Package ‘onlinePCA’

May 7, 2014

Type Package

Title Online Principal Component Analysis

Version 1.0

Date 2014-05-07

Author David Degras

Maintainer David Degras <ddegrasv@depaul.edu>

Description
This package implements two types of online PCA algorithms, recursive and stochastic gradient, as well as convenience functions for updating the sample mean and sample covariance.

License GPL-3

Depends R (>= 3.0.0), stats, graphics, rARPACK

R topics documented:

`onlinePCA-package` *Online Principal Component Analysis*

Description

This package implements two types of online PCA algorithms, recursive and stochastic gradient, as well as convenience functions for updating the sample mean and sample covariance.

Details

Package:	onlinePCA
Type:	Package
Version:	1.0
Date:	2014-05-07
License:	GPL-3

Author(s)

David Degras <ddegrasv@depaul.edu>

References

Gu, M. and Eisenstat, S. C. (1994). A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal of Matrix Analysis and Applications.*
Hegde et al. (2006) Perturbation-Based Eigenvector Updates for On-Line Principal Components Analysis and Canonical Correlation Analysis. *Journal of VLSI Signal Processing*.
Li, W., Yue, H. H., Valles-Cervantes, S. and Qin, S. J. (2000). Recursive PCA for adaptive process monitoring. *Journal of Process Control.*
Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

`batchpca` *Compute the PCA of a covariance matrix*

Description

This function performs the PCA of a covariance matrix, returning an arbitrary number of principal components (eigenvectors) and corresponding eigenvalues.

Usage

```
batch.pca(C, k)
```

Arguments

`C`	covariance matrix.
`k`	number of PC and eigenvalues to compute.

Details

If the number of PC to be computed is small ($k <= ncol(C)/10$), the eigendecomposition is efficiently carried out by the function `eigs_sym` of the package `rARPACK`, which implements an Implicitly Restarted Lanczos Method (IRLM). Otherwise the eigendecomposition is performed with the R base function `eigen`.

Value

A list with components

`values`	the first k eigenvaluesof `C`.
`vectors`	the first k PC/eigenvectors of `C`.

References

http://www.caam.rice.edu/software/ARPACK/

Examples

```
n <- 1e4
p <- 500
k <- 10

mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
C <- cov(mat)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## k first eigenpairs of C using base function eigen
system.time({
  eigenC <- eigen(C, TRUE)
  eigenC$values <- eigenC$values[1:k]
  eigenC$vectors <- eigenC$vectors[,1:k]
  })

## k first eigenpairs of C using function batchpca
system.time(eigenCBatch <- batchpca(C, k))

## Check equality
all.equal(eigenC$values, eigenCBatch$values)
# Reorient eigenvectors if needed
for (i in 1:k) {
  if (sum(eigenCBatch$vectors[,i] * eigenC$vectors[,i]) < 0)
    eigenCBatch$vectors[,i] <- - eigenCBatch$vectors[,i]
}
all.equal(eigenC$vectors, eigenCBatch$vectors)
```

`perturbationRpca` *Update the PCA using a rank 1 perturbation method*

Description

This function recursively updates the PCA with respect to a single new data vector, using the (fast) perturbation method of Hegde et al. (2006).

Usage

```
perturbationRpca(d, Q, x, n, ff, center)
```

Arguments

`d`	vector of eigenvalues.
`Q`	matrix of principal components (eigenvectors of the covariance matrix) stored in columns.
`x`	new data vector.
`n`	current sample size (prior to observation of new data)
`ff`	forgetting factor: a number between 0 and 1, set to `1/n` by default.
`center`	centering vector (optional).

Details

The forgetting factor `ff` determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments `n` and `ff` must be specified. If `ff` is specified, its value overrides the argument `n`; otherwise, `ff` is set to `1/n` which corresponds to the assumption of a stationnary observation process. If `center` is not specified, then the data `x` is not centered prior to PCA.

Value

A list with components

`values`	the updated eigenvalues.
`vectors`	the updated eigenvectors.

Note

This perturbation method is based on large sample approximations. It can be highly inaccurate for small/medium sized samples and should not be used in this case.

References

Hegde et al. (2006) Perturbation-Based Eigenvector Updates for On-Line Principal Components Analysis and Canonical Correlation Analysis. *Journal of VLSI Signal Processing*.

See Also

`secularRpca`

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10

mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## Perturbation-based PCA
xbar <- colMeans(mat[1:n0,])
pca <- batchpca(cov(mat[1:n0,]))
```

```
for (i in (n0+1):n) {
  xbar <- updateMean(xbar, mat[i,],  i - 1)
  pca <- perturbationRpca(pca$values, pca$vectors, mat[i,],
    n = i - 1, center = xbar)
}
pca

# Compare to batch PCA
eigen(cov(mat), TRUE)
```

secularRpca *Update the PCA using secular equations*

Description

This function recursively updates a PCA with respect to a new data vector. The PCA eigenvalues are computed as the roots of a secular equations. The principal components are then either deduced by direct calculation or computed with the approximation method of Gu and Eisenstat (1994).

Usage

```
secularRpca(d, Q, x, n, ff, center, tol = 1e-10, reortho = FALSE)
```

Arguments

`d`	vector of eigenvalues.
`Q`	matrix of corresponding principal components (eigenvectors of the covariance matrix) stored in columns.
`x`	new data vector.
`n`	sample size before new observation.
`ff`	forgetting factor; a real number between 0 and 1 set to `1/n` by default.
`center`	centering vector for the new data (optional).
`tol`	error tolerance for the computation of eigenvalues.
`reortho`	logical; FALSE by default. If TRUE, the approximation method of Gu and Eisenstat (1994) is used to compute the eigenvectors.

Details

The method of secular equations provides accurate eigenvalues in all but pathological cases. On the other hand, the perturbation method implemented by `perturbationRpca` typically runs much faster but is only accurate for large sample sizes `n`.
The default method for computing PC's/eigenvectors (`reortho = FALSE`) is that of Li et al. (2000). It is very accurate for the first few PC's but loss of accuracy and orthogonality may occur for the next PC's. In contrast the method of Gu and Eisenstat (1994) is robust against small errors in the computation of eigenvalues. It provides PC's that may be less accurate than the default method but for which strict orthogonality is guaranteed.
Lower values of the forgetting factor `ff` place more weight on the current PCA decomposition while higher values place more weight on the new data. The default value `ff = 1/n` corresponds to the stationarity assumption.

Value

A list with components

`values`	updated eigenvalues in decreasing order
`vectors`	updated principal components

References

Gu, M. and Eisenstat, S. C. (1994). A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal of Matrix Analysis and Applications.*
Li, W., Yue, H. H., Valles-Cervantes, S. and Qin, S. J. (2000). Recursive PCA for adaptive process monitoring. *Journal of Process Control*.

See Also

`perturbationRpca`

`sgapca` *Stochastic Gradient Ascent PCA*

Description

This function updates the PCA with respect to a new data vector, based on the Stochastic Gradient Ascent algorithm of Oja (1992).

Usage

```
sgapca(Q, x, gamma, center, type = c("exact", "nn"))
```

Arguments

`Q`	matrix of principal components stored in columns.
`x`	new data vector.
`gamma`	gain parameter.
`center`	centering vector for `x` (optional).
`type`	string specifying the type of implementation: "exact" or "nn" (neural network).

Details

The gain parameter `gamma` is a positive number that determines the weight to be placed on the new data vector `x` in the PCA update. For larger values of `gamma`, more weight is placed on `x` and less on `Q`. A common choice for `gamma` is of the form `c/n`, with `n` the sample size and `c` a suitable positive constant.
The Stochastic Gradient Ascent PCA can be implemented exactly or through a neural network. The latter is less accurate but faster.

Value

A matrix of updated principal components of the same dimension as `Q`.

References

Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

See Also

snlpca

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10

mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## SGA-PCA
xbar <- colMeans(mat[1:n0,])
pca <- batchpca(cov(mat[1:n0,]))$vectors
for (i in (n0+1):n) {
  xbar <- updateMean(xbar, mat[i,],  i - 1)
  pca <- sgapca(pca, mat[i,], 2 / i, center = xbar)
}
pca

# Compare to batch PCA
eigen(cov(mat), TRUE)$vectors
```

`snlpca` *Subspace Network Learning PCA*

Description

This function updates the PCA with respect to a new data vector using the Subspace Network Learning algorithm of Oja (1992).

Usage

```
snlpca(Q, x, gamma, center, type = c("exact", "nn"))
```

Arguments

`Q`	matrix of principal components stored in columns.
`x`	new data vector.
`gamma`	gain parameter.
`center`	centering vector for x (optional).
`type`	string specifying the type of implementation: "exact" or "nn" (neural network).

Details

The gain parameter gamma is a positive number that determines the weight to be placed on the new data vector x in the PCA update. For larger values of gamma, more weight is placed on x and less on Q. A common choice for gamma is of the form c/n, with n the sample size and c a suitable positive constant.
The Subspace Network Learning PCA can be implemented exactly or through a neural network. The latter is less accurate but faster.

Value

A matrix of principal components/eigenvectors stored in columns.

Note

Unlike the Stochastic Gradient Ascent algorithm which consistently estimates the principal components (i.e., the eigenvectors of the theoretical covariance matrix), the Subspace Network Learning algorithm only provides the linear space spanned by the PCs and not the PCs themselves.

References

Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

See Also

sgapca

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10

mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## SNL-PCA
xbar <- colMeans(mat[1:n0,])
pca <- batchpca(cov(mat[1:n0,]))$vectors
for (i in (n0+1):n) {
  xbar <- updateMean(xbar, mat[i,],  i - 1)
  pca <- snlpca(pca, mat[i,], 1 / i, center = xbar)
}
pca
# Corresponding projection matrix
snlP <- tcrossprod(pca)

# Compare to batch PCA
(batch <- eigen(cov(mat), TRUE)$vectors)
batchP <- tcrossprod(batch)

# Relative  distance between the two projectors (Frobenius metric)
```

```
norm(snlP - batchP, "2") / norm(batchP, "2")
```

`updateCovariance` *Recursively update a covariance matrix*

Description

This function efficiently updates a covariance matrix for new observations without entirely recomputing the covariance.

Usage

```
updateCovariance(C, x, n, xbar, ff, byrow = TRUE)
```

Arguments

C	covariance matrix.
x	vector/matrix of new data.
n	sample size prior to the observation of new data.
xbar	sample mean (vector) prior to the observation of new data.
ff	forgetting factor: a number beween 0 and 1; set to 1/n by default.
byrow	logical: should be TRUE if the new data vectors are stored in rows (i.e., variables in columns) and FALSE otherwise.

Details

The forgetting factor `ff` determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments `n` and `ff` must be specified. If `ff` is specified, its value overrides the argument `n`; otherwise, `ff` is set to `1/n` which corresponds to the assumption of a stationnary observation process.
If unspecified, the argument `byrow` defaults to TRUE if `x` is a matrix and to FALSE if `x` is a vector.

Value

The updated covariance matrix.

See Also

updateMean

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10
mat <- matrix(runif(n*p), n, p)

## Direct computation of the covariance
C <- cov(mat)
```

```
## Recursive computation of the covariance
xbar0 <- colMeans(mat[1:n0,])
C0 <- cov(mat[1:n0,])
Crec <- updateCovariance(C0, mat[(n0+1):n,], n0, xbar0)

## Check equality
all.equal(C, Crec)
```

`updateMean` *Update the sample mean vector*

Description

This functions updates the sample mean vector with respect to new data.

Usage

```
updateMean(xbar, x, n, ff, byrow = TRUE)
```

Arguments

`xbar`	sample mean vector.
`x`	new data.
`n`	sample size prior to the observation of new data.
`ff`	forgetting factor: a number between 0 and 1, set to `1/n` by default.
`byrow`	logical; should be set to TRUE if the new data vectors are in rows (i.e., variables in columns) and FALSE otherwise.

Details

The forgetting factor `ff` determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments `n` and `ff` must be specified. If `ff` is specified, its value overrides the argument `n`; otherwise, `ff` is set to `1/n` which corresponds to the assumption of a stationnary observation process.
If unspecified, the argument `byrow` defaults to TRUE if `x` is a matrix and to FALSE if `x` is a vector.

Value

The updated sample mean vector.

See Also

`updateCovariance`

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10
mat <- matrix(runif(n*p), n, p)

## Direct computation of the mean
xbar <- colMeans(mat)

## Recursive computation of the covariance
xbar0 <- colMeans(mat[1:n0,])
xbarrec <- updateMean(xbar0, mat[(n0+1):n,], n0)

## Check equality
all.equal(xbar, xbarrec)
```

Index