Package 'JOUSBoost'

October 12, 2022

Type Package

Title Implements Under/Oversampling for Probability Estimation

Version 2.1.0

Description Implements under/oversampling for probability estimation. To be used with machine learning methods such as AdaBoost, random forests, etc.

License MIT + file LICENSE

LazyData TRUE

Suggests testthat, knitr, rmarkdown

LinkingTo Rcpp

Depends R (>= 2.10)

Imports Rcpp, rpart, stats, doParallel, foreach

RoxygenNote 6.0.1

NeedsCompilation yes

Author Matthew Olson [aut, cre]

Maintainer Matthew Olson <maolson@wharton.upenn.edu>

Repository CRAN

Date/Publication 2017-07-12 19:13:02 UTC

R topics documented:

---

adaboost *AdaBoost Classifier*

Description

An implementation of the AdaBoost algorithm from Freund and Shapire (1997) applied to decision tree classifiers.

Usage

```
adaboost(X, y, tree_depth = 3, n_rounds = 100, verbose = FALSE,
  control = NULL)
```

Arguments

`X`	A matrix of continuous predictors.
`y`	A vector of responses with entries in `c(-1, 1)`.
`tree_depth`	The depth of the base tree classifier to use.
`n_rounds`	The number of rounds of boosting to use.
`verbose`	Whether to print the number of iterations.
`control`	A `rpart.control` list that controls properties of fitted decision trees.

Value

Returns an object of class `adaboost` containing the following values:

`alphas`	Weights computed in the adaboost fit.
`trees`	The trees constructed in each round of boosting. Storing trees allows one to make predictions on new data.
`confusion_matrix`	A confusion matrix for the in-sample fits.

Note

Trees are grown using the CART algorithm implemented in the `rpart` package. In order to conserve memory, the only parts of the fitted tree objects that are retained are those essential to making predictions. In practice, the number of rounds of boosting to use is chosen by cross-validation.

References

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of online learning and an application to boosting, Journal of Computer and System Sciences 55: 119-139.

Examples

```
## Not run:
# Generate data from the circle model
set.seed(111)
dat = circle_data(n = 500)
train_index = sample(1:500, 400)

ada = adaboost(dat$X[train_index,], dat$y[train_index], tree_depth = 2,
               n_rounds = 200, verbose = TRUE)
print(ada)
yhat_ada = predict(ada, dat$X[-train_index,])

# calculate misclassification rate
mean(dat$y[-train_index] != yhat_ada)

## End(Not run)
```

---

circle_data — *Simulate data from the circle model.*

Description

Simulate draws from a bernoulli distribution over `c(-1,1)`. First, the predictors x are drawn i.i.d. uniformly over the square in the two dimensional plane centered at the origin with side length `2*outer_r`, and then the response is drawn according to $p(y=1|x)$, which depends on $r(x)$, the euclidean norm of x. If $r(x) \leq inner_r$, then $p(y=1|x)=1$, if $r(x) \geq outer_r$ then $p(y=1|x)=1$, and $p(y=1|x)=(outer_r - r(x))/(outer_r - inner_r)$ when $inner_r <= r(x) <= outer_r$. See Mease (2008).

Usage

```
circle_data(n = 500, inner_r = 8, outer_r = 28)
```

Arguments

`n`	Number of points to simulate.
`inner_r`	Inner radius of annulus.
`outer_r`	Outer radius of annulus.

Value

Returns a list with the following components:

`y`	Vector of simulated response in `c(-1,1)`.
`X`	An `nx2` matrix of simulated predictors.
`p`	The true conditional probability $p(y=1|x)$.

References

Mease, D., Wyner, A. and Buha, A. (2007). Costweighted boosting with jittering and over/undersampling: JOUS-boost. J. Machine Learning Research 8 409-439.

Examples

```
# Generate data from the circle model
set.seed(111)
dat = circle_data(n = 500, inner_r = 1, outer_r = 5)

## Not run:
# Visualization of conditional probability p(y=1|x)
inner_r = 0.5
outer_r = 1.5
x = seq(-outer_r, outer_r, by=0.02)
radius = sqrt(outer(x^2, x^2, "+"))
prob = ifelse(radius >= outer_r, 0, ifelse(radius <= inner_r, 1,
             (outer_r-radius)/(outer_r-inner_r)))
image(x, x, prob, main='Probability Density: Circle Example')

## End(Not run)
```

`friedman_data` *Simulate data from the Friedman model*

Description

Simulate draws from a bernoulli distribution over `c(-1,1)`, where the log-odds is defined according to:

$$logp(y = 1|x)/p(y = -1|x) = gamma * (1 - x_1 + x_2 - ... + x_6) * (x_1 + x_2 + ... + x_6)$$

and x is distributed as N(0, I_dxd). See Friedman (2000).

Usage

```
friedman_data(n = 500, d = 10, gamma = 10)
```

Arguments

`n`	Number of points to simulate.
`d`	The dimension of the predictor variable x.
`gamma`	A parameter controlling the Bayes error, with higher values of gamma corresponding to lower error rates.

Value

Returns a list with the following components:

- `y` — Vector of simulated response in `c(-1,1)`.
- `X` — An `nxd` matrix of simulated predictors.
- `p` — The true conditional probability $p(y = 1|x)$.

References

Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion), Annals of Statistics 28: 337-307.

Examples

```
set.seed(111)
dat = friedman_data(n = 500, gamma = 0.5)
```

`grid_probs` *Function to compute predicted quantiles*

Description

Find predicted quantiles given classification results at different quantiles.

Usage

```
grid_probs(X, q, delta, median_loc)
```

Arguments

- `X` — Matrix of class predictions, where each column gives the predictions for a given quantile in q.
- `q` — The quantiles for which the columns of X are predictions.
- `delta` — The number of quantiles used.
- `median_loc` — Location of median quantile (0-based indexing).

index_over — *Return indices to be used for jittered data in oversampling*

Description

Return indices to be used for jittered data in oversampling

Usage

```
index_over(ix_pos, ix_neg, q)
```

Arguments

ix_pos	Indices for positive examples in data.
ix_neg	Indices for negative examples in data.
q	Quantiles for which to construct tilted datasets.

Value

returns a list, each of element of which gives indices to be used on a particular cut (note: will be of length delta - 1)

index_under — *Return indices to be used in original data for undersampling*

Description

(note: sampling is done without replacement)

Usage

```
index_under(ix_pos, ix_neg, q, delta)
```

Arguments

ix_pos	Indices for positive examples in data.
ix_neg	Indices for negative examples in data.
q	Quantiles for which to construct tilted datasets.
delta	Number of quantiles.

Value

returns a list, each of element of which gives indices to be used on a particular cut (note: will be of length delta - 1) jous *Jittering with Over/Under Sampling*

Description

Perform probability estimation using jittering with over or undersampling.

Usage

```
jous(X, y, class_func, pred_func, type = c("under", "over"), delta = 10,
  nu = 1, X_pred = NULL, keep_models = FALSE, verbose = FALSE,
  parallel = FALSE, packages = NULL)
```

Arguments

X	A matrix of continuous predictors.
y	A vector of responses with entries in `c(-1, 1)`.
class_func	Function to perform classification. This function definition must be exactly of the form `class_func(X, y)` where X is a matrix and y is a vector with entries in `c(-1, 1)`, and it must return an object on which `pred_func` can create predictions. See examples.
pred_func	Function to create predictions. This function definition must be exactly of the form `pred_func(fit_obj, X)` where `fit_obj` is an object returned by class_func and X is a matrix of new data values, and it must return a vector with entries in `c(-1, 1)`. See examples.
type	Type of sampling: "over" for oversampling, or "under" for undersampling.
delta	An integer (greater than 3) to control the number of quantiles to estimate:
nu	The amount of noise to apply to predictors when oversampling data. The noise level is controlled by `nu * sd(X[,j])` for each predictor - the default of `nu = 1` works well. Such "jittering" of the predictors is essential when applying `jous` to boosting type methods.
X_pred	A matrix of predictors for which to form probability estimates.
keep_models	Whether to store all of the models used to create the probability estimates. If `type=FALSE`, the user will need to re-run `jous` when creating probability estimates for test data.
verbose	If `TRUE`, print the function's progress to the terminal.
parallel	If `TRUE`, use parallel `foreach` to fit models. Must register parallel before hand, such as `doParallel`. See examples below.
packages	If `parallel = TRUE`, a vector of strings containing the names of any packages used in `class_func` or `pred_func`. See examples below.

Value

Returns a list containing information about the parameters used in the `jous` function call, as well as the following additional components:

- `q` — The vector of target quantiles estimated by `jous`. Note that the estimated probabilities will be located at the midpoints of the values in `q`.
- `phat_train` — The in-sample probability estimates $p(y = 1|x)$.
- `phat_test` — Probability estimates for the optional test data in `X_test`
- `models` — If `keep_models=TRUE`, a list of models fitted to the resampled data sets.
- `confusion_matrix` — A confusion matrix for the in-sample fits.

Note

The `jous` function runs the classifier `class_func` a total of `delta` times on the data, which can be computationally expensive. Also,`jous` cannot yet be applied to categorical predictors - in the oversampling case, it is not clear how to "jitter" a categorical variable.

References

Mease, D., Wyner, A. and Buja, A. (2007). Costweighted boosting with jittering and over/undersampling: JOUS-boost. J. Machine Learning Research 8 409-439.

Examples

```
## Not run:
# Generate data from Friedman model #
set.seed(111)
dat = friedman_data(n = 500, gamma = 0.5)
train_index = sample(1:500, 400)

# Apply jous to adaboost classifier
class_func = function(X, y) adaboost(X, y, tree_depth = 2, n_rounds = 200)
pred_func = function(fit_obj, X_test) predict(fit_obj, X_test)

jous_fit = jous(dat$X[train_index,], dat$y[train_index], class_func,
                pred_func, keep_models = TRUE)
# get probability
phat_jous = predict(jous_fit, dat$X[-train_index, ], type = "prob")

# compare with probability from AdaBoost
ada = adaboost(dat$X[train_index,], dat$y[train_index], tree_depth = 2,
               n_rounds = 200)
phat_ada = predict(ada, dat$X[train_index,], type = "prob")

mean((phat_jous - dat$p[-train_index])^2)
mean((phat_ada - dat$p[-train_index])^2)

## Example using parallel option
```

```
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)

# n.b. the packages='rpart' is not really needed here since it gets
# exported automatically by JOUSBoost, but for illustration
jous_fit = jous(dat$X[train_index,], dat$y[train_index], class_func,
                pred_func, keep_models = TRUE, parallel = TRUE,
                packages = 'rpart')
phat = predict(jous_fit, dat$X[-train_index,], type = 'prob')
stopCluster(cl)

## Example using SVM

library(kernlab)
class_func = function(X, y) ksvm(X, as.factor(y), kernel = 'rbfdot')
pred_func = function(obj, X) as.numeric(as.character(predict(obj, X)))
jous_obj = jous(dat$X[train_index,], dat$y[train_index], class_func = class_func,
           pred_func = pred_func, keep_models = TRUE)
jous_pred = predict(jous_obj, dat$X[-train_index,], type = 'prob')

## End(Not run)
```

`JOUSBoost` *JOUSBoost: A package for probability estimation*

Description

JOUSBoost implements under/oversampling with jittering for probability estimation. Its intent is to be used to improve probability estimates that come from boosting algorithms (such as AdaBoost), but is modular enough to be used with virtually any classification algorithm from machine learning.

Details

For more theoretical background, consult Mease (2007).

References

Mease, D., Wyner, A. and Buja, A. (2007). Costweighted boosting with jittering and over/undersampling: JOUS-boost. J. Machine Learning Research 8 409-439.

`predict.adaboost` *Create predictions from AdaBoost fit*

Description

Makes a prediction on new data for a given fitted adaboost model.

Usage

```
## S3 method for class 'adaboost'
predict(object, X, type = c("response", "prob"),
  n_tree = NULL, ...)
```

Arguments

object	An object of class adaboost returned by the adaboost function.
X	A design matrix of predictors.
type	The type of prediction to return. If type="response", a class label of -1 or 1 is returned. If type="prob", the probability $p(y = 1\|x)$ is returned.
n_tree	The number of trees to use in the prediction (by default, all them).
...	...

Value

Returns a vector of class predictions if type="response", or a vector of class probabilities $p(y = 1|x)$ if type="prob".

Note

Probabilities are estimated according to the formula:

$$p(y = 1|x) = 1/(1 + exp(-2 * f(x)))$$

where $f(x)$ is the score function produced by AdaBoost. See Friedman (2000).

References

Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion), Annals of Statistics 28: 337-307.

Examples

```
## Not run:
# Generate data from the circle model
set.seed(111)
dat = circle_data(n = 500)
train_index = sample(1:500, 400)
```

```
ada = adaboost(dat$X[train_index,], dat$y[train_index], tree_depth = 2,
               n_rounds = 100, verbose = TRUE)
# get class prediction
yhat = predict(ada, dat$X[-train_index, ])
# get probability estimate
phat = predict(ada, dat$X[-train_index, ], type="prob")

## End(Not run)
```

`predict.jous` *Create predictions*

Description

Makes a prediction on new data for a given fitted `jous` model.

Usage

```
## S3 method for class 'jous'
predict(object, X, type = c("response", "prob"), ...)
```

Arguments

`object`	An object of class `jous` returned by the `jous` function.
`X`	A design matrix of predictors.
`type`	The type of prediction to return. If `type="response"`, a class label of -1 or 1 is returned. If `type="prob"`, the probability $p(y = 1\|x)$ is returned.
`...`	...

Value

Returns a vector of class predictions if `type="response"`, or a vector of class probabilities $p(y = 1|x)$ if `type="prob"`.

Examples

```
## Not run:
# Generate data from Friedman model #
set.seed(111)
dat = friedman_data(n = 500, gamma = 0.5)
train_index = sample(1:500, 400)

# Apply jous to adaboost classifier
class_func = function(X, y) adaboost(X, y, tree_depth = 2, n_rounds = 100)
pred_func = function(fit_obj, X_test) predict(fit_obj, X_test)
```

```
jous_fit = jous(dat$X[train_index,], dat$y[train_index], class_func,
                pred_func, keep_models=TRUE)
# get class prediction
yhat = predict(jous_fit, dat$X[-train_index, ])
# get probability estimate
phat = predict(jous_fit, dat$X[-train_index, ], type="prob")

## End(Not run)
```

---

print.adaboost *Print a summary of adaboost fit.*

Description

Print a summary of adaboost fit.

Usage

```
## S3 method for class 'adaboost'
print(x, ...)
```

Arguments

x	An adaboost object fit using the adaboost function.
...	...

Value

Printed summary of the fit, including information about the tree depth and number of boosting rounds used.

---

print.jous *Print a summary of* jous *fit.*

Description

Print a summary of jous fit.

Usage

```
## S3 method for class 'jous'
print(x, ...)
```

Arguments

x	A jous object.
...	...

Value

Printed summary of the fit

`sonar` *Dataset of sonar measurements of rocks and mines*

Description

A dataset containing sonar measurements used to discriminate rocks from mines.

Usage

```
data(sonar)
```

Format

A data frame with 208 observations on 61 variables. The variables V1-V60 represent the energy within a certain frequency band, and are to be used as predictors. The variable y is a class label, 1 for 'rock' and -1 for 'mine'.

Source

http://archive.ics.uci.edu/ml/

References

Gorman, R. P., and Sejnowski, T. J. (1988). "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" in Neural Networks, Vol. 1, pp. 75-89.

Index