Package: bcv (via r-universe)

October 29, 2026

Version 1.0.2

Date 2023-05-19

Title Cross-Validation for the SVD (Bi-Cross-Validation)

Description Methods for choosing the rank of an SVD (singular value decomposition) approximation via cross validation. The package provides both Gabriel-style ``block'' holdouts and Wold-style ``speckled'' holdouts. It also includes an implementation of the SVDImpute algorithm. For more information about Bi-cross-validation, see Owen & Perry's 2009 AoAS article (at <doi:10.48550/arXiv.0908.2062>) and Perry's 2009 PhD thesis (at <doi:10.48550/arXiv.0909.3052>).

License BSD_3_clause + file LICENSE

URL https://github.com/michbur/bcv

BugReports https://github.com/michbur/bcv/issues

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.2.3

Suggests spelling

Language en-US

Author Michal Burdukiewicz [cre, ctb] (<https://orcid.org/0000-0001-8926-582X>), Patrick O. Perry [aut, cph], Seyed Morteza Najibi [ctb]

Maintainer Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2023-05-19 12:50:02 UTC

RemoteUrl https://github.com/cran/bcv

RemoteRef HEAD

RemoteSha 4d6cfd919c52550d72747c2f09bf5181505d4218

Contents

bcv-package 2
cv.svd.gabriel 3
impute.svd 5
plot.cvsvd 6
print.cvsvd 8
summary.cvsvd 9

Index **10**

---

bcv-package *Cross-Validation for the SVD (Bi-Cross-Validation)*

Description

This package implements methods for choosing the rank of an SVD approximation via cross validation. It provides both Gabriel-style "block" holdouts and Wold-style "speckled" holdouts. Also included is an implementation of the SVDImpute algorithm.

Details

Package:	bcv
Type:	Package
Version:	1.0
Date:	2009-08-15
License:	BSD3

Basic usage is to call either `cv.svd.gabriel` or `cv.svd.wold`.

Author(s)

Patrick O. Perry <patperry@gmail.com>

See Also

`impute.svd`, `cv.svd.gabriel`, `cv.svd.wold`, `plot.cvsvd`, `print.cvsvd` `summary.cvsvd`

`cv.svd.gabriel` *Cross-Validation for choosing the rank of an SVD approximation.*

Description

Perform Wold- or Gabriel-style cross-validation for determining the appropriate rank SVD approximation of a matrix.

Usage

```
cv.svd.gabriel(
  x,
  krow = 2,
  kcol = 2,
  maxrank = floor(min(n - n/krow, p - p/kcol))
)

cv.svd.wold(x, k = 5, maxrank = 20, tol = 1e-04, maxiter = 20)
```

Arguments

`x`	the matrix to cross-validate.
`krow`	the number of row folds (for Gabriel-style CV).
`kcol`	the number of column folds (for Gabriel-style CV).
`maxrank`	the maximum rank to cross-validate up to.
`k`	the number of folds (for Wold-style CV).
`tol`	the convergence tolerance for `impute.svd`.
`maxiter`	the maximum number of iterations for `impute.svd`.

Details

These functions are for cross-validating the SVD of a matrix. They assume a model $X = U D V' + E$ with the terms being signal and noise, and try to find the best rank to truncate the SVD of `x` at for minimizing prediction error. Here, prediction error is measured as sum of squares of residuals between the truncated SVD and the signal part.

For both types of cross-validation, in each replicate we leave out part of the matrix, fit an SVD approximation to the left-in part, and measure prediction error on the left-out part.

In Wold-style cross-validation, the holdout set is "speckled", a random set of elements in the matrix. The missing elements are predicted using `impute.svd`.

In Gabriel-style cross-validation, the holdout set is "blocked". We permute the rows and columns of the matrix, and leave out the lower-right block. We use a modified Schur-complement to predict the held-out block. In Gabriel-style, there are `krow*kcol` total folds.

Value

`call`	the function call
`msep`	the mean square error of prediction (MSEP); this is a matrix whose columns contain the mean square errors in the predictions of the holdout sets for ranks 0, 1, ..., `maxrank` across the different replicates.
`maxrank`	the maximum rank for which prediction error is estimated; this is equal to `nrow(msep)+1`.
`krow`	the number of row folds (for Gabriel-style only).
`kcol`	the number of column folds (for Gabriel-style only).
`rowsets`	the partition of rows into `krow` holdout sets (for Gabriel-style only).
`colsets`	the partition of the columns into `kcol` holdout sets (for Gabriel-style only).
`k`	the number of folds (for Wold-style only).
`sets`	the partition of indices into `k` holdout sets (for Wold-style only).

Note

Gabriel's version of cross-validation was for leaving out a single element of the matrix, which corresponds to n-by-p-fold. Owen and Perry generalized Gabriel's idea to larger holdouts, showing that 2-by-2-fold cross-validation often works better.

Wold's original version of cross-validation did not use the EM algorithm to estimate the SVD. He recommend using the NIPALS algorithm instead, which has since faded into obscurity.

Wold-style cross-validation takes a lot more computation than Gabriel-style. The `maxrank`, `tol`, and `maxiter` have been chosen to give up some accuracy in the name of expediency. They may need to be adjusted to get the best results.

Author(s)

Patrick O. Perry

References

Gabriel, K.R. (2002). Le biplot - outil d'explaration de données multidimensionelles. *Journal de la Société française de statistique*, Volume 143 (2002) no. 3-4, pp. 5-55. B.

Owen, A.B. and Perry, P.O. (2009). Bi-cross-validation of the SVD and the non-negative matrix factorization. *Annals of Applied Statistics* **3**(2) 564–594.

Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics* **20**(4) 397–405.

See Also

`impute.svd`, `plot.cvsvd`, `print.cvsvd` `summary.cvsvd`

Examples

```
# generate a rank-2 matrix plus noise
n <- 50; p <- 20; k <- 2
u <- matrix( rnorm( n*k ), n, k )
v <- matrix( rnorm( p*k ), p, k )
e <- matrix( rnorm( n*p ), n, p )
x <- u %*% t(v) + e

# perform 5-fold Wold-style cross-validtion
(cvw <- cv.svd.wold( x, 5, maxrank=10 ))

# perform (2,2)-fold Gabriel-style cross-validation
(cvg <- cv.svd.gabriel( x, 2, 2, maxrank=10 ))
```

`impute.svd` *Missing value imputation via the SVDImpute algorithm*

Description

Given a matrix with missing values, impute the missing entries using a low-rank SVD approximation estimated by the EM algorithm.

Usage

```
impute.svd(x, k = min(n, p), tol = max(n, p) * 1e-10, maxiter = 100)
```

Arguments

`x`	a matrix to impute the missing entries of.
`k`	the rank of the SVD approximation.
`tol`	the convergence tolerance for the EM algorithm.
`maxiter`	the maximum number of EM steps to take.

Details

Impute the missing values of `x` as follows: First, initialize all `NA` values to the column means, or `0` if all entries in the column are missing. Then, until convergence, compute the first `k` terms of the SVD of the completed matrix. Replace the previously missing values with their approximations from the SVD, and compute the RSS between the *non-missing* values and the SVD.

Declare convergence if `abs(rss0 - rss1) / (.Machine$double.eps + rss1) < tol` , where `rss0` and `rss1` are the RSS values computed from successive iterations. Stop early after `maxiter` iterations and issue a warning.

Value

x	the completed version of the matrix.
rss	the sum of squares between the SVD approximation and the non-missing values in x.
iter	the number of EM iterations before algorithm stopped.

Author(s)

Patrick O. Perry

References

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D. and Altman, R.B. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics* **17**(6), 520–525.

See Also

cv.svd.wold

Examples

```
# Generate a matrix with missing entries
n <- 20
p <- 10
u <- rnorm( n )
v <- rnorm( p )
xfull <- u %*% rbind( v ) + rnorm( n*p )
miss  <- sample( seq_len( n*p ), n )
x       <- xfull
x[miss] <- NA

# impute the missing entries with a rank-1 SVD approximation
xhat <- impute.svd( x, 1 )$x

# compute the prediction error for the missing entries
sum( ( xfull-xhat )^2 )
```

---

plot.cvsvd *Plot the Result of an SVD Cross-Validation*

Description

Plot the result of cv.svd.gabriel or cv.svd.wold, optionally with error bars.

Usage

```
## S3 method for class 'cvsvd'
plot(
  x,
  errorbars = TRUE,
  add = FALSE,
  xlab = "Rank",
  ylab = "Mean Sq. Prediction Error",
  col = "blue",
  col.errorbars = "gray50",
  ...
)
```

Arguments

x	the result of a cv.svd.gabriel or link{cv.svd.wold} computation.
errorbars	indicates whether or not to add error bars.
add	indicates whether or not to add to the current plot.
xlab	the label for the x axis.
ylab	the label for the y axis.
col	the color to use for showing prediction error.
col.errorbars	the color to use for the error bars.
...	additional arguments for plot.

Details

Plot the result of cv.svd.gabriel or cv.svd.wold. This plots a the estimated prediction error as a function of rank, optionally with error bars.

If add is TRUE, the current plot is not cleared.

Author(s)

Patrick O. Perry

See Also

cv.svd.gabriel, cv.svd.wold, print.cvsvd summary.cvsvd

Examples

```
# generate a rank-2 matrix plus noise
n <- 50; p <- 20; k <- 2
u <- matrix( rnorm( n*k ), n, k )
v <- matrix( rnorm( p*k ), p, k )
e <- matrix( rnorm( n*p ), n, p )
x <- u %*% t(v) + e
```

```
# perform 5-fold Wold-style cross-validtion
cvw <- cv.svd.wold( x, 5, maxrank=10 )

# perform (2,2)-fold Gabriel-style cross-validation
cvg <- cv.svd.gabriel( x, 2, 2, maxrank=10 )

# plot the results
par( mfrow=c(2,1) )
plot( cvw, main="Wold-style CV")
plot( cvg, main="Gabriel-style CV")
```

`print.cvsvd` *Print the Result of an SVD Cross-Validation*

Description

Print the result of `cv.svd.gabriel` or `cv.svd.wold`.

Usage

```
## S3 method for class 'cvsvd'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	the result of a `cv.svd.gabriel` or `cv.svd.wold` computation.
`digits`	the digits of precision to show in the output.
`...`	additional arguments to `print`.

Details

Print a table of the estimated prediction errors and the standard errors of the estimate. Put an asterisk (`*`) next to the minimum and a plus (`+`) next to the "one standard error rule" choice.

Author(s)

Patrick O. Perry

See Also

`cv.svd.gabriel`, `cv.svd.wold`, `plot.cvsvd` `summary.cvsvd`

summary.cvsvd *Summarize the Result of an SVD Cross-Validation*

Description

Summarize the result of cv.svd.gabriel or cv.svd.wold.

Usage

```
## S3 method for class 'cvsvd'
summary(object, ...)
```

Arguments

object	the result of a cv.svd.gabriel or cv.svd.wold computation.
...	additional arguments to summary.

Details

Print a table of the estimated prediction errors and the standard errors of the estimate. Put an asterisk (*) next to the minimum and a plus (+) next to the "one standard error rule" choice.

Value

nfolds	the number of cross-validation folds
maxrank	the maximum rank for which prediction error is estimated.
msep.mean	the average mean square error of prediction (MSEP) across all folds for ranks 0, 1, ..., maxrank.
msep.se	the standard errors of the msep.mean estimates.
rank.best	the rank with the minimum msep.mean value.
rank.1se	the smallest rank within one standard error of the minimum msep.mean value.

Author(s)

Patrick O. Perry

See Also

cv.svd.gabriel, cv.svd.wold, plot.cvsvd print.cvsvd

Index

∗ **package**
bcv-package, 2

bcv *(*bcv-package*)*, 2
bcv-package, 2

cv.svd *(*cv.svd.gabriel*)*, 3
cv.svd.gabriel, *2*, 3, *6–9*
cv.svd.wold, *2*, *6–9*

impute.svd, *2–4*, 5

plot.cvsvd, *2*, *4*, 6, *8, 9*
print.cvsvd, *2*, *4*, *7*, 8, *9*

summary.cvsvd, *2*, *4*, *7, 8*, 9