Package ‘softImpute’

October 14, 2022

Type Package

Title Matrix Completion via Iterative Soft-Thresholded SVD

Version 1.4-1

Date 2021-5-08

Author
Trevor Hastie <hastie@stanford.edu> and Rahul Mazumder <rahul.mazumder@gmail.com>

Maintainer Trevor Hastie <hastie@stanford.edu>

Description Iterative methods for matrix completion that use nuclear-norm regularization. There are two main approaches.The one approach uses iterative soft-thresholded svds to impute the missing values. The second approach uses alternating least squares. Both have an 'EM' flavor, in that at each iteration the matrix is completed with the current estimate. For large matrices there is a special sparse-matrix class named ``Incomplete'' that efficiently handles all computations. The package includes procedures for centering and scaling rows, columns or both, and for computing low-rank SVDs on large sparse centered matrices (i.e. principal components).

License GPL-2

Depends Matrix, methods

Suggests knitr, rmarkdown

VignetteBuilder knitr

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-05-09 05:10:03 UTC

R topics documented:

`biScale` *standardize a matrix to have optionally row means zero and variances one, and/or column means zero and variances one.*

Description

A function for standardizing a matrix in a symmetric fashion. Generalizes the `scale` function in R. Works with matrices with NAs, matrices of class "Incomplete", and matrix in "sparseMatrix" format.

Usage

```
biScale(x, maxit = 20, thresh = 1e-09, row.center = TRUE, row.scale =TRUE,
        col.center = TRUE, col.scale = TRUE, trace = FALSE)
```

Arguments

`x`	matrix, possibly with NAs, also of class "Incomplete" or "sparseMatrix" format.
`maxit`	When both row and column centering/scaling is requested, iteration is may be necessary
`thresh`	Convergence threshold
`row.center`	if `row.center==TRUE` (the default), row centering will be performed resulting in a matrix with row means zero. If `row.center` is a vector, it will be used to center the rows. If `row.center=FALSE` nothing is done. See details for more info.
`row.scale`	if `row.scale==TRUE`, the rows are scaled (after possibly centering, to have variance one. Alternatively, if a positive vector is supplied, it is used for row centering. See details for more info.
`col.center`	Similar to `row.center`
`col.scale`	Similar to `row.scale`
`trace`	with `trace=TRUE`, convergence progress is reported, when iteration is needed.

Details

This function computes a transformation

$$\frac{X_{ij} - \alpha_i - \beta_j}{\gamma_i \tau_j}$$

to transform the matrix X. It uses an iterative algorithm based on "method-of-moments". At each step, all but one of the parameter vectors is fixed, and the remaining vector is computed to solve the required condition. Although in genereal this is not guaranteed to converge, it mostly does, and quite rapidly. When there are convergence problems, remove some of the required constraints. When any of the row/column centers or scales are provided, they are used rather than estimated in the above model.

Value

A matrix like `x` is returned, with attributes:

`biScale:row` a list with elements `"center"` and `"scale"` (the $alpha$ and $gamma$ above. If no centering was done, the center component will be a vector of zeros. Likewise, of no row scaling was done, the scale component will be a vector of ones.

`biScale:column` Same details as `biScale:row`

For matrices with missing values, the constraints apply to the non-missing entries. If `x` is of class `"sparseMatrix"`, then the sparsity is maintained, and an object of class `"SparseplusLowRank"` is returned, such that the low-rank part does the centering.

Note

This function will be described in detail in a forthcoming paper

Author(s)

Trevor Hastie, with help from Andreas Buja and Steven Boyd
, Maintainer: Trevor Hastie `<hastie@stanford.edu>`

See Also

`softImpute`,`Incomplete`,`lambda0`,`impute`,`complete`, and class `"SparseplusLowRank"`

Examples

```
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
xc=biScale(x)
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
```

```
xna=x
xna[imiss]=NA
xnab=biScale(xna,row.scale=FALSE,trace=TRUE)
xnaC=as(xna,"Incomplete")
xnaCb=biScale(xnaC)
nnz=trunc(np*.3)
inz=sample(seq(np),nnz,replace=FALSE)
i=row(x)[inz]
j=col(x)[inz]
x=rnorm(nnz)
xS=sparseMatrix(x=x,i=i,j=j)
xSb=biScale(xS)
class(xSb)
```

`complete` *make predictions from an svd object*

Description

These functions produce predictions from the low-rank solution of `softImpute`

Usage

```
complete(x, object, unscale = TRUE)
impute(object, i, j, unscale = TRUE)
```

Arguments

`x`	a matrix with NAs or a matrix of class `"Incomplete"`.
`object`	an svd object with components u, d and v
`i`	vector of row indices for the locations to be predicted
`j`	vector of column indices for the locations to be predicted
`unscale`	if `object` has `biScale` attributes, and `unscale=TRUE`, the imputations reversed the centering and scaling on the predictions.

Details

`impute` returns a vector of predictions, using the reconstructed low-rank matrix representation represented by `object`. It is used by complete, which returns a complete matrix with all the missing values imputed.

Value

Either a vector of predictions or a complete matrix. WARNING: if `x` has large dimensions, the matrix returned by `complete` might be too large.

Author(s)

Trevor Hastie

See Also

`softImpute`, `biScale` and `Incomplete`

Examples

```
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
fit1=softImpute(xna,rank=50,lambda=30)
complete(xna,fit1)
```

---

deBias — *Recompute the* `$d` *component of a* `"softImpute"` *object through regression.*

Description

`softImpute` uses shrinkage when completing a matrix with missing values. This function debiases the singular values using ordinary least squares.

Usage

```
deBias(x, svdObject)
```

Arguments

`x`	matrix with missing entries, or a matrix of class `"Incomplete"`
`svdObject`	an SVD object, the output of `softImpute`

Details

Treating the `"d"` values as parameters, this function recomputes them by linear regression.

Value

An svd object is returned, with components "u", "d", and "v".

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie <hastie@stanford.edu>

Examples

```
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
fit1=softImpute(xna,rank=50,lambda=30)
fit1d=deBias(xna,fit1)
```

`Incomplete` *create a matrix of class* `Incomplete`

Description

creates an object of class `Incomplete`, which inherits from class `dgCMatrix`, a specific instance of class `sparseMatrix`

Usage

```
Incomplete(i, j, x)
```

Arguments

`i`	row indices
`j`	column indices
`x`	a vector of values

Details

The matrix is represented in sparse-matrix format, except the "zeros" represent missing values. Real zeros are represented explicitly as values.

Value

a matrix of class `Incomplete` which inherits from class `dgCMatrix`

Author(s)

Trevor Hastie and Rahul Mazumder

See Also

`softImpute`

Examples

```
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
xnaC=as(xna,"Incomplete")
### here we do it a different way to demonstrate Incomplete
### In practise the observed values are stored in this market-matrix format.
i = row(xna)[-imiss]
j = col(xna)[-imiss]
xnaC=Incomplete(i,j,x=x[-imiss])
```

`Incomplete-class` *Class* `"Incomplete"`

Description

a sparse matrix inheriting from class `dgCMatrix` with the NAs represented as zeros

Objects from the Class

Objects can be created by calls of the form `new("Incomplete", ...)`. or by calling the function `Incomplete`

Slots

`i`**:** Object of class `"integer"` ~~

`p`**:** Object of class `"integer"` ~~

`Dim`**:** Object of class `"integer"` ~~

`Dimnames`**:** Object of class `"list"` ~~

`x`**:** Object of class `"numeric"` ~~

`factors`**:** Object of class `"list"` ~~

Extends

Class "`dgCMatrix`", directly. Class "`CsparseMatrix`", by class "dgCMatrix", distance 2. Class "`dsparseMatrix`", by class "dgCMatrix", distance 2. Class "`generalMatrix`", by class "dgCMatrix", distance 2. Class "`dMatrix`", by class "dgCMatrix", distance 3. Class "`sparseMatrix`", by class "dgCMatrix", distance 3. Class "`compMatrix`", by class "dgCMatrix", distance 3. Class "`Matrix`", by class "dgCMatrix", distance 4.

Methods

as.matrix `signature(x = "Incomplete")`: ...

coerce `signature(from = "matrix", to = "Incomplete")`: ...

complete `signature(x = "Incomplete")`: ...

Author(s)

Trevor Hastie and Rahul Mazumder

See Also

`biScale`,`softImpute`,`Incomplete`,`impute`,`complete`

Examples

```
showClass("Incomplete")
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
xnaC=as(xna,"Incomplete")
```

`lambda0` *compute the smallest value for* `lambda` *such that* `softImpute(x,lambda)` *returns the zero solution.*

Description

this determines the "starting" lambda for a sequence of values for `softImpute`, and all nonzero solutions would require a smaller value for `lambda`.

Usage

```
lambda0(x, lambda = 0, maxit = 100, trace.it = FALSE, thresh = 1e-05)
```

Arguments

`x`	An m by n matrix. Large matrices can be in "sparseMatrix" format, as well as "SparseplusLowRank". The latter arise after centering sparse matrices, for example with `biScale`, as well as in applications such as `softImpute`. The remaining arguments only apply to matrices `x` in `"sparseMatrix"`, `"Incomplete"`, or `"SparseplusLowRank"` format.
`lambda`	As in `svd.als`, using a value for `lambda` can speed up iterations. As long as the solution is not zero, the value returned adds back this value.
`maxit`	maximum number of iterations.
`trace.it`	with `trace.it=TRUE`, convergence progress is reported.
`thresh`	convergence threshold, measured as the relative changed in the Frobenius norm between two successive estimates.

Details

It is the largest singular value for the matrix, with zeros replacing missing values. It uses `svd.als` with `rank=2`.

Value

a single number, the largest singular value

Author(s)

Trevor Hastie, Rahul Mazumder
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Rahul Mazumder, Trevor Hastie and Rob Tibshirani (2010) *Spectral Regularization Algorithms for Learning Large Incomplete Matrices*, https://web.stanford.edu/~hastie/Papers/mazumder10a.pdf
Journal of Machine Learning Research 11 (2010) 2287-2322

See Also

`softImpute`,`Incomplete`, and `svd.als`.

Examples

```
set.seed(101)
n=200
p=100
J=50
```

```
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
lambda0(xna)
```

`softImpute` *impute missing values for a matrix via nuclear-norm regularization.*

Description

fit a low-rank matrix approximation to a matrix with missing values via nuclear-norm regularization. The algorithm works like EM, filling in the missing values with the current guess, and then solving the optimization problem on the complete matrix using a soft-thresholded SVD. Special sparse-matrix classes available for very large matrices.

Usage

```
softImpute(x, rank.max = 2, lambda = 0, type = c("als", "svd"), thresh = 1e-05,
           maxit = 100, trace.it = FALSE, warm.start = NULL, final.svd = TRUE)
```

Arguments

`x`	An m by n matrix with NAs. For large matrices can be of class `"Incomplete"`, in which case the missing values are represented as pseudo zeros leading to dramatic storage reduction. `x` can have been centered and scaled via `biScale`, and this information is carried along with the solution.
`rank.max`	This restricts the rank of the solution. If sufficiently large, and with `type="svd"`, the solution solves the nuclear-norm convex matrix-completion problem. In this case the number of nonzero singular values returned will be less than or equal to `rank.max`. If smaller ranks are used, the solution is not guaranteed to solve the problem, although still results in good local minima. `rank.max` should be no bigger than `min(dim(x)-1`.
`lambda`	nuclear-norm regularization parameter. If `lambda=0`, the algorithm reverts to "hardImpute", for which convergence is typically slower, and to local minimum. Ideally `lambda` should be chosen so that the solution reached has rank slightly less than `rank.max`. See also `lambda0()` for computing the smallest `lambda` with a zero solution.
`type`	two algorithms are implements, `type="svd"` or the default `type="als"`. The "svd" algorithm repeatedly computes the svd of the completed matrix, and soft thresholds its singular values. Each new soft-thresholded svd is used to re-impute the missing entries. For large matrices of class `"Incomplete"`, the svd is achieved by an efficient form of alternating orthogonal ridge regression. The

"als" algorithm uses this same alternating ridge regression, but updates the imputation at each step, leading to quite substantial speedups in some cases. The "als" approach does not currently have the same theoretical convergence guarantees as the "svd" approach.

`thresh` convergence threshold, measured as the relative change in the Frobenius norm between two successive estimates.

`maxit` maximum number of iterations.

`trace.it` with `trace.it=TRUE`, convergence progress is reported.

`warm.start` an svd object can be supplied as a warm start. This is particularly useful when constructing a path of solutions with decreasing values of `lambda` and increasing `rank.max`. The previous solution can be provided directly as a warm start for the next.

`final.svd` only applicable to `type="als"`. The alternating ridge-regressions do not lead to exact zeros. With the default `final.svd=TRUE`, at the final iteration, a one step unregularized iteration is performed, followed by soft-thresholding of the singular values, leading to hard zeros.

Details

SoftImpute solves the following problem for a matrix X with missing entries:

$$\min ||X - M||_o^2 + \lambda ||M||_*.$$

Here $||\cdot||_o$ is the Frobenius norm, restricted to the entries corresponding to the non-missing entries of X, and $||M||_*$ is the nuclear norm of M (sum of singular values). For full details of the "svd" algorithm are described in the reference below. The "als" algorithm will be described in a forthcoming article. Both methods employ special sparse-matrix tricks for large matrices with many missing values. This package creates a new sparse-matrix class `"SparseplusLowRank"` for matrices of the form

$$x + ab',$$

where x is sparse and a and b are tall skinny matrices, hence ab' is low rank. Methods for efficient left and right matrix multiplication are provided for this class. For large matrices, the function `Incomplete()` can be used to build the appropriate sparse input matrix from market-format data.

Value

An svd object is returned, with components "u", "d", and "v". If the solution has zeros in "d", the solution is truncated to rank one more than the number of zeros (so the zero is visible). If the input matrix had been centered and scaled by `biScale`, the scaling details are assigned as attributes inherited from the input matrix.

Author(s)

Trevor Hastie, Rahul Mazumder
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Rahul Mazumder, Trevor Hastie and Rob Tibshirani (2010) *Spectral Regularization Algorithms for Learning Large Incomplete Matrices*, https://web.stanford.edu/~hastie/Papers/mazumder10a.pdf
Journal of Machine Learning Research 11 (2010) 2287-2322

See Also

biScale, svd.als,Incomplete, lambda0, impute, complete

Examples

```
set.seed(101)
n=200
p=100
J=50
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
###uses regular matrix method for matrices with NAs
fit1=softImpute(xna,rank=50,lambda=30)
###uses sparse matrix method for matrices of class "Incomplete"
xnaC=as(xna,"Incomplete")
fit2=softImpute(xnaC,rank=50,lambda=30)
###uses "svd" algorithm
fit3=softImpute(xnaC,rank=50,lambda=30,type="svd")
ximp=complete(xna,fit1)
### first scale xna
xnas=biScale(xna)
fit4=softImpute(xnas,rank=50,lambda=10)
ximp=complete(xna,fit4)
impute(fit4,i=c(1,3,7),j=c(2,5,10))
impute(fit4,i=c(1,3,7),j=c(2,5,10),unscale=FALSE)#ignore scaling and centering
```

SparseplusLowRank-class

Class "SparseplusLowRank"

Description

A structured matrix made up of a sparse part plus a low-rank part, all which can be stored and operated on efficiently.

Objects from the Class

Objects can be created by calls of the form `new("SparseplusLowRank", ...)` or by a call to `splr`

Slots

`x`: Object of class `"sparseMatrix"`

`a`: Object of class `"matrix"`

`b`: Object of class `"matrix"`

Methods

%*% `signature(x = "ANY", y = "SparseplusLowRank")`: ...

%*% `signature(x = "SparseplusLowRank", y = "ANY")`: ...

%*% `signature(x = "Matrix", y = "SparseplusLowRank")`: ...

%*% `signature(x = "SparseplusLowRank", y = "Matrix")`: ...

as.matrix `signature(x = "SparseplusLowRank")`: ...

colMeans `signature(x = "SparseplusLowRank")`: ...

colSums `signature(x = "SparseplusLowRank")`: ...

dim `signature(x = "SparseplusLowRank")`: ...

norm `signature(x = "SparseplusLowRank", type = "character")`: ...

rowMeans `signature(x = "SparseplusLowRank")`: ...

rowSums `signature(x = "SparseplusLowRank")`: ...

svd.als `signature(x = "SparseplusLowRank")`: ...

Author(s)

Trevor Hastie and Rahul Mazumder

See Also

`softImpute`,`splr`

Examples

```
showClass("SparseplusLowRank")
x=matrix(sample(c(3,0),15,replace=TRUE),5,3)
x=as(x,"sparseMatrix")
a=matrix(rnorm(10),5,2)
b=matrix(rnorm(6),3,2)
new("SparseplusLowRank",x=x,a=a,b=b)
splr(x,a,b)
```

splr — *create a* `SparseplusLowRank` *object*

Description

create an object of class `SparseplusLowRank` which can be efficiently stored and for which efficient linear algebra operations are possible.

Usage

```
splr(x, a = NULL, b = NULL)
```

Arguments

`x`	sparse matrix with dimension say m x n
`a`	matrix with m rows and number of columns r less than `min(dim(x))`
`b`	matrix with n rows and number of columns r less than `min(dim(x))`

Value

an object of S4 class `SparseplusLowRank` is returned with slots `x`, `a` and `b`

Author(s)

Trevor Hastie

See Also

`SparseplusLowRank-class`, `softImpute`

Examples

```
x=matrix(sample(c(3,0),15,replace=TRUE),5,3)
x=as(x,"sparseMatrix")
a=matrix(rnorm(10),5,2)
b=matrix(rnorm(6),3,2)
new("SparseplusLowRank",x=x,a=a,b=b)
splr(x,a,b)
```

`svd.als` *compute a low rank soft-thresholded svd by alternating orthogonal ridge regression*

Description

fit a low-rank svd to a complete matrix by alternating orthogonal ridge regression. Special sparse-matrix classes available for very large matrices, including "SparseplusLowRank" versions for row and column centered sparse matrices.

Usage

```
svd.als(x, rank.max = 2, lambda = 0, thresh = 1e-05, maxit = 100,
        trace.it = FALSE, warm.start = NULL, final.svd = TRUE)
```

Arguments

`x`	An m by n matrix. Large matrices can be in "sparseMatrix" format, as well as "SparseplusLowRank". The latter arise after centering sparse matrices, for example with `biScale`, as well as in applications such as `softImpute`.
`rank.max`	The maximum rank for the solution. This is also the dimension of the left and right matrices of orthogonal singular vectors. 'rank.max' should be no bigger than 'min(dim(x)'.
`lambda`	The regularization parameter. `lambda=0` corresponds to an accelerated version of the orthogonal QR-algorithm. With `lambda>0` the algorithm amounts to alternating orthogonal ridge regression.
`thresh`	convergence threshold, measured as the relative changed in the Frobenius norm between two successive estimates.
`maxit`	maximum number of iterations.
`trace.it`	with `trace.it=TRUE`, convergence progress is reported.
`warm.start`	an svd object can be supplied as a warm start. If the solution requested has higher rank than the warm start, the additional subspace is initialized with random Gaussians (and then orthogonalized wrt the rest).
`final.svd`	Although in theory, this algorithm converges to the solution to a nuclear-norm regularized low-rank matrix approximation problem, with potentially some singular values equal to zero, in practice only near-zeros are achieved. This final step does one more iteration with `lambda=0`, followed by soft-thresholding.

Details

This algorithm solves the problem

$$\min ||X - M||_F^2 + \lambda||M||_*$$

subject to $rank(M) \leq r$, where $||M||_*$ is the nuclear norm of M (sum of singular values). It achieves this by solving the related problem

$$\min ||X - AB'||_F^2 + \lambda/2(||A||_F^2 + ||B||_F^2)$$

subject to $rank(A) = rank(B) \leq r$. The solution is a rank-restricted, soft-thresholded SVD of X.

Value

An svd object is returned, with components "u", "d", and "v".

u	an m by `rank.max` matrix with the left orthogonal singular vectors
d	a vector of length `rank.max` of soft-thresholded singular values
v	an n by `rank.max` matrix with the right orthogonal singular vectors

Author(s)

Trevor Hastie, Rahul Mazumder
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Rahul Mazumder, Trevor Hastie and Rob Tibshirani (2010) *Spectral Regularization Algorithms for Learning Large Incomplete Matrices*, https://web.stanford.edu/~hastie/Papers/mazumder10a.pdf
Journal of Machine Learning Research 11 (2010) 2287-2322

See Also

`biScale`, `softImpute`, `Incomplete`, `lambda0`, `impute`, `complete`

Examples

```
#create a matrix and run the algorithm
set.seed(101)
n=100
p=50
J=25
np=n*p
x=matrix(rnorm(n*J),n,J)%*%matrix(rnorm(J*p),J,p)+matrix(rnorm(np),n,p)/5
fit=svd.als(x,rank=25,lambda=50)
fit$d
pmax(svd(x)$d-50,0)
# now create a sparse matrix and do the same
nnz=trunc(np*.3)
inz=sample(seq(np),nnz,replace=FALSE)
i=row(x)[inz]
j=col(x)[inz]
x=rnorm(nnz)
xS=sparseMatrix(x=x,i=i,j=j)
fit2=svd.als(xS,rank=20,lambda=7)
```

```
fit2$d
pmax(svd(as.matrix(xS))$d-7,0)
```

Index