Package 'pbcm'

January 30, 2020

Title Parametric Bootstrap Cross-Fitting Method

Version 0.1.0

Date 2020-01-29

Author Henri Kauhanen <henri@henr.in>

Maintainer Henri Kauhanen <henri@henr.in>

Description

Implements both data-informed and data-uninformed versions of the Parametric Bootstrap Cross-fitting Method (PBCM) for binary model selection, as well as some utility functions.

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

URL https://github.com/hkauhanen/pbcm

BugReports https://github.com/hkauhanen/pbcm/issues

R topics documented:

---

empirical.GoF   *Empirical Goodnesses of Fit*

Description

Fit models 1 and 2 to data and return the empirical goodnesses of fit as well as the difference in goodness of fit.

Usage

```
empirical.GoF(data, fun1, fun2, args1 = NULL, args2 = NULL,
  verbose = TRUE, GoFname = "GoF")
```

Arguments

data	Data frame
fun1	Modelling function 1
fun2	Modelling function 2
args1	List of arguments passed to fun1
args2	List of arguments passed to fun2
verbose	If TRUE, warnings are printed to the console
GoFname	Name of the element returned by fun1 and fun2 holding the goodness of fit

Details

Functions fun1 and fun2 must accept data as an argument in addition to any arguments specified in args1 and args2. They must return a list with an element carrying the calculated goodness of fit; by default the name of this element is taken to be the string "GoF" but this behaviour can be changed through the GoFname argument.

Value

A 1-row data frame of three columns:

GoF1 Goodness of fit for model 1

GoF2 Goodness of fit for model 2

DeltaGoF Equal to GoF1 - GoF2

Author(s)

Henri Kauhanen

Examples

```
x <- seq(from=0, to=1, length.out=100)
mockdata <- data.frame(x=x, y=x + rnorm(100, 0, 0.5))

myfitfun <- function(data, p) {
  res <- nls(y~a*x^p, data, start=list(a=1.1))
  list(a=coef(res), GoF=deviance(res))
}

empirical.GoF(mockdata, fun1=myfitfun, fun2=myfitfun,
              args1=list(p=1), args2=list(p=2))
```

---

kNN.classification   *k Nearest Neighbours Classification*

Description

Carry out k Nearest Neighbours (k-NN) classification on the results of a parametric boostrap.

Usage

```
kNN.classification(df, DeltaGoF.emp, k, ties = "model2",
  verbose = TRUE)
```

Arguments

df	Results of bootstrap; the output of pbcm.di or pbcm.du
DeltaGoF.emp	Empirical value of goodness of fit (e.g. from empirical.GoF)
k	Number of neighbours to employ in classification; may be a vector of integers
ties	Which way should ties (when distance to the two distributions is equal) be broken? By default, we break in favour of model 2, taking this to be the null model in the comparison.
verbose	If TRUE, warnings are issued to the console

Details

Calculates the cumulative distance (sum of squared differences) of DeltaGoF.emp to both DeltaGoF distributions found in df (i.e. one with model 1 as generator and one with model 2 as generator), taking into account the k nearest neighbours only. Decides in favour of model 1 if this cumulative distance to the model 1 distribution is smaller than than the distance to model 2, and vice versa. If distances are equal, decision is made according to the ties argument.

Value

A data frame containing the computed distances and decisions, one row per each value of k

Author(s)

Henri Kauhanen

References

Schultheis, H. & Singhaniya, A. (2015) Decision criteria for model comparison using the parametric bootstrap cross-fitting method. *Cognitive Systems Research*, 33, 100–121.

See Also

empirical.GoF, pbcm.di, pbcm.du

Examples

```
x <- seq(from=0, to=1, length.out=100)
mockdata <- data.frame(x=x, y=x + rnorm(100, 0, 0.5))

myfitfun <- function(data, p) {
  res <- nls(y~a*x^p, data, start=list(a=1.1))
  list(a=coef(res), GoF=deviance(res))
}

mygenfun <- function(model, p) {
  x <- seq(from=0, to=1, length.out=100)
  y <- model$a*x^p + rnorm(100, 0, 0.5)
  data.frame(x=x, y=y)
}

pb <- pbcm.di(data=mockdata, fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun,
        genfun2=mygenfun, reps=20, args1=list(p=1), args2=list(p=2),
        genargs1=list(p=1), genargs2=list(p=2))

emp <- empirical.GoF(mockdata, fun1=myfitfun, fun2=myfitfun,
                     args1=list(p=1), args2=list(p=2))

kNN.classification(df=pb, DeltaGoF.emp=emp$DeltaGoF, k=c(10, 20))
```

---

kNN.confusionmatrix   *Confusion Matrices through k Nearest Neighbours Classification*

Description

Computes confusion matrices (one for each value of k) using k-NN classification from the results of two parametric bootstraps, one of these being labelled a holdout set and tested against the other one.

Usage

```
kNN.confusionmatrix(df, df.holdout, k, ties = "model2", verbose = TRUE)
```

Arguments

df	Data frame output by [pbcm.di](#) or [pbcm.du](#)
df.holdout	Data frame output by [pbcm.di](#) or [pbcm.du](#)
k	Number of neighbours to consider in k-NN classification; may be a vector of integers
ties	Which way to break ties in k-NN classification (see [kNN.classification](#))
verbose	If TRUE, prints a progress bar and issues warnings

Details

The function takes each DeltaGoF value from df.holdout, compares it against the DeltaGoF distributions in df, and decides based on k-NN classification. By convention, we take model 2 as the null hypothesis and model 1 as the alternative. Hence a false positive, for instance, means the situation where model 2 generated the data but the decision was in favour of model 1.

Value

A data frame with the following columns:

k Number of nearest neighbours

P Number of positives

N Number of negatives

TP Number of true positives

FP Number of false positives

TN Number of true negatives

FN Number of false negatives

alpha Type I error (false positive) rate; equal to FP divided by N

beta Type II error (false negative) rate; equal to FN divided by P

In addition to these columns, each argument that was passed via genargs1 and genargs2 to [pbcm.di](#) or [pbcm.du](#) to generate df.holdout is included as a column of its own.

Author(s)

Henri Kauhanen

See Also

[kNN.classification](#), [pbcm.di](#), [pbcm.du](#)

Examples

```
x <- seq(from=0, to=1, length.out=100)
mockdata <- data.frame(x=x, y=x + rnorm(100, 0, 0.5))

myfitfun <- function(data, p) {
  res <- nls(y~a*x^p, data, start=list(a=1.1))
  list(a=coef(res), GoF=deviance(res))
}

mygenfun <- function(model, p) {
  x <- seq(from=0, to=1, length.out=100)
  y <- model$a*x^p + rnorm(100, 0, 0.5)
  data.frame(x=x, y=y)
}

pb1 <- pbcm.di(data=mockdata, fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun,
        genfun2=mygenfun, reps=20, args1=list(p=1), args2=list(p=2),
        genargs1=list(p=1), genargs2=list(p=2))

pb2 <- pbcm.di(data=mockdata, fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun,
        genfun2=mygenfun, reps=20, args1=list(p=1), args2=list(p=2),
        genargs1=list(p=1), genargs2=list(p=2))

kNN.confusionmatrix(df=pb1, df.holdout=pb2, k=1:10)
```

---

pbcm.di   *Data-informed Parametric Bootstrap Cross-fitting*

Description

The data-informed Parametric Bootstrap Cross-fitting Method (PBCM) generates synthetic data from two models of a phenomenon parameterized by fits to an empirical dataset, and then cross-fits the models to these data. The result is two distributions of the goodness of fit metric $\Delta GoF = GoF_1 - GoF_2$, where GoF_1 is the fit of model 1 and GoF_2 the fit of model 2.

Usage

```
pbcm.di(data, fun1, fun2, genfun1, genfun2, reps, args1 = NULL,
  args2 = NULL, genargs1 = NULL, genargs2 = NULL,
  nonparametric_bootstrap = TRUE, verbose = TRUE, GoFname = "GoF")
```

Arguments

data	Data frame
fun1	First modelling function
fun2	Second modelling function
genfun1	Generator function for first model

genfun2	Generator function for second model
reps	Number of Monte Carlo repetitions
args1	List of arguments passed to `fun1`
args2	List of arguments passed to `fun2`
genargs1	List of arguments passed to `genfun1`
genargs2	List of arguments passed to `genfun2`
nonparametric_bootstrap	
	Whether `data` should be nonparametrically bootstrapped before the parametric bootstrap
verbose	If `TRUE`, a progress bar is printed to the console and warnings are issued
GoFname	Name of the element returned by `fun1` and `fun2` holding the goodness of fit; see Details

Details

Functions `fun1` and `fun2` must take `data` as an argument in addition to any arguments specified in `args1` and `args2`. `fun1` and `fun2` must return a list with at least one element carrying the goodness of fit; the name of this element may be specified through the `GoFname` argument, by default the string `"GoF"` is assumed. Functions `genfun1` and `genfun2` must take an argument named `model` (the output of `fun1` and `fun2`).

Value

A data frame in long format with the following columns:

`rep` Monte Carlo repetition number

`generator` Generating model

`GoF1` Goodness of fit of model 1

`GoF2` Goodness of fit of model 2

`DeltaGoF` Equals `GoF1 - GoF2`

In addition to these columns, each argument in the lists `genargs1` and `genargs2` is included as a column of its own, with the argument's name prefixed by `"model1_"` or `"model2_"`.

Author(s)

Henri Kauhanen

References

Wagenmakers, E.-J., Ratcliff, R., Gomez, P. & Iverson, G. J. (2004) Assessing model mimicry using the parametric bootstrap. *Journal of Mathematical Psychology*, 48(1), 28–50.

Examples

```
x <- seq(from=0, to=1, length.out=100)
mockdata <- data.frame(x=x, y=x + rnorm(100, 0, 0.5))

myfitfun <- function(data, p) {
  res <- nls(y~a*x^p, data, start=list(a=1.1))
  list(a=coef(res), GoF=deviance(res))
}

mygenfun <- function(model, p) {
  x <- seq(from=0, to=1, length.out=100)
  y <- model$a*x^p + rnorm(100, 0, 0.5)
  data.frame(x=x, y=y)
}

pbcm.di(data=mockdata, fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun,
        genfun2=mygenfun, reps=20, args1=list(p=1), args2=list(p=2),
        genargs1=list(p=1), genargs2=list(p=2))
```

---

pbcm.du   *Data-uninformed Parametric Bootstrap Cross-fitting*

Description

The data-uninformed Parametric Bootstrap Cross-fitting Method (PBCM) generates synthetic data from two models of a phenomenon with given model parameter values, and then cross-fits the models to these data. The result is two distributions of the goodness of fit metric $\Delta GoF = GoF_1 - GoF_2$, where GoF_1 is the fit of model 1 and GoF_2 the fit of model 2.

Usage

```
pbcm.du(fun1, fun2, genfun1, genfun2, reps, args1 = NULL, args2 = NULL,
  genargs1 = NULL, genargs2 = NULL, verbose = TRUE,
  GoFname = "GoF")
```

Arguments

fun1	First modelling function
fun2	Second modelling function
genfun1	Generator function for first model
genfun2	Generator function for second model
reps	Number of Monte Carlo repetitions
args1	List of arguments passed to fun1
args2	List of arguments passed to fun2
genargs1	List of arguments passed to genfun1

genargs2	List of arguments passed to genfun2
verbose	If TRUE, a progress bar is printed to the console and warnings are issued
GoFname	Name of the element returned by fun1 and fun2 holding the goodness of fit; see Details

Details

Functions fun1 and fun2 must take an argument named data in addition to any arguments specified in args1 and args2; this is used to pass the synthetic data generated by genfun1 and genfun2. fun1 and fun2 must return a list with at least one element carrying the goodness of fit; the name of this element may be specified through the GoFname argument, by default the string "GoF" is assumed.

Value

A data frame in long format with the following columns:

rep Monte Carlo repetition number

generator Generating model

GoF1 Goodness of fit of model 1

GoF2 Goodness of fit of model 2

DeltaGoF Equals GoF1 - GoF2

In addition to these columns, each argument in the lists genargs1 and genargs2 is included as a column of its own, with the argument's name prefixed by "model1_" or "model2_".

Author(s)

Henri Kauhanen

References

Wagenmakers, E.-J., Ratcliff, R., Gomez, P. & Iverson, G. J. (2004) Assessing model mimicry using the parametric bootstrap. *Journal of Mathematical Psychology*, 48(1), 28–50.

Examples

```
x <- seq(from=0, to=1, length.out=100)
mockdata <- data.frame(x=x, y=x + rnorm(100, 0, 0.5))

myfitfun <- function(data, p) {
  res <- nls(y~a*x^p, data, start=list(a=1.1))
  list(a=coef(res), GoF=deviance(res))
}

mygenfun <- function(a, p) {
  x <- seq(from=0, to=1, length.out=100)
  y <- a*x^p + rnorm(100, 0, 0.5)
  data.frame(x=x, y=y)
}
```

```
pbcm.du(fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun, genfun2=mygenfun,
        reps=20, args1=list(p=1), args2=list(p=2),
        genargs1=list(a=1.1, p=1), genargs2=list(a=1.1, p=2))

sweep <- lapply(X=seq(from=0.5, to=1.5, by=0.1),
                FUN=function(X) {
                  pbcm.du(fun1=myfitfun, fun2=myfitfun, genfun1=mygenfun,
                          genfun2=mygenfun, reps=20,
                          args1=list(p=1), args2=list(p=2),
                          genargs1=list(a=X, p=1), genargs2=list(a=X, p=2))
                })

sweep <- do.call(rbind, sweep)

sweep$parameter <- ifelse(is.na(sweep$model1_a), sweep$model2_a, sweep$model1_a)

## Not run:
  library(ggplot2)
  g <- ggplot(sweep, aes(x=DeltaGoF, fill=generator)) + geom_density(alpha=0.5)
  g <- g + facet_wrap(.~parameter)
  print(g)

## End(Not run)
```

Index