Package 'rsvd'

July 29, 2016

Type Package

Title Randomized Singular Value Decomposition

Version 0.6

Date 2016-07-28

Author N. Benjamin Erichson [aut, cre]

Maintainer N. Benjamin Erichson <nbe@st-andrews.ac.uk>

Description Randomized singular value decomposition (rsvd) is a very fast probabilistic algorithm that can be used to compute the near optimal low-rank singular value decomposition of massive data sets with high accuracy. SVD plays a central role in data analysis and scientific computing. SVD is also widely used for computing (randomized) principal component analysis (PCA), a linear dimensionality reduction technique. Randomized PCA (rpca) uses the approximated singular value decomposition to compute the most significant principal components. This package also includes a function to compute (randomized) robust principal component analysis (RPCA). In addition several plot functions are provided.

Depends R (>= 3.2.2)

License GPL (>= 2)

LazyData TRUE

URL https://github.com/Benli11/rSVD

BugReports https://github.com/Benli11/rSVD

RoxygenNote 5.0.1

Suggests ggplot2, plyr, scales, grid, testthat, knitr, rmarkdown

NeedsCompilation no

Repository CRAN

Date/Publication 2016-07-29 06:41:14

R topics documented:

ggbiplot 2
ggcorplot 3
ggscreeplot 3

plot.rpca 4
reigen 5
rpca 7
rrpca 10
rsvd 12
tiger 15

Index **16**

`ggbiplot` *Biplot for* `rPCA` *using ggplot2*

Description

Biplot for `rPCA` using ggplot2

Usage

```
ggbiplot(rpcaObj, pcs = c(1, 2), groups = NULL, ellipse = TRUE,
  alpha.ellipse = 0.2, loadings = TRUE, labels = TRUE, ...)
```

Arguments

`rpcaObj`	object containing the `sdev` component, such as that returned by `rpca`
`pcs`	array_like an array with two values indicating which two PCs should be plotted, by default the first two PCs are used, e.g., $c(1, 2)$.
`groups`	optional, factor factor indicating groups
`ellipse`	draw a 1sd data ellipse for each group
`alpha.ellipse`	alpha transparency for ellipse
`loadings`	draw arrows for the variables
`labels`	label variables
`...`	arguments passed to or from other methods, see `ggplot`.

Author(s)

N. Benjamin Erichson, `<nbe@st-andrews.ac.uk>`

See Also

`rpca`, `ggplot`

Examples

```
#See ?rpca
```

ggcorplot *Correlation plot*

Description

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

Usage

```
ggcorplot(rpcaObj, pcs = c(1, 2))
```

Arguments

rpcaObj	object containing the sdev component, such as that returned by rpca
pcs	array_like an array with two values indicating which two PCs should be plotted, by default the first two PCs are used, e.g., $c(1, 2)$.
.....	.

See Also

rpca, ggplot

Examples

```
#
```

ggscreeplot *Pretty Screeplot*

Description

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

Usage

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

Arguments

rpcaObj	object containing the sdev component, such as that returned by rpca
type	str c('var', 'ratio', 'cum', 'eigenvals'), optional
.....	.

See Also

rpca, ggplot

Examples

```
#
```

---

plot.rpca *Screeplot*

Description

Creates a screeplpot. By default the explained variance is plotted againgst the number of the principal component. Alternatively the eigenvalues, explained variance ratio or the cumulative explained variance ratio can be plotted.

Usage

```
## S3 method for class 'rpca'
plot(x, type = c("var", "ratio", "cum", "eigenvals"), ...)
```

Arguments

x	object containing the sdev component, such as that returned by rpca
type	str c('var', 'ratio', 'cum', 'eigenvals'), optional
...	arguments passed to or from other methods, see plot.
.....	.

See Also

rpca

Examples

```
#
```

`reigen` *Randomized Spectral Decomposition of a matrix (reigen).*

Description

Computes the approximate low-rank eigendecomposition of a symmetric matrix.

Usage

```
reigen(A, k = NULL, p = 10, q = 1, sdist = "unif")
```

Arguments

`A`	array_like a real/complex input matrix (or data frame), with dimensions (m, n).
`k`	int, optional determines the target rank of the low-rank decomposition and should satisfy $k << min(m, n)$.
`p`	int, optional oversampling parameter for (default $p = 10$).
`q`	int, optional number of power iterations (default $q = 1$).
`sdist`	str c('normal', 'unif', 'spixel'), optional Specifies the sampling distribution. 'unif' : (default) Uniform '[-1,1]'. 'normal' : Normal '~N(0,1)'. 'col' : Random column sampling.
.....	.

Details

The eigenvalue decomposition plays a central role in data analysis and scientific computing. Randomized eigen (reigen) is a fast algorithm to compute the the approximate low-rank eigenvalue decomposition of $A'A$ given the rectangular (m, n) matrix A using a probablistic algorithm. Given a target rank $k << n$, `reigen` factors the input matrix A as $A'A = V * diag(d) * V'$. The eigenvectors are the columns of the real or complex unitary matrix V. The eigenvalues d are non-negative and real numbers.

The parameter p is a oversampling parameter to improve the approximation. A value between 2 and 10 is recommended and $p = 10$ is set as default.

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

If $k > (min(n, m)/1.5)$, a deterministic partial or truncated `eigen` algorithm might be faster.

Value

`reigen` returns a list containing the following two components:

`values`	array_like Eigenvalues; 1-d array of length (k).
`vectors`	array_like Eigenvectors; array with dimensions (k, k).
`.....`	.

Note

The eigenvectors are not unique and only defined up to sign (a constant of modulus one in the complex case).

Author(s)

N. Benjamin Erichson, <nbe@st-andrews.ac.uk>

References

1 N. Halko, P. Martinsson, and J. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions" (2009). (available at arXiv http://arxiv.org/abs/0909.4061).

2 S. Voronin and P.Martinsson. "RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures" (2015). (available at 'arXiv http://arxiv.org/abs/1502.05366).

See Also

rsvd, rpca, eigen

Examples

```
library(rsvd)
set.seed(123)

#Create real random test matrix with dimension (m, n) and rank k
m = 10
n = 5
k = 3
A <- matrix(runif(m*k), m, k)
A <- A %*% t(A)
A <- A[,1:n]

AtA = t(A) %*% A

# Randomized low-rank eigenvalue decomposition k=3
reigen.out <- reigen(A, k=3)
AtA.re = reigen.out$vectors %*% diag(reigen.out$values) %*% t(reigen.out$vectors)
```

```
100 * norm( AtA - AtA.re, 'F') / norm( AtA,'F') #Percentage reconstruction error << 1e-8
print(reigen.out$values) # print eigenvalues

# Compare with the deterministic eigenvalue decomposition
eigen.out <- eigen(AtA)
AtA.re2 = eigen.out$vectors %*% diag(eigen.out$values) %*% t(eigen.out$vectors)
100 * norm( AtA - AtA.re2, 'F') / norm( AtA,'F') #Percentage reconstruction error << 1e-8
print(eigen.out$values) # print eigenvalues
all.equal(eigen.out$values[1:k], reigen.out$values)
```

---

rpca *Randomized principal component analysis (rpca).*

Description

Principal components analysis using randomized singular value decomposition.

Usage

```
rpca(A, k = NULL, center = TRUE, scale = TRUE, loading = FALSE,
  retx = FALSE, svdalg = "auto", p = 10, q = 1, ...)
```

Arguments

A	array_like a numeric input matrix (or data frame), with dimensions (m, n). If the data contain NAs na.omit is applied.
k	int, optional determines the number of principle components to compute. It is required that k is smaller or equal to n, but it is recommended that $k << min(m, n)$.
center	bool ($TRUE$, $FALSE$), optional a logical value ($TRUE$ by default) indicating whether the variables should be shifted to be zero centered. Alternatively, a vector of length equal the number of columns of A can be supplied. The value is passed to scale.
scale	bool ($TRUE$, $FALSE$), optional a logical value ($TRUE$ by default) indicating whether the variables should be scaled to have unit variance. Alternatively, a vector of length equal the number of columns of A can be supplied. The value is passed to scale.
loading	bool ($TRUE$, $FALSE$), optional When $TRUE$ (by default $FALSE$) the eigenvectors are unit scaled by the square root of the eigenvalues $W = W * diag(sqrt(eigvals))$.
retx	bool ($TRUE$, $FALSE$), optional a logical value ($FALSE$ by default) indicating whether the rotated variables / scores should be returned.

svdalg — str c('auto', 'reigen', 'rsvd', svd'), optional
Determines which algorithm should be used for computing the singular value decomposition. By default 'auto' is used, which decides whether to use reigen or svd, depending on the number of principle components. If $k < min(n, m)/1.5$ randomized reigen is used. Instead rsvd can be used, as well.

p — int, optional
oversampling parameter for $reigen$ (default $p = 10$), see reigen.

q — int, optional
number of power iterations for $reigen$ (default $q = 1$), see reigen.

... — arguments passed to or from other methods, see reigen.
.....
.

Details

Principal component analysis is a linear dimensionality reduction technique, aiming to keep only the most significant principal components to allow a better interpretation of the data and to project the data to a lower dimensional space.

Traditionally, the computation is done by a (deterministic) singular value decomposition. Randomized PCA is computed using a fast randomized algorithm (rsvd) to compute the approximate low-rank SVD decomposition. The computational gain is high if the desired number of principal components is small, i.e. $k << n$.

rsvd expects a numeric (real/complex) input matrix with dimensions (m, n). Given a target rank k, rsvd factors the input matrix A as $A = W * diag(s) * W'$. The columns of the real or complex unitary matrix W contain the eigenvectors (i.e. principal components). The vector s contains the corresponding eigenvalues. Following prcomp we denote this matrix W as rotation matrix (commonly also called loadings).

The print and summary method can be used to present the results in a nice format. A scree plot can be produced with the plot function or as recommended with ggscreeplot. A biplot can be produced with ggbiplot, and a correlation plot with ggcorplot.

The predict function can be used to compute the scores of new observations. The data will automatically be centred (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
matrix containing the rotation (eigenvectors), or the variable loadings if $loadings = TRUE$; array with dimensions (n, k).

loading — array_like
matrix containing the loadings (scaled eigenvectors), if $loadings = TRUE$; array with dimensions (n, k).

eigvals — array_like
the eigenvalues; 1-d array of length k.

sdev — array_like
the standard deviations of the principal components.

x	array_like if $retx$ is true a matrix containing the scores / rotated data (centred and scaled if requested) is returned.
center, scale	array_like the centering and scaling used, or $FALSE$.
.....	.

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.

Note also that $scale = TRUE$ cannot be used if there are zero or constant (for $center = TRUE$) variables.

Author(s)

N. Benjamin Erichson, <nbe@st-andrews.ac.uk>

See Also

ggscreeplot, ggbiplot, 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 all PCs, similar to \code{\link{prcomp}}
#
iris.rpca <- rpca(log.iris, retx=TRUE)
summary(iris.rpca) # Summary
print(iris.rpca) # Prints the rotations

# You can compare the results with prcomp
# iris.pca <- prcomp(log.iris, center = TRUE, scale. = TRUE)
# summary(iris.pca) # Summary
# print(iris.pca) # Prints the rotations
```

```
#
# Plot functions
#
ggscreeplot(iris.rpca) # Screeplot (explained variance/eigenvalues)
ggscreeplot(iris.rpca, 'ratio') # Proportion of variance
ggscreeplot(iris.rpca, 'cum') # Cummulative proportion

ggcorplot(iris.rpca, pcs=c(1,2)) # The correlation of the original variable with the PCs

ggbiplot(iris.rpca, groups = iris.species, circle = FALSE) #Biplot

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

#
# Compute the scores of new observations
#
preds <- predict(iris.rpca, newdata=data.frame(log.iris))
```

`rrpca` *Randomized robust principal component analysis (rrpca).*

Description

Robust principal components analysis using randomized singular value decomposition.

Usage

```
rrpca(A, k = NULL, lamb = NULL, gamma = 1.25, rho = 1.5, maxiter = 50,
  tol = 0.001, svdalg = "auto", p = 10, q = 1, trace = FALSE, ...)
```

Arguments

`A`	array_like a numeric input matrix (or data frame), with dimensions (m, n). If the data contain NAs na.omit is applied.
`k`	int, optional determines the number of principle components to compute. It is required that k is smaller or equal to n, but it is recommended that $k << min(m, n)$.
`lamb`	real, optional tuning paramter (default $lamb = max(m, n)^{-}0.5$).
`gamma`	real, optional tuning paramter (default $gamma = 1.25$).

`rho` real, optional
tuning paramter (default $rho = 1.5$).

`maxiter` int, optional
determines the maximal numbers of iterations (default $maxiter = 20$)..

`tol` real, optional
tolarance paramter for the desired convergence of the algorithm.

`svdalg` str c('auto', 'rsvd', 'svd'), optional
Determines which algorithm should be used for computing the singular value decomposition. By default 'auto' is used, which decides whether to use `rsvd` or `svd`, depending on the number of principle components. If $k < min(n, m)/1.5$ randomized svd is used.

`p` int, optional
oversampling parameter for $rsvd$ (default $p = 0$), see `rsvd`.

`q` int, optional
number of power iterations for $rsvd$ (default $q = 1$), see `rsvd`.

`trace` bool, optional
print progress.

`...` arguments passed to or from other methods, see `rsvd`.
.....
.

Details

Robust principal component analysis (RPCA) is a method for the robust seperation of a a rectangular (m, n) matrix A into a low-rank component L and a sparse comonent S as follows: $A = L + S$. Here we are using the fast randomized accelerated inexact augmented Lagrange multiplier method (IALM) for obtaining the robust seperation.

Value

`rrpca` returns a list with class $rrpca$ containing the following components:

`L` array_like
Low-rank component, array of shape (m, n).

`S` array_like
Sparse component, array of shape (m, n).

`k` int
target-rank used for the final iteration.

`err` vector
Frobenious error archieved by each iteration.
.....
.

Note

...

Author(s)

N. Benjamin Erichson, <nbe@st-andrews.ac.uk>

References

1 Lin, Zhouchen, Minming Chen, and Yi Ma. "The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices." (2010). (available at arXiv http://arxiv.org/abs/1009.5055).

2 Candes, Emmanuel J., et al. "Robust principal component analysis?." Journal of the ACM (JACM) 58.3 (2011).

Examples

```
library(rsvd)

# Create toy video
# background frame
xy <- seq(-50, 50, length.out=100)
mgrid <- list( x=outer(xy*0,xy,FUN="+"), y=outer(xy,xy*0,FUN="+") )
bg <- 0.1*exp(sin(-mgrid$x**2-mgrid$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[i:(10+i), 45:55] <- 0.2
  toyVideo[,i] =  toyVideo[,i] + c( mobject )
}

# Foreground/Background separation
out <- rrpca(toyVideo, k=1, p=5, q=1, svdalg='rsvd', trace=TRUE)

# Display results of the seperation 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$L[,10], ncol=100, nrow=100)) # seperated background
image(matrix(out$S[,10], ncol=100, nrow=100)) #seperated foreground
```

`rsvd` *Randomized Singular Value Decomposition (rsvd).*

Description

Compute the near-optimal low-rank singular value decomposition (SVD) of a rectangular matrix.

Usage

```
rsvd(A, k = NULL, nu = NULL, nv = NULL, p = 10, q = 1,
  sdist = "unif", vt = FALSE)
```

Arguments

A	array_like a real/complex input matrix (or data frame), with dimensions (m, n).
k	int, optional determines the target rank of the low-rank decomposition and should satisfy $k << min(m, n)$.
nu	int, optional the number of left singular vectors to be computed. This must be between 0 and k.
nv	int, optional the number of right singular vectors to be computed. This must be between 0 and k.
p	int, optional oversampling parameter for (default $p = 10$).
q	int, optional number of power iterations (default $q = 1$).
sdist	str $c('normal', 'unif', 'col')$, optional Specifies the sampling distribution. $'unif'$: (default) Uniform '[-1,1]'. $'normal'$: Normal '~N(0,1)'. $'col'$: Random column sampling.
vt	bool ($TRUE$, $FALSE$), optional $TRUE$: returns the transposed right singular vectors vt. $FALSE$: (default) returns the right singular vectors as v, this is the format as svd returns v.
.....	.

Details

The singular value decomposition (SVD) plays a central role in data analysis and scientific computing. Randomized SVD (rSVD) is a fast algorithm to compute the approximate low-rank SVD of a rectangular (m, n) matrix A using a probablistic algorithm. Given a target rank $k << n$, rsvd factors the input matrix A as $A = U * diag(d) * V'$. The right singluar vectors are the columns of the real or complex unitary matrix U . The left singular vectors are the columns of the real or complex unitary matrix V. The singular values d are non-negative and real numbers.

The parameter p is a 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 normalized power iterations (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 $q = 1$ by default.

If $k > (min(n, m)/1.5)$, 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; 1-d array of length (k).
u	array_like Right singular values; array with dimensions (m, k).
v (or vt)	array_like Left singular values; array with dimensions (n, k). Or if $vt = TRUE$, array with dimensions (k, n).
.....	.

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, <nbe@st-andrews.ac.uk>

References

1 N. Halko, P. Martinsson, and J. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions" (2009). (available at arXiv http://arxiv.org/abs/0909.4061).

2 S. Voronin and P.Martinsson. "RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures" (2015). (available at 'arXiv http://arxiv.org/abs/1502.05366).

See Also

svd, rpca

Examples

```
# Create a n by 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$u %*% diag(s$d) %*% t(s$v) # matrix approximation
print(100 * norm( H - Hre, 'F') / norm( H,'F')) # percentage error
# Compare to truncated base svd
```

```
s <- svd(H)
Hre <- s$u[,1:k] %*% diag(s$d[1:k]) %*% t(s$v[,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

```
library(rsvd)
#data(tiger)

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

Index

∗Topic **image**
tiger, 15

eigen, *5, 6*

ggbiplot, 2, *8, 9*
ggcorplot, 3, *8, 9*
ggplot, *2–4*
ggscreeplot, 3, *8, 9*

plot, *4*
plot.rpca, 4, *9*
prcomp, *8, 9*
predict, *9*

reigen, 5, *8*
rpca, *2–4*, *6*, 7, *14*
rrpca, 10
rsvd, *6*, *8, 9*, *11*, 12

svd, *8*, *11*, *13, 14*

tiger, 15