Package 'adelie'

September 3, 2024

Title Group Lasso and Elastic Net Solver for Generalized Linear Models

Version 1.0.2

Date 2024-08-31

Description Extremely efficient procedures for fitting the entire group lasso and group elastic net regularization path for GLMs, multinomial, the Cox model and multi-task Gaussian models. Similar to the R package glmnet in scope of models, and in computational speed. This package provides R bindings to the C++ code underlying the corresponding Python package 'adelie'. These bindings offer a general purpose group elastic net solver, a wide range of matrix classes that can exploit special structure to allow large-scale inputs, and an assortment of generalized linear model classes for fitting various types of data. The package includes The package is an implementation of Yang, J. and Hastie, T. (2024) <doi:10.48550/arXiv.2405.08631>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

LinkingTo Rcpp, RcppEigen

SystemRequirements C++17

Imports Matrix, r2r, Rcpp, methods, stringr,

Suggests ggplot2, gridExtra, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/JamesYang007/adelie-r

BugReports https://github.com/JamesYang007/adelie-r/issues

NeedsCompilation yes

Author James Yang [aut, cph],
Trevor Hastie [aut, cph, cre],
Balasubramanian Narasimhan [aut]

Maintainer Trevor Hastie <hastie@stanford.edu>

Repository CRAN

Date/Publication 2024-09-03 10:30:14 UTC

Contents

`cv.grpnet` *Cross-validation for grpnet*

Description

Does k-fold cross-validation for grpnet

Usage

```
cv.grpnet(
  X,
  glm,
  n_folds = 10,
  foldid = NULL,
  min_ratio = 0.01,
  lmda_path_size = 100,
  offsets = NULL,
  progress_bar = FALSE,
  n_threads = 1,
  ...
)
```

Arguments

X	Feature matrix. Either a regualr R matrix, or else an adelie custom matrix class, or a concatination of such.
glm	GLM family/response object. This is an expression that represents the family, the reponse and other arguments such as weights, if present. The choices are glm.gaussian(), glm.binomial(), glm.poisson(), glm.multinomial(), glm.cox(), glm.multinomial(), and glm.multigaussian(). This is a required argument, and there is no default. In the simple example below, we use glm.gaussian(y).
n_folds	(default 10). Although n_folds can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is n_folds=3.
foldid	An optional vector of values between 1 and n_folds identifying what fold each observation is in. If supplied, n_folds can be missing.
min_ratio	Ratio between smallest and largest value of lambda. Default is 1e-2.
lmda_path_size	Number of values for lambda, if generated automatically. Default is 100.
offsets	Offsets, default is NULL. If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
progress_bar	Progress bar. Default is FALSE.
n_threads	Number of threads, default 1.
...	Other arguments that can be passed to grpnet

Details

The function runs grpnet n_folds+1 times; the first to get the lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The out-of-fold deviance is accumulated, and the average deviance and standard deviation over the folds is computed. Note that cv.grpnet does NOT search for values for alpha. A specific value should be supplied, else alpha=1 is assumed by default. If users would like to cross-validate alpha as well, they should call cv.grpnet with a pre-computed vector foldid, and then use this same foldid vector in separate calls to cv.grpnet with different values of alpha. Note also that the results of cv.grpnet are random, since the folds are selected at random. Users can reduce this randomness by running cv.grpnet many times, and averaging the error curves.

Value

an object of class "cv.grpnet" is returned, which is a list with the ingredients of the cross-validation fit.

lambda	the values of lambda used in the fits.
cvm	The mean cross-validated deviance - a vector of length length(lambda).
cvsd	estimate of standard error of cvm.
cvup	upper curve = cvm+cvsd.
cvlo	lower curve = cvm-cvsd.
nzero	number of non-zero coefficients at each lambda.
name	a text string indicating type of measure (for plotting purposes). Currently this is "deviance"
grpnet.fit	a fitted grpnet object for the full data.
lambda.min	value of lambda that gives minimum cvm.
lambda.1se	largest value of lambda such that mean deviance is within 1 standard error of the minimum.
index	a one column matrix with the indices of lambda.min and lambda.1se in the sequence of coefficients, fits etc.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Tibshirani,Robert, Bien, J., Friedman, J., Hastie, T.,Simon, N.,Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems, JRSSB, Vol. 74(2), 245-266*, https://arxiv.org/abs/1011.2234.

Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
```

```
fit <- grpnet(X, glm.gaussian(y))
print(fit)
```

---

gaussian_cov *Solves group elastic net via covariance method.*

Description

Solves group elastic net via covariance method.

Usage

```
gaussian_cov(
  A,
  v,
  constraints = NULL,
  groups = NULL,
  alpha = 1,
  penalty = NULL,
  lmda_path = NULL,
  max_iters = as.integer(1e+05),
  tol = 1e-07,
  rdev_tol = 0.001,
  newton_tol = 1e-12,
  newton_max_iters = 1000,
  n_threads = 1,
  early_exit = TRUE,
  screen_rule = "pivot",
  min_ratio = 0.01,
  lmda_path_size = 100,
  max_screen_size = NULL,
  max_active_size = NULL,
  pivot_subset_ratio = 0.1,
  pivot_subset_min = 1,
  pivot_slack_ratio = 1.25,
  check_state = FALSE,
  progress_bar = TRUE,
  warm_start = NULL
)
```

Arguments

A	Positive semi-definite matrix.
v	Linear term.
constraints	Constraints.
groups	Groups.

alpha	Elastic net parameter.
penalty	Penalty factor.
lmda_path	The regularization path.
max_iters	Maximum number of coordinate descents.
tol	Coordinate descent convergence tolerance.
rdev_tol	Relative percent deviance explained tolerance.
newton_tol	Convergence tolerance for the BCD update.
newton_max_iters	Maximum number of iterations for the BCD update.
n_threads	Number of threads.
early_exit	TRUE if the function should exit early.
screen_rule	Screen rule (currently the only value is the default "pivot".
min_ratio	Ratio between largest and smallest regularization parameter, default is 0.01.
lmda_path_size	Number of regularization steps in the path, default is 100.
max_screen_size	Maximum number of screen groups, default is NULL for no maximum.
max_active_size	Maximum number of active groups, default is NULL for no maximum.
pivot_subset_ratio	Subset ratio of pivot rule, default is 0.1.
pivot_subset_min	Minimum subset of pivot rule, default is 1.
pivot_slack_ratio	Slack ratio of pivot rule, default is 1.25.
check_state	Check state, default is FALSE.
progress_bar	Progress bar, default is TRUE.
warm_start	Warm start, default is NULL (no warm start).

Value

State of the solver.

Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
A <- t(X) %*% X / n
v <- t(X) %*% y / n
state <- gaussian_cov(A, v)
```

`glm.binomial` *Creates a Binomial GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.binomial(y, weights = NULL, link = "logit")
```

Arguments

`y`	Binary response vector, with values 0 or 1, or a logical vector. Alternatively, if data are represented by a two-column matrix of proportions (with row-sums = 1), then one can provide one of the columns as the response. This is useful for grouped binomial data, where each observation represents the result of `m[i]` successes out of `n[i]` trials. Then the response is provided as `y[i] = m[i]/n[i]` and the corresponding element of the weight vector as `w[i]=n[i]`. Alternatively can use `glm.multinomial()` instead.
`weights`	Observation weight vector, with default `NULL`.
`link`	The link function type, with choice `"logit"` (default) or `"probit"`).

Value

Binomial GLM object.

Author(s)

Trevor Hastie and James Yang
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
y <- rbinom(n, 1, 0.5)
obj <- glm.binomial(y)
```

`glm.cox` *Creates a Cox GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.cox(
  stop,
  status,
  start = -Inf,
  weights = NULL,
  tie_method = c("efron", "breslow")
)
```

Arguments

`stop`	Stop time vector.
`status`	Binary status vector of same length as `stop`, with 1 a "death", and 0 censored.
`start`	Start time vector. Default is a vector of `-Inf` of same length as `stop`.
`weights`	Observation weights, with default `NULL`.
`tie_method`	The tie-breaking method - one of `"efron"` (default) or `"breslow"`.

Value

Cox GLM object.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
start <- sample.int(20, size=n, replace=TRUE)
stop <- start + 1 + sample.int(5, size=n, replace=TRUE)
status <- rbinom(n, 1, 0.5)
obj <- glm.cox(start, stop, status)
```

`glm.gaussian` *Creates a Gaussian GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.gaussian(y, weights = NULL, opt = TRUE)
```

Arguments

`y`	Response vector.
`weights`	Observation weight vector, with default `NULL`.
`opt`	If `TRUE` (default), an optimized routine is run.

Value

Gaussian GLM

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
y <- rnorm(n)
obj <- glm.gaussian(y)
```

`glm.multigaussian` *Creates a MultiGaussian GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.multigaussian(y, weights = NULL, opt = TRUE)
```

Arguments

`y`	Response matrix, with two or more columns.
`weights`	Observation weight vector, with default `NULL`.
`opt`	If `TRUE` (default), an optimized routine is run.

Value

MultiGaussian GLM object.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
K <- 5
y <- matrix(rnorm(n*K), n, K)
obj <- glm.multigaussian(y)
```

`glm.multinomial` *Creates a Multinomial GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.multinomial(y, weights = NULL)
```

Arguments

`y`	Response matrix with `K>1` columns, and row sums equal to 1. This can either be a "one-hot" encoded version of a K-category factor variable, or else a matrix of proportions. This is useful for grouped multinomial data, where column `y[i, k]` represents the proportion of outcomes in category k in `n[i]` trials. Then the corresponding element of the weight vector is `w[i]=n[i]`.
`weights`	Observation weights.

Value

Multinomial GLM object.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
K <- 5
y <- t(rmultinom(n, 1, rep(1/K, K)))
obj <- glm.multinomial(y)
```

`glm.poisson` *Creates a Poisson GLM family object.*

Description

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

Usage

```
glm.poisson(y, weights = NULL)
```

Arguments

`y`	Response vector of non-negative counts.
`weights`	Observation weight vector, with default `NULL`.

Value

Poisson GLM object.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`glm.gaussian`, `glm.binomial`, `glm.poisson`, `glm.multinomial`, `glm.multigaussian`, `glm.cox`.

Examples

```
n <- 100
y <- rpois(n, 1)
obj <- glm.poisson(y)
```

grpnet *fit a GLM with group lasso or group elastic-net regularization*

Description

Computes a group elastic-net regularization path for a variety of GLM and other families, including the Cox model. This function extends the abilities of the `glmnet` package to allow for grouped regularization. The code is very efficient (core routines are written in C++), and allows for specialized matrix classes.

Usage

```
grpnet(
  X,
  glm,
  constraints = NULL,
  groups = NULL,
  alpha = 1,
  penalty = NULL,
  offsets = NULL,
  lambda = NULL,
  standardize = TRUE,
  irls_max_iters = as.integer(10000),
  irls_tol = 1e-07,
  max_iters = as.integer(1e+05),
  tol = 1e-07,
  adev_tol = 0.9,
  ddev_tol = 0,
  newton_tol = 1e-12,
  newton_max_iters = 1000,
  n_threads = 1,
  early_exit = TRUE,
  intercept = TRUE,
  screen_rule = c("pivot", "strong"),
  min_ratio = 0.01,
  lmda_path_size = 100,
  max_screen_size = NULL,
  max_active_size = NULL,
  pivot_subset_ratio = 0.1,
  pivot_subset_min = 1,
  pivot_slack_ratio = 1.25,
  check_state = FALSE,
  progress_bar = FALSE,
  warm_start = NULL
)
```

Arguments

X	Feature matrix. Either a regualr R matrix, or else an adelie custom matrix class, or a concatination of such.
glm	GLM family/response object. This is an expression that represents the family, the reponse and other arguments such as weights, if present. The choices are glm.gaussian(), glm.binomial(), glm.poisson(), glm.multinomial(), glm.cox(), glm.multinomial(), and glm.multigaussian(). This is a required argument, and there is no default. In the simple example below, we use glm.gaussian(y).
constraints	Constraints on the parameters. Currently these are ignored.
groups	This is an ordered vector of integers that represents the groupings, with each entry indicating where a group begins. The entries refer to column numbers in the feature matrix. If there are p features, the default is 1:p (no groups). (Note that in the output of grpnet this vector might be shifted to start from 0, since internally adelie uses zero-based indexing.)
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2 \sum_j ||\beta_j||_2^2 + \alpha \sum_j ||\beta_j||_2,$$ where thte sum is over groups. alpha=1 is pure group lasso penalty, and alpha=0 the pure ridge penalty.
penalty	Separate penalty factors can be applied to each group of coefficients. This is a number that multiplies lambda to allow differential shrinkage for groups. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is square-root of group sizes for each group.
offsets	Offsets, default is NULL. If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
lambda	A user supplied lambda sequence. Typical usage is to have the program compute its own lambda sequence based on lmda_path_size and min_ratio.
standardize	If TRUE (the default), the columns of X are standardized before the fit is computed. This is good practice if the features are a mixed bag, because it has an impact on the penalty. The regularization path is computed using the standardized features, and the standardization information is saved on the object for making future predictions.
irls_max_iters	Maximum number of IRLS iterations, default is 1e4.
irls_tol	IRLS convergence tolerance, default is 1e-7.
max_iters	Maximum total number of coordinate descent iterations, default is 1e5.
tol	Coordinate descent convergence tolerance, default 1e-7.
adev_tol	Fraction deviance explained tolerance, default 0.9. This can be seen as a limit on overfitting the training data.
ddev_tol	Difference in fraction deviance explained tolerance, default 0. If a step in the path changes the deviance by this amount or less, the algorithm truncates the path.

newton_tol	Convergence tolerance for the BCD update, default 1e-12. This parameter controls the iterations in each block-coordinate step to establish the block solution.
newton_max_iters	Maximum number of iterations for the BCD update, default 1000.
n_threads	Number of threads, default 1.
early_exit	TRUE if the function should be allowed to exit early.
intercept	Default TRUE to include an unpenalized intercept.
screen_rule	Screen rule, with default "pivot". Other option is "strong". (an empirical improvement over "strong", the other option.)
min_ratio	Ratio between smallest and largest value of lambda. Default is 1e-2.
lmda_path_size	Number of values for lambda, if generated automatically. Default is 100.
max_screen_size	Maximum number of screen groups. Default is NULL.
max_active_size	Maximum number of active groups. Default is NULL.
pivot_subset_ratio	Subset ratio of pivot rule. Default is 0.1. Users not expected to fiddle with this.
pivot_subset_min	Minimum subset of pivot rule. Defaults is 1. Users not expected to fiddle with this.
pivot_slack_ratio	Slack ratio of pivot rule, default is 1.25. Users not expected to fiddle with this. See reference for details.
check_state	Check state. Internal parameter, with default FALSE.
progress_bar	Progress bar. Default is FALSE.
warm_start	Warm start (default is NULL). Internal parameter.

Value

A list of class "grpnet". This has a main component called state which represents the fitted path, and a few extra useful components such as the call, the family name, and group_sizes. Users typically use methods like predict(), print(), plot() etc to examine the object.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.

Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Tibshirani,Robert, Bien, J., Friedman, J., Hastie, T.,Simon, N.,Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems, JRSSB, Vol. 74(2), 245-266*, https://arxiv.org/abs/1011.2234.

See Also

`cv.grpnet`, `predict.grpnet`, `plot.grpnet`, `print.grpnet`.

Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
fit <- grpnet(X, glm.gaussian(y))
print(fit)
```

---

io.snp_phased_ancestry

IO handler for SNP phased, ancestry matrix.

Description

IO handler for SNP phased, ancestry matrix.

Usage

```
io.snp_phased_ancestry(filename, read_mode = "file")
```

Arguments

`filename`	File name.
`read_mode`	Reading mode.

Value

IO handler for SNP phased, ancestry data.

Examples

```
n <- 123
s <- 423
A <- 8
filename <- paste(tempdir(), "snp_phased_ancestry_dummy.snpdat", sep="/")
handle <- io.snp_phased_ancestry(filename)
calldata <- matrix(
    as.integer(sample.int(
        2, n * s * 2,
        replace=TRUE,
        prob=c(0.7, 0.3)
    ) - 1),
    n, s * 2
)
ancestries <- matrix(
    as.integer(sample.int(
        A, n * s * 2,
        replace=TRUE,
        prob=rep_len(1/A, A)
    ) - 1),
    n, s * 2
)
handle$write(calldata, ancestries, A, 1)
handle$read()
file.remove(filename)
```

`io.snp_unphased` *IO handler for SNP unphased matrix.*

Description

IO handler for SNP unphased matrix.

Usage

```
io.snp_unphased(filename, read_mode = "file")
```

Arguments

`filename`	File name.
`read_mode`	Reading mode.

Value

IO handler for SNP unphased data.

Examples

```
n <- 123
s <- 423
filename <- paste(tempdir(), "snp_unphased_dummy.snpdat", sep="/")
handle <- io.snp_unphased(filename)
mat <- matrix(
    as.integer(sample.int(
        3, n * s,
        replace=TRUE,
        prob=c(0.7, 0.2, 0.1)
    ) - 1),
    n, s
)
impute <- double(s)
handle$write(mat, "mean", impute, 1)
handle$read()
file.remove(filename)
```

`matrix.block_diag` *Creates a block-diagonal matrix.*

Description

Creates a block-diagonal matrix.

Usage

```
matrix.block_diag(mats, n_threads = 1)
```

Arguments

`mats`	List of matrices.
`n_threads`	Number of threads.

Value

Block-diagonal matrix.

Author(s)

Trevor Hastie and James Yang
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
ps <- c(10, 20, 30)
mats <- lapply(ps, function(p) {
    X <- matrix(rnorm(n * p), n, p)
    matrix.dense(t(X) %*% X, method="cov")
})
out <- matrix.block_diag(mats)
```

`matrix.concatenate` *Creates a concatenation of the matrices.*

Description

Creates a concatenation of the matrices.

Usage

```
matrix.concatenate(mats, axis = 2, n_threads = 1)
```

Arguments

`mats`	List of matrices.
`axis`	The axis along which the matrices will be joined. With axis = 2 (default) this function is equivalent to `cbind()` and axis = 1 is equivalent to `rbind()`.
`n_threads`	Number of threads.

Value

Concatenation of matrices. The object is an S4 class with methods for efficient computation in C++ by adelie. Note that for the object itself axis is represented with base 0 (so 1 less than the argument here).

Author(s)

Trevor Hastie and James Yang
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
ps <- c(10, 20, 30)
ps <- c(10, 20, 30)
n <- 100
mats <- lapply(ps, function(p) {
    matrix.dense(matrix(rnorm(n * p), n, p))
})
out <- matrix.concatenate(mats, axis=2)
```

`matrix.dense` *Creates a dense matrix object.*

Description

Creates a dense matrix object.

Usage

```
matrix.dense(mat, method = c("naive", "cov"), n_threads = 1)
```

Arguments

`mat`	The dense matrix.
`method`	Method type, with default `method="naive"`. If `method="cov"`, the matrix is used with the solver `gaussian_cov()`. Used for `glm.gaussian()` and `glm.multigaussian()` families. Generally "naive" is used for wide matrices, and "cov" for tall matrices.
`n_threads`	Number of threads.

Value

Dense matrix. The object is an S4 class with methods for efficient computation by adelie.

Author(s)

Trevor Hastie and James Yang
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
out <- matrix.dense(X_dense, method="naive")
A_dense <- t(X_dense) %*% X_dense
out <- matrix.dense(A_dense, method="cov")
```

matrix.eager_cov *Creates an eager covariance matrix.*

Description

Creates an eager covariance matrix.

Usage

```
matrix.eager_cov(mat, n_threads = 1)
```

Arguments

mat	A dense matrix to be used with the gaussian_cov() solver.
n_threads	Number of threads.

Value

The dense covariance matrix. This matrix is exactly t(mat)%*%mat, computed with some efficiency.

Examples

```
n <- 100
p <- 20
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.eager_cov(mat)
```

matrix.interaction *Creates a matrix with pairwise interactions.*

Description

Creates a matrix with pairwise interactions.

Usage

```
matrix.interaction(
  mat,
  intr_keys = NULL,
  intr_values,
  levels = NULL,
  n_threads = 1
)
```

Arguments

`mat`	The dense matrix, which can include factors with levels coded as non-negative integers.
`intr_keys`	List of feature indices. This is a list of all features with which interactions can be formed. Default is `1:p` where `p` is the number of columns in `mat`.
`intr_values`	List of list of feature indices. For each of the `m <= p` indices listed in `intr_keys`, there is a list of indices indicating which columns are candidates for interaction with that feature. If a list is `list(NULL)`, that means all other features are candidates for interactions. The default is a list of length `m` where each element is `list(NULL)`; that is `rep(list(NULL), m`.
`levels`	Number of levels for each of the columns of `mat`, with `1` representing a quantitative feature. A factor with `K` levels should be represented by the numbers `0,1,...,K-1`.
`n_threads`	Number of threads.

Value

Pairwise interaction matrix. Logic is used to avoid repetitions. For each factor variable, the column is one-hot-encoded to form a basis for that feature. The object is an S4 class with methods for efficient computation by adelie. Note that some of the arguments are transformed to C++ base 0 for internal use, and if the object is examined, it will reflect that.

Author(s)

Trevor Hastie and James Yang
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 10
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
X_dense[,1] <- rbinom(n, 4, 0.5)
intr_keys <- c(1, 2)
intr_values <- list(NULL, c(1, 3))
levels <- c(c(5), rep(1, p-1))
out <- matrix.interaction(X_dense, intr_keys, intr_values, levels)
```

`matrix.kronecker_eye` *Creates a Kronecker product with an identity matrix.*

Description

Creates a Kronecker product with an identity matrix.

Usage

```
matrix.kronecker_eye(mat, K = 1, n_threads = 1)
```

Arguments

mat	The matrix to view as a Kronecker product.
K	Dimension of the identity matrix (default is 1, which does essentially nothing).
n_threads	Number of threads.

Value

Kronecker product with identity matrix. If mat is n x p, the the resulting matrix will be nK x np. The object is an S4 class with methods for efficient computation by adelie.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
K <- 2
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.kronecker_eye(mat, K)
mat <- matrix.dense(mat)
out <- matrix.kronecker_eye(mat, K)
```

---

matrix.lazy_cov *Creates a lazy covariance matrix.*

Description

Creates a lazy covariance matrix.

Usage

```
matrix.lazy_cov(mat, n_threads = 1)
```

Arguments

mat	A dense data matrix to be used with the gaussian_cov() solver.
n_threads	Number of threads.

Value

Lazy covariance matrix. This is essentially the same matrix, but with a setup to create covariance terms as needed on the fly. The object is an S4 class with methods for efficient computation by adelie.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.lazy_cov(mat)
```

---

matrix.one_hot *Creates a one-hot encoded matrix.*

Description

Creates a one-hot encoded matrix.

Usage

```
matrix.one_hot(mat, levels = NULL, n_threads = 1)
```

Arguments

`mat`	A dense matrix, which can include factors with levels coded as non-negative integers.
`levels`	Number of levels for each of the columns of `mat`, with `1` representing a quantitative feature. A factor with `K` levels should be represented by the numbers `0,1,...,K-1`.
`n_threads`	Number of threads.

Value

One-hot encoded matrix. All the factor columns, with levels>1, are replaced by a collection of one-hot encoded versions (dummy matrices). The resulting matrix has `sum(levels)` columns. The object is an S4 class with methods for efficient computation by adelie. Note that some of the arguments are transformed to C++ base 0 for internal use, and if the object is examined, it will reflect that.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.one_hot(mat)
```

---

matrix.snp_phased_ancestry

Creates a SNP phased, ancestry matrix.

Description

Creates a SNP phased, ancestry matrix.

Usage

```
matrix.snp_phased_ancestry(io, n_threads = 1)
```

Arguments

io	IO handler.
n_threads	Number of threads.

Value

SNP phased, ancestry matrix.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 123
s <- 423
A <- 8
filename <- paste(tempdir(), "snp_phased_ancestry_dummy.snpdat", sep="/")
handle <- io.snp_phased_ancestry(filename)
calldata <- matrix(
    as.integer(sample.int(
        2, n * s * 2,
        replace=TRUE,
```

```
        prob=c(0.7, 0.3)
    ) - 1),
    n, s * 2
)
ancestries <- matrix(
    as.integer(sample.int(
        A, n * s * 2,
        replace=TRUE,
        prob=rep_len(1/A, A)
    ) - 1),
    n, s * 2
)
handle$write(calldata, ancestries, A, 1)
out <- matrix.snp_phased_ancestry(handle)
file.remove(filename)
```

`matrix.snp_unphased` *Creates a SNP unphased matrix.*

Description

Creates a SNP unphased matrix.

Usage

```
matrix.snp_unphased(io, n_threads = 1)
```

Arguments

`io`	IO handler.
`n_threads`	Number of threads.

Value

SNP unphased matrix.

Examples

```
n <- 123
s <- 423
filename <- paste(tempdir(), "snp_unphased_dummy.snpdat", sep="/")
handle <- io.snp_unphased(filename)
mat <- matrix(
    as.integer(sample.int(
        3, n * s,
        replace=TRUE,
        prob=c(0.7, 0.2, 0.1)
    ) - 1),
    n, s
```

```
)
impute <- double(s)
handle$write(mat, "mean", impute, 1)
out <- matrix.snp_unphased(handle)
file.remove(filename)
```

`matrix.sparse` *Creates a sparse matrix object.*

Description

Creates a sparse matrix object.

Usage

```
matrix.sparse(mat, method = c("naive", "cov"), n_threads = 1)
```

Arguments

`mat`	A sparse matrix.
`method`	Method type, with default `method="naive"`. If `method="cov"`, the matrix is used with the solver `gaussian_cov()`. Used for `glm.gaussian()` and `glm.multigaussian()` families. Generally "naive" is used for wide matrices, and "cov" for tall matrices.
`n_threads`	Number of threads.

Value

Sparse matrix object. The object is an S4 class with methods for efficient computation by adelie.

Examples

```
n <- 100
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
X_sp <- as(X_dense, "dgCMatrix")
out <- matrix.sparse(X_sp, method="naive")
A_dense <- t(X_dense) %*% X_dense
A_sp <- as(A_dense, "dgCMatrix")
out <- matrix.sparse(A_sp, method="cov")
```

`matrix.standardize` *Creates a standardized matrix.*

Description

Creates a standardized matrix.

Usage

```
matrix.standardize(
  mat,
  centers = NULL,
  scales = NULL,
  weights = NULL,
  ddof = 0,
  n_threads = 1
)
```

Arguments

`mat`	An `adelie` matrix.
`centers`	The center values. Default is to use the column means.
`scales`	The scale values. Default is to use the sample standard deviations.
`weights`	Observation weight vector, which defaults to 1/n per observation.
`ddof`	Degrees of freedom for standard deviations, with default 0 (1/n). The alternative is 1 leading to 1/(n-1).
`n_threads`	Number of threads.

Value

Standardized matrix. The object is an S4 class with methods for efficient computation by adelie.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
X <- matrix(rnorm(n * p), n, p)
out <- matrix.standardize(matrix.dense(X))
```

`matrix.subset` *Creates a subset of the matrix along an axis.*

Description

Creates a subset of the matrix along an axis.

Usage

```
matrix.subset(mat, indices, axis = 1, n_threads = 1)
```

Arguments

`mat`	The `adelie` matrix to subset.
`indices`	Vector of indices to subset the matrix.
`axis`	The axis along which to subset (2 is columns, 1 is rows).
`n_threads`	Number of threads.

Value

Matrix subsetted along the appropriate axis. The object is an S4 class with methods for efficient computation by adelie.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
n <- 100
p <- 20
X <- matrix.dense(matrix(rnorm(n * p), n, p))
indices <- c(1, 3, 10)
out <- matrix.subset(X, indices, axis=1)
out <- matrix.subset(X, indices, axis=2)
```

`plot.cv.grpnet` *plot the cross-validation curve produced by cv.grpnet*

Description

Plots the cross-validation curve, and upper and lower standard deviation curves, as a function of the `lambda` values used.

Usage

```
## S3 method for class 'cv.grpnet'
plot(x, sign.lambda = -1, ...)
```

Arguments

`x`	fitted `"cv.grpnet"` object
`sign.lambda`	Either plot against `log(lambda)` or its negative (default) if `sign.lambda=-1`
`...`	Other graphical parameters

Details

A plot is produced, and nothing is returned.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
Adelie Python user guide https://jamesyang007.github.io/adelie/

See Also

`grpnet` and `cv.grpnet`.

Examples

```
set.seed(1010)
n = 1000
p = 100
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta = rnorm(nzc)
fx = (x[, seq(nzc)] %*% beta)
eps = rnorm(n) * 5
```

```
y = drop(fx + eps)
px = exp(fx)
px = px/(1 + px)
ly = rbinom(n = length(px), prob = px, size = 1)
cvob1 = cv.grpnet(x, glm.gaussian(y))
plot(cvob1)
title("Gaussian Family", line = 2.5)
frame()
set.seed(1011)
cvob2 = cv.grpnet(x, glm.binomial(ly))
plot(cvob2)
title("Binomial Family", line = 2.5)
```

`plot.grpnet` *plot coefficients from a "grpnet" object*

Description

Produces a coefficient profile plot of the coefficient paths for a fitted `"grpnet"` object.

Usage

```
## S3 method for class 'grpnet'
plot(x, sign.lambda = -1, glm.name = TRUE, ...)
```

Arguments

`x`	fitted `"grpnet"` model
`sign.lambda`	This determines whether we plot against `log(lambda)` or its negative. values are `-1`(default) or `1`
`glm.name`	This is a logical (default `TRUE`), and causes the glm name of the model to be included in the plot.
`...`	Other graphical parameters to plot

Details

A coefficient profile plot is produced. If `x` is a multinomial or multigaussian model, the 2norm of the vector of coefficients is plotted.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.

See Also

grpnet, and print, and coef methods, and cv.grpnet.

Examples

```
x=matrix(rnorm(100*20),100,20)
y=rnorm(100)
fit1=grpnet(x,glm.gaussian(y))
plot(fit1)
g4=diag(4)[sample(1:4,100,replace=TRUE),]
fit2=grpnet(x,glm.multinomial(g4))
plot(fit2,lwd=3)
fit3=grpnet(x,glm.gaussian(y),groups=c(1,5,9,13,17))
plot(fit3)
```

---

predict.cv.grpnet *make predictions from a "cv.grpnet" object.*

Description

This function makes predictions from a cross-validated grpnet model, using the stored "grpnet.fit" object, and the optimal value chosen for lambda.

Usage

```
## S3 method for class 'cv.grpnet'
predict(object, newx, lambda = c("lambda.1se", "lambda.min"), ...)
```

Arguments

object	Fitted "cv.grpnet".
newx	Matrix of new values for x at which predictions are to be made. Can be a matrix, a sparse matrix as in Matrix package, or else any of the matrix forms allowable in the adelie package. This argument is not used for type="coefficients".
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value lambda="lambda.1se" stored on the CV object. Alternatively lambda="lambda.min" can be used. If lambda is numeric, it is taken as the value(s) of lambda to be used.
...	Not used. Other arguments to predict.

Details

This function makes it easier to use the results of cross-validation to make a prediction.

Value

The object returned depends on the arguments.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.

See Also

`grpnet`, and `print`, and `coef` methods, and `cv.grpnet`.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
cv.fit = cv.grpnet(x, glm.gaussian(y))
predict(cv.fit, newx = x[1:5, ])
coef(cv.fit)
coef(cv.fit, lambda = "lambda.min")
predict(cv.fit, newx = x[1:5, ], lambda = c(0.001, 0.002))
```

`predict.grpnet` *make predictions from a "grpnet" object.*

Description

Similar to other predict methods, this functions predicts linear predictors, coefficients and more from a fitted `"grpnet"` object.

Usage

```
## S3 method for class 'grpnet'
predict(
  object,
  newx,
  lambda = NULL,
  type = c("link", "response", "coefficients"),
  newoffsets = NULL,
  ...
)

## S3 method for class 'grpnet'
coef(object, lambda = NULL, ...)
```

Arguments

object	Fitted "grpnet" model.
newx	Matrix of new values for x at which predictions are to be made. Can be a matrix, a sparse matrix as in Matrix package, or else any of the matrix forms allowable in the adelie package. The number of columns must match that of the input matrix used in fitting object. If the model object was fit with standardize=TRUE, the saved centers and scaling will be applied to this matrix. This argument is not used for type="coefficients"
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model. If values of lambda are supplied, the function uses linear interpolation to make predictions for values of lambda that do not coincide with those used in the fitting algorithm.
type	Type of prediction required. Type "link" is the default, and gives the linear predictors. Type "response" applies the inverse link to these predictions. Type "coefficients" extracts the coefficients, intercepts and the active-set sizes.
newoffsets	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients" or type="nonzero")
...	Currently ignored.

Details

The shape of the objects returned are different for "multinomial" and "multigaussian" objects coef(...) is equivalent to predict(type="coefficients",...)

Value

The object returned depends on type.

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
Adelie Python user guide https://jamesyang007.github.io/adelie/

See Also

grpnet, and print, and coef methods, and cv.grpnet.

Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
fit <- grpnet(X, glm.gaussian(y))
coef(fit)
predict(fit,newx = X[1:5,])
```

`print.cv.grpnet` *print a cross-validated grpnet object*

Description

Print a summary of the results of cross-validation for a grpnet model.

Usage

```
## S3 method for class 'cv.grpnet'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	fitted 'cv.grpnet' object
`digits`	significant digits in printout
`...`	additional print arguments

Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan
Maintainer: Trevor Hastie hastie@stanford.edu

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.

See Also

`grpnet`, `predict` and `coef` methods.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = cv.grpnet(x, glm.gaussian(y))
print(fit1)
```

`print.grpnet` *print a grpnet object*

Description

Print a summary of the grpnet path at each step along the path.

Usage

```
## S3 method for class 'grpnet'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	fitted grpnet object
`digits`	significant digits in printout
`...`	additional print arguments

Details

The call that produced the object `x` is printed, followed by a three-column matrix with columns `Df`, `%Dev` and `Lambda`. The `Df` column is the number of nonzero coefficients (Df is a reasonable name only for lasso fits). `%Dev` is the percent deviance explained (relative to the null deviance).

Value

The matrix above is silently returned

References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.

See Also

`grpnet`, `predict`, `plot` and `coef` methods.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = grpnet(x, glm.gaussian(y))
print(fit1)
```

`set_configs` *Set configuration settings.*

Description

Set configuration settings.

Usage

```
set_configs(name, value = NULL)
```

Arguments

`name`	Configuration variable name.
`value`	Value to assign to the configuration variable.

Value

Assigned value.

Examples

```
set_configs("hessian_min", 1e-6)
set_configs("hessian_min")
```

Index