# Package: scR (via r-universe)

June 24, 2026

**Title** Empirical Sample Complexity Bounds

**Version** 0.7.0

**Description** Provides tools for estimating empirical sample complexity bounds for supervised learning tasks. The package supports simulation-based estimates of generalization curves, parametric extrapolation of empirical sample complexity bounds, theoretical bounds based on Vapnik-Chervonenkis dimension, and optional monotone Gaussian process extrapolation for users who install the external 'cmdstanr' workflow. For more details, see Carter and Choi (2024) <doi:10.31219/osf.io/evrcj>.

**License** MIT + file LICENSE

**URL** https://github.com/pjesscarter/scR

**BugReports** https://github.com/pjesscarter/scR/issues

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** dplyr, furrr, future, ggplot2, Matrix, minpack.lm, parallel, parallelly, pbapply, plotly, progressr, stats, tidyr

**Suggests** cmdstanr, posterior, rmarkdown, testthat (>= 3.0.0)

**Additional_repositories** https://stan-dev.r-universe.dev

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**Config/pak/sysreqs** cmake make libicu-dev libuv1-dev libssl-dev

**Repository** https://pjesscarter.r-universe.dev

**Date/Publication** 2026-06-23 08:09:01 UTC

**RemoteUrl** https://github.com/pjesscarter/scr

**RemoteRef** HEAD

**RemoteSha** 8fe5769fd130e7de2530c712a8a57240411cb384

# Contents



---

`acc_sim` *Utility function to generate accuracy metrics, for use with* `estimate_accuracy()`

---

## Description

Utility function to generate accuracy metrics, for use with `estimate_accuracy()`

## Usage

```
acc_sim(
  n,
  method,
  p,
  dat,
  model,
  eta,
  nsample,
  outcome,
  power,
  effect_size,
  powersims,
  alpha,
```

```
  split,
  predictfn,
  replacement,
  ...
)
```

## Arguments

| | |
|---|---|
| n | An integer giving the desired sample size for which the target function is to be calculated. |
| method | An optional string stating the distribution from which data is to be generated. Default is i.i.d. uniform sampling. Currently also supports "Class Imbalance". Can also take a function outputting a vector of probabilities if the user wishes to specify a custom distribution. |
| p | If method is 'Class Imbalance', gives the degree of weight placed on the positive class. |
| dat | A rectangular `data.frame` or matrix-like object giving the full data from which samples are to be drawn. If left unspecified, `gendata()` is called to produce synthetic data with an appropriate structure. |
| model | A function giving the model to be estimated |
| eta | A real number between 0 and 1 giving the probability of misclassification error in the training data. |
| nsample | A positive integer giving the number of samples to be generated for each value of $n$. Larger values give more accurate results. |
| outcome | A string giving the name of the outcome variable. |
| power | A logical indicating whether experimental power based on the predictions should also be reported |
| effect_size | If `power` is `TRUE`, a real number indicating the scaled effect size the user would like to be able to detect. |
| powersims | If `power` is `TRUE`, an integer indicating the number of simulations to be conducted at each step to calculate power. |
| alpha | If `power` is `TRUE`, a real number between 0 and 1 indicating the probability of Type I error to be used for hypothesis testing. Default is 0.05. |
| split | A logical indicating whether the data was passed as a single data frame or separately. |
| predictfn | An optional user-defined function giving a custom predict method. If also using a user-defined model, the `model` should output an object of class `"svrclass"` to avoid errors. |
| replacement | A logical flag indicating whether sampling should be performed with replacement. |
| ... | Additional model parameters to be specified by the user. |

## Value

A data frame giving performance metrics for the specified sample size.

`conduct_interpolation` *Conduct interpolation on a single simulation*

## Description

Wrapper function for fitting extrapolation model to a single object of class "scb_data". Allows fitting of custom functions, but for general use interpolate_scb should be used instead.

## Usage

```
conduct_interpolation(
  scbobject,
  epsilon,
  delta,
  maxN,
  delta_formula,
  epsilon_formula,
  delta_lower_bounds = NULL,
  epsilon_lower_bounds = NULL,
  delta_upper_bounds = NULL,
  epsilon_upper_bounds = NULL,
  delta_start = NULL,
  epsilon_start = NULL
)
```

## Arguments

| | |
|---|---|
| `scbobject` | An object of class "scb_data" for interpolation to be conducted on. |
| `epsilon` | A real number between 0 and 1 giving the targeted maximum out-of-sample (OOS) error rate |
| `delta` | A real number between 0 and 1 giving the targeted maximum probability of observing an OOS error rate higher than `epsilon` |
| `maxN` | A positive integer giving value of the largest N for which extrapolation is to be conducted. |
| `delta_formula` | Formula of the form Delta ~ model(n,...) giving the NLS model to be applied to the delta curve. |
| `epsilon_formula` | Formula of the form Epsilon ~ model(n,...) giving the NLS model to be applied to the epsilon curve. |
| `delta_lower_bounds` | Optional named vector of lower bounds for the delta model parameters. |
| `epsilon_lower_bounds` | Optional named vector of lower bounds for the epsilon model parameters. |
| `delta_upper_bounds` | Optional named vector of upper bounds for the delta model parameters. |

epsilon_upper_bounds
: Optional named vector of upper bounds for the epsilon model parameters.

delta_start
: Optional named vector of starting values for the delta model parameters.

epsilon_start
: Optional named vector of starting values for the model parameters.

## Value

A named `list` containing the interpolated dataframe, the original input data frame, and the given values of epsilon, delta, and maxN.

## See Also

`interpolate_scb()` is the main wrapper for interpolation on a list.

---

# create_scb_model *Create custom model fitting function*

---

## Description

Utility function for creating custom classification function for use in SCB calculations.

## Usage

```
create_scb_model(
  model_fun,
  extra_args = list(),
  split = FALSE,
  arg_map = list(formula = "formula", data = "data", x = "x", y = "y")
)
```

## Arguments

model_fun
: A binary classification model supplied by the user, e.g. glm

extra_args
: A list of additional default arguments to be passed to the classification model, e.g. family = binomial(link = "logit")

split
: Logical indicating whether the model expects a single data argument or separate x/y values.

arg_map
: Named list giving mappings for names of formula, data, x, and y arguments expected in `model_fun`. Must include formula and data if split is FALSE or x and y otherwise.

## Value

A function taking supplied arguments that can be passed to other package functions such as `estimate_accuracy()`

## create_scb_prediction *Create custom prediction function*

### Description

Utility function for creating custom prediction function for use in SCB calculations.

### Usage

```
create_scb_prediction(
  predict_fun,
  extra_args = list(),
  transform_fn = identity,
  arg_map = list(m = "m", newdata = "newdata")
)
```

### Arguments

| | |
|---|---|
| predict_fun | A binary prediction model supplied by the user, e.g. predict.glm |
| extra_args | A list of additional default arguments to be passed to the classification model, e.g. type="response" |
| transform_fn | Function giving the transformation, if any, to be applied to the prediction output to generate binary predictions. Defaults to identity() |
| arg_map | Named list giving mappings for names of m and newdata arguments expected in predict_fun. |

### Value

A prediction function taking supplied arguments that can be passed to other package functions such as estimate_accuracy()

## estimate_accuracy *Estimate sample complexity bounds for a binary classification algorithm using either simulated or user-supplied data.*

### Description

Estimate sample complexity bounds for a binary classification algorithm using either simulated or user-supplied data.

## Usage

```
estimate_accuracy(
  formula,
  model,
  data = NULL,
  dim = NULL,
  maxn = NULL,
  sparse = FALSE,
  density = NULL,
  upperlimit = NULL,
  nsample = 30,
  steps = 50,
  eta = 0.05,
  delta = 0.05,
  epsilon = 0.05,
  predictfn = NULL,
  subsample_size = NULL,
  nboot = 1L,
  power = FALSE,
  effect_size = NULL,
  powersims = NULL,
  alpha = 0.05,
  parallel = TRUE,
  coreoffset = 0,
  packages = list(),
  method = c("Uniform", "Class Imbalance"),
  p = NULL,
 minn = ifelse(is.null(data), ifelse(is.null(x), (dim + 1), (ncol(x) + 1)), (ncol(data)
    + 1)),
  x = NULL,
  y = NULL,
  backend = c("multisession", "multicore", "cluster", "sequential"),
  replacement = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| formula | A formula that can be passed to the model argument to define the classification algorithm |
| model | A binary classification model supplied by the user. Must take arguments formula and data |
| data | Optional. A rectangular data.frame object giving the full data from which samples are to be drawn. If left unspecified, gendata() is called to produce synthetic data with an appropriate structure. |
| dim | Required if data is unspecified. Gives the horizontal dimension of the data (number of predictor variables) to be generated. |

| | |
|---|---|
| maxn | Required if data is unspecified. Gives the vertical dimension of the data (number of observations) to be generated. |
| sparse | Optional. A logical giving whether to generate sparse data, if data was not given. |
| density | Real number between 0 and 1 giving the proportion of non 0 entries in the sparse matrix. Used only if sparse is TRUE. |
| upperlimit | Optional. A positive integer giving the maximum sample size to be simulated, if data was supplied. |
| nsample | A positive integer giving the number of samples to be generated for each value of $n$. Larger values give more accurate results. |
| steps | A positive integer giving the interval of values of $n$ for which simulations should be conducted. Larger values give more accurate results. |
| eta | A real number between 0 and 1 giving the probability of misclassification error in the training data. |
| delta | A real number between 0 and 1 giving the targeted maximum probability of observing an OOS error rate higher than epsilon |
| epsilon | A real number between 0 and 1 giving the targeted maximum out-of-sample (OOS) error rate |
| predictfn | An optional user-defined function giving a custom predict method. If also using a user-defined model, the model should output an object of class "svrclass" to avoid errors. |
| subsample_size | An integer giving the size of the initial 'pilot' sample to be simulated. If left as NULL, the input data size will be used (benchmark SCB). |
| nboot | An integer giving the number of SCB bootstraps to be performed. |
| power | A logical indicating whether experimental power based on the predictions should also be reported |
| effect_size | If power is TRUE, a real number indicating the scaled effect size the user would like to be able to detect. |
| powersims | If power is TRUE, an integer indicating the number of simulations to be conducted at each step to calculate power. |
| alpha | If power is TRUE, a real number between 0 and 1 indicating the probability of Type I error to be used for hypothesis testing. Default is 0.05. |
| parallel | Boolean indicating whether or not to use parallel processing. |
| coreoffset | If parallel is true, a positive integer indicating the number of free threads to be kept unused. Should not be larger than the number of CPU cores. |
| packages | A list of packages that need to be loaded in order to run model. |
| method | An optional string stating the distribution from which data is to be generated. Default is i.i.d. uniform sampling. Can also take a function outputting a vector of probabilities if the user wishes to specify a custom distribution. |
| p | If method is 'Class Imbalance', gives the degree of weight placed on the positive class. |
| minn | Optional argument to set a different minimum n than the dimension of the algorithm. Useful with e.g. regularized regression models such as elastic net. |

| | |
|---|---|
| x | Optional argument for methods that take separate predictor and outcome data. Specifies a matrix-like object containing predictors. Note that if used, the x and y objects are bound together columnwise; this must be handled in the user-supplied helper function. |
| y | Optional argument for methods that take separate predictor and outcome data. Specifies a vector-like object containing outcome values. Note that if used, the x and y objects are bound together columnwise; this must be handled in the user-supplied helper function. |
| backend | One of the parallel backends used by `future::plan()`. See function documentation for more details. |
| replacement | A logical flag indicating whether sampling should be performed with replacement. |
| ... | Additional arguments that need to be passed to `model` |

## Value

A `list` containing two named elements. `Raw` gives the exact output of the simulations, while `Summary` gives a table of accuracy metrics, including the achieved levels of $\epsilon$ and $\delta$ given the specified values. Alternative values can be calculated using `getpac()`

## See Also

`plot.scb_data()`, to represent simulations visually, `getpac()`, to calculate summaries for alternate values of $\epsilon$ and $\delta$ without conducting a new simulation, and `gendata()`, to generated synthetic datasets.

## Examples

```
# See the package README for an end-to-end example.
```

---

`fit_and_predict` *Fit an extrapolation model using nonlinear least squares*

---

## Description

Utility function to fit extrapolation model, for use with `conduct_interpolation()`

## Usage

```
fit_and_predict(
  formula,
  data,
  N_grid,
  maxN_obs,
  start = NULL,
  lower = NULL,
  upper = NULL
)
```

### Arguments

| | |
|---|---|
| formula | A formula object giving the model to be fit. |
| data | A data frame giving the data the model is to be fit on. |
| N_grid | An integer vector of N values to conduct interpolation and extrapolation on. |
| maxN_obs | A positive integer giving value of the largest N in the observed data. |
| start | Optional named vector of starting values for the model parameters. |
| lower | Optional named vector of lower bounds for the model parameters. |
| upper | Optional named vector of upper bounds for the model parameters. |

### Value

A fitted model object of the chosen type.

---

## fit_gp_scb_curve *Fit a monotone Gaussian process sample-complexity curve*

---

### Description

Fits the monotone-integrated Gaussian process extrapolator described in the paper appendix. This is the nonparametric curve-fitting option only; it does not generate the resampled accuracy curves. Use `estimate_accuracy()` first, then pass the resulting `scb_data` object to `interpolate_scb_gp()`.

### Usage

```
fit_gp_scb_curve(
  x,
  y,
  curve = c("delta", "epsilon"),
  maxN = NULL,
  M_grid = 120L,
  epsilon0 = NULL,
  stan_file = NULL,
  seed = 2027,
  chains = 4L,
  parallel_chains = chains,
  iter_warmup = 800L,
  iter_sampling = 800L,
  adapt_delta = 0.97,
  max_treedepth = 12L,
  refresh = 100L,
  ci = 0.9,
  init = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| x | Numeric vector of training sample sizes. |
| y | Numeric vector of observed curve values in [0, 1]. |
| curve | Character string; either "delta" for the exceedance-probability curve or "epsilon" for the generalization-error curve. |
| maxN | Largest sample size on the prediction grid. Defaults to the largest observed value in x. |
| M_grid | Number of evenly spaced prediction-grid points before observed sample sizes are added exactly to the grid. |
| epsilon0 | Baseline error rate for the epsilon curve. If NULL, the observed value at the smallest sample size is used. Ignored for the delta curve. |
| stan_file | Optional path to a Stan model. By default the Stan file shipped in inst/stan is used. |
| seed, chains, parallel_chains, iter_warmup, iter_sampling, adapt_delta, max_treedepth, refresh | Sampling controls passed to cmdstanr. |
| ci | Credible interval level for posterior summaries. |
| init | Optional initialization function or list passed to cmdstanr. If NULL, conservative defaults matching the appendix priors are used. |
| ... | Additional arguments passed to the cmdstanr $sample() method. |

### Details

The implementation uses a Gaussian process prior on an unconstrained latent field, applies a softplus transform to obtain a nonnegative derivative, integrates that derivative on a fixed grid, and maps the integrated latent curve through the paper's delta or epsilon link function.

This function requires the optional packages cmdstanr and posterior, and a working CmdStan installation. These packages are not hard dependencies of scR so that the core package remains light-weight.

### Value

An object of class scR_gp_curve containing the fitted Stan object, posterior summaries on the prediction grid, and the Stan data list.

---

## gendata *Simulate data with appropriate structure to be used in estimating sample complexity bounds*

---

### Description

Simulate data with appropriate structure to be used in estimating sample complexity bounds

## Usage

```
gendata(
  model,
  dim,
  maxn,
  predictfn = NULL,
  varnames = NULL,
  sparse = FALSE,
  density = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| model | A binary classification model supplied by the user. Must take arguments formula and data |
| dim | Gives the horizontal dimension of the data (number of predictor variables) to be generated. |
| maxn | Gives the vertical dimension of the data (number of observations) to be generated. |
| predictfn | An optional user-defined function giving a custom predict method. If also using a user-defined model, the model should output an object of class "svrclass" to avoid errors. |
| varnames | An optional character vector giving the names of variables to be used for the generated data |
| sparse | Logical indicating whether sparse matrix generation should be used to save on memory. Defaults to false for better accuracy. |
| density | Real number between 0 and 1 giving the proportion of non 0 entries in the sparse matrix. Used only if sparse is TRUE. |
| ... | Additional arguments that need to be passed to model |

## Value

A data.frame containing the simulated data.

## See Also

estimate_accuracy(), to estimate sample complexity bounds given the generated data

## Examples

```
mylogit <- function(formula, data) {
  structure(
    suppressWarnings(glm(formula = formula, data = data, family = binomial())),
    class = c("svrclass", "glm")
  )
}
```

```
mypred <- function(m, newdata) {
  out <- predict.glm(m, newdata, type = "response")
  factor(ifelse(out > 0.5, 1, 0), levels = c("0", "1"))
}
set.seed(1)
dat <- gendata(mylogit, dim = 2, maxn = 20, predictfn = mypred)
head(dat)
```

---

`getpac` *Recalculate achieved sample complexity bounds given different parameter inputs*

---

## Description

Recalculate achieved sample complexity bounds given different parameter inputs

## Usage

```
getpac(table, epsilon = 0.05, delta = 0.05)
```

## Arguments

| | |
|---|---|
| `table` | A list containing an element named `Raw`. Should always be used with the output of `estimate_accuracy()` |
| `epsilon` | A real number between 0 and 1 giving the targeted maximum out-of-sample (OOS) error rate |
| `delta` | A real number between 0 and 1 giving the targeted maximum probability of observing an OOS error rate higher than `epsilon` |

## Value

A `list` containing two named elements. `Raw` gives the exact output of the simulations, while `Summary` gives a table of accuracy metrics, including the achieved levels of $\epsilon$ and $\delta$ given the specified values. Alternative values can be calculated using `getpac()` again.

## See Also

`plot.scb_data()`, to represent simulations visually, `getpac()`, to calculate summaries for alternate values of $\epsilon$ and $\delta$ without conducting a new simulation, and `gendata()`, to generated synthetic datasets.

## Examples

```
# Recalculate a stored scb_data object with alternate epsilon and delta values.
```

`interpolate_scb` *Conduct interpolation on a list of data*

## Description

Wrapper function for fitting extrapolation model to a list of objects of class "scb_data" using non-linear least squares.

## Usage

```
interpolate_scb(
  data_list,
  delta_interp_fun = c("logis", "logis5", "logis4", "declin"),
  epsilon_interp_fun = c("gompertz", "exp_plateau", "weibull", "quad_plateau"),
  epsilon,
  delta,
  maxN,
  delta_lower_bounds = NULL,
  epsilon_lower_bounds = NULL,
  delta_upper_bounds = NULL,
  epsilon_upper_bounds = NULL,
  delta_start = NULL,
  epsilon_start = NULL
)
```

## Arguments

| | |
|---|---|
| `data_list` | A list of objects of class "scb_data" for interpolation to be conducted on. |
| `delta_interp_fun` | The interpolation/extrapolation function to be used for the delta curve. Defaults to standard logistic, but 4 and 5 parameter as well as declining logistic functions are also supported. |
| `epsilon_interp_fun` | The interpolation/extrapolation function to be used for the epsilon curve. Defaults to gompertz, but exponential-plateau, weibull, and quadratic plateau functions are also supported. |
| `epsilon` | A real number between 0 and 1 giving the targeted maximum out-of-sample (OOS) error rate |
| `delta` | A real number between 0 and 1 giving the targeted maximum probability of observing an OOS error rate higher than `epsilon` |
| `maxN` | A positive integer giving value of the largest N for which extrapolation is to be conducted. |
| `delta_lower_bounds` | Optional named vector of lower bounds for the delta model parameters. |
| `epsilon_lower_bounds` | Optional named vector of lower bounds for the epsilon model parameters. |

`delta_upper_bounds`
: Optional named vector of upper bounds for the delta model parameters.

`epsilon_upper_bounds`
: Optional named vector of upper bounds for the epsilon model parameters.

`delta_start`
: Optional named vector of starting values for the delta model parameters.

`epsilon_start`
: Optional named vector of starting values for the epsilon model parameters.

## Value

A named `list` containing the interpolated dataframe, the original input data frame, and the given values of epsilon, delta, and maxN.

## See Also

`conduct_interpolation()` can be used to fit custom curves or single simulations without confidence intervals.

---

# interpolate_scb_gp *Interpolate sample-complexity curves using monotone Gaussian processes*

---

## Description

Convenience wrapper around `fit_gp_scb_curve()` for objects produced by `estimate_accuracy()`. The returned object has a structure similar to the parametric output from `interpolate_scb()`, but represents posterior summaries from a single monotone Gaussian process fit rather than bootstrap envelopes over many parametric fits.

## Usage

```
interpolate_scb_gp(
  scbobject,
  epsilon = 0.05,
  delta = 0.05,
  maxN,
  curve = c("both", "delta", "epsilon"),
  ...
)
```

## Arguments

`scbobject`
: An object of class `scb_data`, or a data frame with columns `n`, `Delta`, and/or `Epsilon`.

`epsilon`
: Target maximum generalization error.

`delta`
: Target maximum probability of exceeding `epsilon`.

`maxN`
: Largest sample size on the prediction grid.

`curve`
: Which curve to fit: `"delta"`, `"epsilon"`, or `"both"`.

`...`
: Additional arguments passed to `fit_gp_scb_curve()`.

## Value

An object of class `empirical_scb_gp`.

---

`loss` *Utility function to define the least-squares loss function to be optimized for* `simvcd()`

---

## Description

Utility function to define the least-squares loss function to be optimized for `simvcd()`

## Usage

```
loss(h, ngrid, xi, a = 0.16, a1 = 1.2, a11 = 0.14927)
```

## Arguments

| | |
|---|---|
| `h` | A positive real number giving the current guess at VC dimension |
| `ngrid` | Vector of sample sizes for which the bounding function is estimated. |
| `xi` | Vector of estimated values of the bounding function, usually obtained from `risk_bounds()` |
| `a` | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |
| `a1` | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |
| `a11` | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |

## Value

A real number giving the estimated value of the MSE given the current guess.

## See Also

`simvcd()`, the user-facing function for simulating VC dimension and `risk_bounds()` to generate estimates for xi.

## plot.empirical_scb_gp *Plot a monotone Gaussian process sample-complexity fit*

### Description

Plot a monotone Gaussian process sample-complexity fit

### Usage

```
## S3 method for class 'empirical_scb_gp'
plot(
  x,
  plot_type = c("Delta", "Epsilon"),
  include_legend = TRUE,
  include_title = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| x | An object returned by interpolate_scb_gp(). |
| plot_type | Which curve to plot. |
| include_legend | Logical; whether to include a legend. |
| include_title | Logical; whether to include a title. |
| ... | Ignored. |

### Value

A ggplot2 plot.

## plot.empirical_scb_list *Plot method for an* empirical_scb_list *object*

### Description

Visualizes bootstrap-estimated empirical sample complexity bounds (SCB) for either delta or epsilon.

## Usage

```
## S3 method for class 'empirical_scb_list'
plot(
  x,
  truedata,
  alpha = 0.05,
  plot_type = c("Delta", "Epsilon"),
  include_legend = TRUE,
  include_title = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | An object of class "empirical_scb_list" containing extrapolated SCB values. |
| truedata | A bootstrapped list of benchmark simulations, each of class "scb_data". |
| alpha | Numeric between 0 and 1. Significance level used to compute bootstrap confidence intervals (default: 0.05). |
| plot_type | Character string. Determines which SCB to plot: "Delta" (default) or "Epsilon". |
| include_legend | Logical. Whether to display a legend (default: TRUE). |
| include_title | Logical. Whether to include a title (default: TRUE). |
| ... | Additional arguments passed to methods. |

## Value

A ggplot object displaying either the SCB-Delta or SCB-Epsilon curve with bootstrap confidence bands.

## See Also

interpolate_scb() in order to prepare input data..

---

# plot.scb_data *Plot method for simulated sample complexity bounds (*scb_data *object)*

---

## Description

This method plots performance metrics estimated using estimate_accuracy() for objects of class "scb_data".

### Usage

```
## S3 method for class 'scb_data'
plot(
  x,
 metrics = c("Accuracy", "Precision", "Recall", "Fscore", "Delta", "Epsilon", "Power"),
  plottype = c("ggplot", "plotly"),
  letters = c("greek", "latin"),
  ...
)
```

### Arguments

| | |
|---|---|
| x | An object of class "scb_data", typically the output of estimate_accuracy() |
| metrics | A character vector containing the metrics to display in the plot. Can include any of "Accuracy", "Precision", "Recall", "Fscore", "Delta", "Epsilon", "Power". |
| plottype | A string indicating the graphics system to use. Must be either "ggplot" or "plotly". |
| letters | A string specifying whether "Delta" and "Epsilon" should appear as Greek letters ("greek") or Latin ("latin") in the legend. Defaults to "greek". |
| ... | Additional arguments passed to methods. |

### Value

A ggplot or plot_ly object, depending on the value of plottype.

### See Also

estimate_accuracy(), which generates the data used for plotting.

### Examples

```
# Plot objects returned by estimate_accuracy().
```

---

## risk_bounds *Utility function to generate data points for estimation of the VC Dimension of a user-specified binary classification algorithm given a specified sample size.*

---

### Description

Utility function to generate data points for estimation of the VC Dimension of a user-specified binary classification algorithm given a specified sample size.

### Usage

```
risk_bounds(
  x,
  l,
  m,
  model,
  predictfn = NULL,
  sparse = FALSE,
  density = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| x | An integer giving the desired sample size for which the target function is to be approximated. |
| l | A positive integer giving dimension (number of input features) of the model. |
| m | A positive integer giving the number of simulations to be performed at each design point (sample size value). Higher values give more accurate results but increase computation time. |
| model | A binary classification model supplied by the user. Must take arguments `formula` and `data` |
| predictfn | An optional user-defined function giving a custom predict method. If also using a user-defined model, the `model` should output an object of class `"svrclass"` to avoid errors. |
| sparse | Logical indicating whether sparse matrix generation should be used to save on memory. Defaults to false for better accuracy. |
| density | Real number between 0 and 1 giving the proportion of non 0 entries in the sparse matrix. Used only if sparse is TRUE. |
| ... | Additional model parameters to be specified by the user. |

### Value

A real number giving the estimated value of Xi(n), the bounding function

---

## scb *Calculate sample complexity bounds for a classifier given target accuracy*

---

### Description

Calculate sample complexity bounds for a classifier given target accuracy

### Usage

```
scb(vcd = NULL, epsilon = NULL, delta = NULL, eta = NULL, theor = TRUE, ...)
```

### Arguments

| | |
|---|---|
| vcd | The Vapnik-Chervonenkis dimension (VCD) of the chosen classifier. If theor is FALSE, this can be left unspecified and simvcd() will be called to estimate the VCD |
| epsilon | A real number between 0 and 1 giving the targeted maximum out-of-sample (OOS) error rate |
| delta | A real number between 0 and 1 giving the targeted maximum probability of observing an OOS error rate higher than epsilon |
| eta | A real number between 0 and 1 giving the probability of misclassification error in the training data. |
| theor | A Boolean indicating whether the theoretical VCD is to be used. If FALSE, it will instead be estimated using simvcd() |
| ... | Arguments to be passed to simvcd() |

### Value

A real number giving the sample complexity bound for the specified parameters.

### See Also

simvcd(), to calculate VCD for a chosen model

### Examples

```
scb(vcd = 7, epsilon = 0.05, delta = 0.05, eta = 0.05)
```

---

## simvcd *Estimate the Vapnik-Chervonenkis (VC) dimension of an arbitrary binary classification algorithm.*

---

### Description

Estimate the Vapnik-Chervonenkis (VC) dimension of an arbitrary binary classification algorithm.

### Usage

```
simvcd(
  model,
  dim,
  m = 1000,
  k = 1000,
  maxn = 5000,
  parallel = TRUE,
  coreoffset = 0,
  predictfn = NULL,
  a = 0.16,
```

```
  a1 = 1.2,
  a11 = 0.14927,
  minn = (dim + 1),
  sparse = FALSE,
  density = NULL,
  backend = c("multisession", "multicore", "cluster", "sequential"),
  packages = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| model | A binary classification model supplied by the user. Must take arguments formula and data |
| dim | A positive integer giving dimension (number of input features) of the model. |
| m | A positive integer giving the number of simulations to be performed at each design point (sample size value). Higher values give more accurate results but increase computation time. |
| k | A positive integer giving the number of design points (sample size values) for which the bounding function is to be estimated. Higher values give more accurate results but increase computation time. |
| maxn | Gives the vertical dimension of the data (number of observations) to be generated. |
| parallel | Boolean indicating whether or not to use parallel processing. |
| coreoffset | If parallel is true, a positive integer indicating the number of free threads to be kept unused. Should not be larger than the number of CPU cores. |
| predictfn | An optional user-defined function giving a custom predict method. If also using a user-defined model, the model should output an object of class "svrclass" to avoid errors. |
| a | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |
| a1 | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |
| a11 | Scaling coefficient for the bounding function. Defaults to the value given by Vapnik, Levin and Le Cun 1994. |
| minn | Optional argument to set a different minimum n than the dimension of the algorithm. Useful with e.g. regularized regression models such as elastic net. |
| sparse | Logical indicating whether sparse matrix generation should be used to save on memory. Defaults to false for better accuracy. |
| density | Real number between 0 and 1 giving the proportion of non 0 entries in the sparse matrix. Used only if sparse is TRUE. |
| backend | One of the parallel backends used by future::plan(). See function documentation for more details. |
| packages | A list of strings giving the names of packages to be loaded in order to estimate the model. |
| ... | Additional arguments that need to be passed to model |

## Value

A real number giving the estimated value of the VC dimension of the supplied model.

## See Also

scb(), to calculate sample complexity bounds given estimated VCD.

## Examples

```
# Use small values of m, k, and maxn for quick smoke tests.
# Use larger values in applied work.
```

---

`summary.empirical_scb_gp` *Summarize a monotone Gaussian process sample-complexity fit*

---

## Description

Summarize a monotone Gaussian process sample-complexity fit

## Usage

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

## Arguments

| | |
|---|---|
| `object` | An object returned by `interpolate_scb_gp()`. |
| `...` | Ignored. |

## Value

A list summarizing the first grid point where each fitted curve meets its target.

summary.empirical_scb_list *Summary of empirical sample complexity bound results*

## Description

For an `empirical_scb_list` object, finds

1. the mean-fit SCB crossing $N$ (where the average bootstrap curve first drops below the target),
2. a lower bound on that crossing (the smallest $N$ at which the *lower* CI envelope crosses), and
3. an upper bound on the crossing (the smallest $N$ at which the *upper* CI envelope crosses).

## Usage

```
## S3 method for class 'empirical_scb_list'
summary(object, alpha = 0.05, ...)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `"empirical_scb_list"` containing bootstrap replicates. |
| `alpha` | Numeric in $(0, 1)$. Two-sided CI level for the envelope (default: 0.05 for 95%). |
| `...` | Additional args (ignored). |

## Value

Invisibly, a list of components

`delta` List with `SCB_N` (mean-curve crossing), `status` (`"Observed"`/`"Extrapolated"`, or `NA` if not reached), `CI_lower_N`, `CI_upper_N` (bounds on the crossing).

`epsilon` Same four elements for the $\epsilon$ target.

`alpha` The CI level.

`initial` The initial subsample size.

## See Also

`plot.empirical_scb_list`, `getpac`

# Index