Procedure for Testing CapacityLogReg

T.Jetka

May 31, 2018

In this tutorial, we will demonstrate how to use and test CapacityLogReg package to calculate channel capacity from experimental data. A full analysis starting from preparation of data through estimation and visualisation of results and diagnostic statistics is shown. Also other functionalities accompanying the package will be presented.

We divided testing procedures into three parts (approximate running time with a single core is given in brackets):

1. Quick guide for setting up the package (approx. time: 5-20 minutes),
2. Examples of use (approx. time: 1 hour) with codes in file `testing_procedures.R`,
3. Replicating results from manuscript (approx. time: at least 4 hours) with codes in files `paper_MP.R` and `paper_SI.R`.

R scripts for testing are provided in the archive together with this document.

Quick guide for setting up the package

To run the following exercises, you will need the R environment (tested using version 3.5.0 - see installation instructions) and preferably RStudio (see installation instructions) for an efficient workflow. 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. From hardware perspective, no specific requirements are needed, but we recommend using machine with 4GB+ RAM memory and 2Ghz+ processor, especially if results from the manuscript are to be replicated with all the details or a large dataset will be analysed.

Our package can be obtained from GitHub. For that, a `devtools` package is needed. Therefore, in order to install CapacityLogReg, open RStudio (or base R) and run following commands in the R console (takes ~2 minutes)

```
install_packages("devtools")
library(devtools)
install_github("TJetka/CapacityLogReg")
```

All dependencies (if not present) will be installed as well. Please note that running of some of following instructions could require installing additional packages.

In case you prefer to use an isolated Docker image of our package, we provide it here XXX. Testing scripts are located in `/usr/home/CapacityLogReg/` directory within the container's file system.

Using the package

Our package is primarily designed to aid in applying information theory to experimental data in systems biology. Specifically, 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 $P(Y|X)$.

Preparing data

In a typical example, X represents the concentration of the ligand that activates signaling pathway, while Y is a nuclear concentration of transcription factor (possibly multidimensional, if observed in many time points). Conceptually, a full experimental dataset in such setting can be represented as a table

input	output 1	output 2	output 3	...
n_1 $\left\{\begin{array}{l} x_1 \\ \vdots \\ x_1 \end{array}\right.$	$\begin{array}{c} y^1_{1,1} \\ \vdots \\ y^1_{n_1,1} \end{array}$	$\begin{array}{c} y^1_{1,2} \\ \vdots \\ y^1_{n_1,2} \end{array}$	$\begin{array}{c} y^1_{1,3} \\ \vdots \\ y^1_{n_1,3} \end{array}$	
n_2 $\left\{\begin{array}{l} x_2 \\ \vdots \\ x_2 \end{array}\right.$	$\begin{array}{c} y^2_{1,1} \\ \vdots \\ y^2_{n_2,1} \end{array}$	$\begin{array}{c} y^2_{1,2} \\ \vdots \\ y^2_{n_2,2} \end{array}$	$\begin{array}{c} y^2_{1,3} \\ \vdots \\ y^2_{n_2,3} \end{array}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	...
n_m $\left\{\begin{array}{l} x_m \\ \vdots \\ x_m \end{array}\right.$	$\begin{array}{c} y^m_{1,1} \\ \vdots \\ y^m_{n_m,1} \end{array}$	$\begin{array}{c} y^m_{1,2} \\ \vdots \\ y^m_{n_m,2} \end{array}$	$\begin{array}{c} y^m_{1,3} \\ \vdots \\ y^m_{n_m,3} \end{array}$	

where each row represents a single observation, x_i is a specific concentration of the ligand, while $y^i_{j,d}$ is the measurement of transcription factor for j-th cell stimulated by i-th concentration of the ligand in d-th time point.

As a consequence, the most natural representation of such dataset is a `data.frame` object with :

i) one column representing the input;
ii) columns with experimental measurements of outputs.

Therefore, we assumed that in order to use our package, experimental dataset must be initially converted to such format. See for example our dataset of NfkB experiments, which accompanies the package under the variable `data_nfkb`

	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

	signal	response_0	response_3	response_6
11542	100ng	0.1997087	0.2886905	1.9324093

It can be show in R by running

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

Basic workflow

A generic approach to estimate channel capacity goes in following steps

1. Converting raw experimental data to a `data.frame` object with

- rows representing observations
- a column with values of the input for each observation (variable type: factor)
- columns with output's measurements for each observation (variable type: numerical)

2. Performing analysis using a function `capacity_logreg_main`. It requires:

- the data of class `data.frame`
- a name of the column with input values
- a vector of names of columns with output values
- a directory, where the results of the estimation will be saved to

3. Inspecting results in a returned list, which includes following elements:

- $cc - channel capacity estimate (in bits)
- $p_opt - distribution of the optimal input distribution
- $model - `nnet` object with the summary of estimated logistic regression model

4. Visualising results - a set of graphs is saved into output directory. Main results are presented on `Main_Plot.pdf`.

The use of such scheme in practice is presented below for several examples.

Running simple examples

To demonstrate how to use CapacityLogReg, we will generate a synthetic dataset of a channel, for which the conditional output distribution, $\ln(Y)|X$, is Gaussian. This example is analogous to the Test example 2 from SI (Section 3.2). We assume that

i) input, X, has 6 different levels between 0 and 100;
ii) conditional output, $\ln(Y)|X = x$, is a one-dimensional Gaussian distribution with mean $10 \cdot \frac{x}{1+x}$ and standard deviation 1;
iii) for each x, there are 1000 observations.

We encourage to change different parameters of this example to investigate influence of various parameters on channel capacity and performance of our method. Following the code below will last no more than 1 hour.

Initialization

1. Open `testing_procedures.R` script in RStudio (you can also follow comments in the script)

2. Set a working directory, where output and figures will be saved (line 6), e.g.

```
path_wd <- "~/path/to/folder/testing_procedures/"
setwd(path_wd)
```

3. Load CapacityLogReg package (line 7)

```
library(CapacityLogReg)
```

4. Set a seed of a random number generator for reproduciblity

```
set.seed(3349)
```

5. Set number of cores during computations

```
cores_num <- 8
```

6. Define if you want to display plots in the output.

```
display_plots <- TRUE
```

Generating synthetic dataset

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 commands.

6. 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
```

7. Create output directory

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

8. Generating synthetic data

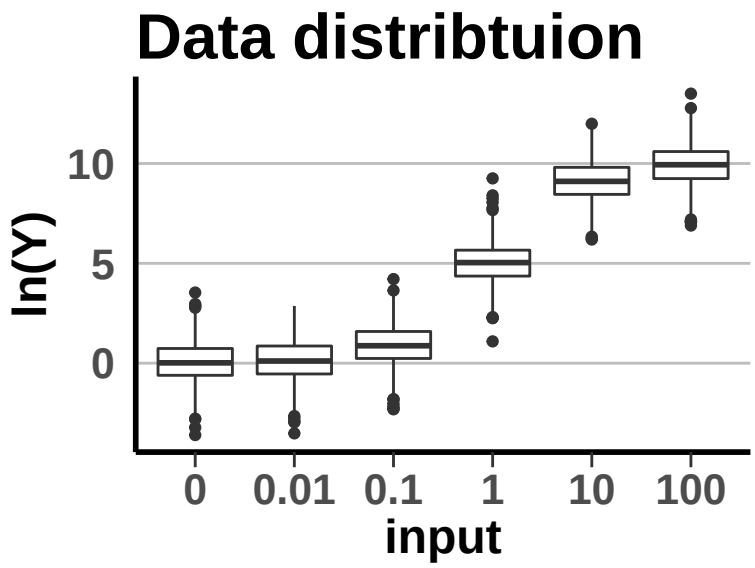
```
# concentration of input; spans from 0 to saturation.
xx <-
  signif(
c(0,
  exp(
    seq(
      from = log(0.01),
```

```
      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  0.34835806
## 2      0 -0.78697483
## 3      0  0.81024597
## 4      0  0.07565623
## 5      0 -0.13292540
## 6      0  0.15963531
```

9. 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)
}
```



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

10. Set required parameters for the algorithm

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

11. Estimate channel capacity (takes several seconds)

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

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)

12. Print results of the estimation in the console

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

```
## [1] "Channel Capacity (bit): 1.57870721310165"
```

```
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 :  0.196414629740068"
## [2] "Optimal input probabilities, x_i =  0.01 :  0.0213765104176143"
## [3] "Optimal input probabilities, x_i =  0.1 :  0.12955652377174"
## [4] "Optimal input probabilities, x_i =  1 :  0.30577294887575"
## [5] "Optimal input probabilities, x_i =  10 :  0.141269433599551"
## [6] "Optimal input probabilities, x_i =  100 :  0.205609953595277"
```

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

```
## [1] "Accuracy of classification: 0.589"
```

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

```
## [1] "Time of computations (sec.): 5.42"
```

13. 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]))
```

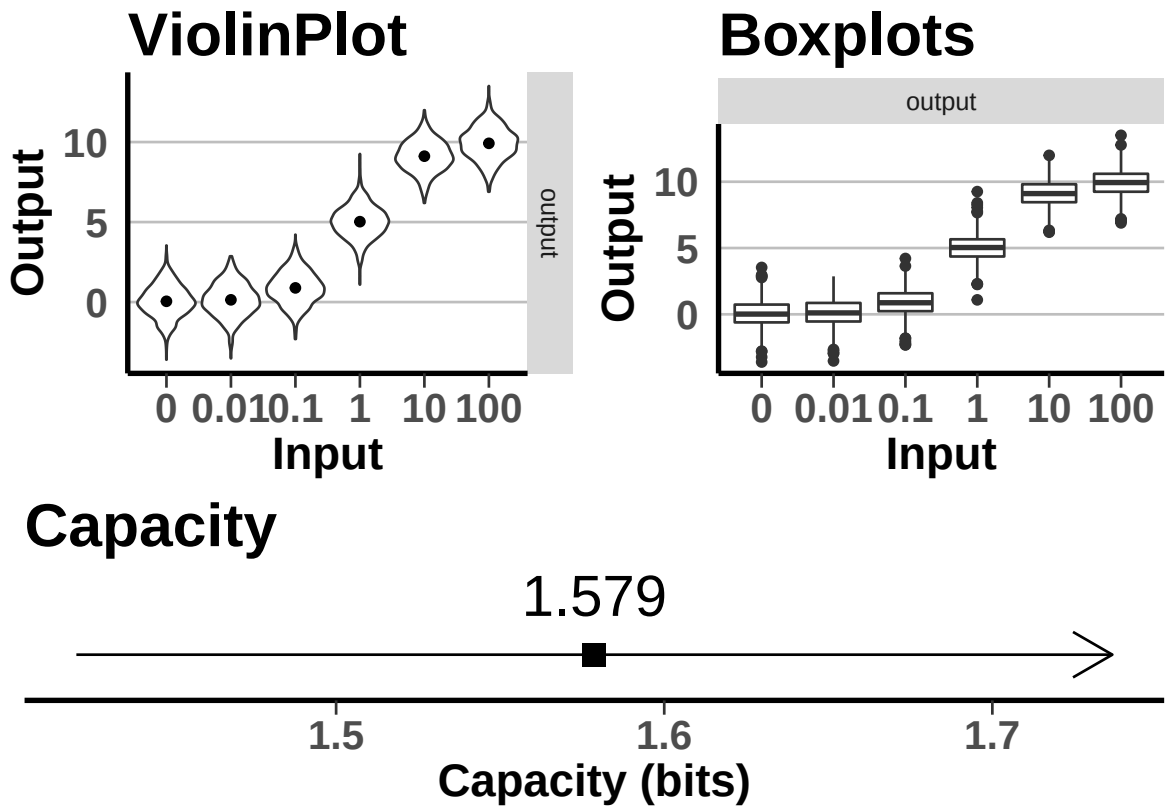
```
## [1] "Accuracy of classification assertion: TRUE"
```

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

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

14. 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]])
}
```



15. 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
)
```

16. 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
)
```

17. 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
)
```

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:

- Bootstrap - where capacity is re-calculated using x% of data sampled from original dataset without replacement. After repeating procedures n times, basic statistical measures can be obtained as a form of an error estimate.
- TrainTest - division data into Training and Testing datasets - where logistic regression model is estimated using x% of data (training dataset) and capacity is computed by evaluating this model using remaining (1-x)% of data (testing dataset). After repeating n times, this ensures the user that there is no problem with over-fitting in the estimation.

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

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

In our simple example, those diagnostic can be calculated by running

18. Set parameters of diagnostic tests

```
# 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)
```

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

```
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
)
```

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

```
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): 1.58(0.01)"
```

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

```
## [1] "Time of computations (sec.): 35.36"
```

21. 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

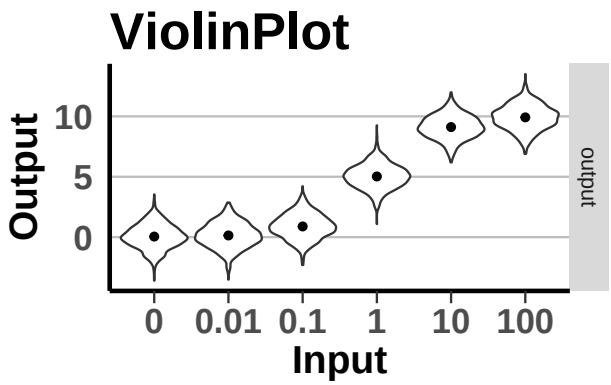
```
if(display_plots){
  grid.arrange(tempoutput$logGraphs[[9]])
```

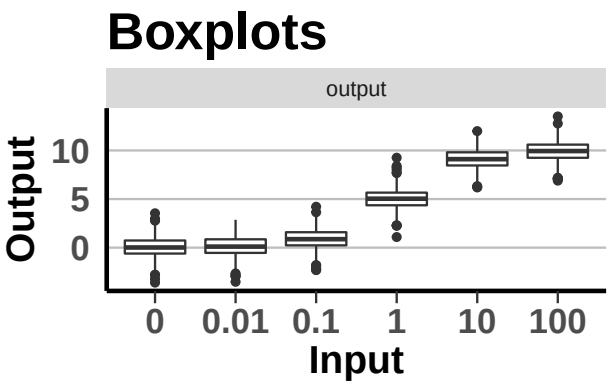
```
}
```

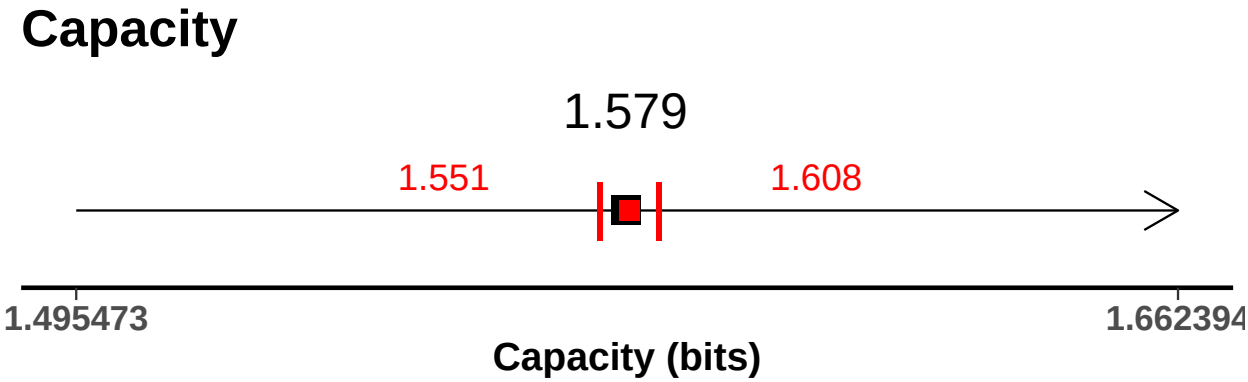
22. For each diagnostic test, we provide left- and right-tailed empirical p-values of the obtained channel capacity. They indicated if the regime of data bootstrapping or dividing data into training and testing sample influence the calculation of capacity in a significant way. A small p-value in any 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

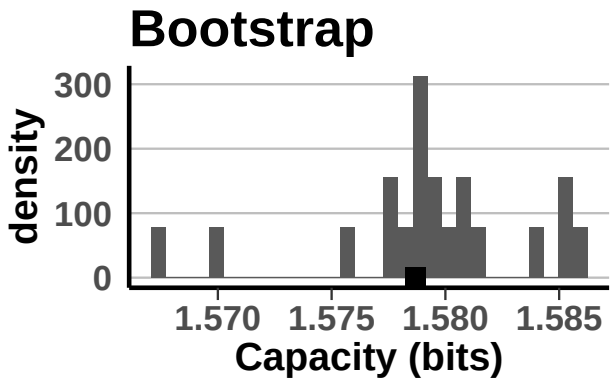
```
print(paste("P-values:",
            tempoutput$testing_pv$bootstrap[1],
            " (left-tailed); ",
            tempoutput$testing_pv$bootstrap[2],
            " (right-tailed) ",
            sep = ""))
```

```
## [1] "P-values:0.65 (left-tailed); 0.35 (right-tailed) "
```



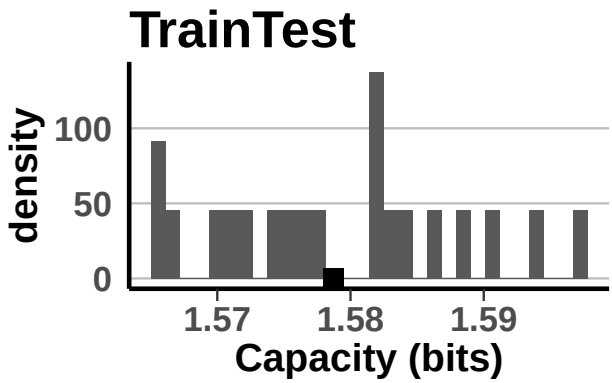






PV−left: 0.3

PV−right: 0.7



PV−left: 0.5

PV−right: 0.5

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 (takes 1 minute)

23. 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.

24. 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
)
```

25 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 = " " ))
```

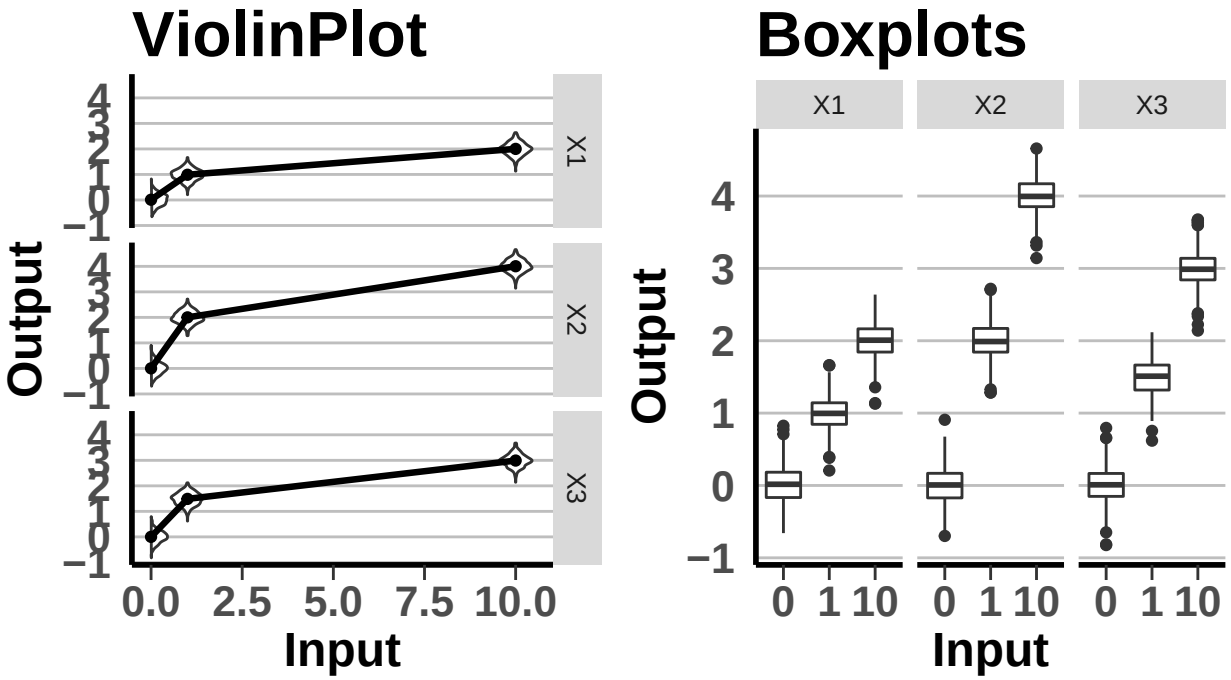
```
## [1] "Accuracy of classification: 1"
```

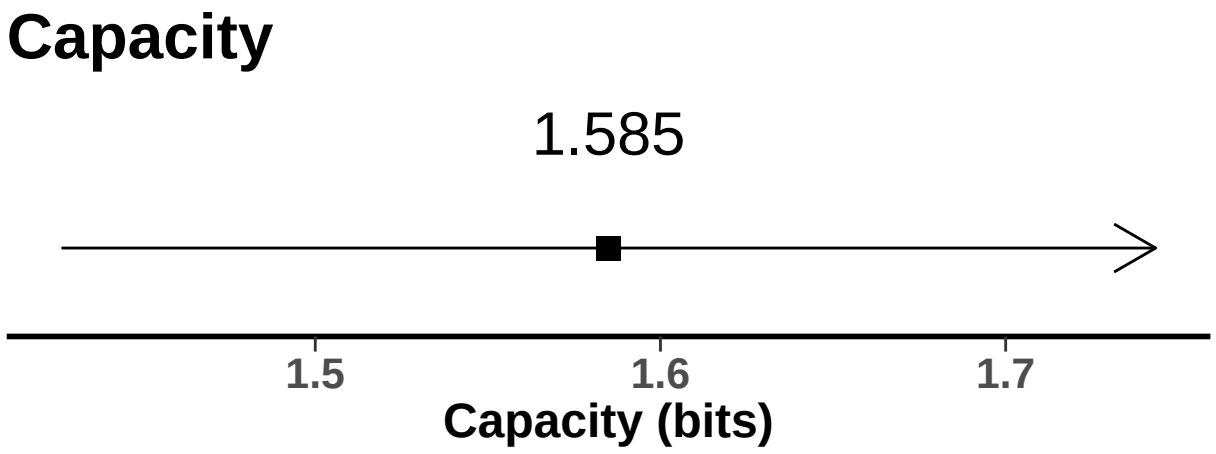
```
print(paste("Time of computations (sec.):",
            tempoutput$time[3],
            sep = " " ))
```

```
## [1] "Time of computations (sec.): 3"
```

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

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





Replicating results from the manuscript

Instructions how to replicate results presented in the manuscript are shown separately for i) main paper (MP) and ii) supplementary methods (SI) with codes in `paper_MP.R` and `paper_SI.R` respectively. In those examples, we explored various aspects of our experimental data or many different scenarios of synthetic datasets. In consequence, re-creating figures with all details can last up to 24 hours, if only a single processor core is used. Therefore, we provide here a simplified pipeline of our workflow, which requires about 4 hours of computations. Simplification of the analysis does not have significant impact on the qualitative aspects of results though. The full analysis can be obtained by uncommenting specified lines in the scripts.

Main Paper - NfkB analysis

Here we replicate the analysis of experimental data describing the behaviour of Nf-kB signalling pathway. Specifically, we replicate Fig. 1 from MP. Single-cell time series in a data.frame object are attached to the package and can be accessed using `data_nfkb` variable (lazy load). For details of experiments see Supplementary Methods.

Our investigation of the influence of the TNF-α on NfkB responses includes the analysis of information transfer by estimating channel capacity (Fig. 1A-B) and assessing the probability of discrimination between each pair of the TNF-α stimulation (Fig. 1C-D). In addition, both of above approaches are applied to two variants of data: i) for time point (TP) measurements; ii) for time-series (TS) measurements.

The code is given in `paper_MP.R` file and is divided into three sections that should be run in order

1) Preliminary - setting up packages and working environment. (lines 54-71)
2) Capacity - replicates Fig.1 A-B (lines 73-174)
3) Probabilities of discrimination - replicates Fig.1 C-D (lines 178-293)

In default mode (5 repetition of bootstrap), running Capacity section takes approx. 2 hours with a single core. Set number of cores for parallel processing in line 77. For a graphs exactly like in MP, set line 78 to

```
analysis_type="long"
```

Running Probability of discrimination section takes about 3 minutes. Please follow instructions and run code in `paper_MP.R` to replicate the results from MP. Installation of additional packages is needed.

Supplementary Methods - validation and comparison

As for last part, in script `paper_SI.R` we show codes to replicate examples presented in Supplementary Methods. There, firstly the validation of our methods is shown for several simple examples and secondly we compare the performance of our method to the KNN method.

Therefore the code is divided into three sections that should be run in order

1) Preliminary - setting up packages and working environment
2) Comparison - replicates Fig. S1 - shows the comparison of our method to the KNN approach.
3) Validation - replicates Fig. S3 - shows the performance of our method in four examples of simple channels

In default mode (10 repetition of data sampling), running Validation section takes 1 hour, similarly computations in Comparison section also take approximately 1 hour with a single core. Set number of cores for parallel processing in line 71. For a graphs exactly like in SI, set line 72 to

```
analysis_type="long"
```

Please follow instructions and run codes in `paper_SI.R` to replicate the results from SI. Installation of additional packages is needed.