

Package ‘TREAT’

December 23, 2013

Title TREe-based Association Test

Version 0.0.3

Date 2013-12-20

Author Kai Yu, Han Zhang

Description A fast and powerful tree-based association test for
detecting complex joint effects in case-control studies

Maintainer Bill Wheeler <wheelerb@imsweb.com>

License GPL-2

Archs x64

R topics documented:

TREAT	1
treat	2

Index	5
--------------	----------

TREAT	<i>TREe-based Association Test</i>
-------	------------------------------------

Description

A fast and powerful tree-based association test for detecting complex joint effects in case-control studies

Details

Multivariate tests derived from the logistic regression model are widely used to assess the joint effect of multiple predictors on a disease outcome in case-control studies. These tests become less optimal if the joint effect cannot be adequately approximated by the additive model. The tree structure model is an attractive alternative as it is more apt to capture non-additive effects. However, the tree model is most commonly used for prediction, seldom for hypothesis testing, mainly due to the computational burden associated with the resampling-based procedure required for estimating the significance level. The main function `treat` includes a fast algorithm for building the tree structure model, and a robust testing procedure that incorporates an adaptive model selection procedure to identify the optimal tree model representing the joint effect.

Author(s)

Han Zhang <han.zhang2@nih.gov> and Kai Yu <yuka@mail.nih.gov>

References

Zhang H, Wheeler W, Wang Z, Taylor P, Yu K. A fast and powerful tree-based test for detecting complex joint effects in case-control studies. Submitted.

treat	<i>TRee-based Association Test</i>
-------	------------------------------------

Description

A fast and powerful tree-based association test for detecting complex joint effects in case-control studies

Usage

```
treat(formula, data, snp.vars, op=NULL)
```

Arguments

formula	A formula describing the response and covariates. The response must be coded as 0 for controls and 1 for cases. The covariates must be categorical variables. If no covariates, use a formula similar to "Y ~ 1".
data	A data frame containing the response, covariates and SNPs.
snp.vars	Vector of column names and/or numbers for the SNPs. These variables must be coded as 0-1-2. If NULL, then all 0-1-2 variables in data that are not in the formula will be used as the SNP variables.
op	A list of options (see details). The default value is NULL.

Details

Missing values: Any row of data containing a missing value for the response, covariates or any SNP will be removed from the analysis.

Options list:

Below are the names for the options list `op`. All names have default values if they are not specified.

- `backward` <TRUE or FALSE> If TRUE, then the p-value is computed with backward searching as well. The default is TRUE.
- `cut.point` <Integer Vector> NULL or a vector of cut points. If NULL, then it is set to 2:nleaf. The default is NULL.
- `nleaf` <Integer> The maximum number of leaves in a tree. The default is 5.
- `nperm` <Integer> Number of permutation steps for computing the p-value. The default is 10000.
- `seed` <Integer> Seed for random number generation. If negative, the program will use the system time as the seed. The default is 0.

- `thr.grp.sample` <Integer> The lower limit of the sample size of each category defined by the covariates in a node to be split. The default is 2.
- `thr.leaf.size` <Integer> The minimum sample size for a leaf node in the tree. The default is 20.
- `thr.sample` <Integer> The minimum sample size for a node to be split. If the sample size in a node is less than `thr.sample`, the node will not be split. The default is 50.
- `max.missRate` <Number between 0 and 1> The maximum missing rate for each SNP. Any SNP with missing rate > `max.missRate` will be removed from the analysis. The default is 0.2.

Value

The returned value is a list with names:

- `adj.pval` The p-value of the TREE test, multiple-comparison adjusted.
- `unadj.pval` Vector of p-values at each cut-point, calculated from the tree.
- `comment` Status of the TREE test. It equals "done" if the estimated p-value is sufficiently accurate. It equals "refine" if the estimated p-value needs to be refined by more permutations.
- `adj.pval.back` The p-value of the TREE test with backward searching, multiple-comparison adjusted.
- `unadj.pval.back` Vector of p-values at each cut-point, calculated from the backward searching tree.
- `comment.back` Status of the TREE test with backward searching.
- `tree.model.forward` Data frame containing details of the built tree, including how the node is split and which indicator is used as the splitting rule. If the condition in the column "rule" holds, then the sample is assigned to "child1(TRUE)" otherwise the sample is assigned to "child2(FALSE)".
- `tree.model.backward` Details of the built tree with backward searching, including how the node is split and merged, and which indicator is used as the splitting rule.
- `snps` Final set of SNPs that were included in the analysis
- `rm.snps.coding` SNPs that were removed due to coding errors
- `rm.snps.missRate` SNPs the exceeded the maximum missing rate

Author(s)

Han Zhang and Kai Yu

Examples

```
set.seed(123)

# Create data
y <- c(rep(0, 1000), rep(1, 1000))
covar <- sample(0:2, 2000, replace=TRUE, prob=c(1/3, 3))
SNPs <- matrix(rbinom(2000*10, 2, c(.36, .48, .16)), nrow=2000)
SNPs <- as.data.frame(SNPs)
colnames(SNPs) <- paste("rs", 1:10, sep="")
data <- data.frame(y, covar, SNPs)
```

```
tr <- treat(y~factor(covar), data=data, snp.vars=colnames(SNPs))  
  
tr$adj.pval  
tr$tree.model.forward
```

Index

*Topic **model**
 treat, [2](#)
*Topic **package**
 TREAT, [1](#)

formula, [2](#)

TREAT, [1](#)
treat, [1](#), [2](#)