Package 'race'

February 20, 2015

Version 0.1.59

Date 2012-04-05

Title Racing methods for the selection of the best

Author Mauro Birattari <mbiro@ulb.ac.be>

Maintainer Leslie Perez <lperez@iridia.ulb.ac.be>

Suggests rpvm, nnet

Description Implementation of some racing methods for the empirical selection of the best. If the R package `rpvm' is installed (and if PVM is available, properly configured, and initialized), the evaluation of the candidates are performed in parallel on different hosts.

License GPL (>= 2)

Repository CRAN

Date/Publication 2012-04-06 10:11:00

NeedsCompilation no

R topics documented:

race *Racing methods for the selection of the best*

Description

Implementation of some racing methods for the empirical selection of the best. If the R package rpvm is installed (and PVM is available, properly configured, and initialized), the evaluation of the candidates are performed in parallel on different hosts.

Usage

```
race(wrapper.file, maxExp=0,
        stat.test=c("friedman","t.bonferroni","t.holm","t.none"),
           conf.level=0.95, first.test=5, interactive=TRUE,
              log.file="", no.slaves=0,...)
```

Arguments

wrapper.file	The name of a file containing the definition of the functions to be provided by the user: i.e. race.wrapper and race.info. The file wrapper.file might also define the functions race.init and race.describe.
maxExp	Maximum number of experiments (i.e. evaluations of the function race.wrapper) that are allowed before selecting the best candidate. If maxExp=0, no limit is imposed... very unrealistic in practice.
stat.test	Statistical test to be used for discarding inferior candidates.
conf.level	The confidence level to be used for the statistical test.
first.test	The first test for discarding inferior candidates is performed only when all candidates have been evaluated on a minimum number of tasks equal to first.test.
interactive	If TRUE, print a progress report on the standard output.
log.file	File for saving periodically the state of the race.
no.slaves	When running under PVM, no.slaves specify the number of slaves to be spawned. If no.slave=0 PVM is not used and all experiments are performed on the local host.
...	All extra parameters are passed to the function race.init defined by the user in the file wrapper.file.

Details

This package implemets some racing procedures for selecting from a set of candidate the one that is able to yield the best performance on a given set of tasks. The time available for selecting the best candidate is limited and, therefore, a brute-force approach is unfeasible. The algorithm implemented in this package sequentially evaluates the set of candidatas on the available tasks while discards bad candidates as soon as statistically sufficient evidence is gathered against them. The elimination of inferior candidates, speeds up the procedure and allows a more reliable evaluation of the promising ones.

Value

The output of `race` is a list containing the following components:

`precis`	A string describing the race for documentation purposes.
`results`	A matrix containing in position `[i,j]` the result obtained by candidate `j` on task `i`.
`no.candidates`	Number of candidates at the beginning of the race.
`no.tasks`	Number of tasks on which the selection was based.
`no.subtasks`	Number of subtasks composing each tasks. Default=1
`no.experiments`	Number of times that the function `race.wrapper` had to be call in order to select the best.
`no.alive`	Number of candidates that completed the race, that is, number of candidates that had not been discarded at the moment in which the race was stopped.
`alive`	List of the candidates that completed the race: no sufficient evidence was gathered, give that the test `stat.test` is adopted, for stating that these candidates are worse than the selected best.
`alive.inTime`	Number of candidates in the race after each time step.
`best`	The candidate selected in the race.
`mean.best`	The average result of the best on the tasks considered.
`description.best`	An object describing the selected candidate.
`timestamp.start`	Time stamp of the beginning of the race.
`timestamp.end`	Time stamp of the end of the race.

Note

Please notice that `race` is a **minimization** algorithm: it selects the candidate that obtains the smallest results on the various tasks considered.

Author(s)

Mauro Birattari

References

O. Maron and A.W. Moore (1994) Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. *Advances in Neural Information Processing Systems 6*, pp. 59–66. Morgan Kaufmann.

A.W. Moore and M.S. Lee (1994) Efficient Algorithms for Minimizing Cross Validation Error. *International Conference on Machine Learning*, pp. 190–198. Morgan Kaufmann.

O. Maron and A.W. Moore (1997) The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, **11**(1–5), pp. 193–225.

M. Birattari, T. Stuetzle, L. Paquete, and K. Varrentrapp (2002) A Racing Algorithm for Configuring Metaheuristics. *GECCO 2002: Genetic and Evolutionary Computation Conference*, pp. 11–18. Morgan Kaufmann.

M. Birattari (2004) *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD Thesis, Universite' Libre de Bruxelles, Brussels, Belgium.

See Also

`race.wrapper`, `race.info`, `race.init`, `race.describe`

Examples

```
# The wrapper and init functions for this example are defined in the
# file examples/example-wrapper.R in the installation directory of the
# package.  Please, have a look at such file before implementing your
# own wrapper.
# This example require the package `nnet'
if (require(nnet)&&require(datasets)){
  example.wrapper<-file.path(system.file(package="race"),
                             "examples","example-wrapper.R")
  # Run the race
  race(example.wrapper)

  # If the package `rpvm' is installed on your system and if PVM is
  # properly installed and configured, you can try the following:
  #race(example.wrapper,no.slaves=6)
}
```

`race.describe` *Describe a candidate*

Description

The function

```
race.describe(candidate,data)
```

may be provided by the user for giving a description of a candidate. It's definition has to be given in the same file in which the functions `race.wrapper` and `race.info` are defined. The name of such file has to be passed as first argument to the function `race`.

Arguments

`candidate`	The candidate for which a description is to be returned.
`data`	It is the object of type `list` (possibly empty) returned by `race.init`, if the latter is defined by the user.

Value

The function `race.describe` should return an object describing the selected candidate. Such object will be printed by `race` through the function `print`.

Author(s)

Mauro Birattari

See Also

`race`, `race.init`

Examples

```
# Please have a look at the function `race.describe'
# defined in the file `example-wrapper.R':
local({
  source(file.path(system.file(package="race"),
                           "examples","example-wrapper.R"),local=TRUE);
  print(race.describe)})
```

`race.info` *Provide information on the race*

Description

The function

```
race.info(data)
```

is to be provided by the user. It's definition has to be given (together with the one of `race.wrapper`) in a file, and the name of such file has to be passed as first argument to the function `race`.

Arguments

`data`	It is the object of type `list` (possibly empty) returned by `race.init`, if the latter is defined by the user.

Value

The function `race.info` is expected to return a list including the following components:

`race.name`	The name of the race for documentation purposes.
`no.candidates`	The number of candidates in the race.
`no.tasks`	Number of tasks available for testing.

`no.subtasks` Each task might consists of `no.subtasks` subtasks. If the element `no.subtasks` is not included in the list, it is assumed that each task is indeed atomic, that is, `no.subtasks=1`. `no.subtasks` may also be a vector of length `no.tasks`. In this case, the i-th task consists of `no.subtasks[i]` subtasks.

`extra` A character string providing extra information on the race for documentation purposes. It can be a long string and the user is not required to introduce newline characters: it will be automatically formatted for pretty-printing.

Author(s)

Mauro Birattari

See Also

`race`, `race.init`

Examples

```
# Please have a look at the function `race.info'
# defined in the file `example-wrapper.R':
local({
  source(file.path(system.file(package="race"),
                           "examples","example-wrapper.R"),local=TRUE);
  print(race.info)})
```

---

race.init *Initialization function*

Description

The function

```
race.init()
```

may be provided by the user for initializing the computation of the slave processes. It's definition has to be given in the same file in which the functions `race.wrapper` and `race.info` are defined. The name of such file has to be passed as first argument to the function `race`.

Arguments

The function `race.init` has to be called with no arguments.

Details

This function should be used for initializing the computation on each slave, e.g. loading libraries or data needed by `race.wrapper`, `race.info`, and/or `race.describe`. The output of `race.init` will be passed to these functions.

Value

The function `race.init` is expected to return an object of mode list.

Author(s)

Mauro Birattari

See Also

`race`, `race.wrapper`, `race.info`, `race.describe`

Examples

```
# Please have a look at the function `race.init'
# defined in the file `example-wrapper.R':
local({
  source(file.path(system.file(package="race"),
                           "examples","example-wrapper.R"),local=TRUE);
  print(race.init)})
```

`race.wrapper` *Test a candidate on a task*

Description

The function

```
race.wrapper(candidate,task,data)
```

is to be provided by the user. It's definition has to be given (together with the one of `race.info`) in a file, and the name of such file has to be passed as first argument to the function `race`.

Arguments

`candidate`	The candidate to be evaluated: a number between 1 and `no.candidates`, where `no.candidates` is the number of candidates and is to be defined within the function `race.wrapper` itself.
`task`	The task on which to the candidate should be evaluated: a number between 1 and `no.tasks`, where `no.tasks` is the number of tasks available for testing, and is to be defined within the function `race.wrapper` itself.
`data`	It is the object of type `list` (possibly empty) returned by `race.init`, if the latter is defined by the user.

Value

A number: the result obtained by the given candidate at the given task. If `no.subtasks>1` (see `race.info`), the function is expected to return a vector of length equal to `no.subtasks` where the component `k` of such vector is the result obtained by the given candidate on the `k`-th subtask composing the given task.

Note

Please notice that race is a **minimization** algorithm: it selects the candidate that obtains the smallest results on the various tasks considered.

Author(s)

Mauro Birattari

See Also

race, race.init, race.info

Examples

```
# Please have a look at the function `race.wrapper'
# defined in the file `example-wrapper.R':
local({
  source(file.path(system.file(package="race"),
                           "examples","example-wrapper.R"),local=TRUE);
  print(race.wrapper)})
```

Index