Package 'abess'

March 22, 2022

Type Package

Title Fast Best Subset Selection

Version 0.4.5

Date 2022-03-22

Maintainer Jin Zhu <zhuj37@mail2.sysu.edu.cn>

Description Extremely efficient toolkit for solving the best subset selection problem <arXiv:2110.09697>. This package is its R interface. The package implements and generalizes algorithms designed in <doi:10.1073/pnas.2014241117> that exploits a novel sequencing-and-splicing technique to guarantee exact support recovery and globally optimal solution in polynomial times for linear model. It also supports best subset selection for logistic regression, Poisson regression, Cox proportional hazard model, Gamma regression, multiple-response regression, multinomial logistic regression, ordinal regression, (sequential) principal component analysis, and robust principal component analysis. The other valuable features such as the best subset of group selection <arXiv:2104.12576> and sure independence screening <doi:10.1111/j.1467-9868.2008.00674.x> are also provided.

License GPL (>= 3) | file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.1.0)

Imports Rcpp, MASS, methods, Matrix

LinkingTo Rcpp, RcppEigen

RoxygenNote 7.1.2

SystemRequirements C++11

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/abess-team/abess, https://abess-team.github.io/abess/, https://abess.readthedocs.io

BugReports https://github.com/abess-team/abess/issues

NeedsCompilation yes

Author Jin Zhu [aut, cre] (<https://orcid.org/0000-0001-8550-5822>),
Liyuan Hu [aut],
Junhao Huang [aut],
Kangkang Jiang [aut],
Yanhang Zhang [aut],
Zezhi Wang [aut],
Borui Tang [aut],
Shiyun Lin [aut],
Junxian Zhu [aut],
Canhong Wen [aut],
Heping Zhang [aut] (<https://orcid.org/0000-0002-0688-4076>),
Xueqin Wang [aut] (<https://orcid.org/0000-0001-5205-9950>),
spectra contributors [cph] (Spectra implementation)

Repository CRAN

Date/Publication 2022-03-22 10:10:24 UTC

R topics documented:

abess.default *Adaptive best subset selection (for generalized linear model)*

Description

Adaptive best-subset selection for regression, (multi-class) classification, counting-response, censored-response, positive response, multi-response modeling in polynomial times.

Usage

```
## Default S3 method:
abess(
  x,
  y,
  family = c("gaussian", "binomial", "poisson", "cox", "mgaussian", "multinomial",
    "gamma", "ordinal"),
  tune.path = c("sequence", "gsection"),
  tune.type = c("gic", "ebic", "bic", "aic", "cv"),
  weight = NULL,
  normalize = NULL,
  c.max = 2,
  support.size = NULL,
  gs.range = NULL,
  lambda = 0,
  always.include = NULL,
  group.index = NULL,
  init.active.set = NULL,
  splicing.type = 2,
  max.splicing.iter = 20,
  screening.num = NULL,
  important.search = NULL,
  warm.start = TRUE,
  nfolds = 5,
  foldid = NULL,
  cov.update = FALSE,
  newton = c("exact", "approx"),
  newton.thresh = 1e-06,
  max.newton.iter = NULL,
  early.stop = FALSE,
  ic.scale = 1,
  num.threads = 0,
  seed = 1,
  ...
)

## S3 method for class 'formula'
abess(formula, data, subset, na.action, ...)
```

Arguments

x	Input matrix, of dimension $n \times p$; each row is an observation vector and each column is a predictor/feature/variable. Can be in sparse matrix format (inherit from class "dgCMatrix" in package Matrix).
y	The response variable, of n observations. For family = "binomial" should have two levels. For family="poisson", y should be a vector with positive integer. For family = "cox", y should be a Surv object returned by the survival package (recommended) or a two-column matrix with columns named "time" and "status". For family = "mgaussian", y should be a matrix of quantitative responses. For family = "multinomial" or "ordinal", y should be a factor of at least three levels. Note that, for either "binomial", "ordinal" or "multinomial", if y is presented as a numerical vector, it will be coerced into a factor.
family	One of the following models: "gaussian" (continuous response), "binomial" (binary response), "poisson" (non-negative count), "cox" (left-censored response), "mgaussian" (multivariate continuous response), "multinomial" (multi-class response), "ordinal" (multi-class ordinal response), "gamma" (positive continuous response). Depending on the response. Any unambiguous substring can be given.
tune.path	The method to be used to select the optimal support size. For tune.path = "sequence", we solve the best subset selection problem for each size in support.size. For tune.path = "gsection", we solve the best subset selection problem with support size ranged in gs.range, where the specific support size to be considered is determined by golden section.
tune.type	The type of criterion for choosing the support size. Available options are "gic", "ebic", "bic", "aic" and "cv". Default is "gic".
weight	Observation weights. When weight = NULL, we set weight = 1 for each observation as default.
normalize	Options for normalization. normalize = 0 for no normalization. normalize = 1 for subtracting the means of the columns of x and y, and also normalizing the columns of x to have $\sqrt{n}$ norm. normalize = 2 for subtracting the mean of columns of x and scaling the columns of x to have $\sqrt{n}$ norm. normalize = 3 for scaling the columns of x to have $\sqrt{n}$ norm. If normalize = NULL, normalize will be set 1 for "gaussian" and "mgaussian", 3 for "cox". Default is normalize = NULL.
c.max	an integer splicing size. Default is: c.max = 2.
support.size	An integer vector representing the alternative support sizes. Only used for tune.path = "sequence". Default is 0:min(n,round(n/(log(log(n))log(p)))).
gs.range	A integer vector with two elements. The first element is the minimum model size considered by golden-section, the later one is the maximum one. Default is gs.range = c(1,min(n,round(n/(log(log(n))log(p))))).
lambda	A single lambda value for regularized best subset selection. Default is 0.
always.include	An integer vector containing the indexes of variables that should always be included in the model.

`group.index` A vector of integers indicating the which group each variable is in. For variables in the same group, they should be located in adjacent columns of `x` and their corresponding index in `group.index` should be the same. Denote the first group as 1, the second 2, etc. If you do not fit a model with a group structure, please set `group.index = NULL` (the default).

`init.active.set`
A vector of integers indicating the initial active set. Default: `init.active.set = NULL`.

`splicing.type` Optional type for splicing. If `splicing.type = 1`, the number of variables to be spliced is `c.max`, ..., 1; if `splicing.type = 2`, the number of variables to be spliced is `c.max`, `c.max/2`, ..., 1. (Default: `splicing.type = 2`.)

`max.splicing.iter`
The maximum number of performing splicing algorithm. In most of the case, only a few times of splicing iteration can guarantee the convergence. Default is `max.splicing.iter = 20`.

`screening.num` An integer number. Preserve `screening.num` number of predictors with the largest marginal maximum likelihood estimator before running algorithm.

`important.search`
An integer number indicating the number of important variables to be splicing. When `important.search` $\ll$ p variables, it would greatly reduce runtimes. Default: `important.search = 128`.

`warm.start` Whether to use the last solution as a warm start. Default is `warm.start = TRUE`.

`nfolds` The number of folds in cross-validation. Default is `nfolds = 5`.

`foldid` an optional integer vector of values between 1, ..., nfolds identifying what fold each observation is in. The default `foldid = NULL` would generate a random foldid.

`cov.update` A logical value only used for `family = "gaussian"`. If `cov.update = TRUE`, use a covariance-based implementation; otherwise, a naive implementation. The naive method is more computational efficient than covariance-based method when $p >> n$ and `important.search` is much large than its default value. Default: `cov.update = FALSE`.

`newton` A character specify the Newton's method for fitting generalized linear models, it should be either `newton = "exact"` or `newton = "approx"`. If `newton = "exact"`, then the exact hessian is used, while `newton = "approx"` uses diagonal entry of the hessian, and can be faster (especially when `family = "cox"`).

`newton.thresh` a numeric value for controlling positive convergence tolerance. The Newton's iterations converge when $|dev - dev_{old}|/(|dev| + 0.1) <$ `newton.thresh`.

`max.newton.iter`
a integer giving the maximal number of Newton's iteration iterations. Default is `max.newton.iter = 10` if `newton = "exact"`, and `max.newton.iter = 60` if `newton = "approx"`.

`early.stop` A boolean value decide whether early stopping. If `early.stop = TRUE`, algorithm will stop if the last tuning value less than the existing one. Default: `early.stop = FALSE`.

ic.scale	A non-negative value used for multiplying the penalty term in information criterion. Default: ic.scale = 1.
num.threads	An integer decide the number of threads to be concurrently used for cross-validation (i.e., tune.type = "cv"). If num.threads = 0, then all of available cores will be used. Default: num.threads = 0.
seed	Seed to be used to divide the sample into cross-validation folds. Default is seed = 1.
...	further arguments to be passed to or from methods.
formula	an object of class "formula": a symbolic description of the model to be fitted. The details of model specification are given in the "Details" section of "formula".
data	a data frame containing the variables in the formula.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").

Details

Best-subset selection aims to find a small subset of predictors, so that the resulting model is expected to have the most desirable prediction accuracy. Best-subset selection problem under the support size s is

$$\min_{\beta} -2\log L(\beta) \ \text{ s.t. } \ \|\beta\|_0 \leq s,$$

where $L(\beta)$ is arbitrary convex functions. In the GLM case, $\log L(\beta)$ is the log-likelihood function; in the Cox model, $\log L(\beta)$ is the log partial-likelihood function. The best subset selection problem is solved by the splicing algorithm in this package, see Zhu (2020) for details. Under mild conditions, the algorithm exactly solve this problem in polynomial time. This algorithm exploits the idea of sequencing and splicing to reach a stable solution in finite steps when s is fixed. The parameters c.max, splicing.type and max.splicing.iter allow user control the splicing technique flexibly. On the basis of our numerical experiment results, we assign properly parameters to the these parameters as the default such that the precision and runtime are well balanced, we suggest users keep the default values unchanged. Please see this online page for more details about the splicing algorithm.

To find the optimal support size s, we provide various criterion like GIC, AIC, BIC and cross-validation error to determine it. More specifically, the sequence of models implied by support.size are fit by the splicing algorithm. And the solved model with least information criterion or cross-validation error is the optimal model. The sequential searching for the optimal model is somehow time-wasting. A faster strategy is golden section (GS), which only need to specify gs.range. More details about GS is referred to Zhang et al (2021).

It is worthy to note that the parameters newton, max.newton.iter and newton.thresh allows user control the parameter estimation in non-guassian models. The parameter estimation procedure use Newton method or approximated Newton method (only consider the diagonal elements in the Hessian matrix). Again, we suggest to use the default values unchanged because the same reason for the parameter c.max.

abess support some well-known advanced statistical methods to analyze data, including

- sure independent screening: helpful for ultra-high dimensional predictors (i.e., $p \gg n$). Use the parameter `screening.num` to retain the marginally most important predictors. See Fan et al (2008) for more details.
- best subset of group selection: helpful when predictors have group structure. Use the parameter `group.index` to specify the group structure of predictors. See Zhang et al (2021) for more details.
- l_2 regularization best subset selection: helpful when signal-to-ratio is relatively small. Use the parameter `lambda` to control the magnitude of the regularization term.
- nuisance selection: helpful when the prior knowledge of important predictors is available. Use the parameter `always.include` to retain the important predictors.

The arbitrary combination of the four methods are definitely support. Please see online vignettes for more details about the advanced features support by `abess`.

Value

A S3 `abess` class object, which is a `list` with the following components:

`beta`	A p-by-`length(support.size)` matrix of coefficients for univariate family, stored in column format; while a list of `length(support.size)` coefficients matrix (with size p-by-`ncol(y)`) for multivariate family.
`intercept`	An intercept vector of length `length(support.size)` for univariate family; while a list of `length(support.size)` intercept vector (with size `ncol(y)`) for multivariate family.
`dev`	the deviance of length `length(support.size)`.
`tune.value`	A value of tuning criterion of length `length(support.size)`.
`nobs`	The number of sample used for training.
`nvars`	The number of variables used for training.
`family`	Type of the model.
`tune.path`	The path type for tuning parameters.
`support.size`	The actual `support.size` values used. Note that it is not necessary the same as the input if the later have non-integer values or duplicated values.
`edf`	The effective degree of freedom. It is the same as `support.size` when `lambda = 0`.
`best.size`	The best support size selected by the tuning value.
`tune.type`	The criterion type for tuning parameters.
`tune.path`	The strategy for tuning parameters.
`screening.vars`	The character vector specify the feature selected by feature screening. It would be an empty character vector if `screening.num = 0`.
`call`	The original call to `abess`.

Author(s)

Jin Zhu, Junxian Zhu, Canhong Wen, Heping Zhang, Xueqin Wang

References

A polynomial algorithm for best-subset selection problem. Junxian Zhu, Canhong Wen, Jin Zhu, Heping Zhang, Xueqin Wang. Proceedings of the National Academy of Sciences Dec 2020, 117 (52) 33117-33123; doi: 10.1073/pnas.2014241117

Certifiably Polynomial Algorithm for Best Group Subset Selection. Zhang, Yanhang, Junxian Zhu, Jin Zhu, and Xueqin Wang (2021). arXiv preprint arXiv:2104.12576.

abess: A Fast Best Subset Selection Library in Python and R. Jin Zhu, Liyuan Hu, Junhao Huang, Kangkang Jiang, Yanhang Zhang, Shiyun Lin, Junxian Zhu, Xueqin Wang (2021). arXiv preprint arXiv:2110.09697.

Sure independence screening for ultrahigh dimensional feature space. Fan, J. and Lv, J. (2008), Journal of the Royal Statistical Society: Series B (Statistical Methodology), 70: 849-911. doi: 10.1111/j.14679868.2008.00674.x

Targeted Inference Involving High-Dimensional Data Using Nuisance Penalized Regression. Qiang Sun & Heping Zhang (2020). Journal of the American Statistical Association, doi: 10.1080/01621459.2020.1737079

See Also

print.abess, predict.abess, coef.abess, extract.abess, plot.abess, deviance.abess.

Examples

```
library(abess)
n <- 100
p <- 20
support.size <- 3

################ linear model ################
dataset <- generate.data(n, p, support.size)
abess_fit <- abess(dataset[["x"]], dataset[["y"]])
## helpful generic functions:
print(abess_fit)
coef(abess_fit, support.size = 3)
predict(abess_fit,
  newx = dataset[["x"]][1:10, ],
  support.size = c(3, 4)
)
str(extract(abess_fit, 3))
deviance(abess_fit)
plot(abess_fit)
plot(abess_fit, type = "tune")

################ logistic model ################
dataset <- generate.data(n, p, support.size, family = "binomial")
## allow cross-validation to tuning
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "binomial", tune.type = "cv"
)
abess_fit
```

```
################ poisson model ################
dataset <- generate.data(n, p, support.size, family = "poisson")
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "poisson", tune.type = "cv"
)
abess_fit

################ Cox model ################
dataset <- generate.data(n, p, support.size, family = "cox")
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "cox", tune.type = "cv"
)

################ Multivariate gaussian model ################
dataset <- generate.data(n, p, support.size, family = "mgaussian")
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "mgaussian", tune.type = "cv"
)
plot(abess_fit, type = "l2norm")

################ Multinomial model (multi-classification) ################
dataset <- generate.data(n, p, support.size, family = "multinomial")
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "multinomial", tune.type = "cv"
)
predict(abess_fit,
  newx = dataset[["x"]][1:10, ],
  support.size = c(3, 4), type = "response"
)

################ Ordinal regression  ################
dataset <- generate.data(n, p, support.size, family = "ordinal", class.num = 4)
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  family = "ordinal", tune.type = "cv"
)
coef <- coef(abess_fit, support.size = abess_fit[["best.size"]])[[1]]
predict(abess_fit,
  newx = dataset[["x"]][1:10, ],
  support.size = c(3, 4), type = "response"
)

########## Best group subset selection #############
dataset <- generate.data(n, p, support.size)
group_index <- rep(1:10, each = 2)
abess_fit <- abess(dataset[["x"]], dataset[["y"]], group.index = group_index)
str(extract(abess_fit))

################ Golden section searching ################
dataset <- generate.data(n, p, support.size)
abess_fit <- abess(dataset[["x"]], dataset[["y"]], tune.path = "gsection")
abess_fit
```

```
################ Feature screening ################
p <- 1000
dataset <- generate.data(n, p, support.size)
abess_fit <- abess(dataset[["x"]], dataset[["y"]],
  screening.num = 100
)
str(extract(abess_fit))

################ Sparse predictor ################
require(Matrix)
p <- 1000
dataset <- generate.data(n, p, support.size)
dataset[["x"]][abs(dataset[["x"]]) < 1] <- 0
dataset[["x"]] <- Matrix(dataset[["x"]])
abess_fit <- abess(dataset[["x"]], dataset[["y"]])
str(extract(abess_fit))


################  Formula interface  ################
data("trim32")
abess_fit <- abess(y ~ ., data = trim32)
abess_fit
```

---

abesspca *Adaptive best subset selection for principal component analysis*

Description

Adaptive best subset selection for principal component analysis

Usage

```
abesspca(
  x,
  type = c("predictor", "gram"),
  sparse.type = c("fpc", "kpc"),
  cor = FALSE,
  kpc.num = NULL,
  support.size = NULL,
  gs.range = NULL,
  tune.path = c("sequence", "gsection"),
  tune.type = c("gic", "aic", "bic", "ebic", "cv"),
  nfolds = 5,
  foldid = NULL,
  ic.scale = 1,
  c.max = NULL,
  always.include = NULL,
  group.index = NULL,
```

```
  screening.num = NULL,
  splicing.type = 1,
  max.splicing.iter = 20,
  warm.start = TRUE,
  num.threads = 0,
  ...
)
```

Arguments

x	A matrix object. It can be either a predictor matrix where each row is an observation and each column is a predictor or a sample covariance/correlation matrix. If x is a predictor matrix, it can be in sparse matrix format (inherit from class "dgCMatrix" in package Matrix).
type	If type = "predictor", x is considered as the predictor matrix. If type = "gram", x is considered as a sample covariance or correlation matrix.
sparse.type	If sparse.type = "fpc", then best subset selection performs on the first principal component; If sparse.type = "kpc", then best subset selection would be sequentially performed on the first kpc.num number of principal components. If kpc.num is supplied, the default is sparse.type = "kpc"; otherwise, is sparse.type = "fpc".
cor	A logical value. If cor = TRUE, perform PCA on the correlation matrix; otherwise, the covariance matrix. This option is available only if type = "predictor". Default: cor = FALSE.
kpc.num	A integer decide the number of principal components to be sequentially considered.
support.size	It is a flexible input. If it is an integer vector. It represents the support sizes to be considered for each principal component. If it is a list object containing kpc.num number of integer vectors, the i-th principal component consider the support size specified in the i-th element in the list. Only used for tune.path = "sequence". The default is support.size = NULL, and some rules in details section are used to specify support.size.
gs.range	A integer vector with two elements. The first element is the minimum model size considered by golden-section, the later one is the maximum one. Default is gs.range = c(1,min(n,round(n/(log(log(n))log(p))))).
tune.path	The method to be used to select the optimal support size. For tune.path = "sequence", we solve the best subset selection problem for each size in support.size. For tune.path = "gsection", we solve the best subset selection problem with support size ranged in gs.range, where the specific support size to be considered is determined by golden section.
tune.type	The type of criterion for choosing the support size. Available options are "gic", "ebic", "bic", "aic" and "cv". Default is "gic". tune.type = "cv" is available only when type = "predictor".
nfolds	The number of folds in cross-validation. Default is nfolds = 5.
foldid	an optional integer vector of values between 1, ..., nfolds identifying what fold each observation is in. The default foldid = NULL would generate a random foldid.

`ic.scale` A non-negative value used for multiplying the penalty term in information criterion. Default: `ic.scale = 1`.

`c.max` an integer splicing size. The default of `c.max` is the maximum of 2 and `max(support.size) / 2`.

`always.include` An integer vector containing the indexes of variables that should always be included in the model.

`group.index` A vector of integers indicating the which group each variable is in. For variables in the same group, they should be located in adjacent columns of `x` and their corresponding index in `group.index` should be the same. Denote the first group as 1, the second 2, etc. If you do not fit a model with a group structure, please set `group.index = NULL` (the default).

`screening.num` An integer number. Preserve `screening.num` number of predictors with the largest marginal maximum likelihood estimator before running algorithm.

`splicing.type` Optional type for splicing. If `splicing.type = 1`, the number of variables to be spliced is `c.max`, ..., 1; if `splicing.type = 2`, the number of variables to be spliced is `c.max`, `c.max/2`, ..., 1. Default: `splicing.type = 1`.

`max.splicing.iter` The maximum number of performing splicing algorithm. In most of the case, only a few times of splicing iteration can guarantee the convergence. Default is `max.splicing.iter = 20`.

`warm.start` Whether to use the last solution as a warm start. Default is `warm.start = TRUE`.

`num.threads` An integer decide the number of threads to be concurrently used for cross-validation (i.e., `tune.type = "cv"`). If `num.threads = 0`, then all of available cores will be used. Default: `num.threads = 0`.

`...` further arguments to be passed to or from methods.

Details

Adaptive best subset selection for principal component analysis (abessPCA) aim to solve the non-convex optimization problem:

$$-\arg\min_{v} v^\top \Sigma v, s.t. \quad v^\top v = 1, \|v\|_0 \leq s,$$

where s is support size. Here, Σ is covariance matrix, i.e.,

$$\Sigma = \frac{1}{n} X^\top X.$$

A generic splicing technique is implemented to solve this problem. By exploiting the warm-start initialization, the non-convex optimization problem at different support size (specified by `support.size`) can be efficiently solved.

The abessPCA can be conduct sequentially for each component. Please see the multiple principal components Section on the website for more details about this function. For `abesspca` function, the arguments `kpc.num` control the number of components to be consider.

When `sparse.type = "fpc"` but `support.size` is not supplied, it is set as `support.size = 1:min(ncol(x),100)` if `group.index = NULL`; otherwise, `support.size = 1:min(length(unique(group.index)),100)`.
When `sparse.type = "kpc"` but `support.size` is not supplied, then for 20\ it is set as `min(ncol(x),100)` if `group.index = NULL`; otherwise, `min(length(unique(group.index)),100)`.

Value

A S3 abesspca class object, which is a `list` with the following components:

`coef`	A p-by-`length(support.size)` loading matrix of sparse principal components (PC), where each row is a variable and each column is a support size;
`nvars`	The number of variables.
`sparse.type`	The same as input.
`support.size`	The actual support.size values used. Note that it is not necessary the same as the input if the later have non-integer values or duplicated values.
`ev`	A vector with size `length(support.size)`. It records the cumulative sums of explained variance at each support size.
`tune.value`	A value of tuning criterion of length `length(support.size)`.
`kpc.num`	The number of principal component being considered.
`var.pc`	The variance of principal components obtained by performing standard PCA.
`cum.var.pc`	Cumulative sums of `var.pc`.
`var.all`	If `sparse.type = "fpc"`, it is the total standard deviations of all principal components.
`pev`	A vector with the same length as `ev`. It records the percent of explained variance (compared to `var.all`) at each support size.
`pev.pc`	It records the percent of explained variance (compared to `var.pc`) at each support size.
`tune.type`	The criterion type for tuning parameters.
`tune.path`	The strategy for tuning parameters.
`call`	The original call to abess.

It is worthy to note that, if `sparse.type == "kpc"`, the `coef`, `support.size`, `ev`, `tune.value`, `pev` and `pev.pc` in list are `list` objects.

Note

Some parameters not described in the Details Section is explained in the document for `abess` because the meaning of these parameters are very similar.

Author(s)

Jin Zhu, Junxian Zhu, Ruihuang Liu, Junhao Huang, Xueqin Wang

References

A polynomial algorithm for best-subset selection problem. Junxian Zhu, Canhong Wen, Jin Zhu, Heping Zhang, Xueqin Wang. Proceedings of the National Academy of Sciences Dec 2020, 117 (52) 33117-33123; doi: 10.1073/pnas.2014241117

Sparse principal component analysis. Hui Zou, Hastie Trevor, and Tibshirani Robert. Journal of computational and graphical statistics 15.2 (2006): 265-286. doi: 10.1198/106186006X113430

See Also

print.abesspca, coef.abesspca, plot.abesspca.

Examples

```
library(abess)

## predictor matrix input:
head(USArrests)
pca_fit <- abesspca(USArrests)
pca_fit
plot(pca_fit)

## covariance matrix input:
cov_mat <- stats::cov(USArrests) * (nrow(USArrests) - 1) / nrow(USArrests)
pca_fit <- abesspca(cov_mat, type = "gram")
pca_fit

## robust covariance matrix input:
rob_cov <- MASS::cov.rob(USArrests)[["cov"]]
rob_cov <- (rob_cov + t(rob_cov)) / 2
pca_fit <- abesspca(rob_cov, type = "gram")
pca_fit

## K-component principal component analysis
pca_fit <- abesspca(USArrests,
  sparse.type = "kpc",
  support.size = 1:4
)
coef(pca_fit)
plot(pca_fit)
plot(pca_fit, "coef")

## select support size via cross-validation ##
n <- 500
p <- 50
support_size <- 3
dataset <- generate.spc.matrix(n, p, support_size, snr = 20)
spca_fit <- abesspca(dataset[["x"]], tune.type = "cv", nfolds = 5)
plot(spca_fit, type = "tune")
```

---

abessrpca *Adaptive best subset selection for robust principal component analysis*

Description

Decompose a matrix into the summation of low-rank matrix and sparse matrix via the best subset selection approach

Usage

```
abessrpca(
  x,
  rank,
  support.size = NULL,
  tune.path = c("sequence", "gsection"),
  gs.range = NULL,
  tune.type = c("gic", "aic", "bic", "ebic"),
  ic.scale = 1,
  lambda = 0,
  always.include = NULL,
  group.index = NULL,
  c.max = NULL,
  splicing.type = 2,
  max.splicing.iter = 1,
  warm.start = TRUE,
  important.search = NULL,
  max.newton.iter = 1,
  newton.thresh = 0.001,
  num.threads = 0,
  seed = 1,
  ...
)
```

Arguments

x	A matrix object.
rank	A positive integer value specify the rank of the low-rank matrix.
support.size	An integer vector representing the alternative support sizes. Only used for tune.path = "sequence". Strongly suggest its minimum value larger than min(dim(x)).
tune.path	The method to be used to select the optimal support size. For tune.path = "sequence", we solve the best subset selection problem for each size in support.size. For tune.path = "gsection", we solve the best subset selection problem with support size ranged in gs.range, where the specific support size to be considered is determined by golden section.
gs.range	A integer vector with two elements. The first element is the minimum model size considered by golden-section, the later one is the maximum one. Default is gs.range = c(1,min(n,round(n/(log(log(n))log(p))))).
tune.type	The type of criterion for choosing the support size. Available options are "gic", "ebic", "bic" and "aic". Default is "gic".
ic.scale	A non-negative value used for multiplying the penalty term in information criterion. Default: ic.scale = 1.
lambda	A single lambda value for regularized best subset selection. Default is 0.
always.include	An integer vector containing the indexes of variables that should always be included in the model.

`group.index` A vector of integers indicating the which group each variable is in. For variables in the same group, they should be located in adjacent columns of `x` and their corresponding index in `group.index` should be the same. Denote the first group as `1`, the second `2`, etc. If you do not fit a model with a group structure, please set `group.index = NULL` (the default).

`c.max` an integer splicing size. Default is: `c.max = 2`.

`splicing.type` Optional type for splicing. If `splicing.type = 1`, the number of variables to be spliced is `c.max`, ..., `1`; if `splicing.type = 2`, the number of variables to be spliced is `c.max`, `c.max/2`, ..., `1`. (Default: `splicing.type = 2`.)

`max.splicing.iter`
The maximum number of performing splicing algorithm. In most of the case, only a few times of splicing iteration can guarantee the convergence. Default is `max.splicing.iter = 20`.

`warm.start` Whether to use the last solution as a warm start. Default is `warm.start = TRUE`.

`important.search`
An integer number indicating the number of important variables to be splicing. When `important.search` $\ll$ `p` variables, it would greatly reduce runtimes. Default: `important.search = 128`.

`max.newton.iter`
a integer giving the maximal number of Newton's iteration iterations. Default is `max.newton.iter = 10` if `newton = "exact"`, and `max.newton.iter = 60` if `newton = "approx"`.

`newton.thresh` a numeric value for controlling positive convergence tolerance. The Newton's iterations converge when $|dev - dev_{old}|/(|dev| + 0.1) <$ `newton.thresh`.

`num.threads` An integer decide the number of threads to be concurrently used for cross-validation (i.e., `tune.type = "cv"`). If `num.threads = 0`, then all of available cores will be used. Default: `num.threads = 0`.

`seed` Seed to be used to divide the sample into cross-validation folds. Default is `seed = 1`.

`...` further arguments to be passed to or from methods.

Details

Adaptive best subset selection for robust principal component analysis aim to find two latent matrices L and S such that the original matrix X can be appropriately approximated:

$$x = L + S + N,$$

where L is a low-rank matrix, S is a sparse matrix, N is a dense noise matrix. Generic splicing technique can be employed to solve this problem by iteratively improve the quality of the estimation of S.

For a given support set Ω, the optimization problem:

$$\min_S \|x - L - S\|_F^2 \ \text{ s.t. } \ S_{ij} = 0 \text{for} (i,j) \in \Omega^c,$$

still a non-convex optimization problem. We use the hard-impute algorithm proposed in one of the reference to solve this problem. The hard-impute algorithm is an iterative algorithm, people

can set `max.newton.iter` and `newton.thresh` to control the solution precision of the optimization problem. (Here, the name of the two parameters are somehow abused to make the parameters cross functions have an unified name.) According to our experiments, we assign properly parameters to the two parameter as the default such that the precision and runtime are well balanced, we suggest users keep the default values unchanged.

Value

A S3 `abessrpca` class object, which is a `list` with the following components:

`S`	A list with `length(support.size)` elements, each of which is a sparse matrix estimation;
`L`	The low rank matrix estimation.
`nobs`	The number of sample used for training.
`nvars`	The number of variables used for training.
`rank`	The rank of matrix `L`.
`loss`	The loss of objective function.
`tune.value`	A value of tuning criterion of length `length(support.size)`.
`support.size`	The actual support.size values used. Note that it is not necessary the same as the input if the later have non-integer values or duplicated values.
`tune.type`	The criterion type for tuning parameters.
`call`	The original call to `abessrpca`.

Note

Some parameters not described in the Details Section is explained in the document for `abess` because the meaning of these parameters are very similar.

At present, l_2 regularization and group selection are not support, and thus, set `lambda` and `group.index` have no influence on the output. This feature will coming soon.

References

A polynomial algorithm for best-subset selection problem. Junxian Zhu, Canhong Wen, Jin Zhu, Heping Zhang, Xueqin Wang. Proceedings of the National Academy of Sciences Dec 2020, 117 (52) 33117-33123; doi: 10.1073/pnas.2014241117

Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. 2011. Robust principal component analysis? Journal of the ACM. 58, 3, Article 11 (May 2011), 37 pages. doi: 10.1145/1970392.1970395

Mazumder, Rahul, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. The Journal of Machine Learning Research 11 (2010): 2287-2322.

Examples

```
library(abess)
n <- 30
p <- 30
true_S_size <- 60
true_L_rank <- 2
dataset <- generate.matrix(n, p, support.size = true_S_size, rank = true_L_rank)
res <- abessrpca(dataset[["x"]], rank = true_L_rank, support.size = 50:70)
print(res)
coef(res)
plot(res, type = "tune")
plot(res, type = "loss")
plot(res, type = "S")
```

`coef.abess` *Extract Model Coefficients from a fitted "abess" object.*

Description

This function provides estimated coefficients from a fitted "abess" object.

Usage

```
## S3 method for class 'abess'
coef(object, support.size = NULL, sparse = TRUE, ...)
```

Arguments

`object`	An "abess" project.
`support.size`	An integer vector specifies the coefficient fitted at given `support.size`. If `support.size` = `NULL`, then all coefficients would be returned. Default: `support.size` = `NULL`.
`sparse`	A logical value, specifying whether the coefficients should be presented as sparse matrix or not. Default: `sparse` = `TRUE`.
`...`	Other arguments.

Value

A coefficient matrix when fitting an univariate model including gaussian, binomial, poisson, and cox; otherwise, a list containing coefficient matrices. For a coefficient matrix, each row is a variable, and each column is a support size.

See Also

`print.abess`, `predict.abess`, `coef.abess`, `extract.abess`, `plot.abess`, `deviance.abess`.

coef.abesspca *Extract Sparse Loadings from a fitted "abesspca" object.*

Description

This function provides estimated coefficients from a fitted "abesspca" object.

Usage

```
## S3 method for class 'abesspca'
coef(object, support.size = NULL, kpc = NULL, sparse = TRUE, ...)
```

Arguments

object	An "abesspca" project.
support.size	An integer vector specifies the coefficient fitted at given support.size. If support.size = NULL, then all coefficients would be returned. Default: support.size = NULL. This parameter is omitted if sparse.type = "kpc".
kpc	An integer vector specifies the coefficient fitted at given principal component. If kpc = NULL, then all coefficients would be returned. Default: kpc = NULL. This parameter is omitted if sparse.type = "fpc".
sparse	A logical value, specifying whether the coefficients should be presented as sparse matrix or not. Default: sparse = TRUE.
...	Other arguments.

Value

A matrix with length(support.size) columns. Each column corresponds to a sparse loading for the first principal component, where the number of non-zeros entries depends on the support.size.

See Also

print.abesspca, coef.abesspca, plot.abesspca.

coef.abessrpca *Extract sparse component from a fitted "abessrpca" object.*

Description

This function provides estimated coefficients from a fitted "abessrpca" object.

Usage

```
## S3 method for class 'abessrpca'
coef(object, support.size = NULL, sparse = TRUE, ...)
```

Arguments

`object`	An "abessrpca" project.
`support.size`	An integer vector specifies the sparse matrix fitted at given `support.size` to be returned. If `support.size = NULL`, then the sparse matrix with the least tuning value would be returned. Default: `support.size = NULL`.
`sparse`	A logical value, specifying whether the coefficients should be presented as sparse matrix or not. Default: `sparse = TRUE`.
`...`	Other arguments.

Value

A list with `length(support.size)` number of dgCMatrix, each of which is the estimation the sparse component.

`deviance.abess` *Extract the deviance from a fitted "abess" object.*

Description

Similar to other deviance methods, which returns deviance from a fitted "abess" object.

Usage

```
## S3 method for class 'abess'
deviance(object, type = c("standard", "gic", "ebic", "bic", "aic"), ...)
```

Arguments

`object`	A "abess" object.
`type`	The type of deviance. One of the following: `"standard"`, `"gic"`, `"ebic"`, `"bic"` and `"aic"`. Default is `"standard"`.
`...`	additional arguments

Value

A numeric vector.

See Also

`print.abess`, `predict.abess`, `coef.abess`, `extract.abess`, `plot.abess`, `deviance.abess`.

extract *Extract one model from a fitted "*abess*" object.*

Description

Extract the fixed-support-size model's information such as the selected predictors, coefficient estimation, and so on.

Usage

```
extract(object, support.size = NULL, ...)

## S3 method for class 'abess'
extract(object, support.size = NULL, ...)
```

Arguments

object	An "abess" project.
support.size	An integer value specifies the model size fitted at given support.size. If support.size = NULL, then the model with the best tuning value would be returned. Default: support.size = NULL.
...	Other arguments.

Value

A list object including the following components:

beta	A p-by-1 matrix of sparse matrix, stored in column format.
intercept	The fitted intercept value.
support.size	The support.size used in the function.
support.beta	The support.size-length vector of fitted coefficients on the support set.
support.vars	The character vector gives variables in the support set.
tune.value	The tuning value of the model.
dev	The deviance of the model.

See Also

print.abess, predict.abess, coef.abess, extract.abess, plot.abess, deviance.abess.

`generate.data` *Generate simulated data*

Description

Generate simulated data under the generalized linear model and Cox proportional hazard model.

Usage

```
generate.data(
  n,
  p,
  support.size = NULL,
  rho = 0,
 family = c("gaussian", "binomial", "poisson", "cox", "mgaussian", "multinomial",
    "gamma", "ordinal"),
  beta = NULL,
  cortype = 1,
  snr = 10,
  sigma = NULL,
  weibull.shape = 1,
  uniform.max = 1,
  y.dim = 3,
  class.num = 3,
  seed = 1
)
```

Arguments

`n`	The number of observations.
`p`	The number of predictors of interest.
`support.size`	The number of nonzero coefficients in the underlying regression model. Can be omitted if `beta` is supplied.
`rho`	A parameter used to characterize the pairwise correlation in predictors. Default is `0`.
`family`	The distribution of the simulated response. `"gaussian"` for univariate quantitative response, `"binomial"` for binary classification response, `"poisson"` for counting response, `"cox"` for left-censored response, `"mgaussian"` for multivariate quantitative response, `"mgaussian"` for multi-classification response, `"ordinal"` for ordinal response.
`beta`	The coefficient values in the underlying regression model. If it is supplied, `support.size` would be omitted.
`cortype`	The correlation structure. `cortype = 1` denotes the independence structure, where the covariance matrix has (i, j) entry equals $I(i \neq j)$. `cortype = 2` denotes the exponential structure, where the covariance matrix has (i, j) entry equals

	$rho^{\|i-j\|}$. codecortype = 3 denotes the constant structure, where the non-diagonal entries of covariance matrix are rho and diagonal entries are 1.
snr	A numerical value controlling the signal-to-noise ratio (SNR). The SNR is defined as as the variance of $x\beta$ divided by the variance of a gaussian noise: $\frac{Var(x\beta)}{\sigma^2}$. The gaussian noise ϵ is set with mean 0 and variance. The noise is added to the linear predictor $\eta = x\beta$. Default is snr = 10. Note that this arguments's effect is overridden if sigma is supplied with a non-null value.
sigma	The variance of the gaussian noise. Default sigma = NULL implies it is determined by snr.
weibull.shape	The shape parameter of the Weibull distribution. It works only when family = "cox". Default: weibull.shape = 1.
uniform.max	A parameter controlling censored rate. A large value implies a small censored rate; otherwise, a large censored rate. It works only when family = "cox". Default is uniform.max = 1.
y.dim	Response's Dimension. It works only when family = "mgaussian". Default: y.dim = 3.
class.num	The number of class. It works only when family = "multinomial". Default: class.num = 3.
seed	random seed. Default: seed = 1.

Details

For family = "gaussian", the data model is

$$Y = X\beta + \epsilon.$$

The underlying regression coefficient β has uniform distribution [m, 100m] and $m = 5\sqrt{2log(p)/n}$.

For family= "binomial", the data model is

$$Prob(Y = 1) = \exp(X\beta + \epsilon)/(1 + \exp(X\beta + \epsilon)).$$

The underlying regression coefficient β has uniform distribution [2m, 10m] and $m = 5\sqrt{2log(p)/n}$.

For family = "poisson", the data is modeled to have an exponential distribution:

$$Y = Exp(\exp(X\beta + \epsilon)).$$

The underlying regression coefficient β has uniform distribution [2m, 10m] and $m = \sqrt{2log(p)/n}/3$.

For family = "gamma", the data is modeled to have a gamma distribution:

$$Y = Gamma(X\beta + \epsilon + 10, shape),$$

where $shape$ is shape parameter in a gamma distribution. The underlying regression coefficient β has uniform distribution [2m, 100m] and $m = \sqrt{2log(p)/n}$.

For family = "ordinal", the data is modeled to have an ordinal distribution.

For family = "cox", the model for failure time T is

$$T = (-\log(U/\exp(X\beta)))^{1/weibull.shape},$$

where U is a uniform random variable with range [0, 1]. The centering time C is generated from uniform distribution $[0, uniform.max]$, then we define the censor status as $\delta = I(T \leq C)$ and observed time as $R = \min\{T, C\}$. The underlying regression coefficient β has uniform distribution [2m, 10m], where $m = 5\sqrt{2log(p)/n}$.

For `family = "mgaussian"`, the data model is

$$Y = X\beta + E.$$

The non-zero values of regression matrix β are sampled from uniform distribution [m, 100m] and $m = 5\sqrt{2log(p)/n}$.

For `family= "multinomial"`, the data model is

$$Prob(Y = 1) = \exp(X\beta + E)/(1 + \exp(X\beta + E)).$$

The non-zero values of regression coefficient β has uniform distribution [2m, 10m] and $m = 5\sqrt{2log(p)/n}$.

In the above models, $\epsilon \sim N(0, \sigma^2)$ and $E \sim MVN(0, \sigma^2 \times I_{q\times q})$, where σ^2 is determined by the `snr` and q is `y.dim`.

Value

A `list` object comprising:

`x`	Design matrix of predictors.
`y`	Response variable.
`beta`	The coefficients used in the underlying regression model.

Author(s)

Jin Zhu

Examples

```
# Generate simulated data
n <- 200
p <- 20
support.size <- 5
dataset <- generate.data(n, p, support.size)
str(dataset)
```

`generate.matrix` *Generate matrix composed of a sparse matrix and low-rank matrix*

Description

Generate simulated matrix that is the superposition of a low-rank component and a sparse component.

Usage

```
generate.matrix(
  n,
  p,
  rank = NULL,
  support.size = NULL,
  beta = NULL,
  snr = Inf,
  sigma = NULL,
  seed = 1
)
```

Arguments

`n`	The number of observations.
`p`	The number of predictors of interest.
`rank`	The rank of low-rank matrix.
`support.size`	The number of nonzero coefficients in the underlying regression model. Can be omitted if `beta` is supplied.
`beta`	The coefficient values in the underlying regression model. If it is supplied, `support.size` would be omitted.
`snr`	A positive value controlling the signal-to-noise ratio (SNR). A larger SNR implies the identification of sparse matrix is much easier. Default `snr = Inf` enforces no noise exists.
`sigma`	A numerical value supplied the variance of the gaussian noise. Default `sigma = NULL` implies it is determined by `snr`.
`seed`	random seed. Default: `seed = 1`.

Details

The low rank matrix L is generated by $L = UV$, where U is an n-by-$rank$ matrix and V is a $rank$-by-p matrix. Each element in U (or V) are i.i.d. drawn from $N(0, 1/n)$.

The sparse matrix S is an n-by-$rank$ matrix. It is generated by choosing a support set of size `support.size` uniformly at random. The non-zero entries in S are independent Bernoulli (-1, +1) entries.

The noise matrix N is an n-by-$rank$ matrix, the elements in N are i.i.d. gaussian random variable with standard deviation σ.

The SNR is defined as as the variance of vectorized matrix $L + S$ divided by σ^2.

The matrix x is the superposition of L, S, N:

$$x = L + S + N.$$

Value

A `list` object comprising:

`x`	An n-by-p matrix.
`L`	The latent low rank matrix.
`S`	The latent sparse matrix.

Author(s)

Jin Zhu

Examples

```
# Generate simulated data
n <- 30
p <- 20
dataset <- generate.matrix(n, p)

stats::heatmap(as.matrix(dataset[["S"]]),
  Rowv = NA,
  Colv = NA,
  scale = "none",
  col = grDevices::cm.colors(256),
  frame.plot = TRUE,
  margins = c(2.4, 2.4)
)
```

`generate.spc.matrix` *Generate matrix with sparse principal component*

Description

Generate simulated matrix that its principal component are sparse linear combination of its columns.

Usage

```
generate.spc.matrix(
  n,
  p,
  support.size = 3,
  snr = 20,
  sigma = NULL,
  sparse.loading = NULL,
  seed = 1
)
```

Arguments

n	The number of observations.
p	The number of predictors of interest.
support.size	A integer specify the number of non-zero entries in the first column of loading matrix.
snr	A positive value controlling the signal-to-noise ratio (SNR). A larger SNR implies the identification of sparse matrix is much easier. Default snr = Inf enforces no noise exists.
sigma	A numerical vector with length p specify the standard deviation of each columns. Default sigma = NULL implies it is determined by snr. If it is supplied, support.size would be omit.
sparse.loading	A p-by-p sparse orthogonal matrix. If it is supplied, support.size would be omit.
seed	random seed. Default: seed = 1.

Details

The methods for generating the matrix is detailedly described in the APPENDIX A: Data generation Section in Schipper et al (2021).

Value

A list object comprising:

x	An n-by-p matrix.
coef	The sparse loading matrix used to generate x.
support.size	A vector recording the number of non-zero entries in each .

References

Model selection techniques for sparse weight-based principal component analysis. de Schipper, Niek C and Van Deun, Katrijn. Journal of Chemometrics. 2021. doi: 10.1002/cem.3289.

`plot.abess` *Creat plot from a fitted "*`abess`*" object*

Description

Produces a coefficient/deviance/tuning-value plot for a fitted "abess" object.

Usage

```
## S3 method for class 'abess'
plot(
  x,
  type = c("coef", "l2norm", "dev", "dev.ratio", "tune"),
  label = FALSE,
  ...
)
```

Arguments

`x`	A "`abess`" object.
`type`	The type of terms to be plot in the y-axis. One of the following: "`coef`" (i.e., coefficients), "`l2norm`" (i.e., L2-norm of coefficients), "`dev`" (i.e., deviance), and "`tune`" (i.e., tuning value). Default is "`coef`".
`label`	A logical value. If `label` = `TRUE` (the default), label the curves with variable sequence numbers.
`...`	Other graphical parameters to plot

Value

No return value, called for side effects.

Note

If `family` = "`mgaussian`", `family` = "`ordinal`" or `family` = "`multinomial`", a coefficient plot is produced for each dimension of multivariate response.

See Also

`print.abess`, `predict.abess`, `coef.abess`, `extract.abess`, `plot.abess`, `deviance.abess`.

Examples

```
dataset <- generate.data(100, 20, 3)
abess_fit <- abess(dataset[["x"]], dataset[["y"]])
plot(abess_fit)
plot(abess_fit, type = "l2norm")
plot(abess_fit, type = "dev")
plot(abess_fit, type = "tune")
```

plot.abesspca *Creat plot from a fitted "abess" object*

Description

Produces a coefficient/deviance/tuning-value plot for a fitted "abess" object.

Usage

```
## S3 method for class 'abesspca'
plot(x, type = c("pev", "coef", "tune"), label = FALSE, ...)
```

Arguments

x	A "abess" object.
type	The type of terms to be plot in the y-axis. One of the following: "pev" (i.e., percent of explained variance), "coef" (i.e., coefficients), and "tune" (i.e., tuning value). Default is "coef".
label	A logical value. If label = TRUE (the default), label the curves with variable sequence numbers.
...	Other graphical parameters to plot

Value

No return value, called for side effects.

Note

If family = "mgaussian" or family = "multinomial", a coefficient plot is produced for each dimension of multivariate response.

See Also

print.abesspca, coef.abesspca, plot.abesspca.

Examples

```
abess_fit <- abesspca(USArrests, support.size = 1:4, sparse.type = "kpc")
plot(abess_fit)
plot(abess_fit, type = "coef")
plot(abess_fit, type = "tune")
```

plot.abessrpca *Creat plot from a fitted "abessrpca" object*

Description

Produces a sparse-matrix/loss/tuning-value plot for a fitted "abessrpca" object.

Usage

```
## S3 method for class 'abessrpca'
plot(x, type = c("S", "loss", "tune"), support.size = NULL, label = TRUE, ...)
```

Arguments

x	A "abessrpca" object.
type	The plot type. One of the following: "S" (i.e., a heatmap for the sparse matrix estimation), "loss" (i.e., a support.size versus loss plot), and "tune" (i.e., , a support.size versus tuning value plot). Default is "coef".
support.size	An integer vector specifies the sparse matrix fitted at given support.size to be returned. If support.size = NULL, then the sparse matrix with the least tuning value would be returned. Default: support.size = NULL.
label	A logical value. If label = TRUE (the default), label the curves with variable sequence numbers.
...	Other graphical parameters to plot or stats::heatmap function

Value

No return value, called for side effects.

predict.abess *Make predictions from a fitted "abess" object.*

Description

Make predictions from a fitted "abess" object.

Usage

```
## S3 method for class 'abess'
predict(object, newx, type = c("link", "response"), support.size = NULL, ...)
```

Arguments

object	An "abess" project.
newx	New data used for prediction. If omitted, the fitted linear predictors are used.
type	type = "link" gives the linear predictors for "binomial", "poisson" or "cox" models; for "gaussian" models it gives the fitted values. type = "response" gives the fitted probabilities for "binomial" and "ordinal", fitted mean for "poisson" and the fitted relative-risk for "cox"; for "gaussian", type = "response" is equivalent to type = "link".
support.size	An integer value specifies the model size fitted at given support.size. If support.size = NULL, then the model with the best tuning value would be returned. Default: support.size = NULL.
...	Additional arguments affecting the predictions produced.

Value

The object returned depends on the types of family.

See Also

print.abess, predict.abess, coef.abess, extract.abess, plot.abess, deviance.abess.

---

print.abess *Print method for a fitted "abess" object*

Description

Prints the fitted model and returns it invisibly.

Usage

```
## S3 method for class 'abess'
print(x, digits = max(5, getOption("digits") - 5), ...)
```

Arguments

x	A "abess" object.
digits	Minimum number of significant digits to be used.
...	additional print arguments

Details

Print a data.frame with three columns: the first column is support size of model; the second column is deviance of model; the last column is the tuning value of the certain tuning type.

Value

No return value, called for side effects

See Also

print.abess, predict.abess, coef.abess, extract.abess, plot.abess, deviance.abess.

---

print.abesspca *Print method for a fitted "abesspca" object*

Description

Prints the fitted model and returns it invisibly.

Usage

```
## S3 method for class 'abesspca'
print(x, digits = max(5, getOption("digits") - 5), ...)
```

Arguments

x	A "abesspca" object.
digits	Minimum number of significant digits to be used.
...	additional print arguments

Details

Print a data.frame with three columns: the first column is support size of model; the second column is the explained variance of model; the last column is the percent of explained variance of model.

Value

No return value, called for side effects

See Also

print.abesspca, coef.abesspca, plot.abesspca.

print.abessrpca *Print method for a fitted "abessrpca" object*

Description

Prints the fitted model and returns it invisibly.

Usage

```
## S3 method for class 'abessrpca'
print(x, digits = max(5, getOption("digits") - 5), ...)
```

Arguments

x	A "abessrpca" object.
digits	Minimum number of significant digits to be used.
...	additional print arguments

Details

Print a data.frame with three columns: the first column is support size of model; the second column is the explained variance of model; the last column is the percent of explained variance of model.

Value

No return value, called for side effects

trim32 *The Bardet-Biedl syndrome Gene expression data*

Description

Gene expression data (500 gene probes for 120 samples) from the microarray experiments of mammalianeye tissue samples of Scheetz et al. (2006).

Format

A data frame with 120 rows and 501 variables, where the first variable is the expression level of TRIM32 gene, and the remaining 500 variables are 500 gene probes.

Details

In this study, laboratory rats (Rattus norvegicus) were studied to learn about gene expression and regulation in the mammalian eye. Inbred rat strains were crossed and tissue extracted from the eyes of 120 animals from the F2 generation. Microarrays were used to measure levels of RNA expression in the isolated eye tissues of each subject. Of the 31,000 different probes, 18,976 were detected at a sufficient level to be considered expressed in the mammalian eye. For the purposes of this analysis, we treat one of those genes, Trim32, as the outcome. Trim32 is known to be linked with a genetic disorder called Bardet-Biedl Syndrome (BBS): the mutation (P130S) in Trim32 gives rise to BBS.

Note

This data set contains 120 samples with 500 predictors. The 500 predictors are features with maximum marginal correlation to Trim32 gene.

References

T. Scheetz, k. Kim, R. Swiderski, A. Philp, T. Braun, K. Knudtson, A. Dorrance, G. DiBona, J. Huang, T. Casavant, V. Sheffield, E. Stone. Regulation of gene expression in the mammalian eye and its relevance to eye disease. Proceedings of the National Academy of Sciences of the United States of America, 2006.

Index