Package 'agghoo'

May 7, 2026

Encoding UTF-8

Title Aggregated Hold-Out Cross Validation

Date 2023-05-23

Version 0.1-0

Description The 'agghoo' procedure is an alternative to usual cross-validation. Instead of choosing the best model trained on V subsamples, it determines a winner model for each subsample, and then aggregates the V outputs. For the details, see ``Aggregated hold-out'' by Guillaume Maillard, Sylvain Arlot, Matthieu Lerasle (2021) <doi:10.48550/arXiv.1909.04890> published in Journal of Machine Learning Research 22(20):1--55.

Depends R (>= 3.5.0)

Imports class, parallel, R6, rpart, FNN

Suggests roxygen2, mlbench

URL https://git.auder.net/?p=agghoo.git

License MIT + file LICENSE

RoxygenNote 7.2.3

Collate 'utils.R' 'checks.R' 'R6_Model.R' 'R6_AgghooCV.R' 'agghoo.R' 'compareTo.R'

NeedsCompilation no

Author Sylvain Arlot [ctb],
Benjamin Auder [aut, cre, cph],
Melina Gallopin [ctb],
Matthieu Lerasle [ctb],
Guillaume Maillard [ctb]

Maintainer Benjamin Auder <benjamin.auder@universite-paris-saclay.fr>

Repository CRAN

Date/Publication 2023-05-25 19:50:02 UTC

Contents

---

agghoo *agghoo*

Description

Run the (core) agghoo procedure. Arguments specify the list of models, their parameters and the cross-validation settings, among others.

Usage

```
agghoo(data, target, task = NULL, gmodel = NULL, params = NULL, loss = NULL)
```

Arguments

`data`	Data frame or matrix containing the data in lines.
`target`	The target values to predict. Generally a vector, but possibly a matrix in the case of "soft classification".
`task`	"classification" or "regression". Default: regression if target is numerical, classification otherwise.
`gmodel`	A "generic model", which is a function returning a predict function (taking X as only argument) from the tuple (dataHO, targetHO, param), where 'HO' stands for 'Hold-Out', referring to cross-validation. Cross-validation is run on an array of 'param's. See params argument. Default: see R6::Model.
`params`	A list of parameters. Often, one list cell is just a numerical value, but in general it could be of any type. Default: see R6::Model.
`loss`	A function assessing the error of a prediction. Arguments are y1 and y2 (comparing a prediction to known values). loss(y1, y2) –> real number (error). Default: see R6::AgghooCV.

Value

An R6::AgghooCV object o. Then, call o$fit() and finally o$predict(newData)

References

Guillaume Maillard, Sylvain Arlot, Matthieu Lerasle. "Aggregated hold-out". Journal of Machine Learning Research 22(20):1–55, 2021.

See Also

Function compareTo

Examples

```
# Basic usage:

# Regression:
a_reg <- agghoo(iris[,-c(2,5)], iris[,2])
a_reg$fit()
pr <- a_reg$predict(iris[,-c(2,5)] + rnorm(450, sd=0.1))
# Classification
a_cla <- agghoo(iris[,-5], iris[,5])
a_cla$fit()
pc <- a_cla$predict(iris[,-5] + rnorm(600, sd=0.1))


# Advanced usage:
data(iris)
library(mlbench)
data(PimaIndiansDiabetes)

# Run only agghoo on iris dataset (split into train/test, etc).
# Default parameters: see ?agghoo and ?AgghooCV
compareTo(iris[,-5], iris[,5], agghoo_run)

# Run both agghoo and standard CV, specifiying some parameters.
compareTo(iris[,-5], iris[,5], list(agghoo_run, standardCV_run), gmodel="tree")
compareTo(iris[,-5], iris[,5], list(agghoo_run, standardCV_run),
          gmodel="knn", params=c(3, 7, 13, 17, 23, 31),
          CV = list(type="vfold", V=5, shuffle=TRUE))

# Run both agghoo and standard CV, averaging errors over N=10 runs
# (possible for a single method but wouldn't make much sense...).
nc <- 1 #for CRAN
compareMulti(PimaIndiansDiabetes[,-9], PimaIndiansDiabetes[,9],
             list(agghoo_run, standardCV_run), N=10, gmodel="tree", nc=nc)

# Compare several values of V
compareRange(PimaIndiansDiabetes[,-9], PimaIndiansDiabetes[,9],
             list(agghoo_run, standardCV_run), N=10, V_range=c(10, 20, 30), nc=nc)

# For example to use average of squared differences.
# Default is "mean(abs(y1 - y2))".
loss2 <- function(y1, y2) mean((y1 - y2)^2)

# In regression on artificial datasets (TODO: real data?)
```

```
data <- mlbench.twonorm(300, 3)$x
target <- rowSums(data)
compareMulti(data, target, list(agghoo_run, standardCV_run), nc=nc,
             N=10, gmodel="ppr", params=c(1, 3, 5, 7, 9), loss=loss2,
             CV = list(type="MC", V=12, test_size=0.3))

compareMulti(data, target, list(agghoo_run, standardCV_run), nc=nc,
             N=10, floss=loss2, CV = list(type="vfold", V=10, shuffle=FALSE))

# Random tests to check that method doesn't fail in 1D case
M <- matrix(rnorm(200), ncol=2)
compareTo(as.matrix(M[,-2]), M[,2], list(agghoo_run, standardCV_run), gmodel="knn")
compareTo(as.matrix(M[,-2]), M[,2], list(agghoo_run, standardCV_run), gmodel="ppr")
```

AgghooCV *R6 class with agghoo functions fit() and predict().*

Description

Class encapsulating the methods to run to obtain the best predictor from the list of models (see 'Model' class).

Methods

Public methods:

- AgghooCV$new()
- AgghooCV$fit()
- AgghooCV$predict()
- AgghooCV$getParams()
- AgghooCV$clone()

Method `new()`**:** Create a new AgghooCV object.

Usage:

```
AgghooCV$new(data, target, task, gmodel, loss)
```

Arguments:

`data` Matrix or data.frame

`target` Vector of targets (generally numeric or factor)

`task` "regression" or "classification". Default: classification if target not numeric.

`gmodel` Generic model returning a predictive function Default: tree if mixed data, knn/ppr otherwise.

`loss` Function assessing the error of a prediction Default: error rate or mean(abs(error)).

Method `fit()`**:** Fit an agghoo model.

Usage:

```
AgghooCV$fit(CV = NULL)
```

Arguments:

`CV` List describing cross-validation to run. Slots:
- type: 'vfold' or 'MC' for Monte-Carlo (default: MC)
- V: number of runs (default: 10)
- test_size: percentage of data in the test dataset, for MC (irrelevant for V-fold). Default: 0.2.
- shuffle: wether or not to shuffle data before V-fold. Irrelevant for Monte-Carlo; default: TRUE

Default (if NULL): type="MC", V=10, test_size=0.2

Method `predict()`**:** Predict an agghoo model (after calling fit())

Usage:

```
AgghooCV$predict(X)
```

Arguments:

`X` Matrix or data.frame to predict

Method `getParams()`**:** Return the list of V best parameters (after calling fit())

Usage:

```
AgghooCV$getParams()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AgghooCV$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`agghoo_run` *agghoo_run*

Description

Run and eval the agghoo procedure.

Usage

```
agghoo_run(dataTrain, dataTest, targetTrain, targetTest, floss, verbose, ...)
```

Arguments

`dataTrain`	Train dataset
`dataTest`	Test dataset
`targetTrain`	Train targets
`targetTest`	Test targets
`floss`	Loss function to compute error on test dataset
`verbose`	Show some execution trace
`...`	List defining the model (gmodel) and its parameters (params)

compareMulti *compareMulti*

Description

Run compareTo N times in parallel.

Usage

```
compareMulti(
  data,
  target,
  method_s,
  N = 100,
  nc = NA,
  floss = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

data	Data matrix or data.frame
target	Target vector (generally)
method_s	Either a single function, or a list (examples: agghoo_run, standardCV_run)
N	Number of calls to method(s)
nc	Number of cores. Set to parallel::detectCores() if undefined. Set it to any value <=1 to say "no parallelism".
floss	Loss function to compute the error on testing dataset.
verbose	TRUE to print task numbers and "Errors:" in the end.
...	arguments passed to method_s function(s)

compareRange *compareRange*

Description

Run compareMulti on several values of the parameter V.

Usage

```
compareRange(
  data,
  target,
  method_s,
  N = 100,
  nc = NA,
  floss = NULL,
  V_range = c(10, 15, 20),
  ...
)
```

Arguments

data	Data matrix or data.frame
target	Target vector (generally)
method_s	Either a single function, or a list (examples: agghoo_run, standardCV_run)
N	Number of calls to method(s)
nc	Number of cores. Set to parallel::detectCores() if undefined. Set it to any value <=1 to say "no parallelism".
floss	Loss function to compute the error on testing dataset.
V_range	Values of V to be tested.
...	arguments passed to method_s function(s)

---

compareTo *compareTo*

Description

Compare a list of learning methods (or run only one), on data/target.

Usage

```
compareTo(
  data,
  target,
  method_s,
  rseed = -1,
  floss = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

data	Data matrix or data.frame
target	Target vector (generally)
method_s	Either a single function, or a list (examples: agghoo_run, standardCV_run)
rseed	Seed of the random generator (-1 means "random seed")
floss	Loss function to compute the error on testing dataset.
verbose	TRUE to request methods to be verbose.
...	arguments passed to method_s function(s)

CVvoting_run *CVvoting_run*

Description

Run and eval the voting cross-validation procedure.

Usage

```
CVvoting_run(dataTrain, dataTest, targetTrain, targetTest, floss, verbose, ...)
```

Arguments

dataTrain	Train dataset
dataTest	Test dataset
targetTrain	Train targets
targetTest	Test targets
floss	Loss function to compute error on test dataset
verbose	Show some execution trace
...	List defining the model (gmodel) and its parameters (params)

Model *R6 class representing a (generic) model.*

Description

"Model" class, containing a (generic) learning function, which from data + target [+ params] returns a prediction function X –> y. Parameters for cross-validation are either provided or estimated. Model family can be chosen among "tree", "ppr" and "knn" for now.

Public fields

`nmodels` Number of parameters (= number of [predictive] models)

Methods

Public methods:

- `Model$new()`
- `Model$get()`
- `Model$getParam()`
- `Model$clone()`

Method `new()`**:** Create a new generic model.

Usage:

```
Model$new(data, target, task, gmodel = NULL, params = NULL)
```

Arguments:

`data` Matrix or data.frame

`target` Vector of targets (generally numeric or factor)

`task` "regression" or "classification"

`gmodel` Generic model returning a predictive function; chosen automatically given data and target nature if not provided.

`params` List of parameters for cross-validation (each defining a model)

Method `get()`**:** Returns the model at index "index", trained on dataHO/targetHO.

Usage:

```
Model$get(dataHO, targetHO, index)
```

Arguments:

`dataHO` Matrix or data.frame

`targetHO` Vector of targets (generally numeric or factor)

`index` Index of the model in 1...nmodels

Method `getParam()`**:** Returns the parameter at index "index".

Usage:

```
Model$getParam(index)
```

Arguments:

`index` Index of the model in 1...nmodels

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`standardCV_run` *standardCV_run*

Description

Run and eval the standard cross-validation procedure.

Usage

```
standardCV_run(
  dataTrain,
  dataTest,
  targetTrain,
  targetTest,
  floss,
  verbose,
  ...
)
```

Arguments

`dataTrain`	Train dataset
`dataTest`	Test dataset
`targetTrain`	Train targets
`targetTest`	Test targets
`floss`	Loss function to compute error on test dataset
`verbose`	Show some execution trace
`...`	List defining the model (gmodel) and its parameters (params)

Index