Package ‘rsvd’

November 6, 2018

Type Package
Title Randomized Singular Value Decomposition
Version 1.0.0
Date 2018-11-05
Author N. Benjamin Erichson [aut, cre]
Maintainer N. Benjamin Erichson <erichson@uw.edu>
Description Low-rank matrix decompositions are fundamental tools and widely used for data analysis, dimension reduction, and data compression. Classically, highly accurate deterministic matrix algorithms are used for this task. However, the emergence of large-scale data has severely challenged our computational ability to analyze big data. The concept of randomness has been demonstrated as an effective strategy to quickly produce approximate answers to familiar problems such as the singular value decomposition (SVD). The rsvd package provides several randomized matrix algorithms such as the randomized singular value decomposition (rsvd), randomized principal component analysis (rpca), randomized robust principal component analysis (rrpca), randomized interpolative decomposition (rid), and the randomized CUR decomposition (rcur). In addition several plot functions are provided. The methods are discussed in detail by Erichson et al. (2016) <arXiv:1608.02148>.
Depends R (>= 3.2.2)
Imports Matrix
License GPL (>= 3)
LazyData TRUE
URL https://github.com/erichson/rSVD
BugReports https://github.com/erichson/rSVD/issues
Suggests ggplot2, testthat
RoxygenNote 6.1.0
NeedsCompilation no
Repository CRAN
Date/Publication 2018-11-06 06:00:03 UTC
digits

digits

ggbiplot

ggcorplot

Index 22

digits

Description

Subsampled MNIST database of handwritten digits. This smaller dataset has 3000 samples for each of the digits corresponding to the class labels 0,1,2,3. Each 28x28 image patch is stored as a flattened row vector.

Usage

data('digits')

Format

An object of class rsvd.

Source

mnist

References

ggbiplot

Examples

```r
## Not run:
library('rsvd')
data('digits')

digit <- matrix(digits[,1], nrow = 28, ncol = 28)
image(digit[28:1], col = gray(255:0 / 255))

## End(Not run)
```

Description

Creates a pretty biplot which is showing the individual factor map overlayed by the variables factor map, i.e, plotting both the principal component scores and directions.

Usage

```r
ggbiplot(rpcaObj, pcs = c(1, 2), loadings = TRUE, groups = NULL,
          alpha = 0.6, ellipse = TRUE, alpha.ellipse = 0.2,
          var_labels = TRUE, var_labels.names = NULL, ind_labels = TRUE,
          ind_labels.names = NULL)
```

Arguments

- **rpcaObj**: Object returned by the `rpca` function.
- **pcs**: Array_like.
  An array with two values indicating the two PCs which should be used for plotting. By default the first two PCs are used, e.g., `c(1, 2)`.
- **loadings**: Bool (`TRUE`, `FALSE`), optional.
  If `TRUE`, the eigenvectors are unit scaled by the square root of the eigenvalues `W = W * diag(sqrt(eigvals))`.
- **groups**: Factor, optional.
  Factor indicating groups.
- **alpha**: Scalar, optional.
  Alpha transparency for scatter plot.
- **ellipse**: Bool (`TRUE`, `FALSE`), optional.
  Draw a 1sd data ellipse for each group, if `TRUE`.
- **alpha.ellipse**: Scalar, optional.
  Alpha transparency for ellipse.
- **var_labels**: Bool (`TRUE`, `FALSE`), optional.
  Plot variable names, if `TRUE`.
ggcorplot

var_labels.names
Array_like, optional.
User specific labels for the individuals.

ind_labels
Bool (TRUE, FALSE), optional.
Plot data point names, if TRUE.

ind_labels.names
Array_like, optional.
User specific labels for data points.

Author(s)
N. Benjamin Erichson, <erichson@uw.edu>

See Also
rpca, ggplot

Examples

#See ?rpca

---

**Description**

Creates a pretty plot which is showing the correlation of the original variable with the principal component (PCs).

**Usage**

```r
ggcorplot(rpcaObj, pcs = c(1, 2), loadings = TRUE,
  var_labels = FALSE, var_labels.names = NULL, alpha = 1,
  top.n = NULL)
```

**Arguments**

rpcaObj
Object returned by the rpca function.

pcs
Array_like.
An array with two values indicating the two PCs which should be used for plotting. By default the first two PCs are used, e.g., c(1, 2).

loadings
Bool (TRUE, FALSE), optional.
If TRUE, the eigenvectors are unit scaled by the square root of the eigenvalues $W = W \times \text{diag}($sqrt(eigvals)$)).

var_labels
Bool (TRUE, FALSE), optional.
Plot variable names, if TRUE.
**ggindplot**

var_labels

- Array_like, optional.
  - User specific labels for the variables

alpha

- Scalar, optional.
  - Alpha transparency of the arrows.

top.n

- Scalar, optional.
  - Number of (most influencial) variables to label with small circles.

**Author(s)**

N. Benjamin Erichson, <erichson@uw.edu>

**See Also**

rpca, ggplot

**Examples**

```r
#
```

---

**Description**

Creates a pretty plot which is showing the individual factor map, i.e, plotting the principal component scores.

**Usage**

```r
ggindplot(rpcaObj, pcs = c(1, 2), groups = NULL, alpha = 0.6, 
  ellipse = TRUE, alpha.ellipse = 0.2, ind_labels = TRUE, 
  ind_labels.names = NULL)
```

**Arguments**

- `rpcaObj` Object returned by the `rpca` function.
- `pcs` Array_like.
  - An array with two values indicating the two PCs which should be used for plotting. By default the first two PCs are used, e.g., `c(1, 2)`.
- `groups` Factor, optional.
  - Factor indicating groups.
- `alpha` Scalar, optional.
  - Alpha transparency for scatter plot.
ggscreeplot

Description

Creates a pretty screeplot using ggplot. By default the explained variance is plotted against the number of the principal component. Alternatively the explained variance ratio, the cumulative explained variance ratio, or the eigenvalues can be plotted.

Usage

ggscreeplot(rpcaObj, type = c("var", "ratio", "cum", "eigenvals"))

Arguments

rpcaObj Object returned by the rpca function.
type String c('var', 'ratio', 'cum', 'eigenvals'), optional.

Author(s)

N. Benjamin Erichson, <erichson@uw.edu>

See Also

rpca, ggplot

Examples

#See ?rpca
plot.rpca

See Also

rpca, ggplot

Examples

#
**rcur**

*Randomized CUR matrix decomposition.*

**Description**
Randomized CUR matrix decomposition.

**Usage**
```r
cur(A, k = NULL, p = 10, q = 0, idx_only = FALSE, rand = TRUE)
```

**Arguments**
- `A`: array_like; numeric \((m, n)\) input matrix (or data frame). If the data contain \(\text{NAs}\) `na.omit` is applied.
- `k`: integer; target rank of the low-rank approximation, i.e., the number of columns/rows to be selected. It is required that \(k\) is smaller or equal to \(\min(m, n)\).
- `p`: integer, optional; oversampling parameter (default \(p = 10\)).
- `q`: integer, optional; number of additional power iterations (default \(q = 0\)).
- `idx_only`: bool, optional; if \(\text{TRUE}\), only the index set \(cNidx\) and \(rNidx\) is returned, but not the matrices \(C\) and \(R\). This is more memory efficient, when dealing with large-scale data.
- `rand`: bool, optional; if \(\text{TRUE}\), a probabilistic strategy is used, otherwise a deterministic algorithm is used.

**Details**
Algorithm for computing the CUR matrix decomposition of a rectangular \((m, n)\) matrix \(A\), with target rank \(k << \min(m, n)\). The input matrix is factored as

\[
A = C \ast U \ast R
\]

using the \textit{rid} decomposition. The factor matrix \(C\) is formed using actual columns of \(A\), also called the partial column skeleton. The factor matrix \(R\) is formed using actual rows of \(A\), also called the partial row skeleton.

If `rand = TRUE` a probabilistic strategy is used to compute the decomposition, otherwise a deterministic algorithm is used.
Value

rcur returns a list with class id containing the following components:

- **C**
  - array_like;
  - column subset $C = A[:, C.idx]$; $(m, k)$ dimensional array.

- **R**
  - Array_like.
  - row subset $R = A[R.idx, :]$; $(k, n)$ dimensional array.

- **U**
  - array_like;
  - connector matrix; $(k, k)$ dimensional array.

- **C.idx**
  - array_like;
  - index set of the $k$ selected columns used to form $C$.

- **R.idx**
  - array_like;
  - index set of the $k$ selected rows used to form $R$.

- **C.scores**
  - array_like;
  - scores of the selected columns.

- **R.scores**
  - array_like;
  - scores of the selected rows.

Author(s)

N. Benjamin Erichson, <erichson@uw.edu>

References


See Also

rid

Examples

```r
## Not run:
# Load test image
data('tiger')

# Compute (column) randomized interpolative decompsition
# Note that the image needs to be transposed for correct plotting
out <- rcur(tiger, k = 150)

tiger.re <- out$C %*% out$U %*% out$R

tiger.re <- out$R %*% out$U %*% out$C

tiger.re <- out$U %*% out$C

tiger.re <- out$C %*% out$U

tiger.re <- out$U

tiger.re <- out$C

tiger.re <- out$U

# Compute relative error
```

```r
```
Randomized interpolative decomposition (ID).

Description
Randomized interpolative decomposition.

Usage
```r
rid(A, k = NULL, mode = "column", p = 10, q = 0,
     idx_only = FALSE, rand = TRUE)
```

Arguments
- **A**: array_like; numeric \((m, n)\) input matrix (or data frame).
  - If the data contain NAs, na.omit is applied.
- **k**: integer, optional; number of rows/columns to be selected.
  - It is required that \(k\) is smaller or equal to \(\min(m, n)\).
- **mode**: string c(‘column’, ‘row’), optional;
  - columns or rows ID.
- **p**: integer, optional;
  - oversampling parameter (default \(p = 10\)).
- **q**: integer, optional;
  - number of additional power iterations (default \(q = 0\)).
- **idx_only**: bool, optional;
  - if \((TRUE)\), the index set \(\text{idx}\) is returned, but not the matrix \(C\) or \(R\).
  - This is more memory efficient, when dealing with large-scale data.
- **rand**: bool, optional;
  - if \((TRUE)\), a probabilistic strategy is used, otherwise a deterministic algorithm is used.

Details
Algorithm for computing the ID of a rectangular \((m, n)\) matrix \(A\), with target rank \(k << \min(m, n)\).

The input matrix is factored as

\[
A = C \ast Z
\]
using the column pivoted QR decomposition. The factor matrix $C$ is formed as a subset of columns of $A$, also called the partial column skeleton. If mode='row', then the input matrix is factored as

$$A = Z \ast R$$

using the row pivoted QR decomposition. The factor matrix $R$ is now formed as a subset of rows of $A$, also called the partial row skeleton. The factor matrix $Z$ contains a $(k, k)$ identity matrix as a submatrix, and is well-conditioned.

If rand = TRUE a probabilistic strategy is used to compute the decomposition, otherwise a deterministic algorithm is used.

Value

$rid$ returns a list containing the following components:

- **C**
  - array_like;
  - column subset $C = A[:, idx]$, if mode='column'; array with dimensions $(m, k)$.

- **R**
  - array_like;
  - row subset $R = A[idx, :]$, if mode='row'; array with dimensions $(k, n)$.

- **Z**
  - array_like;
  - well conditioned matrix; Depending on the selected mode, this is an array with dimensions $(k, n)$ or $(m, k)$.

- **idx**
  - array_like;
  - index set of the $k$ selected columns or rows used to form $C$ or $R$.

- **pivot**
  - array_like;
  - information on the pivoting strategy used during the decomposition.

- **scores**
  - array_like;
  - scores of the columns or rows of the input matrix $A$.

- **scores.idx**
  - array_like;
  - scores of the $k$ selected columns or rows in $C$ or $R$.

Author(s)

N. Benjamin Erichson, <erichson@uw.edu>

References


See Also

- rcur.
Examples

```r
## Not run:
# Load test image
data("tiger")

# Compute (column) randomized interpolative decomposition
# Note that the image needs to be transposed for correct plotting
out <- rid(t(tiger), k = 150)

# Show selected columns
tiger.partial <- matrix(0, 1200, 1600)
tiger.partial[, out$idx] <- t(tiger)[, out$idx]
image(t(tiger.partial), col = gray((0:255)/255), useRaster = TRUE)

# Reconstruct image
tiger.re <- t(out$C %*% out$Z)

# Compute relative error
print(norm(tiger-tiger.re, 'F') / norm(tiger, 'F'))

# Plot approximated image
image(tiger.re, col = gray((0:255)/255))
```

## End(Not run)

---

**Description**

Fast computation of the principal components analysis using the randomized singular value decomposition.

**Usage**

```r
rpca(A, k = NULL, center = TRUE, scale = TRUE, retx = TRUE,
     p = 10, q = 2, rand = TRUE)
```

**Arguments**

- `A` array_like;
  a numeric \((m, n)\) input matrix (or data frame) to be analyzed.
  If the data contain \(NAs\) na.omit is applied.
- `k` integer;
  number of dominant principle components to be computed. It is required that \(k\)
  is smaller or equal to \(min(m, n)\), but it is recommended that \(k << min(m, n)\).
- `center` bool, optional;
  logical value which indicates whether the variables should be shifted to be zero
  centered (\(TRUE\) by default).
scale  bool, optional; logical value which indicates whether the variables should be scaled to have unit variance (TRUE by default).
retx  bool, optional; logical value indicating whether the rotated variables / scores should be returned (TRUE by default).
p    integer, optional; oversampling parameter for rsvd (default p = 10), see rsvd.
q    integer, optional; number of additional power iterations for rsvd (default q = 1), see rsvd.
rand bool, optional; if (TRUE), the rsvd routine is used, otherwise svd is used.

Details

Principal component analysis is an important linear dimension reduction technique. Randomized PCA is computed via the randomized SVD algorithm (rsvd). The computational gain is substantial, if the desired number of principal components is relatively small, i.e. \( k \ll \min(m, n) \).

The print and summary method can be used to present the results in a nice format. A scree plot can be produced with ggscreeplot. The individuals factor map can be produced with ggindplot, and a correlation plot with ggcorplot.

The predict function can be used to compute the scores of new observations. The data will automatically be centered (and scaled if requested). This is not fully supported for complex input matrices.

Value

rpca returns a list with class rpca containing the following components:

rotation  array_like; the rotation (eigenvectors); \((n, k)\) dimensional array.
eigvals   array_like; eigenvalues; \(k\) dimensional vector.
sdev      array_like; standard deviations of the principal components; \(k\) dimensional vector.
x         array_like; the scores / rotated data; \((m, k)\) dimensional array.
center, scale array_like; the centering and scaling used.

Note

The principal components are not unique and only defined up to sign (a constant of modulus one in the complex case) and so may differ between different PCA implementations.

Similar to prcomp the variances are computed with the usual divisor \(N - 1\).
Author(s)

N. Benjamin Erichson, <erichson@uw.edu>

See Also

ggscreepplot, ggindplot, ggcorplot, plot.rpca, predict.rsvd

Examples

library('rsvd')
#
# Load Edgar Anderson's Iris Data
#
data('iris')
#
# log transform
#
log.iris <- log( iris[, 1:4] )
iris.species <- iris[, 5]

# Perform rPCA and compute only the first two PCs
#
iris.rpca <- rpca(log.iris, k=2)
summary(iris.rpca) # Summary
print(iris.rpca) # Prints the rotations

# Use rPCA to compute all PCs, similar to \code{\link{prcomp}}
#
iris.rpca <- rpca(log.iris)
summary(iris.rpca) # Summary
print(iris.rpca) # Prints the rotations
plot(iris.rpca) # Produce screeplot, variable and individuals factor maps.

rqb Randomized QB Decomposition (rqb).

Description

Compute the near-optimal QB decomposition of a rectangular matrix.

Usage

rqb(A, k = NULL, p = 10, q = 2, sdist = "normal", rand = TRUE)
Arguments

A  
array_like;
real/complex $(m, n)$ input matrix (or data frame).

k  
integer, optional;
target rank of the low-rank decomposition. It should satisfy $k << \min(m, n)$.

p  
integer, optional;
oversampling parameter (default $p = 10$).

q  
integer, optional;
number of power iterations (default $q = 2$).

sdist  
string c('unif', 'normal', 'rademacher'), optional;
specifies the sampling distribution:
'unif': Uniform $[-1,1]$.
'normal' (default): Normal $\sim N(0,1)$.
'rademacher': Rademacher random variates.

rand  
bool, optional;
If (TRUE), a probabilistic strategy is used, otherwise a deterministic algorithm is used.

Details

The randomized QB decomposition factors a rectangular $(m, n)$ matrix $A$ as $A = Q \ast B$. $Q$ is an $(m, k)$ matrix with orthogonal columns, and $B$ a $(k, n)$ matrix. The target rank is assumed to be $k << \min(m, n)$.

$p$ is an oversampling parameter to improve the approximation. A value between 5 and 10 is recommended, and $p = 10$ is set by default.

The parameter $q$ specifies the number of power (subspace) iterations to reduce the approximation error. This is recommended if the the singular values decay slowly. In practice 1 or 2 iterations achieve good results, however, computing power iterations increases the computational time. The number of power iterations is set to $q = 2$ by default.

Value

rqb returns a list containing the following components:

Q  
array_like;
matrix with orthogonal columns; $(m, k)$ dimensional array.

B  
array_like;
smaller matrix; $(k, n)$ dimensional array.

Author(s)

N. Benjamin Erichson, <erichson@uw.edu>
References


See Also

svd

rrpca

Randomized robust principal component analysis (rrpca).

Description

Robust principal components analysis separates a matrix into a low-rank plus sparse component.

Usage

rrpca(A, lambda = NULL, maxiter = 50, tol = 1e-05, p = 10, q = 2, trace = FALSE, rand = TRUE)

Arguments

A
array_like;
a real \((m, n)\) input matrix (or data frame) to be decomposed.
a.omit is applied, if the data contain NAs.

lambda
scalar, optional;
tuning parameter (default \(\lambda = \max(m, n)^{-0.5}\)).

maxiter
integer, optional;
maximum number of iterations (default \(maxiter = 50\)).

tol
scalar, optional;
precision parameter (default \(tol = 1.0e - 5\)).

p
integer, optional;
oversampling parameter for rsvd (default \(p = 10\)), see rsvd.

q
integer, optional;
number of additional power iterations for rsvd (default \(q = 2\)), see rsvd.

trace
bool, optional;
print progress.

rand
bool, optional;
if (\(TRUE\)), the rsvd routine is used, otherwise svd is used.
Details

Robust principal component analysis (RPCA) is a method for the robust separation of a rectangular \((m, n)\) matrix \(A\) into a low-rank component \(L\) and a sparse component \(S\):

\[
A = L + S
\]

To decompose the matrix, we use the inexact augmented Lagrange multiplier method (IALM). The algorithm can be used in combination with either the randomized or deterministic SVD.

Value

`rrpca` returns a list containing the following components:

- **L**: array_like; low-rank component; \((m, n)\) dimensional array.
- **S**: array_like; sparse component; \((m, n)\) dimensional array.

Author(s)

N. Benjamin Erichson, <erichson@uw.edu>

References


Examples

```r
library('rsvd')

# Create toy video
# background frame
xy <- seq(-50, 50, length.out=100)
mggrid <- list( x=outer(xy*0,xy,FUN="+"), y=outer(xy,xy*0,FUN="+"))
bg <- 0.1*exp(sin(-mggrid$x**2-mgrid$y**2))
toyVideo <- matrix(rep(c(bg), 100), 100*100, 100)

# add moving object
for(i in 1:90) {
  mobject <- matrix(0, 100, 100)
  mobject[1:(10+i), 45:55] <- 0.2
  toyVideo[,i] = toyVideo[,i] + c( mobject )
}

# Foreground/Background separation
out <- rrpca(toyVideo, trace=TRUE)
```
# Display results of the separation for the 10th frame
par(mfrow=c(1,4))
image(matrix(bg, ncol=100, nrow=100)) # true background
image(matrix(toyVideo[,10], ncol=100, nrow=100)) # frame
image(matrix(out$[,10], ncol=100, nrow=100)) # separated background
image(matrix(out$[,10], ncol=100, nrow=100)) # separated foreground

rsvd

Randomized Singular Value Decomposition (rsvd).

Description

The randomized SVD computes the near-optimal low-rank approximation of a rectangular matrix using a fast probabilistic algorithm.

Usage

rsvd(A, k = NULL, nu = NULL, nv = NULL, p = 10, q = 2,
    sdist = "normal")

Arguments

A
array_like;
a real/complex \((m, n)\) input matrix (or data frame) to be decomposed.

k
integer;
specifies the target rank of the low-rank decomposition. \(k\) should satisfy \(k << min(m, n)\).

nu
integer, optional;
number of left singular vectors to be returned. \(nu\) must be between \(0\) and \(k\).

nv
integer, optional;
number of right singular vectors to be returned. \(nv\) must be between \(0\) and \(k\).

p
integer, optional;
oversampling parameter (by default \(p = 10\)).

q
integer, optional;
number of additional power iterations (by default \(q = 2\)).

sdist
string c('unif', 'normal', 'rademacher'), optional;
specifies the sampling distribution of the random test matrix:
'unif': Uniform \([-1,1]\).
'normal' (default): Normal \(\sim N(0,1)\).
'rademacher': Rademacher random variates.
The singular value decomposition (SVD) plays an important role in data analysis, and scientific computing. Given a rectangular $(m, n)$ matrix $A$, and a target rank $k \ll \min(m, n)$, the SVD factors the input matrix $A$ as

$$A = U_k \text{diag}(d_k)V_k^\top$$

The $k$ left singular vectors are the columns of the real or complex unitary matrix $U$. The $k$ right singular vectors are the columns of the real or complex unitary matrix $V$. The $k$ dominant singular values are the entries of $d$, and non-negative and real numbers.

$p$ is an oversampling parameter to improve the approximation. A value of at least 10 is recommended, and $p = 10$ is set by default.

The parameter $q$ specifies the number of power (subspace) iterations to reduce the approximation error. The power scheme is recommended, if the singular values decay slowly. In practice, 2 or 3 iterations achieve good results, however, computing power iterations increases the computational costs. The power scheme is set to $q = 2$ by default.

If $k > (\min(n, m)/4)$, a deterministic partial or truncated svd algorithm might be faster.

**Value**

`rsvd` returns a list containing the following three components:

- `d`: array_like; singular values; vector of length $(k)$.
- `u`: array_like; left singular vectors; $(m, k)$ or $(m, nu)$ dimensional array.
- `v`: array_like; right singular vectors; $(n, k)$ or $(n, nv)$ dimensional array.

**Note**

The singular vectors are not unique and only defined up to sign (a constant of modulus one in the complex case). If a left singular vector has its sign changed, changing the sign of the corresponding right vector gives an equivalent decomposition.

**Author(s)**

N. Benjamin Erichson, <erichson@uw.edu>

**References**


See Also

svd, rpc

Examples

library('rsvd')

# Create a n x n Hilbert matrix of order n,
# with entries H[i,j] = 1 / (i + j + 1).
hilbert <- function(n) ( i <- 1:n; 1 / outer(i - 1, i, "+") )
H <- hilbert(n=50)

# Low-rank (k=10) matrix approximation using rsvd
k=10
s <- rsvd(H, k=k)
Hre <- s$su %*% diag(s$sd) %*% t(s$sv) # matrix approximation
print(100 * norm(H - Hre, 'F') / norm(H,'F')) # percentage error
# Compare to truncated base svd
s <- svd(H)
Hre <- s$su[,1:k] %*% diag(s$sd[1:k]) %*% t(s$sv[,1:k]) # matrix approximation
print(100 * norm(H - Hre, 'F') / norm(H,'F')) # percentage error


tiger        Tiger

Description

1600x1200 grayscaled (8 bit [0-255]/255) image.

Usage

data('tiger')

Format

An object of class rsvd.

Source

Wikimedia

References

S. Taheri (2006). "Panthera tigris altaica", (Online image)
Examples

## Not run:
library('rsvd')
data('tiger')

# Display image
image(tiger, col = gray((0:255)/255))

## End(Not run)
Index

*Topic **image**
  tiger, 20
*Topic **pattern**
  digits, 2
*Topic **recognition**
  digits, 2

digits, 2

ggbiplot, 3
ggcorplot, 4, 7, 13, 14
ggindplot, 5, 7, 13, 14
ggplot, 3–7
ggsceepplot, 6, 7, 13, 14

plot.rpca, 7, 14
prcomp, 13
predict, 14

rcur, 8, 11
rid, 8, 9, 10
rpca, 3–7, 12, 20
rqb, 14
rrpca, 16
rsrd, 2, 13, 14, 16, 18, 20

svd, 16, 19, 20

tiger, 20