Package 'TSLA'

July 21, 2025

Type Package

Title Tree-Guided Rare Feature Selection and Logic Aggregation

Version 0.1.2

Imports stats, Matrix, Rcpp, pROC, PRROC, ape, phytools, data.tree

LinkingTo Rcpp, RcppArmadillo

Maintainer Jianmin Chen <jianminc000@gmail.com>

Description Implementation of the tree-guided feature selection and logic aggregation approach introduced in Chen et al. (2024) <doi:10.1080/01621459.2024.2326621>. The method enables the selection and aggregation of large-scale rare binary features with a known hierarchical structure using a convex, linearly-constrained regularized regression framework. The package facilitates the application of this method to both linear regression and binary classification problems by solving the optimization problem via the smoothing proximal gradient descent algorithm (Chen et al. (2012) <doi:10.1214/11-AOAS514>).

Depends R (>= 4.1.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

NeedsCompilation yes

Author Jianmin Chen [aut, cre],
Kun Chen [aut]

Repository CRAN

Date/Publication 2025-03-17 20:20:05 UTC

Contents

`TSLA-package` *Tree-Guided Rare Feature Selection and Logic Aggregation*

Description

This package provides functions and visualization tools for fitting the Tree-Guided Rare Feature Selection and Logic Aggregation model.

Author(s)

Jianmin Chen <jianminc000@gmail.com>, Kun Chen

`cal2norm` *Calculate group norms*

Description

Function to output group norms on the gamma coefficients based on `g_idx` and `C2` matrix.

Usage

```
cal2norm(u, g_idx, type)
```

Arguments

`u`	`C2*gamma.coef`, `gamma.coef` is the estimated node coefficient vector, `C2` matrix is the output from function `get_tree_object()`, which gives the weights of the groups.
`g_idx`	Group structure matrix defined by the `C2` matrix. See details in `get_tree_object()`.
`type`	If `type == 1`, return sum of group norms; else return individual norm for each group.

Value

Sum of group norms or individual group norms.

ClassificationExample *Synthesic for the classification example*

Description

Synthetic data used to illustrate how to use TSLA with classification.

Usage

```
data(ClassificationExample)
```

Format

List containing the following elements:

tree.org Original tree structure with 42 leaf nodes and 5 different levels.

x.org Original design matrix with 42 binary features and 400 observations.

x1 Unpenalized covariate.

y Continuous response of length 400.

coef_TSLA *Get coefficients from a fitted TSLA model*

Description

Get coefficients from a TSLA.fit object.

Usage

```
coef_TSLA(object, ...)
```

Arguments

object	A fit output from TSLA.fit().
...	Other parameters.

Value

A list of coefficients for each combination of λ and α. The first dimension is indexed by the coefficient, the second dimension is indexed by λ, and the third dimension is indexed by α.

Intercept	Intercept.
cov.coef	Coefficients for unpenalized features.
gamma.coef	Node coefficients. Also see details in getetmat().

beta.coef	Regression coefficients. Each coefficient is multiplied by $(-1)^{r-1}$, where r is the order of the corresponding interaction term. Also see details in getetmat().
beta.coef.adj	Regression coefficients β. This is the common regression coefficient vector. It corresponds to x.expand.adj and A.adj.

---

cv.TSLA *Cross validation for TSLA*

Description

Conduct cross validation to select the optimal tuning parameters in TSLA.

Usage

```
cv.TSLA(
  y,
  X_1 = NULL,
  X_2,
  treemat,
  family = c("ls", "logit"),
  penalty = c("CL2", "RFS-Sum"),
  pred.loss = c("MSE", "AUC", "deviance"),
  gamma.init = NULL,
  weight = NULL,
  nfolds = 5,
  group.weight = NULL,
  feature.weight = NULL,
  control = list(),
  modstr = list()
)
```

Arguments

y	Response in matrix form, continuous for family = "ls" and binary (0/1) for family = "logit".
X_1	Design matrix for unpenalized features (excluding intercept). Need to be in the matrix form.
X_2	Expanded design matrix for penalty = "CL2"; Original design matrix for penalty = "RFS-Sum". Need to be in the matrix form.
treemat	Expanded tree structure in matrix form for penalty = "CL2"; Original structure for penalty = "RFS-Sum".
family	Two options. Use "ls" for least square problems and "logit" for logistic regression problems.
penalty	Two options for group penalty on γ, "CL2" or "RFS-Sum".

pred.loss	Model performance metrics. If family="ls", default is "MSE" (mean squared error). If family="logit", default is "AUC". For logistic model, another option is "deviance".
gamma.init	Initial value for the optimization. Default is a zero vector. The length should equal to 1+ncol(X_1)+ncol(A). See details of A in get_tree_obj().
weight	A vector of length two and it is used for logistic regression only. The first element corresponds to weight of y=1 and the second element corresponds to weight of y=0.
nfolds	Number of cross validation folds. Default is 5.
group.weight	User-defined weights for group penalty. Need to be a vector and the length equals to the number of groups.
feature.weight	User-defined weights for each predictor after expansion.
control	A list of parameters controlling algorithm convergence. Default values: tol = 1e-5, convergence tolerance; maxit = 10000, maximum number of iterations; mu = 1e-3, smoothness parameter in SPG.
modstr	A list of parameters controlling tuning parameters. Default values: lambda = NULL. If lambda is not provided, the package will give a default lambda sequence; lambda.min.ratio = 1e-04, smallest value for lambda as a fraction of lambda.max (given by default when lambda is NULL); nlambda = 50, number of lambda values (equal spacing on log scale) used when lambda is NULL; alpha = seq(0, 1, length.out = 10), sequence of alpha. Here, alpha is tuning parameter for generalized lasso penalty and 1-alpha is the tuning parameter for group lasso penalty.

Value

A list of cross validation results.

lambda.min	λ value with best prediction performance.
alpha.min	α value with best prediction performance.
cvm	A (number-of-lambda * number-of-alpha) matrix saving the means of cross validation loss across folds.
cvsd	A (number-of-lambda * number-of-alpha) matrix saving standard deviations of cross validation loss across folds.
TSLA.fit	Outputs from TSLA.fit().
Intercept.min	Intercept corresponding to (lambda.min,alpha.min).
cov.min	Coefficients of unpenalized features corresponding to (lambda.min,alpha.min).
beta.min	Coefficients of binary features corresponding to (lambda.min,alpha.min).
gamma.min	Node coefficients corresponding to (lambda.min,alpha.min).
groupnorm.min	Group norms of node coefficients corresponding to (lambda.min,alpha.min).
lambda.min.index	Index of the best λ in the sequence.
alpha.min.index	Index of the best α in the sequence.

Examples

```
# Load the synthetic data
data(ClassificationExample)

tree.org <- ClassificationExample$tree.org   # original tree structure
x2.org <- ClassificationExample$x.org      # original design matrix
x1 <- ClassificationExample$x1
y <- ClassificationExample$y            # response

# Do the tree-guided expansion
expand.data <- getetmat(tree.org, x2.org)
x2 <- expand.data$x.expand              # expanded design matrix
tree.expand <- expand.data$tree.expand  # expanded tree structure

# Do train-test split
idtrain <- 1:200
x1.train <- as.matrix(x1[idtrain, ])
x2.train <- x2[idtrain, ]
y.train <- y[idtrain, ]
x1.test <- as.matrix(x1[-idtrain, ])
x2.test <- x2[-idtrain, ]
y.test <- y[-idtrain, ]

# specify some model parameters
set.seed(100)
control <- list(maxit = 100, mu = 1e-3, tol = 1e-5, verbose = FALSE)
modstr <- list(nlambda = 5,  alpha = seq(0, 1, length.out = 5))
simu.cv <- cv.TSLA(y = y.train, as.matrix(x1[idtrain, ]),
                   X_2 = x2.train,
                   treemat = tree.expand, family = 'logit',
                   penalty = 'CL2', pred.loss = 'AUC',
                   gamma.init = NULL, weight = c(1, 1), nfolds = 5,
                   group.weight = NULL, feature.weight = NULL,
                   control = control, modstr =  modstr)
# Do prediction with the selected tuning parameters on the test set. Report AUC on the test set.
rmid <- simu.cv$TSLA.fit$rmid  # remove all zero columns
if(length(rmid) > 0){
  x2.test <- x2.test[, -rmid]}
  y.new <- predict_cvTSLA(simu.cv, as.matrix(x1[-idtrain, ]), x2.test)
  library(pROC)
  auc(as.vector(y.test), as.vector(y.new))
```

`getaggr` *Generate aggregated features*

Description

Function that generates aggregated features based on the TSLA output.

Usage

```
getaggr(TSLA.object, X_2, X_2.org, lambda.index, alpha.index)
```

Arguments

TSLA.object	A fit output from TSLA.fit(), or the TSLA.fit object in cv.TSLA().
X_2	Expanded design matrix in matrix form.
X_2.org	Original design matrix in matrix form.
lambda.index	Index of the λ value selected.
alpha.index	Index of the α value selected. The α is the tuning parameter for generalized lasso penalty.

Value

A data.frame of the aggregated feature.

dataset	aggregated features.

---

getetmat *Tree-guided expansion*

Description

Give the expanded design matrix and the expanded tree structure by adding interactions in conformity to the structure.

Usage

```
getetmat(tmatrix, dmatrix)
```

Arguments

tmatrix	Tree structure of the original features in matrix form.
dmatrix	Original design matrix in matrix form.

Details

This function is used by the TSLA method only when the penalty is selected as "CL2". The all zero columns produced by the interactions are excluded in the output.

For the TSLA method, the signs of the coefficients in the linear constraints depend on the order of the term. To better extend the method in implementation, we apply the signs on the feature vectors instead of the regression coefficients. For example, we use feature vector -x_{12} instead of x_{12}. The expanded design matrix x.expand from this function is adjusted by the signs. The A matrix and all the coefficients estimated from the package can be explained correspondingly. We also provide x.expand.adj, A.adj, and beta.coef.adj as the quantities with the effects of the signs removed.

The input tree structure of the original features needs to be constructed as the following: each row corresponds to a variable at the finest level; each column corresponds to an ordered classification level with the leaf level at the left-most and the root level at the right-most; the entry values in each column are the index of the ancestor node of the variable at that level. As we move from left to right, the number of unique values in the column becomes fewer.

Value

A list.

`x.expand`	The design matrix after expansion. Each column is multiplied by $(-1)^{r-1}$, where r is the order of the corresponding interaction term.
`tree.expand`	The tree structure after expansion.
`x.expand.adj`	The design matrix after expansion with the effects of signs removed.

`getperform` *Get performance metrics for classification*

Description

Evaluate the prediction performance under the classification settings.

Usage

```
getperform(
  ytest,
  ypretest,
  family,
  threshold.method = c("youden", "specificity.control", "quantile"),
  specificity = NULL
)
```

Arguments

`ytest`	Response vector for test data.
`ypretest`	Predicted probability for test data.
`family`	"ls" or "logic". Return MSE when "ls" is used.
`threshold.method`	Method to get the threshold.
`specificity`	User-defined specificity or quantile.

Details

The function supports three methods to select the threshold of the predicted probability.

`threshold.method = "youden"`: The optimal threshold corresponds to the point that maximizes the distance to the identity (diagonal) line on the ROC curve.

`threshold.method = "specificity.control"`: The optimal threshold corresponds to the smallest value that ensures the required specificity value.

`threshold.method = "quantile"`: The optimal threshold corresponds to the required quantile of the predicted probability.

Value

List of measures.

`AUC`	Area under the ROC curve.
`AUPRC`	Area under the precision-recall curve.
`threshold`	Selected threshold of the probability.
`sensitivity`	Sensitivity with the selected threshold.
`ppv`	Positive predictive value with the selected threshold.
`specificity`	Specificity with the selected threshold.
`true.positive`	Number of true positive with the selected threshold.
`false.positive`	Number of false positive with the selected threshold.

`get_tree_object` *Tree-guided reparameterization*

Description

This function generates all the intermediate quantities based on the tree-guided reparameterization.

Usage

```
get_tree_object(
  X_2,
  treemat,
  penalty = c("CL2", "DL2", "RFS-Sum"),
  group.weight = NULL,
  feature.weight = NULL
)
```

Arguments

X_2	Expanded design matrix for penalty = "CL2"; Original design matrix for penalty = "RFS-Sum". Need to be in the matrix form.
treemat	Expanded tree structure for penalty = "CL2"; Original structure for penalty = "RFS-Sum". Need to be in the matrix form.
penalty	Two options for group penalty on γ, "CL2" or "RFS-Sum".
group.weight	User-defined weights for group penalty. Need to be a vector and the length equals to the number of groups.
feature.weight	User-defined weights for each predictor after expansion.

Value

A list consists of quantities needed for SPG optimization.

C_1	C_1 matrix for generalized lasso penalty.
CNorm_1	Nuclear norm of matrix C_1.
C_2	C_2 matrix for group lasso penalty.
CNorm_2	Nuclear norm of matrix C_2.
A	A (number-of-leaf * number-of-node) binary matrix containing linear constraints. Recall that $\beta = A\gamma$. It is used with beta.coef and x.expand.
g_idx	A (number-of-group * 3) matrix. Each column stands for starting row in C_2 of a group, end row in C_2 of a group, and the group size.
M2	A (number-of-leaf * number-of-level) node index matrix, with index going from 1 to the number of nodes. Root node has index equal to the number of nodes. Each row corresponds to a variable at the finest level, each column corresponds to an ordered classification level; the entry values in each column are the unique indices of the variables at that level. As we move to the right, the number of unique values becomes fewer.
Tree	A (number-of-group * number-of-node) group index matrix. Each row is a group and the column order is the same as the order of node index in M2. If the jth node belongs to the ith group, then the (i, j) element of the matrix is 1; otherwise the element is 0.
A.adj	A (number-of-leaf * number-of-node) binary matrix containing linear constraints. It is used with beta.coef.adj and x.expand.adj.

---

plot_TSLA *Plot aggregated structure*

Description

Return a tree plot.

Usage

```
plot_TSLA(TSLA.object, X_2, X_2.org, lambda.index, alpha.index)
```

Arguments

TSLA.object	A fit output from TSLA.fit(), or the TSLA.fit object in cv.TSLA().
X_2	Expanded design matrix in matrix form.
X_2.org	Original design matrix in matrix form.
lambda.index	Index of the λ value selected.
alpha.index	Index of the α value selected. The α is the tuning parameter for generalized lasso penalty.

Value

A plot

Examples

```
# Load the synthetic data
data(ClassificationExample)

tree.org <- ClassificationExample$tree.org   # original tree structure
x2.org <- ClassificationExample$x.org      # original design matrix
x1 <- ClassificationExample$x1
y <- ClassificationExample$y            # response

# Do the tree-guided expansion
expand.data <- getetmat(tree.org, x2.org)
x2 <- expand.data$x.expand              # expanded design matrix
tree.expand <- expand.data$tree.expand  # expanded tree structure

# Do train-test split
idtrain <- 1:200
x1.train <- as.matrix(x1[idtrain, ])
x2.train <- x2[idtrain, ]
y.train <- y[idtrain, ]
x1.test <- as.matrix(x1[-idtrain, ])
x2.test <- x2[-idtrain, ]
y.test <- y[-idtrain, ]

# specify some model parameters
set.seed(100)
control <- list(maxit = 100, mu = 1e-3, tol = 1e-5, verbose = FALSE)
modstr <- list(nlambda = 5,  alpha = seq(0, 1, length.out = 5))
simu.cv <- cv.TSLA(y = y.train, as.matrix(x1[idtrain, ]),
                   X_2 = x2.train,
                   treemat = tree.expand, family = 'logit',
                   penalty = 'CL2', pred.loss = 'AUC',
                   gamma.init = NULL, weight = c(1, 1), nfolds = 5,
                   group.weight = NULL, feature.weight = NULL,
```

```
                    control = control, modstr =  modstr)
plot_TSLA(simu.cv$TSLA.fit, x2, x2.org, simu.cv$lambda.min.index, simu.cv$alpha.min.index)
```

`predict_cvTSLA` *Prediction from cross validation*

Description

A convenient function to get prediction from the selected tuning parameters by cross validation.

Usage

```
predict_cvTSLA(
  object,
  X_1_new = NULL,
  X_2_new,
  type = c("response", "link"),
  ...
)
```

Arguments

`object`	A fit output from `cv.TSLA()`.
`X_1_new`	New unpenalized features in matrix form.
`X_2_new`	New binary features in matrix form.
`type`	Two options: "response" or "link". The two options only differ for `family="logit"`.
`...`	Other parameters.

Value

Predictions.

predict_TSLA *Prediction from TSLA with new data*

Description

Generate prediction for the response.

Usage

```
predict_TSLA(
  object,
  X_1_new = NULL,
  X_2_new,
  type = c("response", "link"),
  ...
)
```

Arguments

object	A fit output from TSLA.fit().
X_1_new	New unpenalized features in matrix form.
X_2_new	New binary features in matrix form.
type	Two options: "response" or "link". The two options only differ for family="logit".
...	Other parameters.

Value

Predictions. The first dimension is indexed by the observation, the second dimension is indexed by λ, and the third dimension is indexed by α.

RegressionExample *Synthesic for the regression example*

Description

Synthetic data used to illustrate how to use TSLA with regression.

Usage

```
data(RegressionExample)
```

Format

List containing the following elements:

tree.org Original tree structure with 42 leaf nodes and 5 different levels.

x.org Original design matrix with 42 binary features and 400 observations.

y Continuous response of length 400.

`TSLA.fit` *Solve the TSLA optimization problem*

Description

Find the solutions with a Smoothing Proximal Gradient (SPG) algorithm for a sequence of α and λ values.

Usage

```
TSLA.fit(
  y,
  X_1 = NULL,
  X_2,
  treemat,
  family = c("ls", "logit"),
  penalty = c("CL2", "RFS-Sum"),
  gamma.init = NULL,
  weight = NULL,
  group.weight = NULL,
  feature.weight = NULL,
  control = list(),
  modstr = list()
)
```

Arguments

`y`	Response in matrix form, continuous for `family = "ls"` and binary (0/1) for `family = "logit"`.
`X_1`	Design matrix for unpenalized features (excluding intercept). Need to be in the matrix form.
`X_2`	Expanded design matrix for `penalty = "CL2"`; Original design matrix for `penalty = "RFS-Sum"`. Need to be in the matrix form.
`treemat`	Expanded tree structure in matrix form for `penalty = "CL2"`; Original structure for `penalty = "RFS-Sum"`.
`family`	Two options. Use "ls" for least square problems and "logit" for logistic regression problems.
`penalty`	Two options for group penalty on γ, "CL2" or "RFS-Sum".

`gamma.init` Initial value for the optimization. Default is a zero vector. The length should equal to 1+`ncol(X_1)`+`ncol(A)`. See details of A in `get_tree_obj()`.

`weight` A vector of length two and it is used for logistic regression only. The first element corresponds to weight of y=1 and the second element corresponds to weight of y=0.

`group.weight` User-defined weights for group penalty. Need to be a vector and the length equals to the number of groups.

`feature.weight` User-defined weights for each predictor after expansion.

`control` A list of parameters controlling algorithm convergence. Default values: `tol` = `1e-5`, convergence tolerance; `maxit` = `10000`, maximum number of iterations; `mu` = `1e-3`, smoothness parameter in SPG.

`modstr` A list of parameters controlling tuning parameters. Default values: `lambda` = `NULL`. If lambda is not provided, the package will give a default lambda sequence; `lambda.min.ratio` = `1e-04`, smallest value for lambda as a fraction of lambda.max (given by default when lambda is NULL); `nlambda` = `50`, number of lambda values (equal spacing on log scale) used when lambda is NULL; `alpha` = `seq(0, 1, length.out = 10)`, sequence of alpha. Here, alpha is tuning parameter for generalized lasso penalty and 1-alpha is the tuning parameter for group lasso penalty.

Details

We adopt the warm start technique to speed up the calculation. The warm start is applied with a fixed value of α and a descending sequence of λ.

The objective function for "ls" is

$$1/2RSS + \lambda(\alpha P(\beta) + (1-\alpha)P(\gamma)),$$

subject to $\beta = A\gamma$. The objective function for "logit" is

$$-loglik + \lambda(\alpha P(\beta) + (1-\alpha)P(\gamma)),$$

subject to $\beta = A\gamma$. Note that, in this package, the input parameter "alpha" is the tuning parameter for the generalized lasso penalty.

Details for "penalty" option:

For `penalty` = `"CL2"`, see details for the "Child-l2" penalty in the main paper.

For `penalty` = `"RFS-Sum"`, the theoretical optimal weights are used. Please check the details in paper "Rare feature selection in high dimensions".

Value

A list of model fitting results.

`gammacoef` Estimation for γ.

`groupnorm` Weighted norms for each group.

`lambda.seq` Sequence of λ values.

`alpha.seq`	Tuning parameter sequence for the generalized lasso penalty.
`rmid`	Column index for all zero features.
`family`	Option of `family`.
`cov.name`	Names for unpenalized features.
`bin.name`	Names for binary feautres.
`tree.object`	Outputs from `get_tree_obj()`.

References

Chen, J., Aseltine, R. H., Wang, F., & Chen, K. (2024). *Tree-Guided Rare Feature Selection and Logic Aggregation with Electronic Health Records Data. Journal of the American Statistical Association 119(547), 1765-1777*, doi:10.1080/01621459.2024.2326621.

Chen, X., Q. Lin, S. Kim, J. G. Carbonell, and E. P. Xing (2012). *Smoothing proximal gradient method for general structured sparse regression. The Annals of Applied Statistics 6(2), 719–752*, doi:10.1214/11AOAS514.

Yan, X. and J. Bien (2021). *Rare feature selection in high dimensions. Journal of the American Statistical Association 116(534), 887–900*, doi:10.1080/01621459.2020.1796677.

Examples

```
# Load the synthetic data
data(RegressionExample)

tree.org <- RegressionExample$tree.org   # original tree structure
x2.org <- RegressionExample$x.org      # original design matrix
y <- RegressionExample$y            # response

# Do the tree-guided expansion
expand.data <- getetmat(tree.org, x2.org)
x2 <- expand.data$x.expand              # expanded design matrix
tree.expand <- expand.data$tree.expand  # expanded tree structure

# specify some model parameters
set.seed(100)
control <- list(maxit = 100, mu = 1e-3, tol = 1e-5, verbose = FALSE)
# fit model with a pair of lambda and alpha
modstr <- list(lambda = 1,  alpha = 0.1)
x1 <- NULL
fit1 <- TSLA.fit(y, x1, x2, tree.expand, family = 'ls',
                 penalty = 'CL2',
                 gamma.init = NULL, weight = NULL,
                 group.weight = NULL, feature.weight = NULL,
                 control, modstr)
# get group norms from fit1
fit1$groupnorm
```

Index