Package ‘MTPS’

June 14, 2023

Type Package

Title Multi-Task Prediction using Stacking Algorithms

Version 1.0.2

Description Simultaneous multiple outcomes prediction based on revised stacking algorithms, which enables the integration of information from predictions of individual models. An implementation of methodologies proposed in our paper: Li Xing, Mary L Lesperance, Xuekui Zhang. (2019) Bioinformatics, ‘Simultaneous prediction of multiple outcomes using revised stacking algorithms’ <doi:10.1093/bioinformatics/btz531>.

License GPL (>= 2)

URL https://doi.org/10.1093/bioinformatics/btz531

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), glmnet, rpart, MASS, e1071, class

Suggests knitr, rmarkdown, ggplot2, reshape2

VignetteBuilder knitr

NeedsCompilation no

Maintainer Li Xing <sfulxing@gmail.com>

RoxygenNote 7.2.3

Author Li Xing [aut, cre],
  Xiaowen Cao [aut],
  Yuying Huang [aut],
  Peijie Xie [ctb],
  Mary Lesperance [aut],
  Xuekui Zhang [aut]

Repository CRAN

Date/Publication 2023-06-14 05:20:02 UTC
Description

The AUC function calculates the numeric value of area under the ROC curve (AUC) with the trapezoidal rule and optionally plots the ROC curve.

Usage

AUC(prob, outcome, cutoff = 1, ROC.plot = FALSE)

Arguments

- prob: A numeric vector of predicted probability
- outcome: A numeric vector of observed binary outcome
- cutoff: Number between 0 and 1 to specify where threshold of ROC curve should be truncated. The default value is 1 (no truncation)
- ROC.plot: Logical. Whether or not to plot ROC curve

Details

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings.

By default the total area under the curve is computed, but a truncated AUC statistics can be specified with the cutoff argument. It specifies the bounds of FPR. The common choice of cutoff can be 1 (i.e. no truncate) or 0.2 (i.e. specificity > 0.8)

Value

The value of the area under the curve.
Examples

```r
set.seed(1)
# simulate predictors
x1 <- rnorm(200)
x2 <- rnorm(200)
# simulate outcome
pr <- 1/(1+exp(-(3 * x1 + 2 * x2 + 1)))
y <- rbinom(200, 1, pr)
df <- data.frame(y = y, x1 = x1, x2 = x2)
# fit logistic regression model on the first 100 observation
lg.model <- glm(y ~ x1 + x2, data = df[1:100, ], family="binomial")
# predict outcome for the last 100 observation
prob <- predict(lg.model, df[101:200, c("x1", "x2")], type = "response")
# calculate AUC and plot thr ROC Curve
AUC(prob, y[101:200], ROC=TRUE)
# calculate AUC and plot thr ROC Curve with cutoff
AUC(prob, y[101:200], cutoff=0.2, ROC=TRUE)
```

cv.MTPS

**Evaluation using Cross-Validation**

**Description**

Use cross-validation to evaluate model performance.

**Usage**

```r
cv.MTPS(xmat, ymat, family, nfolds = 5,
        cv = FALSE, residual = TRUE,
        cv.stacking.nfold = 5, method.step1, method.step2,
        resid.type=c("deviance", "pearson", "raw"),
        resid.std=FALSE)
```

**Arguments**

- `xmat`: Predictor matrix, each row is an observation vector
- `ymat`: Responses matrix. Quantitative for family = "gaussian" and a factor of two levels for family = "binomial"
- `family`: Response type for each response. If all response variable are within the same family it can be "gaussian" or "binomial", otherwise it is a vector with elements "gaussian" and "binomial" to indicate each response family
- `nfolds`: Integer, number of folds for Cross-Validation to evaluate the performance of stacking algorithms.
- `cv`: Logical, indicate if use Cross-Validation Stacking algorithm
- `residual`: Logical, indicate if use Residual Stacking algorithm
cv.stacking.nfold
  Integer, number of folds for Cross-Validation Stacking algorithm. The default value is 5

method.step1
  Base Learners for fitting models in Step 1 of Stacking Algorithm. It can be one base learner function for all outcomes or a list of base learner functions for each outcome. The list of all base learners can be obtained by `list.learners()`

method.step2
  Base Learners for fitting models in Step 2 of Stacking Algorithm. (see above)

resid.type
  The residual type for Residual Stacking

resid.std
  Logical, whether or not use standardized residual

Value
  It returns the mean squared error of continuous outcomes. AUC, accuracy, recall and precision for binary outcomes of predictions using cross-validation.

Examples
  ```
  data("HIV")
  cv.MTPS(xmat=XX, ymat=YY, family="gaussian", nfolds=2,
         method.step1=rpart1, method.step2=lm1)
  ```

---

### HIV

**HIV Drug Resistance Database**

**Description**

The data from HIV Drug Resistance Database used for demonstration. After processing, YY contains 5 response variables variable for 1246 observations and XX are 228 predictors of those 1246 observations.

**Format**

Data objects used for demonstration

**Details**

In the HIV database, the resistance of five Nucleoside RT Inhibitor (NRTI) drugs were used as multivariate outcomes, including Lamivudine (3TC), Abacavir (ABC), Zidovudine (AZT), Stavudine (D4T), Didanosine (DDI). The mutation variables are used as the predictors. Some mutation variables were removed as they do not contain enough variation. The final outcome data is a matrix of size $1246 \times 5$, and the predictor data is a matrix of $1246 \times 228$ values, which is provided in the package called "HIV". In the example data in the package, "YY" refers the outcome data and "XX" refers the predictor data.
References


Examples

data(HIV)

---

**Description**

The data is for internal use, and is not meant for users.

**Format**

Data objects used for demonstration

**Details**

For speeding up vignette build purpose.
Description

This function lists all base learners provided in the package.

Usage

list.learners()

Details

lm1: linear regression

glm1: generalized linear models

glmnet1: Does k-fold cross-validation to chose best alpha and lambda for generalized linear models via penalized maximum likelihood.

glmnet.lasso: LASSO, lambda is chose by k-fold cross-validation for glmnet

glmnet.ridge: Ridge regression, lambda is chose by k-fold cross-validation for glmnet

rpart1: regression tree

lda1: linear discriminant analysis

qda1: quadratic discriminant analysis

KNN1: k-nearest neighbour classification, k is chose by cross-validation

svm1: support vector machine

Value

The name of all base learners provided in the package

Examples

list.learners()
modify.parameter

Modify Default Parameters For Base Learner

Description
Modify default parameters for methods provided in the package.

Usage
modify.parameter(FUN, ...)

Arguments
FUN Method
...
Modified arguments

Value
It returns a new function with modified parameters.

Examples
glmnet.lasso <- modify.parameter(glmnet1, alpha=1)
glmnet.ridge <- modify.parameter(glmnet1, alpha=0)

MTPS
Fit Models using Revised Stacking Algorithm

Description
Fit a model using standard stacking algorithm or revised stacking algorithms to simultaneous predict multiple outcomes

Usage
MTPS(xmat, ymat, family,
cv = FALSE, residual = TRUE, nfold = 5,
method.step1, method.step2,
resid.type = c("deviance", "pearson", "raw"), resid.std = FALSE)
Arguments

- **xmat**: Predictor matrix, each row is an observation vector
- **ymat**: Responses matrix. Quantitative for family = "gaussian" and a factor of two levels for family = "binomial"
- **family**: Response type for each response. If all response variable are within the same family it can be "gaussian" or "binomial", otherwise it is a vector with elements "gaussian" and "binomial" to indicate each response family
- **cv**: Logical, indicate if use Cross-Validation Stacking algorithm
- **residual**: Logical, indicate if use Residual Stacking algorithm
- **nfold**: Integer, number of folds for Cross-Validation Stacking algorithm. The default value is 5
- **method.step1**: Base Learners for fitting models in Step 1 of Stacking Algorithm. It can be one base learner function for all outcomes or a list of base learner functions for each outcome. The list of all base learners can be obtained by `list.learners()
- **method.step2**: Base Learners for fitting models in Step 2 of Stacking Algorithm. (see above)
- **resid.type**: The residual type for Residual Stacking
- **resid.std**: Logical, whether or not use standardized residual

Value

It returns a MTPS object. It is a list of 4 parameters containing information about step 1 and step 2 models and the revised stacking algorithm method.

Examples

data("HIV")
set.seed(1)
xmat <- as.matrix(XX)
ymat <- as.matrix(YY)
id <- createFolds(rowMeans(XX), k=5, list=FALSE)
training.id <- id != 1
y.train <- ymat[training.id,]
y.test <- ymat[!training.id,]
x.train <- xmat[training.id,]
x.test <- xmat[!training.id,]

# Residual Stacking
fit.rs <- MTPS(xmat = x.train, ymat = y.train,
  family = "gaussian", cv = FALSE, residual = TRUE,
  method.step1 = rpart1, method.step2 = lm1)
predict(fit.rs, x.test)

# using different base learners for different outcomes
fit.mixOut <- MTPS(xmat=x.train, ymat=y.train,
  family="gaussian", cv = FALSE, residual = TRUE,
  method.step1 = c(rpart1,glmnet.ridge,rpart1,lm1,lm1),
  method.step2 = c(rpart1,lm1,lm1,glmnet.ridge))
predict(fit.mixOut, x.test)
multiFit

Fit models on multiple outcomes

Description

This function fits individual models to predict each outcome separately.

Usage

multiFit(xmat, ymat, method, family)

Arguments

- **xmat**: Matrix of predictors, each row is an observation vector
- **ymat**: Matrix of outcomes. Quantitative for family = "gaussian" and a factor of two levels for family = "binomial"
- **method**: Method for fitting models. It can be one base learner function for all outcomes or a list of base learner functions for each outcome. The list of all base learners can be obtained by list.learners().
- **family**: Response type for each response. If all response variables are within the same family it can be "gaussian" or "binomial", otherwise it is a vector of "gaussian" or "binomial" to indicate each response family.

Value

It returns a multiFit object. It is a list of 5 parameters containing information about the fitted models and fitted values for each outcome.

Examples

data("HIV")
set.seed(1)
xmat <- as.matrix(XX)
ymat <- as.matrix(YY)
id <- createFolds(rowMeans(XX), k=5, list=FALSE)
training.id <- id != 1
y.train <- ymat[training.id,]
y.test <- ymat[!training.id,]
x.train <- xmat[training.id,]
x.test <- xmat[!training.id,]
fit <- multiFit(xmat = x.train, ymat = y.train,
                method = rpart1, family = "gaussian")
predict(fit, x.test)

# using different base learners for different outcomes
fit.mixOut <- multiFit(xmat = x.train, ymat = y.train,
                        method = c(rpart1, rpart1, glmnet.ridge, lm1, lm1),
                        family = "gaussian")
predict(fit.mixOut, x.test)
predict.MTPS

Make predictions from a "MTPS" model

Description

This function makes predictions from a revised stacking model.

Usage

## S3 method for class 'MTPS'
predict(object, newdata, ...)

Arguments

object A fitted object from "MTPS"
newdata Matrix of new predictors at which predictions are to be made
... additional arguments affecting the predictions produced

Value

The predicted value from new predictors.

Examples

data("HIV")
set.seed(1)
xmat <- as.matrix(XX)
ymat <- as.matrix(YY)
id <- createFolds(rowMeans(XX), k=5, list=FALSE)
training.id <- id != 1
y.train <- ymat[training.id,]
y.test <- ymat[!training.id,]
x.train <- xmat[training.id,]
x.test <- xmat[!training.id,]
# Cross-Validation Residual Stacking
fit.rs <- MTPS(xmat = x.train, ymat = y.train,
               family = "gaussian", cv = FALSE, residual = TRUE,
               method.step1 = rpart1, method.step2 = lm1)
pred.rs <- predict(fit.rs, x.test)
predict.multiFit  
Make predictions for multiple outcomes

Description

This function makes predictions from a multiFit object.

Usage

```
## S3 method for class 'multiFit'
predict(object, newdata, ...)
```

Arguments

- `object`: A fitted object from "multiFit"
- `newdata`: Matrix of new predictors at which predictions are to be made
- `...`: additional arguments affecting the predictions produced

Value

The predicted value from new predictors.

Examples

```r
data("HIV")
set.seed(1)
xmat <- as.matrix(XX)
ymat <- as.matrix(YY)
id <- createFolds(rowMeans(XX), k=5, list=FALSE)
training.id <- id != 1
y.train <- ymat[training.id, ]
y.test <- ymat[!training.id, ]
x.train <- xmat[training.id, ]
x.test <- xmat[!training.id, ]
fit <- multiFit(xmat = x.train, ymat = y.train,
               method = rpart1, family = "gaussian")
predict(fit, x.test)
```
Index

* datasets
  HIV, 4
  Internal_data, 5

AUC, 2

ctn.plot.data(Internal_data), 5
cv.MTPS, 3

glm1(list.learners), 6
glmnet.lasso(list.learners), 6
glmnet.ridge(list.learners), 6
glmnet1(list.learners), 6

HIV, 4

Internal_data, 5

KNN1(list.learners), 6

lda1(list.learners), 6
list.learners, 6
lm1(list.learners), 6

mix.plot.data(Internal_data), 5
modify.parameter, 7
mse.data(Internal_data), 5
MTPS, 7
multiFit, 9

pred.mix.rs(Internal_data), 5
pred.prs.std(Internal_data), 5
predict.MTPS, 10
predict.multiFit, 11

qda1(list.learners), 6
rpart1(list.learners), 6

svm1(list.learners), 6

XX (HIV), 4

y.test.bin(Internal_data), 5
y.test.mix(Internal_data), 5
YY (HIV), 4