Multidimensional Scaling for Big Data

Pedro Delicado and Cristian Pachón-García

Departament d'Estadística i Investigació Operativa

Universitat Politècica de Catalunya

August 2, 2021

Abstract

We present a set of algorithms implementing Multidimensional Scaling (MDS) for large data sets. MDS is a family of dimensionality reduction techniques using a $n \times n$ distance matrix as input, where n is the number of observed data, and producing a low-dimensional configuration: a $n \times r$ matrix with $r << n$. When n is large, MDS is unaffordable with standard MDS algorithms because their extremely large memory and time requirements. We overcome these difficulties by means of three non-standard algorithms (two of them being original proposals) based on the central idea of partitioning the data set into small pieces, where classical MDS methods can work. In order to check the performance of the algorithms as well as to compare them, we do a simulation study. Additionally, we use the algorithms to obtain an MDS configuration for a EMNSIT: a real large data set with more than $8 \cdot 10^5$ points. We conclude that the three algorithms are appropriate to use for obtaining an MDS configuration, but we recommend to use any of the two new proposals since they are fast algorithms with satisfactory statistical properties when working with Big Data. An R package implementing the algorithms has been created.

Keywords: Computational efficiency; Divide and conquer; Gower's interpolation formula; Procrustes transformation.

1 Introduction

Multidimensional Scaling (MDS) is a family of methods that represents high dimensional data in a low dimensional space with preservation of the Euclidean distance between observations. MDS uses a $n \times n$ distance matrix as input (or, alternatively, a similarity matrix), where n is the number of observed data, and producing a low-dimensional configuration: a $n \times r$ matrix with r much smaller than n. When n is large, MDS is unaffordable with standard MDS algorithms because their extremely large memory ($n(n-1)/2$ values should be stored simultaneously to represent a distance or similarity matrix) and time requirements. The cost of the classical MDS algorithm is

$O(n^3)$, as it requires eigendecomposition of a $n \times n$ matrix, for more details see Trefethen and Bau (1997).

We overcome these difficulties by means of three non-standard algorithms: *divide-and-conquer MDS*, *interpolation MDS*, and *fast MDS*. The first two are original proposals, while the third one was introduced by Yang, Liu, McMillan, and Wang (2006). These algorithms share the global scheme. First, they divide the data set into small pieces, where standard MDS methods can work. Then, the partial configurations obtained by MDS are combined using Procrustes transformations.

The building block standard MDS method used through this work is classical MDS (see, for instance, Section 3.2 in Krzanowski 2000 or Chapter 12 in Borg and Groenen 2005) but other MDS methods (Nonmetric MDS, for instance; see Section 3.3 in Krzanowski 2000, Section 12.5 in Johnson and Wichern 2002 or Chapter 9 in Borg and Groenen 2005) could also be used instead.

The rest of the paper is organized as follows: Section 2 provides a summary of classical MDS. Section 3 describes the three MDS algorithms for Big Data considered in this paper. We compare these algorithms by a simulation study described in Section 4. In Section 5 we challenge the three algorithms with a real large data set. A package implementing the three algorithms has been developed, which details are provided in Section 6. Section 7 summarizes the conclusions of the paper. Appendix A is devoted to provide a detailed explanation of classical MDS and Appendix B explains Procrustes transformations.

2 Classical Multidimensional Scaling

In this section we briefly revise classical Multidimensional Scaling. For a more detailed explanation we refer to Chapter 12 of Borg and Groenen (2005). Given a $n \times n$ matrix $\boldsymbol{\Delta} = (d_{ij}^2)$, where d_{ij}^2 is the squared distance between individuals i and j, the goal of MDS is to obtain a $n \times r$ *configuration matrix* $\mathbf{X}$ with orthogonal zero-mean columns such that the squared Euclidean distances between the rows of $\mathbf{X}$ are approximately equal to $\boldsymbol{\Delta}$. When equality is achieved we say that $\mathbf{X}$ is an Euclidean configuration for $\boldsymbol{\Delta}$.

The columns of $\mathbf{X}$ are called *principal coordinates* and they can be interpreted as the observations of r latent variables for the n individuals. Typically, the goal of MDS is dimensionality reduction, which involves looking for low-dimensional configurations (that is, r much lower than n).

Classical MDS is one of the standard ways to obtain configuration matrices from distance matrices. The main idea behind classical MDS is the following: for any set of n vectors $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ in a Euclidean space, there is a one-to-one relationship between their Euclidean distances $\{d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\| :$

$1 \leq i, j \leq n\}$ and their inner products $\{q_{ij} = \mathbf{y}_i^\mathrm{T} \mathbf{y}_j : 1 \leq i, j \leq n\}$:

$$d_{ij}^2 = q_{ii} + q_{jj} - 2q_{ij}, \ q_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i.}^2 - d_{.j}^2 + d_{..}^2),$$

where we have defined $d_{i.}^2 = (1/n) \sum_{j=1}^n d_{ij}^2$, $d_{.j}^2 = (1/n) \sum_{i=1}^n d_{ij}^2$, and $d_{..}^2 = (1/n^2) \sum_{i=1}^n \sum_{j=1}^n d_{ij}^2$. See Appendix A for the derivation of these expressions, following Borg and Groenen (2005). In order to write the previous relationships in a matrix form, some additional definitions are convenient. Let $\mathbf{I_n}$ be the identity matrix of dimension n, and let $\mathbf{1}_n$ be the n-dimensional vector with components equal to 1. The centering matrix in dimension n is defined as $\mathbf{P} = \mathbf{I_n} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n^\mathrm{T}$.

The following algorithm describes classical MDS:

1. Build the inner product matrix $\mathbf{Q} = -\frac{1}{2} \mathbf{P} \boldsymbol{\Delta} \mathbf{P}$.

2. Obtain the eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ of $\mathbf{Q}$, $i = 1, \ldots, n-1$, sorted in decreasing order of the eigenvalues:

$$\mathbf{Q} = \sum_{i=1}^{n-1} \lambda_i \mathbf{v}_i \mathbf{v}_i^\mathrm{T}.$$

   (Observe that $\mathbf{Q}$ has 0 as another eigenvalue with eigenvector $\mathbf{1}_n$, because it has sum zero by rows, and that some eigenvalues of $\mathbf{Q}$ may be negative when the distance matrix is not derived from a Euclidean distance).

3. Truncate this representation of $\mathbf{Q}$ taking just the r greatest eigenvalues:

$$\mathbf{Q} \approx \sum_{i=1}^{r} \lambda_i \mathbf{v}_i \mathbf{v}_i^\mathrm{T} = (\mathbf{V_r} \boldsymbol{\Lambda_r}^{1/2})(\boldsymbol{\Lambda_r}^{1/2} \mathbf{V_r}^\mathrm{T}),$$

   where $\mathbf{V_r}$ has columns $\mathbf{v}_i$, $i = 1, \ldots, r$, and $\boldsymbol{\Lambda_r} = \mathrm{diag}(\lambda_1, \ldots, \lambda_r)$.

4. Take $\mathbf{X} = \mathbf{V_r} \boldsymbol{\Lambda_r}^{1/2}$ as r-dimensional matrix configuration of $\boldsymbol{\Delta}$.

Observe that the configuration $\mathbf{X}$ has centered columns (because the eigenvectors of $\mathbf{Q}$ in $\mathbf{V_r}$ are orthogonal to $\mathbf{1}_n$, that is another eigenvector of $\mathbf{Q}$) with variance equal to the eigenvalues in $\boldsymbol{\Lambda_r}$ divided by n:

$$\mathrm{Var}(\mathbf{X}) = \frac{1}{n} \mathbf{X}^\mathrm{T} \mathbf{X} = \frac{1}{n} \boldsymbol{\Lambda_r}^{1/2} \mathbf{V_r}^\mathrm{T} \mathbf{V_r} \boldsymbol{\Lambda_r}^{1/2} = \frac{1}{n} \boldsymbol{\Lambda_r}.$$

Two goodness-of-fit measures are usually computed to assess the quality of the data set representation by the MDS configuration (see, e.g., Krzanowski (2000)):

$$G_1 = \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^{n-1} |\lambda_i|}, \ G_2 = \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^{n-1} \max\{\lambda_i, 0\}}.$$

The theoretical cost of the classical MDS algorithm is $\mathcal{O}(n^3)$ in time (because it requires the eigendecomposition of a $n \times n$ matrix) and $\mathcal{O}(n^2)$ memory (because $n \times n$ matrices as $\mathbf{\Delta}$ or $\mathbf{Q}$ must be stored). The theoretical cost of classical MDS makes this algorithm unaffordable when the sample size is large. In order to show that, an experiment is carried out. It consists of measuring the time and the RAM memory needed when performing MDS for a sequence of matrices with increasing number of rows. Classical MDS based on Euclidean distances between rows is used, as it is implemented in the R function `cmdscale` (R Core Team 2020). The computer used is a conventional one (MacBook Pro, processor 2.5GHz Intel Core i7, memory 16GB, 1600 MHz DDR3), which starts running out of memory for values of n greater than $2.2 \cdot 10^4$ (here we show results for $n \leq 10^4$). The source code for all the computations in the paper can be found at `https://github.com/pachoning/MDS`.

Figure 1 represents the amount of time needed in order to obtain an MDS configuration and the amount of RAM memory needed in order to calculate the distance matrix. It shows that the time and memory increase considerably as the sample size n increases, and they agree with the theoretical costs in time and memory.

3 Algorithms for Multidimensional Scaling with Big Data

In order to overcome the computational difficulties of classical MDS, we propose to work with three MDS algorithms designed for dealing with big data. Two of them are original proposals (*divide-and-conquer MDS* and *interpolation MDS*), while the third one (*fast MDS*) was proposed by Yang, Liu, McMillan, and Wang (2006).

3.1 Divide-and-conquer MDS

We base this algorithm on the principle of dividing and conquering. Roughly speaking, a large data set is divided into parts, then MDS is performed over every part and, finally, the partial configurations are combined so that all the points lie on the same coordinate system. Let us go into the details.

Let n be the number of observations of the original data set, which is divided into p parts of size ℓ, where $\ell \leq \bar{\ell}$, being $\bar{\ell}$ the largest number such that classical MDS runs efficiently for a distance matrix of dimension $\bar{\ell} \times \bar{\ell}$. As it will be shown in Section 4, the choice of the size ℓ affects the algorithm efficiency, and choosing ℓ much lower than $\bar{\ell}$ reduces the computing time.

The p parts into which the data set is divided must share a certain number of points, so that it is possible to connect the MDS partial configurations obtained from each part. Let c be the amount of connecting points shared
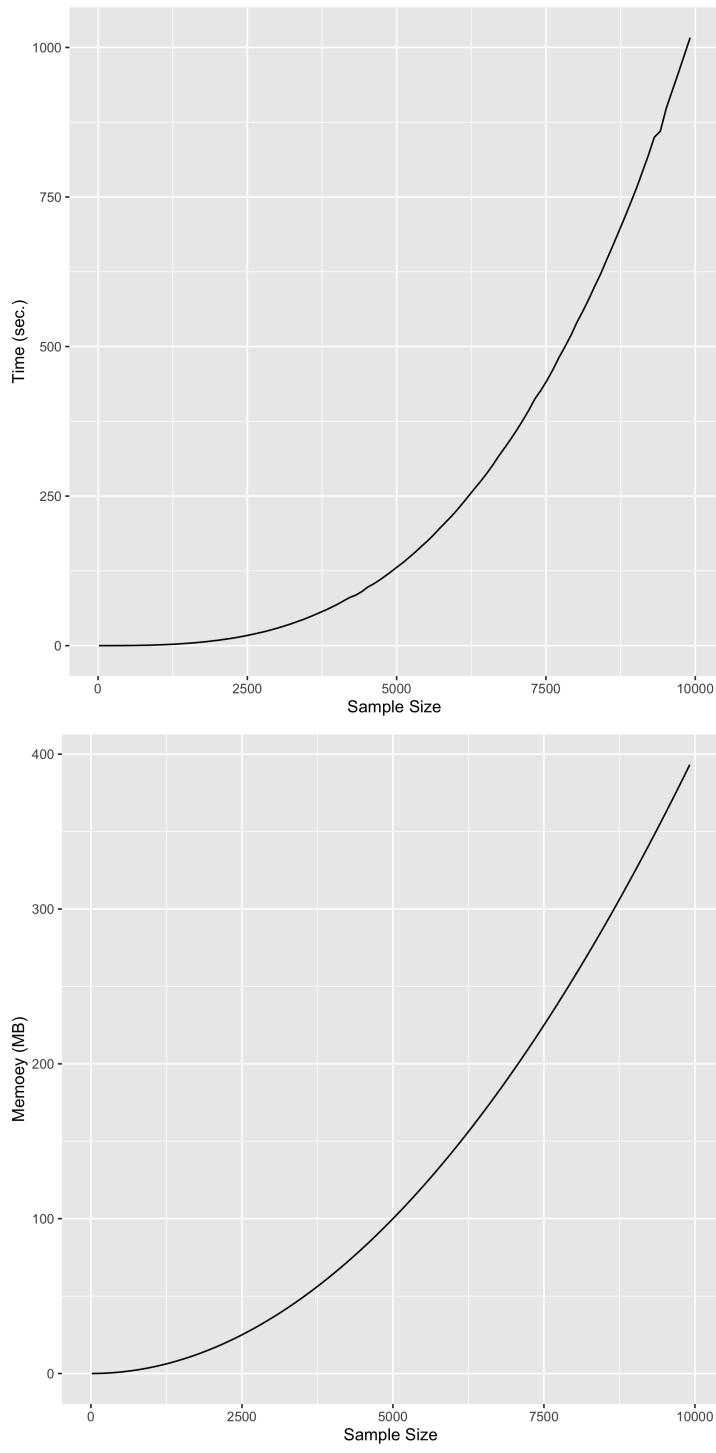
Figure 1: Elapsed time (*upper panel*) and required memory (*lower panel*) to obtain an MDS configuration as a function of the sample size n. Theoretical costs are $\mathcal{O}(n^3)$ in time and $\mathcal{O}(n^2)$ in memory.

by all the configurations. This number c should be large enough to guarantee good links between partial configurations, but as small as possible to favor efficient computations. Given that the partial configurations will be connected by a Procrustes transformation (see Appendix B) c must be at least equal to the required low dimensional configuration we are looking for when applying classical MDS to every part of the data set.

The divide-and-conquer MDS starts selecting at random the c connecting points from the data set. Then $p = \lceil 1 + (n - \ell)/(\ell - c) \rceil$, the lowest integer larger than or equal to $1 + (n - \ell)/(\ell - c)$, and p data subset are defined containing the c connecting points plus $\ell - c$ randomly selected (without replacement) points from the remaining $n - c$. Classical MDS is applied to each data subset, with target low dimensional configuration r. Let $\mathbf{X_j}$, $j = 1, \ldots, p$, be the $\ell \times r$ configuration obtained from the j-th data subset.

Since all the partitions share c points, the first configuration $\mathbf{X_1}$ can be aligned with any other $\mathbf{X_j}$, $j \geq 2$, using a Procrustes transformation. In order to do that, let $\mathbf{X_1^c}$ and $\mathbf{X_j^c}$ be the $c \times r$ matrices corresponding to the connecting points in $\mathbf{X_1}$ and $\mathbf{X_j}$ respectively. The Procrustes procedure (see Appendix B) is applied to $\mathbf{X_1^c}$ and $\mathbf{X_j^c}$ and the parameters $\mathbf{T_j} \in \mathbb{R}^{r \times r}$ and $\mathbf{t}_j \in \mathbb{R}^r$ are obtained so that

$$\mathbf{X_1^c} \approx \mathbf{X_j^c T_j} + \mathbf{1}_c \mathbf{t}_j^{\mathrm{T}}.$$

Let $\mathbf{X_j^a}$ be $\mathbf{X_j}$ without the connecting c points, and let

$$\dot{\mathbf{X}}_{\mathbf{j}} = \mathbf{X_j^a T_j} + \mathbf{1}_{l-c} \mathbf{t}_j^{\mathrm{T}}$$

be the $(\ell - c) \times r$ matrix with the j-configuration (excluding the connecting points) aligned with respect to $\mathbf{X_1}$. Finally, all the aligned partial MDS configurations are concatenated by rows to obtain the global $n \times r$ configuration:

$$\mathbf{X} = \left[\mathbf{X_1^{\mathrm{T}}} | \dot{\mathbf{X}}_{\mathbf{2}}^{\mathrm{T}} | \cdots | \dot{\mathbf{X}}_{\mathbf{p}}^{\mathrm{T}} \right]^{\mathrm{T}}.$$

When classical MDS is applied to each data subset, in addition to the r-dimensional configuration $\mathbf{X_j}$, other relevant information is obtained, namely, goodness-of-fit measures G_1^j and G_2^j, and the the eigenvalues λ_i^j, $i = 1, \ldots, r$, of the inner product matrix $\mathbf{Q_j}$ which, divided by the size of data subset, coincide with the eigenvalues of the variance matrix of the columns of $\mathbf{X_j}$, shared as well by $\dot{\mathbf{X}}_{\mathbf{j}}$ because $\mathbf{T_j}$ is an orthogonal matrix.

Global goodness-of-fit measures are defined in a natural way as

$$G_1 = \frac{1}{n} \sum_{j=1}^{p} n_j G_1^j, \; G_2 = \frac{1}{n} \sum_{j=1}^{p} n_j G_2^j,$$

where n_j is the size of the j-th data subset (ℓ for $j = 1$, and $(\ell - c)$ for

$j = 2, \ldots, p$, except perhaps for the last one). In a similar way, we define

$$\bar{\lambda}_i = \frac{1}{p} \sum_{j=1}^{p} \frac{\lambda_i^j}{n_j}, \; i = 1, \ldots, r,$$

as an estimation of the first r eigenvalues (divided by n) we would have obtained if classical MDS could have been applied to the whole data set. Observe that $\bar{\lambda}_i$ is also an estimation of the variance of the i-th column in final MDS configuration $\mathbf{X}$.

Observe that if the number of rows of the original data set is such that it allows to run classical MDS over the whole data set, then $p = 1$ and divide-and-conquer MDS is just the classical MDS.

In terms of computation time, the most costly operation is to obtain an MDS configuration for an $\ell \times \ell$ matrix, which cost is $\mathcal{O}(\ell^3)$. This operation is performed p times, being $p \approx n/\ell$. Therefore, the total cost is $\mathcal{O}(n\ell^2)$, or just $\mathcal{O}(n)$ when ℓ is considered as a given parameter.

3.2 Interpolation MDS

The basic idea of our second proposal is as follows. Given that the size of the data set is too large, we propose to take a random sample from it of size $\ell \leq \bar{\ell}$ (where $\bar{\ell}$ is the largest size for which classical MDS is applicable), to perform classical MDS to it, and to extend the obtained results to the rest of the data set by using Gower's interpolation formula (see, for instance, the Appendix of Gower and Hand 1995), which allows us to add a new set of points to an existing MDS configuration. Note that this proposal is parallel to what Statistics does when a population is too large to examine all its individuals.

Gower's interpolation procedure works as follows. Given a first data subset of size ℓ, let $\mathbf{D_1} = (d_{ij})$ be the $\ell \times \ell$ distance matrix between its elements, and let $\mathbf{X_1}$ be $\ell \times r$ matrix containing its classical MDS configuration. Consider a new data subset of size m, and let $\mathbf{A_{21}}$ be the $m \times \ell$ distance matrix between its m elements and the ℓ ones in the first data subset. One wants to project these new m elements into the existing MDS configuration, in such a way that the Euclidean distances between the new projected points and the original ones are as close as possible to the elements of $\mathbf{A_{21}}$. We briefly summarize how to do so using Gower's interpolation formula. Define $\mathbf{Q_1} = -\frac{1}{2}\mathbf{P}\mathbf{\Delta_1}\mathbf{P}^{\mathrm{T}}$, where $\mathbf{\Delta_1} = (d_{ij}^2)$ and $\mathbf{P} = \mathbf{I}_\ell - \frac{1}{\ell}\mathbf{1}_\ell\mathbf{1}_\ell^{\mathrm{T}}$. Let $\mathbf{q}_1$ be the diagonal of $\mathbf{Q_1}$, treated as a column vector. Let $\mathbf{A_{21}}^2$ be the matrix of the square of the elements of $\mathbf{A_{21}}$. Let $\mathbf{S_1}$ be the variance-covariance matrix of the r columns of $\mathbf{X_1}$. The Gower's interpolation formula states that the interpolated coordinates for the new m observations are given by

$$\ddot{\mathbf{X}}_2 = \frac{1}{2\ell}(\mathbf{1}_m\mathbf{q}_1^{\mathrm{T}} - \mathbf{A_{21}}^2)\mathbf{X_1}\mathbf{S_1}^{-1}. \tag{1}$$

The resulting MDS for the m observations of $\ddot{\mathbf{X}}_2$ is in the same coordinate system as $\mathbf{X}_1$. So, it is not needed to do any Procrustes transformation.

Observe that Gower's interpolation formula is valid for any number $m \geq 1$ of elements in the second data subset. Nevertheless, for $m \geq \bar{\ell}$ the memory limitations reported for classical MDS could appear here because formula (1) involves matrices of dimension $m \times \ell$. Therefore we propose to use $m = \ell$ when projecting new observations into an existing MDS configuration.

Finally, the proposed interpolation MDS algorithm operates as follows. First, the data set of size n is divided into $p = \lceil n/\ell \rceil$ parts. The first data subset is used to compute $\mathbf{X}_1$ and the other elements in Gower's interpolation formula (1). Then we use this formula to obtain $\ddot{\mathbf{X}}_j$, where $j \in \{2, \ldots, p\}$.

Finally, all the interpolated partial MDS configurations are concatenated by rows to obtain the global $n \times r$ configuration:

$$\mathbf{X} = \left[\mathbf{X}_1^{\mathrm{T}} | \ddot{\mathbf{X}}_2^{\mathrm{T}} | \cdots | \ddot{\mathbf{X}}_p^{\mathrm{T}} \right]^{\mathrm{T}}.$$

As in the previous algorithm, note that if $n \leq \ell$ then $p = 1$ and interpolation MDS is just classical MDS.

The goodness-of-fit measures G_1^1 and G_2^1 obtained when applying classical MDS to the first data subset are taken as estimation of the quality of the global MDS configuration $\mathbf{X}$. Similarly, the eigenvalues λ_i^1, $i = 1, \ldots, r$, divided by ℓ, are estimations of the variance of the columns of $\mathbf{X}$.

The most costly operation in this algorithm is the computation of the distance matrix $\mathbf{A}_{21}$, that in our case is of order $\mathcal{O}(\ell^2)$ because we use $m = \ell$. This operation is repeated p times, with $p \approx n/\ell$. So the computation cost in time of this algorithm is $\mathcal{O}(n\ell)$, or $\mathcal{O}(n)$ for a fixed parameter ℓ.

3.3   Fast MDS

Yang, Liu, McMillan, and Wang (2006) overcome the problem of scalability using recursive programming in combination with a divide and conquer strategy. They name their proposal *Fast Multidimensional Scaling* (fast MDS).

As in the previous approaches, fast MDS also randomly divides the whole sample data set of size n into several data subsets, but now the size of the data subsets can be larger than ℓ (with $\ell \leq \bar{\ell}$, where $\bar{\ell}$ is the limit size for classical MDS) because the recursive strategy: fast MDS is applied again when the data subsets are larger than ℓ. Yang, Liu, McMillan, and Wang (2006) do not give precise indications for choosing ℓ: they say that ℓ must be the size of *the largest matrix that allows MDS to be executed efficiently*. From this imprecise indication it is only clear that ℓ must be lower than or equal to $\bar{\ell}$.

In the last step of the fast MDS algorithm, the partial MDS configurations obtained for each data subset are combined into a global MDS configuration by a Procrustes transformation (as in divide-and-conquer MDS, Section 3.1). To do so, a small subset of size s is randomly selected from each data subset (Yang, Liu, McMillan, and Wang 2006 call them the *sampling points*). The role of s in fast MDS is equivalent to that of c (the amount of connecting points) in divide-and-conquer MDS, and the same considerations for its choice apply here.

The selected sampling points from each data subset are joined to form an *alignment set*, over which classical MDS is performed giving rise to an *alignment configuration*: a $\ell \times r$ matrix $\mathbf{X_{align}}$. In order to be able to apply classical MDS to the alignment set, its size must not exceed the limit size ℓ. Therefore, the number p of data subsets is taken as $p = \lfloor \ell/s \rfloor$ (the integer part of ℓ/s).

Each one of the p data subsets has size $\tilde{n} = \lceil n/p \rceil$ (except perhaps the last one). If $\tilde{n} \leq \ell$ then classical MDS is applied to each data subset. Otherwise, fast MDS is recursively applied. In either case, a final MDS configuration is obtained for each data subset, namely the $\tilde{n} \times r$ matrices $\mathbf{X_j}, j = 1, \ldots, p$.

Every data subset shares s points with the alignment set. Therefore every MDS configuration $\mathbf{X_j}$, $j \geq 1$, can be aligned with the alignment configuration $\mathbf{X_{align}}$ using a Procrustes transformation. Let $\mathbf{X^s_{align,j}}$ and $\mathbf{X^s_j}$ be the $s \times r$ matrices corresponding to the j-th set of sampling points in $\mathbf{X_{align}}$ and $\mathbf{X_j}$ respectively. The Procrustes procedure (see Appendix B) is applied to $\mathbf{X^s_{align,j}}$ and $\mathbf{X^c_j}$ and the parameters, $\mathbf{T_j} \in \mathbb{R}^{r \times r}$ and $\mathbf{t}_j \in \mathbb{R}^r$ are obtained, so that

$$\mathbf{X^s_{align,j}} \approx \mathbf{X^s_j T_j} + \mathbf{1}_s \mathbf{t}_j^{\mathrm{T}}.$$

Let

$$\tilde{\mathbf{X}}_\mathbf{j} = \mathbf{X_j T_j} + \mathbf{1}_{\tilde{n}} \mathbf{t}_j^{\mathrm{T}}$$

be the $\tilde{n} \times r$ matrix with the j-configuration aligned with respect to $\mathbf{X_{align}}$. Finally, all the aligned partial MDS configurations are concatenated by rows to obtain the global $n \times r$ configuration:

$$\mathbf{X} = \left[\tilde{\mathbf{X}}_\mathbf{1}^{\mathrm{T}} | \tilde{\mathbf{X}}_\mathbf{2}^{\mathrm{T}} | \cdots | \tilde{\mathbf{X}}_\mathbf{p}^{\mathrm{T}} \right]^{\mathrm{T}}.$$

As in the previous algorithms, note that if $n \leq \ell$ then $p = 1$ and fast MDS is just classical MDS. Global goodness-of-fit measures and eigenvalues are defined as in divide-and-conquer MDS.

Yang, Liu, McMillan, and Wang (2006) use the Master theorem for divide-and-conquer recurrences (Bentley, Haken, and Saxe 1980) to establish that the computation cost in time of the fast MDS algorithm is $\mathcal{O}(n \log n)$, and they do not explicitly report the dependence of ℓ.

4 Simulation study

In this section, we present a simulation study to evaluate the three methods. In particular, we address the following questions: (1) the ability to capture the right data dimensionality and (2) the speed of the algorithms. Regarding this last point, in order to make the algorithms as fast as possible, we have parallelized them by using the R package parallel (see R Core Team (2020)). This way we take profit of the 5 cores of our computer.

As a previous step, we deal with the right choice of parameter ℓ for each algorithm, according to these two questions.

4.1 Choosing the partitions size ℓ

Before running the complete simulation study, we examine the effect of the partitions size ℓ on the algorithms efficiency, measuring the ability to recover the low dimensional data structure, and the computation time. We run a simple experiment using just one simulated data set that is analyzed with the three algorithms using 10 different values of ℓ: from 200 to 1000 in steps of 100, and 1500. A data matrix $\mathcal{X}$ of dimension $10^6 \times 100$ is generated, with elements being independent random normal observations with zero mean and variance equal to 15 for the first 10 columns, and 1 for the other. The three algorithms are executed with $k = 10$, and $s = c = 20$.

Each algorithm is evaluated for each value of ℓ with three performance measures: the correlation of the 10 first columns of $\mathcal{X}$ with the obtained configuration matrix, the proximity of the $k = 10$ estimated eigenvalues to 15 (their theoretical value), and the computation time (in seconds). More details on these measures are given in Sections 4.3, 4.4 and 4.5, respectively.

Figure 2 shows the performance measures for each algorithm as functions of the ℓ. We look for the values of ℓ at which a compromise between the three criteria is achieved. We consider that the value conditions are met by choosing a value for ℓ equal to 400 for divide-and-conquer MDS, a value for ℓ equal to 1,000 for interpolation MDS as well as for fast MDS. Notice that the three performance measures for the fast MDS algorithm depend on ℓ in a non-monotonous way. It seems that there are discontinuities at certain critic values of ℓ (in this case, at $\ell = 700$), possibly due to successive recursive divisions: it can be checked that for $n = 10^6$ and $\ell = 700$ the algorithm requires a total of 42,875 partitions, with an average size of 23.32 points, while when $\ell = 800$ the number of partitions is 1,600 with average size 625. Observe that the critical points would be different for other sample sizes. Therefore our recommendation of using $\ell = 1,000$ for fast MDS is appropriate for $n = 10^6$ but could not be the best choice for other values of n. In this sense, the recommended values of ℓ are more robust against changes in n for divide-and-conquer MDS and for interpolation MDS than for fast MDS.
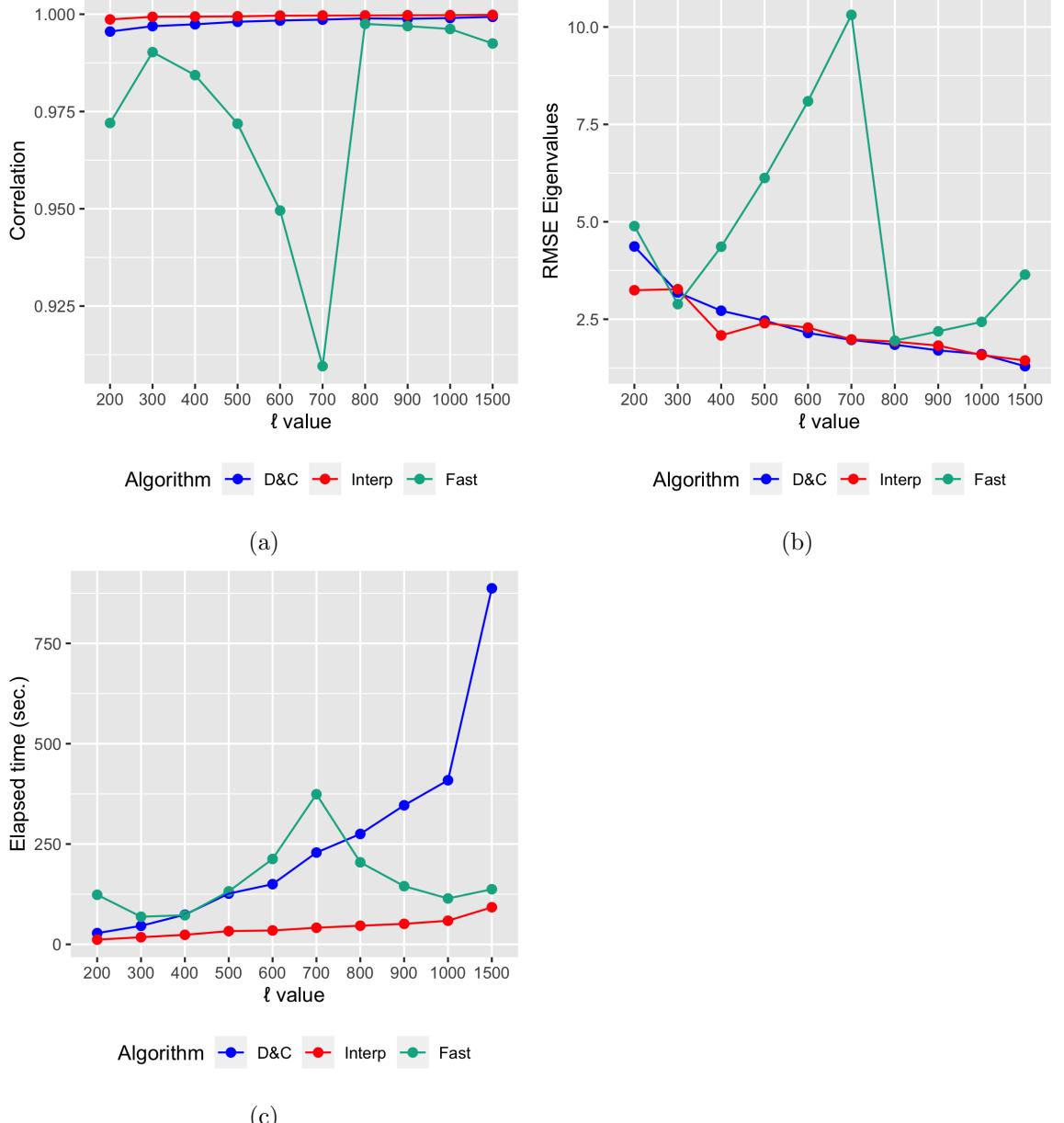
(a)

(b)

(c)

Figure 2: (2a) Mean correlation coefficient for the 10 first columns. (2b) Root mean squared error of λ_i, $i \in \{1, \ldots, 10\}$, as estimators of their theoretical value 15. (2c) Time required to obtain an MDS configuration. The three of them are depicted as a function of ℓ parameter. Each algorithm is represented by a different color.

11

4.2 Design of the simulation

Different experiments are conducted in order to answer the previous questions. At each experiment, data matrices $\mathcal{X}$ of dimension $n \times k$ are generated, which rows are considered to be the individuals in the data set. Euclidean distances between rows of $\mathcal{X}$ are used throughout the study. Several scenarios are explored, taking into account different factors:

Sample size: Different sample sizes n are taken into account, combining small data sets and large ones. A total of six sample sizes are used, which are:

- Small sample sizes: $5 \cdot 10^3$, 10^4, and $2 \cdot 10^4$.
- Large sample sizes: 10^5 and 10^6.

Data dimension: Two different number of columns k are considered: 10 and 100.

Dominant dimension: The first h columns in $\mathcal{X}$ have variance equal to 15, while the other $k-h$ have variance equal to 1. We refer to h as the *dominant dimension*. The idea of this is to see if the algorithms are able to capture the relevant data dimensionality. We have considered values for h from 1 to 10. If the dominant dimension is equal to h, we say that there are h *dominant directions*.

There is a total of 120 scenarios to simulate (6 sample sizes, 2 data dimensions, and 10 dominant dimensions). Each scenario is replicated 100 times. So, a total of 12,000 simulations are carried out. For every simulation, the data matrix $\mathcal{X}$ is generated from a multivariate normal distribution with zero mean independent coordinates and variances 15 for the first h columns and 1 for the others. The three MDS algorithms are run based on Euclidean distances between rows of $\mathcal{X}$.

When running the algorithms, we ask for as many columns r as the dominant dimension exist in the simulated data set, i.e, $r = h$. Therefore, the resulting low-dimensional MDS configurations $\mathbf{X}$ have dimension $n \times h$. In addition to $\mathbf{X}$, the elapsed time is stored for each simulation.

Note that the original data set, $\mathcal{X}$, is already an MDS configuration for itself, since we simulate independent columns with zero mean. Therefore, even though n is so large that classical MDS can not be calculated, the first h columns of $\mathcal{X}$ can be taken as a benchmark classical MDS solution to which compare against the MDS configurations $\mathbf{X}$ provided by the three algorithms.

In order to test the results quality of the algorithms as well as the time needed to compute the MDS configurations, some metrics are calculated:

- The quality of the results is measured by the following statistics:

– Correlation between the dominant directions of the data and the corresponding dimensions provided by the algorithms. Note that any rotation of an MDS configurations $\mathbf{X}$ leads to another equally valid configuration. Therefore, before computing the correlations between the first h columns of $\mathcal{X}$, which we denote by $\mathcal{X}^{\mathbf{h}}$, and those of $\mathbf{X}$, we must be sure that both matrices are *correctly aligned*, in the sense that we are using the rotation of $\mathbf{X}$ that best fit the columns of $\mathcal{X}$. A Procrustes transformation is done to achieve this alignment.

– *bias* and *Mean Squared Error* (MSE) of the eigenvalues $\bar{\lambda}_i$, $i = 1, \ldots, h$, as estimators of the variance of the first h columns of $\mathbf{X}$ (namely, 15).

• The computational efficiency is measured by the average elapsed time to get the MDS configurations over the 100 replications of each scenario.

The three algorithms require to specify a value for ℓ parameter. In Section 4.1 the choice of this parameter has been discussed. The values used in simulation are $\ell = 400$ for divide-and-conquer MDS, and $\ell = 1,000$ for the other algorithms.

Divide-and-conquer MDS and fast MDS have an additional parameter each: the number c of connecting points in divide-and-conquer MDS, and the number s of sampling points in fast MDS. Both c and s must be greater than or equal to the number r of columns required for the MDS configuration. Yang, Liu, McMillan, and Wang (2006) use $s = 2r$, and we do the same when fixing c. Using lower values for c or s could lead to incorrect MDS configurations, as there would be very few points on which to base the Procrustes transformations. On the other hand, using larger values for c or s would lengthen the time of the algorithms.

4.3   Results on correlation with the dominant directions

This section is aimed to study the ability of the three MDS algorithms to capture the dominant directions. Given a simulated data set, $\mathcal{X}$, there are four MDS configurations related to the data set: the data set itself, $\mathcal{X}$, and one per each of the three methods we are proposing. After applying Procrustes to a given MDS configuration, $\mathbf{X}$, the columns of the resulting matrix should be highly correlated with the dominant directions of $\mathcal{X}$ (as described in Section 4.2).

Table 1 contains the 2.5% quantile ($q_{0.025}$), the mean value (mean) and the 97.5% quantile ($q_{0.975}$) for the correlation coefficients for each of the three algorithms. For each scenario described in Section 4.2, a total of h correlation coefficients are computed, where h is the dominant dimension

Table 1: Quantiles of order 2.5% ($q_{0.025}$) and 97.5% ($q_{0.975}$), and mean values for the correlation coefficients between the original variables and the ones recovered by the three MDS methods. All the simulations are taken into account.

Algorithm	$q_{0.025}$	mean	$q_{0.975}$
Divide-and-conquer MDS	0.99683	0.99799	0.99905
Interpolation MDS	0.99967	0.99987	1
Fast MDS	0.91747	0.98392	0.99884

of the scenario. Then, 660 correlation coefficients are derived from a single replication of the 120 scenarios. As performed 100 replications, Table 1 shows descriptive statistics of correlation coefficients sets of size 66,000. It can be seen that there is a high correlation between the MDS configurations and the dominant directions of $\mathcal{X}$ for each of the three algorithms. In addition, interpolation MDS seems to be the algorithm which MDS configuration is most correlated with the dominant directions, followed by divide-and-conquer MDS, and then by fast MDS.

Figure 3 displays the mean value of the correlation coefficients, taking into account the sample size (horizontal axis) and the dominant dimension (h), represented at different plots. This figure shows that highly correlation is preserved across all the dimensions of the configurations. As happens in Table 1, the most correlated configuration with the actual dominant directions is that provided by interpolation MDS configuration, closely follwed by divide-and-conquer MDS, and both preferred to fast MDS (specially for specific sample sizes). As in Table 1, the sample size used to produce Figure 3 is equal to 66,000.

As a particular case, Figure 4 represents the boxplot of the correlation coefficients for the scenario which sample size is 10^6, there are 100 columns ($k = 100$) and 10 out of them are dominant directions ($h = 10$). Since each scenario is simulated 100 times, a sample of length 100 is used to create each boxplot in Figure 4. Notice that each plot represents a dominant direction and each algorithm is represented in a different color. Observe that interpolation MDS gives the best results (the 100 correlations coefficients are almost equal to 1 for all dominant dimension). Additionally, divide-and-conquer MDS results are better than those of fast MDS.

4.4   Results on eigenvalues

In this section we study how the eigenvalues provided by the algorithms estimate the variance of the dominant directions. Since these variances are equal to 15, it is expected the eigenvalues to be close to 15. As goodness-of-fit metrics, we use bias and MSE. A good estimator is one which bias and MSE are close to 0.
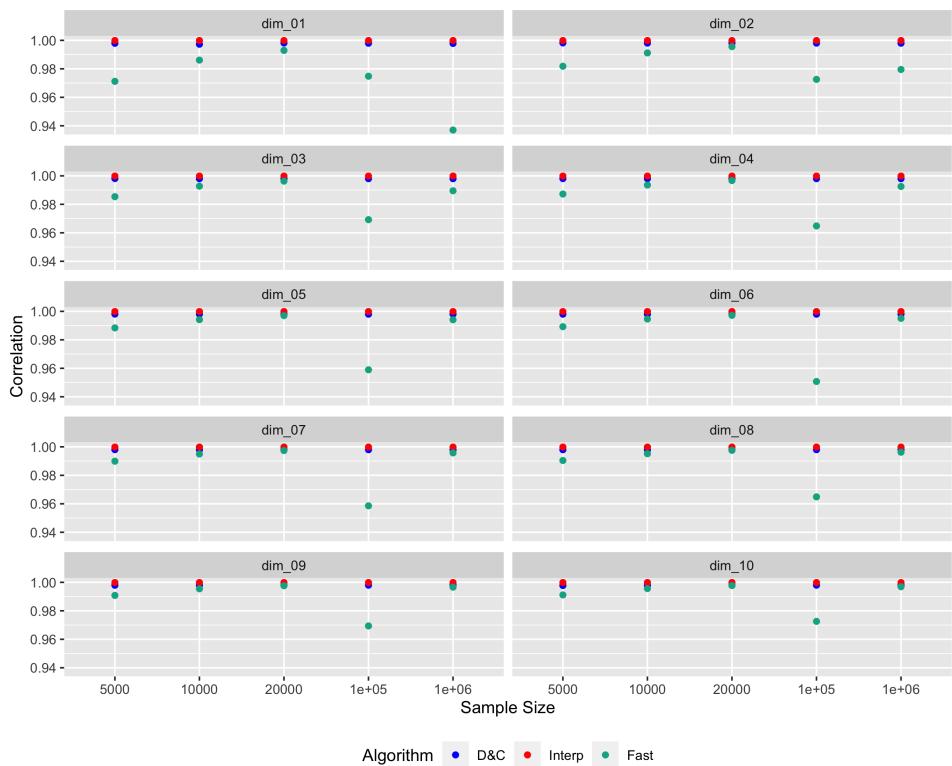
Figure 3: Mean value for the correlation coefficients grouped by sample size and dominant dimension. Different MDS algorithms are represented with different colors.
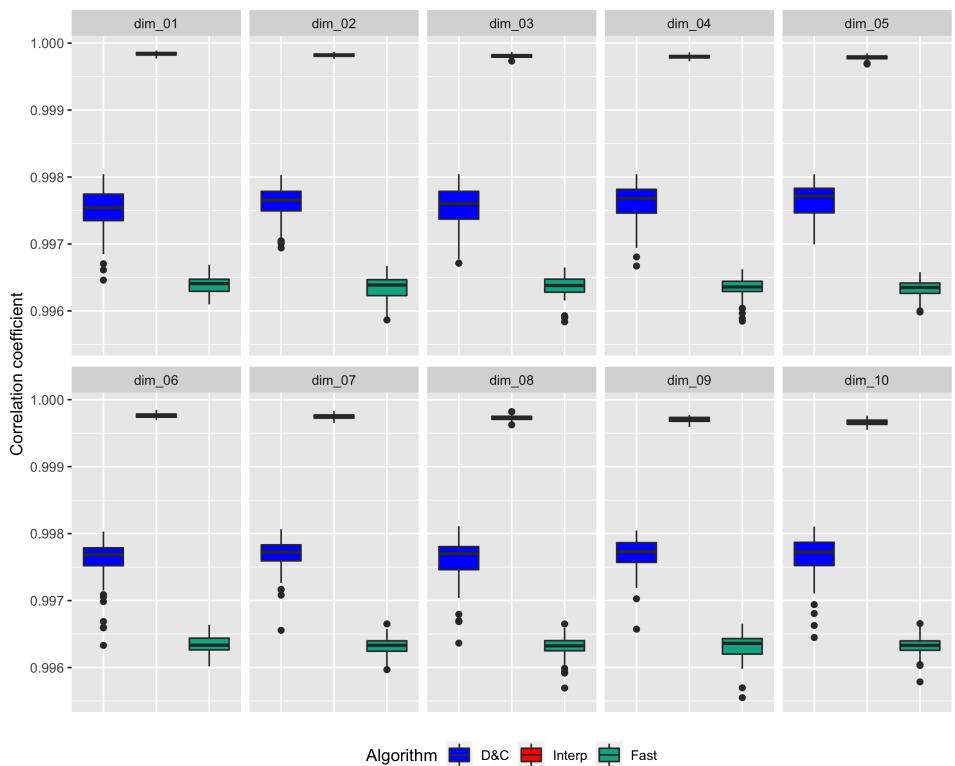
Figure 4: Boxplot of the correlation coefficients for the scenario which sample size is 10^6, there are 100 columns and 10 out of them are dominant directions.
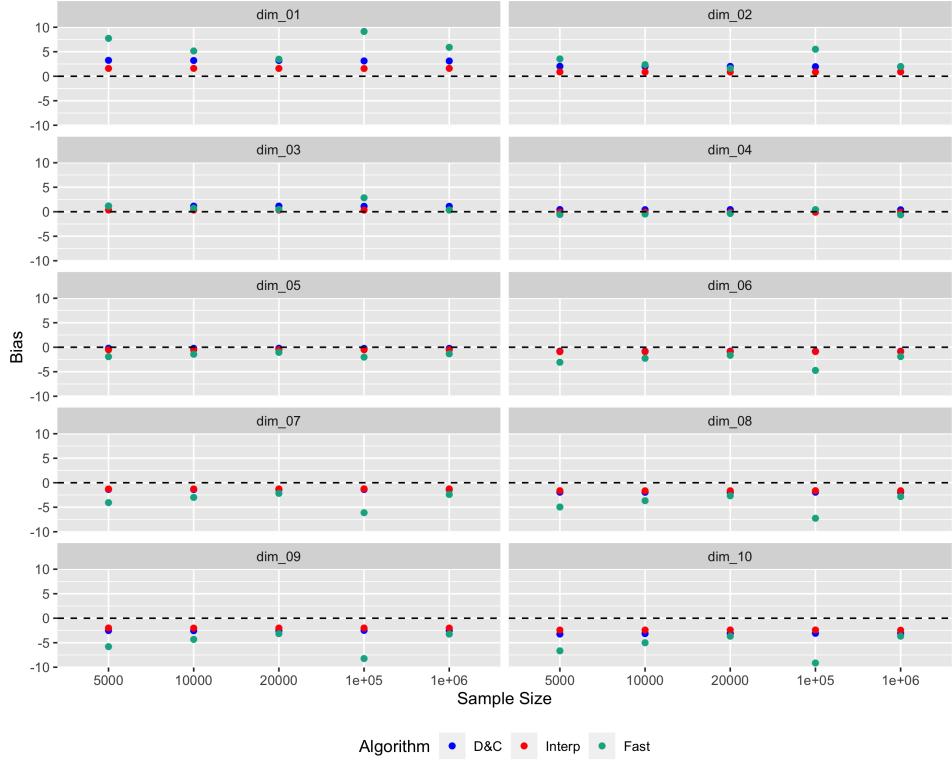
Figure 5: Bias of the estimators for the variance of the dominant directions grouped by sample size and dominant dimension. Different MDS algorithms are represented with different colors.

Figure 5 and Figure 6 display the bias and the MSE, respectively, taking into account the sample size (horizontal axis) and the dominant dimension (h), represented at different plots. Divide-and-conquer MDS and interpolation MDS seem to have lower bias, as well as lower MSE, than fast MDS. As in Table 1, the sample size used to produce Figure 5 and Figure 6 is equal to 66,000.

As a particular case, Figure 7 represents the boxplot for the *estimation error* (the computed eigenvalues minus 15) made for scenario which sample size is 10^6, there are 100 columns ($k = 100$) and 10 out of them are dominant directions ($h = 10$). Since each scenario is simulated 100 times, a sample of length 100 is used to create each boxplot in Figure 7. Notice that each plot represents a dominant direction and each algorithm is represented in a different color. It can be seen that interpolation MDS presents lower bias and larger variance than the other two algorithms. Divide-and-conquer has larger bias than fast MDS for dominant dimensions 1 to 6, and the opposite happens for the remaining directions. We conclude that interpolation MDS is able to manage better than the other algorithms situations where there
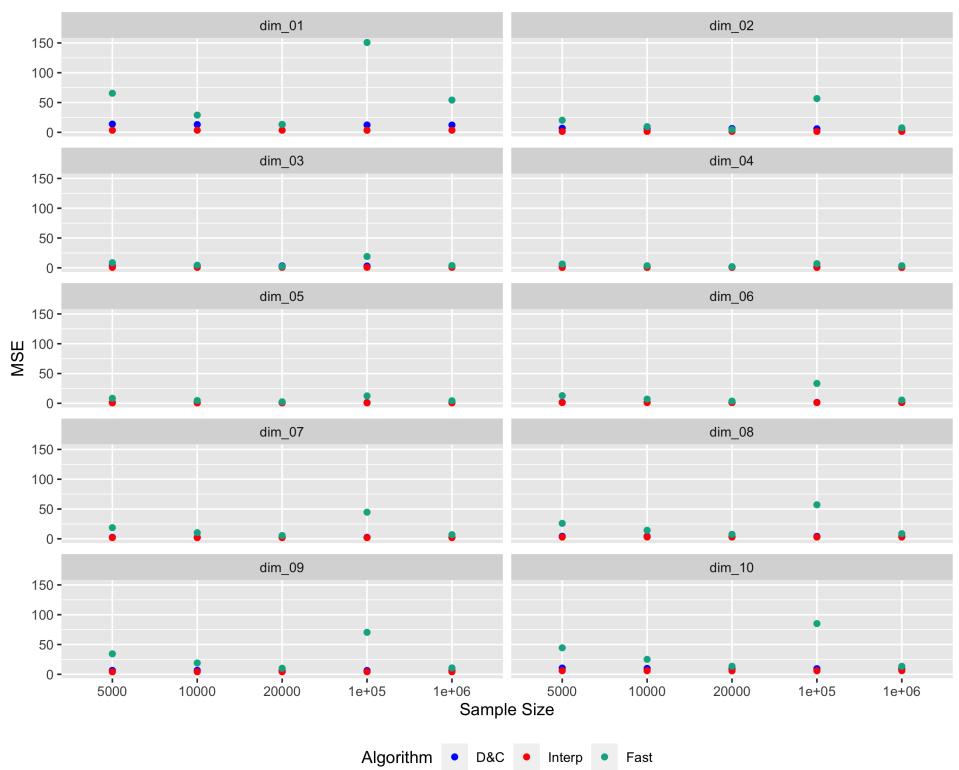
17

Figure 6: MSE of the estimators for the variance of the dominant directions grouped by sample size and dominant dimension. Different MDS algorithms are represented with different colors.
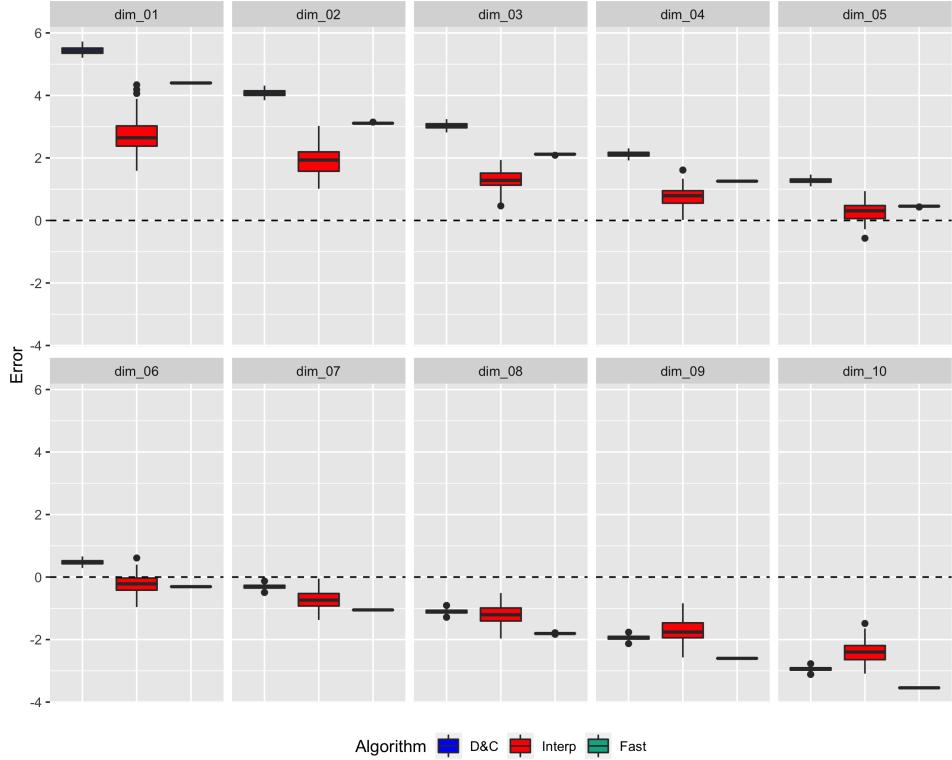
Figure 7: Boxplot of the error in variance estimation for the scenario which sample size is 10^6, there are 100 columns and 10 out of them are dominant directions.

are many dominant directions with similar variability.

4.5 Time to obtain an MDS configuration

In this section we study the cost of each algorithm in terms of speed. Figure 8 represents the mean of the $\log_{10}$ of time (in seconds) needed to obtain an MDS configuration as a function of the $\log_{10}$ of sample size (horizontal axis) and the MDS method (color). It seems that the fastest algorithms are divide-and-conquer MDS for moderate sample sizes, and interpolation MDS for very large sizes. Observe that fast MDS performs reasonably well, but it is never the fastest algorithm. As there are 120 scenarios each replicated 100 times, the sample size used in Figure 8 is equal to 12,000.

As a particular case, Table 2 contains the 2.5% quantile ($q_{0.025}$), the mean value (mean) and the 97.5% quantile ($q_{0.975}$) for the elapsed time related to the scenario which which sample size is 10^6, there are 100 columns ($k = 100$) and 10 out of them are dominant directions ($h = 10$). The results related to the quantiles for the remaining scenarios (graphics not included
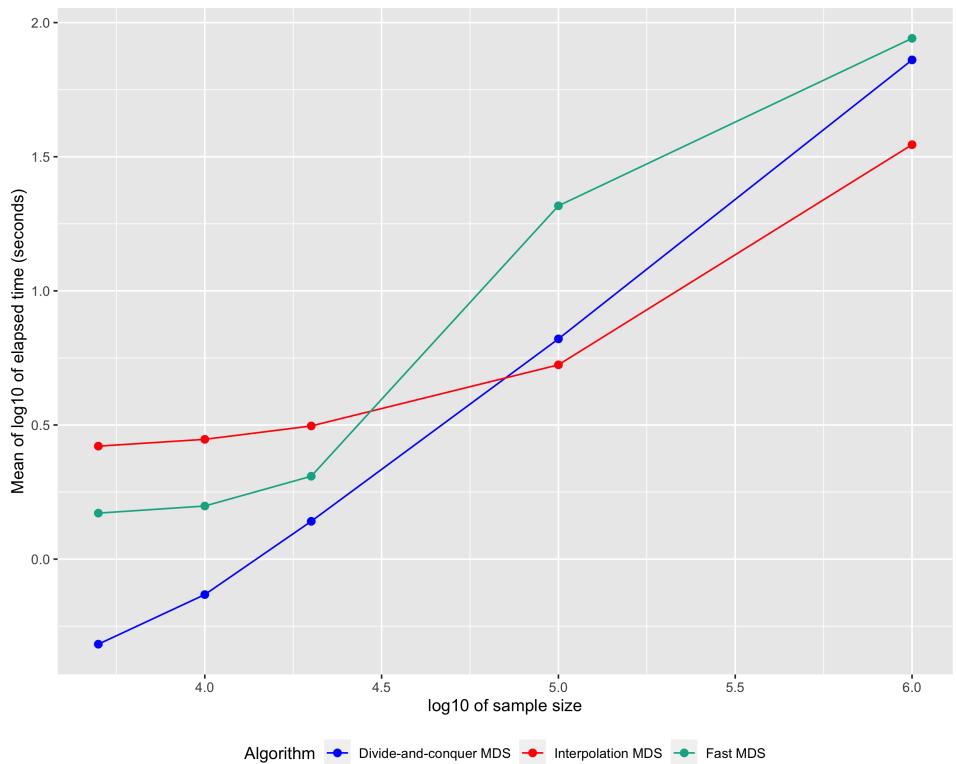
Figure 8: Mean value of the $\log_{10}$ of elapsed time, in seconds, grouped by sample size. Colors represent the method.

Table 2: Quantiles of order 2.5% ($q_{0.025}$) and 97.5% ($q_{0.975}$), and mean values for the elapsed time (in seconds) for the scenario which sample size is 10^6, there are 100 columns and 10 out of them are dominant directions.

Algorithm	$q_{0.025}$	mean	$q_{0.975}$
Divide-and-conquer MDS	72	98	182
Interpolation MDS	47	68	138
Fast MDS	91	123	244

here) are similar to the ones in Table 2.

4.6 Conclusions from the simulation study

The conclusions from the simulation study (Sections 4.3, 4.4 and 4.5) are the following. Regarding their ability to recover the low dimensional configuration eventually given by classical MDS, interpolation MDS is slightly better than divide-and-conquer MDS, and both are clearly preferred to fast MDS. A similar conclusion follows according to the criterion of variance estimation for the leading dimensions. This is due to the number of individuals in each partition: while divide-and-conquer MDS and interpolation MDS work with partitions of fixed size ℓ, fast MDS divides the data set into $p = \ell/s$ parts recursively until the size of the parts is less than or equal to ℓ. Such a criterion could end up creating data subsets with a small number of individuals, which increases the uncertainty of the algorithm.

Finally, when comparing computation times, for moderate sample sizes (up to 10^4) divide-and-conquer MDS is the fastest one, followed by fast MDS and by interpolation MDS, in this order. For sample size 10^5 interpolation MDS and divide-and-conquer MDS are comparable, and both are faster than fast MDS. Finally, for sample size 10^6 interpolation MDS is faster than divide-and-conquer MDS, and it is faster than fast MDS. We summarize telling that divide-and-conquer MDS and interpolation MDS are the algorithms with the best global performance, and that both are preferred to fast MDS.

5 Using MDS algorithms with EMNIST data set

In this section we use the three MDS algorithms with a real large data set: EMNIST (Cohen, Afshar, Tapson, and Schaik 2017), available in `https://www.nist.gov/itl/products-and-services/emnist-dataset`. The EMNIST data set is composed by handwritten character digits, lowercase letters and capital letters. They are derived from the Special Database 19 (Grother 1970) and converted to a 28×28 pixel image format. The images have this size so that they match with the MNIST data set format (LeCun and Cortes

Table 3: Time (in seconds) required and goodness of fit metric obtained in the low dimensional configuration space.

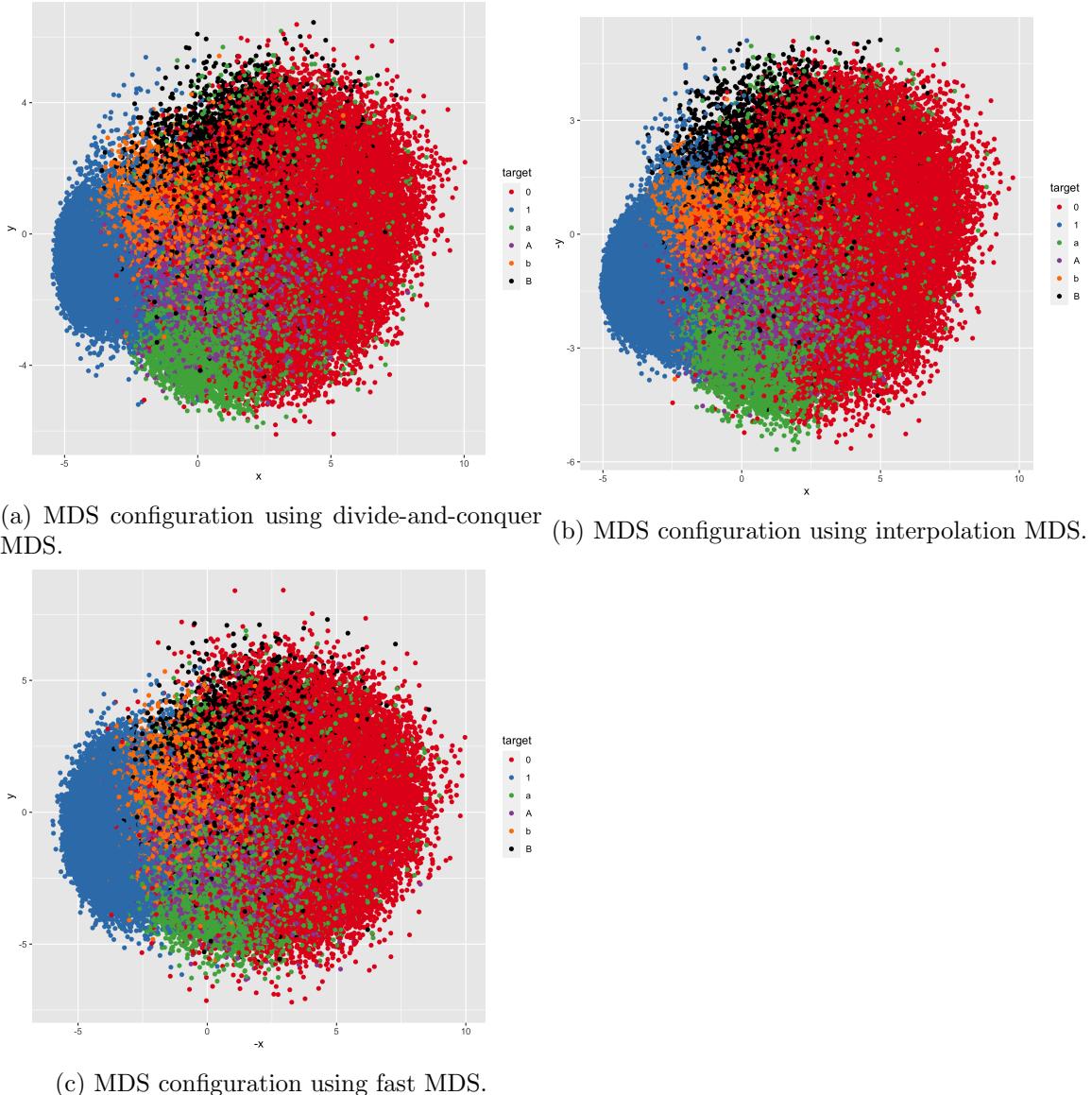| Algorithm	Time	G_1
Divide-and-conquer MDS	251	18%
Interpolation MDS	401	17%
Fast MDS	488	19%

2010). In total, there are 814,255 images divided into 62 classes: 10 digits (from '0' to '9'; the 49.5% of the total), 26 lowercase letters (from 'a' to 'z'; 23.5%) and 26 capital letters (from 'A' to 'Z'; 27%). This data set is usually divided into training and a test data sets when it is used for classification. Nevertheless, we use the whole data set in order to manage a larger data set.

The Euclidean distance between the vector representation of the images in dimension $28^2 = 784$ has been used to perform MDS with the three algorithms. In order to visualize the results, the required low dimensional configuration is equal to 2 ($r = 2$), ℓ parameter is set in same way as described in Section 4.1. Divide-and-conquer MDS and fast MDS required to specify two extra parameters: c and s respectively. They both are set to 4.

Table 3 shows the time needed to obtain an MDS configuration as well as both goodness-of-fit metric G_1 described in Section 3.1, which measure the relative size of the first $r = 2$ eigenvalues. Observe that the metric G_2 (also defined in Section 3.1) coincides with G_1 in this example because we are considering the Euclidean distance in $\mathbb{R}^{784}$, which implies that all the eigenvalues are non-negative. Regarding computing time, divide-and-conquer MDS is the fastest algorithm, being able to process the EMNIST data set in approximately 4 minutes, while the other algorithms require almost twice this time. On the other hand, the quality of the solution is similar for the three configurations: the two first eigenvalues represent around the 18% of the sum of all of them, which we consider a remarkable percentage for only two dimensions.

In order to check that the resulting configuration is an MDS configuration, the mean value for each principal coordinate and the correlation coefficient between the two principal coordinates for each algorithm are computed. All these quantities are lower than 10^{-16} in absolute value. Figure 9 shows the MDS configuration for each of the algorithms. In order to provide a comprehensive figure, we plot a subset composed by 2 digits ('0' and '1'), 2 lowercase letters ('a' and 'b') and 2 capital letters ('A' and 'B'). Therefore, in total there are 114,751 individuals in Figure 9.

A typical question related to methods that perform dimensionality reduction is about the choice of the value of the dominant dimension (h). A

(a) MDS configuration using divide-and-conquer MDS.

(b) MDS configuration using interpolation MDS.

(c) MDS configuration using fast MDS.

Figure 9: (9a) Divide-and-conquer MDS configuration. (9b) Interpolation MDS configuration. (9c) Fast MDS configuration. The classes used are: 2 digits ('0' and '1'), 2 lowercase letters ('a' and 'b') and 2 capital letters ('A' and 'B').
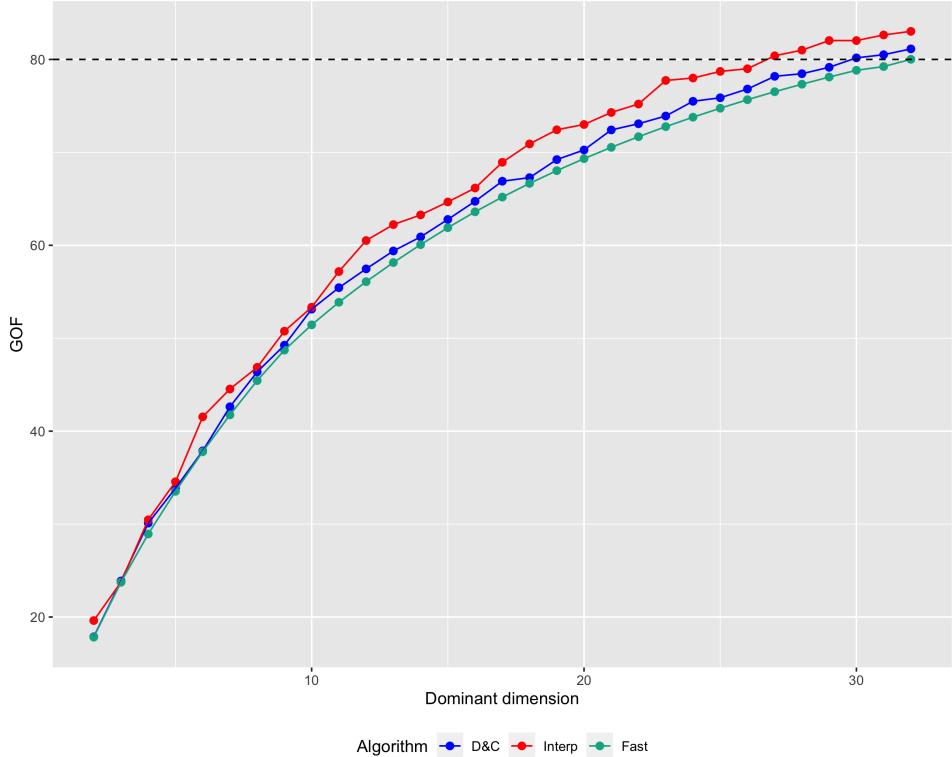
Figure 10: GOF (represented by G_1 metric) as a function of dominant dimension (h) for each algorithm.

rule widely used is to choose h so that the goodness-of-fit reaches a value of 80%. We are interested in answering this question for the EMNIST data set. To be more precise, we want to know what the minimum value of h is needed in order to obtain a goodness-of-fit (G_1 metric) greater than or equal to 80%, i.e, we look for $h^* = \mathrm{argmin}_h\{h \,\text{s.t.}\, G_1(h) \geq 80\%\}$. To find h^*, we compute the value of $G_1(h)$ for every value of h until the desired bound, i.e 80%, is reached. Since each algorithm could reach this bound independently from the others, $G_1(h)$ is calculated for the three algorithms. In Figure 10 a representation of the points $(h, G_1(h))$ is obtained for each of the three algorithms. The value of h^* for divide-and-conquer MDS is equal to 30, for interpolation MDS it is equal to 27 and for fast MDS it is equal to 32. The value for ℓ parameter is the same as used in Section 4.1. Finally, in order to be able to compare the results across all the executions, we fix the values of c and s parameters for divide-and-conquer MDS and fast MDS respectively equal to 100. We use this value since, a priori, we do not know what will be the value of h^*.

6   bigmds: the R package to do MDS with Big Data

In order to make these methods available, we provide an R package which is available in CRAN: `https://cran.r-project.org/web/packages/bigmds` in order to obtain more information about R as well as CRAN).

The core of the package consists of three methods: divide_and_conquer_mds, interpolation_mds and fast_mds. Each of these functions provides an MDS configuration following the procedures described in section 3. We also develop a Procrustes function which is used by the three previous functions. We follow Borg and Groenen (2005) in order to obtain Procrustes parameters. The package has also a development version which is available in GitHub: `https://github.com/pachoning/bigmds`.

As a classical MDS algorithm we use cmdscale function from stats package. Given that interpolation MDS needs a function to calculate the distance matrix $\mathbf{A_{21}}$ (see 3.2 to obtain more details), we use pdist function from pdist package. See Wong (2013) to obtain more details about pdist package.

7   Conclusions

We present three algorithms to obtain an MDS configuration when dealing with large data sets. According to the simulation study, all the algorithms provide low dimensional configuration similar to those eventually given by the classical MDS algorithm. Furthermore, the algorithms considered have two main advantages: first, they are much faster than classical MDS for moderate sample sizes, and second, they are applicable for large data sets for which classical MDS is not feasible.

Considering each algorithm separately, the low dimensional configurations provided by divide-and-conquer MDS or interpolation MDS are closer to the classical MDS solutions than those coming from fast MDS. In terms of speed, the three algorithms provide a low dimensional configuration in a reasonable amount of time, divide-and-conquer MDS being preferred for moderate sample sizes, whereas interpolation MDS is the option at choice for very large data sets.

As a final challenge for the algorithms, we use them to obtain an MDS configuration for the real large data set EMNIST. Since classical MDS algorithm can not be used with this data set, we do not have a gold standard to compare against. However, we consider that the computed goodness-of-fit measures, with values around 18%, meets our expectations given that we use just a 2-dimensional configuration. Additionally, the time needed to obtain the low dimensional configurations is admissible (always below 8.5 minutes for more than $8 \cdot 10^5$ points). In this example, divide-and-conquer MDS has been considerably faster than interpolation MDS (both faster than fast MDS).

As a global conclusion, the three algorithms are suitable for obtaining low dimensional configurations for large data sets, but we recommend to use divide-and-conquer MDS or interpolation MDS, since both are fast and they have satisfactory statistical properties.

Finally, we provide an R package that implements the three MDS algorithms, which we have used to perform all the computations of the paper. An eventual implementation in a low level programming language, such as C, C++ or Java, could speed up the algorithms.

A   Classical Multidimensional Scaling

In this section we follow Chapter 12 of Borg and Groenen (2005). Given a matrix $\mathbf{X}$ of dimension $n \times k$, it is possible to obtain a new one with mean equals to 0 by columns from the previous one: $\widetilde{\mathbf{X}} = \mathbf{PX}$, where

$$\mathbf{P} = \left(\mathbf{I_n} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n^\mathrm{T} \right) \text{ and } \mathbf{1}_n = (1, \ldots, 1)^\mathrm{T}.$$

From matrix $\widetilde{\mathbf{X}}$, it is possible to build two square semi-positive definite matrices: the covariance matrix $\mathbf{S} = \widetilde{\mathbf{X}}^\mathrm{T} \widetilde{\mathbf{X}} / n$, and the cross-products matrix $\mathbf{Q} = \widetilde{\mathbf{X}} \widetilde{\mathbf{X}}^\mathrm{T}$. The element (i, j) of $\mathbf{Q}$ is

$$q_{ij} = \sum_{s=1}^{k} x_{is} x_{js} = \mathbf{x}_i^\mathrm{T} \mathbf{x}_j$$

where $\mathbf{x}_i^\mathrm{T}$ is the $i - th$ row of $\widetilde{\mathbf{X}}$. Given that the scalar product formula is $\mathbf{x}_i^\mathrm{T} \mathbf{x}_j = |\mathbf{x}_i||\mathbf{x}_i| \cos \theta_{ij}$, if the elements i and j have similar coordinates, then $\cos \theta_{ij} \simeq 1$ and q_{ij} will be large. On the contrary, if the elements are very different, then $\cos \theta_{ij} \simeq 0$ and q_{ij} will be small. So, $\widetilde{\mathbf{X}} \widetilde{\mathbf{X}}^\mathrm{T}$ can be interpreted as the similarity matrix between its rows.

The Euclidean distances between the rows of $\widetilde{\mathbf{X}}$ can be deduced from the similarity matrix $\mathbf{Q}$. To show that, consider the squared Euclidean distance between two rows:

$$d_{ij}^2 = \sum_{s=1}^{k} (x_{is} - x_{js})^2 = \sum_{s=1}^{k} x_{is}^2 + \sum_{s=1}^{k} x_{js}^2 - 2 \sum_{s=1}^{k} x_{is} x_{js}.$$

This expression can be obtained directly from the matrix $\mathbf{Q}$ as

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

We have just seen that, given the matrix $\widetilde{\mathbf{X}}$, it is possible to get the similarity matrix $\mathbf{Q} = \widetilde{\mathbf{X}} \widetilde{\mathbf{X}}^\mathrm{T}$ and from it, to get the squared distance matrix, say

$\boldsymbol{\Delta}$ with elements d_{ij}^2. Let $\mathrm{diag}(\mathbf{Q})$ be the vector that contains the diagonal terms of $\mathbf{Q}$. The matrix $\boldsymbol{\Delta}$ is given by

$$\boldsymbol{\Delta} = \mathrm{diag}(\mathbf{Q})\mathbf{1}_n^{\mathrm{T}} + \mathbf{1}_n\,\mathrm{diag}(\mathbf{Q})^{\mathrm{T}} - 2\mathbf{Q}.$$

Let us consider now the reverse problem: to rebuild $\widetilde{\mathbf{X}}$ from a squared distance matrix $\boldsymbol{\Delta}$. We first obtain $\mathbf{Q} = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^{\mathrm{T}}$ from $\boldsymbol{\Delta}$, to get $\widetilde{\mathbf{X}}$ afterwards. Given that the Euclidean distance in $\mathbb{R}^k$ is translation invariant, we can assume, without loss of generality, that the mean of the columns is equal to 0. Therefore we are looking for a matrix $\widetilde{\mathbf{X}}$ such that $\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{1}_n = 0$. This implies that $\mathbf{Q}\mathbf{1}_n = 0$, i.e, the sum of all the elements of a column of $\mathbf{Q}$ is 0. Since the matrix is symmetric, the previous condition should state for the rows as well.

Taking into account these constrains, we sum up (2) at row level:

$$\sum_{i=1}^{n} d_{ij}^2 = \sum_{i=1}^{n} q_{ii} + nq_{jj} = t + nq_{jj} \tag{3}$$

where $t = \sum_{i=1}^{n} q_{ii} = \mathrm{Trace}(\mathbf{Q})$, and we have used that the condition $\mathbf{Q}\mathbf{1}_n = 0$ implies $\sum_{i=1}^{n} q_{ij} = 0$. Summing up (2) at column level

$$\sum_{j=1}^{n} d_{ij}^2 = t + nq_{ii}. \tag{4}$$

Summing up (3) we obtain

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 = 2nt. \tag{5}$$

Replacing in (2) q_{jj} obtained in (3) and q_{ii} obtained in (4), we have the following expression:

$$d_{ij}^2 = \frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 - \frac{t}{n} + \frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 - \frac{t}{n} - 2q_{ij}.$$

Let $d_{i.}^2 = \frac{1}{n}\sum_{j=1}^{n} d_{ij}^2$ and $d_{.j}^2 = \frac{1}{n}\sum_{i=1}^{n} d_{ij}^2$ be the row-mean and column-mean of the elements of $\boldsymbol{\Delta}$. Using (5), we have that

$$d_{ij}^2 = d_{i.}^2 + d_{.j}^2 - d_{..}^2 - 2q_{ij}, \tag{6}$$

where $d_{..}^2$ is the mean of all the elements of $\boldsymbol{\Delta}$, given by

$$d_{..}^2 = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2.$$

Finally, from (6) we get the expression:

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

The previous expression shows how to build the matrix of similarities $\mathbf{Q}$ from the squared distance matrix $\mathbf{\Delta}$. Using matrix notation, we have that

$$\mathbf{Q} = -\frac{1}{2}\mathbf{P}\mathbf{\Delta}\mathbf{P}.$$

The next step is to recover the matrix $\widetilde{\mathbf{X}}$ given the matrix $\mathbf{Q}$. Since $\mathbf{P}\mathbf{1}_n = 0$, range$(\mathbf{Q}) \leq n - 1$, being the vector $\mathbf{1}_n$ an eigenvector with eigenvalue 0. Let's assume that the similarity matrix is positive semidefinite with range $\tau \leq n - 1$. Therefore, it can be represented as

$$\mathbf{Q} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} = \sum_{i=1}^{\tau} \lambda_i \mathbf{v}_i \mathbf{v}_i^{\mathrm{T}},$$

where $\mathbf{\Lambda}$ is the diagonal matrix $\tau \times \tau$ that contains the non-null eigenvalues λ_i of $\mathbf{Q}$ in the diagonal, and $\mathbf{V}$ is a $n \times \tau$ matrix that contains the corresponding eigenvectors $\mathbf{v}_i$ of $\mathbf{Q}$. Re-writing the previous expression, we obtain

$$\mathbf{Q} = (\mathbf{V}\mathbf{\Lambda}^{1/2})(\mathbf{\Lambda}^{1/2}\mathbf{V}^{\mathrm{T}}). \tag{7}$$

Getting

$$\widetilde{\mathbf{X}} = \mathbf{V}\mathbf{\Lambda}^{1/2}$$

we have obtained a matrix with dimensions $n \times \tau$ with τ centered uncorrelated variables that reproduce the initial metric.

By (2), the distance is a function of the terms of the similarity matrix $\mathbf{Q}$ and this matrix is invariant given any rotation or reflection of the variables

$$\mathbf{Q} = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^{\mathrm{T}} = \widetilde{\mathbf{X}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\widetilde{\mathbf{X}}^{\mathrm{T}}$$

for any orthogonal $\mathbf{A}$ matrix. So the centered orthogonal configuration $\widetilde{\mathbf{X}}$ is not unique.

B   Procrustes transformation

The MDS solution is not unique: since orthogonal transformations as well as translations are distance-preserving functions, one can find different MDS configurations for the same data set. Therefore, given two MDS configurations, say $\mathbf{X_1}$ and $\mathbf{X_2}$, both in $\mathbb{R}^{n \times r}$, we are interested in finding an orthogonal matrix $\mathbf{T} \in \mathbb{R}^{r \times r}$ ($\mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{I_r}$) and a vector $\mathbf{t} \in \mathbb{R}^r$ such that $\mathbf{X_1} \approx \mathbf{X_2}\mathbf{T} + \mathbf{1}_n\mathbf{t}^{\mathrm{T}}$. In this section, we provide details on how to proceed in order to obtain the Procrustes parameters $(\mathbf{T}, \mathbf{t})$.

We follow Chapter 20 of Borg and Groenen (2005) for the next explanations of Procrustes transformation, which we divide into two parts: in the first part, we do not allow translation, just orthogonal transformations. In the second one, we allow both transformations. So, in the first part, we are searching for an orthogonal matrix $\mathbf{T} \in \mathbb{R}^{r \times r}$ such that $\mathbf{X_1} \approx \mathbf{X_2}\mathbf{T}$.

B.1 Procrustes without translation

Formally speaking, by "$\approx$" it is meant to find a matrix $\mathbf{T} \in \mathbb{R}^{r \times r}$ such that minimizes the distance between the points in $\mathbf{X_1}$ and $\mathbf{X_2}\mathbf{T}$, expressed as a function of $\mathbf{T}$:

$$L(\mathbf{T}) = \text{Trace}\left[(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})^{\mathrm{T}}(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})\right], \tag{8}$$

subject to $\mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{I_r}$. Notice that the root squared of $L(\mathbf{T})$ is known as the *Frobenius norm* (of the matrix $\mathbf{X_1} - \mathbf{X_2}\mathbf{T}$). This function can be seen as a *loss function*.

Expanding the expression (8) we get

$$
\begin{aligned}
L(\mathbf{T}) &= \text{Trace}\left[(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})^{\mathrm{T}}(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})\right] \\
&= \text{Trace}(\mathbf{X_1}^{\mathrm{T}}\mathbf{X_1}) + \text{Trace}(\mathbf{X_2}^{\mathrm{T}}\mathbf{X_2}) - 2\,\text{Trace}(\mathbf{X_1}^{\mathrm{T}}\mathbf{X_2}\mathbf{T})
\end{aligned} \tag{9}
$$

subject to $\mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{I_r}$. In order to obtain the expression (9) we use the property of invariance of the trace under cyclic permutation, which states that $\text{Trace}\left[\mathbf{T}^{\mathrm{T}}\mathbf{X_2}^{\mathrm{T}}\mathbf{X_2}\mathbf{T}\right] = \text{Trace}\left[\mathbf{X_2}^{\mathrm{T}}\mathbf{X_2}\mathbf{T}\mathbf{T}^{\mathrm{T}}\right]$. Afterwards, we use that $\mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{I_r}$.

Since neither $\text{Trace}(\mathbf{X_1}^{\mathrm{T}}\mathbf{X_1})$ nor $\text{Trace}(\mathbf{X_2}^{\mathrm{T}}\mathbf{X_2})$ depend on $\mathbf{T}$, the problem can be reformulated as finding a minimum value for the function

$$L(\mathbf{T}) = c - 2\,\text{Trace}(\mathbf{X_1}^{\mathrm{T}}\mathbf{X_2}\mathbf{T}), \tag{10}$$

subject to $\mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{I_r}$, where c is a constant term independent of $\mathbf{T}$.

In order to find a minimum value for the function (10), we apply Kristof (1970) theorem, which states that if $\mathbf{Y}$ is a diagonal matrix with nonnegative entries and $\mathbf{R}$ is a orthogonal matrix, then

$$-\text{Trace}(\mathbf{R}\mathbf{Y}) \geq -\text{Trace}(\mathbf{Y}),$$

with equality if and only if $\mathbf{R} = \mathbf{I}$.

In order to apply Kristof's theorem, let us consider the singular value decomposition of $\mathbf{X_1}^{\mathrm{T}}\mathbf{X_2}$: $\mathbf{\Gamma}\mathbf{\Phi}\mathbf{\Omega}^{\mathrm{T}}$, where $\mathbf{\Gamma}^{\mathrm{T}}\mathbf{\Gamma} = \mathbf{I}$, $\mathbf{\Omega}^{\mathrm{T}}\mathbf{\Omega} = \mathbf{I}$, and $\mathbf{\Phi}$ is the diagonal matrix with the singular values. Therefore, $L(\mathbf{T})$ can be written as $c - 2\,\text{Trace}(\mathbf{\Gamma}\mathbf{\Phi}\mathbf{\Omega}^{\mathrm{T}}\mathbf{T})$. Using the invariance of the trace under cyclic permutation we obtain

$$L(\mathbf{T}) = c - 2\,\text{Trace}(\mathbf{\Omega}^{\mathrm{T}}\mathbf{T}\mathbf{\Gamma}\mathbf{\Phi}). \tag{11}$$

Given that $\mathbf{\Gamma}^{\mathrm{T}}\mathbf{\Gamma} = \mathbf{\Omega}^{\mathrm{T}}\mathbf{\Omega} = \mathbf{I}$ and that $\mathbf{T}$ is an orthonormal matrix, it can be check that $\mathbf{\Omega}^{\mathrm{T}}\mathbf{T}\mathbf{\Gamma}$ is also an orthonormal matrix. Therefore, we can apply Kristof's theorem considering $\mathbf{R} = \mathbf{\Omega}^{\mathrm{T}}\mathbf{T}\mathbf{\Gamma}$ and $\mathbf{Y} = \mathbf{\Phi}$. This means that $L(T) \geq c - 2\operatorname{Trace}(\mathbf{\Phi})$. In addition, $L(\mathbf{T})$ is minimal if and only if $\mathbf{R} = \mathbf{I}$, i.e, $\mathbf{\Omega}^{\mathrm{T}}\mathbf{T}\mathbf{\Gamma} = \mathbf{I}$. Choosing $\mathbf{T} = \mathbf{\Omega}\mathbf{\Gamma}^{\mathrm{T}}$, we obtain the minimum value for $L(\mathbf{T})$.

B.2 Procrustes with translation

In the second part, we describe how to obtain the Procrustes parameters allowing translations (general case). To begin with, let consider the loss function $L(\mathbf{T}, \mathbf{t})$ defined as:

$$L(\mathbf{T}, \mathbf{t}) = \operatorname{Trace}\left[(\mathbf{X_1} - (\mathbf{X_2} + \mathbf{1}_n\mathbf{t}^{\mathrm{T}}))^{\mathrm{T}}(\mathbf{X_1} - (\mathbf{X_2} + \mathbf{1}_n\mathbf{t}^{\mathrm{T}}))\right],$$

subject to $\mathbf{T}$ is an orthonormal matrix of size $r \times r$, $\mathbf{t} \in \mathbb{R}^r$. An optimal value for $\mathbf{t}$ is obtained by setting the derivative of $L(\mathbf{T}, \mathbf{t})$ with respect the parameter $\mathbf{t}$ equal to $\mathbf{0}_r$. The equation $\partial \mathbf{L}(\mathbf{T}, \mathbf{t})/\partial\mathbf{t} = \mathbf{0}_r$ drives to obtain a value for $\mathbf{t}$ equal to $\mathbf{t} = (1/n)(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})^{\mathrm{T}}\mathbf{1}_n$. Replacing the value of $\mathbf{t}$ directly in the expression of $L(\mathbf{T}, \mathbf{t})$ gives

$$L(\mathbf{T}) = \operatorname{Trace}\left[(\mathbf{P}\mathbf{X_1} - \mathbf{P}\mathbf{X_2}\mathbf{T})^{\mathrm{T}}(\mathbf{P}\mathbf{X_1} - \mathbf{P}\mathbf{X_2}\mathbf{T})\right], \tag{12}$$

with $\mathbf{P} = \mathbf{I_n} - (1/n)\mathbf{1}_n\mathbf{1}_n^{\mathrm{T}}$ (the centering matrix) and, as before, subject to $\mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{I_r}$. Notice that $\mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{P}$. Using that:

- $\mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{I_r}$,

- $\mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{P}$,

- $\mathbf{P} = \mathbf{P}^{\mathrm{T}}$,

- $\operatorname{Trace}\left[\mathbf{T}^{\mathrm{T}}\mathbf{X_2}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}\mathbf{T}\right] = \operatorname{Trace}\left[\mathbf{X_2}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}\right]$ and

- $\operatorname{Trace}\left[\mathbf{T}^{\mathrm{T}}\mathbf{X_2}^{\mathrm{T}}\mathbf{P}\mathbf{X_1}\right] = \operatorname{Trace}\left[\mathbf{X_1}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}\mathbf{T}\right]$,

the expression (12) can be written as

$$L(\mathbf{T}) = \operatorname{Trace}\left[\mathbf{X_1}^{\mathrm{T}}\mathbf{P}\mathbf{X_1}\right] + \operatorname{Trace}\left[\mathbf{X_2}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}\right] - 2\operatorname{Trace}\left[\mathbf{X_1}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}\mathbf{T}\right]. \tag{13}$$

The same procedure as applied in the optimization problem (11) in Appendix B.1 can be applied in the optimization problem (13) in order to find the matrix $\mathbf{T}$ that minimizes the expression (13).

The process described previously can be used to obtain the optimal Procrustes parameters. These steps can be summarized as:

- Compute $\mathbf{\Psi} = \mathbf{X_1}^{\mathrm{T}}\mathbf{P}\mathbf{X_2}$, where $\mathbf{P}$ is the centering matrix of size $n \times n$ $\mathbf{P} = \mathbf{I_n} - (1/n)\mathbf{1}_n\mathbf{1}_n^{\mathrm{T}}$.

- Compute the SVD of $\mathbf{\Psi}$; that is, $\mathbf{\Psi} = \mathbf{\Gamma}\mathbf{\Phi}\mathbf{\Omega}^{\mathrm{T}}$.

- The optimal value for the rotation matrix $\mathbf{T}$ is $\mathbf{\Omega}\mathbf{\Gamma}^{\mathrm{T}}$.

- The optimal value for the translation vector $\mathbf{t}$ is $(1/n)(\mathbf{X_1} - \mathbf{X_2}\mathbf{T})^{\mathrm{T}}\mathbf{1}_n$.

Acknowledgments

References

Bentley, J. L., D. Haken, and J. B. Saxe (1980). A general method for solving divide-and-conquer recurrences. *ACM SIGACT News 12*(3), 36–44.

Borg, I. and P. Groenen (2005). *Modern Multidimensional Scaling: Theory and Applications*. Springer.

Cohen, G., S. Afshar, J. Tapson, and A. V. Schaik (2017). Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*.

Gower, J. C. and D. J. Hand (1995). *Biplots*, Volume 54. CRC Press.

Grother, P. (1970). Nist special database 19. nist handprinted forms and characters database.

Johnson, R. A. and D. W. Wichern (2002). *Applied Multivariate Statistical Analysis* (5th ed.). Prentice Hall.

Kristof, W. (1970). A theorem on the trace of certain matrix products and some applications. *Journal of Mathematical Psychology 7*(3), 515–530.

Krzanowski, W. (2000). *Principles of Multivariate Analysis* (Revised ed.), Volume 23 of *Oxford Statistical Science Series*. OUP Oxford.

LeCun, Y. and C. Cortes (2010). MNIST handwritten digit database.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Trefethen, L. N. and D. Bau (1997). *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics.

Wong, J. (2013). *pdist: Partitioned Distance Function*. R package version 1.2.

Yang, T., J. Liu, L. McMillan, and W. Wang (2006). A fast approximation to multidimensional scaling. In *Proceedings of the ECCV Workshop on Computation Intensive Methods for Computer Vision (CIMCV)*.