Package ‘filling’

October 13, 2022

Type Package

Title Matrix Completion, Imputation, and Inpainting Methods

Version 0.2.3

Description Filling in the missing entries of a partially observed data is one of fundamental problems in various disciplines of mathematical science. For many cases, data at our interests have canonical form of matrix in that the problem is posed upon a matrix with missing values to fill in the entries under preset assumptions and models. We provide a collection of methods from multiple disciplines under Matrix Completion, Imputation, and Inpainting. See Davenport and Romberg (2016) <doi:10.1109/JSTSP.2016.2539100> for an overview of the topic.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 2.14.0)

Imports CVXR (>= 1.0), Rcpp, Rdpack, ROptSpace, RSpectra, nabor, stats, utils

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Author Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>)

Maintainer Kisung You <kisungyou@outlook.com>

Repository CRAN

Date/Publication 2021-08-21 15:00:02 UTC

R topics documented:

filling-package .......................................................... 2
aux.rndmissing .......................................................... 2
fill.HardImpute ....................................................... 3
fill.KNNimpute ........................................................ 5
Description

Filling in the missing entries of a partially observed data is one of fundamental problems in various disciplines of mathematical science. For many cases, data of interest have canonical form of matrix in that the problem is posed upon a matrix with missing values to fill in the entries under preset assumptions and models. We provide a collection of methods from multiple disciplines under Matrix Completion, Imputation, and Inpainting. Currently, we have following methods implemented.

<table>
<thead>
<tr>
<th>Name of Function</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill.HardImpute</td>
<td>Generalized Spectral Regularization</td>
</tr>
<tr>
<td>fill.KNNimpute</td>
<td>Weighted $K$-nearest Neighbors</td>
</tr>
<tr>
<td>fill.nuclear</td>
<td>Nuclear Norm Optimization</td>
</tr>
<tr>
<td>fill.OptSpace</td>
<td>OptSpace</td>
</tr>
<tr>
<td>fill.simple</td>
<td>Simple Rules of Mean, Median, and Random</td>
</tr>
<tr>
<td>fill.SVDimpute</td>
<td>Spectral Regularization</td>
</tr>
<tr>
<td>fill.SVT</td>
<td>Iterative Regression against Right Singular Vectors</td>
</tr>
<tr>
<td>fill.USVT</td>
<td>Singular Value Thresholding for Nuclear Norm Optimization</td>
</tr>
</tbody>
</table>

aux.rndmissing

Randomly assign NAs to the data matrix with probability $x$

Description

aux.rndmissing randomly selects $100 \cdot x\%$ of entries from a given data matrix and turns them into missing entries, i.e., their values become NA.

Usage

aux.rndmissing(A, x = 0.1)
**fill.HardImpute**

### Arguments

- **A**: an \((n \times p)\) data matrix.
- **x**: percentage of turning current entries into missing (NA).

### Value

an \((n \times p)\) data matrix with missing entries at proportion \(x\).

### Examples

```r
# load lena64 image matrix
data(lena64)

# generate 10% of missing values
lena64_miss <- aux.rndmissing(lena64)

# visualize
par(mfrow=c(1,2))
image(lena64, axes=FALSE, main="original image")
image(lena64_miss, axes=FALSE, main="10% missing entries")
```

### Description

If the assumed underlying model has sufficiently many zeros, the LASSO type shrinkage estimator is known to overestimate the number of non-zero coefficients. `fill.HardImpute` aims at overcoming such difficulty via low-rank assumption and hard thresholding idea, well-known concept in conventional regression analysis. In algorithmic aspect, it takes output of `SoftImpute` as warm-start matrices for iterative estimation process.

### Usage

```r
fill.HardImpute(
  A,
  lambdas = c(10, 1, 0.1),
  maxiter = 100,
  tol = 0.001,
  rk = (min(dim(A)) - 1)
)
```
Arguments

- **A**: an \((n \times p)\) partially observed matrix.
- **lambdas**: a length-\(t\) vector regularization parameters.
- **maxiter**: maximum number of iterations to be performed.
- **tol**: stopping criterion for an incremental progress.
- **rk**: assumed rank of the matrix.

Value

A named list containing

- **X**: an \((n \times p \times t)\) cubic array after completion at each lambda value.

References


See Also

- `fill.SoftImpute`

Examples

```r
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
A <- aux.rndmissing(lena128, x=0.05)

## apply the method with 3 rank conditions
fill1 <- fill.HardImpute(A, lambdas=c(500,100,50), rk=10)
fill2 <- fill.HardImpute(A, lambdas=c(500,100,50), rk=50)
fill3 <- fill.HardImpute(A, lambdas=c(500,100,50), rk=100)

## visualize only the last ones from each run
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill1$X[, ,3], col=gray((0:100)/100), axes=FALSE, main="Rank 10")
image(fill2$X[, ,3], col=gray((0:100)/100), axes=FALSE, main="Rank 50")
image(fill3$X[, ,3], col=gray((0:100)/100), axes=FALSE, main="Rank 100")
par(opar)
```
Description

One of the simplest idea to guess missing entry is to use portion of the data that has most similar characteristics across all covariates. fill.KNNimpute follows such reasoning in that it finds $K$-nearest neighbors based on observed variables and uses weighted average of nearest elements to fill in the missing entry. Note that when there are many missing entries, it’s possible that there are no surrogates to be computed upon. Therefore, if there exists an entire row or column full of missing entries, the algorithm stops.

Usage

fill.KNNimpute(A, k = ceiling(nrow(A)/2))

Arguments

- A: an $(n \times p)$ partially observed matrix.
- k: the number of neighbors to use.

Value

- a named list containing
  - X: an $(n \times p)$ matrix after completion.

References


See Also

fill.SVDimpute

Examples

```r
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
set.seed(5)
A <- aux.rndmissing(lena128, x=0.05)

## apply the method with 3 different neighborhood size
fill1 <- fill.KNNimpute(A, k=5)
fill2 <- fill.KNNimpute(A, k=25)
```
fill3 <- fill.KNNimpute(A, k=50)

## visualize only the last ones from each run
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill1$X, col=gray((0:100)/100), axes=FALSE, main="5-neighbor")
image(fill2$X, col=gray((0:100)/100), axes=FALSE, main="25-neighbor")
image(fill3$X, col=gray((0:100)/100), axes=FALSE, main="50-neighbor")
par(opar)

fill.nuclear

Low-Rank Completion with Nuclear Norm Optimization

Description
In many circumstances, it is natural to assume that there exists an underlying low-rank structure. The assumption of low-rank property leads to an optimization problem for matrix completion problem,

$$\text{minimize} \quad \text{rank}(X)$$

$$\text{s.t} \quad X_{ij} = A_{ij} \quad \text{for} \quad A_{ij} \in E$$

where $A_{ij} \in E$ means the $(i,j)$-th entry of data matrix $A$ is not missing. The objective function can be further relaxed by nuclear norm

$$\|X\|_* = \sum \sigma_i(X)$$

where $\sigma_i(X)$ is $i$-th singular value of the matrix $X$. Note that for modeling purpose, we adopted closeness parameter tolerance for equality constraint. CVXR package was used in implementation. Computational efficiency may not be guaranteed for large data matrix.

Usage
fill.nuclear(A, tolerance = 0.001)

Arguments

A an $(n \times p)$ partially observed matrix.
tolerance level of tolerance for entrywise equality condition.

Value

a named list containing

X an $(n \times p)$ matrix after completion.

norm solution of the minimization problem; approximate rank.
cvxr.status “optimal” denotes the problem was solved. See psolve for more details on solvability.
cvxr.niters the number of iterations taken.
cvxr.solver type of solver used by CVXR.
## fill.OptSpace

### Description

OptSpace is an algorithm for matrix completion when a matrix is partially observed. It performs what authors called trimming and projection repeatedly based on singular value decompositions. Original implementation is borrowed from ROptSpace package, which was independently developed by the maintainer. See OptSpace for more details.

### Usage

```r
fill.OptSpace(A, ropt = NA, niter = 50, tol = 1e-06)
```

### Arguments

- `A` an \((n \times p)\) partially observed matrix.
- `ropt` NA to guess the rank, or a positive integer as a pre-defined rank.
- `niter` maximum number of iterations allowed.
- `tol` stopping criterion for reconstruction in Frobenius norm.

### References


---

```r
## Not run:
## load image data of 'lena64'
data(lena64)

## transform 5% of entries into missing
A <- aux.rndmissing(lena64, x=0.05)

## apply the method
filled <- fill.nuclear(A)

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(filled$X, col=gray((0:100)/100), axes=FALSE, main="processed")
par(opar)

## End(Not run)
```
Value

- a named list containing
  - \( \mathbf{X} \) an \((n \times p)\) matrix after completion.
  - error a vector of reconstruction errors for each successive iteration.

References


Examples

```r
## Not run:
## load image data of ’lena64’
data(lena64)

## transform 5% of entries into missing
A <- aux.rndmissing(lena64, x=0.05)

## apply the method with different rank assumptions
filled10 <- fill.OptSpace(A, ropt=10)
filled20 <- fill.OptSpace(A, ropt=20)

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(filled10$X, col=gray((0:100)/100), axes=FALSE, main="rank 10")
image(filled20$X, col=gray((0:100)/100), axes=FALSE, main="rank 20")
par(opar)

## End(Not run)
```

fill.simple  Imputation by Simple Rules

Description

One of the most simplest ways to fill in the missing entries is to apply any simple rule for each variable. In this example, we provide 3 options, "mean", "median", and "random". It assumes that every column has at least one non-missing entries in that for each column, the rule is applied from the subset of non-missing values.

Usage

```r
fill.simple(A, method = c("mean", "median", "random"))
```
**Arguments**

- **A**
  - An \((n \times p)\) partially observed matrix.
- **method**
  - A simple rule to fill in the missing entries in a columnwise manner.

**Value**

A named list containing:

- \(X\) an \((n \times p)\) matrix after completion.

**References**


**Examples**

```r
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
A <- aux.rndmissing(lena128, x=0.05)

## apply all three methods#
fill1 <- fill.simple(A, method="mean")
fill2 <- fill.simple(A, method="median")
fill3 <- fill.simple(A, method="random")

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="original")
image(fill1$X, col=gray((0:100)/100), axes=FALSE, main="method:mean")
image(fill2$X, col=gray((0:100)/100), axes=FALSE, main="method:median")
image(fill3$X, col=gray((0:100)/100), axes=FALSE, main="method:random")
par(opar)
```

**Description**

`fill.SoftImpute` implements convex relaxation techniques to generate a sequence of regularized low-rank solutions for matrix completion problems. For the nuclear norm optimization problem, it uses soft thresholding technique iteratively in that the algorithm returns several matrices in accordance with the provided vector of regularization parameters \(\lambda\) (\(\lambda\)mbdas).
Usage

fill.SoftImpute(A, lambdas = c(10, 1, 0.1), maxiter = 100, tol = 0.001)

Arguments

- **A**: An \((n \times p)\) partially observed matrix.
- **lambdas**: A length-\(t\) vector of regularization parameters.
- **maxiter**: Maximum number of iterations to be performed.
- **tol**: Stopping criterion for an incremental progress.

Value

A named list containing

- \(X\): An \((n \times p \times t)\) cubic array after completion at each lambda value.

References


See Also

- fill.HardImpute

Examples

```r
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
A <- aux.rndmissing(lena128, x=0.05)

## apply the method with 3 lambda values
fill <- fill.SoftImpute(A, lambdas=c(500,100,50))

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill$X[,1], col=gray((0:100)/100), axes=FALSE, main="lambda=500")
image(fill$X[,2], col=gray((0:100)/100), axes=FALSE, main="lambda=100")
image(fill$X[,3], col=gray((0:100)/100), axes=FALSE, main="lambda=50")
par(opar)
```
fill.SVDimpute

Iterative Regression against Right Singular Vectors

Description

Singular Value Decomposition (SVD) is the best low-rank approximation of a given matrix. fill.SVDimpute exploits such idea. First, it starts with simple filling using column mean values for filling. Second, it finds SVD of a current matrix. Then, each row vector is regressed upon top-\(k\) right singular vectors. Missing entries are then filled with predicted estimates.

Usage

```r
fill.SVDimpute(A, k = ceiling(ncol(A)/2), maxiter = 100, tol = 0.01)
```

Arguments

- **A**: an \((n \times p)\) partially observed matrix.
- **k**: the number of regressors to be used.
- **maxiter**: maximum number of iterations to be performed.
- **tol**: stopping criterion for an incremental progress.

Value

A named list containing

- **X**: an \((n \times p)\) matrix after completion.

References


See Also

`fill.KNNimpute`

Examples

```r
## Not run:
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
set.seed(5)
A <- aux.rndmissing(lena128, x=0.05)

## apply the method with 3 different number of regressors
```
fill1 <- fill.SVTimpute(A, k=5)
fill2 <- fill.SVTimpute(A, k=25)
fill3 <- fill.SVTimpute(A, k=50)

## visualize only the last ones from each run
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill1$X, col=gray((0:100)/100), axes=FALSE, main="5 regressors")
image(fill2$X, col=gray((0:100)/100), axes=FALSE, main="25 regressors")
image(fill3$X, col=gray((0:100)/100), axes=FALSE, main="50 regressors")
par(opar)

## End(Not run)

---

fill.SVT  
Singular Value Thresholding for Nuclear Norm Optimization

Description

fill.SVT is an iterative updating scheme for Nuclear Norm Minimization problem. An unconstrained paraphrase of the problem introduced in fill.nuclear is

\[
\text{minimize } \frac{1}{2} \| P_{\Omega}(X - A) \|_F^2 + \lambda \| X \|_*
\]

where \( P_{\Omega}(X) = X_{ij} \) if it is observed, or 0 otherwise. It performs iterative shrinkage on newly computed singular values.

Usage

fill.SVT(A, lambda = 1, maxiter = 100, tol = 0.001)

Arguments

- \( A \) an \((n \times p)\) partially observed matrix.
- \( \lambda \) a regularization parameter.
- \( \text{maxiter} \) maximum number of iterations to be performed.
- \( \text{tol} \) stopping criterion for an incremental progress.

Value

a named list containing

- \( X \) an \((n \times p)\) matrix after completion.
**References**


**See Also**

`fill.nuclear`

**Examples**

```r
## Not run:
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
A <- aux.rndmissing(lena128, x=0.05)

## apply the method
fill1 <- fill.SVT(A, lambda=0.1)
fill2 <- fill.SVT(A, lambda=1.0)
fill3 <- fill.SVT(A, lambda=20)

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill1$X, col=gray((0:100)/100), axes=FALSE, main="lbd=0.1")
image(fill2$X, col=gray((0:100)/100), axes=FALSE, main="lbd=1")
image(fill3$X, col=gray((0:100)/100), axes=FALSE, main="lbd=10")
par(opar)

## End(Not run)
```

**fill.USVT**

*Matrix Completion by Universal Singular Value Thresholding*

**Description**

`fill.USVT` is a matrix estimation method suitable for low-rank structure. In the context of our package, we provide this method under the matrix completion problem category. It aims at exploiting the idea of thresholding the singular values to minimize the mean-squared error, defined as

\[
\text{MSE}(\hat{A}) := E \left\{ \frac{1}{np} \sum_{i=1}^{n} \sum_{j=1}^{p} (\hat{a}_{ij} - a_{ij})^2 \right\}
\]

where \( A \) is an \((n \times p)\) matrix with some missing values and \( \hat{A} \) is an estimate.
Usage

fill.USVT(A, eta = 0.01)

Arguments

A  
an \((n \times p)\) partially observed matrix.

eta  
control for thresholding \(\in (0, 1)\).

Value

a named list containing

\(X\) an \((n \times p)\) estimated matrix after completion, which is \(\hat{A}\) in the equation above.

References


Examples

```r
## Not run:
## load image data of 'lena128'
data(lena128)

## transform 5% of entries into missing
set.seed(5)
A <- aux.rndmissing(lena128, x=0.05)

## apply the method with 3 different control 'eta'
fill1 <- fill.USVT(A, eta=0.01)
fill2 <- fill.USVT(A, eta=0.5)
fill3 <- fill.USVT(A, eta=0.99)

## visualize only the last ones from each run
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(A, col=gray((0:100)/100), axes=FALSE, main="5% missing")
image(fill1$X, col=gray((0:100)/100), axes=FALSE, main="eta=0.01")
image(fill2$X, col=gray((0:100)/100), axes=FALSE, main="eta=0.5")
image(fill3$X, col=gray((0:100)/100), axes=FALSE, main="eta=0.99")
par(opar)

## End(Not run)
```
lena128

Description

Lena is probably one of the most well-known example in image processing and computer vision.

Usage

data(lena128)

Format

matrix of size \((128 \times 128)\)

Source

USC SIPI Image Database.

References


Examples

data(lena128)
image(lena128, col=gray((0:100)/100), axes=FALSE, main="lena128")

lena256

Description

Lena is probably one of the most well-known example in image processing and computer vision.

Usage

data(lena256)

Format

matrix of size \((256 \times 256)\)

Source

USC SIPI Image Database.
References


Examples

```r
data(lena256)
image(lena256, col=gray((0:100)/100), axes=FALSE, main="lena256")
```

---

**lena64**

*lena image at size of* $64 \times 64$

Description

*Lena* is probably one of the most well-known example in image processing and computer vision.

Usage

```r
data(lena64)
```

Format

matrix of size $(64 \times 64)$

Source

USC SIPI Image Database.

References


Examples

```r
data(lena64)
image(lena64, col=gray((0:100)/100), axes=FALSE, main="lena64")
```
Index

* datasets
  lena128, 15
  lena256, 15
  lena64, 16

aux.rndmissing, 2

fill.HardImpute, 3, 10
fill.KNNimpute, 5, 11
fill.nuclear, 6, 12, 13
fill.OptSpace, 7
fill.simple, 8
fill.SoftImpute, 4, 9
fill.SVDimpute, 5, 11
fill.SVT, 12
fill.USVT, 13
filling-package, 2

lena128, 15
lena256, 15
lena64, 16

OptSpace, 7

psolve, 6