Multidimensional Scaling for Big Data

Cristian Pachón García
Advisor: Pedro Delicado Useros

January 17, 2019

What is MDS?

- MDS is a statistic tool for reduction of dimensionality, using as input a distance matrix.
- Given a square matrix $\mathbf{D}$ $n \times n$, the goal of MDS is to obtain a configuration matrix $\mathbf{X}$ $n \times q$ satisfying:
  - Columns are orthogonal.
  - The Euclidean distance between the rows of $\mathbf{X}$ is approximately equal to $\mathbf{D}$.
- $\mathbf{X}$ can be interpreted as the matrix of q **latent** variables for the n observations.
- The columns of $\mathbf{X}$ are called *principal coordinates*.

Example

Consider the distance between some cities of Europe, as shown in the following matrix:

	Athens	Barcelona	Brussels	Calais	Cherbourg	$\cdots$
Athens	0	3313	2963	3175	3339	$\cdots$
Barcelona	3313	0	1318	1326	1294	$\cdots$
Brussels	2963	1318	0	204	583	$\cdots$
Calais	3175	1326	204	0	460	$\cdots$
Cherbourg	3339	1294	583	460	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table: Distances between European cities (just 5 of them are shown).

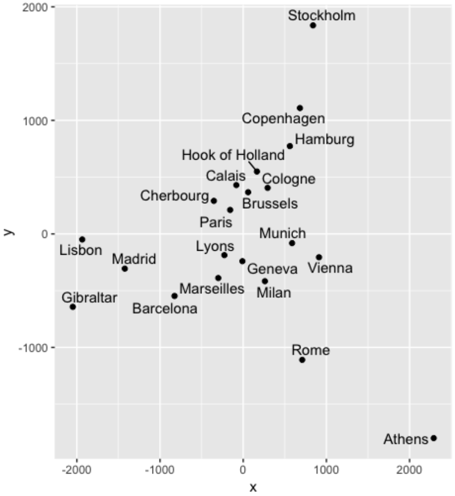


Figure: MDS configuration for European cities

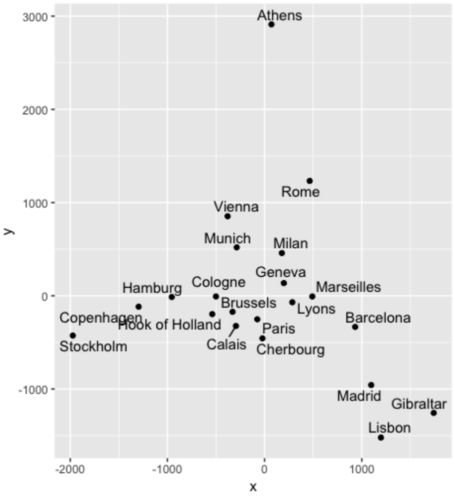


Figure: (Another) MDS configuration for European cities

Procrustes transformation

- Given a MDS configuration, any rotation, reflection or translation is a valid MDS configuration, since they preserve the distance. So, the solution is not unique.
- Procrustes transformation problem:
  - Let $\mathbf{A}$ and $\mathbf{B}$ be two different MDS configurations for the same set of data. One wants to obtain s, $\mathbf{T}$ and $\mathbf{t}$ such that

$$\mathbf{A} = s\mathbf{B}\mathbf{T} + \mathbf{1}\mathbf{t}',$$

    where $\mathbf{T}$ is an orthogonal matrix.
  - In Borg and Groenen (2005) are all the details needed to estimate these parameters.

1 Introduction

2 Algorithms for MDS with Big Data

3 Simulation study
   Design of the simulation
   Results

4 Conclusions

Why is it needed?

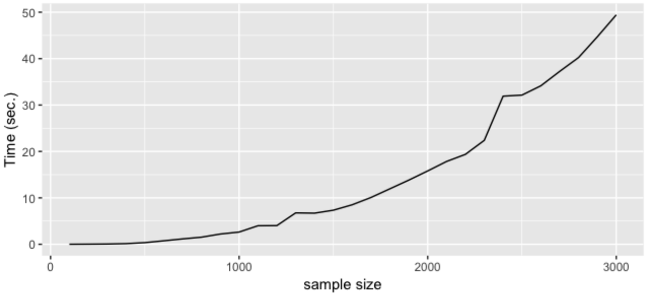


Figure: Elapsed time to compute MDS.

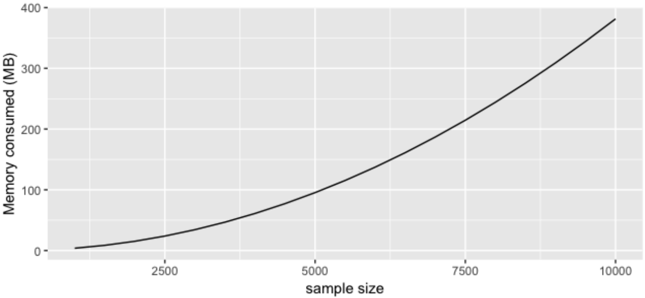
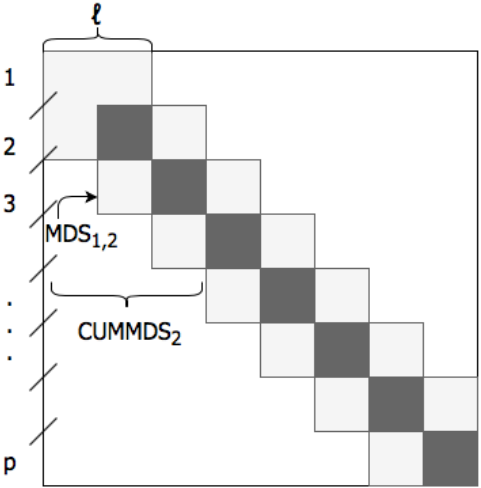


Figure: Memory consumed to compute the distance matrix.

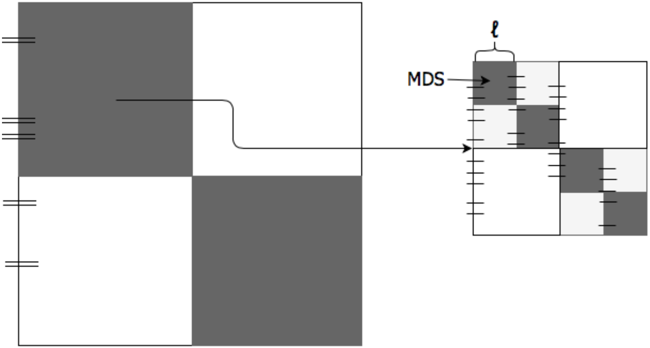
Three algorithms for MDS with Big Data

- *Divide and Conquer MDS:*
  - First approach of this thesis.
- *Fast MDS:*
  - It uses recursive programming.
  - Developed by Tynia, Jinze, Leonard, and Wei (2006).
- *MDS based on Gower interpolation:*
  - It adds a new set of points to an existing MDS configuration.
  - See Appendix of (Gower and Hand 1995).

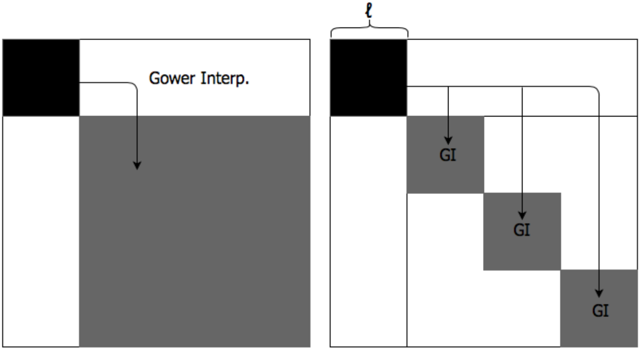
Divide and Conquer MDS



Fast MDS



MDS based on Gower interpolation



Output of the algorithms

The three algorithms have the same type of output. It consists on a list of two parameters, which are

- A MDS configuration for the initial dataset.
- The second parameter is a list of eigenvalues.

Simulation study

Given the three algorithms, we would like to explore their performance:

- Performance in terms of results quality: are they able to capture the right data dimensionality?
- Performance in terms of time: are they " fast" enough? Which one is the fastest?

1. Introduction

2. Algorithms for MDS with Big Data

3. Simulation study
   - Design of the simulation
   - Results

4. Conclusions

Design of the simulation

- *Sample sizes*: A total of six sample sizes are used, which are:
  - Small sample sizes: $10^3, 3 \cdot 10^3, 5 \cdot 10^3$ and 10^4.
  - Large sample sizes: 10^5 and 10^6.
- *Data dimensions*: we generate a matrix with two different number of columns: 10 and 100.
- *Main dimensions*: Postmultiplication by a diagonal matrix:
  - Identity matrix (*noisy scenarios*).
  - One main dimension with λ: 15.
  - Two main dimensions of the same value λ: 15.
  - Two main dimensions of different values λ: 15, 10.
  - Four main dimensions of the same value λ: 15.
- As a probabilistic model, we use a Normal distribution.
- Given a scenario, it is replicated 100 times.

1. Generate the dataset $\mathbf{X}$ according to the scenario.
2. For each algorithm, we do the following steps:
    1. Run the algorithm and get MDS configuration for the algorithm ($\mathbf{MDS}_{\text{alg}}$).
    2. Get the elapsed time to compute MDS configuration and store it.
    3. Get eigenvalues and store them.
    4. Align $\mathbf{MDS}_{\text{alg}}$ and $\mathbf{X}$ using (Partition) Procrustes.
    5. Get the correlation coefficients between the main dimensions of $\mathbf{MDS}_{\text{alg}}$ and $\mathbf{X}$ and store them.

- Performance of results quality:
  - Correlation between the main dimensions of the data and the main dimensions after applying the algorithms. We get the diagonal of the correlation matrix.
  - *Eigenvalues* as an approximation of the standard deviation of the variables of $\mathbf{X}$.
- Elapsed Time to get the MDS configuration.

1. Introduction
2. Algorithms for MDS with Big Data
3. Simulation study
   - Design of the simulation
   - Results
4. Conclusions

Correlation coefficients

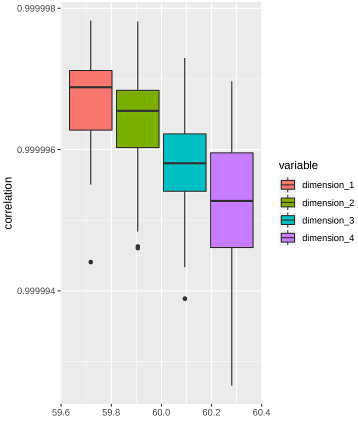


Figure: Correlation coefficients between the main dimensions of $\mathbf{X}$ and the main dimensions of $\mathbf{MDS_{Div}}$ for a scenario with $n = 10^6$, 100 columns and 4 main dimensions with $\lambda = 15$.

Results

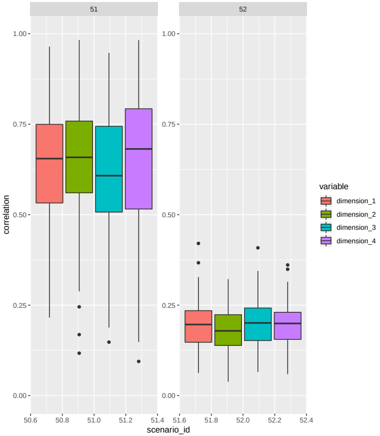


Figure: Correlation coefficients between **X** and $\mathbf{MDS_{Div}}$ for two different *noise scenarios*

Eigenvalues

- Since the original dataset, $\mathbf{X}$, is postmultiplied by a diagonal matrix $k \times k$ that contains $\lambda_1, \ldots, \lambda_k$, then $\text{var}(X_i) = \lambda_i^2$ and $\text{sd}(X_i) = \lambda_i$.
- Let $\phi_1, \ldots, \phi_t$ be the *eigenvalues* of the MDS configuration such that $\phi_1 > \phi_2 > \cdots > \phi_t$. The first highest *eigenvalues* have to verify $\sqrt{\phi_j} \approx \lambda_j$.
- We compute the bias.

sample_size	n_dim	Div. Conq. MDS $\widehat{\sqrt{\phi}}$	Div. Conq. MDS $\widehat{\text{bias}}$	Fast $\widehat{\sqrt{\phi}}$	Fast $\widehat{\text{bias}}$	Gower MDS $\widehat{\sqrt{\phi}}$	Gower MDS $\widehat{\text{bias}}$
10^3	10	14.98	-0.02	15.85	0.15	15.05	0.05
10^3	100	15.03	0.03	15.01	0.01	15.02	0.02
$3 \cdot 10^3$	10	15.00	-0.00	14.91	-0.09	14.04	-0.06
$3 \cdot 10^3$	100	14.96	-0.04	15.10	0.10	15.04	0.04
$5 \cdot 10^3$	10	14.99	-0.01	14.96	-0.04	14.98	-0.02
$5 \cdot 10^3$	100	14.99	-0.01	15.03	0.03	15.02	0.02
10^4	10	14.99	-0.01	14.33	-0.67	14.99	-0.01
10^4	100	14.99	-0.01	15.09	0.09	15.06	0.06
10^5	10	14.99	-0.01	15.00	0.00	15.04	0.04
10^5	100	14.99	-0.01	15.00	0.00	14.97	-0.03
10^6	10	14.98	-0.02	14.86	-0.14	14.98	-0.02
10^6	100	14.99	-0.01	14.90	-0.10	14.90	-0.10

Table: Estimator and $\widehat{\text{bias}}$ for scenarios with one main dimension $\lambda = 15$.

Time to compute MDS

sample_size	n_dim	mean_divide_conquer	mean_fast	mean_gower
10^3	10	0.27	0.14	0.10
10^3	100	0.78	0.69	0.28
$3 \cdot 10^3$	10	0.78	0.32	0.16
$3 \cdot 10^3$	100	2.50	3.14	0.52
$5 \cdot 10^3$	10	1.37	0.54	0.20
$5 \cdot 10^3$	100	4.25	5.69	0.84
10^4	10	2.60	1.81	0.31
10^4	100	8.85	11.79	1.37
10^5	10	28.10	11.46	2.44
10^5	100	106.30	116.46	18.02
10^6	10	420.29	106.59	53.15
10^6	100	2365.46	1070.19	813.15

Table: Mean of elapsed time (in seconds) to compute each algorithm.

- We do an ANOVA test using three factors:
  - The sample size, which has 6 levels.
  - The number of dimensions, which has 2 levels.
  - The algorithm, which has 3 levels.
- Instead of using the *elapsed time* variable, we use its logarithm.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.4058	0.0085	-165.44	$< 2e-16$
algorithmfast	-0.4313	0.0069	-62.17	$< 2e-16$
algorithmgower	-1.6926	0.0069	-243.96	$< 2e-16$
sample_size3000	0.9473	0.0098	96.54	$< 2e-16$
sample_size5000	1.4434	0.0098	147.10	$< 2e-16$
sample_size10000	2.1505	0.0098	219.17	$< 2e-16$
sample_size1e+05	4.4286	0.0098	451.35	$< 2e-16$
sample_size1e+06	7.2782	0.0098	741.78	$< 2e-16$
n_dimensions100	1.6910	0.0057	298.51	$< 2e-16$

Table: Linear model for response log(elapsed_time).

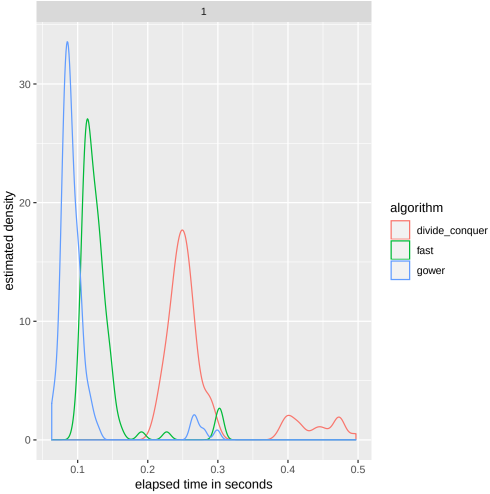


Figure: Estimated density of elapsed time (in sec.) for each algorithm and scenario with $n = 10^3$, 10 columns and $\lambda_i = 1$ $i \in \{1, \ldots, 10\}$.

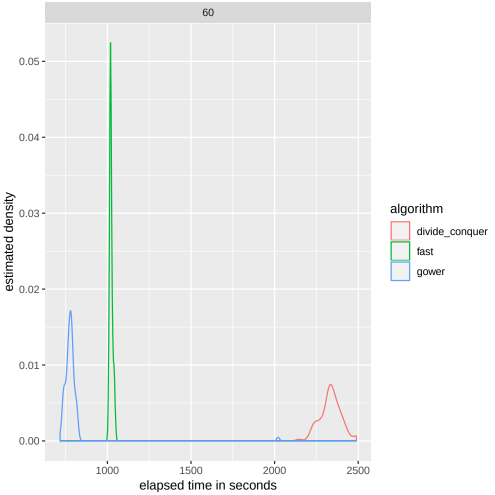


Figure: Estimated density of elapsed time (in sec.) for each algorithm and scenario with $n = 10^6$, 100 columns and 4 main dimensions with $\lambda = 15$.

Conclusions

- The fastest algorithm is *MDS based on Gower interpolation*.
- *Fast MDS* is able to obtain a MDS configuration in a reasonable amount of time.
- The best algorithm capturing the variance of the original dataset is *Divide and Conquer MDS*.
- *MDS based on Gower interpolation* is the best choice, since it is the fastest algorithm and its results quality are good.
- This algorithm does essentially what classical Statistics advises: when your population is too large, take a sample of it.

Next steps

- Modify *Divide and Conquer* in order to use less points for Procrustes.
- Use *real datasets* with the algorithms.
- Build and R library.

Thank You

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

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

Tynia, Y., L. Jinze, M. Leonard, and W. Wei (2006).
A fast approximation to multidimensional scaling.