{\partial y_i} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}) \tag{2.14}$$

We are repeating this procedure for a number of T repetitions, depending on what the user set by argument in the function. Although it is recommended a high value, there may reach a point where there is no sense in increasing anymore this value since it is already stabilized and more iteration will not mean better convergence.

It is important to recall that the gradient update will be repeating that procedure for each iteration. The low dimensional matrix is initialized by using a random Gaussian distribution with zero mean and variance $10^{-4}$, however, it is being affected each iteration by the low dimensional matrix in the previous two instants and some optimization parameters: number of iterations T, learning rate: $\eta$, momentum $\alpha(t)$. Thanks to these parameters, the low dimensional matrix is going to reach an optimal value that reduces the dimension of the high dimensional matrix. Therefore, they are decisive for the correct implementation of the algorithm.

The learning rate is a tuning parameter that will affect the behaviour of the algorithm. If the cost function gets stuck in a bad local minimum increasing the learning rate may help.

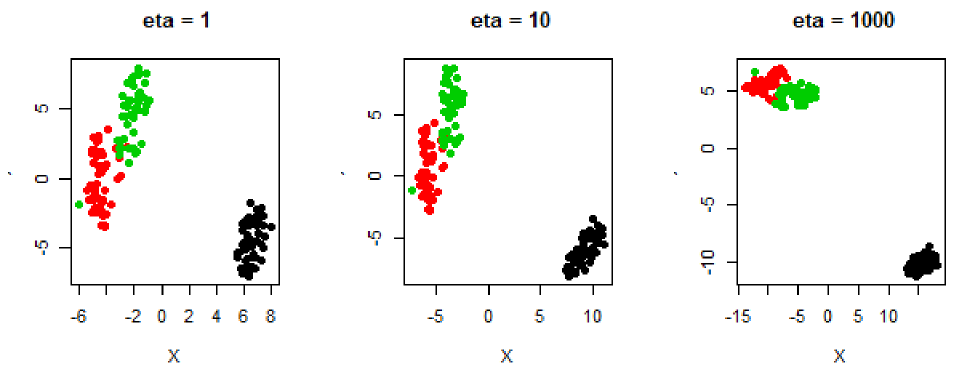


Fig. 2.3. We can see how different $\eta$ will affect the behaviour of $t$-SNE when applied on multivariate Gaussian mixtures just changing the perplexity and using the same parameters for the different runs.

As we can see in the plots, if the learning rate is too high, the data may look like very compressed in clouds of points. On the other hand, if the learning rate is too low, most points may look like very spread with any point approximately equidistant from its nearest neighbours.

The other interesting parameter that infers directly in the algorithm is the momentum ($\alpha$), the author says in the paper that it is usually initialized to 0.5 for the first 250 iterations and it is set to 0.8 after these iterations.

## 2.2. Relation to other techniques

### 2.2.1. Comparing *t*-SNE with SNE

In this section we are going to compare $t$-SNE algorithm with other techniques. We have to take into account that $t$-SNE is the advanced version of SNE, that has been introduced when explaining several parameters.

Notice, $t$-SNE corrected some troubles that were found in SNE, for example it solved the well-known crowded problem, which occurred when trying to compute the probabilities in a high dimensional data. When there were a lot of dimensions there could be so many points so when the algorithm tried to reduce the dimensions we found that it was not possible, and the algorithm crowded the data in one point. In the paper, the authors say that the slight repulsion is created by introducing a uniform background model with small mixing proportion $\rho$ (Van der Maaten and Hinton, 2008). Therefore, no matter how far are two map points, $q_{ij}$ can never fall below $\frac{2\rho}{n(n-1)}$. We can see that datapoints that are far in high dimension, $q_{ij}$ is larger than $p_{ij}$ which causes a slight repulsion. This technique is called UNI-SNE. Although the optimization is difficult, the authors say in the paper that the best optimization method is to start by setting the background mixing proportion to zero. After this, it can be increased to allow some gaps to form between natural clusters

(Van der Maaten and Hinton, 2008).

Other problem that $t$-SNE solves from SNE is the difficult optimization that the cost function had, this time, $t$-SNE is using a symmetric version, that will have the interesting property that the probability $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$. Now we are going to use the matrix $P_{ij} = \frac{P_{j|i}+P_{i|j}}{2n}$. This will also ensure that $\sum_j P_{ij} > \frac{1}{2n}$. Therefore, instead of minimizing the Kullback–Leiber divergences of the conditional probabilities, $t$-SNE will minimize the joint probabilities. This modification modified the cost function, simplifying the optimization and obtaining even better results than SNE.

Besides, other trouble that $t$-SNE improves over SNE is that mismatched tails can compensate for mismatched dimensionalities, that is the reason why $t$-SNE is using a Student $t$-Distribution with one degree of freedom instead of using a Gaussian distribution as we saw in SNE. Therefore, in low dimensional we find a heavier tail and we will alleviate the crowding problem. The authors justify this change by affirming that these two distributions are closely related, since the Student $t$-distribution is an infinite mixture of Gaussians. An important property is that we are reducing the computational time, because it does not involve an exponential. Then the gradient will be much easier based on joint probabilities:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{2.15}$$

If we compare both algorithms applied to a classical dataset as MNIST numbers, which contains lots of handwritten digits, after using both algorithms we obtain two different results (LeCun and Cortes, 2010):

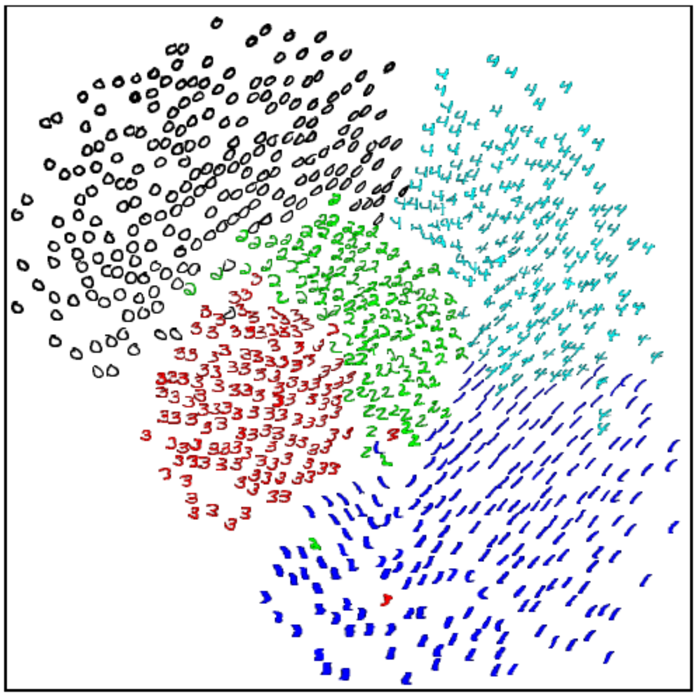


Fig. 2.4. In this figure we can see how SNE applied to MNIST numbers, we see that it has separated the different digits that the dataset contains (Hinton and Roweis, 2003).

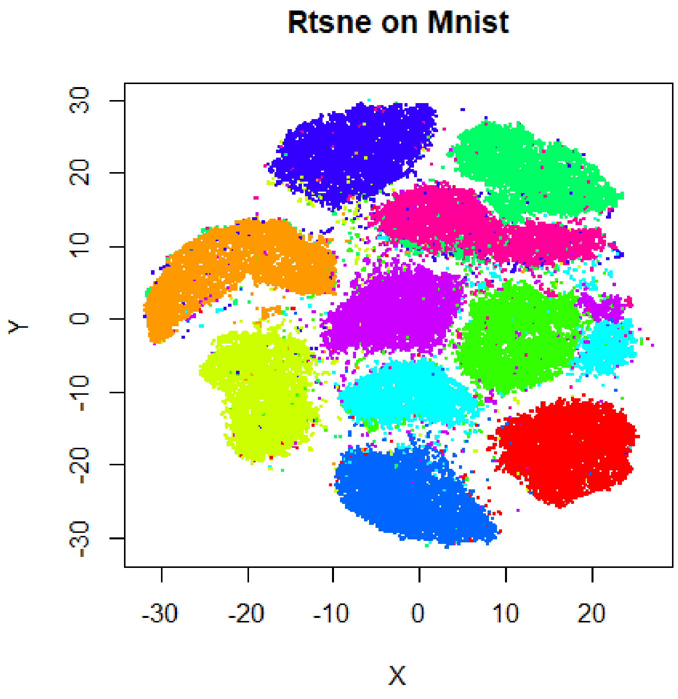


Fig. 2.5. *t*-SNE final stage applied to MNIST dataset, where we can see the different numbers separated in clusters by different colours.

If we pay attention to this two figures, we can see how *t*-SNE has improved considerably its performance. We can see very well differentiated clusters while preserving the neighbour structure. However, if we use the previous algorithm we can see also differentiated clusters but the performance is a little worse. Also the computational cost was higher on SNE.

### 2.2.2. Comparing *t*-SNE with PCA and MDS

Other algorithm that also reduces the dimension and that we have introduced is PCA. There is a huge different between these two algorithms, firstly, as we explained before, PCA is a linear transformation focus that will calculate the maximum variability in a dataset and will assign it to a principal component where later we will project the original data. Alternatively, *t*-SNE is computing a whole new space with the condition of preserving the neighbourhood of the original data.

Due to its computation, *t*-SNE is going to take several hours when trying to reduce million sample datasets, on the other hand, PCA will finish in a few minutes, since the calculus are not as complex. Notice that when we specified that we have to compute the $\sigma^2$ optimum if there is a huge dataset, this optimization can take too long. PCA is based on linear algebra, is a mathematical technique based on calculating the eigenvectors and the eigenvalues, on the other side, *t*-SNE is a probabilistic one, it is using the probabilities to preserve the high dimensional space. Another special difference we can notice is that PCA tends to place dissimilar points far apart in a lower dimension representation (Smith, 2002). However, when we have to represent high dimension on a low dimension, non linear algorithms can represent similar data points close together, in this case PCA can not perform this operation. We have notice that when running *t*-SNE different times with same parameters we may obtain different results while this not occurs when using PCA, this occurs because *t*-SNE is having a random initialization.

For example, comparing the results obtained applying these two algorithms to MNIST numbers:

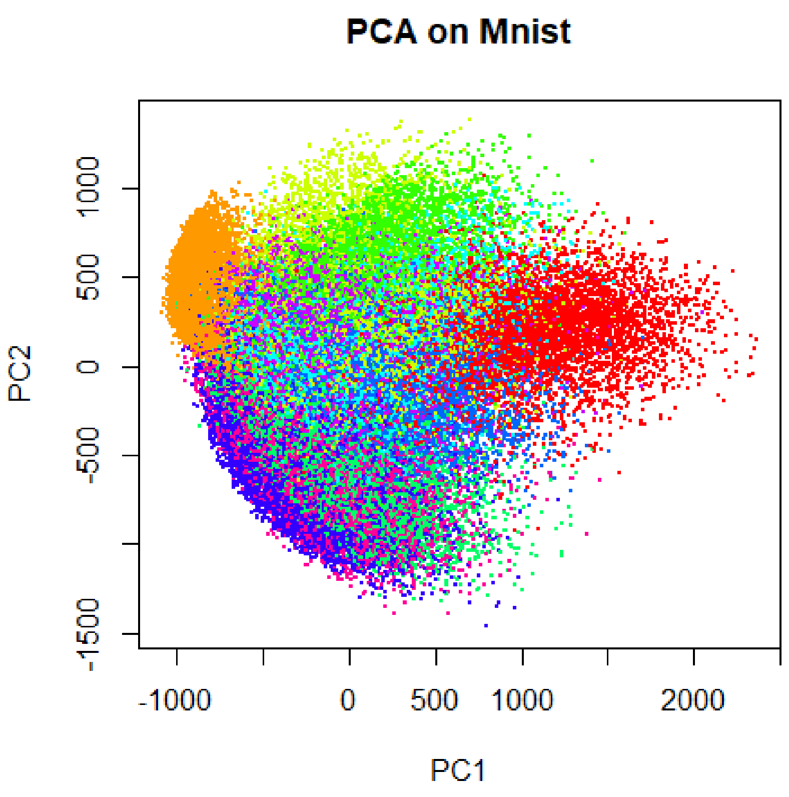


Fig. 2.6. Using PCA to reduce the dimension of MNIST dataset, we can see that numbers are represented in two dimensions, but we do not see a clearly differentiation among the numbers.

We can see different behaviours between the algorithms, on one hand we can see how *t*-SNE has cluster the different numbers preserving the neighbourhood of the datapoints. On the other hand, it can be seen that PCA has obtained the maximum variability and projected the data onto the principal components calculated. PCA is preserving the long distances to maximize the variance, so when two number are very different they are very separated. This can lead to a deficient visualization specially when we use non linear structures, where *t*-SNE works very well, such as cylinders or curves. Instead of that, *t*-SNE preserves small pairwise distances (Miao, 2015). We can see that PCA is not clearly separating all the numbers in different clusters and *t*-SNE does.

On the other hand if we compare *t*-SNE with MDS we notice that MDS is a generalized concept of PCA so the results obtained with both techniques are going to differ. However, we can think that MDS and *t*-SNE have some ideas that are similar. As we explained before, MDS is obtaining what we call a 'similarity matrix' based on squared distances and as we have explained, *t*-SNE is basing its computing also in distances among datapoints. However, the development of the algorithm will be pretty different. MDS is a linear technique that is going to compute the covariance matrix and like PCA it is going to compute the eigenvalues and the eigenvectors in order to reconstruct the original matrix but preserving the distances among the datapoints (Peña, 2002). As we have explained before, *t*-SNE is computing the new low dimensional matrix that will preserve the neighbourhood of the high dimensional matrix. Another difference we notice is that *t*-SNE is preserving better the global neighbour structure, whereas MDS works better in local pairs of datapoints.

If we compare both techniques development on MNIST dataset (LeCun and Cortes, 2010), we obtain the following results:

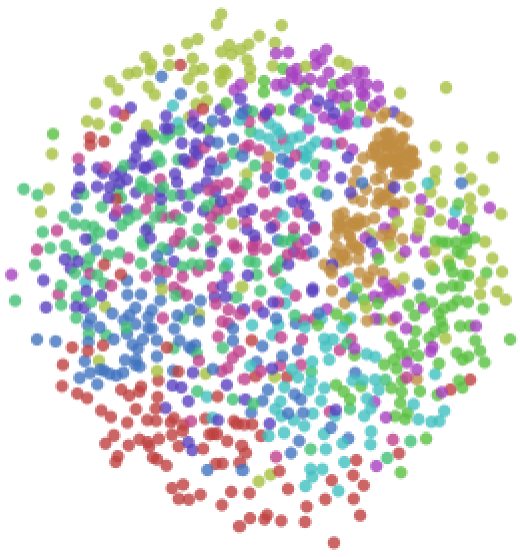


Fig. 2.7. MDS is being applied to MNIST dataset, representing in two dimensions the original data but there is not a clearly differentiation among the digits (Olah, 2014).

We can spot huge differences between these two interpretations of the dataset, MDS has not separated the handwritten digits, although it has managed to differentiate some digits, the definition is not as well as when we use *t*-SNE.

## 2.3. Practical issues

On our learning process of *t*-SNE we have spot that it has some troubles that might difficult the correct development of the algorithm. One of the main problems we have noticed is the moment we introduce a new point in our dataset, the algorithm can not conserve all the computations it did for the first time. Since when we introduce a new point, *t*-SNE must calculate all the distances in order to create a correct low dimensional matrix that preserves the neighbourhood of the original dataset. Due to the development of the algorithm, when we introduce a new point, *t*-SNE must be run again in order to obtain the correct results.

Other practical issue we have noticed is that due to the optimization process of the cost function, the computational cost is increasing when we use a dataset that contain a vast amount of samples and dimensions. Therefore, for a huge number of dimensions, the time we have to wait in order to see results is going to be really large. This is caused because in practice, it means to compute lots of loop for, which is traduced in increasing the time. However, Van der Maaten (2014) came up with the idea of using Barnes Hut implementation and Tree-based algorithm in order to reduce considerably the time. When we were evaluating the optimization of the gradient, it was running an $O(N^2)$, which is considerably increasing the computational cost, with the Barnes–Hut approximation, now it runs on $O(N \log N)$ by treating clusters of faraway objects as single particles(Barnes and Hut, 1986). It requires $O(N)$ memory and it is proved that is considerably faster than the original $t$-SNE. It accelerates the computations. Some drawbacks that are mentioned are that Barnes–Hut–SNE does not provide error bounds and that it can only be used to embed data in two or three dimensions. On the other hand the author mentions that Dual-tree is computing cell-cell interactions whereas Barnes–Hut was point-cell interactions. Therefore, it brings as a consequence a speed up in the algorithm. However, dual-trees needs to store a list for each node during tree construction which means an increase on the cost and memory. The author suggest that Barnes–Hut is better in terms of trade-off between accuracy and speed (Van der Maaten, 2014).

We have to mention another issue we have noticed during the development of the algorithm and it is related to the perplexity. As we have explained, this interesting parameter is going to be fixed when we start the algorithm. As user, we are going to set it, depending on our data, to a higher or lower value. At the beginning we may not know which is a correct value, the author suggest that the optimal values must be between 5 and 50, however depending on our data we may need a higher value. Besides, we had a small problem when we were programming the algorithm and it is number stability. Since we are performing an optimization for the whole matrix with a specific value of perplexity, there are some values that can be optimised in order to obtain that perplexity, as we explained in the development of the $\mathbf{P}_{ij}$ matrix, but there are others that can not, leading to an infinite value. This problem is caused due to the universal perplexity. For the future, we may use different perplexity values for different clusters in order to solve this problem.

One last thing we want to mention as a little issue is that if we pay attention to the plot of the result of the algorithm, we may see that cluster sizes are different, independently of the input data. If we introduce two mixture of Gaussian distributions with different dispersion, we may expect that in the result they are also with different dispersion, however, the algorithms only takes into account distances, so the resulting clusters are not affected by the original dispersion.

# 3. IMPLEMENTATION OF THE ALGORITHM

In this chapter, we are going to expose the implementation of the code we have developed. Disclaimer: this implementation is a pedagogic version and it is not the optimal solution, just an implementation meant to illustrate how the theory materializes into code. We have developed some functions in order to simplify the calculations of the algorithm.

## 3.1. Auxiliar functions

We have developed some auxiliar functions that are going to be useful for the correct development of the algorithm. Since we have used well-known functions as softmax in Equation (2.1).

```r
# This functions will compute the softmax function in a numerically
# stable way. The equation 2.1 will be implemented using this
# concept.

logsumexp <- function (x) {
    y <- max(x)
    y + log(sum(exp(x - y)))
}

softmax <- function (x) {
    exp(x - logsumexp(x))
}
```

## 3.2. Internal functions

To continue the performance of the algorithm, we had to develop some functions that helped us to compute the high dimensional matrix. In this part, we are calculating the optimal $\sigma^2$ that is going to help us to calculate the matrix of $\mathbf{P}_{ij}$, with the help of the perplexity. To do so, we have come up with the following idea, solving the Equations (2.1) and (2.10):

### 3.2.1. Matrix $\mathbf{P}_{ij}$

This part of the code is really important, since we are obtaining the optimal $\sigma^2$ that will make that our perplexity is the same as the value we introduced by argument in our *t*-SNE function. We have used the optimizer `optim` that has given to us good results apart from handling the number instability we obtained when we tried to use other optimizer functions.

```r
# This function is in charge of computing the sigma optimal
# parameters to calculate the P_{ij} Matrix with the values
# adapted to the perplexity fixed by argument. This part of the
# code refers to the Equation (2.1) and (2.10).

calcP <- function(X, perplex) {
  n <- nrow(X)
  # This is the optimal $\sigma ^ 2$ that we are computing
  # according to the perplexity fixed.
  sigma2_opt <- rep(1,n)
  sigma2_opt <- sapply(1:n, function(i) {

  # The following function performs the optimization in order to
  # obtain the sigma associated. This is explained in Section
  # 2.2.
  optim(par = 0.75,
         fn = function(s2) {
             res <- (calc_perplexity(X = X, i = i,
                     sigma2 = s2) - perplex)^2
                     ifelse(is.finite(res), res, 1e6)
          },
          method = "L-BFGS-B", lower = 0.1)$par

  })

  # We are computing the Equation (2.1) with the sigma optimized
  # previously calculated. We use the auxiliar functions we
  # developed in the first section in order to satisfy the
  # equation.
  P_i_cond_j <- matrix(0,n,n)
  P_i_cond_j <- sapply(1:n, function(i) {
        softmax(-rowSums(t(t(X) - X[i, ])^2)/
                (2 * sigma2_opt[i]))

  })

  # Equation (2.10).
  P_ij <- (P_i_cond_j + t(P_i_cond_j)) / (2 * n)
  return(P_ij)

}
```

Returning as a final value the complete high dimensional matrix $\mathbf{P}_{ij}$.

### 3.2.2. Perplexity

In this code, we obtain a solution to calculate the perplexity with the help of the auxiliary functions we developed before. Coding the Equation (2.1) we will be able to compute this parameter. This parameter is being obtained thanks to a sigma given by argument. So the perplexity will depend directly on the value we are introducing. In the previous function we were calculating the optimal sigma that will make that the perplexity is the same as the value we introduced by argument in the *t*-SNE function.

```r
# This function is computing the perplexity for the sigma^2 given
# for a particular i.

calc_perplexity <-function(X, i, sigma2) {

  # Dimensions
  n <- nrow(X)
  p <- ncol(X)

  # it computes the equation (2.1) for a value of sigma given by
  # argument.
  P_i_cond_j <- softmax(-rowSums(t(t(X) - X[i, ])^2)/
                      (2 * sigma2))

  # We are calculating the entropy
  # with the Equation (2.3).
  H_i <- -sum(P_i_cond_j * log2(P_i_cond_j))


  # returns the perplexity given in Equation (2.2).
  return(2^H_i)

}
```

## 3.3. Complete algorithm

In this part we are computing the whole *t*-SNE algorithm. We have followed the pseudoalgorithm described in Chapter 2 and explained in the paper Van der Maaten and Hinton (2008).

Firstly we can see the declaration of the function, we can introduce as a parameter all the arguments we have explained.

```r
# Main function in which we will have to introduce the parameters
# we want to use to develop the algorithm

tsne <- function(X, q = 2, T = 1e3, learning_rate = 100,
                 momentum = 0.5, perplexity = 35,
                 cols = 1) {
  # Dimensions
  n <- nrow(X)
  p <- ncol(X)
```

### 3.3.1. Generating high-dimensional matrix

In this part of the code we are generating the high dimensional matrix: $\mathbf{P}_{ij}$, we have applied the functions we developed in the previous section. This matrix is composed by $n$ samples and $p$ variables.

```r
# This P is calculated using the perplexity given by argument by
# the user. Using the previous code is obtaining an adequate
# matrix of P

P_ij <- calcP(X, perplex = perplexity)
```

### 3.3.2. Generating low-dimensional matrix

To generate the low dimensional matrix $\mathbf{Q}_{ij}$ we have to set the matrix with $n$ rows and $q$ columns. Being $n$ the same number of samples as the data in the high-dimensional matrix. However, this $q$ is the number of the dimensions we want to reduce to. It is initialized, as is described in Van der Maaten and Hinton (2008), with a Gaussian distribution with zero mean and variance $10^{-4}$.

```r
# Initial configuration of the low dimensional data given in the
# pseudocode.
set.seed(123456)
Y_t <- mvtnorm::rmvnorm(n = n, mean = rep(0, q),
                           sigma <- diag(rep(1e-4, q)))
# We initialize the three instances at the begining.
Y_t_1 <- Y_t_2 <- Y_t

# The gradient is initialized as a matrix nxq where n is the rows
# of the data and q the columns of the reduced dimension.
gradient <- matrix(0, nrow = n, ncol = q)
for (t in 1:T) {

# Compute the low-dimensional affinities q_{ij}'s
# We use the following formula (1 + ||y_i - y_j||^2)^{-1} given
# in the equation 2.5.
Q <- as.matrix(1 / (1 + dist(Y_t)^2))
# Q_ij
# Sum for rows, except for the diagonal
v <- sum(Q)
# Quotient with an implicit column recycling
# Obtaining the low dimensional matrix.
Q_ij <- Q / v
```

### 3.3.3. Following the algorithm

To perform the algorithm, the previously generated low dimensional matrix is going to be varied in several iterations. As we explained in the Equation (2.14), we can see that the gradient is affecting directly depending on the learning rate parameter. Besides, the momentum introduced by argument is going to vary the solution taking into account the low dimensional matrix in two different instants of time.

```r
    # Gradient used in the equation (2.13)
    # p_ij - q_ij
    dif_P_Q <- P_ij - Q_ij

    # Fill gradient
    for (i in 1:n) {
          gradient[i, ] <- 4 * colSums(dif_P_Q[i, ] * t(t(Y_t)
                                            - Y_t[i, ]) * Q[i, ]) }
    # Update the Y following the algorithm described in the paper.
    # It is described in the equation (2.14).
    Y_t <- Y_t_1 + learning_rate * gradient
              + momentum * (Y_t_1 - Y_t_2)
    Y_t_2 <- Y_t_1
    Y_t_1 <- Y_t
    if(t > 700) {
        momentum <- 0.8
      }
```

### 3.3.4. Plotting the results

Since the algorithm is converging depending on the situation, we have plotted the results each 200 iterations, so we can see how it is evolving until it finally reaches a solution.

```r
    # There is a plot each 200 iterations to show the development of
    # the algorithm.
    if ((t %% 200) == 0) {
           message("Iteration #", t)
           plot(Y_t, pch = 15, col = cols)
      }
    }
    return(Y_t)
  }
    # The resulted obtain is the data divided in clusters preserving
    # the neighbourhood of points.
```

As a result, we would see how the points are spreading and approaching among them to finally reach a solution. If the algorithm has been applied correctly we would see the data separated in clusters. Once we have reduced the dimensions of visualization we can obtain conclusions. This reduction of dimension has implicated some loss of data, however, the results we can see are faithful to the original data.

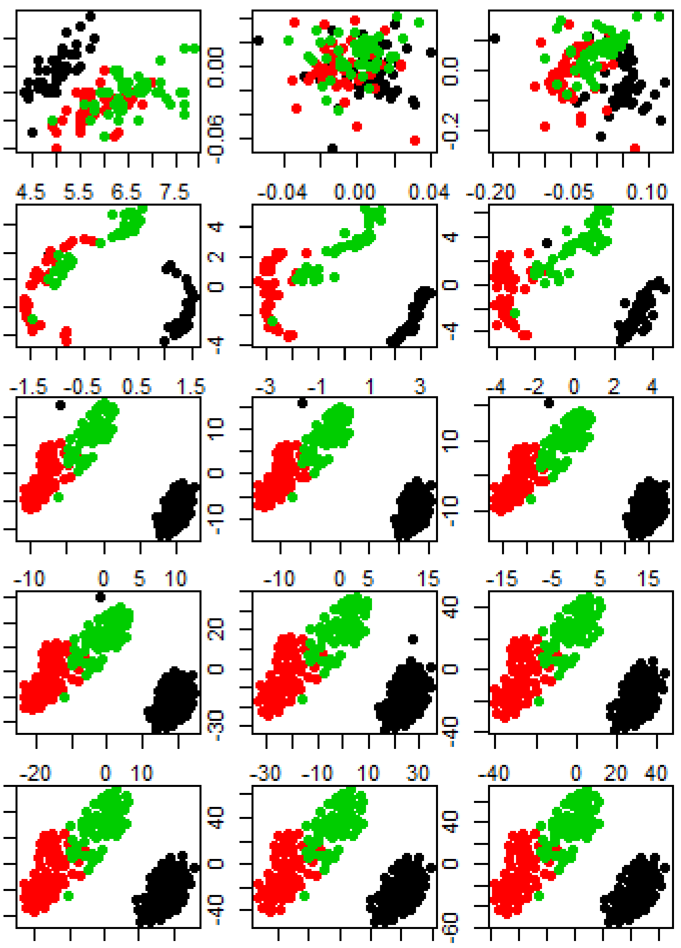


Fig. 3.1. Figure that shows the stages of the *t*-SNE version we have implemented in this chapter.

In this figure we can see at first the original iris dataset (Fisher, 1936) and then a plot of the state every 200 iterations of the algorithm. Notice that *t*-SNE is converging until it reaches an optimal result where we can see clearly differentiated clusters. We can see in the upper part of the sixth plot how there is a black point really close to the red and green clusters and as the algorithm perform operations, the black point is moving until it finally reaches the black cluster.

The code is available on GitHub:

https://github.com/100346868/TSNE

# 4. NUMERICAL EXPERIMENTS

Nowadays, *t*-SNE has gained reputation among the other algorithms, due to its versatility, speed and easy visualization of the results obtained. This algorithm has been implemented in the most common programming languages such as Java (Lejon, 2018), Python (Pedregosa et al., 2011), R (R Development Core Team, 2019) or Matlab (MATLAB, 2010) and so more languages as we can see in the original GitHub page of the authors (Laurens Van der Maaten, 2008). The language we have decided to implement our version is R because its flexibility and its collection of state-of-the art of statistical packages

In order to compare our code with the original implementation, we have used the well-known version of the implementation that is called `Rtsne` (Krijthe, 2015). Just including a library in R-Studio we can fully use *t*-SNE algorithm.

```
    Rtsne(X, dims = 2, perplexity = 30, pca = TRUE, max_iter = 1000,
          Y_init = NULL, pca_center = TRUE, pca_scale = FALSE,
          momentum = 0.5, final_momentum = 0.8,
          check_duplicates = TRUE, eta = 200,
          exaggeration_factor = 12)
```

The first parameter we introduce in the function is our data, we have to notice that it does not contains duplicates for a better performance of the algorithm. However, the algorithm includes a parameter so we can check if our data has duplicates.

In this function already implemented, we can perform a PCA operation before applying *t*-SNE, this is going to be really useful when we manage a huge dataset, since it is going to simplifies our data by reducing the dimension of the data so *t*-SNE can perform its operations in a reduced time. The next interesting parameter we can use is `dims`, we can introduce in it the number of dimensions we want to reduce to, a usual value is 2 or 3 dimensions, this typical values are giving to us a representation humans can interpret easily.

As it was explained in the previous sections, the algorithm must perform some iterations, we can modify this number so it suits to our needs. Notice that a higher number does not implicate a better representation, there is a situation when the algorithm converges so it does not suffer from bigger variations. Some typical values for this parameters should be between 1000 and 5000 iterations.

Other parameter we can also modify is the momentum, besides the author in the paper suggest to modify it when we reach a determined number of iterations, it is really useful in order to speed up the optimization and to avoid poor local minima. As it is specified in the paper it is set to $\alpha(t) = 0.5$ for $T < 250$ and we see a change that makes $\alpha(t) = 0.8$ for $T > 250$.

This function includes also the learning rate $\eta$ as a parameter, and it is updated after every iteration by means of the adaptive learning rate as it is described by Jacobs (1987) where we know it gradually increases the learning rate in directions in which the gradient is stable (Van der Maaten and Hinton, 2008).

The algorithm is optimizing following a trick is called `early-exaggeration`, which uses a real value (factor) to multiply all the points $p_{ij}$ when we start the optimizations. This will produce that almost all the $q_{ij}$ points, which still add up to 1 are too small to model the $p_{ij}$ corresponding. The effect that this produces is that it is going to create widely separated clusters in the low dimension. Due to the empty spaces that is created it is going to be easier to move the points to find a correct distribution. This exaggeration factor can be modified in the original function, so we can fix it to a proper value (Van der Maaten and Hinton, 2008).

Our version of the code has the following parameters:

```
    tsne(X, q = 2, T = 1e3, learning_rate = 100, momentum = 0.5,
         perplexity = 35, cols = 1)
```

To develop our code we have followed the idea `Rtsne` function, so we have set the same parameters. We have explained it deeply in the next chapter.

## 4.1. Applying to synthetic data

### 4.1.1. Gaussian mixtures in $\mathbb{R}^2$

We are going to test both functions with synthetic data, we have prepared several multivariate Gaussian mixtures in different positions with different sizes and variances.

In order to prepare the synthetic data we have used the following function:

```
    data <- mvtnorm::rmvnorm(n = 200, mean = c(-3, -3),
                             sigma= rbind(c(1, 0.2), c(0.2, 1)))
```

We can modify the parameters in order to vary different specifications of the mixtures. Parameter `n` will be the number of points of the mixture, `mean` will define the center of the mixture, `sigma` will represent the covariance matrix.

On the first row we have prepared two Gaussian mixtures with same sizes and different positions. On the second row there are three Gaussian mixtures and the one in the middle has different variance, however, they have the same sizes. For the next row we have four Gaussian mixtures, and they all have different sizes but the same variance. Finally, in the last row, there are four Gaussian mixtures with different variances and sizes. As we can see in the next figure:

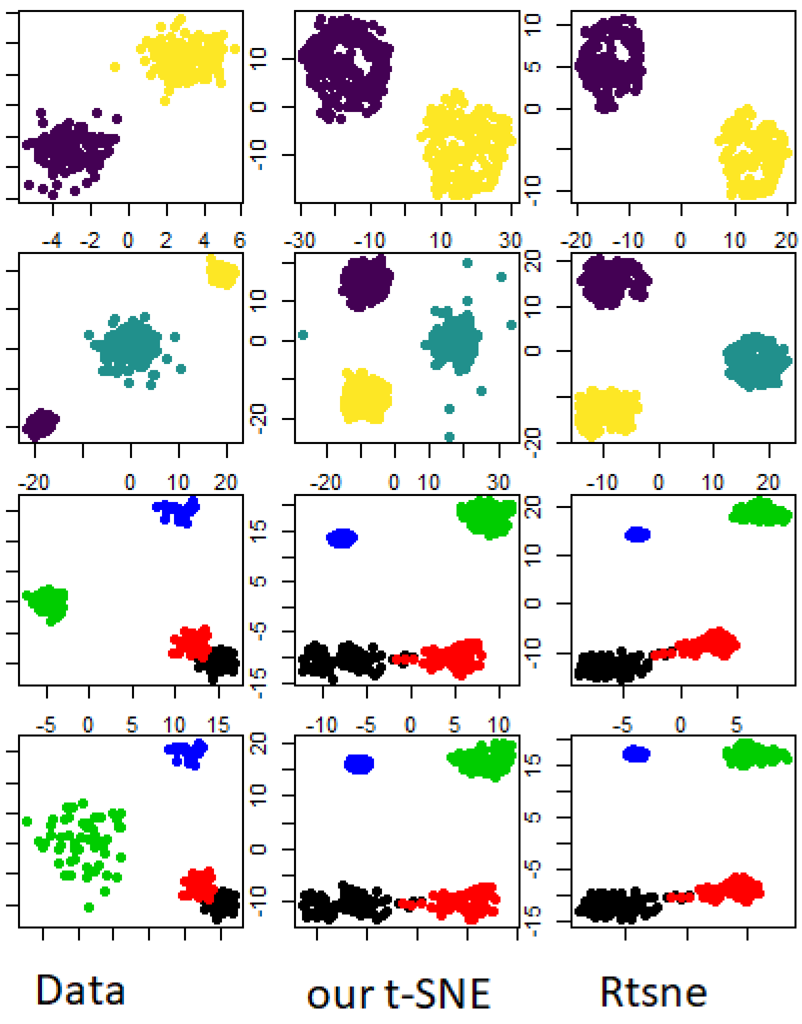


Fig. 4.1. The figure shows the behaviour of two different versions of the algorithm with synthetic data. On the first column we can see the original data plotted, the second column represent a final stage of our implementation of *t*-SNE and the last column shows the final stage of Rtsne package

In this experiment we have used the same values for both functions, we have also used the same low-dimensional initialization for both functions, since `Rtsne` function by Krijthe (2015) has a parameter to modify this value. `perplexity` has been assigned to a value of 35, the dimension we have reduced is 2 in order to see a plane representation we can interpret, `learning rate` ($\eta$) has been tuned to 200. We have decided those values by doing some iterations and we saw that when we used `perplexity` equal to 35, all the results gave a "good global" geometry, which means that the neighbourhood is preserved in all the stages and looks pretty similar. If we used other value we would have obtained other values.

Notice that both algorithms have obtained very similar results, although the distribution is not equal, we can see that the neighbourhood is preserved. The objective function is invariant to rotations and translations hence the output of *t*-SNE may differ due to rotation or translation. Notice that although we have used different Gaussian distributions the size of the clusters may not mean anything, the algorithms is only working to preserve the neighbour distances, but not the size of the clusters. Furthermore, the distances among clusters might not mean anything, what *t*-SNE is worrying about is to preserve the distance, but not to establish a determined distance. When we vary the parameters we can see several differences. We also have to mention that if we added more points to this Gaussian distributions we would have to vary the perplexity in order to compensate this increasing, however, in this experiment we have used the same number of points so we are able to use the same perplexity. Besides, we have noticed that every time we include a new mixture we have to recalculate the algorithm again, we can not reuse the previous calculations.

### 4.1.2. Gaussian mixtures in $\mathbb{R}^P$

In this section, we have decided to go a step beyond by trying the algorithm for Multivariate Gaussian mixtures that are in more than two dimensions.

Firstly, we have tried *t*-SNE on 3 dimension mixtures:

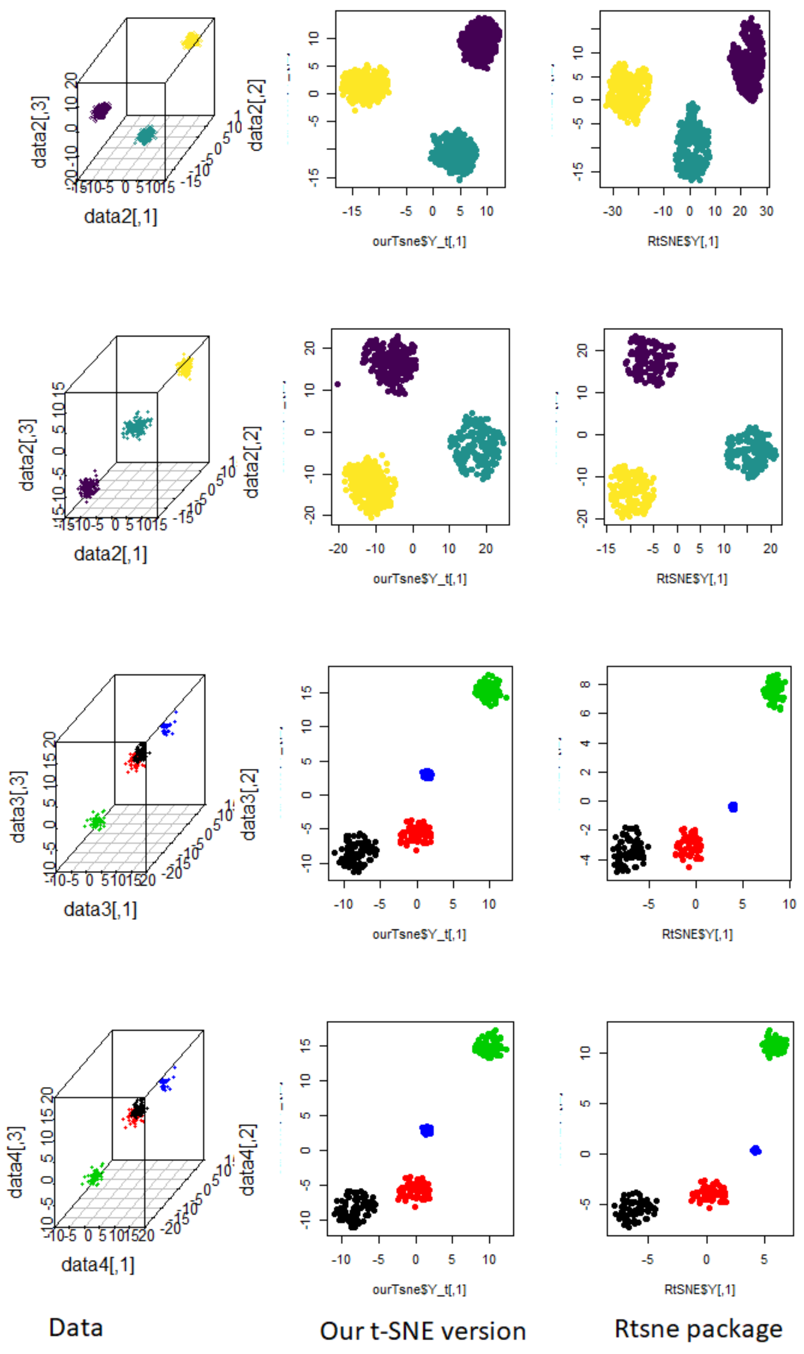


Fig. 4.2. We can see *t*-SNE applied to 3 dimension mixtures with the two different functions. On the first column we have the original data, in the second column we have our version of *t*-SNE applied to the data, whereas in the third column we can see the `Rtsne` version.

In this situation we have applied both versions of the algorithm by finding that both have obtained a very similar result. Notice we have the same behaviour we obtained in the case of two dimensions, we can see that *t*-SNE is preserving the distances of the neighbours. We have used the same parameters in both functions, moreover, we have used the same initialization in both versions.

If we go one step beyond, we have prepared a four dimension Gaussian mixtures in order to see how the algorithm approximate it to three dimensions. Since we do not have a natural way to represent four dimensions, we have used a matrix of scatter plots depending on each variable.

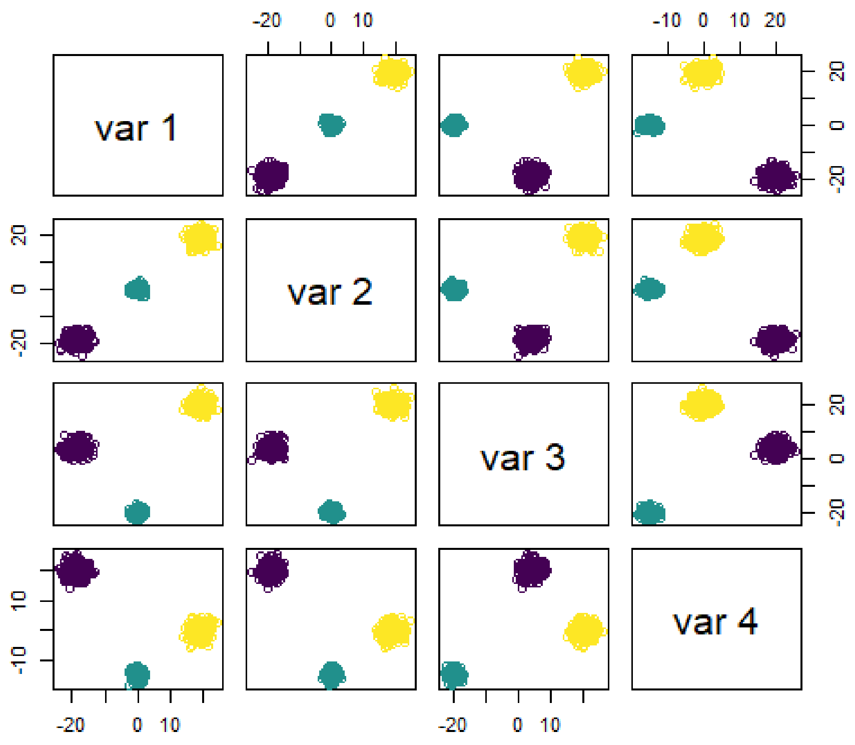


Fig. 4.3. Matrix of scatter plots for three Gaussian mixtures of dimension 4, we can see that yellow and purple mixture have different variances. The matrix of plots help us to visualize the data.

We have reduced the dimension to three in order to obtain a better representation, when we used the matrix of scatter plots we could have an idea of how our data is distributed. However, after applying the algorithm we can see that we have obtained a faithful representation that is more simple to understand to the human.

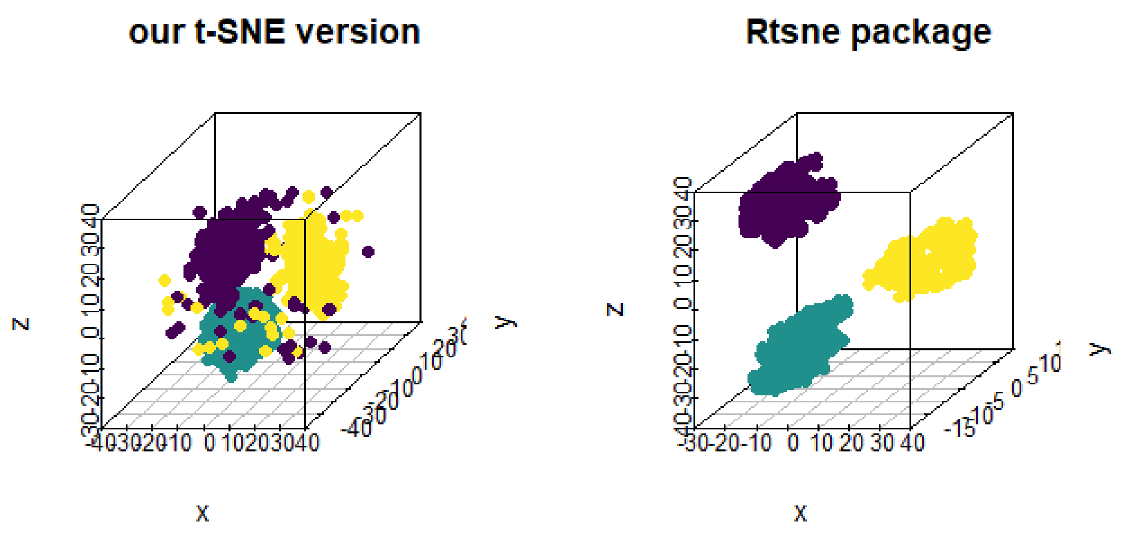


Fig. 4.4. We have applied *t*-SNE applied to 4 dimension Gaussian mixtures in order to reduce dimensions, the left column belongs to the version we have developed and the right one is with `Rtsne` package.

We can see that we have obtained a very similar result when we use the `Rtsne` function and when we use the version we developed, the algorithm has preserved the distance among the datapoints although we have simplified the data.

Notice everything is reproducible, we have uploaded all the code to GitHub so you can do your own tests.

# 5. APPLICATION TO REAL DATASETS

In this part of the project we are going to apply the algorithm to well-known databases like the iris dataset (Fisher, 1936) and MNIST (LeCun and Cortes, 2010). We have chosen real datasets in order to show the application of *t*-SNE in real life and to see how it behaves when we manage a more complex data.

## 5.1. `Iris` dataset

The iris dataset contains 50 samples of three different flowers from the iris Species (setosa, virginica and versicolor). There have been measured four different variables: sepal length, sepal width, petal length and petal width. Basing on the combination we can differentiate these species Fisher (1936).

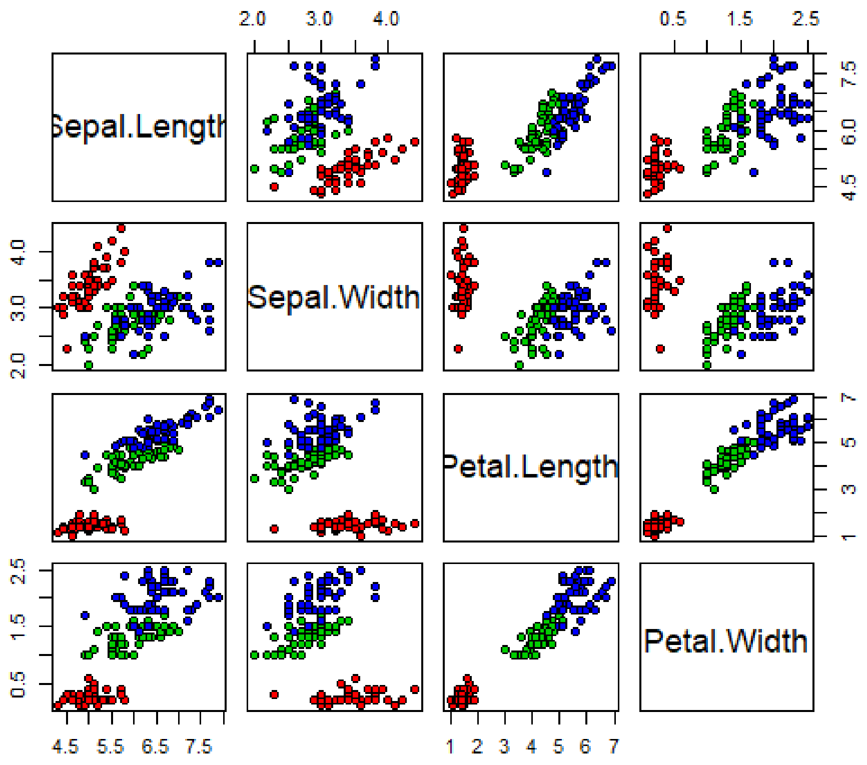


Fig. 5.1. The iris dataset represented in two dimensions depending on different pairs of variables.

By using an R function that is called `Pairs` R Development Core Team (2019), we can plot the dataset, however, as we can see in the figure, since it depends on 4 variables, we have to divide the plots in order to see correctly how variables affect. With the help of *t*-SNE we are obtaining a faithful representation in 2 dimensions.

With the help of the function we have developed in R (R Development Core Team, 2019) we obtain the result in Figure 5.2.

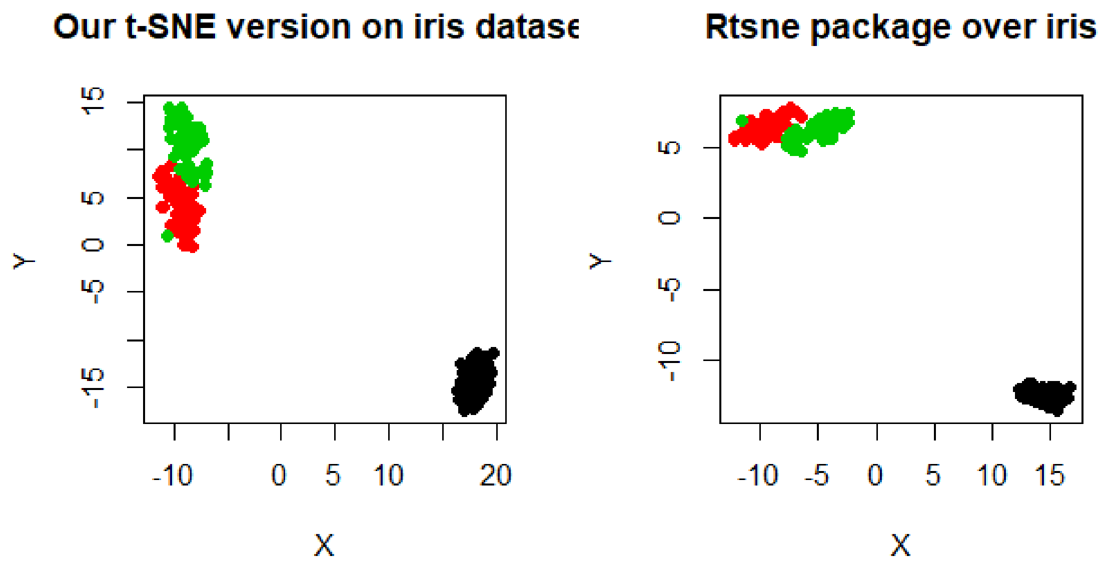


Fig. 5.2. *t*-SNE applied to Iris dataset using our the version we have developed and also the `Rtsne` version.

As we can see in the figures, *t*-SNE has separated the three kinds of flowers preserving the neighbourhood, if we compare our version of the algorithm to the `Rtsne` function by Krijthe (2015), we can see that the behaviour is mostly equal. In both figures we can see how there are two different species that are more similar: virginica and versicolor, so their clusters are closer and there is another kind of iris flower more separated: setosa.

Recall we have used the same values for the parameters in both functions, also to be more optimal we have used the same initialization in both versions.

If we compare the behaviour of *t*-SNE applied to the iris dataset Fisher (1936) in three dimension, the result we obtain is the one as follows.

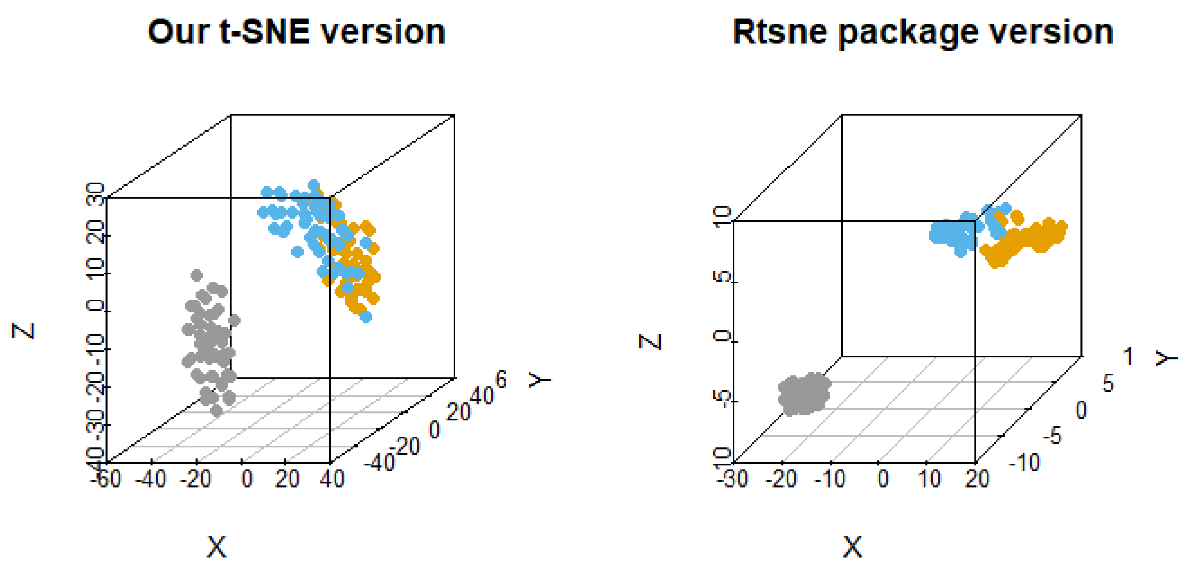


Fig. 5.3. *t*-SNE applied to Iris in three dimensions where the left column is the result of applying the version we developed and the right column belongs to the `Rtsne` version.

Watching the results, *t*-SNE can help us to choose among the different type of flowers. Both implementations have separated the different species easily. We have obtained a good representation of the data, reducing the dimensions we originally had and preserving the distance of the nearest neighbours.

## 5.2. MNIST dataset

MNIST is a very well-known dataset that contains images of handwritten digits in different ways. It is very used nowadays with classification algorithms. In order to see how powerful is *t*-SNE we are going to apply the algorithm to this dataset by LeCun and Cortes (2010).

Recall that in Figure 2.6 we saw what happened to this database when we applied PCA on it. It did not manage very well to identify and separate the different numbers.

When we use *t*-SNE we can see a huge improvement in the clustering over PCA. We can see clearly how the numbers are separated in different colours.

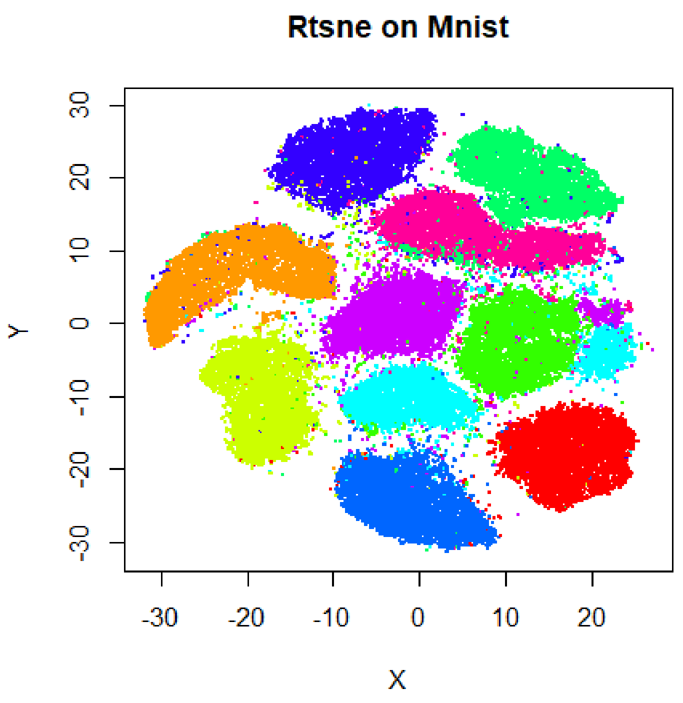


Fig. 5.4. We can see a final stage of *t*-SNE applied to MNIST dataset, we can see the clearly differentiated numbers in different colours.

We have applied firstly PCA over the MNIST dataset and later we have applied again *t*-SNE, although the computational time has been extremely large, we have obtained a more defined clustering than when we used directly *t*-SNE.

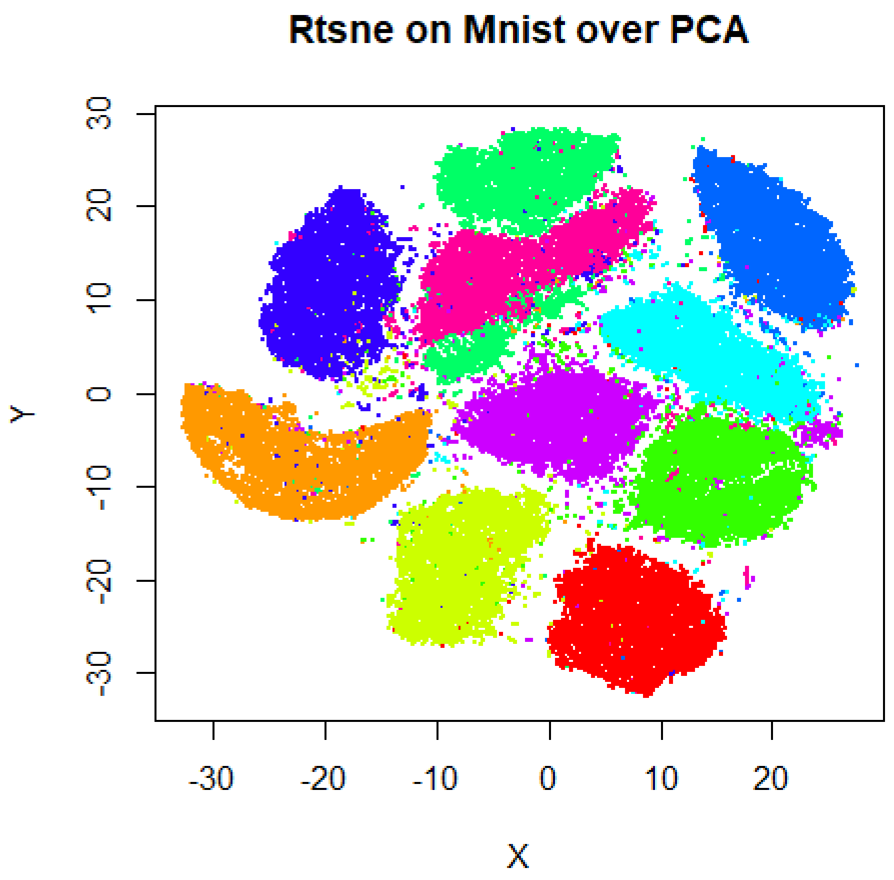


Fig. 5.5. Result of applying *t*-SNE over MNIST dataset but firstly applying PCA on it.

We can see that these techniques do not compete against, but they can be used together in order to obtain better results.

As we explained in the previous chapters, *t*-SNE is based on the concept of similarity, which means the distance among the points. As we saw in the paper, the similarity between $x_i$ and $x_j$ is the conditional probability that $x_i$ chooses $x_j$ as it neighbour when neighbours are chosen proportionally to its density under the Gaussian distribution curve centered in the point $x_i$. Therefore, *t*-SNE has chosen as neighbours digits that are more similar. However, we have to remember that *t*-SNE is taking the high dimensional points to the low dimensional space initializing on a random way. Then, different runs of the algorithm should lead to different results.

Moreover, since the algorithm runs in a random initialization, one interesting fact that we have to mention is that to choose the correct output we have followed the same procedure that it is done in K-Means. It consist on running several times the algorithm with different values and keeping the values that produce the minor cost.

## 5.3. Glass identification dataset

This dataset is known as "glass identification dataset" (German, 1987). It suggest that we are situated in a scene where a murder have occurred. We have different kind of glasses and we have to correctly identify them in order to use it as an evidence (Dua and Graff, 2017).

Analysing the different characteristics of each glass and thanks to *t*-SNE, we can separate in groups the different kind of glass that we have found. The dataset contains a total of 214 samples of glass with a total of 10 attributes each. These attributes are chemical components such as magnesium, aluminium, sodium silicon, iron among others. Depending on the quantity of each chemical there is produced one type of glass or another.

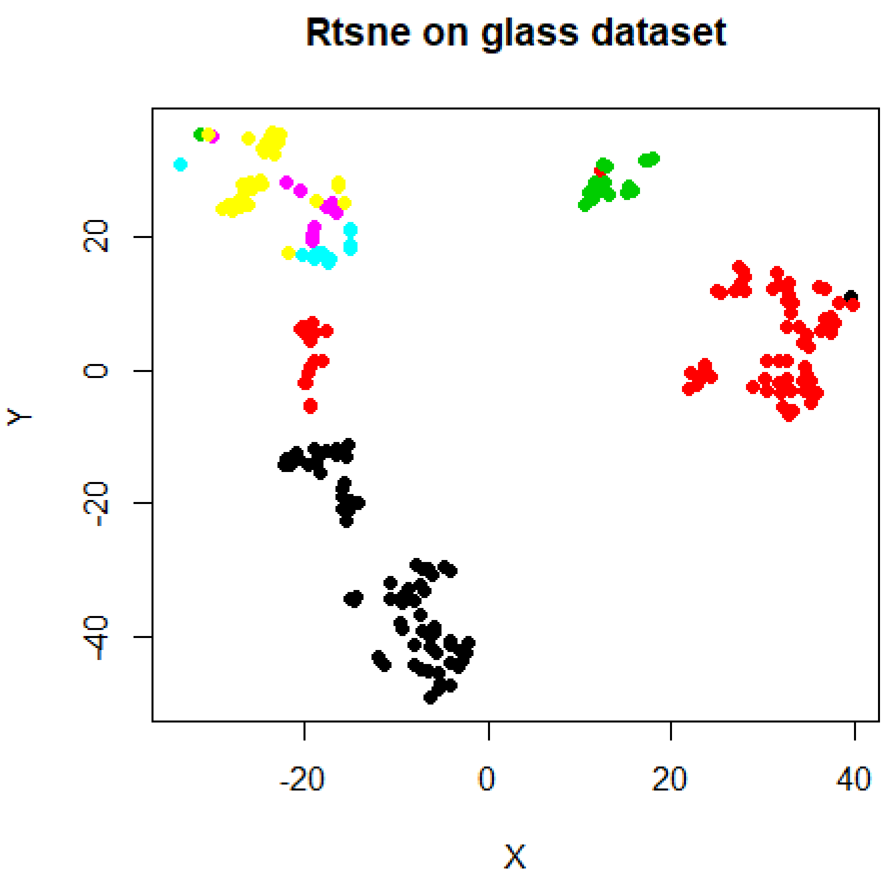


Fig. 5.6. *t*-SNE applied to the glass identification dataset where we can see that each cluster belong to a different kind of glass in function of its characteristics.

As we can see above, *t*-SNE has managed to separate the different kind of glass using the composition of each sample. Depending on the amount of chemical components it will belong to one type or another. Therefore, we can extract as a conclusion from the graphs that there are some kind of glass that are more similar than the others. The yellow, purple and cyan present characteristics resemblance to them, whereas the other types are separated.

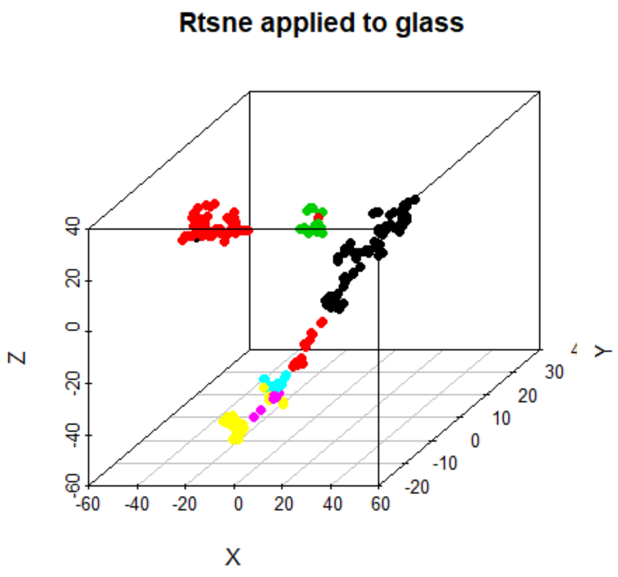


Fig. 5.7. `Rtsne` applied to the glass dataset reducing the dimension to three. We can see different clusters depending on the type of glass.

Although the visualization is good enough when we reduce to two dimensions, when we reduce the dimension of the data into three dimensions we can see how *t*-SNE has managed to create different clusters with the different type of glass. Thanks to this algorithm we can reduce the dimension in order to identify different types that in other situations would be really difficult.

## 5.4. Wines dataset

We have applied *t*-SNE on a wines dataset (Forina, 1991). This wine dataset is composed by the result of the three different wines that belong to the same region in Italy. These wines are composed by 13 different chemicals that constitute each individual wine. The dataset is composed by 178 samples of 13 attributes.

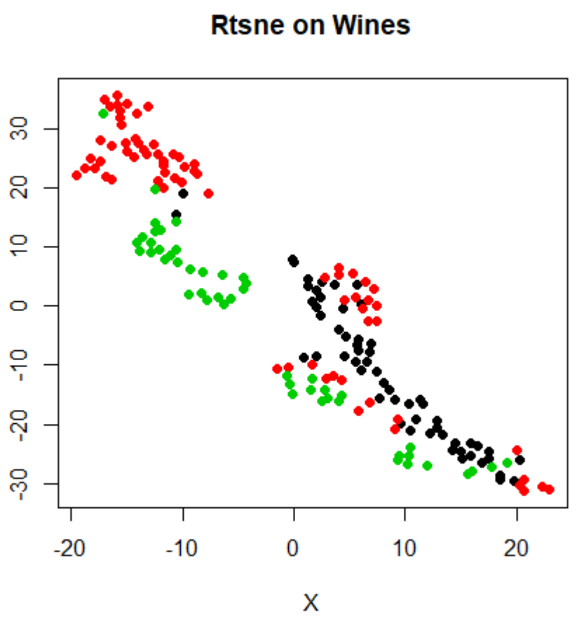


Fig. 5.8. `Rtsne` applied on the wines dataset where we can see the three different kind of wines depending on its characteristics.

If set the algorithm to represent the data in three dimensions:

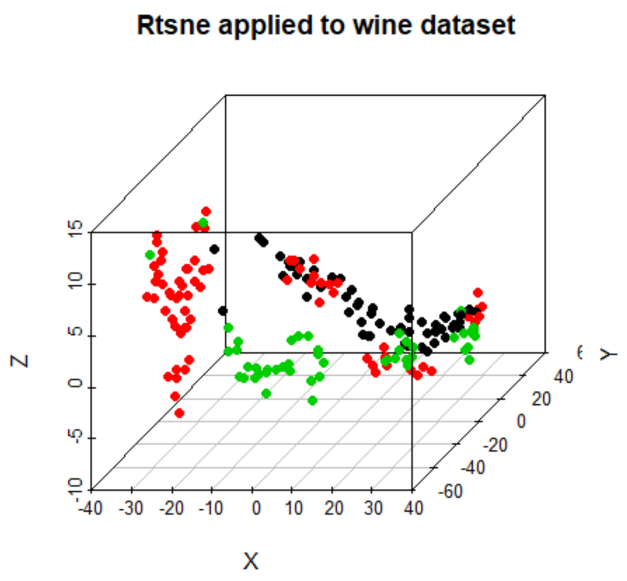


Fig. 5.9. We can see the different types of wine separated in three different groups depending on its characteristics.

When we want to have an idea of how our data is distributed, we can not do a simple plot. Since each sample of this dataset is depending on 13 variables, we would need a 13 dimension figure in order to represent it. We could put in practice some well-known techniques such as using a pairs plot, however it is going to be very difficult since it depends on a huge quantity of variables. This dimension-reduction technique allows us to have a faithful visualization of the original data,although as a consequence it loses some information.

When we apply the algorithm, as we can see in the plots, *t*-SNE has managed to correctly identify each type of wine depending on the characteristics similarities. We can consider it as a powerful tool that using just a few characteristics of the wine is able to identify its similarities and create clusters with the different existing types. We can conclude that the red and green color wines have more similar characteristics than the black color wine, due to its proximity in Figure 5.8.

## 5.5. Echocardiogram dataset

In this section of the project, we have decided to combine our technical knowledge with the health field. We have analysed a dataset that contains data of patients that have suffered from a heart attack in the past year (Kinney, 1989). Some of them are alive, but others are not. The main goal of this dataset is to classify the patients whether if they are going to survive for at least one year or not depending on different attributes. This dataset contains 132 sample of different people and 12 different attributes. The dataset contains attributes such as the month the person has survived until his death, if he is still alive or not, the age he suffered from the heart attack among others.

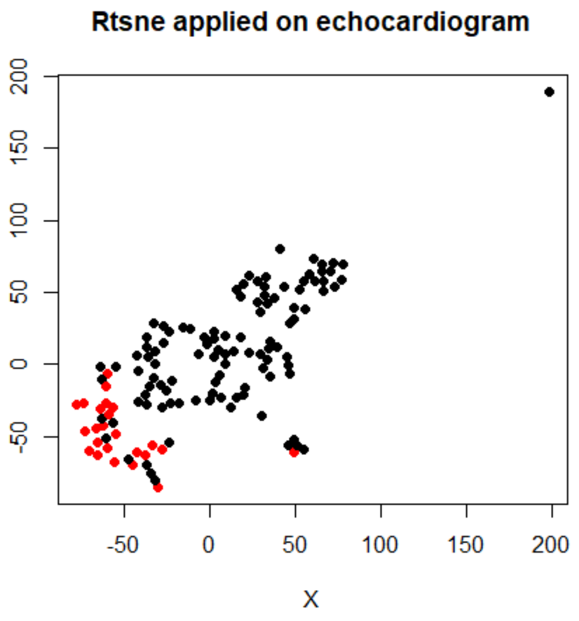


Fig. 5.10. We can see two different groups of people that have suffered from a heart attack in the last year separated in function if they are alive or not one year later.

We can also see how *t*-SNE has reduced the dimension of the original data in three dimensions:

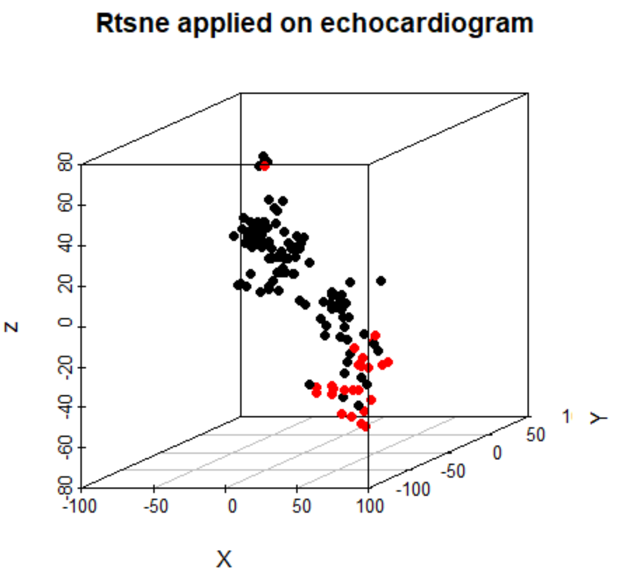


Fig. 5.11. We have applied *t*-SNE to reduce the dimension on the data onto three dimension to have a different view of the data. We can see that `Rtsne` has managed to differentiate the clusters to identify both groups.

As we can see in the picture, *t*-SNE has differentiated correctly whether if the person has survived for at least one year or not. We can identify that red points represent the people that have survived whereas black points represent the people who have not. If we add data of another person who has suffered from a heart attack, we can predict if it is going to survive or not if he belongs to the red group or the black group. Depending on its characteristics, *t*-SNE is going to recalculate the clusters by grouping in this two different groups. If we try to have an idea of our data at first, we find that is nearly impossible to extract relations or conclusions because of the multiple dimensions. Once we have applied the algorithm, we find that it is easier to create patterns of the data. We can see how *t*-SNE has found a relation among the people who has survived for at least one year and those who has not by creating different clusters.

# 6. REGULATORY FRAMEWORK

In this part of the project, we are going to explain how the different laws are applied to our work. Since we have used different software in order to develop our work, we have been affected by these. The main law we identified at first was related to the usage of the software.

## 6.1. Software

Firstly, the programming language we selected to develop this code was R. It is an open source code and a package of GNU, which is a recursive acronym of GNU is not Unix, GNU is commonly used with a kernel Linux. It is distributed freely under the license GNU GPL, which means "General Public License" (R Development Core Team, 2019).

This license is widely used nowadays when we use software open source. The main advantage of this software is that we can freely study, share and modify the software we need for the development of our project. This has been really useful to us when we needed to develop our application. By using these open source functions, for example, the one to read images, we have been able to develop ours in order to use it as the way we desired. On the other hand, the original function of $t$-SNE, `Rtsne` package, (Krijthe, 2015) is also free to use, this has helped us a lot in order to do our experiments, allowing to us to compare with the function we developed.

Open source license protects the rights of the developers, it is going to offer the user the decision to copy, distribute and modify the software.

Intellectual property law is going to give these rights we have just explained simply for the fact of creating the code. It is going to assign the author the full disposition and the exclusive right to explode the software only limited by the law Intellectual Property no 28 (1995).

Therefore, this law does not guarantee that the software is fully protected. However, if the software is modified and redistributed, it is ensuring that the future users know it is not a original copy.

### 6.1.1. R environment

The environment we have used to develop the code is *RStudio* (RStudio Team, 2019) it is distributed under the Affero GNU GPL v3. The Affero GNU is an open source license that is going to guarantee that the user can freely share and distribute the software and it ensures that the software is free for the user who want to employ this software. The company offer two open source versions, these are RStudio Desktop and RStudio Server.

As its names define them, we can download a version to work in our computers locally and other version to work within the servers they provide. Besides, the company has other paid versions which could give us support and other tools, however, we have not find them interesting to this project. These versions works both in MacOS and in Microsoft Windows. As we specified before, the packages we have used in this project are also open source.

### 6.1.2. Overleaf and TexStudio for LaTeX

To write the final project we have used the scientific text editor LaTeX (LaTeX, 1984). We selected this programming language due to its stability its design and it is really useful for science since it has an elegant way to display equations. It is going to adjust the sizes of the parenthesis, integrals, subindex and superindex. This software is an open source code distributed under the license LPPL *LaTeX Project Public License* (LaTeX, 1984).

In order to write the latex code we have used two different platforms:

- TexStudio under the GPL license (TexStudio, 2019).
- Overleaf under personal license (Hammersley, 2019).

We have used these two different tools in order to write this project. At home, we have used Tex Studio software due to its robustness and efficient work. When we needed to add a minor change while we were not at home we have used the online tool Overleaf. The license we have used in this project is free for one person, however, if we need up to ten developers we can pay for the collaborative version.

In order to obtain a good bibliography, we have used the famous application JabRef (JabRef, 2018), which is distributed under the MIT license. The application is open source and can be downloaded directly from its webpage. It is programmed in Java and it is available for Windows, Mac or Linux. Thanks to this, we simply manage the standard LaTeX bibliography format: BibTeX.

## 6.2. Data set

When we use other people's data, we need to ensure that we comply the law GDPR (General Data Protection Regulation, 2016) of the European Union. Thanks to this law we have control above our personal data. We have three basic rights:

1. **Right of access**: Physical people can demand to an organization why are they storing his personal data, where and what for.
2. **Right to forgetfulness**: An individual can demand to delete his personal data if they are no longer necessary.

3. **Right to migrate data**: A person can demand his data to a company in a format he can migrate to other company.

This law is protecting the personal data so we can not “attack” to the honor nor personal and family privacy. In this project we are using real datasets. In order to comply with the law, we are not broadcasting any information about the information we have used for the study, only the different relations among them we have found.

# 7. PLANNING

The development of this project is going to take from November to July, so we can present our work the first of July, a total of nine months. We have prepared a Gant Diagram where we are going to show how are we developing this project. These are the activities that compose the whole project:

- Establish objectives: We prepared an initial planing to fix our objectives.[1 month]
- Analyse Regulatory framework: We analysed the regulatory framework in order to see we comply with the law in our studies [1 week].
- Cost plan: We have analysed how much would cost our project to see if it is profitable [1 week].
- Get familiar with the software we use in the project: R and LaTeX [1 month].
- Analyse state of the art: We need to study how is the problem actually [2 months].
- Deepen in the *t*-SNE: Perform a complete study in detail of the algorithm how it works, how it behaves in different situations [3 months].
- Do some experiments: We have used synthetic data to see it behaviour, and other simple datasets [2 months].
- Prepare the application to real dataset: We have used a real dataset in order to see how the algorithm works with real data [2 months].
- Analyse results: Obtain conclusions of how the algorithm has worked with the data [1 month].
- Prepare final memory: Once we have finished the study, we have to prepare a document with the whole development of the project [2 months].
- Prepare a presentation of the project [3 weeks].

The planning of this project is estimated, however it could suffer some changes during the development. We have prepared a continence plan, leaving a free final week for each activity just in case we have some delay.

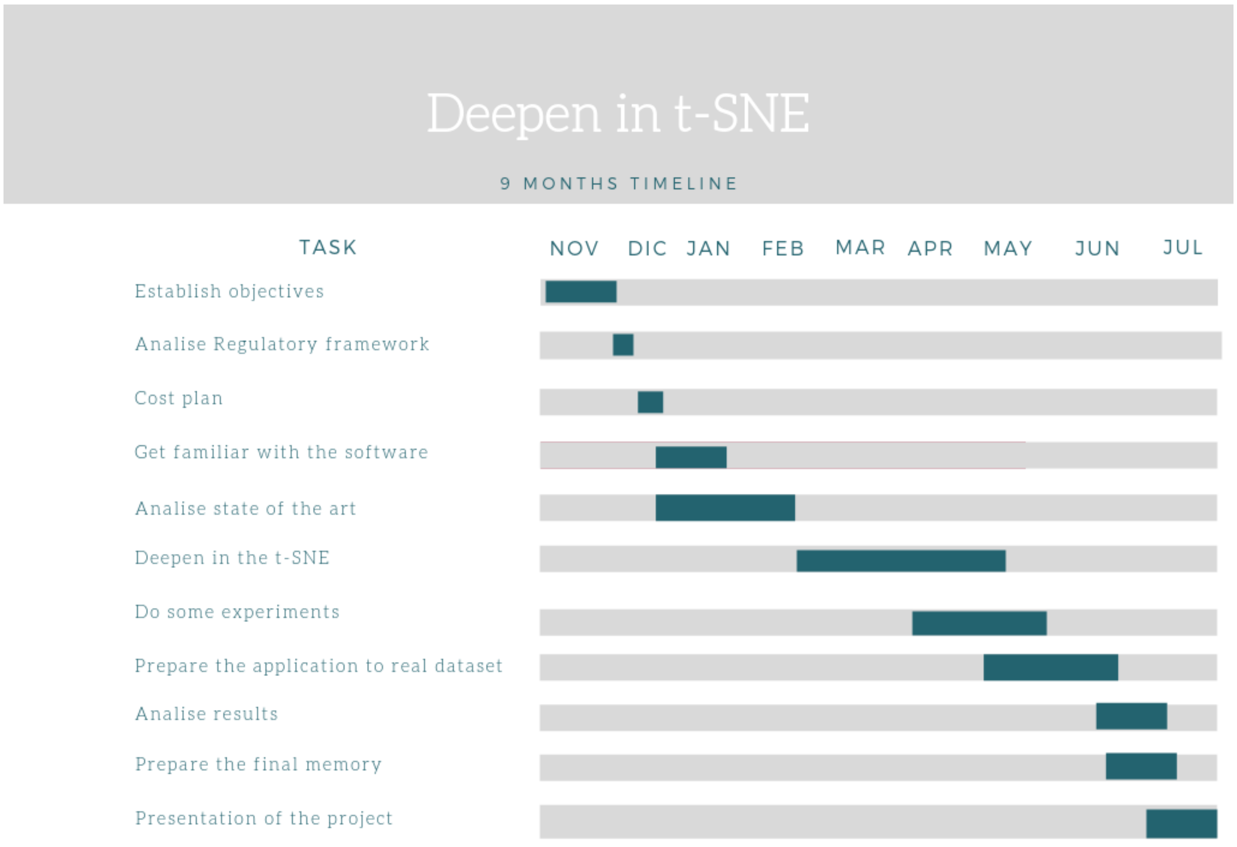


Fig. 7.1. Gantt Diagram that shows the development of our project. It shows the estimated time every activity is going to take.

In order to calculate the total hours we are going to invest in this project, we are assuming we are going to work a total of 1 hour in average per day, seven days a week for 4 weeks a month, which is a total of 28 hours per month, in nine months is a total of 252 hours.

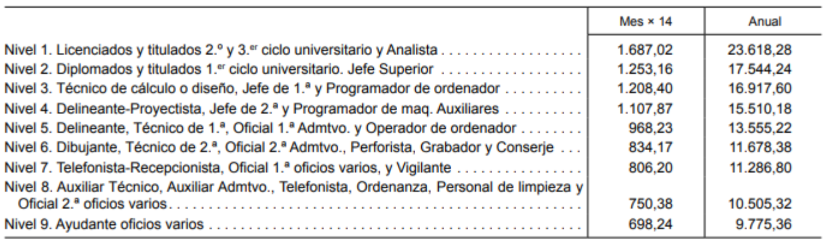
# 8. ECONOMICAL BUDGET

In this chapter of the project, we have calculated a budget to show how much would cost to hire our service. In this cost we have explained in detail a price breakdown. We have different kind of costs.

## 8.1. Personnel costs

Taking into account the number of hours we are going to dedicate to this project, that we previously specified in the previous chapter we are going to calculate how much does the personnel cost.

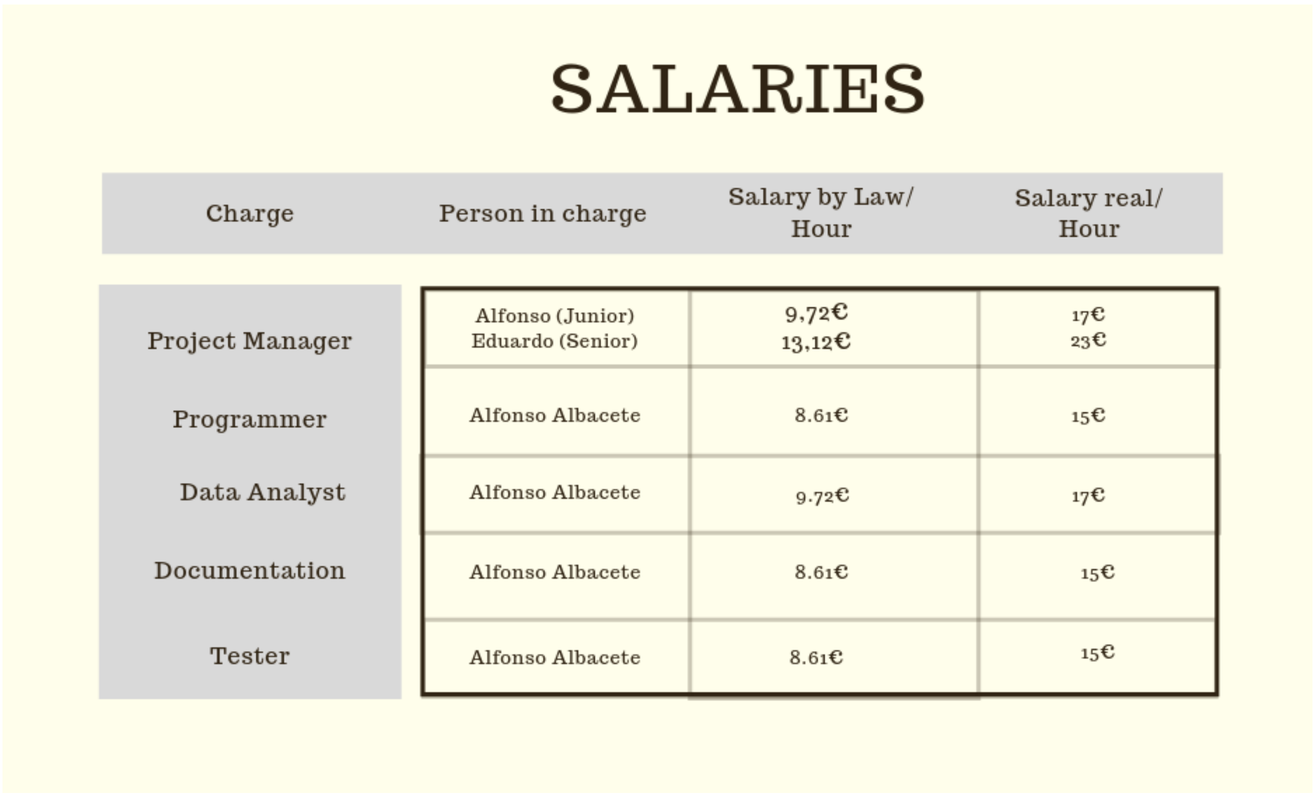
We assume that this work is under the *Disposición 542 del BOE núm. 15 de 2017* which regulates the minimum salary range for engineering companies Disposición 542 del BOE núm. 15 (2017) .



| | Mes × 14 | Anual |
|---|---|---|
| Nivel 1. Licenciados y titulados 2.º y 3.er ciclo universitario y Analista | 1.687,02 | 23.618,28 |
| Nivel 2. Diplomados y titulados 1.er ciclo universitario. Jefe Superior | 1.253,16 | 17.544,24 |
| Nivel 3. Técnico de cálculo o diseño, Jefe de 1.ª y Programador de ordenador | 1.208,40 | 16.917,60 |
| Nivel 4. Delineante-Proyectista, Jefe de 2.ª y Programador de maq. Auxiliares | 1.107,87 | 15.510,18 |
| Nivel 5. Delineante, Técnico de 1.ª, Oficial 1.ª Admtvo. y Operador de ordenador | 968,23 | 13.555,22 |
| Nivel 6. Dibujante, Técnico de 2.ª, Oficial 2.ª Admtvo., Perforista, Grabador y Conserje | 834,17 | 11.678,38 |
| Nivel 7. Telefonista-Recepcionista, Oficial 1.ª oficios varios, y Vigilante | 806,20 | 11.286,80 |
| Nivel 8. Auxiliar Técnico, Auxiliar Admtvo., Telefonista, Ordenanza, Personal de limpieza y Oficial 2.ª oficios varios | 750,38 | 10.505,32 |
| Nivel 9. Ayudante oficios varios | 698,24 | 9.775,36 |

Fig. 8.1. Capture of Disposición 542 del BOE núm. 15 (2017) where shows the minimum salary depending on the charge.

In this Article, it is established that the maximum quantity of hours that an employee can work is 1.800 hours. We can calculate how much do we have to pay for hour. In this project there are different work position that will have different costs. We can see in the table as follows:

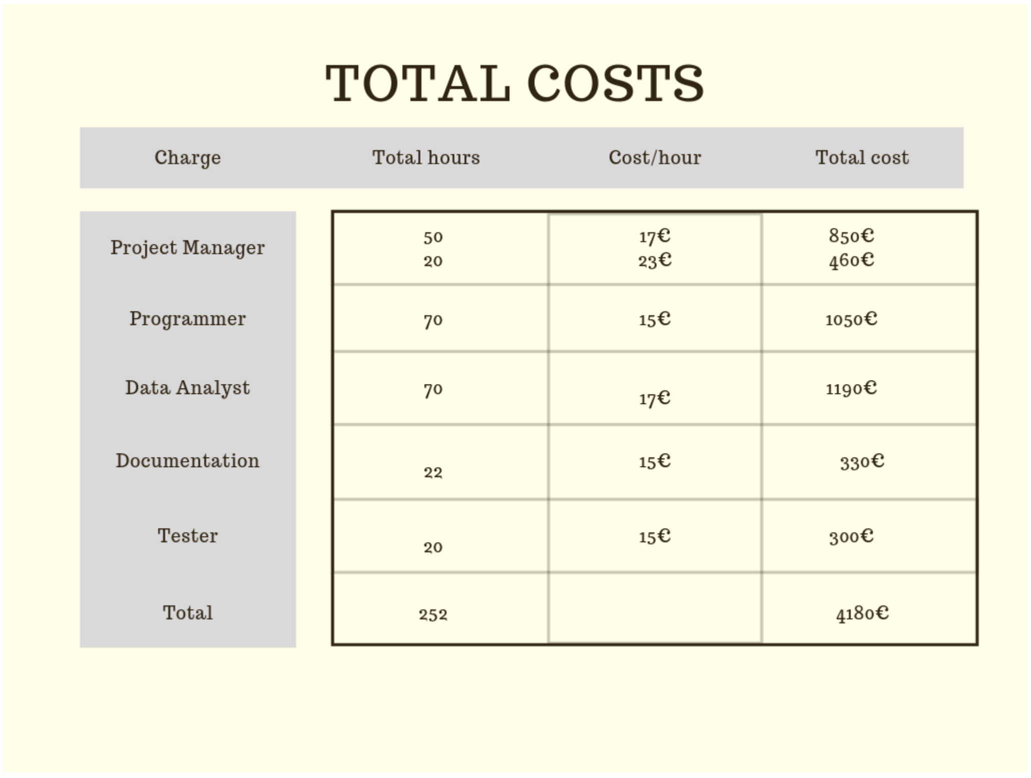


## SALARIES

| Charge | Person in charge | Salary by Law/ Hour | Salary real/ Hour |
|---|---|---|---|
| Project Manager | Alfonso (Junior)<br>Eduardo (Senior) | 9,72€<br>13,12€ | 17€<br>23€ |
| Programmer | Alfonso Albacete | 8.61€ | 15€ |
| Data Analyst | Alfonso Albacete | 9.72€ | 17€ |
| Documentation | Alfonso Albacete | 8.61€ | 15€ |
| Tester | Alfonso Albacete | 8.61€ | 15€ |

Fig. 8.2. Table that shows the salaries of the different charges we are taking in this project.

We have decided to increase the salary compared to BOE’s recommendation because we thought it was too low. In this cost/hour we have included the legal taxes and the social insurance.



TOTAL COSTS

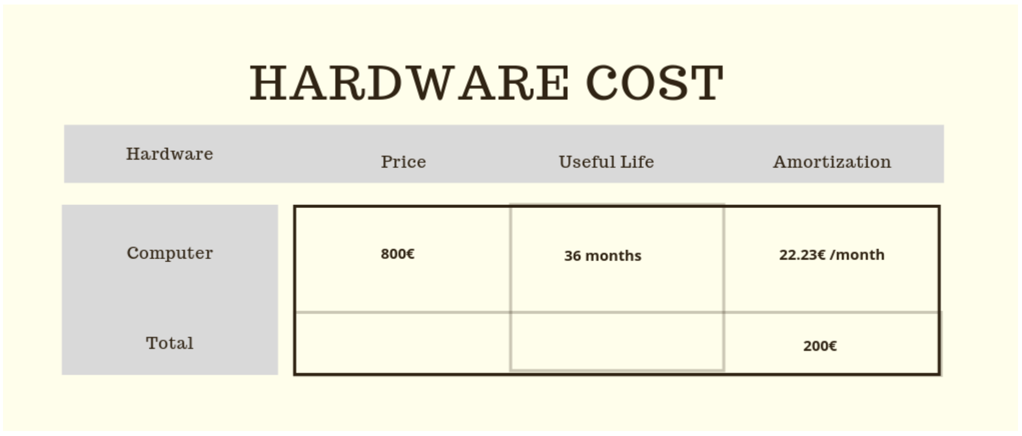
| Charge | Total hours | Cost/hour | Total cost |
|---|---|---|---|
| Project Manager | 50<br>20 | 17€<br>23€ | 850€<br>460€ |
| Programmer | 70 | 15€ | 1050€ |
| Data Analyst | 70 | 17€ | 1190€ |
| Documentation | 22 | 15€ | 330€ |
| Tester | 20 | 15€ | 300€ |
| Total | 252 | | 4180€ |

Fig. 8.3. Table that represent the total cost of the personnel taking into account the different hours we have worked.

In this figure we show the estimated number of hours that we are going to work depending on the charge in the project. We have calculated the total cost by charge by using the estimated number of hours and the fixed cost/hour. We can see that the total personnel cost is 4.180 €.

## 8.2. Hardware and software costs

In order to determine the material resources we have to take into account the amortization of the computers and the software used. We are going to divide the total price by the useful estimated life and we are going to multiply by the months that the project last. In terms of software, since we are using open-source programs the cost is going to be minimum. The hardware cost will be an amount of 200 €.

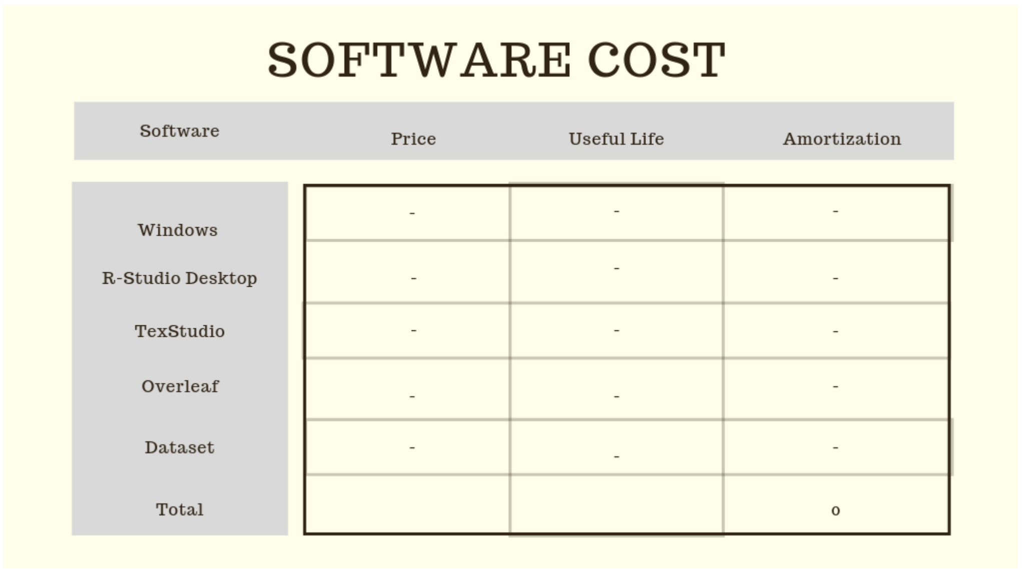


HARDWARE COST

| Hardware | Price | Useful Life | Amortization |
|---|---|---|---|
| Computer | 800€ | 36 months | 22.23€ /month |
| Total | | | 200€ |

Fig. 8.4. Table that represents the hardware cost we have acquired for the development of the project.

In this figure we can see a price breakdown of the software cost, as we mentioned before, it is going to be zero due to the fact that we are using open-source programs. On the other hand, the windows license is coming included with the computer.



SOFTWARE COST

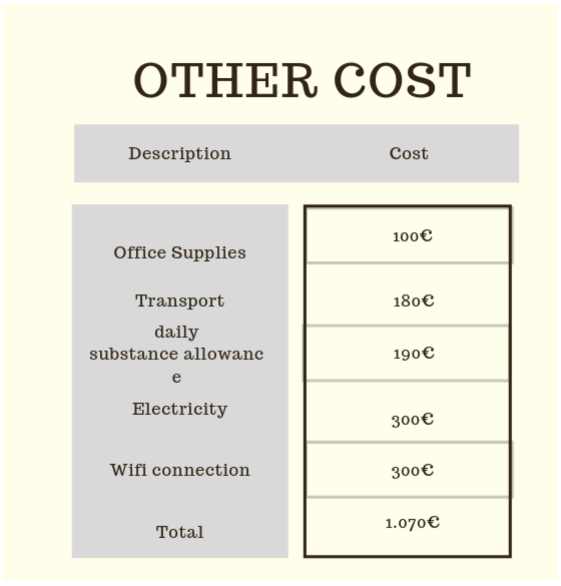
| Software | Price | Useful Life | Amortization |
|---|---|---|---|
| Windows | - | - | - |
| R-Studio Desktop | - | - | - |
| TexStudio | - | - | - |
| Overleaf | - | - | - |
| Dataset | - | - | - |
| Total | | | 0 |

Fig. 8.5. Table that shows the software cost for this project, as we can see it is zero cost since we have used open-source software.

## 8.3. Other costs

In this section we have included other costs such as office supplies, pen drives, paper, printings and bindings, electricity, WiFi connection, a daily substance allowance and transports for the meetings.

Since we have done the project at home and at the university we have reduced the cost of renting an office.



OTHER COST

| Description | Cost |
|---|---|
| Office Supplies | 100€ |
| Transport | 180€ |
| daily substance allowance | 190€ |
| Electricity | 300€ |
| Wifi connection | 300€ |
| Total | 1.070€ |

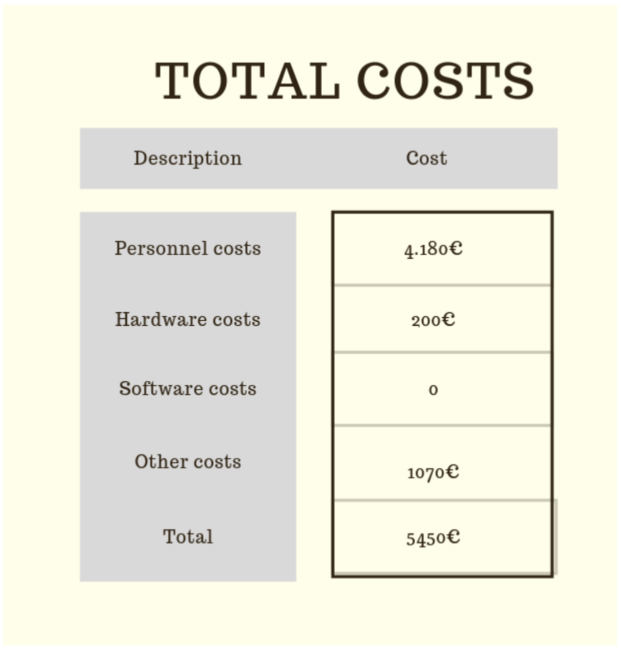
Fig. 8.6. Othercosts we have to take into account for the development of this project.

In this table we have included also the transports cost and the daily substance allowance. Although most of the work is done on remote, it is necessary to attend some project manager meetings in order to take decisions and to explain how is the development of the project going.

Moreover, we have included all the cost associated to the place of work such as WiFi connection and electricity consumed.

## 8.4. Total costs

Once we have calculated all the partial costs we are going to show the total cost and include how much would it cost including the legal taxes. The total cost raise to an amount of 5.540 €.

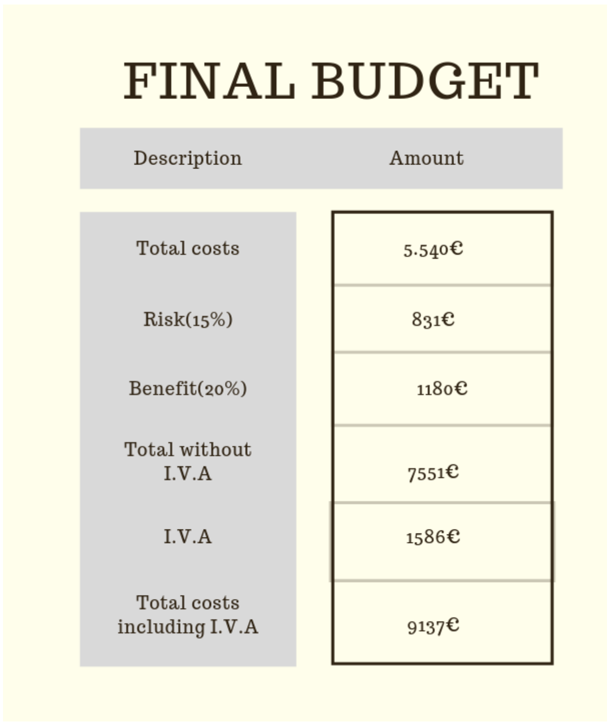


TOTAL COSTS

| Description | Cost |
|---|---|
| Personnel costs | 4.180€ |
| Hardware costs | 200€ |
| Software costs | 0 |
| Other costs | 1070€ |
| Total | 5450€ |

Fig. 8.7. Table that shows the total project cost including all the costs we have previously calculated.

Assuming that the total cost is 5.540 €, we estimate this would be the final budget for the development of this project. We have include a small amount that we will use for unexpected events. Besides, we have included how much are we going to earn with the development of this project.



FINAL BUDGET

| Description | Amount |
|---|---|
| Total costs | 5.540€ |
| Risk(15%) | 831€ |
| Benefit(20%) | 1180€ |
| Total without I.V.A | 7551€ |
| I.V.A | 1586€ |
| Total costs including I.V.A | 9137€ |

Fig. 8.8. Table that presents the final budget for this project, including the breakdown of taxes we must pay.

**The final budget for this project is 9.137 €**.

**Nine thousand one hundred thirty seven euro**.

Signed by

Alfonso Albacete Zapata

# 9. CONCLUSIONS

In this project we have reviewed *t*-SNE technique in depth, achieving our main goal of being able to implement our version of the algorithm, joining theory and practice. We have done lots of experiments that have helped us to know how *t*-SNE works. We have understood why it has been very notorious in the machine learning community. We have seen that it is very flexible and manage to preserve structures where other algorithms that were used for the same purpose did not work as well. It has become the direct competitor of classical dimension-reduction techniques. Studying this algorithm we have seen that the method is applicable in a wide range of fields.

Unfortunately, it has some negative aspects, its flexibility make that sometimes it is hard to interpret the data. Perplexity is a tuneable parameter that will give this flexibility, however, there is no fully objective guide in order to choose a value or other. The only option we have is to try new values and check which value produces the minimum cost. Besides, when we have a huge dataset the computational cost is really large, if we have to try several values of perplexity it can take a long time in order to determine the optimal value. We have performed some experiments with the MNIST database, seeing that it took more than 20 minutes to obtain an optimal solution. Other negative aspect is that if we computed *t*-SNE for a huge dataset but we decided to insert a new sample, *t*-SNE needs to recalculate everything again. It can not reuse the previous calculation easily, which would speed it up. The fact that it is based on randomness is not very efficient, since if we run the algorithm two times in a row we are going to obtain different results, it will preserve the original structure but in a different way, which could make us difficult to interpret the results.

One interesting experiment that we would have liked to implement if we had more time is to use *t*-SNE as a clustering algorithm and later use it as an input to a classification method such as a Support Vector Machine (SVM) to see if the joining of these two techniques lead to better results. Recalling that when we have more dimensions, it is easier to separate our data, besides, there are techniques that allow us to create new "artificial" dimensions to see if we can separate better our data. Otherwise, *t*-SNE is trying to do the contrary and doing so it gives an important characteristic we have mentioned the whole project and it is visualization. The more dimensions you have the more difficult is to obtain a way to visualize the data. *t*-SNE is going to help on visualizing the classification procedure by providing a faithful visualization so we can have an idea of how is our data and extract conclusions from it.

Along this project we have performed several tests using the algorithm, we have uploaded the code in an open repository to GitHub.

# BIBLIOGRAPHY

Barnes, J. and Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449.

Datascience, E. (2017). R basics: PCA with R. `http://enhancedatascience.com/2017/05/07/r-basics-pca-r/`.

Disposición 542 del BOE núm. 15 (2017). Agencia Estatal Boletin Oficial del Estado. `https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf`.

Dua, D. and Graff, C. (2017). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Forina (1991). Wine database. *U.C. Irvine Machine Learning Repository*. `http://archive.ics.uci.edu/ml/datasets/Wine`.

Gao, B. and Pavel, L. (2017). On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv:1704.00805*.

General Data Protection Regulation (2016). European Parlament. `https://www.boe.es/doue/2016/119/L00001-00088.pdf`.

German, B. (1987). Glass identification database. *U.C. Irvine Machine Learning Repository*. `http://archive.ics.uci.edu/ml/datasets/Glass+Identification`.

Hammersley, J. (2019). Overleaf, LaTeX online editor. `https://es.overleaf.com/`.

Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbour embedding. *Advances in Neural Information Processing Systems 15*, pages 857–864.

Hohenwarter, M., Borcherds, M., Ancsin, G., Bencze, B., Blossier, M., Éliás, J., Frank, K., Gál, L., Hofstätter, A., Jordan, F., Konečný, Z., Kovács, Z., Lettner, E., Lizelfelner, S., Parisse, B., Solyom-Gecse, C., Stadlbauer, C., and Tomaschko, M. (2018). GeoGebra 5.0.507.0. `http://www.geogebra.org`.

Intellectual Property no 28 (1995). Agencia Estatal Boletin Oficial del Estado. `https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930`.

JabRef (2018). JabRef version: 4.3.1 citation for LaTeX. `http://www.jabref.org/`.

Jacobs, R. A. (1987). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, New York.

Kinney, D. E. (1989). Echocardiogram dataset. *U.C. Irvine Machine Learning Repository*. `http://archive.ics.uci.edu/ml/datasets/Echocardiogram`.

Krijthe, J. H. (2015). *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation*. R package version 0.15.

LaTeX (1984). LaTeX a document preparation system. `https://www.latex-project.org/`.

Laurens Van der Maaten (2008). t-SNE GitHub repository. `https://lvdmaaten.github.io/tsne/`.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`.

Lejon (2018). lejon/t-SNE-java. `https://github.com/lejon/T-SNE-Java`.

MATLAB (2010). *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts.

Miao, Y. (2015). High dimensional data visualizing using t-SNE. *High Dimensional Data Visualizing using t-SNE*. f `http://yinsenm.github.io/2015/01/01/High-Dimensional-Data-Visualizing-using-tSNE/`.

Olah, C. (2014). Visualizing Mnist: An exploration of dimensionality reduction. *Visualizing Mnist: An Exploration of Dimensionality Reduction*. `https://colah.github.io/posts/2014-10-Visualizing-MNIST/`.

Otterbach, J. (2016). Curse of dimensionality, t-SNE and kullback-leibler divergence. *Curse of Dimensionality, t-SNE and Kullback-Leibler Divergence*. `http://jotterbach.github.io/2016/05/23/TSNE/`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peña, D. (2002). *Análisis de Datos Multivariantes*. McGraw-Hill Interamericana de España, Madrid.

R Development Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. `http://www.R-project.org`, version 3.6.0.

RStudio Team (2019). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA. Version 1.1.463.

Smith, L. I. (2002). A tutorial on principal components analysis. Technical report, Cornell University, USA.

TexStudio (2019). TexStudio integrated writing environment for creating LaTeX documents. Version 2.12.16 `https://www.texstudio.org/`.

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

Van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-SNE effectively. *Distill Publications*. `https://distill.pub/2016/misread-tsne/`.