

# Package ‘logicDT’

April 26, 2022

**Type** Package

**Title** Identifying Interactions Between Binary Predictors

**Version** 1.0.2

**Description** A global statistical learning method which tries to find the best set of predictors and interactions between predictors for modeling binary or quantitative response data. Several search algorithms and ensembling techniques are implemented allowing for finetuning the method to the specific problem. Interactions with single quantitative covariables can be properly taken into account by also splitting after those or by fitting local four parameter logistic models.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** graphics, stats, utils

**RoxygenNote** 7.1.2

**NeedsCompilation** yes

**Author** Michael Lau [aut, cre] (<<https://orcid.org/0000-0002-5327-8351>>)

**Maintainer** Michael Lau <michael.lau@hhu.de>

**Repository** CRAN

**Date/Publication** 2022-04-26 11:20:02 UTC

## R topics documented:

|                            |   |
|----------------------------|---|
| bestBoostingIter . . . . . | 2 |
| calcAUC . . . . .          | 3 |
| calcBrier . . . . .        | 4 |
| calcDev . . . . .          | 4 |
| calcMis . . . . .          | 5 |
| calcMSE . . . . .          | 5 |
| calcNCE . . . . .          | 6 |
| calcNRMSE . . . . .        | 7 |
| cooling.schedule . . . . . | 7 |

|                          |    |
|--------------------------|----|
| fit4plModel              | 9  |
| fitLinearBoostingModel   | 10 |
| fitLinearLogicModel      | 11 |
| fitLinearModel           | 11 |
| getDesignMatrix          | 12 |
| gxe.test                 | 13 |
| gxe.test.boosting        | 15 |
| importance.test.boosting | 16 |
| logicDT                  | 17 |
| logicDT.bagging          | 22 |
| logicDT.boosting         | 23 |
| partial.predict          | 24 |
| plot.logicDT             | 25 |
| plot.vim                 | 26 |
| predict.4pl              | 26 |
| predict.linear           | 27 |
| predict.linear.logic     | 28 |
| predict.logicDT          | 28 |
| refitTrees               | 30 |
| splitSNPs                | 30 |
| tree.control             | 31 |
| vim                      | 32 |

## Index 37

---

|                  |   |
|------------------|---|
| bestBoostingIter | <i>Get the best number of boosting iterations</i> |
|------------------|---|

---

### Description

This function can be used to compute the ideal number of boosting iterations for the fitted `logic.boosted` model using independent validation data.

### Usage

```
bestBoostingIter(model, X, y, Z = NULL, consec.iter = 5, scoring_rule = "auc")
```

### Arguments

|                          |  |
|--------------------------|--|
| <code>model</code>       | Fitted <code>logic.boosted</code> model  |
| <code>X</code>           | Matrix or data frame of binary validation input data. This object should correspond to the binary matrix for fitting the model.          |
| <code>y</code>           | Validation response vector. 0-1 coding for binary outcomes.  |
| <code>Z</code>           | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them.       |
| <code>consec.iter</code> | Number of consecutive boosting iterations that do not increase the validation performance for determining the ideal number of iterations |

`scoring_rule` Scoring rule computing the validation performance. This can either be "auc" for the area under the receiver operating characteristic curve (default for binary reponses), "deviance" for the deviance, "nce" for the normalized cross entropy or "brier" for the Brier score. For regression purposes, the MSE (mean squared error) is automatically chosen.

### Details

If the model performance (on the validation data) cannot be increased for `consec.iter` consecutive boosting iterations, the last iteration which increased the validation performance induces the ideal number of boosting iterations.

### Value

The ideal number of boosting iterations

---

calcAUC

*Fast computation of the AUC w.r.t. to the ROC.*

---

### Description

This function computes the area under the receiver operating characteristic curve.

### Usage

```
calcAUC(preds, y, fast = TRUE, sorted = FALSE)
```

### Arguments

|                     |  |
|---------------------|--|
| <code>preds</code>  | Numeric vector of predicted scores   |
| <code>y</code>      | True binary outcomes coded as 0 or 1. Must be an integer vector.   |
| <code>fast</code>   | Shall the computation be as fast as possible?  |
| <code>sorted</code> | Are the predicted scores already sorted increasingly? If so, this can slightly speed up the computation. |

### Value

The AUC between 0 and 1

---

|           |                                  |
|-----------|----------------------------------|
| calcBrier | <i>Calculate the Brier score</i> |
|-----------|----------------------------------|

---

**Description**

Computation of the Brier score, i.e., the mean squared error for risk estimates in a binary classification problem.

**Usage**

```
calcBrier(preds, y)
```

**Arguments**

|       |                               |
|-------|-------------------------------|
| preds | Numeric vector of predictions |
| y     | True outcomes                 |

**Value**

The Brier score

---

|         |                               |
|---------|-------------------------------|
| calcDev | <i>Calculate the deviance</i> |
|---------|-------------------------------|

---

**Description**

Computation of the deviance, i.e., two times the negative log likelihood for risk estimates in a binary classification problem.

**Usage**

```
calcDev(preds, y)
```

**Arguments**

|       |                               |
|-------|-------------------------------|
| preds | Numeric vector of predictions |
| y     | True outcomes                 |

**Value**

The deviance

---

|         |   |
|---------|---|
| calcMis | <i>Calculate the misclassification rate</i> |
|---------|---|

---

**Description**

Computation of the misclassification rate for risk estimates in a binary classification problem.

**Usage**

```
calcMis(preds, y, cutoff = 0.5)
```

**Arguments**

|        |   |
|--------|---|
| preds  | Numeric vector of predictions                                 |
| y      | True outcomes   |
| cutoff | Classification cutoff. By default, scores above 50 otherwise. |

**Value**

The misclassification rate

---

|         |                          |
|---------|--------------------------|
| calcMSE | <i>Calculate the MSE</i> |
|---------|--------------------------|

---

**Description**

Computation of the mean squared error.

**Usage**

```
calcMSE(preds, y)
```

**Arguments**

|       |                               |
|-------|-------------------------------|
| preds | Numeric vector of predictions |
| y     | True outcomes                 |

**Value**

The MSE

---

`calcNCE`*Calculate the normalized cross entropy*

---

**Description**

This function computes the normalized cross entropy (NCE) which is given by

$$\text{NCE} = \frac{\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)}{p \cdot \log(p) + (1 - p) \cdot \log(1 - p)}$$

where (for  $i \in \{1, \dots, N\}$ )  $y_i \in \{0, 1\}$  are the true classes,  $p_i$  are the risk/probability predictions and  $p = \frac{1}{N} \sum_{i=1}^N y_i$  is total unrestricted empirical risk estimate.

**Usage**

```
calcNCE(preds, y)
```

**Arguments**

|                    |                                  |
|--------------------|----------------------------------|
| <code>preds</code> | Numeric vector of risk estimates |
| <code>y</code>     | Vector of true binary outcomes   |

**Details**

Smaller values towards zero are generally preferred. A NCE of one or above would indicate that the used model yields comparable or worse predictions than the naive mean model.

**Value**

The normalized cross entropy

**References**

- He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., Candela, J. Q. (2014). Practical Lessons from Predicting Clicks on Ads at Facebook. Proceedings of the Eighth International Workshop on Data Mining for Online Advertising 1-9. doi: [10.1145/2648584.2648589](https://doi.org/10.1145/2648584.2648589)

---

|           |                            |
|-----------|----------------------------|
| calcNRMSE | <i>Calculate the NRMSE</i> |
|-----------|----------------------------|

---

**Description**

Computation of the normalized root mean squared error.

**Usage**

```
calcNRMSE(preds, y, type = "sd")
```

**Arguments**

|       |  |
|-------|--|
| preds | Numeric vector of predictions  |
| y     | True outcomes  |
| type  | "sd" uses the standard deviation of y for normalization. "range" uses the whole span of y. |

**Value**

The NRMSE

---

|                  |  |
|------------------|--|
| cooling.schedule | <i>Define the cooling schedule for simulated annealing</i> |
|------------------|--|

---

**Description**

This function should be used to configure a search with simulated annealing.

**Usage**

```
cooling.schedule(  
  type = "adaptive",  
  start_temp = 1,  
  end_temp = -1,  
  lambda = 0.01,  
  total_iter = 2e+05,  
  markov_iter = 1000,  
  markov_leave_frac = 1,  
  acc_type = "probabilistic",  
  frozen_def = "acc",  
  frozen_acc_frac = 0.01,  
  frozen_markov_count = 5,  
  frozen_markov_mode = "total",  
  start_temp_steps = 10000,
```

```

    start_acc_ratio = 0.95,
    auto_start_temp = TRUE,
    remember_models = TRUE,
    print_iter = 1000
)

```

## Arguments

|                     |   |
|---------------------|---|
| type                | Type of cooling schedule. "adaptive" (default) or "geometric"   |
| start_temp          | Start temperature on a log10 scale. Only used if auto_start_temp = FALSE.   |
| end_temp            | End temperature on a log10 scale. Only used if type = "geometric".  |
| lambda              | Cooling parameter for the adaptive schedule. Values between 0.01 and 0.1 are recommended such that in total, several hundred thousand iterations are performed. Lower values lead to a more fine search with more iterations while higher values lead to a more rigorous search with less total iterations.   |
| total_iter          | Total number of iterations that should be performed. Only used for the geometric cooling schedule.  |
| markov_iter         | Number of iterations for each Markov chain. The standard value does not need to be tuned, since the temperature steps and number of iterations per chain act complementary to each other, i.e., less iterations can be compensated by smaller temperature steps.  |
| markov_leave_frac   | Fraction of accepted moves leading to an early temperature reduction. This is primarily used at (too) high temperatures lowering the temperature if essentially a random walk is performed. E.g., a value of 0.5 together with markov_iter = 1000 means that the chain will be left if $0.5 \cdot 1000 = 500$ states were accepted in a single chain. |
| acc_type            | Type of acceptance function. The standard "probabilistic" uses the conventional function $\exp((\text{Score}_{\text{old}} - \text{Score}_{\text{new}})/t)$ for calculating the acceptance probability. "deterministic" accepts the new state, if and only if $\text{Score}_{\text{new}} - \text{Score}_{\text{old}} < t$ .                            |
| frozen_def          | How to define a frozen chain. "acc" means that if less than frozen_acc_frac · markov_iter states with different scores were accepted in a single chain, this chain is marked as frozen. Several frozen chains indicate that the search is finished.   |
| frozen_acc_frac     | If frozen_def = frozen_acc_frac, this parameter determines the fraction of iterations which defines a frozen chain.   |
| frozen_markov_count | How many frozen chains need to be observed for finishing the search?  |
| frozen_markov_mode  | Do the frozen chains have to occur consecutively ("consecutive") or is the total number of frozen chains relevant ("total")?  |
| start_temp_steps    | If auto_start_temp = TRUE, how many iterations should be used for estimating the ideal start temperature?   |



|                 |  |
|-----------------|--|
| start_acc_ratio | Which acceptance ratio should be achieved with the automatically configured start temperature?                           |
| auto_start_temp | Should the start temperature be configured automatically? TRUE or FALSE  |
| remember_models | Should already evaluated models be saved in a 2-dimensional hash table to prevent fitting the same trees multiple times? |
| print_iter      | After how many iterations shall a progress report be printed?  |

### Details

type = "adapative" (default) automatically choses the temperature steps by using the standard deviation of the scores in a Markov chain together with the current temperature to evaluate if equilibrium is achieved. If the standard deviation is small or the temperature is high, equilibrium can be assumed leading to a strong temperature reduction. Otherwise, the temperature is only merely lowered. The parameter lambda is essential to control how fast the schedule will be executed and, thus, how many total iterations will be performed.

type = "geometric" is the conventional approach which requires more finetuning. Here, temperatures are uniformly lowered on a log10 scale. Thus, a start and an end temperature have to be supplied.

### Value

An object of class `cooling.schedule` which is a list of all necessary cooling parameters.

---

|             |                           |
|-------------|---------------------------|
| fit4plModel | <i>Fitting 4pL models</i> |
|-------------|---------------------------|

---

### Description

Method for fitting four parameter logistic models. In the fashion of this package, only binary and quantitative outcomes are supported.

### Usage

```
fit4plModel(y, Z)
```

### Arguments

|   |  |
|---|--|
| y | Response vector. 0-1 coding for binary outcomes, otherwise conventional regression is performed. |
| Z | Numeric vector of (univariate) input samples.  |

**Details**

4pL models are non-linear regression models of the shape

$$Y = f(x, b, c, d, e) + \varepsilon = c + \frac{d - c}{1 + \exp(b \cdot (x - e))} + \varepsilon$$

with  $\varepsilon$  being a random error term.

**Value**

An object of class "4pl" which contains a numeric vector of the fitted parameters b, c, d, and e.

---

fitLinearBoostingModel

*Linear models based on boosted models*

---

**Description**

This function uses a fitted logic.boosted model for fitting a linear or logistic (depending on the type of outcome) regression model.

**Usage**

```
fitLinearBoostingModel(model, n.iter)
```

**Arguments**

|        |  |
|--------|--|
| model  | Fitted logic.boosted model               |
| n.iter | Number of boosting iterations to be used |

**Details**

In this procedure, the logic terms are extracted from the individual logicDT models and the set of unique terms are used as predictors in a regression model. For incorporating a continuous covariable the covariable itself as well as products of the covariable with the extracted logic terms are included as predictors in the regression model.

**Value**

A linear.logic model. This is a list containing the logic terms used as predictors in the model and the fitted glm model.

---

fitLinearLogicModel     *Linear models based on logic terms*

---

### Description

This function fits a linear or logistic regression model (based on the type of outcome) using the supplied logic terms, e.g., \$disj from a fitted logicDT model.

### Usage

```
fitLinearLogicModel(X, y, Z = NULL, disj, Z.interactions = TRUE)
```

### Arguments

|                |  |
|----------------|--|
| X              | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.   |
| y              | Response vector. 0-1 coding for binary outcomes.   |
| Z              | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them.   |
| disj           | Integer matrix of logic terms. As in <a href="#">logicDT</a> , each row corresponds to a term/conjunction. Negative values indicate negations. The absolute values of an entry correspond to the predictor index in X. |
| Z.interactions | Shall interactions with the continuous covariable Z be taken into account by including products of the terms with Z?   |

### Value

A linear.logic model. This is a list containing the logic terms used as predictors in the model and the fitted glm model.

---

fitLinearModel     *Fitting linear models*

---

### Description

Method for fitting linear models. In the fashion of this package, only binary and quantitative outcomes are supported.

### Usage

```
fitLinearModel(y, Z)
```

**Arguments**

|   |  |
|---|--|
| y | Response vector. 0-1 coding for binary outcomes, otherwise conventional regression is performed. |
| Z | Numeric vector of (univariate) input samples.  |

**Details**

For binary outcomes, predictions are cut at 0 or 1 for generating proper probability estimates.

**Value**

An object of class "linear" which contains a numeric vector of the fitted parameters b and c.

---

|                 |  |
|-----------------|--|
| getDesignMatrix | <i>Design matrix for the set of conjunctions</i> |
|-----------------|--|

---

**Description**

Transform the original predictor matrix X into the conjunction design matrix which contains for each conjunction a corresponding column.

**Usage**

```
getDesignMatrix(X, disj)
```

**Arguments**

|      |  |
|------|--|
| X    | The original (binary) predictor matrix. This has to be of type integer.                      |
| disj | The conjunction matrix which can, e.g., be extracted from a fitted logicDT model via \$disj. |

**Value**

The transformed design matrix.

---

|          |  |
|----------|--|
| gxe.test | <i>Gene-environment interaction test</i> |
|----------|--|

---

### Description

Using a fitted logicDT model, a general GxE interaction test can be performed.

### Usage

```
gxe.test(model, X, y, Z, perm.test = TRUE, n.perm = 10000)
```

### Arguments

|           |  |
|-----------|--|
| model     | A fitted logicDT model with 4pL models in its leaves.  |
| X         | Binary predictor data for testing the interaction effect. This can be equal to the training data.                      |
| y         | Response vector for testing the interaction effect. This can be equal to the training data.                            |
| Z         | Quantitative covariable for testing the interaction effect. This can be equal to the training data.                    |
| perm.test | Should additionally permutation testing be performed? Useful if likelihood ratio test asymptotics cannot be justified. |
| n.perm    | Number of random permutations for permutation testing  |

### Details

The testing is done by fitting one shared 4pL model for all tree branches with different offsets, i.e., allowing main effects of SNPs. This shared model is compared to the individual 4pL models fitted in the `logicDT` procedure using a likelihood ratio test which is asymptotically  $\chi^2$  distributed. The degrees of freedom are equal to the difference in model parameters. For regression tasks, alternatively, a F-test can be utilized.

The shared 4pL model is given by

$$Y = \tilde{f}(x, z, b, c, d, e, \beta_1, \dots, \beta_{G-1}) + \varepsilon = c + \frac{d - c}{1 + \exp(b \cdot (x - e))} + \sum_{g=1}^{G-1} \beta_g \cdot 1(z = g) + \varepsilon$$

with  $z \in \{1, \dots, G\}$  being a grouping variable,  $\beta_1, \dots, \beta_{G-1}$  being the offsets for the different groups, and  $\varepsilon$  being a random error term. Note that the last group  $G$  does not have an offset parameter, since the model is calibrated such that the curve without any  $\beta$ 's fits to the last group.

The likelihood ratio test statistic is given by

$$\Lambda = -2(\ell_{\text{shared}} - \ell_{\text{full}})$$

for the log likelihoods of the shared and full 4pL models, respectively. In the regression case, the test statistic can be calculated as

$$\Lambda = N(\log(\text{RSS}_{\text{shared}}) - \log(\text{RSS}_{\text{full}}))$$

with RSS being the residual sum of squares for the respective model.

For regression tasks, the alternative F test statistic is given by

$$f = \frac{\frac{1}{df_1}(\text{RSS}_{\text{shared}} - \text{RSS}_{\text{full}})}{\frac{1}{df_2}\text{RSS}_{\text{full}}}$$

with

$$df_1 = \text{Difference in the number of model parameters} = 3 \cdot n_{\text{scenarios}} - 3,$$

$$df_2 = \text{Degrees of freedom of the full model} = N - 4 \cdot n_{\text{scenarios}},$$

and  $n_{\text{scenarios}}$  being the number of identified predictor scenarios/groups by `logicDT`.

Alternatively, if linear models were fitted in the supplied `logicDT` model, shared linear models can be used to test for a GxE interaction. For continuous outcomes, the shared linear model is given by

$$Y = \tilde{f}(x, z, \alpha, \beta_1, \dots, \beta_G) + \varepsilon = \alpha \cdot x + \sum_{g=1}^G \beta_g \cdot 1(z = g) + \varepsilon.$$

For binary outcomes, LDA (linear discriminant analysis) models are fitted. In contrast to the 4pL-based test for binary outcomes, varying offsets for the individual groups are injected to the linear predictor instead of to the probability (response) scale.

If only few samples are available and the asymptotics of likelihood ratio tests cannot be justified, alternatively, a permutation test approach can be employed by setting `perm.test = TRUE` and specifying an appropriate number of random permutations via `n.perm`. For this approach, computed likelihoods of the shared and (paired) full likelihood groups are randomly interchanged approximating the null distribution of equal likelihoods. A p-value can be computed by determining the fraction of more extreme null samples compared to the original likelihood ratio test statistic, i.e., using the fraction of higher likelihood ratios in the null distribution than the original likelihood ratio.

## Value

A list containing

|                         |   |
|-------------------------|---|
| <code>p.chisq</code>    | The p-value of the chi-squared test statistic.                  |
| <code>p.f</code>        | The p-value of the F test statistic.                            |
| <code>p.perm</code>     | The p-value of the optional permutation test.                   |
| <code>ll.shared</code>  | Log likelihood of the shared parameters 4pL model.              |
| <code>ll.full</code>    | Log likelihood of the full <code>logicDT</code> model.          |
| <code>rss.shared</code> | Residual sum of squares of the shared parameters 4pL model.     |
| <code>rss.full</code>   | Residual sum of squares of the full <code>logicDT</code> model. |

---

|                   |   |
|-------------------|---|
| gxe.test.boosting | <i>Gene-environment (GxE) interaction test based on boosted linear models</i> |
|-------------------|---|

---

## Description

This function takes a fitted `linear.logic` model and independent test data as input for testing if there is a general GxE interaction. This hypothesis test is based on a likelihood-ratio test.

## Usage

```
gxe.test.boosting(model, X, y, Z)
```

## Arguments

|       |   |
|-------|---|
| model | A fitted <code>linear.logic</code> model (i.e., a model created via <a href="#">fitLinearLogicModel</a> or <a href="#">fitLinearBoostingModel</a> ) |
| X     | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.                                |
| y     | Response vector. 0-1 coding for binary outcomes.  |
| Z     | Quantitative covariable supplied as a matrix or data frame  |

## Details

In detail, the null hypothesis

$$H_0 : \delta_1 = \dots = \delta_B = 0$$

using the supplied linear model

$$g(E[Y]) = \beta_0 + \sum_{i=1}^B \beta_i \cdot 1[C_i] + \delta_0 \cdot E + \sum_{i=1}^B \delta_i \cdot 1[C_i] \cdot E$$

is tested.

## Value

A list containing

|          |  |
|----------|--|
| Deviance | The deviance used for performing the likelihood-ratio test |
| p.value  | The p-value of the test                                    |

---

 importance.test.boosting

*Term importance test based on boosted linear models*


---

### Description

This function takes a fitted `linear.logic` model and independent test data as input for testing if the included terms are influential with respect to the outcome. This hypothesis test is based on a likelihood-ratio test.

### Usage

```
importance.test.boosting(model, X, y, Z, Z.interactions = TRUE)
```

### Arguments

|                |   |
|----------------|---|
| model          | A fitted <code>linear.logic</code> model (i.e., a model created via <code>fitLinearLogicModel</code> or <code>fitLinearBoostingModel</code> ) |
| X              | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.                          |
| y              | Response vector. 0-1 coding for binary outcomes.  |
| Z              | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them.            |
| Z.interactions | A Boolean value determining whether interactions with quantitative covariable Z shall be taken into account                                   |

### Details

In detail, the null hypotheses

$$H_0 : \beta_j = \delta_j = 0$$

using the linear model

$$g(E[Y]) = \beta_0 + \sum_{i=1}^B \beta_i \cdot 1[C_i] + \delta_0 \cdot E + \sum_{i=1}^B \delta_i \cdot 1[C_i] \cdot E$$

are tested for each  $j \in \{1, \dots, B\}$  if `Z.interactions` is set to `TRUE`. Otherwise, the null hypotheses

$$H_0 : \beta_j = 0$$

using the linear model

$$g(E[Y]) = \beta_0 + \sum_{i=1}^B \beta_i \cdot 1[C_i] + \delta_0 \cdot E$$

are tested.



**Value**

A data frame consisting of three columns,

|         |   |
|---------|---|
| var     | The tested term,  |
| vim     | The associated variable importance, and                           |
| p.value | The corresponding p-value for testing if the term is influential. |

---

|         |                                     |
|---------|-------------------------------------|
| logicDT | <i>Fitting logic decision trees</i> |
|---------|-------------------------------------|

---

**Description**

Main function for fitting logicDT models.

**Usage**

```
## Default S3 method:
logicDT(
  X,
  y,
  max_vars = 3,
  max_conj = 3,
  Z = NULL,
  search_algo = "sa",
  cooling_schedule = cooling.schedule(),
  scoring_rule = "auc",
  tree_control = tree.control(),
  gamma = 0,
  simplify = "vars",
  val_method = "none",
  val_frac = 0.5,
  val_reps = 10,
  allow_conj_removal = TRUE,
  conjsize = 1,
  randomize_greedy = FALSE,
  greedy_mod = TRUE,
  greedy_rem = FALSE,
  max_gen = 10000,
  gp_sigma = 0.15,
  gp_fs_interval = 1,
  ...
)

## S3 method for class 'formula'
logicDT(formula, data, ...)
```

**Arguments**

|                                 |   |
|---------------------------------|---|
| <code>X</code>                  | Matrix or data frame of binary predictors coded as 0 or 1.  |
| <code>y</code>                  | Response vector. 0-1 coding for binary responses. Otherwise, a regression task is assumed.  |
| <code>max_vars</code>           | Maximum number of predictors in the set of predictors. For the set $[X_1 \wedge X_2^c, X_1 \wedge X_3]$ , this parameter is equal to 4.   |
| <code>max_conj</code>           | Maximum number of input variables for the decision trees. For the set $[X_1 \wedge X_2^c, X_1 \wedge X_3]$ , this parameter is equal to 2.  |
| <code>Z</code>                  | Optional matrix or data frame of quantitative/continuous covariables. Multiple covariables allowed for splitting the trees. If four parameter logistic models shall be fitted in the leaves, only the first given covariable is used.   |
| <code>search_algo</code>        | Search algorithm for guiding the global search. This can either be "sa" for simulated annealing, "greedy" for a greedy search or "gp" for genetic programming.  |
| <code>cooling_schedule</code>   | Cooling schedule parameters if simulated annealing is used. The required object should be created via the function <a href="#">cooling.schedule</a> .   |
| <code>scoring_rule</code>       | Scoring rule for guiding the global search. This can either be "auc" for the area under the receiver operating characteristic curve (default for binary responses), "deviance" for the deviance, "nce" for the normalized cross entropy or "brier" for the Brier score. For regression purposes, the MSE (mean squared error) is automatically chosen.  |
| <code>tree_control</code>       | Parameters controlling the fitting of decision trees. This should be configured via the function <a href="#">tree.control</a> .   |
| <code>gamma</code>              | Complexity penalty added to the score. If $\gamma > 0$ is given, $\gamma \cdot \ m\ _0$ is added to the score with $\ m\ _0$ being the total number of variables contained in the current model $m$ . The main purpose of this penalty is for fitting logicDT stumps in conjunction with boosting. For regular logicDT models or bagged logicDT models, instead, the model complexity parameters <code>max_vars</code> and <code>max_conj</code> should be tuned. |
| <code>simplify</code>           | Should the final fitted model be simplified? This means, that unnecessary terms as a whole ("conj") will be removed if they cannot improve the score. <code>simplify = "vars"</code> additionally tries to prune individual conjunctions by removing unnecessary variables in those. <code>simplify = "none"</code> will not modify the final model.  |
| <code>val_method</code>         | Inner validation method. "rv" leads to a repeated validation where <code>val_reps</code> times the original data set is divided into <code>val_frac</code> · 100% validation data and $(1 - \text{val\_frac}) \cdot 100\%$ training data. "bootstrap" draws bootstrap samples and uses the out-of-bag data as validation data. "cv" employs cross-validation with <code>val_reps</code> folds.  |
| <code>val_frac</code>           | Only used if <code>val_method = "rv"</code> . See description of <code>val_method</code> .  |
| <code>val_reps</code>           | Number of inner validation partitionings.   |
| <code>allow_conj_removal</code> | Should it be allowed to remove complete terms/conjunctions in the search? If exact numbers of terms are aimed at, this should be set to FALSE. If extensive hyperparameter optimizations are feasible, <code>allow_conj_removal = FALSE</code> with a   |

|                               |   |
|-------------------------------|---|
|                               | proper search over <code>max_vars</code> and <code>max_conj</code> is advised for fitting single models. For bagging or boosting with a greedy search, <code>allow_conj_removal = TRUE</code> together with a small number for <code>max_vars = max_conj</code> is recommended, e.g., 2 or 3. |
| <code>conjsize</code>         | The minimum of training samples that have to belong to a conjunction. This parameters prevents including unnecessarily complex conjunctions that rarely occur.  |
| <code>randomize_greedy</code> | Should the greedy search be randomized by only considering $\sqrt{\text{Neighbour}}$ states neighbors at each iteration, similar to random forests. Speeds up the greedy search but can lead to inferior results.   |
| <code>greedy_mod</code>       | Should modifications of conjunctions be considered in a greedy search? Speeds up the greedy search but can lead to inferior results.  |
| <code>greedy_rem</code>       | Should the removal of conjunctions be considered in a greedy search? Speeds up the greedy search but can lead to inferior results.  |
| <code>max_gen</code>          | Maximum number of generations for genetic programming.  |
| <code>gp_sigma</code>         | Parameter $\sigma$ for fitness sharing in genetic programming. Very small values (e.g., 0.001) are recommended leading to only penalizing models with yield the exact same score.   |
| <code>gp_fs_interval</code>   | Interval for fitness sharing in genetic programming. The fitness calculation can be computationally expensive if many models exist in one generation. <code>gp_fs_interval = 10</code> leads to performing fitness sharing only every 10th generation.  |
| <code>...</code>              | Arguments passed to <code>logicDT.default</code>  |
| <code>formula</code>          | An object of type <code>formula</code> describing the model to be fitted.   |
| <code>data</code>             | A data frame containing the data for the corresponding <code>formula</code> object. Must also contain quantitative covariables if they should be included as well.  |

## Details

`logicDT` is a method for finding response-associated interactions between binary predictors. A global search for the best set of predictors and interactions between predictors is performed trying to find the global optimal decision trees. On the one hand, this can be seen as a variable selection. On the other hand, Boolean conjunctions between binary predictors can be identified as impactful which is particularly useful if the corresponding marginal effects are negligible due to the greedy fashion of choosing splits in decision trees.

Three search algorithms are implemented:

- Simulated annealing. An exhaustive stochastic optimization procedure. Recommended for single models (without [outer] bagging or boosting).
- Greedy search. A very fast search always looking for the best possible improvement. Recommended for ensemble models.
- Genetic programming. A more or less intensive search holding several competitive models at each generation. Niche method which is only recommended if multiple (simple) models do explain the variation in the response.

Furthermore, the option of a so-called "inner validation" is available. Here, the search is guided using several train-validation-splits and the average of the validation performance. This approach is computationally expensive but can lead to more robust single models.

For minimizing the computation time, two-dimensional hash tables are used saving evaluated models. This is irrelevant for the greedy search but can heavily improve the fitting times when employing a search with simulated annealing or genetic programming, especially when choosing an inner validation.

## Value

An object of class `logicDT`. This is a list containing

|                              |  |
|------------------------------|--|
| <code>disj</code>            | A matrix of identified set of predictors and conjunctions of predictors. Each entry corresponds to the column index in X. Negative values indicate negations. Missing values mean that the term does not contain any more variables. |
| <code>real_disj</code>       | Human readable form of <code>disj</code> . Here, variable names are directly depicted.   |
| <code>score</code>           | Score of the best model. Smaller values are preferred.   |
| <code>pet</code>             | Decision tree fitted on the best set of input terms. This is a list containing the pointer to the C representation of the tree and R representations of the tree structure such as the splits and predictions.                       |
| <code>ensemble</code>        | List of decision trees. Only relevant if inner validation was used.  |
| <code>total_iter</code>      | The total number of search iterations, i.e., tested configurations by fitting a tree (ensemble) and evaluating it.   |
| <code>prevented_evals</code> | The number of prevented tree fitting by using the 2-dimensional hash table.  |
| <code>...</code>             | Supplied parameters of the functional call to <code>logicDT</code> .   |

## Saving and Loading

`logicDT` models can be saved and loaded using `save(...)` and `load(...)`. The internal C structures will not be saved but rebuilt from the R representations if necessary.

## References

- Lau, M., Schikowski, T. & Schwender, H. (2021). `logicDT`: A Procedure for Identifying Response-Associated Interactions Between Binary Predictors. To be submitted.
- Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. (1984). Classification and Regression Trees. CRC Press. doi: [10.1201/9781315139470](https://doi.org/10.1201/9781315139470)
- Kirkpatrick, S., Gelatt C. D. & Vecchi M. P. (1983). Optimization by Simulated Annealing. Science 220(4598):671–680. doi: [10.1126/science.220.4598.671](https://doi.org/10.1126/science.220.4598.671)

## Examples

```
# Generate toy data
set.seed(123)
maf <- 0.25
n.snps <- 50
```

```

N <- 2000
X <- matrix(sample(0:2, n.snps * N, replace = TRUE,
                 prob = c((1-maf)^2, 1-(1-maf)^2-maf^2, maf^2)), ncol = n.snps)
colnames(X) <- paste("SNP", 1:n.snps, sep="")
X <- splitSNPs(X)
Z <- matrix(rnorm(N, 20, 10), ncol = 1)
colnames(Z) <- "E"
Z[Z < 0] <- 0
y <- -0.75 + log(2) * (X[, "SNP1D"] != 0) +
  log(4) * Z/20 * (X[, "SNP2D"] != 0 & X[, "SNP3D"] == 0) +
  rnorm(N, 0, 1)

# Fit and evaluate single logicDT model
model <- logicDT(X[1:(N/2),], y[1:(N/2)], Z = Z[1:(N/2),,drop=FALSE],
                max_vars = 3, max_conj = 2,
                search_algo = "sa",
                tree_control = tree.control(nodesize = floor(0.05 * nrow(X)/2)),
                simplify = "vars",
                allow_conj_removal = FALSE, conjsize = floor(0.05 * nrow(X)/2))
calcNRMSE(predict(model, X[(N/2+1):N,], Z = Z[(N/2+1):N,drop=FALSE]),
           y[(N/2+1):N])
plot(model)
print(model)

# Fit and evaluate bagged logicDT model
model.bagged <- logicDT.bagging(X[1:(N/2),], y[1:(N/2)], Z = Z[1:(N/2),,drop=FALSE],
                               bagging.iter = 50,
                               max_vars = 3, max_conj = 3,
                               search_algo = "greedy",
                               tree_control = tree.control(nodesize = floor(0.05 * nrow(X)/2)),
                               simplify = "vars",
                               conjsize = floor(0.05 * nrow(X)/2))
calcNRMSE(predict(model.bagged, X[(N/2+1):N,], Z = Z[(N/2+1):N,drop=FALSE]),
           y[(N/2+1):N])
print(model.bagged)

# Fit and evaluate boosted logicDT model
model.boosted <- logicDT.boosting(X[1:(N/2),], y[1:(N/2)], Z = Z[1:(N/2),,drop=FALSE],
                                  boosting.iter = 50, learning.rate = 0.01,
                                  subsample.frac = 0.75, replace = FALSE,
                                  max_vars = 3, max_conj = 3,
                                  search_algo = "greedy",
                                  tree_control = tree.control(nodesize = floor(0.05 * nrow(X)/2)),
                                  simplify = "vars",
                                  conjsize = floor(0.05 * nrow(X)/2))
calcNRMSE(predict(model.boosted, X[(N/2+1):N,], Z = Z[(N/2+1):N,drop=FALSE]),
           y[(N/2+1):N])
print(model.boosted)

# Calculate VIMs (variable importance measures)
vims <- vim(model.bagged)
plot(vims)

```

```

print(vims)

# Single greedy model
model <- logicDT(X[1:(N/2)], y[1:(N/2)], Z = Z[1:(N/2)], drop=FALSE,
               max_vars = 3, max_conj = 2,
               search_algo = "greedy",
               tree_control = tree.control(nodesize = floor(0.05 * nrow(X)/2)),
               simplify = "vars",
               allow_conj_removal = FALSE, conjsize = floor(0.05 * nrow(X)/2))
calcNRMSE(predict(model, X[(N/2+1):N], Z = Z[(N/2+1):N], drop=FALSE),
          y[(N/2+1):N])
plot(model)
print(model)

```

---

logicDT.bagging      *Fitting bagged logicDT models*

---

## Description

Function for fitting bagged logicDT models.

## Usage

```

## Default S3 method:
logicDT.bagging(X, y, Z = NULL, bagging.iter = 500, ...)

## S3 method for class 'formula'
logicDT.bagging(formula, data, ...)

```

## Arguments

|              |   |
|--------------|---|
| X            | Matrix or data frame of binary predictors coded as 0 or 1.  |
| y            | Response vector. 0-1 coding for binary responses. Otherwise, a regression task is assumed.  |
| Z            | Optional matrix or data frame of quantitative/continuous covariables. Multiple covariables allowed for splitting the trees. If four parameter logistic models shall be fitted in the leaves, only the first given covariable is used. |
| bagging.iter | Number of bagging iterations  |
| ...          | Arguments passed to logicDT   |
| formula      | An object of type formula describing the model to be fitted.  |
| data         | A data frame containing the data for the corresponding formula object. Must also contain quantitative covariables if they should be included as well.   |

## Details

Details on single logicDT models can be found in [logicDT](#).

**Value**

An object of class `logic.bagged`. This is a list containing

|                     |   |
|---------------------|---|
| <code>models</code> | A list of fitted <code>logicDT</code> models                                    |
| <code>bags</code>   | A list of observations indices which were used to train each model              |
| <code>...</code>    | Supplied parameters of the functional call to <a href="#">logicDT.bagging</a> . |

---

|                               |                                       |
|-------------------------------|---------------------------------------|
| <code>logicDT.boosting</code> | <i>Fitting boosted logicDT models</i> |
|-------------------------------|---------------------------------------|

---

**Description**

Function for fitting gradient boosted `logicDT` models.

**Usage**

```
## Default S3 method:
logicDT.boosting(
  X,
  y,
  Z = NULL,
  boosting.iter = 500,
  learning.rate = 0.01,
  subsample.frac = 1,
  replace = TRUE,
  line.search = "min",
  ...
)

## S3 method for class 'formula'
logicDT.boosting(formula, data, ...)
```

**Arguments**

|                             |   |
|-----------------------------|---|
| <code>X</code>              | Matrix or data frame of binary predictors coded as 0 or 1.  |
| <code>y</code>              | Response vector. 0-1 coding for binary responses. Otherwise, a regression task is assumed.  |
| <code>Z</code>              | Optional matrix or data frame of quantitative/continuous covariables. Multiple covariables allowed for splitting the trees. If four parameter logistic models shall be fitted in the leaves, only the first given covariable is used. |
| <code>boosting.iter</code>  | Number of boosting iterations   |
| <code>learning.rate</code>  | Learning rate for boosted models. Values between 0.001 and 0.1 are recommended.   |
| <code>subsample.frac</code> | Subsample fraction for each boosting iteration. E.g., 0.5 means that a random draw of 50 is used in each iteration.   |

|                          |   |
|--------------------------|---|
| <code>replace</code>     | Should the random draws with <code>subsample.frac</code> in boosted models be performed with or without replacement? TRUE or FALSE  |
| <code>line.search</code> | Type of line search for gradient boosting. "min" performs a real minimization while "binary" performs a loose binary search for a boosting coefficient that just reduces the score. |
| <code>...</code>         | Arguments passed to <code>logicDT</code>  |
| <code>formula</code>     | An object of type <code>formula</code> describing the model to be fitted.   |
| <code>data</code>        | A data frame containing the data for the corresponding <code>formula</code> object. Must also contain quantitative covariables if they should be included as well.                  |

### Details

Details on single `logicDT` models can be found in [logicDT](#).

### Value

An object of class `logic.boosted`. This is a list containing

|                           |  |
|---------------------------|--|
| <code>models</code>       | A list of fitted <code>logicDT</code> models                                     |
| <code>rho</code>          | A vector of boosting coefficient corresponding to each model                     |
| <code>initialModel</code> | Initial model which is usually the observed mean                                 |
| <code>...</code>          | Supplied parameters of the functional call to <a href="#">logicDT.boosting</a> . |

---

|                              |  |
|------------------------------|--|
| <code>partial.predict</code> | <i>Partial prediction for boosted models</i> |
|------------------------------|--|

---

### Description

Alternative prediction function for `logic.boosted` models using up to `n.iter` boosting iterations. An array of predictions for every number of boosting iterations up to `n.iter` is returned.

### Usage

```
partial.predict(model, X, Z = NULL, n.iter = 1, ...)
```

### Arguments

|                     |  |
|---------------------|--|
| <code>model</code>  | Fitted <code>logic.boosted</code> model  |
| <code>X</code>      | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.               |
| <code>Z</code>      | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them. |
| <code>n.iter</code> | Maximum number of boosting iterations for prediction   |
| <code>...</code>    | Parameters supplied to <a href="#">predict.logicDT</a>   |



**Details**

The main purpose of this function is to retrieve the optimal number of boosting iterations (early stopping) using a validation data set and to restrict future predictions on this number of iterations.

**Value**

An array of dimension  $(N, n.iter)$  containing the partial predictions

---

|              |                                   |
|--------------|-----------------------------------|
| plot.logicDT | <i>Plot a logic decision tree</i> |
|--------------|-----------------------------------|

---

**Description**

This function plots a logicDT model on the active graphics device.

**Usage**

```
fancy.plot(x, ...)

## S3 method for class 'logicDT'
plot(x, fancy = TRUE, x_scaler = 0.5, margin_scaler = 0.2, cex = 1, ...)
```

**Arguments**

|               |   |
|---------------|---|
| x             | An object of the class logicDT  |
| ...           | Arguments passed to fancy plotting function   |
| fancy         | Should the fancy mode be used for plotting? Default is TRUE.  |
| x_scaler      | Scaling factor on the horizontal axis for deeper trees, i.e., $x\_scaler = 0.5$ means that the horizontal distance between two adjacent nodes is halved for every vertical level. |
| margin_scaler | Margin factor. Smaller values lead to smaller margins.  |
| cex           | Scaling factor for the plotted text elements.   |

**Details**

There are two plotting modes:

- `fancy = FALSE` which draws a tree with direct edges between the nodes. Leaves are represented by their prediction value which is obtained by the (observed) conditional mean.
- `fancy = TRUE` plots a tree similar to those in the `rpart` (Therneau and Atkinson, 2019) and `splinetree` (Neufeld and Heggeseth, 2019) R packages. The trees are drawn in an angular manner and if 4pL modeling was used for the leaves, appropriate plots of the fitted curves are depicted in the leaves. Otherwise, the usual prediction values are shown.

**Value**

No return value, called for side effects

**References**

- Therneau, T. & Atkinson, B. (2019). rpart: Recursive Partitioning and Regression Trees. <https://CRAN.R-project.org/package=rpart>
- Neufeld, A. & Heggseth, B. (2019). splinetree: Longitudinal Regression Trees and Forests. <https://CRAN.R-project.org/package=splinetree>

---

|          |                             |
|----------|-----------------------------|
| plot.vim | <i>Plot calculated VIMs</i> |
|----------|-----------------------------|

---

**Description**

This function plots variable importance measures yielded by the function `vim` in a dotchart.

**Usage**

```
## S3 method for class 'vim'
plot(x, p = 10, ...)
```

**Arguments**

|     |  |
|-----|--|
| x   | An object of the class <code>vim</code>  |
| p   | The number of most important terms which will be included in the plot. A value of 0 leads to plotting all terms. |
| ... | Ignored additional parameters  |

**Value**

No return value, called for side effects

---

|             |                                  |
|-------------|----------------------------------|
| predict.4pl | <i>Prediction for 4pL models</i> |
|-------------|----------------------------------|

---

**Description**

Use new input data and a fitted four parameter logistic model to predict corresponding outcomes.

**Usage**

```
## S3 method for class '`4pl`'
predict(object, Z, ...)
```

**Arguments**

|        |                                     |
|--------|-------------------------------------|
| object | Fitted 4pl model                    |
| Z      | Numeric vector of new input samples |
| ...    | Ignored additional parameters       |

**Value**

A numeric vector of predictions. For binary outcomes, this is a vector with estimates for  $P(Y = 1 | X = x)$ .

---

predict.linear      *Prediction for linear models*

---

**Description**

Use new input data and a fitted linear model to predict corresponding outcomes.

**Usage**

```
## S3 method for class 'linear'
predict(object, Z, ...)
```

**Arguments**

|        |                                     |
|--------|-------------------------------------|
| object | Fitted linear model                 |
| Z      | Numeric vector of new input samples |
| ...    | Ignored additional parameters       |

**Details**

For binary outcomes, predictions are cut at 0 or 1 for generating proper probability estimates.

**Value**

A numeric vector of predictions. For binary outcomes, this is a vector with estimates for  $P(Y = 1 | X = x)$ .

---

predict.linear.logic    *Prediction for linear.logic models*

---

### Description

Use new input data and a fitted linear.logic model to predict corresponding outcomes.

### Usage

```
## S3 method for class 'linear.logic'
predict(object, X, Z = NULL, ...)
```

### Arguments

|        |  |
|--------|--|
| object | Fitted linear.logic model  |
| X      | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.               |
| Z      | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them. |
| ...    | Ignored additional parameters  |

### Value

A numeric vector of predictions. For binary outcomes, this is a vector with estimates for  $P(Y = 1 | X = x)$ .

---

predict.logicDT    *Prediction for logicDT models*

---

### Description

Supply new input data for predicting the outcome with a fitted logicDT model.

### Usage

```
## S3 method for class 'logic.bagged'
predict(object, X, Z = NULL, type = "prob", ...)
```

```
## S3 method for class 'logic.boosted'
predict(object, X, Z = NULL, type = "prob", ...)
```

```
## S3 method for class 'logicDT'
predict(
  object,
  X,
```

```

    Z = NULL,
    type = "prob",
    ensemble = FALSE,
    leaves = "4pl",
    ...
)

## S3 method for class 'geneticLogicPET'
predict(
  object,
  X,
  Z = NULL,
  models = "best",
  n_models = 10,
  ensemble = NULL,
  leaves = "4pl",
  ...
)

```

### Arguments

|          |   |
|----------|---|
| object   | Fitted logicDT model. Usually a product of a call to <a href="#">logicDT</a> .  |
| X        | Matrix or data frame of binary input data. This object should correspond to the binary matrix for fitting the model.  |
| Z        | Optional quantitative covariables supplied as a matrix or data frame. Only used (and required) if the model was fitted using them.  |
| type     | Prediction type. This can either be "prob" for probability estimates or "class" for classification in binary responses. Ignored for regression.   |
| ...      | Parameters supplied to <a href="#">predict.logicDT</a>  |
| ensemble | If the model was fitted using the inner validation approach, shall the prediction be constructed using the final validated ensemble (TRUE) or using the single final tree (FALSE)?  |
| leaves   | If four parameter logistic models were fitted for each leaf, shall they be used for the prediction ("4pl") or shall the constant leaf means be used ("constant")?   |
| models   | Which models of logicDT model fitted with genetic programming shall be used for prediction? "best" leads to the single best model in the final generation, "all" uses the average over the final generation and "n_models" uses the n_models best models. |
| n_models | How many models shall be used if models = "n_models" and genetic programming was employed?  |

### Value

A numeric vector of predictions. For binary outcomes, this is a vector with estimates for  $P(Y = 1 | X = x)$ .

---

|            |                                       |
|------------|---------------------------------------|
| refitTrees | <i>Refit the logic decision trees</i> |
|------------|---------------------------------------|

---

**Description**

Newly fit the decision trees in the logicDT model using the supplied tree control parameters. This is especially useful if, e.g., the model was initially trained without utilizing a continuous covariable or fitting linear models and now 4pL model shall be fitted.

**Usage**

```
refitTrees(model, tree_control)
```

**Arguments**

|              |   |
|--------------|---|
| model        | A fitted logicDT model  |
| tree_control | Tree control parameters. This object should be constructed using the function <a href="#">tree.control</a> . Alternatively, the old tree_control from model can be modified and specified here. |

**Value**

The logicDT model with newly fitted trees

---

|           |   |
|-----------|---|
| splitSNPs | <i>Split biallelic SNPs into binary variables</i> |
|-----------|---|

---

**Description**

This function takes a matrix or data frame of SNPs coded as 0, 1, 2 or 1, 2, 3 and returns a data frame with twice as many columns. SNPs are splitted into dominant and recessive modes, i.e., for a  $\text{SNP} \in \{0, 1, 2\}$ , two variables  $\text{SNP}_D = (\text{SNP} \neq 0)$  and  $\text{SNP}_R = (\text{SNP} = 2)$  are generated.

**Usage**

```
splitSNPs(data)
```

**Arguments**

|      |   |
|------|---|
| data | A matrix or data frame only consisting of SNPs to be splitted |
|------|---|

**Value**

A data frame of the splitted SNPs

---

|              |  |
|--------------|--|
| tree.control | <i>Control parameters for fitting decision trees</i> |
|--------------|--|

---

### Description

Configure the fitting process of individual decision trees.

### Usage

```
tree.control(
  nodesize = 10,
  split_criterion = "gini",
  alpha = 0.05,
  cp = 0.001,
  smoothing = "none",
  mtry = "none",
  covariable = "final_4pl"
)
```

### Arguments

|                 |  |
|-----------------|--|
| nodesize        | Minimum number of samples contained in a terminal node. This parameter ensures that enough samples are available for performing predictions which includes fitting 4pL models.   |
| split_criterion | Splitting criterion for deciding when and how to split. The default is "gini"/"mse" which utilizes the Gini splitting criterion for binary risk estimation tasks and the mean squared error as impurity measure in regression tasks. Alternatively, "4pl" can be used if a quantitative covariable is supplied and the parameter covariable is chosen such that 4pL model fitting is enabled, i.e., covariable = "final_4pl" or covariable = "full_4pl". A fast modeling alternative is given by "linear" which also requires the parameter covariable to be properly chosen, i.e., covariable = "final_linear" or covariable = "full_linear". |
| alpha           | Significance threshold for the likelihood ratio tests when using split_criterion = "4pl". Only splits that achieve a p-value smaller than alpha are eligible.  |
| cp              | Complexity parameter. This parameter determines by which amount the impurity has to be reduced to further split a node. Here, the total tree impurity is considered. See details for a concrete formula. Only used if split_criterion = "gini" or "mse".   |
| smoothing       | Shall the leaf predictions for risk estimation be smoothed? "laplace" yields Laplace smoothing. The default is "none" which does not employ smoothing.#  |
| mtry            | Shall the tree fitting process be randomized as in random forests? Currently, only "sqrt" for using $\sqrt{p}$ random predictors at each node for splitting and "none" (default) for fitting conventional decision trees are supported.  |

**covariable** How shall optional quantitative covariables be handled? "constant" ignores them. Alternatively, they can be considered as splitting variables ("`_split`"), used for fitting 4pL models in each leaf ("`_4pl`"), or used for fitting linear models in each leaf ("`_linear`"). If either splitting or model fitting is chosen, one should state if this should be handled over the whole search ("`_full_`", computationally expensive) or just the final trees ("`_final_`"). Thus, "`_final_4pl`" would lead to fitting 4pL in each leaf but only for the final fitting of trees.

### Details

For the Gini or MSE splitting criterion, if any considered split  $s$  leads to

$$P(t) \cdot \Delta I(s, t) > cp$$

for a node  $t$ , the empirical node probability  $P(t)$  and the impurity reduction  $\Delta I(s, t)$ , then the node is further splitted. If not, the node is declared as a leaf. For continuous outcomes, `cp` will be scaled by the empirical variance of  $y$  to ensure the right scaling, i.e., `cp <- cp * var(y)`. Since the impurity measure for continuous outcomes is the mean squared error, this can be interpreted as controlling the minimum reduction of the normalized mean squared error (NRMSE to the power of two).

If one chooses the 4pL or linear splitting criterion, likelihood ratio tests testing the alternative of better fitting individual models are employed. The corresponding test statistic asymptotically follows a  $\chi^2$  distribution where the degrees of freedom are given by the difference in the number of model parameters, i.e., leading to  $2 \cdot 4 - 4 = 4$  degrees of freedom in the case of 4pL models and to  $2 \cdot 2 - 2 = 2$  degrees of freedom in the case of linear models.

For binary outcomes, choosing to fit linear models for evaluating the splits or for modeling the leaves actually leads to fitting LDA (linear discriminant analysis) models.

### Value

An object of class `tree.control` which is a list of all necessary tree parameters.

---

`vim` *Variable Importance Measures (VIMs)*

---

### Description

Calculate variable importance measures (VIMs) based on different approaches.

### Usage

```
vim(
  model,
  scoring_rule = "auc",
  vim_type = "logic",
  adjust = TRUE,
  interaction_order = 3,
  nodesize = NULL,
```



```

    alpha = 0.05,
    X_oob = NULL,
    y_oob = NULL,
    Z_oob = NULL,
    leaves = "4pl",
    ...
)

```

## Arguments

|                   |  |
|-------------------|--|
| model             | The fitted <code>logicDT</code> or <code>logic.bagged</code> model   |
| scoring_rule      | The scoring rule for assessing the model performance. As in <code>logicDT</code> , "auc", "nce", "deviance" and "brier" are possible for binary outcomes. For regression, the mean squared error is used.  |
| vim_type          | The type of VIM to be calculated. This can either be "logic", "remove" or "permutation". See below for details.  |
| adjust            | Shall adjusted interaction VIMs be additionally (to the VIMs of identified terms) computed? See below for details.   |
| interaction_order | If <code>adjust = TRUE</code> , up to which interaction order shall adjusted interaction VIMs be computed?   |
| nodesize          | If <code>adjust = TRUE</code> , how many observations need to be discriminated by an interaction in order to being considered? Similar to <code>conjsize</code> in <code>logicDT</code> and <code>nodesize</code> in <code>tree.control</code> .   |
| alpha             | If <code>adjust = TRUE</code> , a further adjustment can be performed trying to identify the concrete conjunctions responsible for the interaction of the considered binary predictors. <code>alpha</code> specifies the significance level for statistical tests testing the alternative of a difference in the response for specific conjunctions. <code>alpha = 0</code> leads to no further adjustment. See below for details.   |
| X_oob             | The predictor data which should be used for calculating the VIMs. Preferably some type of validation data independent of the training data.  |
| y_oob             | The outcome data for computing the VIMs. Preferably some type of validation data independent of the training data.   |
| Z_oob             | The optional covariable data for computing the VIMs. Preferably some type of validation data independent of the training data.   |
| leaves            | The prediction mode if 4pL models were fitted in the leaves. As in <code>predict.logicDT</code> , "4pl" and "constant" are the possible settings.  |
| ...               | Parameters passed to the different VIM type functions. For <code>vim_type = "logic"</code> , the argument <code>average</code> can be specified as "before" or "after". For <code>vim_type = "permutation"</code> , <code>n.perm</code> can be set to the number of random permutations. See below for details. For <code>vim_type = "remove"</code> , <code>empty.model</code> can be specified as either "none" ignoring empty models with all predictive terms removed or "mean" using the response mean as prediction in the case of an empty model. |

## Details

Three different VIM methods are implemented:

- Permutation VIMs: Random permutations of the respective identified logic terms
- Removal VIMs: Removing single logic terms
- Logic VIMs: Prediction with both possible outcomes of a logic term

Details on the calculation of these VIMs are given below.

By variable importance, importance of identified logic terms is meant. These terms can also be single predictors but also conjunctions in the spirit of this software package.

## Value

A data frame with two columns:

|     |   |
|-----|---|
| var | Short descriptions of the terms for which the importance was measured. For example $-X_1 \wedge X_2$ for $X_1^c \wedge X_2$ . |
| vim | The actual calculated VIM values.   |

The rows of such a data frame are sorted decreasingly by the VIM values.

## Permutation VIMs

Permutation VIMs are computed by comparing the the model's performance using the original data and data with random permutations of single terms. This approach was originally proposed by Breiman & Cutler (2003).

## Removal VIMs

Removal VIMs are constructed removing specific logic term from the set of predictors, refitting the decision tree and comparing the performance to the original model. Thus, this approach requires that at least two terms were found by the algorithm. Therefore, no VIM will be calculated if `empty.model = "none"` was specified. Alternatively, `empty.model = "mean"` can be set to use the constant mean response model for approximating the empty model.

## Logic VIMs

Logic VIMs use the fact that Boolean conjunctions are Boolean variables themselves and therefore are equal to 0 or 1. To compute the VIM for a specific term, predictions are performed once for this term fixed to 0 and once for this term fixed to 1. Then, the arithmetic mean of these two (risk or regression) predictions is used for calculating the performance. This performance is then compared to the original one as in the other VIM approaches (`average = "before"`). Alternatively, predictions for each fixed 0-1 scenario of the considered term can be performed leading to individual performances which then are averaged and compared to the original performance (`average = "after"`).

### Validation

Validation data sets which were not used in the fitting of the model are preferred preventing an overfitting of the VIMs themselves. These should be specified by the `_oob` arguments, if neither bagging nor inner validation was used for fitting the model.

### Bagging

For the bagging version, out of bag (OOB) data are naturally used for the calculation of VIMs.

### VIM Adjustment for Interactions

Since decision trees can naturally include interactions between single predictors (especially when strong marginal effects are present as well), logicDT models might, e.g., include the single input variables  $X_1$  and  $X_2$  but not their interaction  $X_1 \wedge X_2$  although an interaction effect is present. We, therefore, developed and implemented an adjustment approach for calculating VIMs for such unidentified interactions nonetheless. For predictors  $X_{i_1}, \dots, X_{i_k} =: Z$ , this interaction importance is given by

$$\text{VIM}(X_{i_1} \wedge \dots \wedge X_{i_k}) = \text{VIM}(X_{i_1}, \dots, X_{i_k} \mid X \setminus Z) - \sum_{\{j_1, \dots, j_l\} \subsetneq \{i_1, \dots, i_k\}} \text{VIM}(X_{j_1} \wedge \dots \wedge X_{j_l} \mid X \setminus Z)$$

and can basically be applied to all black-box models. By  $\text{VIM}(A \mid X \setminus Z)$ , the VIM of  $A$  considering the predictor set excluding the variables in  $Z$  is meant, i.e., the improvement of additionally considering  $A$  while regarding only the predictors in  $X \setminus Z$ . The proposed interaction VIM can be recursively calculated through

$$\text{VIM}(X_{i_1} \wedge X_{i_2}) = \text{VIM}(X_{i_1}, X_{i_2} \mid X \setminus Z) - \text{VIM}(X_{i_1} \mid X \setminus Z) - \text{VIM}(X_{i_2} \mid X \setminus Z)$$

for  $Z = X_{i_1}, X_{i_2}$ . This leads to the relationship

$$\text{VIM}(X_{i_1} \wedge \dots \wedge X_{i_k}) = \sum_{\{j_1, \dots, j_l\} \subsetneq \{i_1, \dots, i_k\}} (-1)^{k-l} \cdot \text{VIM}(X_{j_1}, \dots, X_{j_l} \mid X \setminus Z).$$

### Identification of Concrete Conjunctions

The aforementioned VIM adjustment approach only captures the importance of a general definition of interactions, i.e., it just considers the question whether some variables do interact in any way. Since logicDT is aimed at identifying specific conjunctions (and also assigns them VIMs if they were identified by `logicDT`), a further adjustment approach is implemented which tries to identify the specific conjunction leading to an interaction effect. The idea of this method is to consider the response for each possible scenario of the interacting variables, e.g., for  $X_1 \wedge (X_2^c \wedge X_3)$  where the second term  $X_2^c \wedge X_3$  was identified by `logicDT` and, thus, two interacting terms are regarded, the  $2^2 = 4$  possible scenarios  $\{(i, j) \mid i, j \in \{0, 1\}\}$  are considered. For each setting, the corresponding response is compared with outcome values of the complementary set. For continuous outcomes, a two sample t-test (with Welch correction for potentially unequal variances) is performed comparing the means between these two groups. For binary outcomes, Fisher's exact test is performed testing different underlying case probabilities. If at least one test rejects the null hypothesis of equal outcomes (without adjusting for multiple testing), the combination with the lowest p-value is chosen as the explanatory term for the interaction effect. For example, if the most significant deviation results from  $X_1 = 0$  and  $(X_2^c \wedge X_3) = 1$  from the example above, the term  $X_1^c \wedge (X_2^c \wedge X_3)$  is chosen.

**References**

- Breiman, L. (2001). Random Forests. *Machine Learning* 45(1):5-32. doi: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324)
- Breiman, L. & Cutler, A. (2003). Manual on Setting Up, Using, and Understanding Random Forests V4.0. University of California, Berkeley, Department of Statistics. [https://www.stat.berkeley.edu/~breiman/Using\\_random\\_forests\\_v4.0.pdf](https://www.stat.berkeley.edu/~breiman/Using_random_forests_v4.0.pdf)

# Index

bestBoostingIter, 2

calcAUC, 3  
calcBrier, 4  
calcDev, 4  
calcMis, 5  
calcMSE, 5  
calcNCE, 6  
calcNRMSE, 7  
cooling.schedule, 7, 18

fancy.plot (plot.logicDT), 25  
fit4plModel, 9  
fitLinearBoostingModel, 10, 15, 16  
fitLinearLogicModel, 11, 15, 16  
fitLinearModel, 11

getDesignMatrix, 12  
gxe.test, 13  
gxe.test.boosting, 15

importance.test.boosting, 16

logicDT, 11, 13, 14, 17, 20, 22, 24, 29, 33, 35  
logicDT.bagging, 22, 23  
logicDT.boosting, 23, 24  
logicDT.default, 19

partial.predict, 24  
plot.logicDT, 25  
plot.vim, 26  
predict.4pl, 26  
predict.geneticLogicPET  
    (predict.logicDT), 28  
predict.linear, 27  
predict.linear.logic, 28  
predict.logic.bagged (predict.logicDT),  
    28  
predict.logic.boosted  
    (predict.logicDT), 28  
predict.logicDT, 24, 28, 29, 33

refitTrees, 30

splitSNPs, 30

tree.control, 18, 30, 31, 33

vim, 26, 32