Package 'CAISEr'

October 29, 2017

Type Package

Title Comparison of Algorithms with Iterative Sample Size Estimation

Version 0.2.1

Date 2017-10-27

Maintainer Felipe Campelo <fcampelo@ufmg.br>

Description Functions for performing experimental comparisons of algorithms using adequate sample sizes for power and accuracy.

License GPL-2

Depends R (>= 3.4.0)

Imports assertthat (>= 0.1),

Suggests MOEADr, smoof, knitr, rmarkdown, car, dplyr, ggplot2, ggridges

LazyData TRUE

Encoding UTF-8

RoxygenNote 6.0.1

VignetteBuilder knitr

URL https://github.com/fcampelo/CAISEr

NeedsCompilation no

Author Felipe Campelo [aut, cre], Fernanda Takahashi [aut]

Repository CRAN

Date/Publication 2017-10-29 14:45:27 UTC

R topics documented:

`boot_sdm` *Bootstrap the sampling distribution of the mean*

Description

Bootstraps the sampling distribution of the means for a given vector of observations

Usage

```
boot_sdm(x, boot.R = 999, seed = NULL)
```

Arguments

`x`	vector of observations
`boot.R`	(optional) number of bootstrap resamples #@param ncpus (optional) number of cores to use #//DoParallel
`seed`	seed for the PRNG

Value

vector of bootstrap estimates of the sample mean

References

- A.C. Davison, D.V. Hinkley: Bootstrap methods and their application. Cambridge University Press (1997)
- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
x <- rnorm(15, mean = 4, sd = 1)
my.sdm <- boot_sdm(x)
hist(my.sdm)
qqnorm(my.sdm, pch = 20)

x <- runif(12)
my.sdm <- boot_sdm(x)
qqnorm(my.sdm, pch = 20)

# Convergence of the SDM to a Normal distribution as sample size is increased
X <- rchisq(1000, df = 3)
x1 <- rchisq(10, df = 3)
x2 <- rchisq(20, df = 3)
x3 <- rchisq(40, df = 3)
par(mfrow = c(2, 2))
plot(density(X), main = "Estimated pop distribution");
hist(boot_sdm(x1), breaks = 25, main = "SDM, n = 10")
hist(boot_sdm(x2), breaks = 25, main = "SDM, n = 20")
hist(boot_sdm(x3), breaks = 25, main = "SDM, n = 40")
par(mfrow = c(1, 1))
```

---

calc_instances *Calculates number of instances for the comparison of two algorithms*

Description

Calculates either the number of instances, the standardized effect size, or the power of a comparison of two algorithms

Usage

```
calc_instances(ninstances = NULL, power = NULL, d, sig.level = 0.05,
  alternative = "two.sided", test.type = "t.test")
```

Arguments

ninstances	the number of instances to be used in the experiment.
power	(desired) test power
d	minimally relevant effect size (MRES, expressed as a standardized effect size, i.e., "deviation from H0" / "standard deviation")
sig.level	significance level (alpha) for the experiment
alternative	type of alternative hypothesis ("two.sided" or "one.sided")
test.type	type of test ("t.test", "wilcoxon", "binomial")

Details

This routine uses the closed formula of the t-test to calculate the number of instances required for a comparison of two algorithms, considering a desired power, standardized effect size, and significance level. for cases where the number of instances is predefined, it can return the test power instead.

Value

a list object containing the following items:

- `ninstances` - number of instances
- `power` - the power of the comparison
- `d` - the effect size
- `sig.level` - significance level
- `alternative` - type of alternative hypothesis
- `test.type` - type of test

Sample Sizes for Nonparametric Methods

If the parameter `test.type` is set to either `Wilcoxon` or `Binomial`, this routine approximates the number of instances using the ARE of these tests in relation to the paired t.test, using the formulas:

- $$n.wilcox = n.ttest/0.86 = 1.163 * n.ttest$$
- $$n.binom = n.ttest/0.637 = 1.570 * n.ttest$$

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>), Fernanda Takahashi (<fernandact@ufmg.br>)

`calc_nreps2` *Determine sample sizes for a pair of algorithms on a problem instance*

Description

Iteratively calculates the required sample sizes for two algorithms on a given problem instance, so that the standard error of the estimate of the difference (either simple or percent) in mean performance is controlled at a predefined level.

Usage

```
calc_nreps2(instance, algorithm1, algorithm2, se.max, dif, method = "param",
  nstart = 20, nmax = 200, seed = NULL, boot.R = 999,
  force.balanced = FALSE)
```

Arguments

`instance`	a list object containing the definitions of the problem instance. See Section *Problems and Algorithms* for details.
`algorithm1`	a list object containing the definitions of algorithm 1. See Section *Problems and Algorithms* for details.
`algorithm2`	a list object containing the definitions of algorithm 2. See Section *Problems and Algorithms* for details.
`se.max`	desired upper limit for the standard error of the estimated difference between the two algorithms. See Section *Types of Differences* for details.
`dif`	type of difference to be used. Accepts "perc" (for percent differences) or "simple" (for simple differences)
`method`	method to use for estimating the standard error. Accepts "param" (for parametric) or "boot" (for bootstrap)
`nstart`	initial number of algorithm runs for each algorithm. See Section *Initial Number of Observations* for details.
`nmax`	maximum total allowed sample size.
`seed`	seed for the random number generator
`boot.R`	number of bootstrap resamples
`force.balanced`	logical flag to force the use of balanced sampling for the algorithms on each instance #@param ncpus number of cores to use (under development.) #//DoParallel

Value

a list object containing the following items:

- `x1j` - vector of observed performance values for `algorithm1`
- `x2j` - vector of observed performance values for `algorithm2`
- `phi.est` - estimated value for the statistic of interest
- `se` - standard error of the estimate
- `n1j` - number of observations generated for algorithm 1
- `n2j` - number of observations generated for algorithm 2
- `r.opt = n1j / n2j`
- `seed` - the seed used for the PRNG
- `dif` - the type of difference used
- `method` - the method used ("param" / "boot")

Instances and Algorithms

Parameters `instance`, `algorithm1` and `algorithm2` must each be a list of instance (algorithm) specifications, defined according to the instructions given below.

`instance` is a named list containing all relevant parameters that define the problem instance. This list must contain at least the field `instance$FUN`, with the name of the problem instance function,

that is, a routine that calculates y = f(x). If the instance requires additional parameters, these must also be provided as named fields.

Similarly, `algorithm1` and `algorithm2` must each be a named list containing all relevant parameters that define the algorithm to be applied for solving the problem instance. In what follows we use `algorithm` to refer to both `algorithm1` and `algorithm2`

`algorithm` must contain a `algorithm$FUN` field (the name of the function that calls the algorithm) and any other elements/parameters that `algorithm$FUN` requires (e.g., stop criteria, operator names and parameters, etc.).

The function defined by the routine `algorithm$FUN` must have the following structure: supposing that the list in `algorithm` has fields `algorithm$FUN = myalgo` and `algorithm$par1 = "a", algorithm$par2 = 5`, then:

```
myalgo <- function(par1, par2, instance, ...){
      # do stuff
      # ...
      return(results)
}
```

That is, it must be able to run if called as:

```
# remove '$FUN' field from list of arguments
# and include the problem definition as field 'instance'
myargs          <- algorithm[names(algorithm) != "FUN"]
myargs          <- myargs[names(myargs) != "alias"]
myargs$instance <- instance

# call function
do.call(algorithm$FUN,
        args = myargs)
```

The `algorithm$FUN` routine must return a list containing (at least) the performance value of the final solution obtained, in a field named `value` (e.g., `result$value`) after a given run.

Initial Number of Observations

In the **general case** the initial number of observations / algorithm / instance (`nstart`) should be relatively high. For the parametric case we recommend ~20 if outliers are not expected, ~50 (at least) if that assumption cannot be made. For the bootstrap approach we recommend using at least 20. However, if some distributional assumptions can be made - particularly low skewness of the population of algorithm results on the test instances), then `nstart` can in principle be as small as 5 (if the output of the algorithm were known to be normal, it could be 1).

In general, higher sample sizes are the price to pay for abandoning distributional assumptions. Use lower values of `nstart` with caution.

Types of Differences

Parameter `dif` informs the type of difference in performance to be used for the estimation (mu1 and mu2 represent the mean performance of each algorithm on the problem instance):

- If `dif == "perc"` it estimates (mu2 - mu1) / mu1.
- If `dif == "simple"` it estimates mu2 - mu1.

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>), Fernanda Takahashi (<fernandact@ufmg.br>)

References

- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).
- P. Mathews. Sample size calculations: Practical methods for engineers and scientists. Mathews Malnar and Bailey, 2010.
- A.C. Davison, D.V. Hinkley: Bootstrap methods and their application. Cambridge University Press (1997)
- E.C. Fieller: Some problems in interval estimation. Journal of the Royal Statistical Society. Series B (Methodological) 16(2), 175–185 (1954)
- V. Franz: Ratios: A short guide to confidence limits and proper use (2007). https://arxiv.org/pdf/0710.2024v1.pdf
- D.C. Montgomery, C.G. Runger: Applied Statistics and Probability for Engineers, 6th ed. Wiley (2013)

Examples

```
# Uses dummy algorithms and a dummy instance to illustrate the
# use of calc_nreps2
algorithm1 <- list(FUN = "dummyalgo", alias = "algo1",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 10, sd = 1))
algorithm2 <- list(FUN = "dummyalgo", alias = "algo2",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 20, sd = 4))
instance <- list(FUN = "dummyinstance")

# Theoretical results for an SE = 0.5 on the simple difference:
# phi = 10; n1 = 20; n2 = 80
# (using the parametric approach)
my.reps  <- calc_nreps2(instance, algorithm1, algorithm2,
                        se.max = 0.5, dif = "simple", seed = 1234)
cat("n1j   =", my.reps$n1j, "\nn2j   =", my.reps$n2j,
    "\nphi_j =", my.reps$phi.est, "\nse    =", my.reps$se)

# Forcing equal sample sizes:
my.reps  <- calc_nreps2(instance, algorithm1, algorithm2,
                        se.max = 0.5, dif = "simple", seed = 1234,
                        force.balanced = TRUE)
```

```
cat("n1j   =", my.reps$n1j, "\nn2j   =", my.reps$n2j,
    "\nphi_j =", my.reps$phi.est, "\nse    =", my.reps$se)

## Not run:
# Using the bootstrap approach
algorithm3 <- list(FUN = "dummyalgo", alias = "algo3",
                   distribution.fun = "rchisq",
                   distribution.pars = list(df = 2, ncp = 3))

my.reps  <- calc_nreps2(instance, algorithm1, algorithm3,
                        se.max = 0.05, dif = "perc",
                        method = "boot", seed = 1234,
                        nstart = 20)
cat("n1j   =", my.reps$n1j, "\nn2j   =", my.reps$n2j,
    "\nphi_j =", my.reps$phi.est, "\nse    =", my.reps$se)

## End(Not run)
```

`calc_phi` *Calculates the sample estimator of (simple or percent) differences*

Description

Calculates the sample estimator of the (simple or percent) differences between the means of two algorithms on a given instance.

- If 'dif == "simple": estimates $mu2 - mu1$,
- If 'dif == "perc": estimates $(mu2 - mu1)/mu1$,

Usage

```
calc_phi(x1, x2, dif)
```

Arguments

`x1`	vector of observations
`x2`	vector of observations
`dif`	type of difference to estimate ("simple" or "perc")

Details

where $mu1, mu2$ are the means of the populations that generated the sample vectors $x1, x2$.

Value

Estimated value of the statistic given in `dif`

References

- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
x1 <- rnorm(25, 3, 0.5)
x2 <- runif(15, 4, 6)
calc_phi(x1, x2, "simple")
calc_phi(x1, x2, "perc")
```

---

calc_power_curve *Calculate the power curve for an experiment*

Description

Calculates the power curve (d x power) for an experiment with a fixed number of instances. See `calc_instances()` for details.

Usage

```
calc_power_curve(ninstances, sig.level = 0.05, alternative = "two.sided",
  test.type = "t.test", npoints = 100)
```

Arguments

`ninstances`	the number of instances to be used in the experiment.
`sig.level`	significance level (alpha) for the experiment
`alternative`	type of alternative hypothesis ("two.sided" or "one.sided")
`test.type`	type of test ("t.test", "wilcoxon", "binomial")
`npoints`	number of points for the power curve.

Value

an object of class `caiser.powercurve` containing fields `d`, the (standardized) effect size; and `power`, the (expected) power to detect each effect size in `d`.

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
my.cpc <- calc_power_curve(ninstances = 10)
summary(my.cpc)
plot(my.cpc)
```

`calc_ropt` *Calculates the optimal ratio of sample sizes*

Description

Calculates the optimal ratio of sample sizes of two algorithms on a given instance, for either simple or percent differences, using the parametric approach.

Usage

```
calc_ropt(x1, x2, dif)
```

Arguments

`x1`	vector of observations
`x2`	vector of observations
`dif`	name of the difference for which the SE is desired. Accepts "perc" (for percent differences) or "simple" (for simple differences)

Value

numeric value: optimal ratio n1 / n2

References

- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
set.seed(1234)
x1 <- rnorm(25, 5, 1)
x2 <- runif(35, 8, 10)
calc_ropt(x1, x2, "simple")
calc_ropt(x1, x2, "perc")
```

`calc_se` *Calculates the standard error for simple and percent differences*

Description

Calculates the sample standard error for the estimator differences between two algorithms on a given instance.

Usage

```
calc_se(x1, x2, dif, method = "param", boot.R = 999)
```

Arguments

`x1`	vector of observations
`x2`	vector of observations
`dif`	name of the difference for which the SE is desired. Accepts "perc" (for percent differences) or "simple" (for simple differences)
`method`	method used to calculate the interval. Accepts "param" (using parametric formulas based on normality of the sampling distribution of the means) or "boot" (for bootstrap).
`boot.R`	(optional) number of bootstrap resamples.

Details

- If `dif == "perc"` it returns the SE for sample estimator of $(mu2 - mu1)/mu1$, where $mu1, mu2$ are the means of the populations that generated the sample vectors $x1, x2$.
- If `dif == "simple"` it returns the SE for sample estimator of $(mu2 - mu1)$

Value

a list object containing the following items:

- `x.est` - estimated value
- `se` - standard error

References

- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>), Fernanda Takahashi (<fernandact@ufmg.br>)

Examples

```
# two vectors of normally distributed observations
set.seed(1234)
x1 <- rnorm(100, 5, 1)  # mean = 5, sd = 1
x2 <- rnorm(200, 10, 2) # mean = 10, sd = 2

# Theoretical SE for simple difference: 0.1732051
calc_se(x1, x2, dif = "simple", method = "param")

# Theoretical (Fieller, no covariance) SE for percent differences: 0.04
calc_se(x1, x2, dif = "perc", method = "boot")
```

---

dummyalgo *Dummy algorithm routine to test the sampling procedures*

Description

This is a dummy algorithm routine to test the sampling procedures. It basically returns values according to the distribution function given in `distribution.fun` with parameters given by `distribution.pars`.

Usage

```
dummyalgo(distribution.fun = "rnorm", distribution.pars = list(mean = 0, sd
  = 1), instance)
```

Arguments

`distribution.fun`	name of a function that generates random values according to a given distribution, e.g., "rnorm", "runif", "rexp" etc.
`distribution.pars`	list of named parameters required by the function in `distribution.fun`. Parameter `n` (number of points to generate) is unnecessary (this routine always considers `n = 1`).
`instance`	instance parameters. This parameter is always ignored, and was only included for compatibility with `run_nreps2` and other `nreps` functions.

Value

a list object with a single field `$value`, containing a value distributed according to `distribution.fun` and `distribution.pars`.

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

`dummyinstance` *Dummy instance (for testing only) - a function that does nothing and returns nothing*

Description

Dummy instance (for testing only) - a function that does nothing and returns nothing

Usage

```
dummyinstance()
```

`get_observations` *Run an algorithm on a problem.*

Description

Call algorithm routine for the solution of a problem instance

Usage

```
get_observations(algo, instance, n = 1)
```

Arguments

`algo`	a list object containing the definitions of the algorithm. See `calc_nreps2()` for details.
`instance`	a list object containing the definitions of the problem instance. See `calc_nreps2()` for details.
`n`	number of observations to generate.

Value

vector of observed performance values

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

See Also

calc_nreps2

Examples

```
algorithm <- list(FUN = "dummyalgo", alias = "myalgo",
                  distribution.fun = "rnorm",
                  distribution.pars = list(mean = 50, sd = 10))
instance <- list(FUN = "dummyinstance")
x <- get_observations(algorithm, instance, n = 1000)
hist(x)
```

`plot.CAISErPowercurve` *plot.caiser.powercurve*

Description

S3 method for plotting *caiser.powercurve* objects (the output of `calc_power_curve()`).

Usage

```
## S3 method for class 'CAISErPowercurve'
plot(x, ...)
```

Arguments

`x`	list object of class *caiser.powercurve* (generated by `calc_power_curve()`)
`...`	other parameters to be passed down to specific plotting functions (currently unused)

Examples

```
my.cpc <- calc_power_curve(ninstances = 10)
plot(my.cpc)
```

`print.CAISEr` *print.CAISEr*

Description

S3 method for printing *CAISEr* objects (the output of `run_experiment()`).

Usage

```
## S3 method for class 'CAISEr'
print(x, ..., digits = 6, right = TRUE,
  breakrows = FALSE)
```

Arguments

x	list object of class *CAISEr* (generated by run_experiment())
...	other parameters to be passed down to specific summary functions (currently unused)
digits	the minimum number of significant digits to be used. See print.default().
right	logical, indicating whether or not strings should be right-aligned.
breakrows	logical, indicating whether to "widen" the output table by placing the bottom half to the right of the top half.

Examples

```
# Example using dummy algorithms and instances. See ?dummyalgo for details.
# In this case all instances are the same, so we expect all cases to return
# a percent difference of approx. phi.j = 1.0 and sample sizes of
# approx. n1 = 31, n2 = 87
algorithm1 <- list(FUN = "dummyalgo", alias = "algo1",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 10, sd = 1))
algorithm2 <- list(FUN = "dummyalgo", alias = "algo2",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 20, sd = 4))
algolist <- list(algorithm1, algorithm2)
instlist <- vector(100, mode = "list")
for (i in 1:100) instlist[[i]] <- list(FUN = "dummyinstance",
                                       alias = paste0("Inst. ", i))

out <- run_experiment(Instance.list = instlist,
                      Algorithm.list = algolist,
                      power = 0.8,
                      d = 1,
                      sig.level = 0.01,
                      se.max = 0.05,
                      dif = "perc",
                      nmax   = 200,
                      seed   = 1234)
out
```

---

run_experiment *Run a full experiment*

Description

Design and run a full experiment - calculate the required number of instances, run the algorithms on each problem instance using the iterative approach based on optimal sample size ratios, and return the results of the experiment. This routing builds upon calc_instances() and calc_nreps2(), so refer to the documentation of these two functions for details.

Usage

```
run_experiment(Instance.list, Algorithm.list, power, d, sig.level = 0.05,
  alternative = "two.sided", test.type = "t.test", se.max, dif,
  method = "param", nstart = 20, nmax = 1000, seed = NULL,
  boot.R = 999, force.balanced = FALSE)
```

Arguments

Instance.list	list object containing the definitions of the *available* instances. this list may (or may not) be exhausted in the experiment. To estimate the number of required instances, see calc_instances(). For more detail on the definition of each instance, see calc_nreps2().
Algorithm.list	list object containing the definitions of the algorithms to be compared. See calc_nreps2() for details.
power	(desired) test power. See calc_instances() for details.
d	minimally relevant effect size (MRES), expressed as a standardized effect size, i.e., "deviation from H0" / "standard deviation". See calc_instances() for details.
sig.level	significance level (alpha) for the experiment. See calc_instances() for details.
alternative	type of alternative hypothesis ("two.sided" or "one.sided"). See calc_instances() for details.
test.type	type of test ("t.test", "wilcoxon", "binomial"). See calc_instances() for details.
se.max	desired upper limit for the standard error of the estimated difference between the two algorithms on each instance. See calc_nreps2() for details.
dif	type of difference to be used on each instance. Accepts "perc" (for percent differences) or "simple" (for simple differences). See calc_nreps2() for details.
method	method to use for estimating the standard errors. Accepts "param" (for parametric) or "boot" (for bootstrap). See calc_nreps2() for details.
nstart	initial number of algorithm runs for each algorithm in each instance. See calc_nreps2() for details.
nmax	maximum total allowed sample size in each instance See calc_nreps2() for details.
seed	seed for the random number generator
boot.R	number of bootstrap resamples. See calc_nreps2() for details.
force.balanced	logical flag to force the use of balanced sampling for the algorithms on each instance

Value

a list object containing the full input configuration plus the following fields:

- data.raw - data frame containing all observations generated

- `data.summary` - data frame summarizing the experiment.
- `N` - number of instances sampled
- `N.star` - number of instances required
- `instances.sampled` - names of the instances sampled
- `Underpowered` - flag: TRUE if N < N.star

Instance List

Parameter `Instance.list` must contain a list of instance objects, where each field is itself a list, as defined in Section _Instances and Algorithms of the documentation of `_calc_nreps2()`. In summary, each element of `Instance.list` is an `instance`, i.e., a named list containing all relevant parameters that define the problem instance. This list must contain at least the field `instance$FUN`, with the name of the problem instance function, that is, a routine that calculates y = f(x). If the instance requires additional parameters, these must also be provided as named fields. An additional field, "instance$alias", can be used to provide the instance with a unique identifier (e.g., when using an instance generator).

Algorithms

Parameter `Algorithm.list` must contain a list of instance objects, where each field is itself a list, as defined in Section *Instances and Algorithms* of the documentation of `calc_nreps2()`. In summary, each element of `Algorithm.list` is an `algorithm`, i.e., a named list containing all relevant parameters that define the algorithm.

An `algorithm` must contain a `algorithm$FUN` field (the name of the function that calls the algorithm) and any other elements/parameters that `algorithm$FUN` requires (e.g., stop criteria, operator names and parameters, etc.). An additional field, `algorithm$alias`, can be used to provide the algorithm with a unique identifier (e.g., when comparing two different configurations of the same algorithm).

The function defined by the routine `algorithm$FUN` must have the following structure: supposing that the list in `algorithm` has fields `algorithm$FUN = myalgo` and `algorithm$par1 = "a", algorithm$par2 = 5`, then:

```
myalgo <- function(par1, par2, instance, ...){
      # do stuff
      # ...
      return(results)
}
```

That is, it must be able to run if called as:

```
# remove '$FUN' and '$alias' from list of arguments
# and include the problem definition as field 'instance'
myargs          <- algorithm[names(algorithm) != "FUN"]
myargs          <- myargs[names(myargs) != "alias"]
myargs$instance <- instance
```

```
# call function
do.call(algorithm$FUN,
        args = myargs)
```

The `algorithm$FUN` routine must return a list object containing (at least) the performance value of the final solution obtained after a given run, in a field named `value` (e.g., `result$value`) .

Initial Number of Observations

In the *general case* the initial number of observations / algorithm / instance (`nstart`) should be relatively high. For the parametric case we recommend ~15 if outliers are not expected, ~50 (at least) if that assumption cannot be made. For the bootstrap approch we recommend using at least 15 or 20. However, if some distributional assumptions can be made - particularly low skewness of the population of algorithm results on the test instances), then `nstart` can in principle be as small as 5 (if the output of the algorithm were known to be normal, it could be 1).

In general, higher sample sizes are the price to pay for abandoning distributional assumptions. Use lower values of `nstart` with caution.

Types of Differences

Parameter `dif` informs the type of difference in performance to be used for the estimation (mu1 and mu2 represent the mean performance of each algorithm on the problem instance):

- If `dif == "perc"` it estimates (mu2 - mu1) / mu1.
- If `dif == "simple"` it estimates mu2 - mu1.

Sample Sizes for Nonparametric Methods

If the parameter `test.type` is set to either `Wilcoxon` or `Binomial`, this routine approximates the number of instances using the ARE of these tests in relation to the paired t.test, using the formulas:

- `n.wilcox = n.ttest / 0.86 = 1.163 * n.ttest`
- `n.binom = n.ttest / 0.637 = 1.570 * n.ttest`

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

References

- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).
- P. Mathews. Sample size calculations: Practical methods for engineers and scientists. Mathews Malnar and Bailey, 2010.
- A.C. Davison, D.V. Hinkley: Bootstrap methods and their application. Cambridge University Press (1997)
- E.C. Fieller: Some problems in interval estimation. Journal of the Royal Statistical Society. Series B (Methodological) 16(2), 175–185 (1954)

- V. Franz: Ratios: A short guide to confidence limits and proper use (2007). https://arxiv.org/pdf/0710.2024v1.pdf
- D.C. Montgomery, C.G. Runger: Applied Statistics and Probability for Engineers, 6th ed. Wiley (2013)

Examples

```
# Example using dummy algorithms and instances. See ?dummyalgo for details.
# In this case all instances are the same, so we expect all cases to return
# a percent difference of approx. phi.j = 1.0 and sample sizes of
# approx. n1 = 31, n2 = 87
algorithm1 <- list(FUN = "dummyalgo", alias = "algo1",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 10, sd = 1))
algorithm2 <- list(FUN = "dummyalgo", alias = "algo2",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 20, sd = 4))
algolist <- list(algorithm1, algorithm2)
instlist <- vector(100, mode = "list")
for (i in 1:100) instlist[[i]] <- list(FUN = "dummyinstance",
                                       alias = paste0("Inst. ", i))

my.results <- run_experiment(Instance.list = instlist,
                             Algorithm.list = algolist,
                             power = 0.8,
                             d = 1,
                             sig.level = 0.01,
                             se.max = 0.05,
                             dif = "perc",
                             nmax   = 200,
                             seed   = 1234)

# Take a look at the summary table
my.results$data.summary

# To perform inference on the results:
t.test(my.results$data.summary$phi.j, conf.level = 0.95)

# Test assumption of normality (of the data)
shapiro.test(my.results$data.summary$phi.j)
```

---

se_boot *Bootstrap standard errors*

Description

Calculates the standard error of a given statistic using bootstrap

Usage

```
se_boot(x1, x2, dif, boot.R = 999)
```

Arguments

x1	vector of observations
x2	vector of observations
dif	name of the difference for which the SE is desired. Accepts "simple" (simple differences) or "perc" (percent differences).
boot.R	(optional) number of bootstrap resamples

Value

estimated standard error

References

- A.C. Davison, D.V. Hinkley: Bootstrap methods and their application. Cambridge University Press (1997)
- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
# two vectors of normally distributed observations
set.seed(1234)
x1 <- rnorm(100, 5, 1)  # mean = 5, sd = 1
x2 <- rnorm(200, 10, 2) # mean = 10, sd = 2

# Theoretical SE for simple difference: 0.1732051
se_boot(x1, x2, dif = "simple")

# Theoretical (Fieller, no covariance) SE for percent differences: 0.04
se_boot(x1, x2, dif = "perc")
```

---

se_param *Parametric standard errors*

Description

Calculates the standard error of a given statistic using parametric formulas

Usage

```
se_param(x1, x2, dif, ...)
```

Arguments

x1	vector of observations
x2	vector of observations
dif	name of the difference for which the SE is desired. Accepts "simple" (simple differences) or "perc" (percent differences).
...	other parameters (used only for compatibility with calls to se_boot(), unused in this function)

Value

estimated standard error

References

- E.C. Fieller: Some problems in interval estimation. Journal of the Royal Statistical Society. Series B (Methodological) 16(2), 175–185 (1954)
- V. Franz: Ratios: A short guide to confidence limits and proper use (2007). https://arxiv.org/pdf/0710.2024v1.pdf
- D.C. Montgomery, C.G. Runger: Applied Statistics and Probability for Engineers, 6th ed. Wiley (2013)
- F. Campelo, F. Takahashi: Sample size estimation for power and accuracy in the experimental comparison of algorithms (submitted, 2017).

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>)

Examples

```
# two vectors of normally distributed observations
set.seed(1234)
x1 <- rnorm(100, 5, 1)  # mean = 5, sd = 1
x2 <- rnorm(200, 10, 2) # mean = 10, sd = 2

# Theoretical SE for simple difference: 0.1732051
se_param(x1, x2, dif = "simple")

# Theoretical (Fieller, no covariance) SE for percent differences: 0.04
se_param(x1, x2, dif = "perc")
```

summary.CAISEr *summary.CAISEr*

Description

S3 method for summarizing *CAISEr* objects (the output of run_experiment()).

Usage

```
## S3 method for class 'CAISEr'
summary(object, ...)
```

Arguments

object	list object of class *CAISEr* (generated by run_experiment())
...	other parameters to be passed down to specific summary functions (currently unused)

Examples

```
# Example using dummy algorithms and instances. See ?dummyalgo for details.
# In this case all instances are the same, so we expect all cases to return
# a percent difference of approx. phi.j = 1.0 and sample sizes of
# approx. n1 = 31, n2 = 87
algorithm1 <- list(FUN = "dummyalgo", alias = "algo1",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 10, sd = 1))
algorithm2 <- list(FUN = "dummyalgo", alias = "algo2",
                   distribution.fun = "rnorm",
                   distribution.pars = list(mean = 20, sd = 4))
algolist <- list(algorithm1, algorithm2)
instlist <- vector(100, mode = "list")
for (i in 1:100) instlist[[i]] <- list(FUN = "dummyinstance",
                                       alias = paste0("Inst. ", i))

out <- run_experiment(Instance.list = instlist,
                      Algorithm.list = algolist,
                      power = 0.8,
                      d = 1,
                      sig.level = 0.01,
                      se.max = 0.05,
                      dif = "perc",
                      nmax   = 200,
                      seed   = 1234)
summary(out)
```

summary.CAISErPowercurve *summary.CAISErPowercurve*

Description

S3 method for summarizing *caiser.powercurve* objects (the output of `calc_power_curve()`).

Usage

```
## S3 method for class 'CAISErPowercurve'
summary(object, ...)
```

Arguments

`object`	list object of class *caiser.powercurve* (generated by `calc_power_curve()`)
`...`	other parameters to be passed down to specific summary functions (currently unused)

Examples

```
my.cpc <- calc_power_curve(ninstances = 10)
summary(my.cpc)
```

Index