Documentation of CapacityLogReg

T.Jetka

May 31, 2018

Contents

Preliminaries . . . 1
Structure of the package . . . 3
Basic workflow of the analysis . . . 4
Diagnostic procedures . . . 8
Other Functionalities . . . 10
Examples . . . 17
List of all functions in the package . . . 28

The package CapacityLogReg is designed to estimate channel capacity between a discrete input and multidimensional output from experimental data. For efficient computations, it uses iterative algorithm based on logistic regression. The core function `capacity_logreg_main()` is the interface to all functionalities provided in the package.

CapacityLogReg is released under the GNU licence and is freely available from GitHub. A comprehensive documentation is available also on github.io. In case of any bugs, problems and questions regarding the package or inquiries about information theory, contact t.jetka at gmail.com. Methodological details are presented in

> Jetka et al., Estimator of channel capacity using statistical learning, 2018 (to be published)

Preliminaries

Requirements - Hardware

- A 32 or 64 bit processor (recommended: 64bit)
- 1GHz processor (recommended: multicore for a comprehensive analysis)
- 2GB MB RAM (recommended: 4GB+, depends on the size of experimental data)

Requirements - Software

The main requirement is the installation of the R environment (version: >= 3.2), which can be obtained from R project webstie and is distributed for all popular operating systems. We tested our software on Windows 7, 10; Mac OS X 10.11 - 10.13 and Ubuntuu 18.04 with no significant differences in the performance. The use of a dedicated IDE, e.g. RStudio is recommended.

Apart from a base installation of R, CapacityLogReg requires following R packages:

1. for estimation

- e1071
- nnet

- glmnet
- caret
- doParallel (if parallel computation are needed)

2. for visualisation

- ggplot2
- ggthemes
- gridExtra

3. for data handling

- reshape2
- stringr
- plyr

Each of the above packages can be installed by running

```
install_packages("name_of_a_package")
```

in the R console.

Installation

Our package can be obtained from GitHub. For a direct installation, a `devtools` package is needed. Therefore, in order to install CapacityLogReg, open RStudio (or base R) and run following commands in the R console

```
install_packages("devtools") # run if not installed
library(devtools)
install_github("TJetka/CapacityLogReg")
```

If not found, all dependencies will be installed as well.

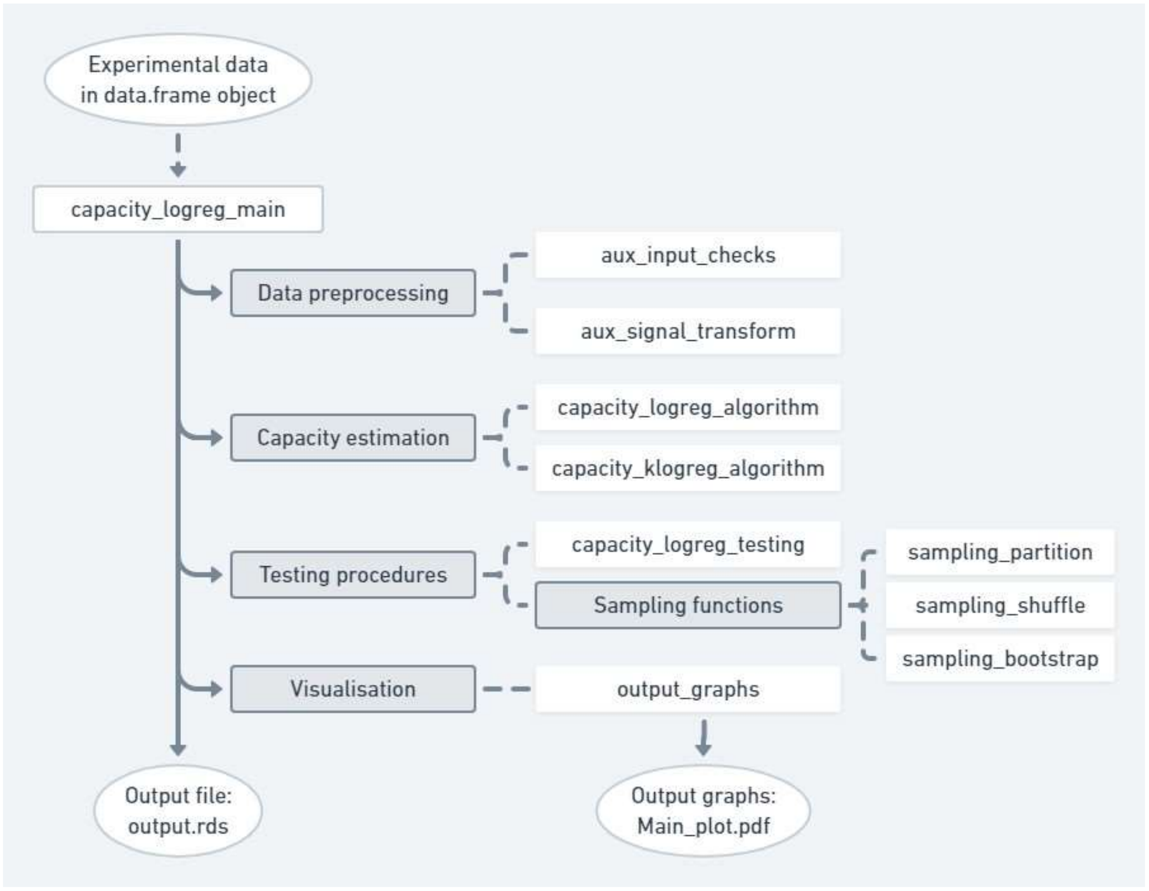
Docker

For portability and reproducibility of the research, we also provide an isolated Docker image of our package based on `tidyverse` image from Rocker project, which includes basic debian functionalities, base R, RStudio and our package with all dependencies. It can be obtained from the link: XXX.

Please see Instruction for details on the usage of R Dockers.

Structure of the package

Estimation of channel capacity can be performed by calling function `capacity_logreg_main()`. It triggers 1) preprocessing of the data; 2) estimation of channel capacity; 3) running diagnostic procedures; 4) visualisation. Dependencies between functions implemented in the package are presented in the graph below.



Our package expects experimental data to be loaded as a `data.frame` object with observations in rows, value of the input in one of the columns and values of the corresponding output in remaining columns. The main algorithm is implemented in a function `capacity_logreg_algorithm()`, which uses logistic regression from `nnet package`. For more advanced uses, we also included implementation of multinomial regression model from `glmnet` package in a function `capacity_klogreg_algorithm()`. Diagnostic procedures (significance and uncertainties of estimates) are given in `capacity_logreg_testing()` function, which includes a) bootstrapping data and b) overfitting test. For visualisation, a set of graphs is created by a function `capacity_output_graphs()` and saved in a specified directory. In addition, `capacity_logreg_main()` returns a list with capacity estimates, optimal input probability distribution, diagnostic measures and other summary information about the analysis.

Full specifications of functions distributed with the package are listed in tables at the end of the documentation.

Basic workflow of the analysis

The CapacityLogReg package is primarily designed to aid in applying information theory to experimental data in systems biology. Specifically, we assume that a signaling pathway is modeled as an information channel that transforms a specific level of input (signal), X, into output (response) of the pathway, Y, according to a probability distribution $P(Y|X)$. In a typical example, X represents the concentration of the ligand that activates signaling pathway, while Y is a nuclear concentration of transcription factor. In order to quantify information transmission in such case, we need to

i) establish a finite set of stimulation concentrations $x_1, x_2, \ldots, x_m$ from 0 to saturation level;
ii) measure experimentally the concentration of Y for a fixed level of $X = x_i$ to obtain a sample of size n_i that is a representation of the distribution $P(Y|X = x_i)$.

As the output of the system can be measured in different time points, or multiple transcription factor are involved, Y is often multidimensional. See Section 1 of Supplementary Methods of corresponding publication for details.

Preparing data

Conceptually, a full experimental dataset in a described setting can be represented as a table

	input	output 1	output 2	output 3	...
	x_1	$y^1_{1,1}$	$y^1_{1,2}$	$y^1_{1,3}$	
n_1	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	x_1	$y^1_{n_1,1}$	$y^1_{n_1,2}$	$y^1_{n_1,3}$	
	x_2	$y^2_{1,1}$	$y^2_{1,2}$	$y^2_{1,3}$	
n_2	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	x_2	$y^2_{n_2,1}$	$y^2_{n_2,2}$	$y^2_{n_2,3}$	
					...
$\vdots$		$\vdots$	$\vdots$	$\vdots$	
	x_m	$y^m_{1,1}$	$y^m_{1,2}$	$y^m_{1,3}$	
n_m	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	x_m	$y^m_{n_m,1}$	$y^m_{n_m,2}$	$y^m_{n_m,3}$	

In consequence, to use our package, it is required to organise experimental data in a similar way and save it as a `data.frame` object. Specifically, we expect that for this data frame:

- each row represent a single observation
- one column contains values of the input (X). We recommend to set the type of this column to class `factor` and specify the correct order of the possible levels of the input.
- one or more columns contain values of the measured output(s). Those columns should be of type `numeric`.

According to the validation tests carried out in the paper accompying this package, the number of unique values of the input should be strictly finite (at most dozens) and there should be at least a few hundreds of observations per a single value of the input to correctly estimate the channel capacity.

See for example our experimental data containing the measurements of NfkB transcription factor after the stimulation by TNF-α. It accompanies the package under the variable `data_nfkb` and has the following format

	signal	response_0	response_3	response_6
1	0ng	0.3840744	0.4252835	0.4271986
2	0ng	0.4709216	0.5777821	0.5361948
3	0ng	0.4274474	0.6696011	0.8544916
10001	8ng	0.3120216	0.3475484	1.0925967
10002	8ng	0.2544961	0.6611051	2.2894928
10003	8ng	0.1807391	0.4336810	1.9783171
11540	100ng	1.3534083	3.0158004	5.1592848
11541	100ng	1.7007936	2.2224497	3.5463418
11542	100ng	0.1997087	0.2886905	1.9324093

where each row represents measurements of a single-cell, the column named `signal` specifies the level of stimulation, while response_T is the concentration of the NfkB in the cell's nucleus at the time point T. The above table can be show in R by calling

```
library(CapacityLogReg)
rbind(data_nkfb[1:3,1:4],data_nkfb[10001:10003,1:4],tail(data_nkfb[,1:4],3))
```

Running

The package is based on a main wrapper function - `capacity_logreg_main()`, which triggers subsequent steps of the analysis implemented within the package. To calculate channel capacity between X and Y user must call

```
capacity_logreg_main(dataRaw,signal, response, output_path)
```

where the required arguments are

- `dataRaw` - data frame with column of type `factor` containing values of input (X) and columns of type `numeric` containing values of output (Y), where each row represents a single observation
- `signal` - a character which indicates the name of the column in `dataRaw` with values of input (X)
- `response` - a character vector which indicates the names of columns in `dataRaw` with values of output (Y)
- `output_path` - a character with the directory, to which output should be saved

Firstly, the above function conducts initial preprocessing of data (scaling and centering) that ensures the stability of numerical computations (via `preProcess` from `caret` package). Next, an iterative algorithm to estimate channel capacity is triggered, where the main step involves fitting a logistic regression model. Based on those computations graphs visualising data and estimation are created in the subsequent step. Finally, additional diagnostic measures are collected. The function `capacity_logreg_main()` returns a list with elements

- cc - a numeric with channel capacity estimate (in bits)
- p_opt - a numeric vector with the optimal input distribution
- model - a `nnet` object describing fitted logistic regression model
- data - a data.frame with the raw experimental data (if `dataout=TRUE`)
- time - processing time of the algorithm
- params - a vector of parameters used in the algorithm
- regression - a confusion matrix of logistic regression predictions
- logGraphs - a list of gg or ggtables objects with a standard set of exploratory graphs

Defaultly, all returned elements are also saved in `output_path` directory in a file `output.rds` together with additional exploratory graphs which include:

- MainPlot.pdf - a simple summary plot with basic distribution visualisation and capacity estimate

- MainPlot_full.pdf - a comprehensive summary plot with distribution visualisation and capacity estimate
- capacity.pdf - a diagram presenting capacity estimate
- io_relation.pdf - a graph with input-output relation
- kdensities.pdf - kernel density estimator of data distribution
- histograms.pdf - histograms of data
- boxplots.pdf - boxplots of data
- violin.pdf - violin plots of data

Minimal working example

Below we present a minimal working example. Firstly, we create a simple synthetic dataset. We assume a channel with four different levels of stimulations: 0, 0.1, 1 and 10 and Gaussian output. It can be generated in data frame `tempdata` with columns `input` and `output` by running

```
tempdata = data.frame(input = factor(c(t(replicate(1000,c(0,0.1,1,10)))),
                      levels=c(0,0.1,1,10)),
                      output =  c(matrix(
                      rnorm(4000,mean=5+c(0,0.1,1,10),sd=c(1,1,1,1)),
                      ncol=4,byrow=TRUE)))
```

and has following structure

	input	output
1	0	5.947838
2	0	4.176852
2001	1	6.875569
2002	1	6.254148
3999	10	14.681075
4000	10	13.777880

Then, we call `capacity_logreg_main()` function to find an estimate of channel capacity. As `dataRaw` argument we pass our data frame `tempdata`, as names of columns we must specify `input` and `output` and the output directory we set to `minimal_example/`. It is done be using

```
tempoutput  <- capacity_logreg_main(dataRaw=tempdata,
                                    signal="input", response="output",
                                    output_path="minimal_example/")
```

Most conveniently, results can be presented in the following graph (defaultly saved as MainPlot.pdf in `minimal_example/` directory). It shows how the distribution of output changes with different inputs and gives the corresponding channel capacity

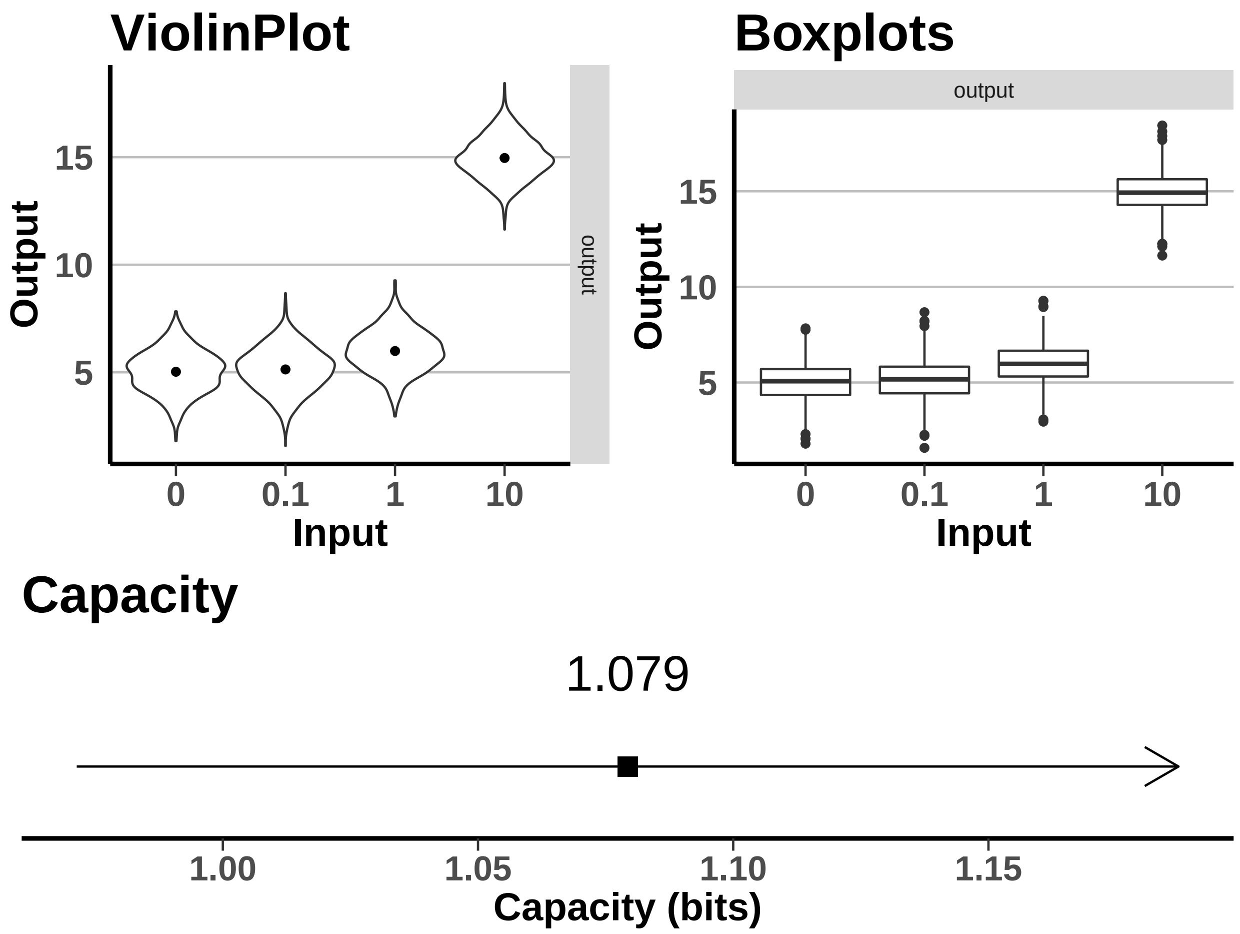


Figure 1: Standard output graph of minimal working example

Diagnostic procedures

We implemented two diagnostic procedures to control the performance of channel capacity estimation and to measure uncertainity due to finite sample size and model over-fitting. These include:

1. Bootstrap test - capacity is re-calculated using x% of data, sampled from original dataset without replacement. After repeating procedure n times, its standard deviation can be treated as an error of original estimate.
2. Over-fitting test - original data is divided into Training and Testing datasets. Then, logistic regression is estimated using x% of data (training dataset) and integrals of channel capacity are calculated via Monte Carlo using remaining $(1-x)$% of data (testing dataset). It is repeated n times.

In order to use those procedures, user must provide additional arguments to function `logreg_capacity_main()`, i.e.

- testing (default=FALSE) - a logical value that turn on/off testing mode,
- TestingSeed (default= 1234) - the seed for the random number generator used to sample original dataset,
- testing_cores (default= 4) - a number of cores to use (via `doParallel` package) in parallel computing,
- boot_num (default= 40) - a number of repetitions of the bootstrap ,
- boot_prob (default= 0.8) - a fraction of initial observations to use in the bootstrap,
- traintest_num (default= 40) - a number of repetitions of the overfitting test,
- partition_trainfrac (default= 0.6) - a fraction of initial observations to use as a training dataset in the overfitting test

For example, user can, by utilising a synthetic dataset accompying the package, run

```
dir.create("example1_testing/")
outputCLR=capacity_logreg_main(dataRaw=data_example1,
                              signal="signal",response="response",
                              output_path="example1_testing/",
                              testing=TRUE, TestingSeed = 1234, testing_cores = 4,
                              boot_num = 40, boot_prob = 0.8,
                              traintest_num = 40,partition_trainfrac = 0.6)
```

to run diagnostics with 40 re-sampling of data, where bootstrap is caluclated using 80% of data, while over-fitting test include testing dataset of size 60% of original data.

It provides following result

The top diagram shows the value of original capacity estimate (in black) and the mean value of bootstrap repetitions with indicated +/- sd (in red). Plots that follow, show histograms of calculated capacities for different testing regimes. Black dot represents the basic channel capacity estimate. In addition, empirical p-values (left- and right-sided) are calculated to assess the randomness of obtained results.

To ensure the correctness of the analysis

1. Bootstrap should yield distribution of capacity with small variance and basic capacity should not be an outlier (p-value>0.05). Otherwise, it would indicate that the sample size is too low for an accurate estimation of channel capacity.
2. Over-fitting test should also provide a conclusion that the basic capacity estimate is non-significantly different than the distribution of capacities from the test. In the opposite case, it could mean that the logistic regression model does not fully grasps the important aspects of the dependence in the data.

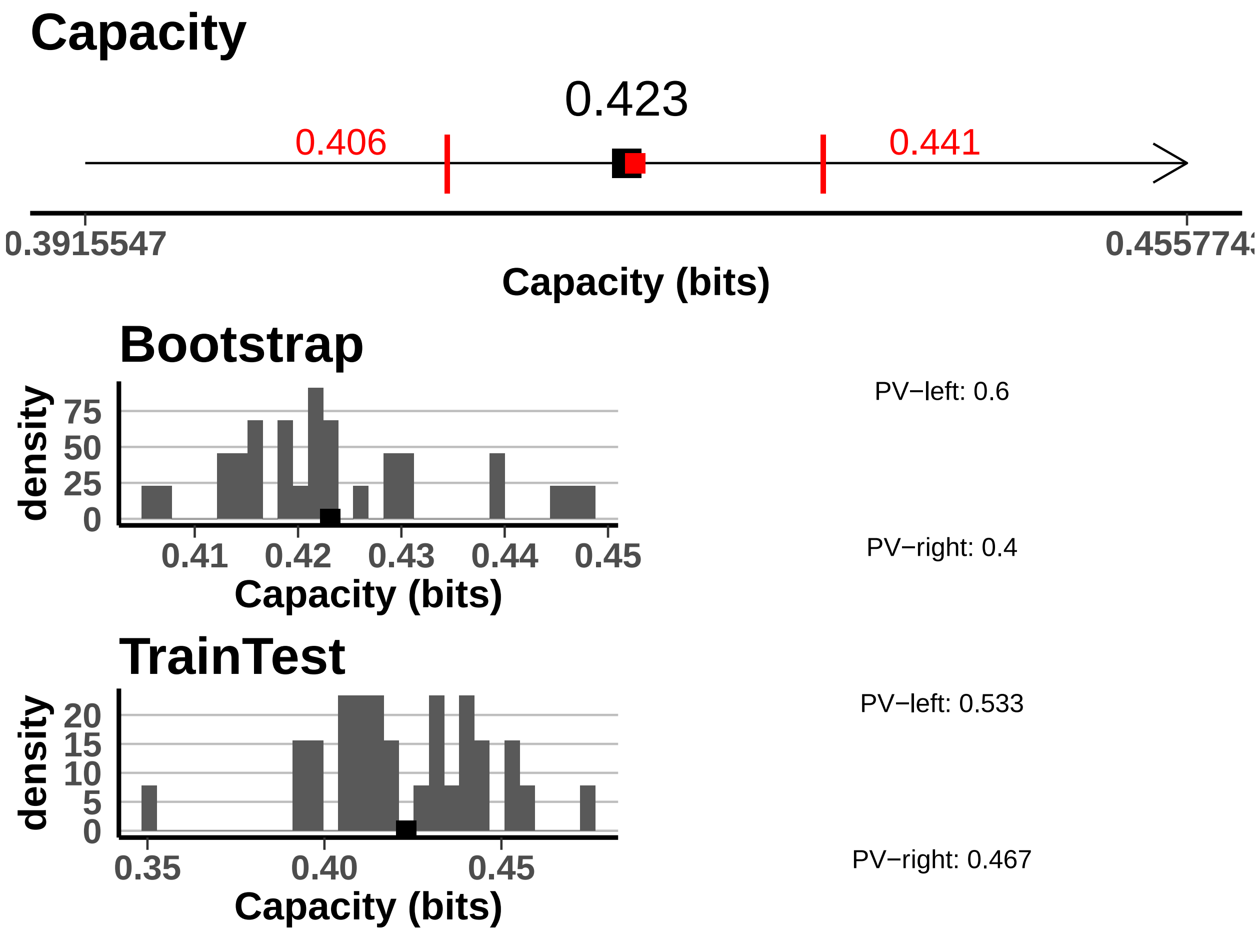


Figure 2: Standard output graph of testing procedures. PV are based on empirical test either left- or right-sided.

Other Functionalities

Additional parameters

Apart from required arguments, the function `capacity_logreg_main` has also other parameters than can be used to tune the activity of the algorithm. These are

- `model_out` (`default=TRUE`) - logical, specify if `nnet` model object should be saved into output file
- `graphs` (`default=TRUE`) - logical, controls creating diagnostic plots in the output directory.
- `plot_width` (`default = 6`) - numeric, the basic width of created plots
- `plot_height` (`default = 4`) - numeric, the basic height of created plots
- `scale` (`default = TRUE`) - logical, value indicating if the columns of `dataRaw` are to be centered and scaled, what is usually recommended for the purpose of stability of numerical computations. From a purely theoretical perspective, such transformation does not influence the value of channel capacity.

The basic mode of our algorithm is based on logistic regression implementation of `nnet` package. We allow to pass to our wrapper function `capacity_logreg_main()` some parameters of original `nnet::nnet` and `nnet::multinom` functions. These are:

- `lr_maxit` (`default = 1000`) - a maximum number of iterations of optimisation step in logistic regression algorithm. Set to higher value if your data is more complex or of high dimension.
- `MaxNWts` (`default = 5000`) - a maximum number of paramters in logistic regression model. Set to higher value if you data has many dimensions or input has many states.

Extended Logistic Regression

Our method of channel capacity estimation is based on classification performance of a conventional logistic regression model. As in the case of any model, the assumed functional form not always describes patterns in the data to full extent. Still, adjustment are usually easy to implement to achieve a better classificator and a better estimate of the channel capacity as a result.

Extending formula directly

The most straightforward method to correct the algorithm performance is tuning the formula of model variables. It can be achieved by directly specifying argument `formula_string` of `capacity_logreg_main()` function.

Let's consider a simple example of channel $X \to Y$, where X is a two-state variable, $X = 1$ or $X = 2$ and conditional distributions of $Y|X$ are Gaussians:

$$Y|X_1 \sim \mathcal{N}(10, 1^2), \quad Y|X_2 \sim \mathcal{N}(10.5, 2^2)$$

In brief, means of $Y|X$ differs slightly, whereas variances differs much. Below density plots of these two distributions are presented. If small sample sizes are involved, a standard approach can fail here. But the correction can be introduced easily without much of additional computational effort.

Indeed, it is enough to extended the formula of logistic regression to

$$X \sim Y + Y^2$$

which can be run by calling

```
capacity_logreg_main(dataRaw=data,signal="X",response = "Y",
                       output_path =path,formula_string="X~Y+I(Y^2),scale=FALSE)")
```

where `formula_string` argument is structured as the convention in R's scripting language.

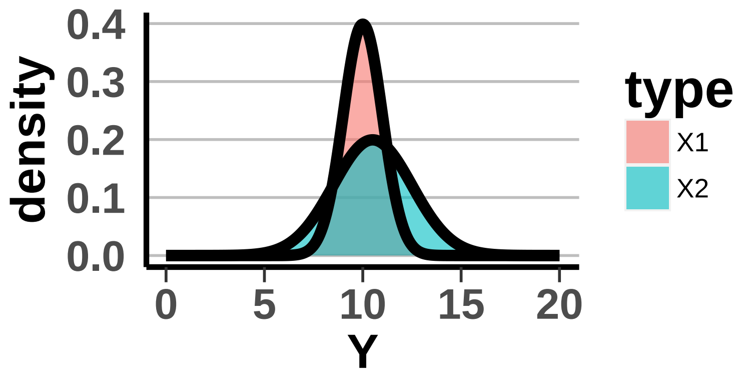


Figure 3: Graphs of density plots of Y|X distributions

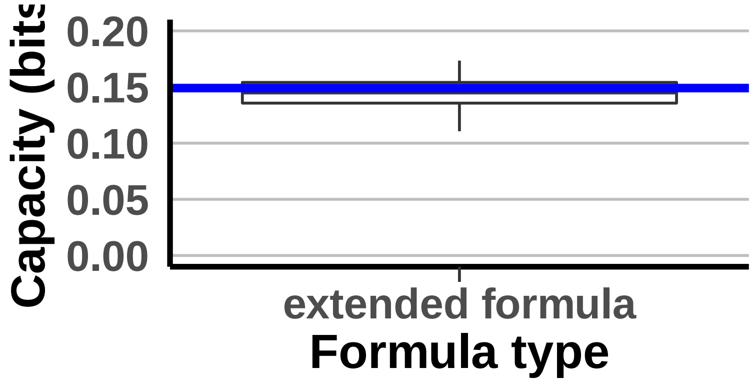


Figure 4: Channel capacity calculated using an extended formula of logistic regression. Boxplots based on 30 repetition of algorithm. Blue line is true channel capacity.

In our example, 500 observations for each of X levels is sampled. Then, whole procedure is repeated 30 times. Analytically computed channel capacity (by brute force optimisation and numerical evaluation of integrals) is calculated for benchmark.

Here is the plot with the capacity estimate compared to a true value, which shows a good aggregatem, confirming the applicability of such functional form in this example

Kernel Logistic Regression

There is also a more comprehensive way to deal with a complicated data structure. It is the kernel approach with a proper feature selection method. To ease the integration of such procedure within our method, we also allow to use `glmnet` implementation of logistic regression from `glmnet` package. It is possible by setting in main wrapper function `capacity_logreg_main()` arguments:

- `glmnet_algorithm` (default=FALSE) - a logical, indicates if `glmnet` package should be used
- `dataMatrix` (default = NULL) - numerical matrix, data to be used as output variables (the same as needed in `glmnet` function)
- `glmnet_cores` (default = 1) - numerical, number of cores to use in cross-validation of glmnet
- `glmnet_lambdanum` (default = 10) - numerical, parameter of glmnet algorithm, should be indetified by cross-validation

If Kernel Logistic Regression is to be used, then we propose to utilise `KernSmooth` or `kernlab` package to generate a proper kernel extension of data and then include it into our algorithm as described above via glmnet dataMatrix object.

Below is an example of combining our package, kernel methods and glmnet algorithm. Channel is defined in the same way as in the previous example.

```
library(CapacityLogReg)
library(ggplot2)
library(KernSmooth)
library(kernlab)

n=500
m1=10
m2=10.5
s1=1
s2=2

df=data.frame(x=c(rep("A",n),rep("B",n)),
                  y=c(rnorm(n,mean=m1,sd=s1),rnorm(n,mean=m2,sd=s2)))

yklr=as.matrix(df$y)
xklr=as.matrix(df$x)
rbf<-rbfdot(sigma=1.4)
kMatrix<-kernelMatrix(rbf,yklr)

path="example_kformula/"
dir.create(path,recursive = TRUE)
capacity=capacity_logreg_main(dataRaw=df, signal="x", response="y",side_variables=NULL,
                                        formula_string=NULL,
                                        glmnet_algorithm=TRUE,dataMatrix=kMatrix,
                                        glmnet_cores=1,glmnet_lambdanum=10,
                                        cc_maxit=100,lr_maxit=1000, MaxNWts = 5000,
                                        output_path=path,
                                        testing=FALSE,graphs=TRUE)
```

Conditional channel capacity

Mutual information is an excellent tool to quantify cause-effect interactions, as long as the observed heterogeneity is solely the result of random fluctuations and as such consitutes the natural environemnt of the phenomenon in question. Nonetheless, the variability of the measured behaviour could also be a consequence of our inability to discover some other important factors. Then, the data is not representative of the underlying probability distributions leading to underestimation of the true statistical depdendence and information capacity. Therefore, a proper identification of subpopulations in the data, can lead to a more grounded capacity estimate.

We incorporate these concepts with our inofrmation theoretic framework. Formally, we hypothesize that some of the variability of the response can be a result of an unobserved covariate that indirectly influence the response. If this is the case, not accounting for this covariate generates artificial heterogeneity and, if information transmission is concerned, underestimate the true capacity of such channel. We thus assume that a channel $X \xrightarrow{P(y|x)} Y$ is additionally dependent on a side variable, S, categorical or continuous that can interact with response, Y. Practically, in the case of systems biology, S represents the phase of cell cycle, the initial abundance of an important kinase, cell's morphology or microenviornment of the population. In result, the channel is described by a relation $Y \sim P(y|x;s)$.

Then, from basic probabilistic law $P(y|x) = \int P(y|x,s)P(s)ds$. In addition, input X is usually applied irrespective of the state, in which cells are, thus we can assume that X and S are independent: $P(x,s) = P(x)P(s)$.

In this setting, the notion of mutual information can be extended by the use of conditional mutual information (CMI): $\mathrm{CMI}(X,Y|S) = H(X|S) - H(X|Y,S)$. Based on that we introduce conditional channel capacity (C^*_{cond})

$$C^*_{cond} = \max_{P(X)} \ \mathrm{CMI}(X,Y|S)$$

In many applications, CMI has proven to be an excellent measure to analyze interactions, couplings of time serie or network topology. From practical point of view, the assumption of independence between X and S, allows to rewrite this formula as

$$C^*_{cond} = \max_{P(X)} \int \sum_{j=1}^{m} P(y,s|x_j)p(x_j) \log \frac{p(x_j|y,s)}{p(x_j)} dyds,$$

and as such can be directly incorporated into our algorithm. The only change would be to estimate classifier with respect not only to output Y, but also all possible interactions between Y and S.

The use of capacity C^*_{cond} primarily gives a more realistic estimate of information that can be processed by a channel by explaining a portion of observed heterogeneity of the output. However, it can be also used to confirm, which features (side variables) manifest meaningful subpopulations by a formal statistical test.

To present a quantitative example of the use of C^*_{cond}, consider a population, in which cells can be in three states, represented by side variable $S \in \{s_1, s_2, s_3\}$, which occur with probabilities 0.6, 0.3 and 0.1, respectively. Cells are stimulated by four concentration of input, $X \in \{x_1, x_2, x_3, x_4\}$. Let's assume that the response of cell follows normal distribution given by a formula

$$Y|X = x_j, S = s_g \quad \sim \quad \mathcal{N}(j \cdot g, 1^2).$$

It can be interpreted that cells in state s_1 are rather insensitive to changes in input, whereas states s_2, s_3 corresponds to a responsive behaviour. The resulting distributions are depicted in below Figure.

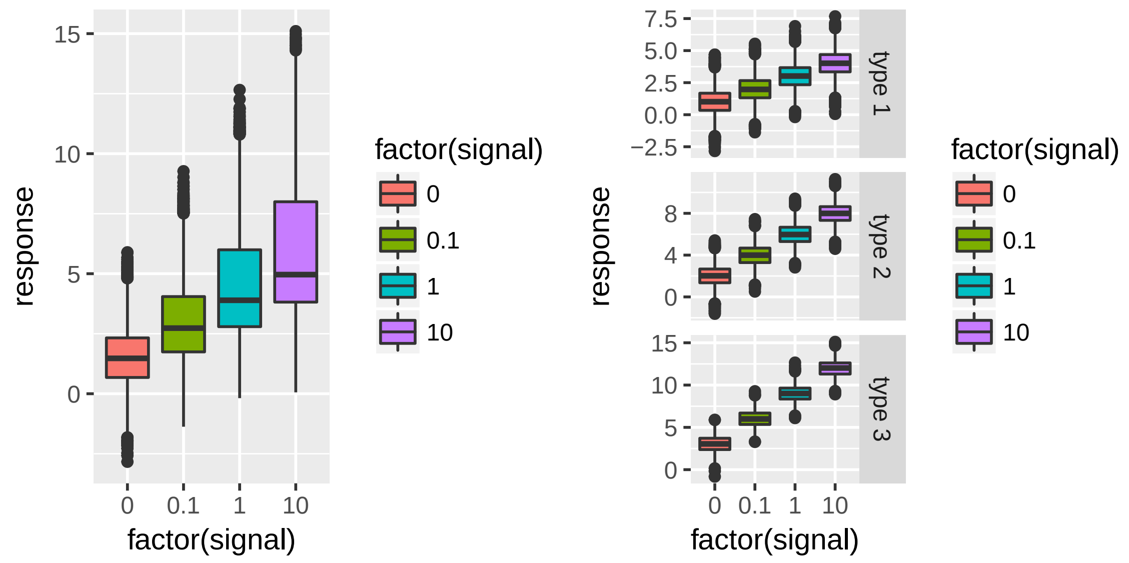


Figure 5: Graphs of density plots of Y|X distributions in the example of conditional channel capacity

We sampled data from the assumed distributions, so that there are 1000 observations for each signal level by running

```
signal=c("0","0.1","1","10")
means=list()
sds=list()
means[["0"]]=   c(1, 2   ,  3 )
means[["0.1"]]= c(2, 4   ,  6 )
means[["1"]]=   c(3, 6   , 9   )
means[["10"]]=  c(4, 8   , 12   )
sds[["0"]]    = c(1    , 1 ,1)
sds[["0.1"]] = c(1 , 1    ,1   )
sds[["1"]]    = c(1 ,1 ,1)
sds[["10"]]   = c(1 , 1 ,1   )
N=c(1000,1000,1000,1000)
names(N)<-signal
sideTypes=c("type 1","type 2","type 3")
probs=c(0.6,0.3,0.1)
data=do.call(rbind,lapply(signal,function(x){
  components<-sample(1:3,prob=probs,size=N[x],replace=TRUE)
  sideVarSample <- sideTypes[components]
  ResponseSample <- rnorm(n=N[x],mean=means[[x]][components],sd=sds[[x]][components])
  data.frame(sideVar=sideVarSample,response=ResponseSample,signal=x)
}))
colnames(data)<-c("sideVar","response","signal")
```

It creates an object with following structure

	sideVar	response	signal
1	type 1	2.142033	0
2	type 2	1.687121	0
2001	type 1	1.714896	0
2002	type 2	4.386438	0

	sideVar	response	signal
39999	type 1	4.372514	10
40000	type 1	4.304621	10

Now, in order to calculate conditional channel capacity within our package you can also use the function `capacity_logreg_main`, but additional arguments are required. Specifically, it is enough to add `side_variable`, where the the use must give the name of the column in the data frame which corresponds to the side variable, e.g.

```
capacity_logreg_main(dataRaw=data,
                     signal="signal", response="response",
                     side_variables = "sideVar")
```

If the knowledge about cellular state is not taken into account, differences of response between subsequent inputs for cellular states s_2 and s_3 are blurred by observed output of cells in state s_1, especially if this is the most prevalent case. Precisely, variances of distributions $Y|X = x_j$ are several times higher. This is visible in low capacity of such system achieving only 0.65 bit. On the other hand, incorporating the variable S into the analysis by calculating C^*_{cond} can correct for this phenomenon and give a significantly higher estimate of capacity of 0.95 bit. Capacities calculated based on observations restricted only to a single subpopulation corraborate the sensitivness of cells in states s_2 and s_3 since they achieve significantly higher capacities .

For inspecting the significance of the `side_variable` influence on the channel capacity, there are two additional diagnostic procedures implemented. They are initiated if `side_variable` is not `NULL` and `testing` is `TRUE`. Those are

3. Side Variable Shuffling - where values side variables' are randomly re-shuffled. In other words, a new dataset is created where side variable is randomly sampled without replacement from original side variable values and capacity is recalculated.
4. Side Variable Shuffling with bootstrap - the above with addition of bootstrapping the data.

The number of repetitio of each of above diagnostic tests are controlled by additional argument:

- sidevar_num (default= 40) - a number of repetitions of side variable shuffling

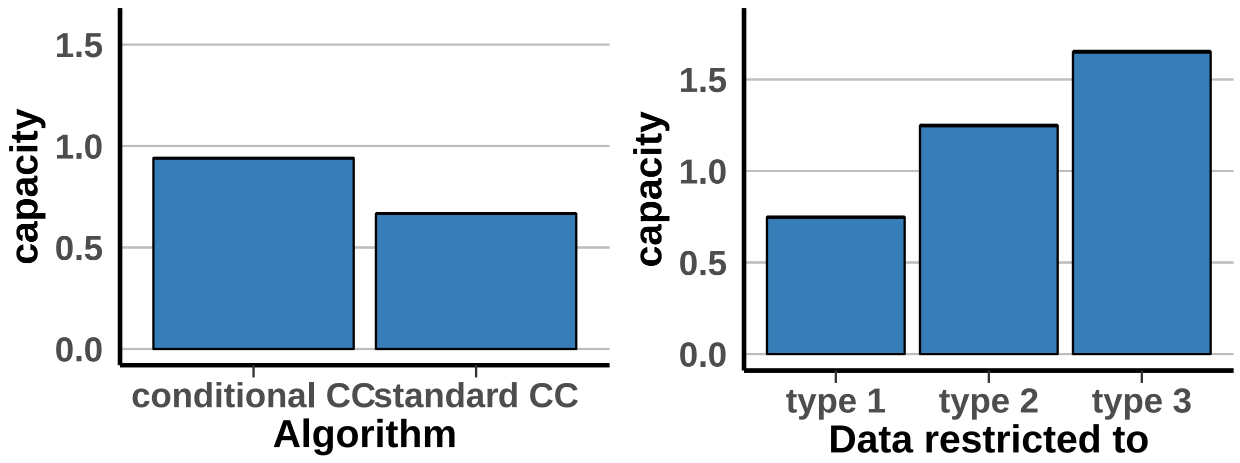


Figure 6: Capacities of example using different setups in comparison to conditional channel capacity.

Examples

Below we present two step-by-step examples how to use CapacityLogReg package to estimate channel capacity. You can access a corresponding R script with those two examples from directory `testing_procedures` at GitHub respository

One-dimensional example

Our method expects as input a `data.frame` with a column indicating the value of input (recommended type: factor) and other columns with corresponding output measurements (type: numeric). It will be created by following steps

1. Set parameters of the example

```
# sample size for each input concentration;
# change to investigate its influence on estimation
n_sample <- c(1000)
# standard deviation of ln(Y)|X=x;
# change to investigate its influence on estimation
dist_sd  <- 1
# number of concentration of input X considered;
# change to investigate its influence on estimation
input_num <- 6
```

2. Create output directory

```
i_type <- "testing_basic"
path_output_main <-
  paste('output/',
    i_type,'/',
    sep="")
dir.create(
  path_output_main,
  recursive = TRUE)
```

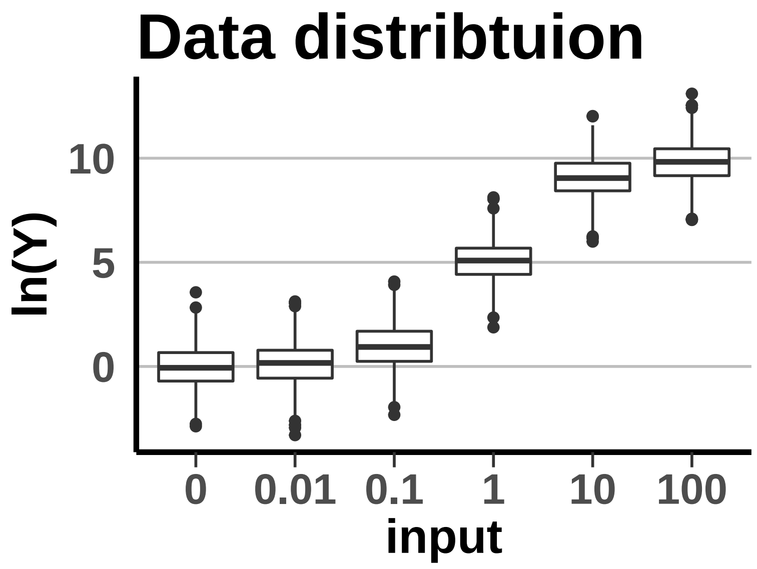
3. Generating synthetic data

```
# concentration of input; spans from 0 to saturation.
xx <-
  signif(
c(0,
  exp(
    seq(
      from = log(0.01),
      to = log(100),
      length.out = input_num-1))),
digits = 2)
# mean of the dose-response relation; Michealis Menten assumed
example_means <- 10*(xx/(1+xx))
example_sds <- rep(dist_sd, input_num)
tempdata <-
  data.frame(
signal = c(t(replicate(n_sample, xx))),
output = c(matrix(
  rnorm(n = input_num*n_sample,
        mean = example_means,
        sd = example_sds),
  ncol = input_num,
  byrow = TRUE)))
tempdata$signal <-
  factor(
x = tempdata$signal,
levels = sort(unique(tempdata$signal)))
print(head(tempdata))
```

```
##   signal     output
## 1      0 -1.2070657
## 2      0 -0.5747400
## 3      0 -0.7762539
## 4      0 -0.8371717
## 5      0 -0.6937202
## 6      0  1.1022975
```

4. Preview the distribution of the data. It will create a graph as below

```
g_plot <- ggplot(
  data = tempdata,
  aes(x = factor(signal),
  group = signal,
  y = output)) +
  geom_boxplot() +
  theme_publ(version = 2)
if(display_plots){
  print(g_plot)
}
```



5. Running algorithm The estimation of channel capacity can be performed by using `capacity_logreg_main()` function. In most basic setting it expects four arguments: 1) data.frame with data, 2) name of input column, 3) names of output columns and 4) a path to the output directory. It can be obtained by

6. Set required parameters for the algorithm

```
signal_name <- "signal"
response_name <- "output"
```

7. Estimate channel capacity (takes several seconds)

```
tempoutput  <-
  capacity_logreg_main(
dataRaw = tempdata,
signal = signal_name,
response = response_name,
output_path = path_output_main
  )
```

8. Access and visualise results With the algorithm finished, results can be accessed by either exploring a list returned by the function or inspecting the visualisation implemented within the package (graphs created in output directory)

9. Print results of the estimation in the console

```
print(paste("Channel Capacity (bit):",
            tempoutput$cc,
            sep=" "))
```

```
## [1] "Channel Capacity (bit): 1.60456660237405"
```

```
print(paste("Optimal input probabilities,",
            "x_i = ",sort(unique(tempdata$signal)),": ",
            paste(tempoutput$p_opt,
               sep = "\n"),
            sep = " " ))
```

```
## [1] "Optimal input probabilities, x_i =  0 :  0.20276215140046"
## [2] "Optimal input probabilities, x_i =  0.01 :  0.00351480233210017"
## [3] "Optimal input probabilities, x_i =  0.1 :  0.145420938360615"
## [4] "Optimal input probabilities, x_i =  1 :  0.308751686587878"
## [5] "Optimal input probabilities, x_i =  10 :  0.126971845357898"
## [6] "Optimal input probabilities, x_i =  100 :  0.212578575961048"
```

```
print(paste("Accuracy of classification:",
            tempoutput$regression$overall[1],
            sep = " " ))
```

```
## [1] "Accuracy of classification: 0.610333333333333"
```

```
print(paste("Time of computations (sec.):",
            tempoutput$time[3],
            sep = " " ))
```

```
## [1] "Time of computations (sec.): 2.94000000000005"
```

10. Inspect object generated during computation. We verify if results saved in rds are the same as in the returned list. Full output of the estimation should be saved in "output_path/output.rds".

```
tempoutput_rds = readRDS(
  paste0(
    path_output_main,
    "/output.rds"))
print(paste("Channel Capacity assertion:",
            tempoutput_rds$cc == tempoutput$cc))
```

```
## [1] "Channel Capacity assertion: TRUE"
```

```
print(paste("Optimal input probabilities assertion:",
            sum(tempoutput_rds$p_opt != tempoutput$p_opt) == 0))
```

```
## [1] "Optimal input probabilities assertion: TRUE"
```

```
print(paste("Accuracy of classification assertion:",
            tempoutput_rds$regression$overall[1] ==
              tempoutput$regression$overall[1]))
```

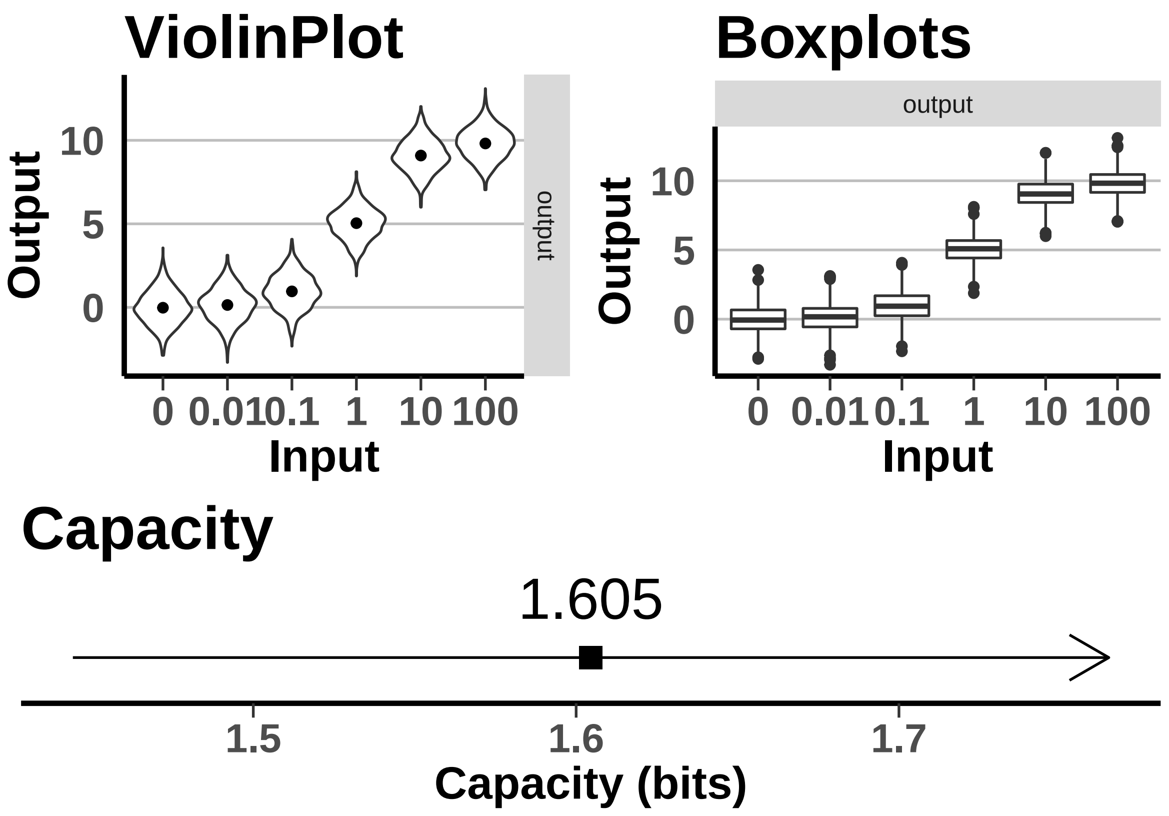
```
## [1] "Accuracy of classification assertion: TRUE"
```

```
print(paste("Time of computations assertion:",
            tempoutput_rds$time[3] ==
              tempoutput$time[3]))
```

```
## [1] "Time of computations assertion: TRUE"
```

11. Explore visualisation of the results. See pdf files in output_path/ for the visualisation of the data and capacity estimation. In the "MainPlot.pdf" the most important information are presented: mean input-output relation; distributions of output and channel capacity (see below). Graphs, as gg or gtable objects, are saved in `logGraphs` element of function output.

    ```
    if(display_plots){
      grid.arrange(tempoutput$logGraphs[[9]])
    }
    ```



12. Generate graphs of different size by specifying parameters `plot_width` and `plot_height`

    ```
    # specify paramters `plot_width` and `plot_height`
    plot_width_new  <- 10
    ```

```
plot_height_new <- 8
i_type <- "testing_basic_graphs_size"
path_output_main <- paste('output/', i_type,'/', sep="")
dir.create(path_output_main,
       recursive = TRUE)
tempoutput  <- capacity_logreg_main(
  dataRaw = tempdata,
  signal = signal_name,
  response = response_name,
  output_path = path_output_main,
  plot_width = plot_width_new,
  plot_height = plot_height_new
)
```

13. Run analysis without generating graphs by setting argument `graphs` to FALSE

```
# set argument `graphs` to FALSE
graphs_generate <- FALSE
i_type <- "testing_basic_nographs"
path_output_main <- paste('output/',i_type,'/',sep="")
dir.create(path_output_main,
       recursive = TRUE)
tempoutput  <- capacity_logreg_main(
  dataRaw = tempdata,
  signal = signal_name,
  response = response_name,
  output_path = path_output_main,
  graphs = graphs_generate
)
```

14. Run analysis with the minimal output by setting `graphs`, `scale`, `dataout`, `model_out` to FALSE. Respectively, it prevents creating graphs, scaling of the data, including data and regression model in the returned list. Such setting is useful mainly for batch processing.

```
graphs_generate <- FALSE
data_rescale <- FALSE
data_save <- FALSE
model_save <- FALSE
i_type <- "testing_basic_minimalOutput"
path_output_main=paste('output/',i_type,'/',sep="")
dir.create(path_output_main,recursive = TRUE)
tempoutput  <- capacity_logreg_main(
  dataRaw = tempdata,
  signal = signal_name,
  response = response_name,
  output_path = path_output_main,
  graphs = graphs_generate,
  scale = data_rescale,
  dataout = data_save,
  model_out = model_save
)
```

15. Diagnostics We implemented two diagnostic procedures to test the correctness of channel capacity estimation and to compute uncertainties due to sample size and over-fitting. These include:

16. Set parameters of diagnostic tests

```r
# Set a seed of a random number generator for reproducability
seed_to_use <- 12345
# number of bootstrap repetitions
bootstrap_num <- 20
# fraction of data to sample
bootstrap_frac <- 0.8
# number of repetition of overfitting test
overfitting_num <- 20
# fraction of data to use as training sample
training_frac <- 0.6
i_type <- "testing_basic_diagnostic"
path_output_main <- paste('output/',i_type,'/',sep="")
dir.create(path_output_main,
           recursive = TRUE)
```

17. Run estimation with full diagnostics by using parameters and set `testing` argument to TRUE (takes up to 5 min)

```r
tempoutput  <- capacity_logreg_main(
  dataRaw = tempdata,
  signal = signal_name,
  response = response_name,
  output_path = path_output_main,
  testing = TRUE,
  plot_width = 10 ,
  plot_height = 8,
  TestingSeed = seed_to_use,
  testing_cores = cores_num,
  boot_num = bootstrap_num,
  boot_prob = bootstrap_frac,
  traintest_num = overfitting_num,
  partition_trainfrac = training_frac
)
```

18. Inspect results of diagnostic tests. It is saved in the `testing` element of the returned list

```r
print(paste("Channel Capacity, bootstrap mean (sd): ",
            round(mean(sapply(tempoutput$testing$bootstrap,
                        function(x) x$cc)),digits = 2),
            "(",round(sd(sapply(tempoutput$testing$bootstrap,
                          function(x) x$cc)),digits=2),")",
            sep = "" ))
```

```
## [1] "Channel Capacity, bootstrap mean (sd): 1.6(0)"
```

```r
print(paste("Time of computations (sec.):",
            tempoutput$time[3],
            sep = " "))
```

```
## [1] "Time of computations (sec.): 37.39"
```

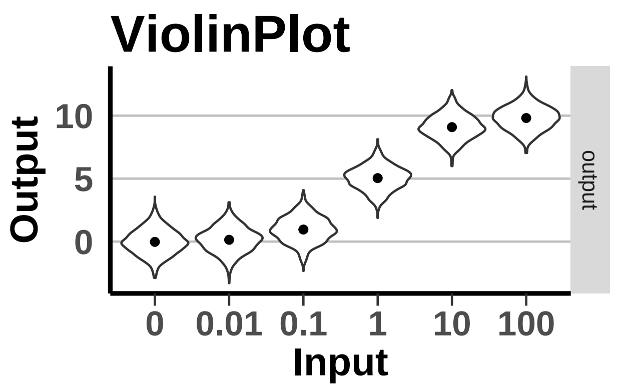
19. See the visualisation in the output directory - MainPlot.pdf. For each diagnostic test, there is a corresponding histogram of calculated capacities. This graph is also obtainable from the returned list by a command

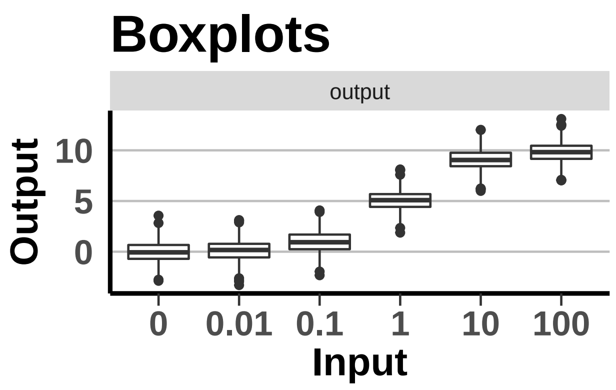
```r
if(display_plots){
  grid.arrange(tempoutput$logGraphs[[9]])
}
```

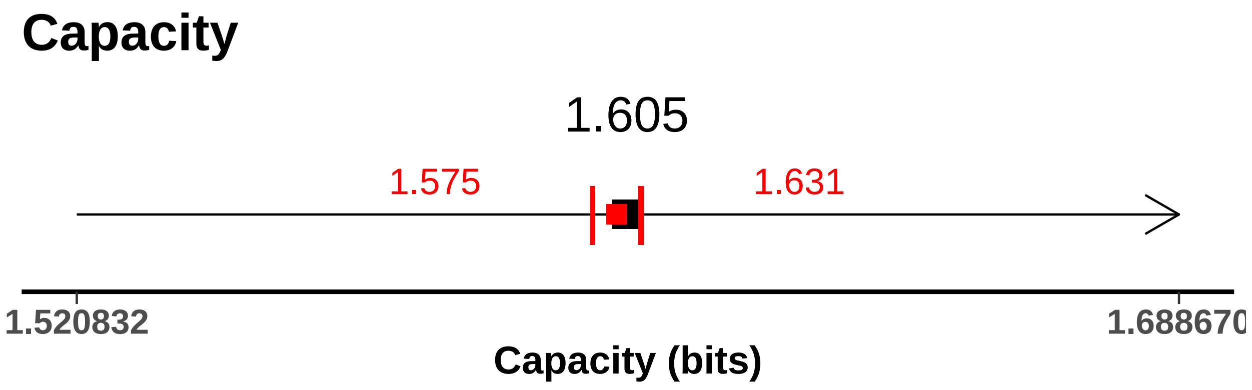
20. For each diagnostic test, we provide left- and right-tailed empirical p-values of the obtained channel capacity. They indicated if the regime of data bootstrapping or dividing data into training and testing sample influence the calculation of capacity in a significant way. A small p-value in any of these tests (e.g. <0.05) means a problem with the stability of channel capacity estimation and a possible bias due to too small sample size. P-values are printed on the MainPlot.pdf graph or can be obtained in

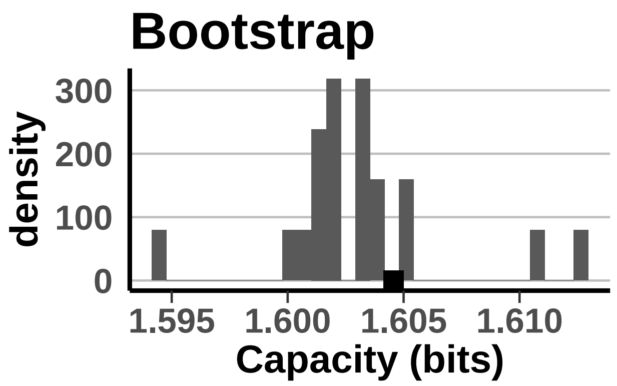
```
print(paste("P-values:",
            tempoutput$testing_pv$bootstrap[1],
            " (left-tailed); ",
            tempoutput$testing_pv$bootstrap[2],
            " (right-tailed) ",
            sep = ""))
```

```
## [1] "P-values:0.8 (left-tailed); 0.2 (right-tailed) "
```



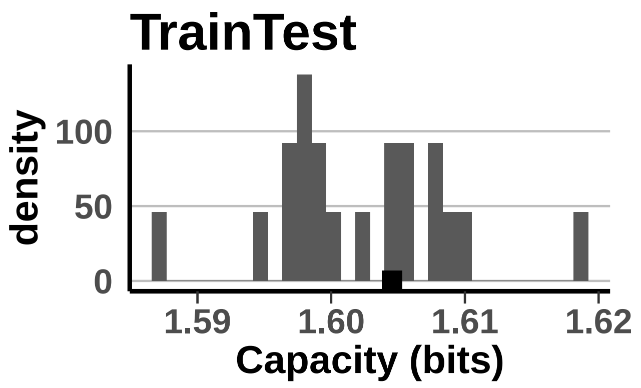






PV−left: 0.8

PV−right: 0.2



PV−left: 0.65

PV−right: 0.35

Multidimensional example

In a typical case, the use of the package will include preparing data beforehand and loading it from a file. Also, one of the most important advantages of our package includes analysis of multidimensional data (e.g. time series). To demonstrate both of these situation, we created a synthetic data set, where output is three dimensional and there are three different inputs. It can be accessed by variable `data_example2` and is processed below

1. Load and inspect synthetic dataset.

```
tempdata <- data_example2
head(tempdata)
```

```
##   signal          X1          X2          X3
## 1      0  0.15580917  0.18045768  0.02999694
## 2      0 -0.16423212  0.04744891 -0.42674054
## 3      0  0.13896586 -0.06779270 -0.05983137
## 4      0 -0.16338335  0.04204709  0.40134308
## 5      0  0.07555051  0.09766543 -0.14256936
## 6      0  0.16996827 -0.21497172 -0.09494647
```

Here, in the column named ‘signal’ the values of input are given, while columns ‘X1’, ‘X2’, ‘X3’ represents measurements of the output in three different time points.

2. Run algorithm to calculate capacity. Firstly, we set names of signal and response columns respectively, then the output directory is set up and eventually `capacity_logreg_main` is run

```
signal_name <- "signal"
response_name <- c("X1","X2","X3")
i_type <- "testing_multivariate"
path_output_main <- paste('output/',
                          i_type,
                          '/',
                          sep = "")
dir.create(path_output_main,
           recursive = TRUE,
           showWarnings = FALSE)
# computation of channel capacity
tempoutput  <- capacity_logreg_main(
  dataRaw = tempdata,
  signal = signal_name,
  response = response_name,
  output_path = path_output_main
)
```

3. Print results of the estimation in the console

```
print(paste("Channel Capacity (bit):",
            tempoutput$cc,
            sep=" "))
```

```
## [1] "Channel Capacity (bit): 1.58496240563212"
```

```
print(paste("Optimal input probabilities,",
            "x_i = ",sort(unique(tempdata$signal)),": ",
            paste(tempoutput$p_opt,
               sep = "\n"),
            sep = " " ))
```

```
## [1] "Optimal input probabilities, x_i =  0 :  0.333333330717346"
## [2] "Optimal input probabilities, x_i =  1 :  0.33333331434954"
## [3] "Optimal input probabilities, x_i =  10 :  0.333333354933114"
```

```
print(paste("Accuracy of classification:",
            tempoutput$regression$overall[1],
            sep = " " ))
```

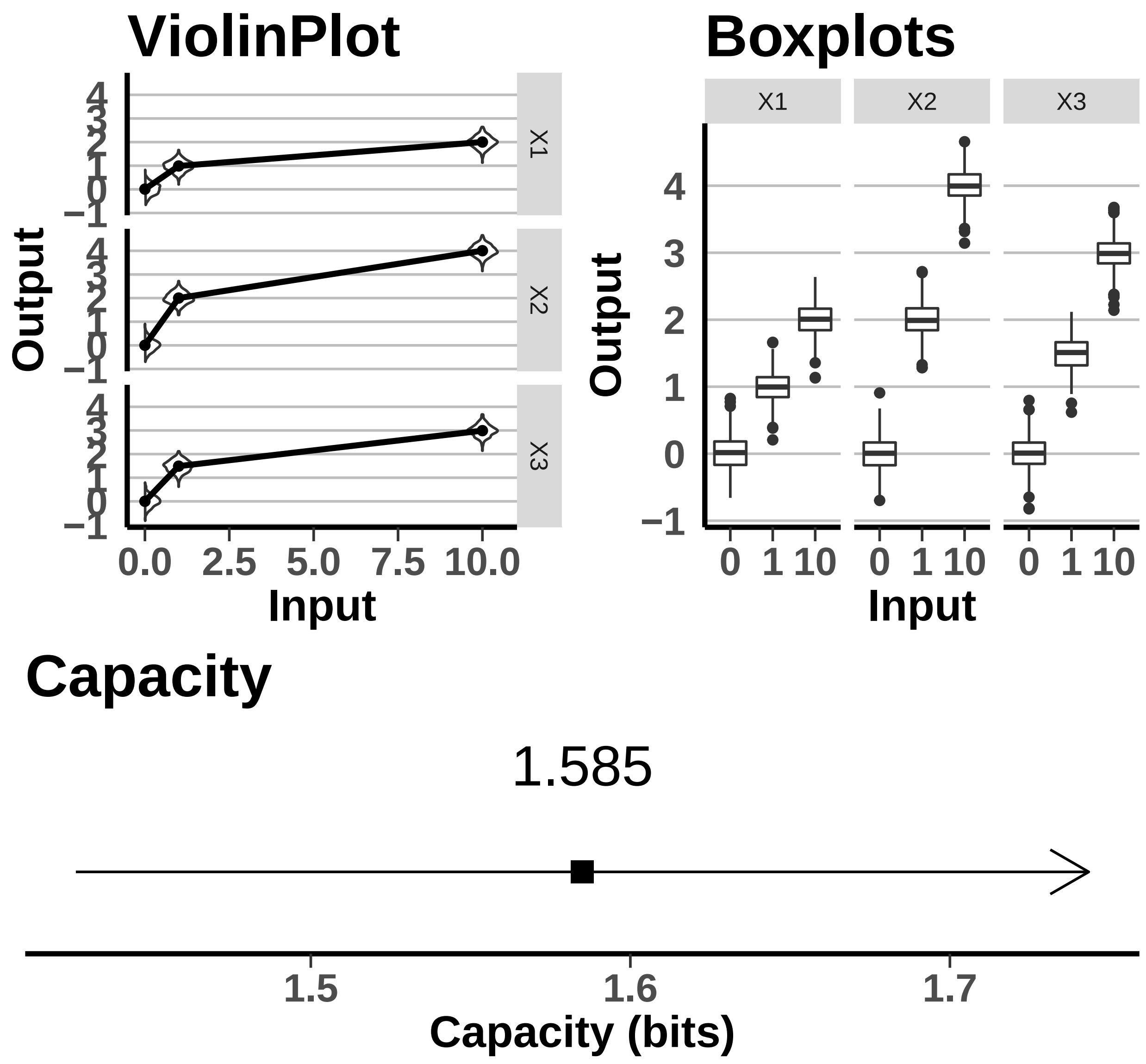
```
## [1] "Accuracy of classification: 1"
```

```
print(paste("Time of computations (sec.):",
            tempoutput$time[3],
            sep = " " ))
```

```
## [1] "Time of computations (sec.): 2.67"
```

4. See the visualisation of results in MainPlot.pdf in output directory or in the object

```
grid.arrange(tempoutput$logGraphs[[9]])
```



NfkB experimental data

In the paper describing methodological aspects of our algorithm we present the analysis of information transmission in NfkB pathway upn the stimulation of TNF-α. Experimental data from this experiment in the form of single-cell time series are attached to the package as a data.frame object and can be accessed using `data_nfkb` variable.

Each row of `data_nfkb` represents a single observation of a cell. Column 'signal' indicates the level of TNF-α stimulation for a given cell, while columns 'response_T', gives the normalised ratio of nuclear and cytoplasmic transcription factor as described in Supplementary Methods of the corresponding publication.

List of all functions in the package

Following function from CapacityLogReg package are available to the user:

- `capacity_logreg_main()` - Main wrapper function to perform analysis of channel capacity of experimental data
- `capacity_logreg_algorithm()` - Implements algorithm to estimate channel capacity using `nnet` package
- `capacity_klogreg_algorithm()` - Implements algorithm to estimate channel capacity using `glmnet` package
- `capacity_logreg_testing()` - Performs diagnostic procedures
- `capacity_output_graphs()` - Generates exploratory graphs
- `formula_generator()` - Generates a formula object based on input and output specification
- `sampling\_bootstrap()`, `sampling\_partition()`, `sampling\_shuffle()` - Used to generate subsets of data to use in diagnostic procedures
- `theme\_publ()` - Changes the visual elements of ggplot object

Function: `capacity_logreg_main()`
Main wrapper function to perform analysis of channel capacity from experimental data

Arguments

name	description	default
dataRaw	data frame with input (X) and output (Y) values in separate columns	**(required)**
signal	character indicating a name of column of dataRaw with input (X)	**(required)**
response	character vector indicating names of columns of dataRaw with measurements of outputs (Y)	**(required)**
output_path	directory in which result and graphs will be saved	**(required)**
scale	logical indicating if preprocessing (centering and scaling) should be carried out before the analysis	`TRUE`
graphs	logical indicating if standard graphs should be created	`TRUE`
model_out	logical indicating if the model object should be returned	`TRUE`
dataout	logical indicating if the dataRaw should be returned with results	`TRUE`
testing	logical indicating if diagnostics should be performed	`FALSE`
TestingSeed	the seed of random number generator to be used in diagnostics	1234
testing_cores	number of cores to use in parallel computing in diagnostics	1
boot_num	the number of bootstrap tests to be performed (used if `testing=TRUE`)	10
boot_prob	the proportion of data to be used in bootstrap (used if `testing=TRUE`)	0.8
traintest_num	the number of over-fitting tests to be performed (used if `testing=TRUE`)	10
partition_trainfrac	the proportion of data to be used as a training dataset (used if `testing=TRUE`)	0.6
side_variables	an optional character vector indicating names of columns in dataRaw with side variables, if `NULL` no side variables are included in estimation	`NULL`
resamp_num	is the number of resmapling tests to be performed (used if `testing=TRUE`)	10
plot_height	the basic dimnesion of plots (height)	4
plot_width	the basic dimnesions of plots (width)	6
cc_maxit	the maximum number of iteration to optimise channel capacity	100
lr_maxit	the maximum number of iteration to estimate logisitic model	1000
maxNWts	the maximum number of parameters in logistic regression algorithm	5000
formula_string	character object that includes a formula syntax to use in logistic model	`NULL`
glmnet_algorithm	logical indicating if the `glmnet` package should be used	`FALSE`
dataMatrix	numeric matrix with columns treated as explanatory variables in logistic model (used if `glmnet_algorithm=TRUE`)	`NULL`
glmnet_cores	the number of cores to use in parallel computing of `glmnet package` (used if `glmnet_algorithm=TRUE`)	1
glmnet_lambdanum	is the lambda parameter as in `glmnet` package (used if `glmnet_algorithm=TRUE`)	10

Values – a list with elements

name	description
cc	a numeric with the estimate of channel capacity (in bits)
p_opt	a numeric vector with estimated optimal input probability
time	processing time of the algorithm
params	a vector of parameters used in the algorithm
data	a data.frame with the raw experimental data (if `dataout=TRUE`)
regression	confusion matrix of logistic regression predictions
model	nnet object describing logistic regression model (if `model_out=TRUE`)
logGraphs	a list of gg or ggtables objects with a standard set of exploratory graphs (if `graphs=TRUE`)
testing	a list of results of diagnostic procedures, e.g. \$testing\$bootstrap has `boot_num` elements, each with results of the algorithm of each diagnostic run
testing_pv	a list of left- and right-tailed p-values of diagnostic procedures

Function: `capacity_logreg_algorithm()`
Implements algorithm to estimate channel capacity using `nnet` package

Arguments

name	description	default
data	data frame with input (X) and output (Y) values in separate columns	**(required)**
signal	character indicating a name of column of dataRaw with input (X)	**(required)**
response	character vector indicating names of columns of dataRaw with measurements of outputs (Y)	**(required)**
model_out	logical indicating if the model object should be returned	`TRUE`
side_variables	an optional character vector indicating names of columns in dataRaw with side variables, if `NULL` no side variables are included in estimation	`NULL`
cc_maxit	the maximum number of iteration to optimise channel capacity	100
lr_maxit	the maximum number of iteration to estimate logisitic model	1000
maxNWts	the maximum number of parameters in logistic regression algorithm	5000
formula_string	character object that includes a formula syntax to use in logistic model	`NULL`

Values – a list with elements

name	description
cc	a numeric with the estimate of channel capacity (in bits)
p_opt	a numeric vector with estimated optimal input probability
regression	confusion matrix of logistic regression predictions
model	`nnet` object describing logistic regression model (if `model_out=TRUE`)

Function: `capacity_klogreg_algorithm()`
Implements algorithm to estimate channel capacity using `glmnet` package

Arguments

name	description	default
dataMatrix	numeric matrix with columns treated as explanatory variables (output, Y, of the channel)	**(required)**
dataSignal	factor vector with inputs (X) of the channel length must be equal to the number of rows of dataMatrix	**(required)**
cv_core_num	the number of cores to use in parallel computing of `glmnet package`	1
lambda_num	is the lambda parameter as in `glmnet` package	10
model_out	logical indicating if the model object should be returned	`TRUE`
cc_maxit	the maximum number of iteration to optimise channel capacity	100

Values – a list with elements

name	description
cc	a numeric with the estimate of channel capacity (in bits)
p_opt	a numeric vector with estimated optimal input probability
regression	confusion matrix of logistic regression predictions
model	`glmnet` object describing logistic regression model (if `model_out=TRUE`)

Function: `capacity_logreg_testing()`
Performs diagnostic procedures

Arguments

name	description	default
data	data frame with input (X) and output (Y) values in separate columns	**(required)**
signal	character indicating a name of column of dataRaw with input (X)	**(required)**
response	character vector indicating names of columns of dataRaw with measurements of outputs (Y)	**(required)**
output_path	directory in which result and graphs will be saved	**(required)**
TestingSeed	the seed of random number generator to be used in diagnostics	1234
testing_cores	number of cores to use in parallel computing in diagnostics	1
boot_num	the number of bootstrap tests to be performed (used if `testing=TRUE`)	10
boot_prob	the proportion of data to be used in bootstrap (used if `testing=TRUE`)	0.8
traintest_num	the number of over-fitting tests to be performed (used if `testing=TRUE`)	10
partition_trainfrac	the proportion of data to be used as a training dataset (used if `testing=TRUE`)	0.6
side_variables	an optional character vector indicating names of columns in dataRaw with side variables, if `NULL` no side variables are included in estimation	`NULL`
resamp_num	is the number of resmapling tests to be performed (used if `testing=TRUE`)	10
cc_maxit	the maximum number of iteration to optimise channel capacity	100
lr_maxit	the maximum number of iteration to estimate logisitic model	1000
maxNWts	the maximum number of parameters in logistic regression algorithm	5000
formula_string	character object that includes a formula syntax to use in logistic model	`NULL`
glmnet_algorithm	logical indicating if the `glmnet` package should be used	`FALSE`
dataMatrix	numeric matrix with columns treated as explanatory variables in logistic model (used if `glmnet_algorithm=TRUE`)	`NULL`
glmnet_cores	the number of cores to use in parallel computing of `glmnet package` (used if `glmnet_algorithm=TRUE`)	1
glmnet_lambdanum	is the lambda parameter as in `glmnet` package (used if `glmnet_algorithm=TRUE`)	10

Values – a list with elements

bootstrap	list of size `boot_num`, where each element is the returned value of `capacity_logreg_algorithm()` from a single run of bootstrap	
traintest	list of size `traintest_num`, where each element is the returned value of `capacity_logreg_algorithm()` from a single run of over-fitting test	
resamplingMorph	list of size `resamp_num`, where each element is the returned value of `capacity_logreg_algorithm()` from a single run of resampling test (used if `side_variables` is not `NULL`)	
bootResampMorph	list of size `resamp_num`, where each element is the returned value of `capacity_logreg_algorithm()` from a single run of resampling test II (used if `side_variables` is not `NULL`)	

Function: `capacity_output_graphs()`
Generates exploratory graphs

Arguments

name	description	default
data	data frame with input (X) and output (Y) values in separate columns	**(required)**
signal	character indicating a name of column of dataRaw with input (X)	**(required)**
response	character vector indicating names of columns of dataRaw with measurements of outputs (Y)	**(required)**
output_path	directory in which result and graphs will be saved	**(required)**
cc_output	logical indicating if preprocessing (centering and scaling)	`TRUE`
height	the basic dimnesion of plots (height)	4
width	the basic dimnesions of plots (width)	6

Values – a list with elements

name	description
1	A comprehensive summary plot
2	Input-Output relation
3	Boxplots of data
4	Violin plots of data
5	Histograms of data
6	Boxplot of side variables in data
7	Capacity results
8	Density plots
9	A simple summary plot

Function: `formula_generator()`
Generates a formula object based on input and output specification

Arguments

name	description	default
signal	character indicating a name of column of dataRaw with input (X)	**(required)**
response	character vector indicating names of columns of dataRaw with measurements of outputs (Y)	**(required)**
side_variables	an optional character vector indicating names of columns in dataRaw with side variables, if `NULL` no side variables are included in estimation	`NULL`

Values – a list with elements

name	description	
formula_string	character object that includes a formula syntax to use in logistic model	`NULL`

Function: `sampling_bootstrap()`, `sampling_partition()`, `sampling_shuffle()`
Used to generate subsets of data to use in diagnostic procedures

Arguments

name	description	default
data	data.frame to be resampled	**(required)**
dataDiv	character indicating column of data, with respect to which split the data; only in `sampling_bootstrap()` and `sampling_partition()`	**(required)**
prob	the of data that should be sampled from the whole dataset; only in `sampling_bootstrap()`	**(required)**
partition_trainfrac	the proportion of data to be used as a training dataset; only in `sampling_partition()`	**(required)**
side_variables	vector of characters indicating columns of data the will be reshuffled; only in `sampling_shuffle()`	**(required)**

Values – a data.frame with the same structure as initial data object

Function: `theme_publ()`
Changes the visual elements of ggplot object

Arguments

name	description	default
version	possible values: 1,2,3. Selects different coloring and presentation options	1
base_size	the size of font to use in graph	12
base_family	the type of font to use in graph	sans

Values – a ggplot theme object