Package 'bigmds'

October 12, 2022

Title Multidimensional Scaling for Big Data

Version 2.0.1

Description MDS is a statistic tool for reduction of dimensionality, using as input a distance matrix of dimensions n × n. When n is large, classical algorithms suffer from computational problems and MDS configuration can not be obtained.
With this package, we address these problems by means of three algorithms:
- Divide-and-conquer MDS proposed by Delicado P. and C. Pachón-García (2021) <arXiv:2007.11919>.
- Interpolation MDS, also proposed by Delicado P. and C. Pachón-García (2021) <arXiv:2007.11919>, which uses Gower's interpolation formula as described in Gower, J. C. and D. J. Hand (1995).
- Fast MDS, which is an implementation of the algorithm proposed by Yang, T., J. Liu, L. McMillan, and W. Wang (2006).

The main idea of these algorithms is based on partitioning the data set into small pieces, where classical methods can work. In order to align all the solutions, Procrustes formula is used as described in Borg, I. and P. Groenen (2005).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Imports stats, parallel

Suggests testthat

URL https://github.com/pachoning/bigmds

BugReports https://github.com/pachoning/bigmds/issues

NeedsCompilation no

Author Cristian Pachón García [aut, cre] (<https://orcid.org/0000-0001-9518-4874>), Pedro Delicado [aut] (<https://orcid.org/0000-0003-3933-4852>)

Maintainer Cristian Pachón García <cc.pachon@gmail.com>

Repository CRAN

Date/Publication 2021-10-05 10:00:02 UTC

R topics documented:

divide_conquer_mds 2
fast_mds 3
interpolation_mds 5

Index **8**

`divide_conquer_mds` *Divide-and-conquer MDS*

Description

Roughly speaking, a large data set, `x`, of size n is divided into parts, then classical MDS is performed over every part and, finally, the partial configurations are combined so that all the points lie on the same coordinate system with the aim to obtain a global MDS configuration.

Usage

```
divide_conquer_mds(x, l, c_points, r, n_cores = 1, dist_fn = stats::dist, ...)
```

Arguments

`x`	A matrix with n points (rows) and k variables (columns).
`l`	The size for which classical MDS can be computed efficiently (using `cmdscale` function). It means that if $\bar{l}$ is the limit size for which classical MDS is applicable, then `l`$\leq \bar{l}$.
`c_points`	Number of points used to align the MDS solutions obtained by the division of `x` into p data subsets. Recommended value: `2·r`.
`r`	Number of principal coordinates to be extracted.
`n_cores`	Number of cores wanted to use to run the algorithm.
`dist_fn`	Distance function used to compute the distance between the rows.
`...`	Further arguments passed to `dist_fn` function.

Details

The divide-and-conquer MDS starts dividing the n points into p partitions: the first partition contains `l` points and the others contain `l-c_points` points. Therefore, $p = 1+(n-$`l`$)/($`l-c_points`$)$. The partitions are created at random.

Once the partitions are created, `c_points` different random points are taken from the first partition and concatenated to the other partitions After that, classical MDS is applied to each partition, with target low dimensional configuration `r`.

Since all the partitions share `c_points` points with the first one, Procrustes can be applied in order to align all the configurations. Finally, all the configurations are concatenated in order to obtain a global MDS configuration.

Value

Returns a list containing the following elements:

points A matrix that consists of n points (rows) and `r` variables (columns) corresponding to the principal coordinates. Since a dimensionality reduction is performed, `r`$<< k$

eigen The first `r` largest eigenvalues: $\bar{\lambda}_i, i \in \{1, \ldots, r\}$, where $\bar{\lambda}_i = 1/p \sum_{j=1}^{p} \lambda_i^j / n_j$, being λ_i^j the $i-th$ eigenvalue from partition j and n_j the size of the partition j.

GOF A numeric vector of length 2.

The first element corresponds to $1/n \sum_{j=1}^{p} n_j G_1^j$, where $G_1^j = \sum_{i=1}^{r} \lambda_i^j / \sum_{i=1}^{n-1} |\lambda_i^j|$.

The second element corresponds to $1/n \sum_{j=1}^{p} n_j G_2^j$ where $G_2^j = \sum_{i=1}^{r} \lambda_i^j / \sum_{i=1}^{n-1} max(\lambda_i^j, 0)$.

References

Delicado P. and C. Pachón-García (2021). *Multidimensional Scaling for Big Data*. https://arxiv.org/abs/2007.11919.

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

Examples

```
set.seed(42)
x <- matrix(data = rnorm(4*10000), nrow = 10000) %*% diag(c(9, 4, 1, 1))
mds <- divide_conquer_mds(x = x, l = 200, c_points = 2*2, r = 2, n_cores = 1, dist_fn = stats::dist)
head(mds$points)
mds$eigen
mds$GOF
points <- mds$points
plot(x[1:10, 1],
     x[1:10, 2],
     xlim = range(c(x[1:10,1],points[1:10,1])),
     ylim = range(c(x[1:10,2], points[1:10,2])),
     pch = 19,
     col = "green")
text(x[1:10, 1], x[1:10, 2], labels=1:10)
points(points[1:10, 1], points[1:10, 2], pch = 19, col = "orange")
text(points[1:10, 1], points[1:10, 2], labels=1:10)
abline(v = 0, lwd=3, lty=2)
abline(h = 0, lwd=3, lty=2)
```

`fast_mds` *Fast MDS*

Description

Fast MDS uses recursive programming in combination with a divide and conquer strategy in order to obtain an MDS configuration for a given large data set `x`.

Usage

```
fast_mds(x, l, s_points, r, n_cores = 1, dist_fn = stats::dist, ...)
```

Arguments

x	A matrix with n individuals (rows) and k variables (columns).
l	The size for which classical MDS can be computed efficiently (using cmdscale function). It means that if $\bar{l}$ is the limit size for which classical MDS is applicable, then l$\leq \bar{l}$.
s_points	Number of points used to align the MDS solutions obtained by the division of x into p submatrices. Recommended value: 2$\cdot$r.
r	Number of principal coordinates to be extracted.
n_cores	Number of cores wanted to use to run the algorithm.
dist_fn	Distance function used to compute the distance between the rows.
...	Further arguments passed to dist_fn function.

Details

Fast MDS randomly divides the whole sample data set, x, of size n into p =l/s_points data subsets, where l $\leq \bar{l}$ being $\bar{l}$ the limit size for which classical MDS is applicable. Each one of the p data subsets has size $\tilde{n} = n/p$. If $\tilde{n} \leq$ l 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.

In order to align all the partial solutions, a small subset of size s_points is randomly selected from each data subset. They are joined to form an alignment set, over which classical MDS is performed giving rise to an alignment configuration. Every data subset shares s_points points with the alignment set. Therefore every MDS configuration can be aligned with the alignment configuration using a Procrustes transformation.

Value

Returns a list containing the following elements:

points A matrix that consists of n individuals (rows) and r variables (columns) corresponding to the principal coordinates. Since we are performing a dimensionality reduction, r$<<k$

eigen The first r largest eigenvalues: $\bar{\lambda}_i, i \in \{1, \ldots, r\}$, where $\bar{\lambda}_i = 1/p \sum_{j=1}^{p} \lambda_i^j / n_j$, being λ_i^j the $i-th$ eigenvalue from partition j and n_j the size of the partition j.

GOF A numeric vector of length 2.

The first element corresponds to $1/n \sum_{j=1}^{p} n_j G_1^j$, where $G_1^j = \sum_{i=1}^{r} \lambda_i^j / \sum_{i=1}^{n-1} |\lambda_i^j|$.

The second element corresponds to $1/n \sum_{j=1}^{p} n_j G_2^j$ where $G_2^j = \sum_{i=1}^{r} \lambda_i^j / \sum_{i=1}^{n-1} max(\lambda_i^j, 0)$.

References

Delicado P. and C. Pachón-García (2021). *Multidimensional Scaling for Big Data*. https://arxiv.org/abs/2007.11919.

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).

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

Examples

```
set.seed(42)
x <- matrix(data = rnorm(4*10000), nrow = 10000) %*% diag(c(9, 4, 1, 1))
mds <- fast_mds(x = x, l = 200, s_points = 2*2, r = 2, n_cores = 1, dist_fn = stats::dist)
head(mds$points)
mds$eigen
mds$GOF
points <- mds$points
plot(x[1:10, 1],
     x[1:10, 2],
     xlim = range(c(x[1:10,1],points[1:10,1])),
     ylim = range(c(x[1:10,2], points[1:10,2])),
     pch = 19,
     col = "green")
text(x[1:10, 1], x[1:10, 2], labels=1:10)
points(points[1:10, 1], points[1:10, 2], pch = 19, col = "orange")
text(points[1:10, 1], points[1:10, 2], labels=1:10)
abline(v = 0, lwd=3, lty=2)
abline(h = 0, lwd=3, lty=2)
```

`interpolation_mds` *Interpolation MDS*

Description

Given that the size of the data set is too large, this algorithm consists of taking a random sample from it of size $l \leq \bar{l}$, being $\bar{l}$ the limit 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, which allows to add a new set of points to an existing MDS configuration.

Usage

```
interpolation_mds(x, l, r, n_cores = 1, dist_fn = stats::dist, ...)
```

Arguments

x	A matrix with n individuals (rows) and k variables (columns).
l	The size for which classical MDS can be computed efficiently (using cmdscale function). It means that if $\bar{l}$ is the limit size for which classical MDS is applicable, then l$\leq \bar{l}$.
r	Number of principal coordinates to be extracted.
n_cores	Number of cores wanted to use to run the algorithm.
dist_fn	Distance function used to compute the distance between the rows.
...	Further arguments passed to dist_fn function.

Details

Gower's interpolation formula is the central piece of this algorithm since it allows to add a new set of points to an existing MDS configuration so that the new one has the same coordinate system.

Given the matrix x with n points (rows) and and k variables (columns), a first data subsets (based on a random sample) of size l is taken and it is used to compute a MDS configuration.

The remaining part of x is divided into $p = (n-\text{l})/\text{l}$ data subsets (randomly). For every data subset, it is obtained a MDS configuration by means of *Gower's interpolation formula* and the first MDS configuration obtained previously. Every MDS configuration is appended to the existing one so that, at the end of the process, a global MDS configuration for x is obtained.

Value

Returns a list containing the following elements:

points A matrix that consists of n individuals (rows) and r variables (columns) corresponding to the principal coordinates. Since we are performing a dimensionality reduction, r$<<k$

eigen The first r largest eigenvalues: $\lambda_i, i \in \{1, \ldots, r\}$, where each λ_i is obtained from applying classical MDS to the first data subset.

GOF A numeric vector of length 2.

The first element corresponds to $\sum_{i=1}^{r} \lambda_i / \sum_{i=1}^{n-1} |\lambda_i|$.

The second element corresponds to $\sum_{i=1}^{r} \lambda_i / \sum_{i=1}^{n-1} max(\lambda_i, 0)$.

References

Delicado P. and C. Pachón-García (2021). *Multidimensional Scaling for Big Data*. https://arxiv.org/abs/2007.11919.

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

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

Examples

```
set.seed(42)
x <- matrix(data = rnorm(4*10000), nrow = 10000) %*% diag(c(9, 4, 1, 1))
mds <- interpolation_mds(x = x, l = 200, r = 2, n_cores = 1, dist_fn = stats::dist)
head(mds$points)
mds$eigen
mds$GOF
points <- mds$points
plot(x[1:10, 1],
     x[1:10, 2],
     xlim = range(c(x[1:10,1],points[1:10,1])),
     ylim = range(c(x[1:10,2], points[1:10,2])),
     pch = 19,
     col = "green")
text(x[1:10, 1], x[1:10, 2], labels=1:10)
points(points[1:10, 1], points[1:10, 2], pch = 19, col = "orange")
text(points[1:10, 1], points[1:10, 2], labels=1:10)
abline(v = 0, lwd=3, lty=2)
abline(h = 0, lwd=3, lty=2)
```

Index

divide_conquer_mds, 2

fast_mds, 3

interpolation_mds, 5