Package 'GuessCompx'

June 3, 2019

Type Package

Title Empirically Estimates Algorithm Complexity

Version 1.0.3

Author Marc Agenis <marc.agenis@gmail.com> and Neeraj Bokde <neerajdhanraj@gmail.com>

Maintainer Marc Agenis <marc.agenis@gmail.com>

Description

Make an empirical guess on the time and memory complexities of an algorithm or a function. Tests multiple, increasing size random samples of your data and tries to fit various complexity functions o(n), o(n2), o(log(n)), etc.
Based on best fit, it predicts the full computation time on your whole dataset. Results are plotted with 'ggplot2'.

BugReports https://github.com/agenis/GuessCompx/issues

URL https://github.com/agenis/GuessCompx

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports dplyr, reshape2, ggplot2, lubridate, boot

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-06-03 12:50:34 UTC

R topics documented:

`CompEst` *Complexity Estimation and Prediction*

Description

_Main function for the complexity estimation of an algorithm

Usage

```
CompEst(d, f, random.sampling = FALSE, max.time = 30,
  start.size = NULL, replicates = 4, strata = NULL,
  power.factor = 2, alpha.value = 0.005, plot.result = TRUE)
```

Arguments

`d`	the data.frame, vector or matrix on which the algorithm is to be tested
`f`	a user-defined function that runs the algorithm, taking d as first argument. No return value is needed.
`random.sampling`	boolean; if TRUE a random sample is taken at each step, if FALSE the first N observations are taken at each step. Choosing a random sampling is relevant whith the use of replicates to help the discrimination power for complexity functions.
`max.time`	maximum time in seconds allowed for each step of the analysis. The function will stop once this time limit has been reached. Default is 30 seconds. There is no such limitation regarding memory.
`start.size`	the size in rows of the first sample to run the algorithm. Default is 'floor(log2(nrow(d)))'. If strata is not NULL, we recommend to enter a multiple of the number of categories.
`replicates`	the number of replicated runs of the algorithm for a specific sample size. Allows to better discriminate the complexity functions. Default is 2.
`strata`	a string, name of the categorical column of d that must be used for stratified sampling. A fixed proportion of the categories will be sampled, always keeping at least one observation per category.
`power.factor`	the common ratio of the geometric progression of the sample sizes. Default is 2, and will make sample sizes double every step. Decimal numbers are allowed.
`alpha.value`	the alpha risk of the test whether the model is significantly different from a constant relation. Default is 0.005.
`plot.result`	boolean indicatif if a summary plot of all the complexity functions is to be displayed

Details

The fit of a complexity function is one among: constant, linear, quadratic, cubic, logarithmic, square.root, n.log(n). Model comparison is achieved using Leave-One-Out error minimisation of the MSE (see 'boot::cv.glm' doc). Note that when a CONSTANT relationship is predicted, it might simply mean that the max.time value is too low to show any tendency. For time series, the sampling removes the ts attribute to the input vector, so the user's function shall include again this ts() if a frequency is needed; also remind to avoid random sampling for it will break the series.

Value

A list with the best complexity function and the computation time on the whole dataset, for both time and memory complexity (Windows) and time complexity only (all other OS).

Examples

```
# Dummy function that mimics a constant time complexity and
# N.log(N) memory complexity:
f1 = function(df){
  Sys.sleep(rnorm(1, 0.1, 0.02))
  v = rnorm(n = nrow(df)*log(nrow(df))*(runif(1, 1e3, 1.1e3)))
}
out = CompEst(d = mtcars, f = f1, replicates=2, start.size=2, max.time = 1)
# Raises an alert for TIME complexity.
# Sometimes confuses MEMORY complexity with linear:
print(out)

# Real dist function analysis (everything is quadratic here):
f2 = dist
d  = ggplot2::diamonds[, 5:8]
CompEst(d = d, f = f2, replicates = 1, max.time = 1)

# For time series functions, your `f` argument may include ts()
# to avoid loosing this ts attribute at sampling
# It is also recommended to set `start.size` argument to 3 periods at least.
f = function(d) arima(ts(d, freq = 12), order=c(1,0,1), seasonal = c(0,1,1))
d = ggplot2::txhousing$sales
# Should return a linear trend for TIME:
CompEst(d, f, start.size = 4*12, random.sampling = FALSE)
```

`CompEstBenchmark` *Complexity Estimation and Prediction*

Description

_Benchmark procedure to fit complexity functions to a data.frame of time or memory values

Usage

```
CompEstBenchmark(to.model, use = "time")
```

Arguments

to.model	A data.frame produced by the CompEst() function, comprised of the following columns: size, time, memory, NlogN_X
use	a string indicatif if the function deals "time" or "memory" data

Value

a list with all the fitted complexity model.

CompEstPlot *Complexity Estimation and Prediction*

Description

_Plot function for the results of algorithms complexity

Usage

```
CompEstPlot(to.plot, element_title = list("", ""), use = "time")
```

Arguments

to.plot	a dataset produced by CompEst() function
element_title	a string that will be added to the subtitle of the plot
use	a string indicatif if the function deals "time" or "memory" data

Value

a ggplot object

CompEstPred *Complexity Estimation and Prediction*

Description

_Prediction function for the computation time of a whole dataset

Usage

```
CompEstPred(model.list, benchmark, N, use = "time")
```

Arguments

model.list	A list containing the fitted complexity functions, produced by CompEst()
benchmark	A vector of LOO errors of complexity functions, produced by CompEst()
N	number of rows of the whole dataset, produced by CompEst()
use	a string indicatif if the function deals "time" or "memory" data

Value

a string of the predicted time for the whole dataset

GroupedSampleFracAtLeastOneSample

Fraction Sampling without empty output

Description

Sample a random proportion of the data, keeping at least one observation

Usage

```
GroupedSampleFracAtLeastOneSample(d_subset, prop, is.random = TRUE)
```

Arguments

d_subset	A dtta.frame from which a small sample is to be returned
prop	A number between 0 and 1, being the desired sampling fraction.
is.random	a boolean. If TRUE, a random sample is drawn, else it takes the head() of the data

Details

This function is designed to allow its use with group splitting or do.by methods.

Value

A random sample from the data, of proportion prop, but always returning at least one observation even if prop is too low.

rhead *Random Head*

Description

Small Random Sample from a vector or data.frame

Usage

```
rhead(data, rows = 7, is.random = TRUE)
```

Arguments

data	A vector or data.frame from which a small sample is to be returned
rows	A positive integer, representing the number of lines or elements to print, default is 7.
is.random	a boolean. If TRUE, a random sample is drawn, else it takes the head() of the data

Details

If the rows parameter is greater than the actual number of rows/elements of the data, the returned value is the initial dataset after shuffling

Value

A random sample without replacement taken from the data, in the same format than the input.

Examples

```
set.seed(1234)
rhead(mtcars)
```

Index