

Package ‘rbooster’

July 23, 2025

Type Package

Title AdaBoost Framework for Any Classifier

Version 1.1.0

Description This is a simple package which provides a function that boosts pre-ready or custom-made classifiers. Package uses Discrete AdaBoost (<[doi:10.1006/jcss.1997.1504](https://doi.org/10.1006/jcss.1997.1504)>) and Real AdaBoost (<[doi:10.1214/aos/1016218223](https://doi.org/10.1214/aos/1016218223)>) for two class, SAMME (<[doi:10.4310/SII.2009.v2.n3.a8](https://doi.org/10.4310/SII.2009.v2.n3.a8)>) and SAMME.R (<[doi:10.4310/SII.2009.v2.n3.a8](https://doi.org/10.4310/SII.2009.v2.n3.a8)>) for multiclass classification.

Depends R (> 4.0.4)

Imports stats, rpart, earth, Hmisc

Suggests knitr, imbalance, rmarkdown, mlbench

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Fatih Saglam [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-2084-2008>>),
Hasan Bulut [ctb]

Maintainer Fatih Saglam <fatih.saglam@omu.edu.tr>

Repository CRAN

Date/Publication 2021-10-27 12:00:05 UTC

Contents

booster	2
discretize	8

predict.booster	9
predict.w_naive_bayes	10
w_naive_bayes	11

Index	15
--------------	-----------

booster	<i>AdaBoost Framework for Any Classifier</i>
---------	--

Description

This function allows you to use any classifier to be used in Discrete or Real AdaBoost framework.

Usage

```
booster(
  x_train,
  y_train,
  classifier = "rpart",
  predictor = NULL,
  method = "discrete",
  x_test = NULL,
  y_test = NULL,
  weighted_bootstrap = FALSE,
  max_iter = 50,
  lambda = 1,
  print_detail = TRUE,
  print_plot = FALSE,
  bag_frac = 0.5,
  p_weak = NULL,
  ...
)
```

```
discrete_adaboost(
  x_train,
  y_train,
  classifier = "rpart",
  predictor = NULL,
  x_test = NULL,
  y_test = NULL,
  weighted_bootstrap = FALSE,
  max_iter = 50,
  lambda = 1,
  print_detail = TRUE,
  print_plot = FALSE,
  bag_frac = 0.5,
  p_weak = NULL,
  ...
)
```

```

)

real_adaboost(
  x_train,
  y_train,
  classifier = "rpart",
  predictor = NULL,
  x_test = NULL,
  y_test = NULL,
  weighted_bootstrap = FALSE,
  max_iter = 50,
  lambda = 1,
  print_detail = TRUE,
  print_plot = FALSE,
  bag_frac = 0.5,
  p_weak = NULL,
  ...
)

```

Arguments

<code>x_train</code>	feature matrix.
<code>y_train</code>	a factor class variable. Boosting algorithm allows for $k \geq 2$. However, not all classifiers are capable of multiclass classification.
<code>classifier</code>	pre-ready or a custom classifier function. Pre-ready classifiers are "rpart", "glm", "gnb", "dnn", "earth".
<code>predictor</code>	prediction function for classifier. It's output must be a factor variable with the same levels of <code>y_train</code>
<code>method</code>	"discrete" or "real" for Discrete or Real Adaboost.
<code>x_test</code>	optional test feature matrix. Can be used instead of predict function. <code>print_detail</code> and <code>print_plot</code> gives information about test.
<code>y_test</code>	optional a factor test class variable with the same levels as <code>y_train</code> . Can be used instead of predict function. <code>print_detail</code> and <code>print_plot</code> gives information about test.
<code>weighted_bootstrap</code>	If classifier does not support case weights, <code>weighted_bootstrap</code> must be TRUE used for weighting. If classifier supports weights, it must be FALSE. default is FALSE.
<code>max_iter</code>	maximum number of iterations. Default to 30. Probably should be higher for classifiers other than decision tree.
<code>lambda</code>	a parameter for model weights. Default to 1. Higher values leads to unstable weak classifiers, which is good sometimes. Lower values leads to slower fitting.
<code>print_detail</code>	a logical for printing errors for each iteration. Default to TRUE
<code>print_plot</code>	a logical for plotting errors. Default to FALSE.

<code>bag_frac</code>	a value between 0 and 1. It represents the proportion of cases to be used in each iteration. Smaller datasets may be better to create weaker classifiers. 1 means all cases. Default to 0.5. Ignored if <code>weighted_bootstrap == TRUE</code> .
<code>p_weak</code>	number of variables to use in weak classifiers. It is the number of columns in <code>x_train</code> by default. Lower values lead to weaker classifiers.
<code>...</code>	additional arguments for classifier and predictor functions. weak classifiers.

Details

method can be "discrete" and "real" at the moment and indicates Discrete AdaBoost and Real AdaBoost. For multiclass classification, "discrete" means SAMME, "real" means SAMME.R algorithm.

Pre-ready classifiers are "rpart", "glm", "dmb", "gnb", "earth", which means CART, logistic regression, Gaussian naive bayes, discrete naive bayes and MARS classifier respectively.

`predictor` is valid only if a custom classifier function is given. A custom classifier function should be as `function(x_train, y_train, weights, ...)` and its output is a model object which can be placed in `predictor`. `predictor` function is `function(model, x_new, type ...)` and its output must be a vector of class predictions. `type` must be "pred" or "prob", which gives a vector of classes or a matrix of probabilities, which each column represents each class. See `vignette("booster", package = "booster")` for examples.

`lambda` is a multiplier of model weights.

`weighted_bootstrap` is for bootstrap sampling in each step. If the classifier accepts case weights then it is better to turn it off. If classifier does not accept case weights, then weighted bootstrap will make it into weighted classifier using bootstrap. Learning may be slower this way.

`bag_frac` helps a classifier to be "weaker" by reducing sample size. Stronger classifiers may require lower proportions of `bag_frac`. `p_weak` does the same by reducing number of variables.

Value

a booster object with below components.

<code>n_train</code>	Number of cases in the input dataset.
<code>w</code>	Case weights for the final boost.
<code>p</code>	Number of features.
<code>weighted_bootstrap</code>	TRUE if weighted bootstrap applied. Otherwise FALSE.
<code>max_iter</code>	Maximum number of boosting steps.
<code>lambda</code>	The multiplier of model weights.
<code>predictor</code>	Function for prediction
<code>alpha</code>	Model weights.
<code>err_train</code>	A vector of train errors in each step of boosting.
<code>err_test</code>	A vector of test errors in each step of boosting. If there are no test data, it returns NULL
<code>models</code>	Models obtained in each boosting step

<code>x_classes</code>	A list of datasets, which are <code>x_train</code> separated for each class.
<code>n_classes</code>	Number of cases for each class in input dataset.
<code>k_classes</code>	Number of classes in class variable.
<code>bag_frac</code>	Proportion of input dataset used in each boosting step.
<code>class_names</code>	Names of classes in class variable.

Author(s)

Fatih Saglam, fatih.saglam@omu.edu.tr

References

- Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119-139.
- Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class AdaBoost. *Statistics and its Interface*, 2(3), 349-360.

See Also

`predict.booster`

Examples

```
require(rbooster)
## n number of cases, p number of variables, k number of classes.
cv_sampler <- function(y, train_proportion) {
  unlist(lapply(unique(y), function(m) sample(which(y==m), round(sum(y==m))*train_proportion)))
}

data_simulation <- function(n, p, k, train_proportion){
  means <- seq(0, k*2.5, length.out = k)
  x <- do.call(rbind, lapply(means,
                             function(m) matrix(data = rnorm(n = round(n/k)*p,
                                                             mean = m,
                                                             sd = 2),
                                                             nrow = round(n/k))))
  y <- factor(rep(letters[1:k], each = round(n/k)))
  train_i <- cv_sampler(y, train_proportion)

  data <- data.frame(x, y = y)
  data_train <- data[train_i,]
  data_test <- data[-train_i,]
  return(list(data = data,
              data_train = data_train,
              data_test = data_test))
}

### binary classification
dat <- data_simulation(n = 500, p = 2, k = 2, train_proportion = 0.8)

mm <- booster(x_train = dat$data_train[,1:2],
```

```

        y_train = dat$data_train[,3],
        classifier = "rpart",
        method = "discrete",
        x_test = dat$data_test[,1:2],
        y_test = dat$data_test[,3],
        weighted_bootstrap = FALSE,
        max_iter = 100,
        lambda = 1,
        print_detail = TRUE,
        print_plot = TRUE,
        bag_frac = 1,
        p_weak = 2)

## test prediction
mm$test_prediction
## or
pp <- predict(object = mm, newdata = dat$data_test[,1:2], type = "pred")
## test error
tail(mm$err_test, 1)
sum(dat$data_test[,3] != pp)/nrow(dat$data_test)

### multiclass classification
dat <- data_simulation(n = 800, p = 5, k = 3, train_proportion = 0.8)

mm <- booster(x_train = dat$data_train[,1:5],
              y_train = dat$data_train[,6],
              classifier = "rpart",
              method = "real",
              x_test = dat$data_test[,1:5],
              y_test = dat$data_test[,6],
              weighted_bootstrap = FALSE,
              max_iter = 100,
              lambda = 1,
              print_detail = TRUE,
              print_plot = TRUE,
              bag_frac = 1,
              p_weak = 2)

## test prediction
mm$test_prediction
## or
pp <- predict(object = mm, newdata = dat$data_test[,1:5], type = "pred", print_detail = TRUE)
## test error
tail(mm$err_test, 1)
sum(dat$data_test[,6] != pp)/nrow(dat$data_test)

### binary classification, custom classifier
dat <- data_simulation(n = 500, p = 10, k = 2, train_proportion = 0.8)
x <- dat$data[,1:10]
y <- dat$data[,11]

x_train <- dat$data_train[,1:10]
y_train <- dat$data_train[,11]

```

```

x_test <- dat$data_test[,1:10]
y_test <- dat$data_test[,11]

## a custom regression classifier function
classifier_lm <- function(x_train, y_train, weights, ...){
  y_train_code <- c(-1,1)
  y_train_coded <- sapply(levels(y_train), function(m) y_train_code[(y_train == m) + 1])
  y_train_coded <- y_train_coded[,1]

  model <- lm.wfit(x = as.matrix(cbind(1,x_train)), y = y_train_coded, w = weights)
  return(list(coefficients = model$coefficients,
              levels = levels(y_train)))
}

## predictor function

predictor_lm <- function(model, x_new, type = "pred", ...) {
  coef <- model$coefficients
  levels <- model$levels

  fit <- as.matrix(cbind(1, x_new))%*%coef
  probs <- 1/(1 + exp(-fit))
  probs <- data.frame(probs, 1 - probs)
  colnames(probs) <- levels

  if (type == "pred") {
    preds <- factor(levels[apply(probs, 1, which.max)], levels = levels, labels = levels)
    return(preds)
  }
  if (type == "prob") {
    return(probs)
  }
}

## real AdaBoost
mm <- booster(x_train = x_train,
              y_train = y_train,
              classifier = classifier_lm,
              predictor = predictor_lm,
              method = "real",
              x_test = x_test,
              y_test = y_test,
              weighted_bootstrap = FALSE,
              max_iter = 50,
              lambda = 1,
              print_detail = TRUE,
              print_plot = TRUE,
              bag_frac = 0.5,
              p_weak = 2)

## test prediction
mm$test_prediction

```

```

pp <- predict(object = mm, newdata = x_test, type = "pred", print_detail = TRUE)
## test error
tail(mm$err_test, 1)
sum(y_test != pp)/nrow(x_test)

## discrete AdaBoost
mm <- booster(x_train = x_train,
              y_train = y_train,
              classifier = classifier_lm,
              predictor = predictor_lm,
              method = "discrete",
              x_test = x_test,
              y_test = y_test,
              weighted_bootstrap = FALSE,
              max_iter = 50,
              lambda = 1,
              print_detail = TRUE,
              print_plot = TRUE,
              bag_frac = 0.5,
              p_weak = 2)

## test prediction
mm$test_prediction
pp <- predict(object = mm, newdata = x_test, type = "pred", print_detail = TRUE)
## test error
tail(mm$err_test, 1)
sum(y_test != pp)/nrow(x_test)

# plot function can be used to plot errors
plot(mm)

# more examples are in vignette("booster", package = "rbooster")

```

discretize

Discretize

Description

Discretizes numeric variables

Usage

```
discretize(xx, breaks = 3, boundaries = NULL, categories = NULL, w = NULL)
```

Arguments

xx	matrix or data.frame whose variables needs to be discretized.
breaks	number of categories for each variable. Ignored if boundaries != NULL.

boundaries	user-defined upper and lower limit matrix of discretization for each variable. Default is NULL.
categories	user-defined category names for each variable. Default is NULL.
w	sample weights for quantile calculation.

Details

Uses quantiles for discretization. However, quantiles may be equal in some cases. Then equal interval discretization used instead.

Value

a list consists of:

x_discrete	data.frame of discretized variables. Each variable is a factor.
boundaries	upper and lower limit matrix of discretization for each variable.
categories	category names for each variable.

Author(s)

Fatih Saglam, fatih.saglam@omu.edu.tr

predict.booster	<i>Prediction function for Adaboost framework</i>
-----------------	---

Description

Makes predictions based on booster function

Usage

```
## S3 method for class 'booster'
predict(object, newdata, type = "pred", print_detail = FALSE, ...)

## S3 method for class 'discrete_adaboost'
predict(object, newdata, type = "pred", print_detail = FALSE, ...)

## S3 method for class 'real_adaboost'
predict(object, newdata, type = "pred", print_detail = FALSE, ...)
```

Arguments

object	booster object
newdata	a factor class variable. Boosting algorithm allows for pre-ready or a custom classifier function.
type	prints the prediction process. Default is FALSE.
print_detail	additional arguments.
...	

Details

Type "pred" will give class predictions. "prob" will give probabilities for each class.

Value

A vector of class predictions or a matrix of class probabilities depending of type

See Also

[predict()]

predict.w_naive_bayes *Predict Discrete Naive Bayes*

Description

Function for Naive Bayes algorithm prediction.

Usage

```
## S3 method for class 'w_naive_bayes'
predict(object, newdata = NULL, type = "prob", ...)

## S3 method for class 'w_discrete_naive_bayes'
predict(object, newdata, type = "prob", ...)

## S3 method for class 'w_gaussian_naive_bayes'
predict(object, newdata = NULL, type = "prob", ...)
```

Arguments

object	"w_bayes" class object..
newdata	new observations which predictions will be made on.
type	"pred" or "prob".
...	additional arguments.

Details

Calls predict.w_discrete_naive_bayes or predict.w_gaussian_naive_bayes accordingly

Type "pred" will give class predictions. "prob" will give probabilities for each class.

Value

A vector of class predictions or a matrix of class probabilities depending of type

See Also

[predict()], [rbooster::predict.w_discrete_naive_bayes()], [rbooster::predict.w_gaussian_naive_bayes()]

w_naive_bayes

Naive Bayes algorithm with case weights

Description

Function for Naive Bayes algorithm classification with case weights.

Usage

```
w_naive_bayes(x_train, y_train, w = NULL, discretize = TRUE, breaks = 3)
```

```
w_gaussian_naive_bayes(x_train, y_train, w = NULL)
```

```
w_discrete_naive_bayes(x_train, y_train, breaks = 3, w = NULL)
```

Arguments

x_train	explanatory variables.
y_train	a factor class variable.
w	a vector of case weights.
discretize	If TRUE numerical variables are discretized and discrete naive bayes is applied,
breaks	number of break points for discretization. Ignored if discretize = TRUE.

Details

w_naive_bayes calls w_gaussian_naive_bayes or w_discrete_naive_bayes.

if discrete = FALSE, w_gaussian_naive_bayes is called. It uses Gaussian densities with case weights and allows multiclass classification.

if discrete = TRUE, w_discrete_naive_bayes is called. It uses conditional probabilities for each category with laplace smoothing and allows multiclass classification.

Value

a w_naive_bayes object with below components.

n_train	Number of cases in the input dataset.
p	Number of explanatory variables.
x_classes	A list of datasets, which are x_train separated for each class.
n_classes	Number of cases for each class in input dataset.
k_classes	Number of classes in class variable.
priors	Prior probabilities.
class_names	Names of classes in class variable.
means	Weighted mean estimations for each variable.

stds	Weighted standart deviation estimations for each variable.
categories	Labels for discretized variables.
boundaries	Upper and lower boundaries for discretization.
ps	probabilities for each variable categories.

Examples

```
library(rbooster)
## short functions for cross-validation and data simulation
cv_sampler <- function(y, train_proportion) {
  unlist(lapply(unique(y), function(m) sample(which(y==m), round(sum(y==m))*train_proportion)))
}

data_simulation <- function(n, p, k, train_proportion){
  means <- seq(0, k*1.5, length.out = k)
  x <- do.call(rbind, lapply(means,
                             function(m) matrix(data = rnorm(n = round(n/k)*p,
                                                             mean = m,
                                                             sd = 2),
                                                             nrow = round(n/k))))
  y <- factor(rep(letters[1:k], each = round(n/k)))
  train_i <- cv_sampler(y, train_proportion)

  data <- data.frame(x, y = y)
  data_train <- data[train_i,]
  data_test <- data[-train_i,]
  return(list(data = data,
              data_train = data_train,
              data_test = data_test))
}

### binary classification example
n <- 500
p <- 10
k <- 2
dat <- data_simulation(n = n, p = p, k = k, train_proportion = 0.8)
x <- dat$data[,1:p]
y <- dat$data[,p+1]

x_train <- dat$data_train[,1:p]
y_train <- dat$data_train[,p+1]

x_test <- dat$data_test[,1:p]
y_test <- dat$data_test[,p+1]

## discretized Naive Bayes classification
mm1 <- w_naive_bayes(x_train = x_train, y_train = y_train, discretize = TRUE, breaks = 4)
preds1 <- predict(object = mm1, newdata = x_test, type = "pred")
table(y_test, preds1)
# or
mm2 <- w_discrete_naive_bayes(x_train = x_train, y_train = y_train, breaks = 4)
preds2 <- predict(object = mm2, newdata = x_test, type = "pred")
```

```

table(y_test, preds2)

## Gaussian Naive Bayes classification
mm3 <- w_naive_bayes(x_train = x_train, y_train = y_train, discretize = FALSE)
preds3 <- predict(object = mm3, newdata = x_test, type = "pred")
table(y_test, preds3)

#or
mm4 <- w_gaussian_naive_bayes(x_train = x_train, y_train = y_train)
preds4 <- predict(object = mm4, newdata = x_test, type = "pred")
table(y_test, preds4)

## multiclass example
n <- 500
p <- 10
k <- 5
dat <- data_simulation(n = n, p = p, k = k, train_proportion = 0.8)
x <- dat$data[,1:p]
y <- dat$data[,p+1]

x_train <- dat$data_train[,1:p]
y_train <- dat$data_train[,p+1]

x_test <- dat$data_test[,1:p]
y_test <- dat$data_test[,p+1]

# discretized
mm5 <- w_discrete_naive_bayes(x_train = x_train, y_train = y_train, breaks = 4)
preds5 <- predict(object = mm5, newdata = x_test, type = "pred")
table(y_test, preds5)

# gaussian
mm6 <- w_gaussian_naive_bayes(x_train = x_train, y_train = y_train)
preds6 <- predict(object = mm6, newdata = x_test, type = "pred")
table(y_test, preds6)

## example for case weights
n <- 500
p <- 10
k <- 5
dat <- data_simulation(n = n, p = p, k = k, train_proportion = 0.8)
x <- dat$data[,1:p]
y <- dat$data[,p+1]

x_train <- dat$data_train[,1:p]
y_train <- dat$data_train[,p+1]

# discretized
weights <- ifelse(y_train == "a" | y_train == "c", 1, 0.01)

mm7 <- w_discrete_naive_bayes(x_train = x_train, y_train = y_train, breaks = 4, w = weights)

preds7 <- predict(object = mm7, newdata = x_test, type = "pred")

```

```
table(y_test, preds7)

# gaussian
weights <- ifelse(y_train == "b" | y_train == "d", 1, 0.01)

mm8 <- w_gaussian_naive_bayes(x_train = x_train, y_train = y_train, w = weights)

preds8 <- predict(object = mm8, newdata = x_test, type = "pred")
table(y_test, preds8)
```

Index

`booster`, [2](#)

`discrete_adaboost (booster)`, [2](#)

`discretize`, [8](#)

`predict.booster`, [9](#)

`predict.discrete_adaboost`
 (`predict.booster`), [9](#)

`predict.real_adaboost`
 (`predict.booster`), [9](#)

`predict.w_discrete_naive_bayes`
 (`predict.w_naive_bayes`), [10](#)

`predict.w_gaussian_naive_bayes`
 (`predict.w_naive_bayes`), [10](#)

`predict.w_naive_bayes`, [10](#)

`real_adaboost (booster)`, [2](#)

`w_discrete_naive_bayes (w_naive_bayes)`,
 [11](#)

`w_gaussian_naive_bayes (w_naive_bayes)`,
 [11](#)

`w_naive_bayes`, [11](#)