Package 'BAGofT'

September 14, 2021

Type Package

Title A Binary Regression Adaptive Goodness-of-Fit Test (BAGofT)

Version 1.0.0

Description The BAGofT assesses the goodness-of-fit of binary classifiers. Details can be found in Zhang, Ding and Yang (2021) <arXiv:1911.03063v2>.

Depends R (>= 3.6.0)

License GPL-3

Encoding UTF-8

Imports randomForest (>= 4.6.14), dcov (>= 0.1.1)

Suggests glmnet (>= 2.0.18), xgboost (>= 1.2.0.1)

RoxygenNote 7.1.1

NeedsCompilation no

Author Jiawei Zhang [aut, cre],
Jie Ding [aut],
Yuhong Yang [aut]

Maintainer Jiawei Zhang <zhan4362@umn.edu>

Repository CRAN

Date/Publication 2021-09-14 20:40:03 UTC

R topics documented:

BAGofT *A Binary Regression Adaptive Goodness-of-fit Test (BAGofT)*

Description

BAGofT is used to test the goodness-of-fit of binary classifiers. The test statistic is constructed based on the results from multiple splittings. In each split, the test first splits the data into a training set and a validation set. Then, it adaptively obtains a partition based on the training set and performs a goodness-of-fit test on the validation set. Details can be found in Zhang, Ding and Yang (2021).

Usage

```
BAGofT(testModel, parFun = parRF(), data, nsplits = 100,
ne = floor(5*nrow(data)^(1/2)), nsim = 100)
```

Arguments

testModel	a function that generates predicted results from the classifier to test. Details can be found in "testGlmBi" for binomial regression, "testGlmnet" for penalized logistic regression, "testRF" for random forest, and "testXGboost" for XGboost.
parFun	a function that generates the adaptive partition. The default is 'parRF()' that generates a partition by random forest. More information can be found in "parRF".
data	a data frame containing the response and covariates used in the model together with the other covariates not in the model but considered used to generate the partition.
nsplits	number of splits. The default is 100.
ne	the size of the validation set. The default is floor(5*nrow(data)^(1/2)).
nsim	the number of simulated datasets to calculate the bootstrap p-value.

Value

p.value	the bootstrap p-value of the BAGofT test statistic (which combines the results from multiple splitting by taking the average).
p.value2	the bootstrap p-value from an alternative version of the BAGofT test statistic (which combines the results from multiple splitting by taking the sample median).
p.value3	the bootstrap p-value from an alternative version of the BAGofT test statistic (which combines the results from multiple splitting by taking the minimum).
pmean	the BAGofT test statistic (which combines the results from multiple splitting by taking the average).
pmedian	an alternative BAGofT test statistic (which combines the results from multiple splitting by taking the sample median).

`pmin` an alternative BAGofT test statistic (which combines the results from multiple splitting by taking the minimum).

`simRes` a list that contains the simulated test statitics used to generate the bootstrap p-values. 'simRes$pmeanSim', 'simRes$pmediansim', 'simRes$pmeanSim' corresepond to the three kinds of BAGofT statistics, respectively.

`singleSplit.results`
a list that contains the results from each splitting. Its elements are as follows.

'singleSplit.results[[k]]$chisq': The chi-squared statistic of the BAGofT test from the kth splitting.

'singleSplit.results[[k]]$p.value': The p-value calculated from the chi-squared statistic.

'singleSplit.results[[k]]$ngp': The number of groups chosen by the adaptive partition.

'singleSplit.results[[k]]$contri': The weighted sum of squares from each group.

'singleSplit.results[[k]]$parRes': Variable importance (or other results from custom partition functions) from the adaptive partition.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
#####################################################
# Generate a sample dataset.
#####################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200

# generate covariates data
x1dat <- runif(n, -3, 3)
x2dat <- rnorm(n, 0, 1)
x3dat <- rchisq(n, 4)

# set coefficients
beta1 <- 1
beta2 <- 1
beta3 <- 1

# calculate the linear predictor data
lindat <- x1dat * beta1 + x2dat * beta2 + x3dat * beta3
# calculate the probabilities by inverse logit link
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))
```

```
# generate the dataset
dat <- data.frame(y = ydat, x1 = x1dat, x2 = x2dat,
                  x3 = x3dat)

################################################
# Obtain the testing result
################################################
# Test a logistic regression that misses 'x3'. The partition
# variables are 'x1', 'x2', and 'x3'.
testRes <- BAGofT(testModel =testGlmBi(formula = y ~ x1 + x2 , link = "logit"),
       parFun = parRF(parVar = c("x1", "x2", "x3")),
       data = dat)

# the bootstrap p-value is 0. Therefore, the test is rejected
print(testRes$p.value)

# the variable importance from the adaptive partition shows that x3 is likely
# to be the reason for the overfitting (,which is correct since the formula
# fm misses the x3).
print(VarImp(testRes))

## End(Not run)
```

---

parRF *Adaptive partition based on random forests*

Description

parRF generates an adaptive partition based on the training set data and training set predictions. It controls the group sizes by the covariates data from the validation set.

Usage

```
parRF(parVar = ".", Kmax = NULL, nmin = NULL, ntree = 60, mtry = NULL, maxnodes = NULL)
```

Arguments

parVar	a character vector that contains the names of the covariates to generate the adaptive partition. The default is taking all the variables except the response from the 'data'.
Kmax	the maximum number of groups. The default is floor(nrow(Test.data)/nmin).
nmin	a numerical vector of the training set Pearson residuals from the classifier to test. The default is ceiling(sqrt(nrow(Validation.data))).
ntree	number of trees to grow. The default is 60.
mtry	number of variables randomly sampled as candidates at each split. The default value is floor(length(parVarNew)/3) where parVarNew is the number of covariates after the preselection.
maxnodes	maximum number of terminal nodes trees in the forest can have.

Value

gup	a factor that contains the grouping result of the validation set data.
parRes	a list that contains the variable importance from the random forest.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
##################################################
# Generate a sample dataset.
##################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200

# generate covariates data
x1dat <- runif(n, -3, 3)
x2dat <- rnorm(n, 0, 1)
x3dat <- rchisq(n, 4)

# set coefficients
beta1 <- 1
beta2 <- 1
beta3 <- 1

# calculate the linear predictor data
lindat <- x1dat * beta1 + x2dat * beta2 + x3dat * beta3
# calculate the probabilities by inverse logit link
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, x1 = x1dat, x2 = x2dat,
                  x3 = x3dat)

##################################################
# Apply parRF to generate an adaptive partition
##################################################
# number of rows in the dataset
nr <- nrow(dat)
# size of the validation set
ne <- floor(5*nrow(dat)^(1/2))
# obtain the training set size
nt <- nr - ne
# the indices for training set observations
```

```
trainIn <- sample(c(1 : nr), nt)

#split the data
datT <- dat[trainIn, ]
datE <- dat[-trainIn, ]
# fit a logistic regression model to test by training data
testModel <- testGlmBi(formula = y ~ x1 + x2 , link = "logit")
# output training set predictions and pearson residuals
testMod <- testModel(Train.data = datT, Validation.data = datE)

# obtain adaptive partition result from parFun
parFun <- parRF(parVar = c("x1", "x2", "x3"))
par <- parFun(Rsp = testMod$Rsp, predT = testMod$predT, res = testMod$res,
              Train.data = datT, Validation.data = datE)

# print the grouping result of the validataion set data
print(par$gup)

# print variable importance from the random forest
print(par$parRes)
```

`testGlmBi` *Testing binomial regressions*

Description

`testGlmBi` specifies a binomial regression as the classifier to test. It returns a function that can be taken as the input of 'testModel'.

Usage

```
testGlmBi(formula, link)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to test.
`link`	a specification for the model link function. Can be one of "logit", "probit", "cloglog".

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
##################################################
# Generate a sample dataset.
##################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200

# generate covariates data
x1dat <- runif(n, -3, 3)
x2dat <- rnorm(n, 0, 1)
x3dat <- rchisq(n, 4)

# set coefficients
beta1 <- 1
beta2 <- 1
beta3 <- 1

# calculate the linear predictor data
lindat <- x1dat * beta1 + x2dat * beta2 + x3dat * beta3
# calculate the probabilities by inverse logit link
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, x1 = x1dat, x2 = x2dat,
                    x3 = x3dat)

##################################################
# Obtain the testing result
##################################################
# Test a logistic regression that misses 'x3'. The partition
# variables are 'x1', 'x2', and 'x3'.
testRes <- BAGofT(testModel =testGlmBi(formula = y ~ x1 + x2 , link = "logit"),
       parFun = parRF(parVar = c("x1", "x2", "x3")),
       data = dat)

# the bootstrap p-value is 0. Therefore, the test is rejected
print(testRes$p.value)

# the variable importance from the adaptive partion shows that x3 is likely
# to be the reason for the overfitting (,which is correct since the formula
# fm misses the x3).
print(VarImp(testRes))

## End(Not run)
```

`testGlmnet` *Testing penalized logistic regressions*

Description

`testGlmnet` specifies a penalized logistic regression as the classifier to test. It returns a function that can be taken as the input of 'testModel'. R package 'glmnet' is required.

Usage

```
testGlmnet(formula, alpha = 1)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to test.
`alpha`	the elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$ `alpha=1` is the lasso penalty, and `alpha=0` the ridge penalty.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
#####################################################
# Generate a sample dataset.
#####################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200
# set the number of covariates
p <- 20

# generate covariates data
Xdat <- matrix(runif((n*p), -5,5), nrow = n, ncol = p)
colnames(Xdat) <- paste("x", c(1:p), sep = "")

# generate random coefficients
betaVec <- rnorm(6)
# calculate the linear predictor data
lindat <-  3 * (Xdat[,1] < 2 & Xdat[,1] > -2) + -3 * (Xdat[,1] > 2 | Xdat[,1] < -2) +
  0.5 * (Xdat[,2] + Xdat[, 3] + Xdat[,4] + Xdat[, 5])
```

```
# calculate the probabilities
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, Xdat)

################################################
# Obtain the testing result
################################################

# 50 percent training set
testRes1 <- BAGofT(testModel = testGlmnet(formula = y~., alpha = 1),
                   data = dat,
                   ne = n*0.5,
                   nsplits = 20,
                   nsim = 40)
# 75 percent training set
testRes2 <- BAGofT(testModel = testGlmnet(formula = y~., alpha = 1),
                    data = dat,
                    ne = n*0.75,
                    nsplits = 20,
                    nsim = 40)
# 90 percent training set
testRes3 <- BAGofT(testModel = testGlmnet(formula = y~., alpha = 1),
                    data = dat,
                    ne = n*0.9,
                    nsplits = 20,
                    nsim = 40)

# print the testing result.
print(c(testRes1$p.value, testRes2$p.value, testRes3$p.value))

## End(Not run)
```

`testRF` *Testing random forests*

Description

`testRF` specifies a random forest as the classifier to test. It returns a function that can be taken as the input of 'testModel'.

Usage

```
testRF(formula, ntree = 500, mtry = NULL, maxnodes = NULL)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to test.
ntree	number of trees to grow. The default is 500.
mtry	number of variables randomly sampled as candidates at each split. The default value is sqrt(p) where p is the number of covariates.
maxnodes	maximum number of terminal nodes trees in the forest can have.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
##################################################
# Generate a sample dataset.
##################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200
# set the number of covariates
p <- 20

# generate covariates data
Xdat <- matrix(runif((n*p), -5,5), nrow = n, ncol = p)
colnames(Xdat) <- paste("x", c(1:p), sep = "")

# generate random coefficients
betaVec <- rnorm(6)
# calculate the linear predictor data
lindat <-  3 * (Xdat[,1] < 2 & Xdat[,1] > -2) + -3 * (Xdat[,1] > 2 | Xdat[,1] < -2) +
  0.5 * (Xdat[,2] + Xdat[, 3] + Xdat[,4] + Xdat[, 5])
# calculate the probabilities
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, Xdat)

##################################################
# Obtain the testing result
##################################################

# 50 percent training set
testRes1 <- BAGofT(testModel = testRF(formula = y ~.),
```

```
                  data = dat,
                  ne = n*0.5,
                  nsplits = 20,
                  nsim = 40)
# 75 percent training set
testRes2 <- BAGofT(testModel = testRF(formula = y ~.),
                   data = dat,
                   ne = n*0.75,
                   nsplits = 20,
                   nsim = 40)
# 90 percent training set
testRes3 <- BAGofT(testModel = testRF(formula = y ~.),
                   data = dat,
                   ne = n*0.9,
                   nsplits = 20,
                   nsim = 40)

# print the testing result.
print(c(testRes1$p.value, testRes2$p.value, testRes3$p.value))

## End(Not run)
```

`testXGboost` *Testing XGboosts*

Description

`testXGboost` specifies an XGboost as the classifier to test. It returns a function that can be taken as the input of 'testModel'. R package 'xgboost' is required.

Usage

```
testXGboost(formula, params = list(), nrounds = 25)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to test.
`params`	the list of parameters. The complete list of parameters is available in the online documentation.
`nrounds`	max number of boosting iterations.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
##################################################
# Generate a sample dataset.
##################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200
# set the number of covariates
p <- 20

# generate covariates data
Xdat <- matrix(runif((n*p), -5,5), nrow = n, ncol = p)
colnames(Xdat) <- paste("x", c(1:p), sep = "")

# generate random coefficients
betaVec <- rnorm(6)
# calculate the linear predictor data
lindat <-  3 * (Xdat[,1] < 2 & Xdat[,1] > -2) + -3 * (Xdat[,1] > 2 | Xdat[,1] < -2) +
  0.5 * (Xdat[,2] + Xdat[, 3] + Xdat[,4] + Xdat[, 5])
# calculate the probabilities
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, Xdat)

##################################################
# Obtain the testing result
##################################################

# 50 percent training set
testRes1 <- BAGofT(testModel = testXGboost(formula = y ~.),
                   data = dat,
                   ne = n*0.5,
                   nsplits = 20,
                   nsim = 40)
# 75 percent training set
testRes2 <- BAGofT(testModel = testXGboost(formula = y ~.),
                    data = dat,
                    ne = n*0.75,
                    nsplits = 20,
                    nsim = 40)
# 90 percent training set
testRes3 <- BAGofT(testModel = testXGboost(formula = y ~.),
                    data = dat,
                    ne = n*0.9,
                    nsplits = 20,
                    nsim = 40)
```

```
# print the testing result.
print(c(testRes1$p.value, testRes2$p.value, testRes3$p.value))

## End(Not run)
```

VarImp *Variable Importance*

Description

VarImp averages the variable importance generated by "parRF" from different splittings.

Usage

```
VarImp(TestRes)
```

Arguments

TestRes	an output from "BAGofT".

Value

Var.imp	the averaged variable importance from multiple splittings. A high variable importance indicates that the corresponding covariate is likely to be related to the possible underfitting. When the number of partition covariates is larger than 5, output the result of 5 covariates with the largest averaged variable importance.
preVar.imp	the averaged variable importance for all of the variables. Output only when the number of partition covariates is larger than 5.

References

Zhang, Ding and Yang (2021) "Is a Classification Procedure Good Enough?-A Goodness-of-Fit Assessment Tool for Classification Learning" arXiv preprint arXiv:1911.03063v2 (2021).

Examples

```
## Not run:
##################################################
# Generate a sample dataset.
##################################################
# set the random seed
set.seed(20)
# set the number of observations
n <- 200

# generate covariates data
x1dat <- runif(n, -3, 3)
x2dat <- rnorm(n, 0, 1)
```

```
x3dat <- rchisq(n, 4)

# set coefficients
beta1 <- 1
beta2 <- 1
beta3 <- 1

# calculate the linear predictor data
lindat <- x1dat * beta1 + x2dat * beta2 + x3dat * beta3
# calculate the probabilities by inverse logit link
pdat <- 1/(1 + exp(-lindat))

# generate the response data
ydat <- sapply(pdat, function(x) stats :: rbinom(1, 1, x))

# generate the dataset
dat <- data.frame(y = ydat, x1 = x1dat, x2 = x2dat,
                    x3 = x3dat)

##################################################
# Obtain the testing result
##################################################
# Test a logistic regression that misses 'x3'. The partition
# variables are 'x1', 'x2', and 'x3'.
testRes <- BAGofT(testModel =testGlmBi(formula = y ~ x1 + x2 , link = "logit"),
       parFun = parRF(parVar = c("x1", "x2", "x3")),
       data = dat)

# the bootstrap p-value is 0. Therefore, the test is rejected
print(testRes$p.value)

# the variable importance from the adaptive partion shows that x3 is likely
# to be the reason for the overfitting (,which is correct since the formula
# fm misses the x3).
print(VarImp(testRes))

## End(Not run)
```

Index