Package: rare (via r-universe)

October 29, 2026

Type Package

Title Linear Model with Tree-Based Lasso Regularization for Rare Features

Version 0.1.2

Description Implementation of an alternating direction method of multipliers algorithm for fitting a linear model with tree-based lasso regularization, which is proposed in Algorithm 1 of Yan and Bien (2020) <doi:10.1080/01621459.2020.1796677>. The package allows efficient model fitting on the entire 2-dimensional regularization path for large datasets. The complete set of functions also makes the entire process of tuning regularization parameters and visualizing results hassle-free.

Depends R (>= 3.2.1)

Imports Matrix, glmnet, Rcpp

Suggests knitr, dendextend, rmarkdown

License GPL-3

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 6.1.0

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/yanxht/rare

BugReports https://github.com/yanxht/rare/issues

NeedsCompilation yes

Author Xiaohan Yan [aut, cre], Jacob Bien [aut]

Maintainer Xiaohan Yan <xy257@cornell.edu>

Repository https://cran.r-universe.dev

Date/Publication 2025-06-16 05:10:02 UTC

RemoteUrl https://github.com/cran/rare

RemoteRef HEAD

RemoteSha fa367f2bfc0f21a73b3f34a534a47092d98ea9e5

Contents

rare-package 2
data.dtm 3
data.hc 3
data.rating 4
find.leaves 4
group.plot 5
group.recover 6
rarefit 7
rarefit.cv 10
rarefit.predict 11
tree.matrix 12

Index **14**

`rare-package` *Model path for tree-based lasso framework for selecting rare features*

Description

The package fits the linear model with tree-based lasso regularization proposed in Yan and Bien (2018) using alternating direction method of multipliers (ADMM). The ADMM algorithm is proposed in Algorithm 1 of the same paper. The package also provides tools for tuning regularization parameters, making predictions from the fitted model and visualizing recovered groups of the covariates in a dendrogram.

Details

Its main functions are `rarefit`, `rarefit.cv`, `rarefit.predict`, `group.recover` and `group.plot`.

Author(s)

Xiaohan Yan <xy257@cornell.edu>, Jacob Bien

References

Yan, X. and Bien, J. (2018) *Rare Feature Selection in High Dimensions*, https://arxiv.org/abs/1803.06675.

data.dtm *Document-term matrix for adjectives in TripAdvisor hotel reviews*

Description

A 500-by-200 document-term matrix for 200 adjectives appearing in 500 TripAdvisor reviews. The document-term matrix is in sparse format.

Usage

```
data.dtm
```

Format

An object of class `dgCMatrix` with 500 rows and 200 columns.

See Also

`data.rating`, `data.hc`.

data.hc *Hierarchical clustering tree for adjectives in TripAdvisor data set*

Description

An `hclust` tree for the 200 adjectives appearing in the TripAdvisor reviews. The tree was generated with 100-dimensional word embeddings pre-trained by GloVe (Pennington et al., 2014) on Gigaword5 and Wikipedia2014 corpora for the adjectives.

Usage

```
data.hc
```

Format

An object of class `hclust` of length 7.

Source

Embeddings available at http://nlp.stanford.edu/data/glove.6B.zip

References

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *In Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

data.rating *TripAdvisor hotel review ratings*

Description

A length-500 TripAdvisor review ratings on the scale 1 to 5.

Usage

```
data.rating
```

Format

An object of class `integer` of length 500.

Source

TripAdvisor Data Set used in https://www.cs.virginia.edu/~hw5x/paper/rp166f-wang.pdf

find.leaves *Find all descendant leaves of a node in an hclust tree*

Description

The function recursively finds all leaves that are descendants of a node in an `hclust` tree.

Usage

```
find.leaves(ind, merge)
```

Arguments

`ind`	Index of the tree node. For an `hclust` tree of `p` leaves, `-j` denotes the `j`th leaf and `k` denotes the interior node formed at the `k`th merging in constructing the tree. The range of `ind` is {-1, ..., -p, 1,..., p-1} where `p-1` is the number of interior nodes.
`merge`	A (`p-1`)-by-2 matrix that encodes the order of mergings in constructing the tree. `merge` uses the same notation for nodes and mergings in an `hclust` object. See `hclust` for details.

Value

Returns a sequence of indices for descendant leaves in the leaf set {1, ..., `p`}. Unlike the notation used in `ind`, we use positive integers to denote leaves here.

Examples

```
## Not run:
hc <- hclust(dist(USArrests), "ave")
# Descendant leaves of the 10th leaf (should be iteself)
find.leaves(-10, hc$merge)

# Descendant leaves of the 10th interior node
find.leaves(10, hc$merge)

# Descendant leaves of the root (should be all leaves)
ind_root <- nrow(hc$merge)
all.equal(find.leaves(ind_root, hc$merge), hc$order)

## End(Not run)
```

`group.plot` *Visualize groups by coloring branches and leaves of an hclust tree*

Description

The function plots an `hclust` tree with branches and leaves colored based on group membership. The groups span the covariate indices {1, ..., `nvars`}. Covariates from the same group share equal coefficient (`beta`), and sibling groups have different coefficients. The function determines groups based on the sparsity in `gamma`. In an `hclust` tree with `beta[i]` on the `i`th leaf, the branch and leaf are colored in blue, red or gray according to `beta[i]` being positive, negative or zero, respectively. The larger the magnitude of `beta[i]` is, the darker the color will be. So branches and leaves from the same group will have the same color.

Usage

```
group.plot(beta, gamma, A, hc, nbreaks = 20)
```

Arguments

`beta`	Length-`nvars` vector of covariate coefficient.
`gamma`	Length-`nnodes` vector of latent variable coefficient. Note that `rarefit` returns `NA` as `gamma` value when `alpha` is zero, in which case our problem becomes the lasso on `beta`.
`A`	`nvars`-by-`nnodes` binary matrix encoding ancestor-descendant relationships between leaves and nodes in the tree.
`hc`	An `hclust` tree of `nvars` leaves where each leaf corresponds to a covariate.
`nbreaks`	Number of breaks in binning `beta` elements (positive part and negative part are done separately). Each bin is associated with a color based on the magnitude and positivity/negativity of `beta` elements in the bin.

Examples

```
## Not run:
# See vignette for more details.
set.seed(100)
ts <- sample(1:length(data.rating), 400) # Train set indices
# Fit the model on train set
ourfit <- rarefit(y = data.rating[ts], X = data.dtm[ts, ], hc = data.hc, lam.min.ratio = 1e-6,
                  nlam = 20, nalpha = 10, rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Cross validation
ourfit.cv <- rarefit.cv(ourfit, y = data.rating[ts], X = data.dtm[ts, ],
                        rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Visualize the groups at optimal beta and gamma
ibest.lambda <- ourfit.cv$ibest[1]
ibest.alpha <- ourfit.cv$ibest[2]
beta.opt <- ourfit$beta[[ibest.alpha]][, ibest.lambda]
gamma.opt <- ourfit$gamma[[ibest.alpha]][, ibest.lambda] # works if ibest.alpha > 1
# Visualize the groups at optimal beta and gamma
group.plot(beta.opt, gamma.opt, ourfit$A, data.hc)

## End(Not run)
```

`group.recover` *Recover aggregated groups of leaf indices*

Description

The function finds aggregated groups of leaf indices by traversing non-zero `gamma` elements and finding descendant leaves at each `gamma` element. In our problem, `gamma` are latent variables corresponding to tree nodes. The order of the traversal is post-order, i.e., a node is visited after its descendants.

Usage

```
group.recover(gamma, A, postorder = seq(ncol(A)))
```

Arguments

`gamma`	Length-`nnodes` latent variable coefficients. Note that `rarefit` returns `NA` as `gamma` value when `alpha` is zero, in which case our problem becomes the lasso on `beta`.
`A`	`nvars`-by-`nnodes` binary matrix encoding ancestor-descendant relationships between leaves and nodes in the tree.
`postorder`	Length-`nnodes` integer vector encoding post-order traversal of the tree nodes such that `seq(nnodes)[postorder]` ensures a node appear after its descendants. Default is `seq(nnodes)`, which gives post-order when `A` is generated using `tree.matrix` for an `hclust` tree.

Value

Returns a list of recovered groups of leaf indices.

Examples

```
## Not run:
# See vignette for more details.
set.seed(100)
ts <- sample(1:length(data.rating), 400) # Train set indices
# Fit the model on train set
ourfit <- rarefit(y = data.rating[ts], X = data.dtm[ts, ], hc = data.hc, lam.min.ratio = 1e-6,
                nlam = 20, nalpha = 10, rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Cross validation
ourfit.cv <- rarefit.cv(ourfit, y = data.rating[ts], X = data.dtm[ts, ],
                        rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Group recovered at optimal beta and gamma
ibest.lambda <- ourfit.cv$ibest[1]
ibest.alpha <- ourfit.cv$ibest[2]
gamma.opt <- ourfit$gamma[[ibest.alpha]][, ibest.lambda] # works if ibest.alpha > 1
groups.opt <- group.recover(gamma.opt, ourfit$A)

## End(Not run)
```

`rarefit` *Fit the rare feature selection model*

Description

Fit the rare feature selection model proposed in Yan and Bien (2018):

$$min_{\beta,\gamma} 0.5 * ||y - X\beta - \beta_0 1_n||_2^2 + \lambda * (\alpha * ||\gamma_{-root}||_1 + (1-\alpha) * ||\beta||_1)$$

using an alternating direction method of multipliers (ADMM) algorithm described in Algorithm 1 of the same paper. The regularization path is computed over a two-dimensional grid of regularization parameters: `lambda` and `alpha`. Of the two, `lambda` controls the overall amount of regularization, and `alpha` controls the tradeoff between sparsity and fusion of β (larger `alpha` induces more fusion in β).

Usage

```
rarefit(y, X, A = NULL, Q = NULL, hc, intercept = T, lambda = NULL,
  alpha = NULL, nlam = 50, lam.min.ratio = 1e-04, nalpha = 10,
  rho = 0.01, eps1 = 1e-06, eps2 = 1e-05, maxite = 1e+06)
```

Arguments

y	Length-nobs response variable.
X	nobs-by-nvars input matrix: each row is an observation vector and each column stores a count covariate.
A	nvars-by-nnodes binary matrix encoding ancestor-descendant relationships between leaves and tree nodes, where nnodes is the total number of tree nodes. A[i,j] is 1 if the ith leaf is a descendant of the jth node in the tree, and 0 otherwise. A should be in sparse matrix format (inherit from class sparseMatrix as in package Matrix). When A is NULL, the function will learn A from hc.
Q	(nvars+nnodes)-by-nnodes matrix with columns forming an orthonormal basis for the null space of $[I_n vars : -A]$. When Q is NULL, the function will learn Q using the singular value decomposition.
hc	An hclust tree of nvars leaves where each leaf corresponds to a covariate. If the tree is not an hclust object, user needs to provide the matrix A instead.
intercept	Whether intercept be fitted (default = TRUE) or set to zero (FALSE).
lambda	A user-supplied lambda sequence. Typical usage is to have the program compute its own lambda sequence based on nlam and lam.min.ratio.
alpha	A user-supplied alpha sequence. If letting the program compute its own alpha sequence, a length-nalpha sequence of equally-spaced alpha values between 0 and 1 will be used. In practice, user may want to provide a more fine alpha sequence to tune the model to its best performance (e.g., alpha = c(1-exp(seq(0, log(1e-2), len = nalpha - 1)), 1)).
nlam	Number of lambda values (default = 50).
lam.min.ratio	Smallest value for lambda, as a fraction of lambda.max (i.e., the smallest value for which all coefficients are zero). The default value is 1e-4.
nalpha	Number of alpha values (default = 10).
rho	Penalty parameter for the quadratic penalty in the ADMM algorithm. The default value is 1e-2.
eps1	Convergence threshold in terms of the absolute tolerance level for the ADMMM algorithm. The default value is 1e-6.
eps2	Convergence threshold in terms of the relative tolerance level for the ADMM algorithm. The default value is 1e-5.
maxite	Maximum number of passes over the data for every pair of (lambda, alpha). The default value is 1e6.

Details

The function splits model fitting path by alpha. At each alpha value, the model is fit on the entire sequence of lambda with warm start. We recommend including an intercept (by setting intercept=T) unless the input data have been centered.

Value

Returns regression coefficients for beta and gamma and intercept beta0. We use a *matrix-nested-within-list* structure to store the coefficients: each list item corresponds to an alpha value; matrix (or vector) in that list item stores coefficients at various lambda values by columns (or entries).

beta0	Length-nalpha list with each item storing intercept across various lambda in a vector: beta0[[j]][i] is intercept fitted at (lambda[i], alpha[j]). If intercept = FALSE, beta0 is NULL.
beta	Length-nalpha list with each item storing beta coefficient at various lambda in columns of a nvars-by-nlam matrix: beta[[j]][, i] is beta coeffcient fitted at (lambda[i], alpha[j]).
gamma	Length-nalpha list with each item storing gamma coefficient at various lambda in columns of a nnodes-by-nlam matrix: gamma[[j]][, i] is gamma coeffcient vector fitted at (lambda[i], alpha[j]). If alpha[j] = 0, the problem becomes the lasso on beta and is solved with glmnet on beta, in which case gamma[[j]] = NA.
lambda	Sequence of lambda values used in model fit.
alpha	Sequence of alpha values used in model fit.
A	Binary matrix encoding ancestor-descendant relationship between leaves and nodes in the tree.
Q	Matrix with columns forming an orthonormal basis for the null space of $[I_n vars : -A]$.
intercept	Whether an intercept is included in model fit.

References

Yan, X. and Bien, J. (2018) *Rare Feature Selection in High Dimensions*, https://arxiv.org/abs/1803.06675.

See Also

rarefit.cv, rarefit.predict

Examples

```
## Not run:
# See vignette for more details.
set.seed(100)
ts <- sample(1:length(data.rating), 400) # Train set indices
# Fit the model on train set
ourfit <- rarefit(y = data.rating[ts], X = data.dtm[ts, ], hc = data.hc, lam.min.ratio = 1e-6,
                  nlam = 20, nalpha = 10, rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)

## End(Not run)
```

`rarefit.cv` *Perform K-fold cross validation*

Description

The function does K-fold cross validaton (CV) to choose an optimal pair of (`lambda`, `alpha`) on which the model performs best according to the chosen error metric: mean squared error or mean absolute error.

Usage

```
rarefit.cv(fitObj, y, X, errtype = "mean-squared-error", nfolds = 5,
  ...)
```

Arguments

`fitObj`	Output of `rarefit`
`y`	Response variable.
`X`	`nobs`-by-`nvars` input matrix: each row is an observation vector and each column stores a count covariate.
`errtype`	Type of error metric used in cross validation. Available choices are *mean-squared-error* (default) and *mean-absolute-error*.
`nfolds`	Number of folds (default is 5)
`...`	Other arguments that can be passed to `rarefit`

Value

`folds`	A length-`nfolds` list with the kth element being elements in the kth fold.
`errs`	A `nlam`-by-`nalpha`-by-`nfolds` 3-dimensional array of errors. `errs[i,j,k]` is error incurred in using `lambda[i]` and `alpha[j]` on the kth fold.
`m`	A `nlam`-by-`nalpha` matrix for storing CV error (i.e., mean error across folds). `m[i,j]` is CV error incurred in using `lambda[i]` and `alpha[j]`.
`se`	A `nlam`-by-`nalpha` matrix for storing standard error across folds. `se[i,j]` is standard error incurred in using `lambda[i]` and `alpha[j]`.
`ibest`	Indices of pair of (`lambda`, `alpha`) minimizing CV error.
`lambda.best`	Value of `lambda` minimizing CV error.
`alpha.best`	Value of `alpha` minimizing CV error.

See Also

`rarefit`, `rarefit.predict`

Examples

```
## Not run:
# See vignette for more details.
set.seed(100)
ts <- sample(1:length(data.rating), 400) # Train set indices
# Fit the model on train set
ourfit <- rarefit(y = data.rating[ts], X = data.dtm[ts, ], hc = data.hc, lam.min.ratio = 1e-6,
                nlam = 20, nalpha = 10, rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Cross validation
ourfit.cv <- rarefit.cv(ourfit, y = data.rating[ts], X = data.dtm[ts, ],
                        rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)

## End(Not run)
```

`rarefit.predict` *Make predictions from a rarefit object and a rarefit.cv object*

Description

The function makes predictions using a `rarefit` object at optimal (`lambda`, `alpha`) chosen by `rarefit.cv`.

Usage

```
rarefit.predict(fitObj, cvObj, newx)
```

Arguments

`fitObj`	Output of `rarefit`.
`cvObj`	Output of `rarefit.cv`.
`newx`	Matrix of new values for x at which predictions are made.

Value

Returns a sequence of predictions.

See Also

`rarefit`, `rarefit.cv`

Examples

```
## Not run:
# See vignette for more details.
set.seed(100)
ts <- sample(1:length(data.rating), 400) # Train set indices
# Fit the model on train set
```

```
ourfit <- rarefit(y = data.rating[ts], X = data.dtm[ts, ], hc = data.hc, lam.min.ratio = 1e-6,
                  nlam = 20, nalpha = 10, rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Cross validation
ourfit.cv <- rarefit.cv(ourfit, y = data.rating[ts], X = data.dtm[ts, ],
                        rho = 0.01, eps1 = 1e-5, eps2 = 1e-5, maxite = 1e4)
# Prediction on test set
pred <- rarefit.predict(ourfit, ourfit.cv, data.dtm[-ts, ])
pred.error <- mean((pred - data.rating[-ts])^2)

## End(Not run)
```

`tree.matrix` *Generate matrix A encoding ancestor-descendant relationships in an hclust tree*

Description

The function generates the binary matrix `A` defined in Yan and Bien (2018). The matrix encodes ancestor-descendant relationships between leaves and tree nodes in an `hclust` tree.

Usage

```
tree.matrix(hc)
```

Arguments

`hc`	An `hclust` object.

Value

Returns a `nvars`-by-`nnodes` binary matrix `A` where `nvars` is the number of leaves (we associate covariate with leaf), and `nnodes` is the number of tree nodes (including both leaves and interior nodes). For an `hclust` tree, `nnodes = 2*nvars-1`. `A[i,j]` is 1 if the `i`th leaf is a descendant of the `j`th node in the tree, and 0 otherwise. *By default, we let the first* `nvars` *columns correspond to leaves and the remaining* `nvars-1` *columns correspond to interior nodes.* `A` is in sparse matrix format (inherit from class `sparseMatrix` as in package `Matrix`).

References

Yan, X. and Bien, J. (2018) *Rare Feature Selection in High Dimensions*, https://arxiv.org/abs/1803.06675.

See Also

`find.leaves` for finding descendant leaves of a node.

Examples

```
## Not run:
# For a perfect binary tree of depth 2 below
#
#      3
#      /\
#    1    2
#   /\    /\
# -1 -2 -3 -4
#
# A can expressed as the following:
A_true <- cbind(diag(4),
                as.matrix(c(1, 1, 0, 0)),
                as.matrix(c(0, 0, 1, 1)),
                as.matrix(c(1, 1, 1, 1)))
# Now use tree.matrix to generate A
tree0 <- list()
tree0$merge <- matrix(c(-1, -2, -3, -4, 1, 2),
                      ncol = 2, byrow = TRUE)
tree0$labels <- c("leaf1", "leaf2", "leaf3", "leaf4")
A <- tree.matrix(tree0)
all(A_true == as.matrix(A))

# Another example
hc <- hclust(dist(USArrests), "ave")
A <- tree.matrix(hc)

## End(Not run)
```

Index

∗ **datasets**
data.dtm, 3
data.hc, 3
data.rating, 4

data.dtm, 3
data.hc, 3, *3*
data.rating, *3*, 4

find.leaves, 4, *12*

glmnet, *9*
group.plot, *2*, 5
group.recover, *2*, 6

hclust, *4*

rare-package, 2
rarefit, *2*, *5*, *6*, 7, *10*, *11*
rarefit.cv, *2*, *9*, 10, *11*
rarefit.predict, *2*, *9*, *10*, 11

sparseMatrix, *8*, *12*

tree.matrix, *6*, 12