Package ‘bigstatsr’

March 12, 2020

Encoding UTF-8

Type Package

Title Statistical Tools for Filebacked Big Matrices

Version 1.2.3

Date 2020-03-11

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
<doi:10.1093/bioinformatics/bty185>.

License GPL-3

Language en-US

LazyData TRUE

ByteCompile TRUE

Depends R (>= 3.3)

Imports bigassertr (>= 0.1.1), bigparallelr (>= 0.2.3), cowplot,
foreach, ggplot2 (>= 3.0), graphics, methods, Rcpp, RSpectra,
stats, tibble, utils

LinkingTo Rcpp, RcppArmadillo, rmio (>= 0.1.3)

Suggests biglasso, bigmemory (>= 4.5.33), bigreadr (>= 0.2), covr,
data.table, glmnet, hexbin, memuse, ModelMetrics, RhpcBLASctl,
spelling (>= 1.2), testthat

RoxygenNote 7.0.2

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

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

Collate 'AUC.R' 'FBM-attach.R' 'crochet.R' 'FBM.R' 'FBM-code256.R'
'FBM-copy.R' 'RcppExports.R' 'SVD.R' 'apply-parallelize.R'
'biglasso.R' 'bigstatsr-package.R' 'colstats.R'
'crossprodSelf.R' 'mult-mat.R' 'mult-vec.R' 'plot.R'
'predict.R' 'randomSVD.R' 'read-write.R' 'scaling.R'

'summary.R' 'tcrossprodSelf.R' 'transpose.R' 'univLinReg.R' 'univLogReg.R' 'utils-assert.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 2020-03-12 13:20:05 UTC

R topics documented:

`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 <doi:10.1093/bioinformatics/bty185>.

Arguments

`X`	An object of class FBM.
`X.code`	An object of class 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: $$\frac{X_{i,j} - mean_j}{sd_j}.$$ Default doesn't use any scaling.
`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. You can use `covar_from_df()` to convert from a data frame.

`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. You can use `covar_from_df()` to convert from a data frame.
`center`	Vector of same length of `ind.col` to subtract from columns of `X`.
`scale`	Vector of same length of `ind.col` to divide from columns of `X`.

Matrix parallelization

Large matrix computations 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

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 used to evaluate the 95% CI. Default is `1e4`.
`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).

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(0, 0, 1)) # Perfect AUC
x <- rnorm(100)
z <- rnorm(length(x), x, abs(x))
y <- as.numeric(z > 0)
print(AUC(x, y))
print(AUCBoot(x, y))

# Partial AUC
pAUC <- function(pred, target, p = 0.1) {
  val.min <- min(target)
  q <- quantile(pred[target == val.min], probs = 1 - p)
  ind <- (target != val.min) | (pred > q)
  bigstatsr::AUC(pred[ind], target[ind]) * p
}
pAUC(x, y)
pAUC(x, y, 0.2)
```

`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 = NULL,
  ind = cols_along(X),
  ncores = 1,
  block.size = block_size(nrow(X), ncores),
  ...
)
```

Arguments

`X`	An object of class 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 to combine the results with `do.call`. This function should accept multiple arguments (`...`). For example, you can use `c`, `cbind`, `rbind`. This package also provides function `plus` to add multiple arguments together. The default is `NULL`, in which case the results are not combined and are returned as a list, each element being the result of a block.
`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).

See Also

big_parallelize bigparallelr::split_parapply

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 = 'plus'))
```

`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), ncores = 1)
```

Arguments

`X`	An object of class 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.**
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.

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 as a Filebacked Big Matrix*

Description

Deep copy of a Filebacked Big Matrix with possible subsetting. This should also work for any matrix-like object.

Usage

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

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"` (double precision – 64 bits)
- `"float"` (single precision – 32 bits)
- `"integer"`
- `"unsigned short"`: can store integer values from 0 to 65535. It has vocation to become the basis for a `FBM.code65536`.
- `"raw"` or `"unsigned char"`: can store integer values from 0 to 255. It is the basis for class FBM.code256 in order to access 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.

`block.size`	Maximum number of columns read at once. Default uses block_size.
`is_read_only`	Whether the FBM is read-only? Default is `FALSE`.

Value

A copy of `X` as a new FBM object.

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")  # as_FBM() would be faster here
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`	An object of class 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 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

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 for class FBM.code256*

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`	An object of class 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 by columns? Default is `FALSE` (count by 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)  # big_counts() is available for class FBM.code256 only
X[1:5, 1:8]

# by columns
big_counts(X, ind.row = 1:5, ind.col = 1:8)

# by rows
big_counts(X, ind.row = 1:5, ind.col = 1:8, byrow = TRUE)
```

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)),
  center = NULL,
  scale = NULL
)

## S4 method for signature 'FBM,matrix'
crossprod(x, y)

## S4 method for signature 'FBM,matrix'
tcrossprod(x, y)

## S4 method for signature 'matrix,FBM'
crossprod(x, y)

## S4 method for signature 'matrix,FBM'
tcrossprod(x, y)
```

Arguments

`X`	An object of class 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.
`center`	Vector of same length of `ind.col` to subtract from columns of `X`.
`scale`	Vector of same length of `ind.col` to divide from columns of `X`.
`x`	A 'double' FBM or a matrix.
`y`	A 'double' FBM or a matrix.

Value

$X^T \cdot A$.

Matrix parallelization

Large matrix computations 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

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)

X2 <- big_copy(X, type = "double")
all.equal(crossprod(X2, A), 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),
  center = NULL,
  scale = NULL
)
```

Arguments

`X`	An object of class 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.**
`center`	Vector of same length of `ind.col` to subtract from columns of `X`.
`scale`	Vector of same length of `ind.col` to divide from columns of `X`.

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 = big_scale(center = FALSE, scale = FALSE),
  ind.row = rows_along(X),
  ind.col = cols_along(X),
  block.size = block_size(nrow(X))
)

## S4 method for signature 'FBM,missing'
crossprod(x, y)
```

Arguments

`X`	An object of class 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: $$\frac{X_{i,j} - mean_j}{sd_j}.$$ Default doesn't use any scaling.
`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.
`x`	A 'double' FBM.
`y`	Missing.

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 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

See Also

crossprod

Examples

```
X <- FBM(13, 17, init = rnorm(221))
true <- crossprod(X[])

# No scaling
K1 <- crossprod(X)
class(K1)
all.equal(K1, true)

K2 <- big_crossprodSelf(X)
class(K2)
K2$backingfile
all.equal(K2[], true)

# big_crossprodSelf() provides some scaling and subsetting
# Example using only half of the data:
n <- nrow(X)
ind <- sort(sample(n, n/2))
K3 <- big_crossprodSelf(X, fun.scaling = big_scale(), ind.row = ind)
true2 <- crossprod(scale(X[ind, ]))
all.equal(K3[], true2)
```

`big_increment` *Increment an FBM*

Description

Increment an FBM

Usage

```
big_increment(X, add, use_lock = FALSE)
```

Arguments

`X`	An `FBM` (of type double) to increment.
`add`	A matrix of same dimensions as `X`. Or a vector of same size.
`use_lock`	Whether to use locks when incrementing. Default is `FALSE`. This is useful when incrementing in parallel.

Value

Returns nothing (NULL, invisibly).

Examples

```
X <- FBM(10, 10, init = 0)
mat <- matrix(rnorm(100), 10, 10)

big_increment(X, mat)
all.equal(X[], mat)

big_increment(X, mat)
all.equal(X[], 2 * mat)
```

---

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 = NULL,
  ind = cols_along(X),
  ncores = nb_cores(),
  ...
)
```

Arguments

X	An object of class 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 to combine the results with do.call. This function should accept multiple arguments (...). For example, you can use c, cbind, rbind. This package also provides function plus to add multiple arguments together. The default is NULL, in which case the results are not combined and are returned as a list, each element being the result of a block.
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

Return a list of `ncores` elements, each element being the result of one of the cores, computed on a block. The elements of this list are then combined with `do.call(p.combine,.)` if `p.combined` is given.

See Also

big_apply bigparallelr::split_parapply

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)),
  center = NULL,
  scale = NULL
)

## S4 method for signature 'FBM,matrix'
x %*% y

## S4 method for signature 'matrix,FBM'
x %*% y
```

Arguments

`X`	An object of class 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.
`center`	Vector of same length of `ind.col` to subtract from columns of `X`.
`scale`	Vector of same length of `ind.col` to divide from columns of `X`.
`x`	A 'double' FBM or a matrix.
`y`	A 'double' FBM or a matrix.

Value

$X \cdot A$.

Matrix parallelization

Large matrix computations 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

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)

X2 <- big_copy(X, type = "double")
all.equal(X2 %*% A, 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),
  center = NULL,
  scale = NULL
)
```

Arguments

`X`	An object of class 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.**
`center`	Vector of same length of `ind.col` to subtract from columns of `X`.
`scale`	Vector of same length of `ind.col` to divide from columns of `X`.

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 = big_scale(center = FALSE, scale = FALSE),
  ind.row = rows_along(X),
  ind.col = cols_along(X),
  k = 10,
  tol = 1e-04,
  verbose = FALSE,
  ncores = 1,
  fun.prod = big_prodVec,
  fun.cprod = big_cprodVec
)
```

Arguments

`X`	An object of class 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: $$\frac{X_{i,j} - mean_j}{sd_j}.$$ Default doesn't use any scaling.
`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.
`fun.prod`	Function that takes 6 arguments (in this order):

- a matrix-like object X,
- a vector x,
- a vector of row indices ind.row of X,
- a vector of column indices ind.col of X,
- a vector of column centers (corresponding to ind.col),
- a vector of column scales (corresponding to ind.col), and compute the product of X (subsetted and scaled) with x.

fun.cprod Same as fun.prod, but for the *transpose* of X.

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. https://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 as FBM*

Description

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

Usage

```
big_read(
  file,
  select,
  filter = NULL,
 type = c("double", "float", "integer", "unsigned short", "unsigned char", "raw"),
  backingfile = drop_ext(file),
  ...
)
```

Arguments

`file`	File to read.
`select`	Indices of columns to read (sorted). The length of `select` will be the number of columns of the resulting FBM.
`filter`	Vector used to subset the rows of each data frame.

`type`	Type of the Filebacked Big Matrix (default is `double`). Either • `"double"` (double precision – 64 bits) • `"float"` (single precision – 32 bits) • `"integer"` • `"unsigned short"`: can store integer values from 0 to 65535. It has vocation to become the basis for a `FBM.code65536`. • `"raw"` or `"unsigned char"`: can store integer values from 0 to 255. It is the basis for class FBM.code256 in order to access 256 arbitrary different numeric values. It is used in package **bigsnpr**.
`backingfile`	Path to the file storing the FBM data on disk. An extension ".bk" will be automatically added. Default uses `file` without its extension.
`...`	Arguments passed on to `bigreadr::big_fread2` `nb_parts` Number of parts in which to split reading (and transforming). Parts are referring to blocks of selected columns. Default uses `part_size` to set a good value. `skip` Number of lines to skip at the beginning of `file`. `progress` Show progress? Default is `FALSE`. `part_size` Size of the parts if `nb_parts` is not supplied. Default is `500 * 1024^2` (500 MB).

Value

A Filebacked Big Matrix of type `type` with `length(select)` columns.

`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),
  ind.col = cols_along(X),
  covar.train = NULL,
  base.train = NULL,
  pf.X = NULL,
  pf.covar = NULL,
  alphas = 1,
  K = 10,
  ind.sets = NULL,
  nlambda = 200,
  nlam.min = 50,
  n.abort = 10,
  dfmax = 50000,
  warn = TRUE,
  ncores = 1,
  ...
)
```

Arguments

`X`	An object of class 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. You can use `covar_from_df()` to convert from a data frame.
`base.train`	Vector of base predictions. Model will be learned starting from these predictions. This can be useful if you want to previously fit a model with large-effect variables that you don't want to penalize.
`pf.X`	A multiplicative factor for the penalty applied to each coefficient. If supplied, `pf.X` must be a numeric vector of the same length as `ind.col`. Default is all `1`. The purpose of `pf.X` is to apply differential penalization if some coefficients are thought to be more likely than others to be in the model. Setting SOME to 0 allows to have unpenalized coefficients.
`pf.covar`	Same as `pf.X`, but for `covar.train`. You might want to set some to 0 as variables with large effects can mask small effects in penalized regression.
`alphas`	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-4`) and `1` is the elastic-net penalty. Default is `1`. **You can pass multiple values, and only one will be used (optimized by grid-search).**
`K`	Number of sets used in the Cross-Model Selection and Averaging (CMSA) procedure. Default is `10`.

`ind.sets`	Integer vectors of values between `1` and `K` specifying which set each index of the training set is in. Default randomly assigns these values but it can be useful to set this vector for reproducibility, or if you want to refine the grid-search over `alphas` using the same sets.
`nlambda`	The number of lambda values. Default is `200`.
`nlam.min`	Minimum number of lambda values to investigate. Default is `50`.
`n.abort`	Number of lambda values for which prediction on the validation set must decrease before stopping. Default is `10`.
`dfmax`	Upper bound for the number of nonzero coefficients. Default is `50e3` because, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.
`warn`	Whether to warn if some models may not have reached a minimum. Default is `TRUE`.
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
`...`	Arguments passed on to `COPY_biglasso_main` `lambda.min.ratio` The smallest value for lambda, **as a fraction of lambda.max**. Default is `.0001` if the number of observations is larger than the number of variables and `.001` otherwise. `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-5`. `max.iter` Maximum number of iterations. Default is `1000`. `return.all` Deprecated. Now always return all models.

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).

Also, to remove the choice of the lambda parameter, we introduce the Cross-Model Selection and Averaging (CMSA) procedure:

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 log-likelihood is determined,
    - the vector of coefficients corresponding to the previous vector of scores is chosen.
3. The `K` resulting vectors of coefficients are then averaged into one final vector of coefficients.

Value

Return an object of class `big_sp_list` (a list of `length(alphas)` x `K`) that has 3 methods `predict`, `summary` and `plot`.

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. https://doi.org/10.1111/j.1467-9868.2011.01004.x.

Zeng, Y., and Breheny, P. (2017). 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.

Privé, F., Aschard, H., and Blum, M. G.B. (2019). Efficient implementation of penalized regression for genetic risk prediction. Genetics, 212: 65-74. https://doi.org/10.1534/genetics.119.302019.

See Also

glmnet biglasso

Examples

```
set.seed(1)

# simulating some data
N <- 230
M <- 730
X <- FBM(N, M, init = rnorm(N * M, sd = 5))
y <- rowSums(X[, 1:10]) + rnorm(N)
covar <- matrix(rnorm(N * 3), N)

ind.train <- sort(sample(nrow(X), 150))
ind.test <- setdiff(rows_along(X), ind.train)

# fitting model for multiple lambdas and alphas
test <- big_spLinReg(X, y[ind.train], ind.train = ind.train,
                     covar.train = covar[ind.train, ],
                     alphas = c(1, 0.5, 0.1, 0.01), warn = FALSE)

# peek at the models
plot(test)
summary(test, sort = TRUE)
summary(test, sort = TRUE)$message

# prediction for other data -> only the best alpha is used
summary(test, best.only = TRUE)
pred <- predict(test, X, ind.row = ind.test, covar.row = covar[ind.test, ])
plot(pred, y[ind.test], pch = 20); abline(0, 1, col = "red")
```

`big_spLogReg` *Sparse logistic regression*

Description

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

Usage

```
big_spLogReg(
  X,
  y01.train,
  ind.train = rows_along(X),
  ind.col = cols_along(X),
  covar.train = NULL,
  base.train = NULL,
  pf.X = NULL,
  pf.covar = NULL,
  alphas = 1,
  K = 10,
  ind.sets = NULL,
  nlambda = 200,
  nlam.min = 50,
  n.abort = 10,
  dfmax = 50000,
  warn = TRUE,
  ncores = 1,
  ...
)
```

Arguments

`X`	An object of class 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. You can use `covar_from_df()` to convert from a data frame.
`base.train`	Vector of base predictions. Model will be learned starting from these predictions. This can be useful if you want to previously fit a model with large-effect variables that you don't want to penalize.

`pf.X`	A multiplicative factor for the penalty applied to each coefficient. If supplied, `pf.X` must be a numeric vector of the same length as `ind.col`. Default is all `1`. The purpose of `pf.X` is to apply differential penalization if some coefficients are thought to be more likely than others to be in the model. Setting SOME to 0 allows to have unpenalized coefficients.
`pf.covar`	Same as `pf.X`, but for `covar.train`. You might want to set some to 0 as variables with large effects can mask small effects in penalized regression.
`alphas`	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-4`) and `1` is the elastic-net penalty. Default is `1`. **You can pass multiple values, and only one will be used (optimized by grid-search).**
`K`	Number of sets used in the Cross-Model Selection and Averaging (CMSA) procedure. Default is `10`.
`ind.sets`	Integer vectors of values between `1` and `K` specifying which set each index of the training set is in. Default randomly assigns these values but it can be useful to set this vector for reproducibility, or if you want to refine the grid-search over `alphas` using the same sets.
`nlambda`	The number of lambda values. Default is `200`.
`nlam.min`	Minimum number of lambda values to investigate. Default is `50`.
`n.abort`	Number of lambda values for which prediction on the validation set must decrease before stopping. Default is `10`.
`dfmax`	Upper bound for the number of nonzero coefficients. Default is `50e3` because, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.
`warn`	Whether to warn if some models may not have reached a minimum. Default is `TRUE`.
`ncores`	Number of cores used. Default doesn't use parallelism. You may use nb_cores.
`...`	Arguments passed on to `COPY_biglasso_main`

`lambda.min.ratio` The smallest value for lambda, **as a fraction of lambda.max**. Default is `.0001` if the number of observations is larger than the number of variables and `.001` otherwise.

`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-5`.

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

`return.all` Deprecated. Now always return all models.

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).

Also, to remove the choice of the lambda parameter, we introduce the Cross-Model Selection and Averaging (CMSA) procedure:

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 log-likelihood is determined,
    - the vector of coefficients corresponding to the previous vector of scores is chosen.
3. The `K` resulting vectors of coefficients are then averaged into one final vector of coefficients.

Value

Return an object of class `big_sp_list` (a list of `length(alphas)` x `K`) that has 3 methods `predict`, `summary` and `plot`.

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. https://doi.org/10.1111/j.1467-9868.2011.01004.x.

Zeng, Y., and Breheny, P. (2017). 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.

Privé, F., Aschard, H., and Blum, M. G.B. (2019). Efficient implementation of penalized regression for genetic risk prediction. Genetics, 212: 65-74. https://doi.org/10.1534/genetics.119.302019.

See Also

glmnet biglasso

Examples

```
set.seed(2)

# simulating some data
N <- 230
M <- 730
X <- FBM(N, M, init = rnorm(N * M, sd = 5))
y01 <- as.numeric((rowSums(X[, 1:10]) + 2 * rnorm(N)) > 0)
covar <- matrix(rnorm(N * 3), N)

ind.train <- sort(sample(nrow(X), 150))
ind.test <- setdiff(rows_along(X), ind.train)
```

```
# fitting model for multiple lambdas and alphas
test <- big_spLogReg(X, y01[ind.train], ind.train = ind.train,
                     covar.train = covar[ind.train, ],
                     alphas = c(1, 0.5, 0.1, 0.01), warn = FALSE)

# peek at the models
plot(test)
summary(test, sort = TRUE)
summary(test, sort = TRUE)$message

# prediction for other data -> only the best alpha is used
summary(test, best.only = TRUE)
pred <- predict(test, X, ind.row = ind.test, covar.row = covar[ind.test, ])
AUC(pred, y01[ind.test])
library(ggplot2)
qplot(pred, fill = as.logical(y01[ind.test]),
      geom = "density", alpha = I(0.4)) +
  labs(fill = "Case?") +
  theme_bigstatsr() +
  theme(legend.position = c(0.52, 0.8))
```

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 = big_scale(center = FALSE, scale = FALSE),
  ind.row = rows_along(X),
  ind.col = cols_along(X),
  k = 10,
  block.size = block_size(nrow(X))
)
```

Arguments

`X`	An object of class 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: $$\frac{X_{i,j} - mean_j}{sd_j}.$$ Default doesn't use any scaling.

`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.

Matrix parallelization

Large matrix computations 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

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 = big_scale(center = FALSE, scale = FALSE),
  ind.row = rows_along(X),
  ind.col = cols_along(X),
  block.size = block_size(nrow(X))
)
```

```
## S4 method for signature 'FBM,missing'
tcrossprod(x, y)
```

Arguments

`X`	An object of class 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: $$\frac{X_{i,j} - mean_j}{sd_j}.$$ Default doesn't use any scaling.
`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.
`x`	A 'double' FBM.
`y`	Missing.

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 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 or OpenBLAS in order to accelerate these block matrix computations. You can also control the number of cores used with `bigparallelr::set_blas_ncores()`.

See Also

tcrossprod

Examples

```
X <- FBM(13, 17, init = rnorm(221))
true <- tcrossprod(X[])

# No scaling
K1 <- tcrossprod(X)
class(K1)
all.equal(K1, true)
```

```
K2 <- big_tcrossprodSelf(X)
class(K2)
K2$backingfile
all.equal(K2[], true)

# big_tcrossprodSelf() provides some scaling and subsetting
# Example using only half of the data:
n <- nrow(X)
ind <- sort(sample(n, n/2))
K3 <- big_tcrossprodSelf(X, fun.scaling = big_scale(), ind.row = ind)
true2 <- tcrossprod(scale(X[ind, ]))
all.equal(K3[], true2)
```

`big_transpose` *Transpose an FBM*

Description

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

Usage

```
big_transpose(X, backingfile = tempfile())
```

Arguments

`X`	An object of class 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.

Value

The new transposed FBM. Dimensions and type are automatically determined from the input FBM.

Examples

```
X <- FBM(6, 5, init = rnorm(30))
X[]
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`	An object of class 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. You can use `covar_from_df()` to convert from a data frame.
`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`	An object of class 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. You can use `covar_from_df()` to convert from a data frame.
`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, ]
```

---

big_write *Write an FBM to a file*

Description

Write a file from a Filebacked Big Matrix (by parts).

Usage

```
big_write(
  X,
  file,
  every_nrow,
  ...,
  ind.row = rows_along(X),
  ind.col = cols_along(X),
  progress = FALSE
)
```

Arguments

`X`	An object of class FBM.
`file`	File to write to.
`every_nrow`	Number of rows to write at once.
`...`	Other arguments to be passed to data.table::fwrite, except `x`, `file`, `append`, `row.names`, `col.names` and `showProgress`.
`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.**
`progress`	Show progress? Default is `FALSE`.

Value

Input parameter `file`, invisibly.

Examples

```
X <- big_attachExtdata()
csv <- big_write(X, tempfile(), every_nrow = 100, progress = interactive())
```

`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)
```

`covar_from_df` *Numeric matrix from data frame*

Description

Transform a data frame to a numeric matrix by one-hot encoding factors. The last factor value is always omitted to prevent having a singular matrix when adding a column of 1s (intercept) in models.

Usage

```
covar_from_df(df)
```

Arguments

`df`	A data frame.

Value

A numeric matrix.

Examples

```
mat <- covar_from_df(iris)
head(mat)
```

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 (see the corresponding vignette).

Convert a matrix (or a data frame) to an FBM.

Usage

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

as_FBM(
  x,
  type = c("double", "float", "integer", "unsigned short", "unsigned char", "raw"),
  backingfile = tempfile(),
  is_read_only = FALSE
)
```

Arguments

`nrow`	Number of rows.
`ncol`	Number of columns.
`type`	Type of the Filebacked Big Matrix (default is `double`). Either • `"double"` (double precision – 64 bits) • `"float"` (single precision – 32 bits) • `"integer"` • `"unsigned short"`: can store integer values from 0 to 65535. It has vocation to become the basis for a `FBM.code65536`. • `"raw"` or `"unsigned char"`: can store integer values from 0 to 255. It is the basis for class FBM.code256 in order to access 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`	Whether to 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 (see the corresponding vignette).
`is_read_only`	Whether the FBM is read-only? Default is `FALSE`.
`x`	A matrix or an data frame (2-dimensional data).

Details

An object of class FBM has many fields:

- $address: address of the external pointer containing the underlying C++ object for read-only mapping, to be used as a XPtr<FBM> in C++ code
- $extptr: (internal) use $address instead
- $address_rw: address of the external pointer containing the underlying C++ object for read/write mapping, to be used as a XPtr<FBM_RW> in C++ code
- $extptr_rw: (internal) use $address_rw instead
- $nrow: number of rows
- $ncol: number of columns
- $type: (internal) use `type_size` or `type_chr` instead
- $type_chr: FBM type as character, e.g. "double"
- $type_size: size of FBM type in bytes (e.g. "double" is 8 and "float" is 4)
- $backingfile or $bk: File with extension 'bk' that stores the numeric data of the FBM
- $rds: 'rds' file (that may not exist) corresponding to the 'bk' file
- $is_saved: whether this object is stored in $rds?
- $is_read_only: whether it is (not) allowed to modify data?

And some methods:

- $save(): Save the FBM object in $rds. Returns the FBM.
- add_columns(<ncol_add>): Add some columns to the FBM by appending the backingfile with some data. Returns the FBM invisibly.
- $bm(): Get this object as a `filebacked.big.matrix` to be used by package bigmemory.
- $bm.desc(): Get this object as a `filebacked.big.matrix` descriptor to be used by package bigmemory.
- $check_write_permissions(): Error if the FBM is read-only.

See Also

big_copy

Examples

```
mat <- matrix(1:4, 2)
X_from_mat <- as_FBM(mat)

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[]  # access as standard R matrix

X <- FBM(150, 5)
X[] <- iris   ## you can replace with a df (but factors -> integers)
X2 <- as_FBM(iris)
identical(X[], X2[])
```

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)

## S4 method for signature 'FBM'
diag(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.

Usage

```
FBM.code256(
  nrow,
  ncol,
  code = rep(NA_real_, 256),
  init = NULL,
  backingfile = tempfile(),
  create_bk = TRUE,
  is_read_only = FALSE
)

add_code256(x, code)
```

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 to you.
`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`	Whether to 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 (see the corresponding vignette).
`is_read_only`	Whether the FBM is read-only? Default is `FALSE`.
`x`	A FBM.

Examples

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

# From an FBM of type 'raw' ('unsigned char')
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[]

# Get a new FBM.code256 object with another code (but same underlying data)
X.code3 <- X.code$copy(code = rnorm(256))
all.equal(X.code$code256, code)
```

`get_beta` *Combine sets of coefficients*

Description

Combine sets of coefficients

Usage

```
get_beta(betas, method = c("geometric-median", "mean-wise", "median-wise"))
```

Arguments

`betas`	Matrix of coefficient vectors to be combined.
`method`	Method for combining vectors of coefficients. The default uses the geometric median.

Value

A vector of resulting coefficients.

`pasteLoc` *Get coordinates on plot*

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 co-ordinates).

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_sp_list *Plot method*

Description

Plot method for class big_sp_list.

Usage

```
## S3 method for class 'big_sp_list'
plot(x, coeff = 1, ...)
```

Arguments

x	An object of class big_sp_list.
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**.

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,
  ncol = NULL,
  coeff = 1,
  viridis = TRUE,
  cols = 2,
  ...
)
```

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 two PCs. If providing more than two, it produces many plots.
`loadings`	Indices of PC loadings to plot. Default plots the first vector of loadings.
`ncol`	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`	Deprecated argument.
`cols`	Deprecated. Use `ncol` instead.
`...`	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))
plot(svd, type = "scores", scores = 1:4, ncol = 2, coeff = 0.7)
## 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 = 1:2, coeff = 0.7, ncol = 1)

# Percentage of variance explained by the PCs
# See https://github.com/privefl/bigstatsr/issues/83

# 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("hist", "Manhattan", "Q-Q", "Volcano"), coeff = 1, ...)
```

Arguments

`x`	An object of class `mhtest`.
`type`	Either. • "hist": histogram of p-values (the default). • "Manhattan": plot of the negative logarithm (in base 10) of p-values. • "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)
`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")
plot(test, type = "Manhattan")
plot(test, type = "Manhattan") + ggplot2::ggtitle(NULL)
```

`predict.big_sp` *Predict method*

Description

Predict method for class `big_sp`.

Usage

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

Arguments

`object`	Object of class `big_sp`.
`X`	An object of class 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.**
`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. You can use `covar_from_df()` to convert from a data frame.
`...`	Not used.

Value

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

See Also

big_spLinReg and big_spLogReg.

`predict.big_sp_list` *Predict method*

Description

Predict method for class `big_sp_list`.

Usage

```
## S3 method for class 'big_sp_list'
predict(
  object,
  X,
  ind.row = rows_along(X),
  ind.col = attr(object, "ind.col"),
  covar.row = NULL,
  proba = (attr(object, "family") == "binomial"),
  base.row = NULL,
  ...
)
```

Arguments

`object`	Object of class `big_sp_list`.
`X`	An object of class 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.**
`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. You can use `covar_from_df()` to convert from a data frame.
`proba`	Whether to return probabilities?
`base.row`	Vector of base predictions, corresponding to `ind.row`.
`...`	Not used.

Value

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

See Also

big_spLinReg and big_spLogReg.

`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`	An object of class 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.

`sub_bk` *Replace extension '.bk'*

Description

Replace extension '.bk'

Usage

```
sub_bk(path, replacement = "", stop_if_not_ext = TRUE)
```

Arguments

`path`	String with extension '.bk'.
`replacement`	Replacement of '.bk'. Default replaces by nothing.
`stop_if_not_ext`	If `replacement != ""`, whether to error if replacement is not an extension (i.e. starting with a dot).

Value

String with extension '.bk' replaced by `replacement`.

Examples

```
path <- "toto.bk"
sub_bk(path)
sub_bk(path, ".rds")
```

`summary.big_sp_list` *Summary method*

Description

Summary method for class `big_sp_list`.

Usage

```
## S3 method for class 'big_sp_list'
summary(object, best.only = FALSE, sort = FALSE, ...)
```

Arguments

`object`	An object of class `big_sp_list`.
`best.only`	Whether to return only one row corresponding to the best model? The best model is the one smallest $validation_loss.
`sort`	Whether to sort by $validation_loss. Default is `FALSE`.
`...`	Not used.

Value

A tibble with, for each $alpha, a mean $validation_loss, a mean vector of coefficients $beta, the corresponding number of non-zero coefficients $nb_var, and the reasons of method completion $message.

`theme_bigstatsr` *Theme ggplot2*

Description

Theme ggplot2 used by this package.

Usage

```
theme_bigstatsr(size.rel = 1)
```

Arguments

`size.rel`	Relative size. Default is `1`.

Examples

```
library(ggplot2)
qplot(y = 1:10)
qplot(y = 1:10) + theme_bw()
qplot(y = 1:10) + theme_bigstatsr()
```

`without_downcast_warning` *Temporarily disable downcast warning*

Description

Temporarily disable downcast warning

Usage

```
without_downcast_warning(expr)
```

Arguments

`expr`	The expression to evaluate without downcast warning.

Value

The result of the evaluated expression.

Examples

```
without_downcast_warning(FBM(10, 10, type = "integer", init = 1.5))
```

Index