Package 'bigstatsr'

November 30, 2017

Encoding UTF-8

Type Package

Title Statistical Tools for Filebacked Big Matrices

Version 0.2.3

Date 2017-11-17

Description Easy-to-use, efficient, flexible and scalable statistical tools.
Package bigstatsr provides and uses Filebacked Big Matrices via memory-mapping.
It provides for instance matrix operations, Principal Component Analysis,
sparse linear supervised models, utility functions and more.
Preprint: <doi:10.1101/190926>.

License GPL-3

LazyData TRUE

ByteCompile TRUE

Depends R (>= 3.3.2)

Imports cowplot, doParallel, foreach, ggplot2, glue, graphics,
magrittr, Matrix, methods, parallel, Rcpp, RSpectra, stats

LinkingTo BH, Rcpp, RcppArmadillo

Suggests spelling, biglasso, bigmemory, covr, glmnet, grid, LiblineaR,
ModelMetrics, sparseSVM, testthat, viridis

RoxygenNote 6.0.1

URL https://privefl.github.io/bigstatsr

BugReports https://github.com/privefl/bigstatsr/issues

Collate 'AUC.R' 'CMSA.R' 'FBM-attach.R' 'crochet.R' 'FBM.R'
'FBM-code256.R' 'FBM-copy.R' 'RcppExports.R' 'SVD.R'
'apply-parallelize.R' 'biglasso.R' 'bigstatsr.R' 'colstats.R'
'counts.R' 'crossprodSelf.R' 'plot.R' 'predict.R' 'randomSVD.R'
'read.R' 'scaling.R' 'sparseSVM.R' 'tcrossprodSelf.R'
'transpose.R' 'univLinReg.R' 'univLogReg.R' 'utils-assert.R'
'utils-mult.R' 'utils.R' 'zzz.R'

NeedsCompilation yes

Author Florian Privé [aut, cre],
Michael Blum [ths],
Hugues Aschard [ths]

Maintainer Florian Privé <florian.prive.21@gmail.com>

Repository CRAN

Date/Publication 2017-11-30 13:48:52 UTC

R topics documented:

bigstatsr-package . . . 3
asPlotlyText . . . 4
AUC . . . 5
big_apply . . . 6
big_CMSA . . . 8
big_colstats . . . 9
big_copy . . . 10
big_cor . . . 11
big_counts . . . 13
big_cprodMat . . . 14
big_cprodVec . . . 15
big_crossprodSelf . . . 16
big_parallelize . . . 17
big_prodMat . . . 19
big_prodVec . . . 20
big_randomSVD . . . 21
big_read . . . 23
big_scale . . . 24
big_spLinReg . . . 25
big_spLogReg . . . 28
big_spSVM . . . 31
big_SVD . . . 34
big_tcrossprodSelf . . . 36
big_transpose . . . 37
big_univLinReg . . . 38
big_univLogReg . . . 40
block_size . . . 41
Extract . . . 42
FBM-class . . . 43
FBM-methods . . . 44
FBM.code256-class . . . 45
FBM.code256-methods . . . 46
nb_cores . . . 47
pasteLoc . . . 47
plot.big_SVD . . . 48
plot.mhtest . . . 50
predict.big_CMSA . . . 51
predict.big_sp . . . 51

predict.big_SVD 52
predict.mhtest 54
Replace 55

Index **56**

`bigstatsr-package` *bigstatsr: Statistical Tools for Filebacked Big Matrices*

Description

Easy-to-use, efficient, flexible and scalable statistical tools. Package bigstatsr provides and uses Filebacked Big Matrices via memory-mapping. It provides for instance matrix operations, Principal Component Analysis, sparse linear supervised models, utility functions and more. Preprint: <doi:10.1101/190926>.

Arguments

`X`	A FBM.
`X.code`	A FBM.code256.
`y.train`	Vector of responses, corresponding to `ind.train`.
`y01.train`	Vector of responses, corresponding to `ind.train`. **Must be only 0s and 1s.**
`ind.train`	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`block.size`	Maximum number of columns read at once. Default uses block_size.
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
`fun.scaling`	A function that returns a named list of `mean` and `sd` for every column, to scale each of their elements such as followed: $$\frac{X_{i,j} - mean_j}{sd_j}.$$
`covar.train`	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.train`. Default is `NULL` and corresponds to only adding an intercept to each model.
`covar.row`	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.row`. Default is `NULL` and corresponds to only adding an intercept to each model.

Matrix parallelization

Large matrix computations (crossprods) are made block-wise and won't be parallelized in order to not have to reduce the size of these blocks. Instead, you may use Microsoft R Open in order to accelerate these block matrix computations.

Author(s)

Maintainer: Florian Privé <florian.prive.21@gmail.com>

Other contributors:

- Michael Blum <michael.blum@univ-grenoble-alpes.fr> [thesis advisor]
- Hugues Aschard <hugues.aschard@pasteur.fr> [thesis advisor]

See Also

Useful links:

- https://privefl.github.io/bigstatsr
- Report bugs at https://github.com/privefl/bigstatsr/issues

---

asPlotlyText *Plotly text*

Description

Convert a data.frame to plotly text

Usage

```
asPlotlyText(df)
```

Arguments

df	A data.frame

Value

A character vector of the length of df's number of rows.

Examples

```
set.seed(1)

X <- big_attachExtdata()
svd <- big_SVD(X, big_scale(), k = 10)

p <- plot(svd, type = "scores")

pop <- rep(c("POP1", "POP2", "POP3"), c(143, 167, 207))
df <- data.frame(Population = pop, Index = 1:517)

plot(p2 <- p + ggplot2::aes(text = asPlotlyText(df)))
## Not run: plotly::ggplotly(p2, tooltip = "text")
```

AUC *AUC*

Description

Compute the Area Under the ROC Curve (AUC) of a predictor and possibly its 95% confidence interval.

Usage

```
AUC(pred, target, digits = NULL)

AUCBoot(pred, target, nboot = 10000, seed = NA, digits = NULL)
```

Arguments

`pred`	Vector of predictions.
`target`	Vector of true labels (must have exactly two levels, no missing values).
`digits`	See round. Default doesn't use rounding.
`nboot`	Number of bootstrap samples to evaluate the 95% CI. Default is `1e3`.
`seed`	See set.seed. Use it for reproducibility. Default doesn't set any seed.

Details

Other packages provide ways to compute the AUC (see this answer). I chose to compute the AUC through its statistical definition as a probability:

$$P(score(x_{case}) > score(x_{control})).$$

Note that I consider equality between scores as a 50%-probability of one being greater than the other.

Value

The AUC, a probability, and possibly its 2.5% and 97.5% quantiles (95% CI).

References

Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology, 143(1), 29-36. http://dx.doi.org/10.1148/radiology.143.1.7063747.

Tom Fawcett. 2006. An introduction to ROC analysis. Pattern Recogn. Lett. 27, 8 (June 2006), 861-874. http://dx.doi.org/10.1016/j.patrec.2005.10.010.

See Also

wilcox.test

Examples

```
set.seed(1)

AUC(c(0, 0), 0:1) # Equality of scores
AUC(c(0.2, 0.1, 1), c(-1, -1, 1)) # Perfect AUC
x <- rnorm(100)
z <- rnorm(length(x), x, abs(x))
y <- sign(z)
print(AUC(x, y))
print(AUCBoot(x, y))
```

---

big_apply *Split-Apply-Combine*

Description

A Split-Apply-Combine strategy to apply common R functions to a Filebacked Big Matrix.

Usage

```
big_apply(X, a.FUN, a.combine, ind = cols_along(X), ncores = 1,
  block.size = block_size(nrow(X), ncores), ...)
```

Arguments

X	A FBM.
a.FUN	The function to be applied to each subset matrix. It must take a Filebacked Big Matrix as first argument and ind, a vector of indices, which are used to split the data. For example, if you want to apply a function to X[ind.row, ind.col], you may use X[ind.row, ind.col[ind]] in a.FUN.

a.combine	function that is used by foreach to process the tasks results as they generated. This can be specified as either a function or a non-empty character string naming the function. Specifying 'c' is useful for concatenating the results into a vector, for example. The values 'cbind' and 'rbind' can combine vectors into a matrix. The values '+' and '*' can be used to process numeric data. By default, the results are returned in a list.
ind	Initial vector of subsetting indices. Default is the vector of all column indices.
ncores	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
block.size	Maximum number of columns (or rows, depending on how you use ind for subsetting) read at once. Default uses block_size.
...	Extra arguments to be passed to a.FUN.

Details

This function splits indices in parts, then apply a given function to each subset matrix and finally combine the results. If parallelization is used, this function splits indices in parts for parallelization, then split again them on each core, apply a given function to each part and finally combine the results (on each cluster and then from each cluster).

Value

The result of foreach.

See Also

big_parallelize

Examples

```
X <- big_attachExtdata()

# get the means of each column
colMeans_sub <- function(X, ind) colMeans(X[, ind])
str(colmeans <- big_apply(X, a.FUN = colMeans_sub, a.combine = 'c'))

# get the norms of each column
colNorms_sub <- function(X, ind) sqrt(colSums(X[, ind]^2))
str(colnorms <- big_apply(X, colNorms_sub, a.combine = 'c'))

# get the sums of each row
# split along rows: need to change the "complete" `ind` parameter
str(rowsums <- big_apply(X, a.FUN = function(X, ind) rowSums(X[ind, ]),
                         ind = rows_along(X), a.combine = 'c',
                         block.size = 100))
# it is usually preferred to split along columns
# because matrices are stored by column.
str(rowsums2 <- big_apply(X, a.FUN = function(X, ind) rowSums(X[, ind]),
                          a.combine = '+'))

## Every extra parameter to `a.FUN` should be passed to `big_apply`
```

```
# get the crossproduct between X and a matrix A
# note that we don't explicitly pass `ind.col` to `a.FUN`
body(big_cprodMat)

# get the product between X and a matrix B
# here, we must explicitly pass `ind.col` to `a.FUN`
# because the right matrix also needs to be subsetted.
body(big_prodMat)
```

`big_CMSA` *Cross-Model Selection and Averaging.*

Description

1. This function separates the training set in `K` folds (e.g. 10).
2. **In turn**,
   - each fold is considered as an inner validation set and the others (K - 1) folds form an inner training set,
   - the model is trained on the inner training set and the corresponding predictions (scores) for the inner validation set are computed,
   - the vector of scores which maximizes `feval` is determined,
   - the vector of coefficients corresponding to the previous vector of scores is chosen.
3. The `K` resulting vectors of coefficients are then combined into one vector (see the `method` parameter).

Usage

```
big_CMSA(FUN, feval, X, y.train, ind.train = rows_along(X),
  covar.train = NULL, K = 10, method = c("geometric-median", "mean-wise",
  "median-wise"), ncores = 1, ...)
```

Arguments

`FUN`	Function that computes a linear scores for different values along a regularization path. For now, this is relevant to big_spLinReg, big_spLogReg and big_spSVM. The corresponding `<FUN>.predict` function should exists.
`feval`	Customized evaluation function. You should always aim at maximizing this function. If `feval` is like a loss function (which you want to minimize), you should use its opposite instead. Its only two arguments should be `pred` and `target` (e.g. AUC).
`X`	A FBM.
`y.train`	Vector of responses, corresponding to `ind.train`.
`ind.train`	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**

covar.train	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to ind.train. Default is NULL and corresponds to only adding an intercept to each model.
K	Number of folds that are used.
method	Method for combining vectors of coefficients. The default uses the geometric median.
ncores	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
...	Extra parameters to be passed to FUN.

Details

For "mean-wise" or "median-wise" method, tests sometimes crash. I haven't been able to reproduce the bug.

Value

A vector of resulting coefficients (intercept excluded), which corresponds to the training set. You might want to recompute an optimal intercept if needed.

---

big_colstats *Standard univariate statistics*

Description

Standard **univariate statistics** for columns of a Filebacked Big Matrix. For now, the sum and var are implemented (the mean and sd can easily be deduced, see examples).

Usage

```
big_colstats(X, ind.row = rows_along(X), ind.col = cols_along(X))
```

Arguments

X	A FBM.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**

Value

Data.frame of two numeric vectors sum and var with the corresponding column statistics.

See Also

colSums apply

Examples

```
set.seed(1)

X <- big_attachExtdata()

# Check the results
str(test <- big_colstats(X))

# Only with the first 100 rows
ind <- 1:100
str(test2 <- big_colstats(X, ind.row = ind))
plot(test$sum, test2$sum)
abline(lm(test2$sum ~ test$sum), col = "red", lwd = 2)

X.ind <- X[ind, ]
all.equal(test2$sum, colSums(X.ind))
all.equal(test2$var, apply(X.ind, 2, var))

# deduce mean and sd
# note that the are also implemented in big_scale()
means <- test2$sum / length(ind) # if using all rows,
                                 # divide by nrow(X) instead
all.equal(means, colMeans(X.ind))
sds <- sqrt(test2$var)
all.equal(sds, apply(X.ind, 2, sd))
```

---

big_copy *Copy a Filebacked Big Matrix*

Description

Copy a Filebacked Big Matrix with possible subsetting.

Usage

```
big_copy(X, ind.row = rows_along(X), ind.col = cols_along(X),
  type = typeof(X), backingfile = tempfile(), save = FALSE,
  block.size = block_size(length(ind.row)))
```

Arguments

X	Could be any matrix-like object.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
type	Type of the Filebacked Big Matrix (default is double). Either

- "double"
- "integer"
- "unsigned short": can store integer values from 0 to 65535. It has vocation to become the basis for a FBM.code65536 class for accessing strings.
- "raw" or "unsigned char": can store integer values from 0 to 255. It is the basis for the FBM.code256 class for accessing 256 arbitrary different numeric values. It is used in package bigsnpr.

backingfile — Path to the file storing the Big Matrix on disk. An extension ".bk" will be automatically added. Default stores in the temporary directory.

save — Whether to save the result object in an ".rds" file alongside the backingfile. Default is FALSE.

block.size — Maximum number of columns read at once. Default uses block_size.

Value

A copy of the FBM.

Examples

```
X <- FBM(10, 10, init = 1:100)
X[]
X2 <- big_copy(X, ind.row = 1:5)
X2[]

mat <- matrix(101:200, 10)
X3 <- big_copy(mat, type = "double")
X3[]

X.code <- big_attachExtdata()
class(X.code)
X2.code <- big_copy(X.code)
class(X2.code)
all.equal(X.code[], X2.code[])
```

---

big_cor *Correlation*

Description

Compute the correlation matrix of a Filebacked Big Matrix.

Usage

```
big_cor(X, ind.row = rows_along(X), ind.col = cols_along(X),
  block.size = block_size(nrow(X)))
```

Arguments

X	A FBM.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
block.size	Maximum number of columns read at once. Default uses block_size.

Value

A temporary FBM, with the following two attributes:

- a numeric vector `center` of column scaling,
- a numeric vector `scale` of column scaling.

Matrix parallelization

Large matrix computations (crossprods) are made block-wise and won't be parallelized in order to not have to reduce the size of these blocks. Instead, you may use Microsoft R Open in order to accelerate these block matrix computations.

See Also

cor big_crossprodSelf

Examples

```
X <- FBM(13, 17, init = rnorm(221))

# Comparing with cor
K <- big_cor(X)
class(K)
dim(K)
K$backingfile

true <- cor(X[])
all.equal(K[], true)

# Using only half of the data
n <- nrow(X)
ind <- sort(sample(n, n/2))
K2 <- big_cor(X, ind.row = ind)

true2 <- cor(X[ind, ])
all.equal(K2[], true2)
```

big_counts *Counts*

Description

Counts by columns (or rows) the number of each unique element of a FBM.code256.

Usage

```
big_counts(X.code, ind.row = rows_along(X.code),
  ind.col = cols_along(X.code), byrow = FALSE)
```

Arguments

X.code	A FBM.code256.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
byrow	Count by rows rather than columns? Default is FALSE (columns).

Value

A matrix of counts of K x m (or n) elements, where

- K is the number of unique elements of the BM.code,
- n is its number of rows,
- m is its number of columns.

Beware that K is up to 256. So, if you apply this on a Filebacked Big Matrix of one million columns, you will create a matrix of nearly 1GB!.

Examples

```
X <- big_attachExtdata()
class(X)
X[1:5, 1:10]

# Without the "decoding"
X2 <- big_copy(X)
class(X2)
X2[1:5, 1:10]

# Change the code
code <- rep(NA_real_, 256)
code[1:3] <- c(2, 5, 9)
X$code256 <- code
X[1:5, 1:10]
```

```
# by columns
big_counts(X, ind.col = 1:10)

apply(X[, 1:10], 2, table, exclude = NULL)

# by rows
big_counts(X, ind.row = 1:10, byrow = TRUE)
apply(X[1:10, ], 1, table, exclude = NULL)
```

`big_cprodMat` *Cross-product with a matrix*

Description

Cross-product between a Filebacked Big Matrix and a matrix.

Usage

```
big_cprodMat(X, A.row, ind.row = rows_along(X), ind.col = cols_along(X),
  ncores = 1, block.size = block_size(nrow(X), ncores))
```

Arguments

`X`	A FBM.
`A.row`	A matrix with `length(ind.row)` rows.
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
`block.size`	Maximum number of columns read at once. Default uses block_size.

Value

$X^T \cdot A$.

Examples

```
X <- big_attachExtdata()
n <- nrow(X)
m <- ncol(X)
A <- matrix(0, n, 10); A[] <- rnorm(length(A))

test <- big_cprodMat(X, A)
true <- crossprod(X[], A)
all.equal(test, true)
```

```
# subsetting
ind.row <- sample(n, n/2)
ind.col <- sample(m, m/2)

tryCatch(test2 <- big_cprodMat(X, A, ind.row, ind.col),
         error = function(e) print(e))
# returns an error. You need to use the subset of A:
test2 <- big_cprodMat(X, A[ind.row, ], ind.row, ind.col)
true2 <- crossprod(X[ind.row, ind.col], A[ind.row, ])
all.equal(test2, true2)
```

`big_cprodVec` *Cross-product with a vector*

Description

Cross-product between a Filebacked Big Matrix and a vector.

Usage

```
big_cprodVec(X, y.row, ind.row = rows_along(X), ind.col = cols_along(X))
```

Arguments

`X`	A FBM.
`y.row`	A vector of same size as `ind.row`.
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**

Value

$X^T \cdot y$.

Examples

```
X <- big_attachExtdata()
n <- nrow(X)
m <- ncol(X)
y <- rnorm(n)

test <- big_cprodVec(X, y)             # vector
true <- crossprod(X[], y)  # one-column matrix
all.equal(test, as.numeric(true))
```

```
# subsetting
ind.row <- sample(n, n/2)
ind.col <- sample(m, m/2)

tryCatch(test2 <- big_cprodVec(X, y, ind.row, ind.col),
         error = function(e) print(e))
# returns an error. You need to use the subset of y:
test2 <- big_cprodVec(X, y[ind.row], ind.row, ind.col)
true2 <- crossprod(X[ind.row, ind.col], y[ind.row])
all.equal(test2, as.numeric(true2))
```

`big_crossprodSelf` *Crossprod*

Description

Compute $X.row^T X.row$ for a Filebacked Big Matrix `X` after applying a particular scaling to it.

Usage

```
big_crossprodSelf(X, fun.scaling, ind.row = rows_along(X),
  ind.col = cols_along(X), block.size = block_size(nrow(X)))
```

Arguments

`X`	A FBM.
`fun.scaling`	A function that returns a named list of `mean` and `sd` for every column, to scale each of their elements such as followed: $$\frac{X_{i,j} - mean_j}{sd_j}.$$
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`block.size`	Maximum number of columns read at once. Default uses block_size.

Value

A temporary FBM, with the following two attributes:

- a numeric vector `center` of column scaling,
- a numeric vector `scale` of column scaling.

Matrix parallelization

Large matrix computations (crossprods) are made block-wise and won't be parallelized in order to not have to reduce the size of these blocks. Instead, you may use Microsoft R Open in order to accelerate these block matrix computations.

See Also

crossprod

Examples

```
X <- FBM(13, 17, init = rnorm(221))

# Comparing with tcrossprod
big_noscale <- big_scale(center = FALSE)
K <- big_crossprodSelf(X, fun.scaling = big_noscale)
class(K)
dim(K)
K$backingfile

true <- crossprod(X[])
all.equal(K[], true)

# Using only half of the data
n <- nrow(X)
ind <- sort(sample(n, n/2))
K2 <- big_crossprodSelf(X, fun.scaling = big_noscale, ind.row = ind)

true2 <- crossprod(X[ind, ])
all.equal(K2[], true2)
```

---

big_parallelize *Split-parApply-Combine*

Description

A Split-Apply-Combine strategy to parallelize the evaluation of a function.

Usage

```
big_parallelize(X, p.FUN, p.combine, ind = cols_along(X),
  ncores = nb_cores(), ...)
```

Arguments

X	A FBM.
p.FUN	The function to be applied to each subset matrix. It must take a Filebacked Big Matrix as first argument and ind, a vector of indices, which are used to split the data. For example, if you want to apply a function to X[ind.row, ind.col], you may use X[ind.row, ind.col[ind]] in a.FUN.
p.combine	function that is used by foreach to process the tasks results as they generated. This can be specified as either a function or a non-empty character string naming the function. Specifying 'c' is useful for concatenating the results into a vector, for example. The values 'cbind' and 'rbind' can combine vectors into a matrix. The values '+' and '*' can be used to process numeric data. By default, the results are returned in a list.
ind	Initial vector of subsetting indices. Default is the vector of all column indices.
ncores	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
...	Extra arguments to be passed to p.FUN.

Details

This function splits indices in parts, then apply a given function to each part and finally combine the results.

Value

The result of foreach.

See Also

big_apply

Examples

```
## Not run:  # CRAN is super slow when parallelism.
  X <- big_attachExtdata()

  ### Computation on all the matrix
  true <- big_colstats(X)

  big_colstats_sub <- function(X, ind) {
    big_colstats(X, ind.col = ind)
  }
  # 1. the computation is split along all the columns
  # 2. for each part the computation is done, using `big_colstats`
  # 3. the results (data.frames) are combined via `rbind`.
  test <- big_parallelize(X, p.FUN = big_colstats_sub,
                          p.combine = 'rbind', ncores = 2)
  all.equal(test, true)

  ### Computation on a part of the matrix
  n <- nrow(X)
```

```
  m <- ncol(X)
  rows <- sort(sample(n, n/2)) # sort to provide some locality in accesses
  cols <- sort(sample(m, m/2)) # idem

  true2 <- big_colstats(X, ind.row = rows, ind.col = cols)

  big_colstats_sub2 <- function(X, ind, rows, cols) {
    big_colstats(X, ind.row = rows, ind.col = cols[ind])
  }
  # This doesn't work because, by default, the computation is spread
  # along all columns. We must explictly specify the `ind` parameter.
  tryCatch(big_parallelize(X, p.FUN = big_colstats_sub2,
                           p.combine = 'rbind', ncores = 2,
                           rows = rows, cols = cols),
           error = function(e) message(e))

  # This now works, using `ind = seq_along(cols)`.
  test2 <- big_parallelize(X, p.FUN = big_colstats_sub2,
                           p.combine = 'rbind', ncores = 2,
                           ind = seq_along(cols),
                           rows = rows, cols = cols)
  all.equal(test2, true2)


## End(Not run)
```

`big_prodMat` *Product with a matrix*

Description

Product between a Filebacked Big Matrix and a matrix.

Usage

```
big_prodMat(X, A.col, ind.row = rows_along(X), ind.col = cols_along(X),
  ncores = 1, block.size = block_size(nrow(X), ncores))
```

Arguments

`X`	A FBM.
`A.col`	A matrix with `length(ind.col)` rows.
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
`block.size`	Maximum number of columns read at once. Default uses block_size.

Value

$X \cdot A$.

Examples

```
X <- big_attachExtdata()
n <- nrow(X)
m <- ncol(X)
A <- matrix(0, m, 10); A[] <- rnorm(length(A))

test <- big_prodMat(X, A)
true <- X[] %*% A
all.equal(test, true)

# subsetting
ind.row <- sample(n, n/2)
ind.col <- sample(m, m/2)

tryCatch(test2 <- big_prodMat(X, A, ind.row, ind.col),
         error = function(e) print(e))
# returns an error. You need to use the subset of A:
test2 <- big_prodMat(X, A[ind.col, ], ind.row, ind.col)
true2 <- X[ind.row, ind.col] %*% A[ind.col, ]
all.equal(test2, true2)
```

---

big_prodVec *Product with a vector*

Description

Product between a Filebacked Big Matrix and a vector.

Usage

```
big_prodVec(X, y.col, ind.row = rows_along(X), ind.col = cols_along(X))
```

Arguments

X	A FBM.
y.col	A vector of same size as ind.col.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**

Value

$X \cdot y$.

Examples

```
X <- big_attachExtdata()
n <- nrow(X)
m <- ncol(X)
y <- rnorm(m)

test <- big_prodVec(X, y)      # vector
true <- X[] %*% y  # one-column matrix
all.equal(test, as.numeric(true))

# subsetting
ind.row <- sample(n, n/2)
ind.col <- sample(m, m/2)

tryCatch(test2 <- big_prodVec(X, y, ind.row, ind.col),
         error = function(e) print(e))
# returns an error. You need to use the subset of y:
test2 <- big_prodVec(X, y[ind.col], ind.row, ind.col)
true2 <- X[ind.row, ind.col] %*% y[ind.col]
all.equal(test2, as.numeric(true2))
```

---

big_randomSVD *Randomized partial SVD*

Description

An algorithm for partial SVD (or PCA) of a Filebacked Big Matrix based on the algorithm in RSpectra (by Yixuan Qiu and Jiali Mei).
This algorithm is linear in time in all dimensions and is very memory-efficient. Thus, it can be used on very large big.matrices.

Usage

```
big_randomSVD(X, fun.scaling, ind.row = rows_along(X),
  ind.col = cols_along(X), k = 10, tol = 1e-04, verbose = FALSE,
  ncores = 1)
```

Arguments

`X`	A FBM.
`fun.scaling`	A function that returns a named list of `mean` and `sd` for every column, to scale each of their elements such as followed: $$\frac{X_{i,j} - mean_j}{sd_j}.$$

ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
k	Number of singular vectors/values to compute. Default is 10. **This algorithm should be used to compute only a few singular vectors/values.**
tol	Precision parameter of svds. Default is 1e-4.
verbose	Should some progress be printed? Default is FALSE.
ncores	Number of cores used. Default doesn't use parallelism. You may use nb_cores.

Value

A named list (an S3 class "big_SVD") of

- d, the singular values,
- u, the left singular vectors,
- v, the right singular vectors,
- niter, the number of the iteration of the algorithm,
- nops, number of Matrix-Vector multiplications used,
- center, the centering vector,
- scale, the scaling vector.

Note that to obtain the Principal Components, you must use predict on the result. See examples.

Note

The idea of using this Implicitly Restarted Arnoldi Method algorithm comes from G. Abraham, Y. Qiu, and M. Inouye, FlashPCA2: principal component analysis of biobank-scale genotype datasets, bioRxiv: https://doi.org/10.1101/094714.
It proved to be faster than our implementation of the "blanczos" algorithm in Rokhlin, V., Szlam, A., & Tygert, M. (2010). A Randomized Algorithm for Principal Component Analysis. SIAM Journal on Matrix Analysis and Applications, 31(3), 1100–1124. http://dx.doi.org/10.1137/080736417.

See Also

svds

Examples

```
set.seed(1)

X <- big_attachExtdata()
K <- 10

# Using only half of the data for "training"
n <- nrow(X)
```

```
ind <- sort(sample(n, n/2))
test <- big_randomSVD(X, fun.scaling = big_scale(), ind.row = ind, k = K)
str(test)

pca <- prcomp(X[ind, ], center = TRUE, scale. = TRUE)

# same scaling
all.equal(test$center, pca$center)
all.equal(test$scale,  pca$scale)

# use this function to predict scores
class(test)
scores <- predict(test)
# scores and loadings are the same or opposite
plot(scores, pca$x[, 1:K])
plot(test$v, pca$rotation[, 1:K])
plot(test$u)
plot(test, type = "scores")

# projecting on new data
ind2 <- setdiff(rows_along(X), ind)
scores.test2 <- predict(test, X, ind.row = ind2)
scores.test3 <- predict(pca, X[-ind, ])
plot(scores.test2, scores.test3[, 1:K])
```

`big_read` *Read a file*

Description

Read a file as a Filebacked Big Matrix. For a mini-tutorial, please see this vignette.

Usage

```
big_read(file, file.nheader = 0, file.nline = NULL, info.nelem = 0,
  split = " ", read.what = double(), read.transfo = identity,
  BM.type = typeof(read.what), transpose = FALSE, ...)
```

Arguments

`file`	The path to the file to be read.
`file.nheader`	Number of lines to read at the beginning of the file, as a separate header information. Default is `0`. Each line is read as one string. You may need to post-process it with strsplit. See examples.
`file.nline`	Number of total lines of the file. Default is `NULL` and a function computes it. This function doesn't work for compressed files so that you will have to explicitly specify the number of lines of the file.

`info.nelem` Number of elements of extra information to read at the beginning of each line. Default is `0`.

`split` The separator used in the file. Default is a space.

`read.what` What type of elements to scan? Default is `double()`. You can also use `character()` or `integer()`.

`read.transfo` Function that transforms each line you read.

`BM.type` Type of the resulting Filebacked Big Matrix. Default uses the type of `read.what`. This can be useful if you have only small integers or that the `read.transfo` function transforms the type of what is read.

`transpose` Should the resulting Filebacked Big Matrix be transposed? Default is `FALSE`.

`...` Arguments passed on to `FBM`

backingfile Path to the file storing the Big Matrix on disk. An extension ".bk" will be automatically added. Default stores in the temporary directory.

create_bk Create a backingfile (the default) or use an existing one (which should be named by the `backingfile` parameter and have an extension ".bk"). For example, this could be used to convert a filebacked `big.matrix` from package **bigmemory** to a FBM.

save Whether to save the result object in an ".rds" file alongside the backingfile. Default is `FALSE`.

Value

A Filebacked Big Matrix with two attributes `header` and `info`.

`big_scale` *Some scaling functions*

Description

Some scaling functions for a Filebacked Big Matrix to be used as the `fun.scaling` parameter of some functions of this package.

Usage

```
big_scale(center = TRUE, scale = TRUE)
```

Arguments

`center` A logical value: whether to return means or 0s.

`scale` A logical value: whether to return standard deviations or 1s. **You can't use scale without using center.**

Details

One could think about less common scalings, such as for example the "y-aware" scaling which uses the inverse of betas of column-wise linear regression as scaling. See this post for details. It would be easy to implement it using `big_colstats` to get column means and `big_univLinReg` to get betas (and then inverse them).

Value

A new **function** that returns a data.frame of two vectors "center" and "scale" which are of the length of `ind.col`.

See Also

scale

Examples

```
X <- big_attachExtdata()

# No scaling
big_noscale <- big_scale(center = FALSE, scale = FALSE)
class(big_noscale) # big_scale returns a new function
str(big_noscale(X))
big_noscale2 <- big_scale(center = FALSE)
str(big_noscale2(X)) # you can't scale without centering

# Centering
big_center <- big_scale(scale = FALSE)
str(big_center(X))
# + scaling
str(big_scale()(X))
```

`big_spLinReg` *Sparse linear regression*

Description

Fit lasso penalized linear regression path for a Filebacked Big Matrix. Covariates can be added to correct for confounders.

Usage

```
big_spLinReg(X, y.train, ind.train = rows_along(X), covar.train = NULL, ...)
```

Arguments

X	A FBM.
y.train	Vector of responses, corresponding to ind.train.
ind.train	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**
covar.train	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to ind.train. Default is NULL and corresponds to only adding an intercept to each model.
...	Arguments passed on to COPY_biglasso

alpha The elastic-net mixing parameter that controls the relative contribution from the lasso (l1) and the ridge (l2) penalty. The penalty is defined as

$$\alpha||\beta||_1 + (1-\alpha)/2||\beta||_2^2.$$

alpha = 1 is the lasso penalty and alpha in between 0 (1e-6) and 1 is the elastic-net penalty.

lambda.min The smallest value for lambda, as a fraction of lambda.max. Default is .001 if the number of observations is larger than the number of covariates and .01 otherwise.

nlambda The number of lambda values. Default is 100.

lambda.log.scale Whether compute the grid values of lambda on log scale (default) or linear scale.

lambda A user-specified sequence of lambda values. By default, a sequence of values of length nlambda is computed, equally spaced on the log scale.

eps Convergence threshold for inner coordinate descent. The algorithm iterates until the maximum change in the objective after any coefficient update is less than eps times the null deviance. Default value is 1e-7.

max.iter Maximum number of iterations. Default is 1000.

dfmax Upper bound for the number of nonzero coefficients. Default is no upper bound. However, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.

penalty.factor A multiplicative factor for the penalty applied to each coefficient. If supplied, penalty.factor must be a numeric vector of length equal to sum of the number of columns of X and the number of covariables (intercept excluded). The purpose of penalty.factor is to apply differential penalization if some coefficients are thought to be more likely than others to be in the model. Current package doesn't allow unpenalized coefficients. That is penalty.factor cannot be 0.

warn Return warning messages for failures to converge and model saturation? Default is TRUE.

verbose Whether to print out the start, the timing of each lambda iteration and the end. Default is FALSE.

Details

This is a modified version of one function of package biglasso. It adds the possibility to train models with covariables and use many types of FBM (not only double ones). Yet, it only corresponds to screen = "SSR" (Sequential Strong Rules).

Value

A named list with following variables:

intercept	A vector of intercepts, corresponding to each lambda.
beta	The fitted matrix of coefficients, store in sparse matrix representation. The number of rows is equal to the number of coefficients, and the number of columns is equal to nlambda.
iter	A vector of length nlambda containing the number of iterations until convergence at each value of lambda.
lambda	The sequence of regularization parameter values in the path.
penalty	Penalty used. See the input parameter alpha.
family	Either "gaussian" or "binomial" depending on the function used.
alpha	Input parameter.
loss	A vector containing either the residual sum of squares (for linear models) or negative log-likelihood (for logistic models) of the fitted model at each value of lambda.
penalty.factor	Input parameter.
n	The number of observations used in the model fitting. It's equal to length(row.idx).
p	The number of dimensions (including covariables, but not the intercept).
center	The sample mean vector of the variables, i.e., column mean of the sub-matrix of X used for model fitting.
scale	The sample standard deviation of the variables, i.e., column standard deviation of the sub-matrix of X used for model fitting.
y	The response vector used in the model fitting. Depending on row.idx, it could be a subset of the raw input of the response vector y.
col.idx	The indices of features that have 'scale' value greater than 1e-6. Features with 'scale' less than 1e-6 are removed from model fitting.
rejections	The number of features rejected at each value of lambda.

References

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, R. J. (2012), Strong rules for discarding predictors in lasso-type problems. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 74: 245–266. http://dx.doi.org/10.1111/j.1467-9868.2011.01004.x.

Zeng, Y., and Breheny, P. (2016). The biglasso Package: A Memory- and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R. arXiv preprint arXiv:1701.05936. https://arxiv.org/abs/1701.05936.

See Also

glmnet biglasso

Examples

```
set.seed(1)

# simulating some data
N <- 73
M <- 430
X <- FBM(N, M, init = rnorm(N * M, sd = 5), type = "integer")
y <- rnorm(N)
covar <- matrix(rnorm(N * 3), N)

# Error, only handle `double` `big.matrix` objects
X2 <- bigmemory::as.big.matrix(X[], type = "integer", shared = FALSE)
## Not run: biglasso::biglasso(X2, y, family = "gaussian")

# OK here
test2 <- big_spLinReg(X, y)
str(test2)

# How to use covariables?
X2 <- bigmemory::as.big.matrix(cbind(X[], covar), type = "double",
                               shared = FALSE)
test <- biglasso::biglasso(X2, y, family = "gaussian", lambda.min = 0.01,
                           alpha = 0.5, penalty = "enet")
test2 <- big_spLinReg(X, y, covar.train = covar, alpha = 0.5)

# Verification
all.equal(test2$lambda, test$lambda)
all.equal(test2$beta@x, test$beta[-1, ]@x)
all.equal(test2$intercept, test$beta[1, ])
```

`big_spLogReg` *Sparse logistic regression*

Description

Sparse logistic regression

Usage

```
big_spLogReg(X, y01.train, ind.train = rows_along(X), covar.train = NULL,
  ...)
```

Arguments

`X`	A FBM.
`y01.train`	Vector of responses, corresponding to `ind.train`. **Must be only 0s and 1s.**
`ind.train`	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**

`covar.train` Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.train`. Default is `NULL` and corresponds to only adding an intercept to each model.

`...` Arguments passed on to `COPY_biglasso`

alpha The elastic-net mixing parameter that controls the relative contribution from the lasso (l1) and the ridge (l2) penalty. The penalty is defined as

$$\alpha||\beta||_1 + (1-\alpha)/2||\beta||_2^2.$$

`alpha = 1` is the lasso penalty and `alpha` in between `0` (`1e-6`) and `1` is the elastic-net penalty.

lambda.min The smallest value for lambda, as a fraction of lambda.max. Default is `.001` if the number of observations is larger than the number of covariates and `.01` otherwise.

nlambda The number of lambda values. Default is `100`.

lambda.log.scale Whether compute the grid values of lambda on log scale (default) or linear scale.

lambda A user-specified sequence of lambda values. By default, a sequence of values of length `nlambda` is computed, equally spaced on the log scale.

eps Convergence threshold for inner coordinate descent. The algorithm iterates until the maximum change in the objective after any coefficient update is less than `eps` times the null deviance. Default value is `1e-7`.

max.iter Maximum number of iterations. Default is `1000`.

dfmax Upper bound for the number of nonzero coefficients. Default is no upper bound. However, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.

penalty.factor A multiplicative factor for the penalty applied to each coefficient. If supplied, `penalty.factor` must be a numeric vector of length equal to sum of the number of columns of `X` and the number of covariables (intercept excluded). The purpose of `penalty.factor` is to apply differential penalization if some coefficients are thought to be more likely than others to be in the model. Current package doesn't allow unpenalized coefficients. That is `penalty.factor` cannot be 0.

warn Return warning messages for failures to converge and model saturation? Default is `TRUE`.

verbose Whether to print out the start, the timing of each lambda iteration and the end. Default is `FALSE`.

Details

This is a modified version of one function of package biglasso. It adds the possibility to train models with covariables and use many types of `FBM` (not only `double` ones). Yet, it only corresponds to `screen = "SSR"` (Sequential Strong Rules).

Value

A named list with following variables:

intercept	A vector of intercepts, corresponding to each lambda.
beta	The fitted matrix of coefficients, store in sparse matrix representation. The number of rows is equal to the number of coefficients, and the number of columns is equal to nlambda.
iter	A vector of length nlambda containing the number of iterations until convergence at each value of lambda.
lambda	The sequence of regularization parameter values in the path.
penalty	Penalty used. See the input parameter alpha.
family	Either "gaussian" or "binomial" depending on the function used.
alpha	Input parameter.
loss	A vector containing either the residual sum of squares (for linear models) or negative log-likelihood (for logistic models) of the fitted model at each value of lambda.
penalty.factor	Input parameter.
n	The number of observations used in the model fitting. It's equal to length(row.idx).
p	The number of dimensions (including covariables, but not the intercept).
center	The sample mean vector of the variables, i.e., column mean of the sub-matrix of X used for model fitting.
scale	The sample standard deviation of the variables, i.e., column standard deviation of the sub-matrix of X used for model fitting.
y	The response vector used in the model fitting. Depending on row.idx, it could be a subset of the raw input of the response vector y.
col.idx	The indices of features that have 'scale' value greater than 1e-6. Features with 'scale' less than 1e-6 are removed from model fitting.
rejections	The number of features rejected at each value of lambda.

References

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, R. J. (2012), Strong rules for discarding predictors in lasso-type problems. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 74: 245–266. http://dx.doi.org/10.1111/j.1467-9868.2011.01004.x.

Zeng, Y., and Breheny, P. (2016). The biglasso Package: A Memory- and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R. arXiv preprint arXiv:1701.05936. https://arxiv.org/abs/1701.05936.

See Also

glmnet biglasso

Examples

```
set.seed(1)

# simulating some data
N <- 73
M <- 430
X <- FBM(N, M, init = rnorm(N * M, sd = 5), type = "integer")
y <- sample(0:1, size = N, replace = TRUE)
covar <- matrix(rnorm(N * 3), N)

# error, only handle `double` `big.matrix` objects
X2 <- bigmemory::as.big.matrix(X[], type = "integer", shared = FALSE)
## Not run: biglasso::biglasso(X2, y, family = "binomial")

# OK here
test2 <- big_spLogReg(X, y)
str(test2)

# how to use covariables?
X2 <- bigmemory::as.big.matrix(cbind(X[,], covar), type = "double",
                               shared = FALSE)
test <- biglasso::biglasso(X2, y, family = "binomial", lambda.min = 0.01,
                           alpha = 0.5, penalty = "enet")
test2 <- big_spLogReg(X, y, covar.train = covar, alpha = 0.5)
# verification
all.equal(test2$lambda, test$lambda)
all.equal(test2$beta@x, test$beta[-1, ]@x)
all.equal(test2$intercept, test$beta[1, ])
```

`big_spSVM` *Sparse SVM*

Description

Fit solution paths for sparse linear SVM regularized by lasso or elastic-net over a grid of values for the regularization parameter lambda.

Usage

```
big_spSVM(X, y01.train, ind.train = rows_along(X), covar.train = NULL, ...)
```

Arguments

`X`	A FBM.
`y01.train`	Vector of responses, corresponding to `ind.train`. **Must be only 0s and 1s.**
`ind.train`	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**

`covar.train` Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.train`. Default is `NULL` and corresponds to only adding an intercept to each model.

`...` Arguments passed on to `COPY_sparseSVM`

alpha The elastic-net mixing parameter that controls the relative contribution from the lasso and the ridge penalty. It must be a number between `0` and `1`. `alpha=1` is the lasso penalty and `alpha=0` the ridge penalty.

gamma The tuning parameter for huberization smoothing of hinge loss. Default is `0.1`.

nlambda The number of lambda values. Default is `100`.

lambda.min The smallest value for lambda, as a fraction of `lambda.max`, the data derived entry value. Default is `0.01` if the number of observations is larger than the number of variables and `0.05` otherwise.

lambda A user-specified sequence of lambda values. Typical usage is to leave blank and have the program automatically compute a `lambda` sequence based on `nlambda` and `lambda.min`. Specifying `lambda` overrides this. This argument should be used with care and supplied with a decreasing sequence instead of a single value.

screen Screening rule to be applied at each `lambda` that discards variables for speed. Either "ASR" (default), "SR" or "none". "SR" stands for the strong rule, and "ASR" for the adaptive strong rule. Using "ASR" typically requires fewer iterations to converge than "SR", but the computing time are generally close. Note that the option "none" is used mainly for debugging, which may lead to much longer computing time.

max.iter Maximum number of iterations. Default is 1000.

eps Convergence threshold. The algorithms continue until the maximum change in the objective after any coefficient update is less than `eps` times the null deviance. Default is `1E-7`.

dfmax Upper bound for the number of nonzero coefficients. The algorithm exits and returns a partial path if `dfmax` is reached. Useful for very large dimensions.

penalty.factor A numeric vector of length equal to the number of variables. Each component multiplies `lambda` to allow differential penalization. Can be 0 for some variables, in which case the variable is always in the model without penalization. Default is `1` for all variables.

message If set to `TRUE`, `sparseSVM` will inform the user of its progress. Default is `FALSE`.

Details

This is a modified version of one function of package sparseSVM. This algorithm uses semi-newton coordinate descent. There may exist some faster algorithm.

For now, this function is not exported anymore. This is because it doesn't match the results from package sparseSVM anymore, due to a change the author made. I have contacted him multiple times, without success. If you really want to use this function, use `bigstatsr:::big_spSVM`.

Value

A named list containing:

`call`	The call that produced this object.
`intercept`	A vector of intercepts, corresponding to each lambda.
`beta`	The fitted matrix of coefficients. The number of rows is equal to the number of coefficients, and the number of columns is equal to `nlambda`.
`iter`	A vector of length `nlambda` containing the number of iterations until convergence at each value of `lambda`.
`saturated`	A logical flag for whether the number of nonzero coefficients has reached `dfmax`.
`lambda`	The sequence of regularization parameter values in the path.
`alpha`	Input parameter.
`gamma`	Input parameter.
`penalty.factor`	Input parameter.
`levels`	Levels of the output class labels.

See Also

LiblineaR sparseSVM

Examples

```
set.seed(1)

big_spSVM <- bigstatsr:::big_spSVM

# simulating some data
N <- 73
M <- 430
X <- FBM(N, M, init = rnorm(N * M, sd = 5))
y <- sample(0:1, size = N, replace = TRUE)
y.factor <- factor(y, levels = c(1, 0))
covar <- matrix(rnorm(N * 3), N)

# error, only handle standard R matrices
## Not run: test <- sparseSVM::sparseSVM(X, y)

# OK here
test2 <- big_spSVM(X, y)
str(test2)

# how to use covariables?
X2 <- cbind(X[], covar)
test <- sparseSVM::sparseSVM(X2, y.factor, alpha = 0.5)
test2 <- big_spSVM(X, y, covar.train = covar, alpha = 0.5)
# verification
all.equal(test2$lambda, test$lambda)
all.equal(as.matrix(test2$beta), test$weights[-1, ], check.attributes = FALSE)
all.equal(test2$intercept, test$weights[1, ])
```

`big_SVD` *Partial SVD*

Description

An algorithm for partial SVD (or PCA) of a Filebacked Big Matrix through the eigen decomposition of the covariance between variables (primal) or observations (dual). **Use this algorithm only if there is one dimension that is much smaller than the other. Otherwise use big_randomSVD.**

Usage

```
big_SVD(X, fun.scaling, ind.row = rows_along(X), ind.col = cols_along(X),
  k = 10, block.size = block_size(nrow(X)))
```

Arguments

`X`	A FBM.
`fun.scaling`	A function that returns a named list of `mean` and `sd` for every column, to scale each of their elements such as followed: $$\frac{X_{i,j} - mean_j}{sd_j}.$$
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`k`	Number of singular vectors/values to compute. Default is `10`.
`block.size`	Maximum number of columns read at once. Default uses block_size.

Details

To get $X = U \cdot D \cdot V^T$,

- if the number of observations is small, this function computes $K_(2) = X \cdot X^T \approx U \cdot D^2 \cdot U^T$ and then $V = X^T \cdot U \cdot D^{-1}$,
- if the number of variable is small, this function computes $K_(1) = X^T \cdot X \approx V \cdot D^2 \cdot V^T$ and then $U = X \cdot V \cdot D^{-1}$,
- if both dimensions are large, use big_randomSVD instead.

Value

A named list (an S3 class "big_SVD") of

- `d`, the singular values,
- `u`, the left singular vectors,

- v, the right singular vectors,
- center, the centering vector,
- scale, the scaling vector.

Note that to obtain the Principal Components, you must use predict on the result. See examples.

See Also

prcomp

Examples

```
set.seed(1)

X <- big_attachExtdata()
n <- nrow(X)

# Using only half of the data
ind <- sort(sample(n, n/2))

test <- big_SVD(X, fun.scaling = big_scale(), ind.row = ind)
str(test)
plot(test$u)

pca <- prcomp(X[ind, ], center = TRUE, scale. = TRUE)

# same scaling
all.equal(test$center, pca$center)
all.equal(test$scale,  pca$scale)

# scores and loadings are the same or opposite
# except for last eigenvalue which is equal to 0
# due to centering of columns
scores <- test$u %*% diag(test$d)
class(test)
scores2 <- predict(test) # use this function to predict scores
all.equal(scores, scores2)
dim(scores)
dim(pca$x)
tail(pca$sdev)
plot(scores2, pca$x[, 1:ncol(scores2)])
plot(test$v[1:100, ], pca$rotation[1:100, 1:ncol(scores2)])

# projecting on new data
X2 <- sweep(sweep(X[-ind, ], 2, test$center, '-'), 2, test$scale, '/')
scores.test <- X2 %*% test$v
ind2 <- setdiff(rows_along(X), ind)
scores.test2 <- predict(test, X, ind.row = ind2) # use this
all.equal(scores.test, scores.test2)
scores.test3 <- predict(pca, X[-ind, ])
plot(scores.test2, scores.test3[, 1:ncol(scores.test2)])
```

`big_tcrossprodSelf` *Tcrossprod*

Description

Compute $X.row X.row^T$ for a Filebacked Big Matrix `X` after applying a particular scaling to it.

Usage

```
big_tcrossprodSelf(X, fun.scaling, ind.row = rows_along(X),
  ind.col = cols_along(X), block.size = block_size(nrow(X)))
```

Arguments

`X`	A FBM.
`fun.scaling`	A function that returns a named list of `mean` and `sd` for every column, to scale each of their elements such as followed: $$\frac{X_{i,j} - mean_j}{sd_j}.$$
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`block.size`	Maximum number of columns read at once. Default uses block_size.

Value

A temporary FBM, with the following two attributes:

- a numeric vector `center` of column scaling,
- a numeric vector `scale` of column scaling.

Matrix parallelization

Large matrix computations (crossprods) are made block-wise and won't be parallelized in order to not have to reduce the size of these blocks. Instead, you may use Microsoft R Open in order to accelerate these block matrix computations.

See Also

tcrossprod

Examples

```
X <- big_attachExtdata()

# Comparing with tcrossprod
big_noscale <- big_scale(center = FALSE)
K <- big_tcrossprodSelf(X, fun.scaling = big_noscale)
class(K)
dim(K)
K$backingfile

true <- tcrossprod(X[])
all.equal(K[], true)

# Using only half of the data
n <- nrow(X)
ind <- sort(sample(n, n/2))
K2 <- big_tcrossprodSelf(X, fun.scaling = big_noscale, ind.row = ind)

true2 <- tcrossprod(X[ind, ])
all.equal(K2[], true2)
```

---

big_transpose *Transposition*

Description

This function implements a simple cache-oblivious algorithm for the transposition of a Filebacked Big Matrix.

Usage

```
big_transpose(X, ...)
```

Arguments

`X`	A FBM.
`...`	Arguments passed on to `FBM`

backingfile Path to the file storing the Big Matrix on disk. An extension ".bk" will be automatically added. Default stores in the temporary directory.

create_bk Create a backingfile (the default) or use an existing one (which should be named by the `backingfile` parameter and have an extension ".bk"). For example, this could be used to convert a filebacked `big.matrix` from package **bigmemory** to a FBM.

save Whether to save the result object in an ".rds" file alongside the backingfile. Default is `FALSE`.

Value

The new transposed Filebacked Big Matrix (or its descriptor). Its dimensions and type are automatically determined from the input Filebacked Big Matrix.

Examples

```
X <- FBM(10, 5, init = rnorm(50))
X[]
Xt <- big_transpose(X)
identical(t(X[]), Xt[])

X <- big_attachExtdata()
Xt <- big_transpose(X)
identical(t(X[]), Xt[])
```

---

big_univLinReg *Column-wise linear regression*

Description

Slopes of column-wise linear regressions of each column of a Filebacked Big Matrix, with some other associated statistics. Covariates can be added to correct for confounders.

Usage

```
big_univLinReg(X, y.train, ind.train = rows_along(X),
  ind.col = cols_along(X), covar.train = NULL, thr.eigval = 1e-04,
  ncores = 1)
```

Arguments

X	A FBM.
y.train	Vector of responses, corresponding to ind.train.
ind.train	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
covar.train	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to ind.train. Default is NULL and corresponds to only adding an intercept to each model.
thr.eigval	Threshold to remove "insignificant" singular vectors. Default is 1e-4.
ncores	Number of cores used. Default doesn't use parallelism. You may use nb_cores.

Value

A data.frame with 3 elements:

1. the slopes of each regression,
2. the standard errors of each slope,
3. the t-scores associated with each slope. This is also an object of class `mhtest`. See `methods(class = "mhtest")`.

See Also

lm

Examples

```
set.seed(1)

X <- big_attachExtdata()
n <- nrow(X)
y <- rnorm(n)
covar <- matrix(rnorm(n * 3), n)

X1 <- X[, 1] # only first column of the Filebacked Big Matrix

# Without covar
test <- big_univLinReg(X, y)
## New class `mhtest`
class(test)
attr(test, "transfo")
attr(test, "predict")
## plot results
plot(test)
plot(test, type = "Volcano")
## To get p-values associated with the test
test$p.value <- predict(test, log10 = FALSE)
str(test)
summary(lm(y ~ X1))$coefficients[2, ]

# With all data
str(big_univLinReg(X, y, covar = covar))
summary(lm(y ~ X1 + covar))$coefficients[2, ]

# With only half of the data
ind.train <- sort(sample(n, n/2))
str(big_univLinReg(X, y[ind.train],
                   covar.train = covar[ind.train, ],
                   ind.train = ind.train))
summary(lm(y ~ X1 + covar, subset = ind.train))$coefficients[2, ]
```

`big_univLogReg` *Column-wise logistic regression*

Description

Slopes of column-wise logistic regressions of each column of a Filebacked Big Matrix, with some other associated statistics. Covariates can be added to correct for confounders.

Usage

```
big_univLogReg(X, y01.train, ind.train = rows_along(X),
  ind.col = cols_along(X), covar.train = NULL, tol = 1e-08,
  maxiter = 20, ncores = 1)
```

Arguments

`X`	A FBM.
`y01.train`	Vector of responses, corresponding to `ind.train`. **Must be only 0s and 1s.**
`ind.train`	An optional vector of the row indices that are used, for the training part. If not specified, all rows are used. **Don't use negative indices.**
`ind.col`	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
`covar.train`	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.train`. Default is `NULL` and corresponds to only adding an intercept to each model.
`tol`	Relative tolerance to assess convergence of the coefficient. Default is `1e-8`.
`maxiter`	Maximum number of iterations before giving up. Default is `20`. Usually, convergence is reached within 3 or 4 iterations. If there is not convergence, glm is used instead for the corresponding column.
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.

Details

If convergence is not reached by the main algorithm for some columns, the corresponding `niter` element is set to `NA` and a message is given. Then, glm is used instead for the corresponding column. If it can't converge either, all corresponding estimations are set to `NA`.

Value

A data.frame with 4 elements:

1. the slopes of each regression,
2. the standard errors of each slope,
3. the number of iteration for each slope. If is `NA`, this means that the algorithm didn't converge, and glm was used instead.
4. the z-scores associated with each slope. This is also an object of class `mhtest`. See `methods(class = "mhtest")`.

See Also

glm

Examples

```
set.seed(1)

X <- big_attachExtdata()
n <- nrow(X)
y01 <- sample(0:1, size = n, replace = TRUE)
covar <- matrix(rnorm(n * 3), n)

X1 <- X[, 1] # only first column of the Filebacked Big Matrix

# Without covar
test <- big_univLogReg(X, y01)
## new class `mhtest`
class(test)
attr(test, "transfo")
attr(test, "predict")
## plot results
plot(test)
plot(test, type = "Volcano")
## To get p-values associated with the test
test$p.value <- predict(test, log10 = FALSE)
str(test)
summary(glm(y01 ~ X1, family = "binomial"))$coefficients[2, ]

# With all data
str(big_univLogReg(X, y01, covar.train = covar))
summary(glm(y01 ~ X1 + covar, family = "binomial"))$coefficients[2, ]

# With only half of the data
ind.train <- sort(sample(n, n/2))
str(big_univLogReg(X, y01[ind.train],
                   covar.train = covar[ind.train, ],
                   ind.train = ind.train))
summary(glm(y01 ~ X1 + covar, family = "binomial",
            subset = ind.train))$coefficients[2, ]
```

---

block_size *Determine a correct value for the block.size parameter*

Description

It determines the value of `block.size` such that a matrix of doubles of size `n` x `block.size` takes less memory than `getOption("bigstatsr.block.sizeGB")` GigaBytes (default is 1GB).

Usage

```
block_size(n, ncores = 1)
```

Arguments

n	The number of rows.
ncores	The number of cores.

Value

An integer >= 1.

Examples

```
block_size(1e3)
block_size(1e6)
block_size(1e6, 6)
```

Extract *Create an Implementation of [For Custom Matrix-Like Types*

Description

`extract` is a function that converts different index types such as negative integer vectors or logical vectors passed to the `[` function as `i` (e.g. `X[i]`) or `i` and `j` (e.g. `X[i, j]`) into positive integer vectors. The converted indices are provided as the `i` parameter of `extract_vector` or `i` and `j` parameters of `extract_matrix` to facilitate implementing the extraction mechanism for custom matrix-like types.

Usage

```
Extract(extract_vector, extract_matrix)
```

Arguments

extract_vector	A function in the form of `function(x, i)` that takes a subset of `x` based on a single vector of indices `i` and returns a vector.
extract_matrix	A function in the form of `function(x, i, j)` that takes a subset of `x` based on two vectors of indices `i` and `j` and returns a matrix.

Details

The custom type must implement methods for `dim` for this function to work. Implementing methods for `nrow` and `ncol` is not necessary as the default method of those generics calls `dim` internally.

This idea initially comes from package crochet.

Value

A function in the form of `function(x, i, j, ..., drop = TRUE)` that is meant to be used as a method for `[` for a custom type.

`FBM-class` *Class FBM*

Description

A reference class for storing and accessing matrix-like data stored in files on disk. This is very similar to Filebacked Big Matrices provided by the **bigmemory** package. Yet, the implementation is much more lighter.

Wrapper constructor for class `FBM`.

Usage

```
FBM(nrow, ncol, type = c("double", "integer", "unsigned short",
  "unsigned char", "raw"), init = NULL, backingfile = tempfile(),
  create_bk = TRUE, save = FALSE)
```

Arguments

`nrow`	Number of rows.
`ncol`	Number of columns.
`type`	Type of the Filebacked Big Matrix (default is `double`). Either • `"double"` • `"integer"` • `"unsigned short"`: can store integer values from 0 to 65535. It has vocation to become the basis for a `FBM.code65536` class for accessing strings. • `"raw"` or `"unsigned char"`: can store integer values from 0 to 255. It is the basis for the FBM.code256 class for accessing 256 arbitrary different numeric values. It is used in package bigsnpr.
`init`	Either a single value (e.g. `0`) or as many value as the number of elements of the FBM. **Default doesn't initialize the matrix.**
`backingfile`	Path to the file storing the Big Matrix on disk. An extension ".bk" will be automatically added. Default stores in the temporary directory.
`create_bk`	Create a backingfile (the default) or use an existing one (which should be named by the `backingfile` parameter and have an extension ".bk"). For example, this could be used to convert a filebacked `big.matrix` from package **bigmemory** to a FBM.
`save`	Whether to save the result object in an ".rds" file alongside the backingfile. Default is `FALSE`.

Examples

```
X <- FBM(10, 10)
typeof(X)
X[] <- rnorm(length(X))
X[, 1:6]
X[] <- 1:100
X[, 1]
X[1, ]  # not recommended for large matrices
X[, -1]
X[, c(TRUE, FALSE)]
X[cbind(1:10, 1:10)] <- NA_real_
X[]
```

`FBM-methods` *Methods for the FBM class*

Description

Methods for the FBM class

Accessor methods for class `FBM`. You can use positive and negative indices, logical indices (that are recycled) and also a matrix of indices (but only positive ones).

Dimension and type methods for class `FBM`.

Usage

```
## S4 method for signature 'FBM,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'FBM,ANY,ANY,ANY'
x[i, j, ...] <- value

## S4 method for signature 'FBM'
dim(x)

## S4 method for signature 'FBM'
length(x)

## S4 method for signature 'FBM'
typeof(x)
```

Arguments

`x`	A FBM object.
`i`	A vector of indices (or nothing). You can use positive and negative indices, logical indices (that are recycled) and also a matrix of indices (but only positive ones).

j	A vector of indices (or nothing). You can use positive and negative indices, logical indices (that are recycled).
...	Not used. Just to make nargs works.
drop	Whether to delete the dimensions of a matrix which have one dimension equals to 1.
value	The values to replace. Should be of length 1 or of the same length of the subset to replace.

`FBM.code256-class` *Class FBM.code256*

Description

A reference class for storing and accessing up to 256 arbitrary different values using a Filebacked Big Matrix of type `unsigned char`. Compared to a Filebacked Big Matrix, it adds a slot `code` which is used as a lookup table of size 256.

Wrapper constructor for class `FBM.code256`.

Converter from class `FBM` to `FBM.code256`.

Usage

```
FBM.code256(nrow, ncol, code, ...)

add_code256(x, code, save = FALSE)
```

Arguments

nrow	Number of rows.
ncol	Number of columns.
code	A numeric vector (of length 256). You should construct it with `rep(NA_real_, 256)` and then replace the values which are of interest for you.
...	Arguments passed on to `FBM` **init** Either a single value (e.g. `0`) or as many value as the number of elements of the FBM. **Default doesn't initialize the matrix.** **backingfile** Path to the file storing the Big Matrix on disk. An extension ".bk" will be automatically added. Default stores in the temporary directory. **create_bk** Create a backingfile (the default) or use an existing one (which should be named by the `backingfile` parameter and have an extension ".bk"). For example, this could be used to convert a filebacked `big.matrix` from package **bigmemory** to a FBM. **save** Whether to save the result object in an ".rds" file alongside the backingfile. Default is `FALSE`.
x	A FBM.
save	Whether to save the result object in an ".rds" file alongside the backingfile. Default is `FALSE`.

Examples

```
X <- FBM(10, 10, type = "raw")
X[] <- sample(as.raw(0:3), size = length(X), replace = TRUE)
X[]

code <- rep(NA_real_, 256)
code[1:3] <- c(1, 3, 5)

X.code <- add_code256(X, code)
X.code[]

# Or directly
X.code2 <- FBM.code256(10, 10, code, init = sample(as.raw(0:3), 100, TRUE))
X.code2[]

# Copy the FBM with another code
X.code3 <- X.code$copy(code = rnorm(256))
stopifnot(all.equal(X.code$code256, code))
```

FBM.code256-methods *Methods for the FBM.code256 class*

Description

Methods for the FBM.code256 class

Accessor method for class FBM.code256.

Usage

```
## S4 method for signature 'FBM.code256,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	A FBM.code256.
i	A vector of indices (or nothing). You can use positive and negative indices, logical indices (that are recycled) and also a matrix of indices (but only positive ones).
j	A vector of indices (or nothing). You can use positive and negative indices, logical indices (that are recycled).
...	Not used. Just to make nargs works.
drop	Whether to delete the dimensions of a matrix which have one dimension equals to 1.

nb_cores *Recommended number of cores to use*

Description

This is base on the following rule: use only physical cores and if you have only physical cores, leave one core for the OS/UI.

Usage

```
nb_cores(all.tests = FALSE)
```

Arguments

all.tests	Logical: if true apply all known tests.

Value

The recommended number of cores to use.

See Also

parallel::detectCores

Examples

```
# Number of cores in total
parallel::detectCores()
# Number of physical cores
parallel::detectCores(logical = FALSE)
# Recommended number of cores to use
nb_cores()
```

pasteLoc *Get coordinates*

Description

Get coordinates on a plot by mouse-clicking.

Usage

```
pasteLoc(nb, digits = c(3, 3))
```

Arguments

nb	Number of positions.
digits	2 integer indicating the number of decimal places (respectively for x and y coordinates).

Value

A list of coordinates. Note that if you don't put the result in a variable, it returns as the command text for generating the list. This can be useful to get coordinates by mouse-clicking once, but then using the code for convenience and reproducibility.

Examples

```
## Not run:
plot(runif(20, max = 5000))
# note the negative number for the rounding of $y
coord <- pasteLoc(3, digits = c(2, -1))
text(coord, c("a", "b", "c"))

## End(Not run)
```

---

plot.big_SVD *Plot method*

Description

Plot method for class big_SVD.

Usage

```
## S3 method for class 'big_SVD'
plot(x, type = c("screeplot", "scores", "loadings"),
  nval = length(x$d), scores = c(1, 2), loadings = 1, cols = 2,
  coeff = 1, viridis = requireNamespace("viridis", quietly = TRUE), ...)
```

Arguments

x	An object of class big_SVD.
type	Either • "screeplot": plot of decreasing singular values (the default). • "scores": plot of the scores associated with 2 Principal Components. • "loadings": plot of loadings associated with 1 Principal Component.
nval	Number of singular values to plot. Default plots all computed.
scores	Vector of indices of the two PCs to plot. Default plots the first 2 PCs.
loadings	Indices of PC loadings to plot. Default plots the first vector of loadings.

cols	If multiple vector of loadings are to be plotted, this defines the number of columns of the resulting multiplot.
coeff	Relative size of text. Default is 1.
viridis	Whether to use colors of package viridis when plotting multiple loadings. Default is TRUE if the package is installed.
...	Not used.

Value

A ggplot2 object. You can plot it using the print method. You can modify it as you wish by adding layers. You might want to read this chapter to get more familiar with the package **ggplot2**.

See Also

big_SVD, big_randomSVD and asPlotlyText.

Examples

```
set.seed(1)

X <- big_attachExtdata()
svd <- big_SVD(X, big_scale(), k = 10)

# screeplots
plot(svd) # 3 PCs seems "significant"
plot(svd, coeff = 1.5) # larger font for papers

# scores plot
plot(svd, type = "scores") # first 2 PCs
plot(svd, type = "scores", scores = c(1, 3))
## add color (recall that this return a `ggplot2` object)
class(obj <- plot(svd, type = "scores"))
pop <- rep(c("POP1", "POP2", "POP3"), c(143, 167, 207))
library(ggplot2)
print(obj2 <- obj + aes(color = pop) + labs(color = "Population"))
## change the place of the legend
print(obj3 <- obj2 + theme(legend.position = c(0.82, 0.17)))
## change the title and the labels of the axes
obj3 + ggtitle("Yet another title") + xlab("with an other 'x' label")

# loadings
plot(svd, type = "loadings", loadings = 2)
## all loadings
plot(svd, type = "loadings", loadings = 1:10, coeff = 0.5)

# dynamic plots, require the package **plotly**
## Not run: plotly::ggplotly(obj3)
```

`plot.mhtest` *Plot method*

Description

Plot method for class `mhtest`.

Usage

```
## S3 method for class 'mhtest'
plot(x, type = c("Manhattan", "Q-Q", "Volcano"),
  main = paste(type, "Plot"), coeff = 1, ...)
```

Arguments

`x`	An object of class `mhtest`.
`type`	Either • "Manhattan": plot of the negative logarithm (in base 10) of p-values (the default). • "Q-Q": Q-Q plot. • "Volcaco": plot of the negative logarithm of p-values against the estimation of coefficients (e.g. betas in linear regression).
`main`	The title of the plot. Default use the `type`.
`coeff`	Relative size of text. Default is `1`.
`...`	Not used.

Value

A `ggplot2` object. You can plot it using the `print` method. You can modify it as you wish by adding layers. You might want to read this chapter to get more familiar with the package **ggplot2**.

See Also

big_univLinReg, big_univLogReg, plot.big_SVD and asPlotlyText.

Examples

```
set.seed(1)

X <- big_attachExtdata()
y <- rnorm(nrow(X))
test <- big_univLinReg(X, y)

plot(test)
plot(test, type = "Volcano")
plot(test, type = "Q-Q")
```

predict.big_CMSA *Predict method*

Description

Predict method for class `big_CMSA`.

Usage

```
## S3 method for class 'big_CMSA'
predict(object, X, ind.row = rows_along(X),
  covar.row = NULL, ...)
```

Arguments

`object`	Object of class `big_CMSA`.
`X`	A FBM.
`ind.row`	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
`covar.row`	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to `ind.row`. Default is `NULL` and corresponds to only adding an intercept to each model.
`...`	Not used.

Value

A vector of prediction scores, corresponding to `ind.row`.

See Also

big_CMSA

predict.big_sp *Predict method*

Description

Predict method for class `big_sp`.

Usage

```
## S3 method for class 'big_sp'
predict(object, X, ind.row = rows_along(X),
  covar.row = NULL, block.size = block_size(nrow(X)), ...)
```

Arguments

object	Object of class big_sp.
X	A FBM.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
covar.row	Matrix of covariables to be added in each model to correct for confounders (e.g. the scores of PCA), corresponding to ind.row. Default is NULL and corresponds to only adding an intercept to each model.
block.size	Maximum number of columns read at once. Default uses block_size.
...	Not used.

Value

A matrix of scores, with rows corresponding to ind.row and columns corresponding to lambda.

See Also

big_spLinReg, big_spLogReg and big_spSVM.

Examples

```
# simulating some data
N <- 73
M <- 430
X <- FBM(N, M, init = rnorm(N * M, sd = 5))
y <- sample(0:1, size = N, replace = TRUE)
covar <- matrix(rnorm(N * 3), N)

ind.train <- sort(sample(N, N/2))
mod <- big_spLogReg(X, y[ind.train], ind.train = ind.train,
                    covar.train = covar[ind.train, ])
ind.test <- setdiff(rows_along(X), ind.train)
pred <- predict(mod, X = X, ind.row = ind.test,
                covar.row = covar[ind.test, ])
```

---

predict.big_SVD *Scores of PCA*

Description

Get the scores of PCA associated with an svd decomposition (class big_SVD).

Usage

```
## S3 method for class 'big_SVD'
predict(object, X = NULL, ind.row = rows_along(X),
  ind.col = cols_along(X), block.size = block_size(nrow(X)), ...)
```

Arguments

object	A list returned by big_SVD or big_randomSVD.
X	A FBM.
ind.row	An optional vector of the row indices that are used. If not specified, all rows are used. **Don't use negative indices.**
ind.col	An optional vector of the column indices that are used. If not specified, all columns are used. **Don't use negative indices.**
block.size	Maximum number of columns read at once. Default uses block_size.
...	Not used.

Value

A matrix of size $n \times K$ where n is the number of samples corresponding to indices in ind.row and K the number of PCs computed in object. If X is not specified, this just returns the scores of the training set of object.

See Also

predict big_SVD big_randomSVD

Examples

```
set.seed(1)

X <- big_attachExtdata()
n <- nrow(X)

# Using only half of the data
ind <- sort(sample(n, n/2))

test <- big_SVD(X, fun.scaling = big_scale(), ind.row = ind)
str(test)
plot(test$u)

pca <- prcomp(X[ind, ], center = TRUE, scale. = TRUE)

# same scaling
all.equal(test$center, pca$center)
all.equal(test$scale,  pca$scale)

# scores and loadings are the same or opposite
# except for last eigenvalue which is equal to 0
# due to centering of columns
scores <- test$u %*% diag(test$d)
class(test)
scores2 <- predict(test) # use this function to predict scores
all.equal(scores, scores2)
dim(scores)
dim(pca$x)
```

```
tail(pca$sdev)
plot(scores2, pca$x[, 1:ncol(scores2)])
plot(test$v[1:100, ], pca$rotation[1:100, 1:ncol(scores2)])

# projecting on new data
X2 <- sweep(sweep(X[-ind, ], 2, test$center, '-'), 2, test$scale, '/')
scores.test <- X2 %*% test$v
ind2 <- setdiff(rows_along(X), ind)
scores.test2 <- predict(test, X, ind.row = ind2) # use this
all.equal(scores.test, scores.test2)
scores.test3 <- predict(pca, X[-ind, ])
plot(scores.test2, scores.test3[, 1:ncol(scores.test2)])
```

`predict.mhtest` *Predict method*

Description

Predict method for class `mhtest`.

Usage

```
## S3 method for class 'mhtest'
predict(object, scores = object$score, log10 = TRUE, ...)
```

Arguments

`object`	An object of class `mhtest` from you get the probability function with possibly pre-transformation of scores.
`scores`	Raw scores (before transformation) that you want to transform to p-values.
`log10`	Are p-values returned on the `log10` scale? Default is `TRUE`.
`...`	Not used.

Value

Vector of `log10(p-values)` associated with `scores` and `object`.

See Also

big_univLinReg and big_univLogReg.

`Replace` *Create an Implementation of [<- For Custom Matrix-Like Types*

Description

`replace` is a function that converts different index types such as negative integer vectors or logical vectors passed to the `[<-` function as `i` (e.g. `X[i]`) or `i` and `j` (e.g. `X[i, j]`) into positive integer vectors. The converted indices are provided as the `i` parameter of `replace_vector` or `i` and `j` parameters of `replace_matrix` to facilitate implementing the replacement mechanism for custom matrix-like types. Values are recycled to match the replacement length.

Usage

```
Replace(replace_vector, replace_matrix)
```

Arguments

`replace_vector`	A function in the form of `function(x, i, value)` that replaces a vector subset of `x` based on a single vector of indices `i` with the values in `value` and returns `x`, invisibly.
`replace_matrix`	A function in the form of `function(x, i, j, value)` that replaces a matrix subset of `x` based on two vectors of indices `i` and `j` with the values in `value` and returns `x`, invisibly.

Details

The custom type must implement methods for `dim` for this function to work. Implementing methods for `nrow` and `ncol` is not necessary as the default method of those generics calls `dim` internally.

This idea initially comes from package crochet.

Value

A function in the form of `function(x, i, j, ..., value)` that is meant to be used as a method for `[<-` for a custom type.

Index

[,FBM,ANY,ANY,ANY-method *(FBM-methods)*, 44
[,FBM.code256,ANY,ANY,ANY-method *(FBM.code256-methods)*, 46
[<-,FBM,ANY,ANY,ANY-method *(FBM-methods)*, 44

add_code256 *(FBM.code256-class)*, 45
apply, *9*
asPlotlyText, 4, *49, 50*
AUC, 5, *8*
AUCBoot *(AUC)*, 5

big_apply, 6, *18*
big_CMSA, 8, *51*
big_colstats, 9
big_copy, 10
big_cor, 11
big_counts, 13
big_cprodMat, 14
big_cprodVec, 15
big_crossprodSelf, *12*, 16
big_parallelize, *7*, 17
big_prodMat, 19
big_prodVec, 20
big_randomSVD, 21, *34*, *49*, *53*
big_read, 23
big_scale, 24
big_spLinReg, *8*, 25, *52*
big_spLogReg, *8*, 28, *52*
big_spSVM, *8*, 31, *52*
big_SVD, 34, *49*, *53*
big_tcrossprodSelf, 36
big_transpose, 37
big_univLinReg, 38, *50*, *54*
big_univLogReg, 40, *50*, *54*
biglasso, *27*, *30*
bigstatsr *(bigstatsr-package)*, 3
bigstatsr-package, 3

block_size, *3*, *7*, *11, 12*, *14*, *16*, *19*, *34*, *36*, 41, *52, 53*

colSums, *9*
cor, *12*
crossprod, *17*

dim,FBM-method *(FBM-methods)*, 44

Extract, 42

FBM, *3*, *6*, *8, 9*, *11, 12*, *14–16*, *18–21*, *24*, *26*, *28*, *31*, *34*, *36–38*, *40*, *43–45*, *51–53*
FBM *(FBM-class)*, 43
FBM-class, 43
FBM-methods, 44
FBM.code256, *3*, *11*, *13*, *43*, *46*
FBM.code256 *(FBM.code256-class)*, 45
FBM.code256-class, 45
FBM.code256-methods, 46
FBM.code256_RC *(FBM.code256-class)*, 45
FBM_RC *(FBM-class)*, 43
Filebacked Big Matrix, *6*, *18*, *45*
foreach, *7*, *18*

glm, *40, 41*
glmnet, *27*, *30*

length,FBM-method *(FBM-methods)*, 44
LiblineaR, *33*
lm, *39*

nargs, *45, 46*
nb_cores, *3*, *7*, *9*, *14*, *18, 19*, *22*, *38*, *40*, 47

parallel::detectCores, *47*
pasteLoc, 47
plot.big_SVD, 48, *50*
plot.mhtest, 50
prcomp, *35*
predict, *22*, *35*, *53*

predict.big_CMSA, 51
predict.big_sp, 51
predict.big_SVD, 52
predict.mhtest, 54

Replace, 55
round, *5*

scale, *25*
set.seed, *5*
sparseSVM, *33*
strsplit, *23*
svds, *22*

tcrossprod, *36*
typeof,FBM-method *(*FBM-methods*)*, 44

wilcox.test, *6*