# Package 'hal9001'

May 8, 2026

**Title** The Scalable Highly Adaptive Lasso

**Version** 0.4.6

**Description** A scalable implementation of the highly adaptive lasso algorithm, including routines for constructing sparse matrices of basis functions of the observed data, as well as a custom implementation of Lasso regression tailored to enhance efficiency when the matrix of predictors is composed exclusively of indicator functions. For ease of use and increased flexibility, the Lasso fitting routines invoke code from the 'glmnet' package by default. The highly adaptive lasso was first formulated and described by MJ van der Laan (2017) <doi:10.1515/ijb-2015-0097>, with practical demonstrations of its performance given by Benkeser and van der Laan (2016) <doi:10.1109/DSAA.2016.93>. This implementation of the highly adaptive lasso algorithm was described by Hejazi, Coyle, and van der Laan (2020) <doi:10.21105/joss.02526>.

**Depends** R (>= 3.1.0), Rcpp

**License** GPL-3

**URL** https://github.com/tlverse/hal9001

**BugReports** https://github.com/tlverse/hal9001/issues

**Encoding** UTF-8

**LazyData** true

**Imports** Matrix, stats, utils, methods, assertthat, origami (>= 1.0.3), glmnet, data.table, stringr

**Suggests** testthat, knitr, rmarkdown, microbenchmark, future, ggplot2, dplyr, tidyr, survival, SuperLearner

**LinkingTo** Rcpp, RcppEigen

**VignetteBuilder** knitr

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Author** Jeremy Coyle [aut, cre] (ORCID: <https://orcid.org/0000-0002-9874-6649>), Nima Hejazi [aut] (ORCID: <https://orcid.org/0000-0002-7127-2789>), Rachael Phillips [aut] (ORCID: <https://orcid.org/0000-0002-8474-591X>),

Lars van der Laan [aut],
David Benkeser [ctb] (ORCID: <https://orcid.org/0000-0002-1019-8343>),
Oleg Sofrygin [ctb],
Weixin Cai [ctb] (ORCID: <https://orcid.org/0000-0003-2680-3066>),
Mark van der Laan [aut, cph, ths] (ORCID:
<https://orcid.org/0000-0003-1432-5511>)

**Maintainer** Jeremy Coyle <jeremyrcoyle@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-11-14 15:00:02 UTC

# Contents

## +.formula_hal9001 *HAL Formula addition: Adding formula term object together into a single formula object term.*

### Description

HAL Formula addition: Adding formula term object together into a single formula object term.

### Usage

```
## S3 method for class 'formula_hal9001'
x + y
```

### Arguments

| | |
|---|---|
| x | A formula_hal9001 object as outputted by h. |
| y | A formula_hal9001 object as outputted by h. |

## apply_copy_map *Apply copy map*

### Description

OR duplicate training set columns together

### Usage

```
apply_copy_map(X, copy_map)
```

### Arguments

| | |
|---|---|
| X | Sparse matrix containing columns of indicator functions. |
| copy_map | the copy map |

### Value

A dgCMatrix sparse matrix corresponding to the design matrix for a zero-th order highly adaptive lasso, but with all duplicated columns (basis functions) removed.

## Examples

```
gendata <- function(n) {
  W1 <- runif(n, -3, 3)
  W2 <- rnorm(n)
  W3 <- runif(n)
  W4 <- rnorm(n)
  g0 <- plogis(0.5 * (-0.8 * W1 + 0.39 * W2 + 0.08 * W3 - 0.12 * W4))
  A <- rbinom(n, 1, g0)
  Q0 <- plogis(0.15 * (2 * A + 2 * A * W1 + 6 * A * W3 * W4 - 3))
  Y <- rbinom(n, 1, Q0)
  data.frame(A, W1, W2, W3, W4, Y)
}
set.seed(1234)
data <- gendata(100)
covars <- setdiff(names(data), "Y")
X <- as.matrix(data[, covars, drop = FALSE])
basis_list <- enumerate_basis(X)
x_basis <- make_design_matrix(X, basis_list)
copy_map <- make_copy_map(x_basis)
x_basis_uniq <- apply_copy_map(x_basis, copy_map)
```

---

`as_dgCMatrix` *Fast Coercion to Sparse Matrix*

---

## Description

Fast and efficient coercion of standard matrix objects to sparse matrices. Borrowed from http://gallery.rcpp.org/articles/sparse-matrix-coercion/. INTERNAL USE ONLY.

## Usage

```
as_dgCMatrix(XX_)
```

## Arguments

| | |
|---|---|
| `XX_` | An object of class `Matrix` that has a sparse structure suitable for coercion to a sparse matrix format of `dgCMatrix`. |

## Value

An object of class `dgCMatrix`, coerced from input `XX_`.

basis_list_cols *List Basis Functions*

## Description

Build a list of basis functions from a set of columns

## Usage

```
basis_list_cols(
  cols,
  x,
  smoothness_orders,
  include_zero_order,
  include_lower_order = FALSE
)
```

## Arguments

| | |
|---|---|
| cols | Index or indices (as numeric) of covariates (columns) of interest in the data matrix x for which basis functions ought to be generated. Note that basis functions for interactions of these columns are computed automatically. |
| x | A matrix containing observations in the rows and covariates in the columns. Basis functions are computed for these covariates. |
| smoothness_orders | An integer vector of length ncol(x) specifying the desired smoothness of the function in each covariate. k = 0 is no smoothness (indicator basis), k = 1 is first order smoothness, and so on. For an additive model, the component function for each covariate will have the degree of smoothness as specified by smoothness_orders. For non-additive components (tensor products of univariate basis functions), the univariate basis functions in each tensor product have smoothness degree as specified by smoothness_orders. |
| include_zero_order | A logical, indicating whether the zeroth order basis functions are included for each covariate (if TRUE), in addition to the smooth basis functions given by smoothness_orders. This allows the algorithm to data-adaptively choose the appropriate degree of smoothness. |
| include_lower_order | A logical, like include_zero_order, except including all basis functions of lower smoothness degrees than specified via smoothness_orders. |

## Value

A list containing the basis functions generated from a set of input columns.

basis_of_degree *Compute Degree of Basis Functions*

## Description

Find the full list of basis functions up to a particular degree

## Usage

```
basis_of_degree(
  x,
  degree,
  smoothness_orders,
  include_zero_order,
  include_lower_order
)
```

## Arguments

| | |
|---|---|
| x | An input matrix containing observations and covariates following standard conventions in problems of statistical learning. |
| degree | The highest order of interaction terms for which the basis functions ought to be generated. The default (NULL) corresponds to generating basis functions for the full dimensionality of the input matrix. |
| smoothness_orders | An integer vector of length ncol(x) specifying the desired smoothness of the function in each covariate. k = 0 is no smoothness (indicator basis), k = 1 is first order smoothness, and so on. For an additive model, the component function for each covariate will have the degree of smoothness as specified by smoothness_orders. For non-additive components (tensor products of univariate basis functions), the univariate basis functions in each tensor product have smoothness degree as specified by smoothness_orders. |
| include_zero_order | A logical, indicating whether the zeroth order basis functions are included for each covariate (if TRUE), in addition to the smooth basis functions given by smoothness_orders. This allows the algorithm to data-adaptively choose the appropriate degree of smoothness. |
| include_lower_order | A logical, like include_zero_order, except including all basis functions of lower smoothness degrees than specified via smoothness_orders. |

## Value

A list containing basis functions and cutoffs generated from a set of input columns up to a particular pre-specified degree.

enumerate_basis *Enumerate Basis Functions*

## Description

Generate basis functions for all covariates and interaction terms thereof up to a specified order/degree.

## Usage

```
enumerate_basis(
  x,
  max_degree = NULL,
  smoothness_orders = rep(0, ncol(x)),
  include_zero_order = FALSE,
  include_lower_order = FALSE,
  num_knots = NULL
)
```

## Arguments

| | |
|---|---|
| x | An input matrix containing observations and covariates following standard conventions in problems of statistical learning. |
| max_degree | The highest order of interaction terms for which the basis functions ought to be generated. The default (NULL) corresponds to generating basis functions for the full dimensionality of the input matrix. |
| smoothness_orders | An integer vector of length ncol(x) specifying the desired smoothness of the function in each covariate. k = 0 is no smoothness (indicator basis), k = 1 is first order smoothness, and so on. For an additive model, the component function for each covariate will have the degree of smoothness as specified by smoothness_orders. For non-additive components (tensor products of univariate basis functions), the univariate basis functions in each tensor product have smoothness degree as specified by smoothness_orders. |
| include_zero_order | A logical, indicating whether the zeroth order basis functions are included for each covariate (if TRUE), in addition to the smooth basis functions given by smoothness_orders. This allows the algorithm to data-adaptively choose the appropriate degree of smoothness. |
| include_lower_order | A logical, like include_zero_order, except including all basis functions of lower smoothness degrees than specified via smoothness_orders. |
| num_knots | A vector of length max_degree, which determines how granular the knot points to generate basis functions should be for each degree of basis function. The first entry of num_knots determines the number of knot points to be used for each univariate basis function. More generally, The kth entry of num_knots determines the number of knot points to be used for the kth degree basis functions. |

Specifically, for a kth degree basis function, which is the tensor product of k univariate basis functions, this determines the number of knot points to be used for each univariate basis function in the tensor product.

## Value

A `list` of basis functions generated for all covariates and interaction thereof up to a pre-specified degree.

## Examples

```
gendata <- function(n) {
  W1 <- runif(n, -3, 3)
  W2 <- rnorm(n)
  W3 <- runif(n)
  W4 <- rnorm(n)
  g0 <- plogis(0.5 * (-0.8 * W1 + 0.39 * W2 + 0.08 * W3 - 0.12 * W4))
  A <- rbinom(n, 1, g0)
  Q0 <- plogis(0.15 * (2 * A + 2 * A * W1 + 6 * A * W3 * W4 - 3))
  Y <- rbinom(n, 1, Q0)
  data.frame(A, W1, W2, W3, W4, Y)
}
set.seed(1234)
data <- gendata(100)
covars <- setdiff(names(data), "Y")
X <- as.matrix(data[, covars, drop = FALSE])
basis_list <- enumerate_basis(X)
```

---

# `evaluate_basis` *Generate Basis Functions*

---

## Description

Populates a column (indexed by basis_col) of x_basis with basis indicators.

## Usage

```
evaluate_basis(basis, X, x_basis, basis_col)
```

## Arguments

| | |
|---|---|
| `basis` | The basis function. |
| `X` | The design matrix, containing the original data. |
| `x_basis` | The HAL design matrix, containing indicator functions. |
| `basis_col` | Numeric indicating which column to populate. |

fit_hal *HAL: The Highly Adaptive Lasso*

## Description

Estimation procedure for HAL, the Highly Adaptive Lasso

## Usage

```
fit_hal(
  X,
  Y,
  formula = NULL,
  X_unpenalized = NULL,
  max_degree = ifelse(ncol(X) >= 20, 2, 3),
  smoothness_orders = 1,
  num_knots = num_knots_generator(max_degree = max_degree, smoothness_orders =
    smoothness_orders, base_num_knots_0 = 200, base_num_knots_1 = 50),
  reduce_basis = NULL,
  family = c("gaussian", "binomial", "poisson", "cox", "mgaussian"),
  lambda = NULL,
  id = NULL,
  weights = NULL,
  offset = NULL,
  fit_control = list(cv_select = TRUE, use_min = TRUE, lambda.min.ratio = 1e-04,
    prediction_bounds = "default"),
  basis_list = NULL,
  return_lasso = TRUE,
  return_x_basis = FALSE,
  yolo = FALSE
)
```

## Arguments

| | |
|---|---|
| X | An input matrix with dimensions number of observations -by- number of covariates that will be used to derive the design matrix of basis functions. |
| Y | A numeric vector of observations of the outcome variable. For family="mgaussian", Y is a matrix of observations of the outcome variables. |
| formula | A character string formula to be used in formula_hal. See its documentation for details. |
| X_unpenalized | An input matrix with the same number of rows as X, for which no L1 penalization will be performed. Note that X_unpenalized is directly appended to the design matrix; no basis expansion is performed on X_unpenalized. |
| max_degree | The highest order of interaction terms for which basis functions ought to be generated. |

smoothness_orders
: An integer, specifying the smoothness of the basis functions. See details for smoothness_orders for more information.

num_knots
: An integer vector of length 1 or max_degree, specifying the maximum number of knot points (i.e., bins) for any covariate for generating basis functions. If num_knots is a unit-length vector, then the same num_knots are used for each degree (this is not recommended). The default settings for num_knots are recommended, and these defaults decrease num_knots with increasing max_degree and smoothness_orders, which prevents (expensive) combinatorial explosions in the number of higher-degree and higher-order basis functions generated. This allows the complexity of the optimization problem to grow scalably. See details of num_knots more information.

reduce_basis
: Am optional numeric value bounded in the open unit interval indicating the minimum proportion of 1's in a basis function column needed for the basis function to be included in the procedure to fit the lasso. Any basis functions with a lower proportion of 1's than the cutoff will be removed. Defaults to 1 over the square root of the number of observations. Only applicable for models fit with zero-order splines, i.e. smoothness_orders = 0.

family
: A character or a family object (supported by glmnet) specifying the error/link family for a generalized linear model. character options are limited to "gaussian" for fitting a standard penalized linear model, "binomial" for penalized logistic regression, "poisson" for penalized Poisson regression, "cox" for a penalized proportional hazards model, and "mgaussian" for multivariate penalized linear model. Note that passing in family objects leads to slower performance relative to passing in a character family (if supported). For example, one should set family = "binomial" instead of family = binomial() when calling fit_hal.

lambda
: User-specified sequence of values of the regularization parameter for the lasso L1 regression. If NULL, the default sequence in cv.glmnet will be used. The cross-validated optimal value of this regularization parameter will be selected with cv.glmnet. If fit_control's cv_select argument is set to FALSE, then the lasso model will be fit via glmnet, and regularized coefficient values for each lambda in the input array will be returned.

id
: A vector of ID values that is used to generate cross-validation folds for cv.glmnet. This argument is ignored when fit_control's cv_select argument is FALSE.

weights
: observation weights; defaults to 1 per observation.

offset
: a vector of offset values, used in fitting.

fit_control
: List of arguments, including the following, and any others to be passed to cv.glmnet or glmnet.

  - cv_select: A logical specifying if the sequence of specified lambda values should be passed to cv.glmnet in order for a single, optimal value of lambda to be selected according to cross-validation. When cv_select = FALSE, a glmnet model will be used to fit the sequence of (or single) lambda.
  - use_min: Specify the choice of lambda to be selected by cv.glmnet. When TRUE, "lambda.min" is used; otherwise, "lambda.1se". Only used when cv_select = TRUE.

- `lambda.min.ratio`: A `glmnet` argument specifying the smallest value for `lambda`, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). We've seen that not setting `lambda.min.ratio` can lead to no `lambda` values that fit the data sufficiently well.
- `prediction_bounds`: An optional vector of size two that provides the lower and upper bounds predictions; not used when `family = "cox"`. When `prediction_bounds = "default"`, the predictions are bounded between `min(Y) - sd(Y)` and `max(Y) + sd(Y)` for each outcome (when `family = "mgaussian"`, each outcome can have different bounds). Bounding ensures that there is no extrapolation.

`basis_list` The full set of basis functions generated from `X`.

`return_lasso` A `logical` indicating whether or not to return the `glmnet` fit object of the lasso model.

`return_x_basis` A `logical` indicating whether or not to return the matrix of (possibly reduced) basis functions used in `fit_hal`.

`yolo` A `logical` indicating whether to print one of a curated selection of quotes from the HAL9000 computer, from the critically acclaimed epic science-fiction film "2001: A Space Odyssey" (1968).

## Details

The procedure uses a custom C++ implementation to generate a design matrix of spline basis functions of covariates and interactions of covariates. The lasso regression is fit to this design matrix via `cv.glmnet` or a custom implementation derived from **origami**. The maximum dimension of the design matrix is $n$ -by- $(n * 2^(d - 1))$, where where $n$ is the number of observations and $d$ is the number of covariates.

For `smoothness_orders = 0`, only zero-order splines (piece-wise constant) are generated, which assume the true regression function has no smoothness or continuity. When `smoothness_orders = 1`, first-order splines (piece-wise linear) are generated, which assume continuity of the true regression function. When `smoothness_orders = 2`, second-order splines (piece-wise quadratic and linear terms) are generated, which assume a the true regression function has a single order of differentiability.

`num_knots` argument specifies the number of knot points for each covariate and for each `max_degree`. Fewer knot points can significantly decrease runtime, but might be overly simplistic. When considering `smoothness_orders = 0`, too few knot points (e.g., < 50) can significantly reduce performance. When `smoothness_orders = 1` or higher, then fewer knot points (e.g., 10-30) is actually better for performance. We recommend specifying `num_knots` with respect to `smoothness_orders`, and as a vector of length `max_degree` with values decreasing exponentially. This prevents combinatorial explosions in the number of higher-degree basis functions generated. The default behavior of `num_knots` follows this logic — for `smoothness_orders = 0`, `num_knots` is set to $500/2^{j-1}$, and for `smoothness_orders = 1` or higher, `num_knots` is set to $200/2^{j-1}$, where $j$ is the interaction degree. We also include some other suitable settings for `num_knots` below, all of which are less complex than default `num_knots` and will thus result in a faster runtime:

- Some good settings for little to no cost in performance:
  - If `smoothness_orders = 0` and `max_degree = 3`, `num_knots = c(400, 200, 100)`.

  - If smoothness_orders = 1+ and max_degree = 3, num_knots = c(100, 75, 50).
- Recommended settings for fairly fast runtime:
  - If smoothness_orders = 0 and max_degree = 3, num_knots = c(200, 100, 50).
  - If smoothness_orders = 1+ and max_degree = 3, num_knots = c(50, 25, 15).
- Recommended settings for fast runtime:
  - If smoothness_orders = 0 and max_degree = 3, num_knots = c(100, 50, 25).
  - If smoothness_orders = 1+ and max_degree = 3, num_knots = c(40, 15, 10).
- Recommended settings for very fast runtime:
  - If smoothness_orders = 0 and max_degree = 3, num_knots = c(50, 25, 10).
  - If smoothness_orders = 1+ and max_degree = 3, num_knots = c(25, 10, 5).

## Value

Object of class hal9001, containing a list of basis functions, a copy map, coefficients estimated for basis functions, and timing results (for assessing computational efficiency).

## Examples

```
n <- 100
p <- 3
x <- xmat <- matrix(rnorm(n * p), n, p)
y_prob <- plogis(3 * sin(x[, 1]) + sin(x[, 2]))
y <- rbinom(n = n, size = 1, prob = y_prob)
hal_fit <- fit_hal(X = x, Y = y, family = "binomial")
preds <- predict(hal_fit, new_data = x)
```

---

# formula_hal *HAL Formula: Convert formula or string to* formula_HAL *object.*

---

## Description

HAL Formula: Convert formula or string to formula_HAL object.

## Usage

```
formula_hal(formula, smoothness_orders, num_knots, X = NULL)
```

## Arguments

| | |
|---|---|
| formula | A formula_hal9001 object as outputted by h. |
| smoothness_orders | A default value for s if not provided explicitly to the function h. |
| num_knots | A default value for k if not provided explicitly to the function h. |
| X | Controls inheritance of the variable X from parent environment. When NULL (the default), such a variable is inherited. |

h *HAL Formula term: Generate a single term of the HAL basis*

## Description

HAL Formula term: Generate a single term of the HAL basis

## Usage

```
h(
  ...,
  k = NULL,
  s = NULL,
  pf = 1,
  monotone = c("none", "i", "d"),
  . = NULL,
  dot_args_as_string = FALSE,
  X = NULL
)
```

## Arguments

| | |
|---|---|
| `...` | Variables for which to generate multivariate interaction basis function where the variables can be found in a matrix `X` in a parent environment/frame. Note, just like standard `formula` objects, the variables should not be characters (e.g. do h(W1,W2) not h("W1", "W2")) h(W1,W2,W3) will generate three-way HAL basis functions between W1, W2, and W3. It will `not` generate the lower dimensional basis functions. |
| `k` | The number of knots for each univariate basis function used to generate the tensor product basis functions. If a single value then this value is used for the univariate basis functions for each variable. Otherwise, this should be a variable named list that specifies for each variable how many knots points should be used. `h(W1,W2,W3, k = list(W1 = 3, W2 = 2, W3=1))` is equivalent to first binning the variables `W1`, `W2` and `W3` into `3`, `2` and `1` unique values and then calling `h(W1,W2,W3)`. This coarsening of the data ensures that fewer basis functions are generated, which can lead to substantial computational speed-ups. If not provided and the variable `num_knots` is in the parent environment, then `s` will be set to `num_knots`‘. |
| `s` | The `smoothness_orders` for the basis functions. The possible values are `0` for piece-wise constant zero-order splines or `1` for piece-wise linear first-order splines. If not provided and the variable `smoothness_orders` is in the parent environment, then `s` will be set to `smoothness_orders`. |
| `pf` | A `penalty.factor` value the generated basis functions that is used by `glmnet` in the LASSO penalization procedure. `pf = 1` (default) is the standard penalization factor used by `glmnet` and `pf = 0` means the generated basis functions are unpenalized. |

monotone
: Whether the basis functions should enforce monotonicity of the interaction term. If \code{s} = 0, this is monotonicity of the function, and, if \code{s} = 1, this is monotonicity of its derivative (e.g., enforcing a convex fit). Set "none" for no constraints, "i" for a monotone increasing constraint, and "d" for a monotone decreasing constraint. Using "i" constrains the basis functions to have positive coefficients in the fit, and "d" constrains the basis functions to have negative coefficients.

.
: Just like with formula, . as in h(.) or h(.,.) is treated as a wildcard variable that generates terms using all variables in the data. The argument . should be a character vector of variable names that . iterates over. Specifically, h(., k=1, . = c("W1", "W2", "W3")) is equivalent to h(W1, k=1) + h(W2, k=1) + h(W3, k=1), and h(., ., k=1, . = c("W1", "W2", "W3")) is equivalent to h(W1,W2, k=1) + h(W2,W3, k=1) + h(W1, W3, k=1)

dot_args_as_string
: Whether the arguments ... are characters or character vectors and should thus be evaluated directly. When TRUE, the expression h("W1", "W2") can be used.

X
: An optional design matrix where the variables given in ... can be found. Otherwise, X is taken from the parent environment.

---

## hal9000 *HAL 9000 Quotes*

---

### Description

Prints a quote from the HAL 9000 robot from 2001: A Space Odyssey

### Usage

```
hal9000()
```

---

## hal9001 *hal9001*

---

### Description

Package for fitting the Highly Adaptive LASSO (HAL) estimator

## hal_quotes *HAL9000 Quotes from "2001: A Space Odyssey"*

### Description

Curated selection of quotes from the HAL9000 computer, from the critically acclaimed epic science-fiction film "2001: A Space Odyssey" (1968).

### Usage

```
hal_quotes
```

### Format

A vector of quotes.

## index_first_copy *Find Copies of Columns*

### Description

Index vector that, for each column in X, indicates the index of the first copy of that column

### Usage

```
index_first_copy(X)
```

### Arguments

| | |
|---|---|
| X | Sparse matrix containing columns of indicator functions. |

## make_basis_list *Sort Basis Functions*

### Description

Build a sorted list of unique basis functions based on columns, where each basis function is a list

### Usage

```
make_basis_list(X_sub, cols, order_map)
```

### Arguments

| | |
|---|---|
| X_sub | A subset of the columns of X, the original design matrix. |
| cols | An index of the columns that were reduced to by sub-setting. |
| order_map | A vector with length the original unsubsetted matrix X which specifies the smoothness of the function in each covariate. |

### Details

Note that sorting of columns is performed such that the basis order equals cols.length() and each basis function is a list(cols, cutoffs).

---

## make_copy_map *Build Copy Maps*

---

### Description

Build Copy Maps

### Usage

```
make_copy_map(x_basis)
```

### Arguments

| | |
|---|---|
| x_basis | A design matrix consisting of basis (indicator) functions for covariates (X) and terms for interactions thereof. |

### Value

A `list` of `numeric` vectors indicating indices of basis functions that are identical in the training set.

### Examples

```
gendata <- function(n) {
  W1 <- runif(n, -3, 3)
  W2 <- rnorm(n)
  W3 <- runif(n)
  W4 <- rnorm(n)
  g0 <- plogis(0.5 * (-0.8 * W1 + 0.39 * W2 + 0.08 * W3 - 0.12 * W4))
  A <- rbinom(n, 1, g0)
  Q0 <- plogis(0.15 * (2 * A + 2 * A * W1 + 6 * A * W3 * W4 - 3))
  Y <- rbinom(n, 1, Q0)
  data.frame(A, W1, W2, W3, W4, Y)
}
set.seed(1234)
data <- gendata(100)
covars <- setdiff(names(data), "Y")
X <- as.matrix(data[, covars, drop = FALSE])
```

```
basis_list <- enumerate_basis(X)
x_basis <- make_design_matrix(X, basis_list)
copy_map <- make_copy_map(x_basis)
```

---

`make_design_matrix` *Build HAL Design Matrix*

---

## Description

Make a HAL design matrix based on original design matrix X and a list of basis functions in argument blist

## Usage

```
make_design_matrix(X, blist, p_reserve = 0.5)
```

## Arguments

| | |
|---|---|
| `X` | Matrix of covariates containing observed data in the columns. |
| `blist` | List of basis functions with which to build HAL design matrix. |
| `p_reserve` | Sparse matrix pre-allocation proportion. Default value is 0.5. If one expects a dense HAL design matrix, it is useful to set p_reserve to a higher value. |

## Value

A `dgCMatrix` sparse matrix of indicator basis functions corresponding to the design matrix in a zero-order highly adaptive lasso.

## Examples

```
gendata <- function(n) {
  W1 <- runif(n, -3, 3)
  W2 <- rnorm(n)
  W3 <- runif(n)
  W4 <- rnorm(n)
  g0 <- plogis(0.5 * (-0.8 * W1 + 0.39 * W2 + 0.08 * W3 - 0.12 * W4))
  A <- rbinom(n, 1, g0)
  Q0 <- plogis(0.15 * (2 * A + 2 * A * W1 + 6 * A * W3 * W4 - 3))
  Y <- rbinom(n, 1, Q0)
  data.frame(A, W1, W2, W3, W4, Y)
}
set.seed(1234)
data <- gendata(100)
covars <- setdiff(names(data), "Y")
X <- as.matrix(data[, covars, drop = FALSE])
basis_list <- enumerate_basis(X)
x_basis <- make_design_matrix(X, basis_list)
```

## make_reduced_basis_map *Mass-based reduction of basis functions*

### Description

A helper function that finds which basis functions to keep (and equivalently which to discard) based on the proportion of 1's (observations, i.e., "mass") included in a given basis function.

### Usage

```
make_reduced_basis_map(x_basis, reduce_basis_crit)
```

### Arguments

| | |
|---|---|
| x_basis | A matrix of basis functions with all redundant basis functions already removed. |
| reduce_basis_crit | A scalar numeric value bounded in the open interval (0,1) indicating the minimum proportion of 1's in a basis function column needed for the basis function to be included in the procedure to fit the Lasso. Any basis functions with a lower proportion of 1's than the specified cutoff will be removed. This argument defaults to NULL, in which case all basis functions are used in the lasso-fitting stage of the HAL algorithm. |

### Value

A binary numeric vector indicating which columns of the matrix of basis functions to keep (given a one) and which to discard (given a zero).

## meets_basis *Compute Values of Basis Functions*

### Description

Computes and returns the indicator value for the basis described by cols and cutoffs for a given row of X

### Usage

```
meets_basis(X, row_num, cols, cutoffs, orders)
```

### Arguments

| | |
|---|---|
| X | The design matrix, containing the original data. |
| row_num | Numeri for a row index over which to evaluate. |
| cols | Numeric for the column indices of the basis function. |
| cutoffs | Numeric providing thresholds. |
| orders | Numeric providing smoothness orders |

---

## predict.hal9001 *Prediction from HAL fits*

---

### Description

Prediction from HAL fits

### Usage

```
## S3 method for class 'hal9001'
predict(
  object,
  new_data,
  new_X_unpenalized = NULL,
  offset = NULL,
  type = c("response", "link"),
  ...
)
```

### Arguments

| | |
|---|---|
| object | An object of class hal9001, containing the results of fitting the Highly Adaptive Lasso, as produced by fit_hal. |
| new_data | A matrix or data.frame containing new data (i.e., observations not used for fitting the hal9001 object that's passed in via the object argument) for which the hal9001 object will compute predicted values. |
| new_X_unpenalized | If the user supplied X_unpenalized during training, then user should also supply this matrix with the same number of observations as new_data. |
| offset | A vector of offsets. Must be provided if provided at training. |
| type | Either "response" for predictions of the response, or "link" for un-transformed predictions (on the scale of the link function). |
| ... | Additional arguments passed to predict as necessary. |

### Details

Method for computing and extracting predictions from fits of the Highly Adaptive Lasso estimator, returned as a single S3 objects of class hal9001.

## Value

A numeric vector of predictions from a hal9001 object.

## Note

This prediction method does not function similarly to the equivalent method from **glmnet**. In particular, this procedure will not return a subset of lambdas originally specified in calling fit_hal nor result in re-fitting. Instead, it will return predictions for all of the lambdas specified in the call to fit_hal that constructs object, when fit_control's cv_select is set to FALSE. When fit_control's cv_select is set to TRUE, predictions will only be returned for the value of lambda selected by cross-validation.

---

# predict.SL.hal9001 *predict.SL.hal9001*

---

## Description

Predict method for objects of class SL.hal9001

## Usage

```
## S3 method for class 'SL.hal9001'
predict(object, newdata, ...)
```

## Arguments

| | |
|---|---|
| object | A fitted object of class hal9001. |
| newdata | A matrix of new observations on which to obtain predictions. |
| ... | Not used. |

## Value

A numeric vector of predictions from a SL.hal9001 object based on the provide newdata.

## print.formula_hal9001 *Print formula_hal9001 object*

### Description

Print formula_hal9001 object

### Usage

```
## S3 method for class 'formula_hal9001'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | A formula_hal9001 object. |
| ... | Other arguments (ignored). |

## print.summary.hal9001 *Print Method for Summary Class of HAL fits*

### Description

Print Method for Summary Class of HAL fits

### Usage

```
## S3 method for class 'summary.hal9001'
print(x, length = NULL, ...)
```

### Arguments

| | |
|---|---|
| x | An object of class summary.hal9001. |
| length | The number of ranked coefficients to be summarized. |
| ... | Other arguments (ignored). |

SL.hal9001 *Wrapper for Classic SuperLearner*

## Description

Wrapper for **SuperLearner** for objects of class hal9001

## Usage

```
SL.hal9001(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  max_degree = 2,
  smoothness_orders = 1,
  num_knots = 5,
  ...
)
```

## Arguments

| | |
|---|---|
| Y | A numeric vector of observations of the outcome variable. |
| X | An input matrix with dimensions number of observations -by- number of covariates that will be used to derive the design matrix of basis functions. |
| newX | A matrix of new observations on which to obtain predictions. The default of NULL computes predictions on training inputs X. |
| family | A family object (one that is supported by glmnet) specifying the error/link family for a generalized linear model. |
| obsWeights | A numeric vector of observational-level weights. |
| id | A numeric vector of IDs. |
| max_degree | The highest order of interaction terms for which basis functions ought to be generated. |
| smoothness_orders | An integer vector of length 1 or greater, specifying the smoothness of the basis functions. See the argument smoothness_orders of fit_hal for more information. |
| num_knots | An integer vector of length 1 or max_degree, specifying the maximum number of knot points (i.e., bins) for each covariate for generating basis functions. See num_knots argument in fit_hal for more information. |
| ... | Additional arguments to fit_hal. |

## Value

An object of class `SL.hal9001` with a fitted `hal9001` object and corresponding predictions based on the input data.

---

`squash_hal_fit` *Squash HAL objects*

---

## Description

Reduce footprint by dropping basis functions with coefficients of zero

## Usage

```
squash_hal_fit(object)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `hal9001`, containing the results of fitting the Highly Adaptive LASSO, as produced by a call to `fit_hal`. |

## Value

Object of class `hal9001`, similar to the input object but reduced such that coefficients belonging to bases with coefficients equal to zero removed.

## Examples

```
# generate simple test data
n <- 100
p <- 3
x <- matrix(rnorm(n * p), n, p)
y <- sin(x[, 1]) * sin(x[, 2]) + rnorm(n, mean = 0, sd = 0.2)

# fit HAL model and squash resulting object to reduce footprint
hal_fit <- fit_hal(X = x, Y = y, yolo = FALSE)
squashed <- squash_hal_fit(hal_fit)
```

`summary.hal9001` *Summary Method for HAL fit objects*

## Description

Summary Method for HAL fit objects

## Usage

```
## S3 method for class 'hal9001'
summary(
  object,
  lambda = NULL,
  only_nonzero_coefs = TRUE,
  include_redundant_terms = FALSE,
  round_cutoffs = 3,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `hal9001`, containing the results of fitting the Highly Adaptive Lasso, as produced by `fit_hal`. |
| `lambda` | Optional `numeric` value of the lambda tuning parameter, for which corresponding coefficient values will be summarized. Defaults to `fit_hal`'s optimal value, `lambda_star`, or the minimum value of `lambda_star`. |
| `only_nonzero_coefs` | A `logical` specifying whether the summary should include only terms with non-zero coefficients. |
| `include_redundant_terms` | A `logical` specifying whether the summary should remove so-called "redundant terms". We define a redundant term (say x1) as a term (1) with basis function corresponding to an existing basis function, a duplicate; and (2) the duplicate contains the x1 term as part of its term, so that x1 terms inclusion would be redundant. For example, say the same coefficient corresponds to these three terms: (1) "I(age >= 50)*I(bmi >= 18)", (2) "I(age >= 50)", and (3) "I(education >= 16)". When `include_redundant_terms` is `FALSE` (default), the second basis function is omitted. |
| `round_cutoffs` | An `integer` indicating the number of decimal places to be used for rounding cutoff values in the term. For example, if "bmi" was numeric that was rounded to the third decimal, in the example above we would have needed to specify `round_cutoffs = 0` in order to yield a term like "I(bmi >= 18)" opposed to something like "I(bmi >= 18.111)". This rounding is intended to simplify the term-wise part of the output and only rounds the basis cutoffs, the `hal9001` model's coefficients are not rounded. |
| `...` | Additional arguments passed to `summary`, not supported. |

## Details

Method for summarizing the coefficients of the Highly Adaptive Lasso estimator in terms of the basis functions corresponding to covariates and interactions of covariates, returned as a single S3 object of class `hal9001`.

Due to the nature of the basis function terms, the summary tables can be extremely wide. The R environment might not be the optimal location to view the summary. Tables can be exported from R to LaTeX with **xtable** package (or similar). Here's an example: `print(xtable(summary(fit)$table, type = "latex"), file = "dt.tex")`.

## Value

A list summarizing a `hal9001` object's coefficients.

# Index