Testing procedures of Statistical Learning based Estimation of Mutual Information (SLEMI) R package

T. Jetka, K. Nienałtowski, T. Winarski, S. Błoński, M. Komorowski

July 22, 2018

Contents

This document provides testing procedures of the SLEMI package. The procedures aim to verify installation of the package as well as the performance of its key functions. The document can also serve as a quick package tutorial, alternative to the more comprehensive *User Manual*. The theory behind the package, as well as its broader applicability, is described in the paper

Jetka et al., Information-theoretic analysis of multivariate signaling responses using SLEMI,

hereafter, referred to as *Main Paper* (MP). The Supplementary Information of the paper will be referred to as SI. R scripts that implement testing steps described in the subsequent sections are provided in the folder `testing_procedures` of the package.

The testing procedure is divided into four components listed below.

1. Quick guide for setting up the package (implemented in the script `testing_procedures.R`; approx.running time: 5-20 minutes)
2. Numerical test examples (implemented in the script `testing_procedures.R`; approx. running time: 1 hour)
3. Replication of the analysis of the NFkB dataset presented in the main paper (implemented in the script `paper_MP.R`; approx. running time: at least 2 hours)
4. Replication of the selected results presented in supplementary information, specifically of Fig. S1 and S3 (implemented in the script `paper_SI.R`; approx. running time: at least 2 hours);

The running times ware estimated using single core computations on a regular laptop.

Execution of the testing procedure requires the following hardware and software requirements.

1 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)

2 Requirements - Software

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

Apart from the base installation of R, SLEMI requires several additional R packages, as described in the User Manual. Are these packages not found, they will be installed automatically within the installation procedure described below.

3 Installation

The package can be directly installed from GitHub. For installation, open RStudio (or base R) and run the following commands in the R console

```
install_packages("devtools") # run if not installed
library(devtools)
install_github("sysbiosig/SLEMI")
```

4 Key functions of the package

The three functions listed below constitute the key components of the package.

1. `mi_logreg_main()` enables calculation of the mutual information
2. `capacity_logreg_main()` enables calculation of the information capacity
3. `prob_discr_pairwise()`serves to calculate probabilities of correct discrimination between pairs of input values

In the subsequent, we focus on the function `capacity_logreg_main()` ,which is the main contribution of the package, however the use of the functions `prob_discr_pairwise()` and `mi_logreg_main()` are also briefly demonstrated.

5 Input data

Each of the above functions takes, as the input, experimental data of the form described in Section 1.1 of the Supplementary Information. Single cell responses, y_j^i, are assumed to follow the probability distribution $P(Y|X = x_i)$

$$y_j^i \quad \sim P(Y|X = x_i),$$

and, hence, the input dataset can be conceptually represented as the table

input	output 1	output 2	output 3	...
n_1 { x_1	$y_1^1(1)$	$y_1^1(2)$	$y_1^1(3)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
x_1	$y_{n_1}^1(1)$	$y_{n_1}^1(2)$	$y_{n_1}^1(3)$	
n_2 { x_2	$y_1^2(1)$	$y_1^2(2)$	$y_1^2(3)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
x_2	$y_{n_2}^2(1)$	$y_{n_2}^2(2)$	$y_{n_2}^2(3)$	
				...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
n_m { x_m	$y_1^m(1)$	$y_1^m(2)$	$y_1^m(3)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
x_m	$y_{n_m}^m(1)$	$y_{n_m}^m(2)$	$y_{n_m}^m(3)$	

where each row represents a single cell response to the input value x_i. The single cell response y_j^i can have multiple entries referred to as $y_j^i(d)$. For each input values x_i different number of cells, n_i, are assumed to be measured.

Therefore, we assumed that the input dataset is represented as the `data.frame` object with:

i) the first column representing the input value;
ii) the subsequent columns representing the experimental measurements of (multidimensional) output.

This data structure is exemplified by the NfkB dataset discussed in the main paper. The dataset is available within the package under the variable `data_nfkb`

signal	response_0	response_3	response_6
0ng	0.3840744	0.4252835	0.4271986
0ng	0.4709216	0.5777821	0.5361948
0ng	0.4274474	0.6696011	0.8544916
8ng	0.3120216	0.3475484	1.0925967
8ng	0.2544961	0.6611051	2.2894928
8ng	0.1807391	0.4336810	1.9783171
100ng	1.3534083	3.0158004	5.1592848
100ng	1.7007936	2.2224497	3.5463418
100ng	0.1997087	0.2886905	1.9324093

Selected rows of the data set can be displayed in R by running

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

6 Numerical example 1

To demonstrate, how to use SLEMI, we first present a numerical example that involves the generation of a synthetic dataset. We consider the conditional output probability, $P(Y|X = x_i)$, given by the log-normal distribution. Specifically, we assume that

i) input, X, takes 6 different values between 0 and 100;
ii) conditional output, $Y|X = x$, is a one-dimensional log-normal distribution $\exp\{\mathcal{N}(10 \cdot \frac{x}{1+x}, 1)\}$;
iii) for each value of X, the sample consist of 1000 one-dimensional observations.

This example is analogous to the Test example 2 of the SI (Section 3.2).

Execution of the code below demonstrates, step-by-step, how to generate the dataset, calculate capacity, visualize results as well as how to perform diagnostics. Description of the diagnostic tests assumes that the reader is familiar with the theory of statistical hypothesis testing and bootstrap techniques. The diagnostics section can be skipped by less experience users, as it is not necessary to use the key functionalities of the package.

6.1 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 SLEMI package (line 62)

```
library(SLEMI)
```

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

```
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
```

6.2 Generation of the synthetic dataset

Functions of SLEMI package require a dataset in a format of the `data.frame` described above. For the presented test example, it will be generated by the following commands.

7. Setting parameter values

```
# 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
```

8. Creation of the output directory

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

9. Generation of the synthetic data

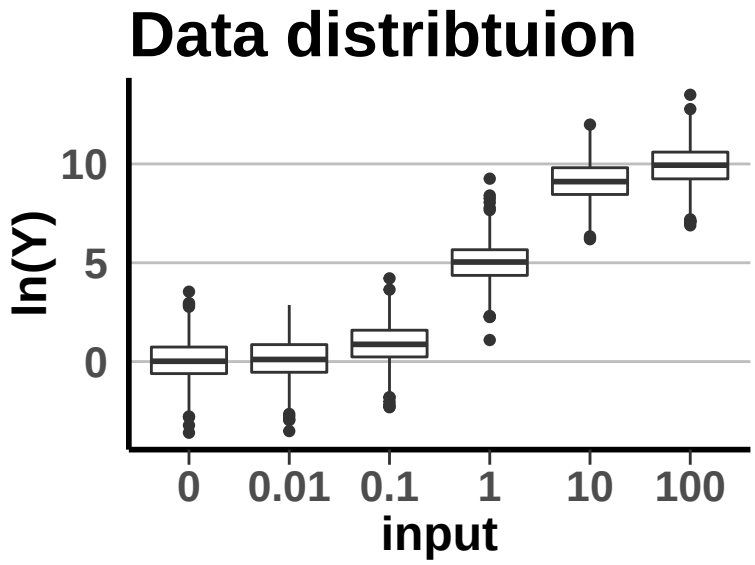
```
# concentration of input; spans from 0 to saturation.
xx <-
signif(
c(0,
exp(
seq(
from = log(0.01),
to = log(100),
length.out = input_num-1))),
digits = 2)
# mean of the dose-response relation; Michaelis-Menten assumed
example_means <- 10*(xx/(1+xx))
example_sds <- rep(dist_sd, input_num)
tempdata <-
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
```

10. Preview of the distribution of the data.

```
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)
}
```



6.3 Calculation of the information capacity

The estimation of the channel capacity is performed by the function `capacity_logreg_main`. In the basic setting it expects four arguments: 1) `dataRaw` - data frame with experimental data, 2) `signal` - name of input column, 3) `response` - names of output columns and 4) `output_path` - a path to the output directory. It is executed by the following commands

11. Setting required parameters for the algorithm

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

12. Estimation of the channel capacity (takes several seconds)

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

6.4 Visualisation of results

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

13. Displaying 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.): 5.36"
```

14. Inspection of the object generated during the computations. We verify if results saved in rds file 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]))
```

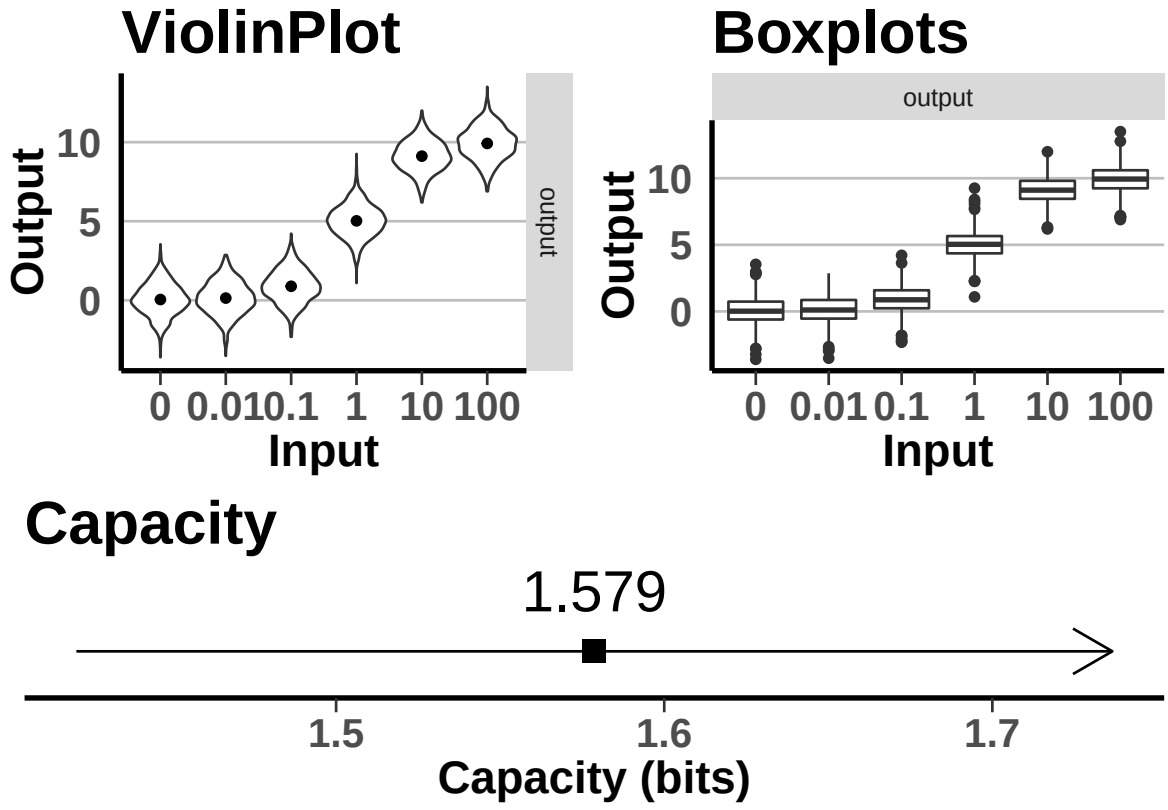
```
## [1] "Accuracy of classification assertion: TRUE"
```

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

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

15. Visualization of the results. See pdf files in output_path/ for the visualization of the data and of the capacity estimation. In the "MainPlot.pdf" the most important information are presented: mean input-output relation; distributions of output and channel capacity (see below). Those graphs, as `gg` or `gtable` objects, are saved in `logGraphs` element of the list returned by the function.

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



16. Generation of the graphs of different sizes 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
)
```

17. Running of the 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
)
```

18. Running of the 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
)
```

6.5 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 of the 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 occurrence of the over-fitting,
- partition_trainfrac (default= 0.6) - a fraction of initial observations to use as a training dataset in the testing the occurrence of the over-fitting

In our simple example, the above diagnostic tests can be calculated by running

19. Set parameters of diagnostic tests

```
# Set a seed of a random number generator for reproducibility
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)
```

20. Running of the estimation with full diagnostics by using parameters and set `testing` argument to TRUE (takes up to 5 min)

```
tempoutput_diag   <- 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
)
```

21. Inspection of the 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_diag$testing$bootstrap,
function(x) x$cc)),digits = 2),
"(",round(sd(sapply(tempoutput_diag$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_diag$time[3],
sep = " "))
```

```
## [1] "Time of computations (sec.): 37.06"
```

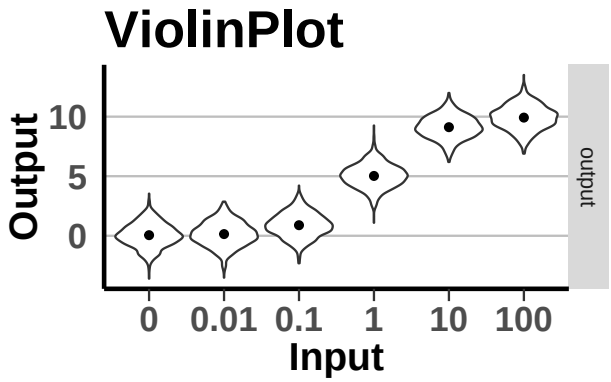
22. See the visualization in the output directory - MainPlot.pdf. For each diagnostic test, there is a corresponding histogram of calculated capacities. This graph is also obtainable from the returned list by a command

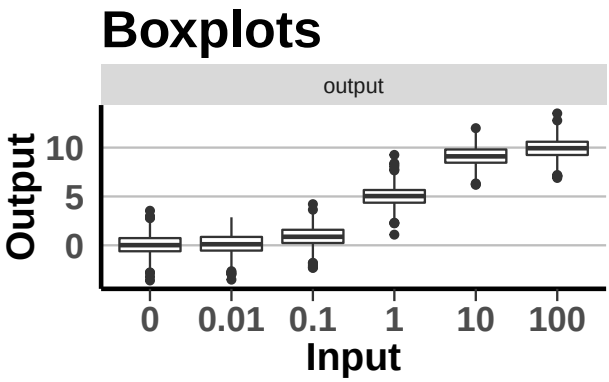
```
if(display_plots){
grid.arrange(tempoutput_diag$logGraphs[[9]])
}
```

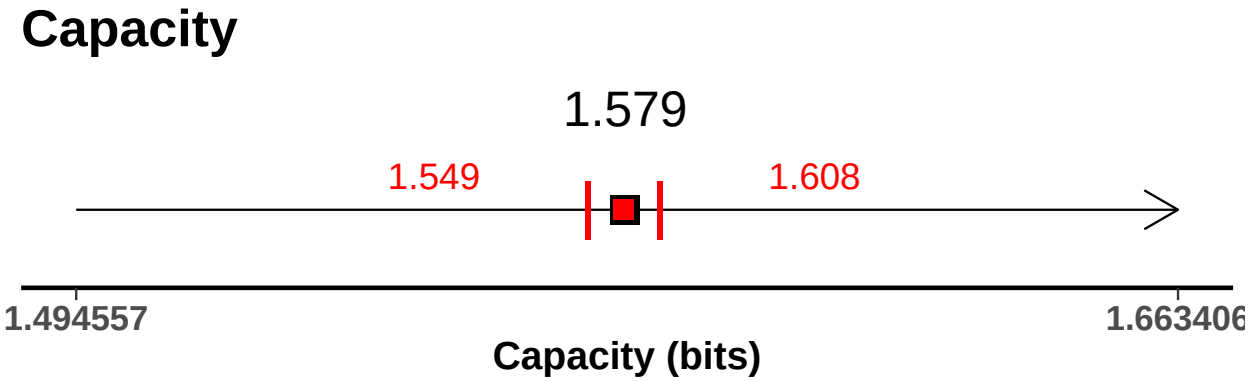
23. For each diagnostic test, we provide left- and right-tailed empirical p-values of the obtained channel capacity. The p-values serve to verify whether the regime of data bootstrapping or dividing data into training and testing sample have a significant impact on the calculation of the capacity. A small p-value in any of these tests (e.g. <0.05) indicates 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

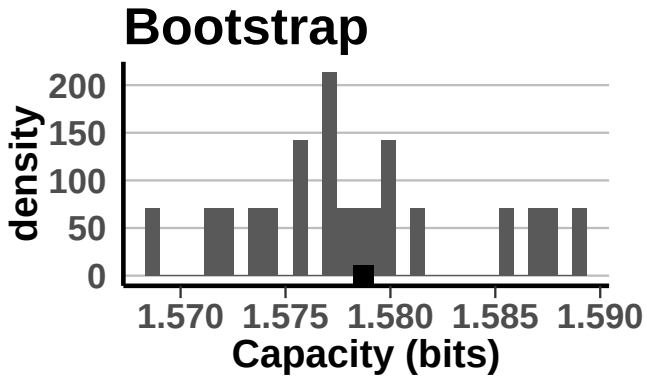
```
print(paste("P-values:",
tempoutput_diag$testing_pv$bootstrap[1],
" (left-tailed); ",
tempoutput_diag$testing_pv$bootstrap[2],
" (right-tailed) ",
sep = ""))
```

```
## [1] "P-values:0.6 (left-tailed); 0.4 (right-tailed) "
```



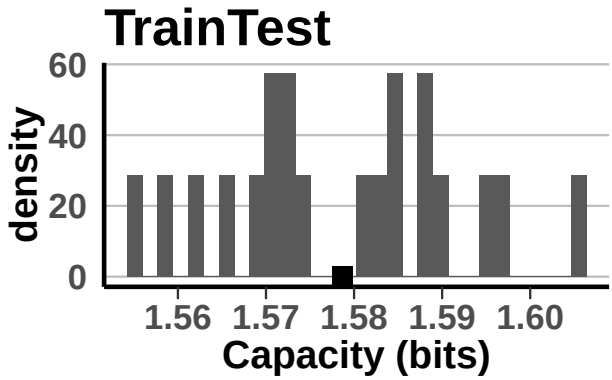






PV−left: 0.6

PV−right: 0.4



PV−left: 0.5

PV−right: 0.5

6.6 Additional exploration

Our methodology to estimate channel capacity faciliates also the investigation of

- mutual information between input and output (i.e. with an arbitrary distribtuion of input instead of optmising it as in channel capacity)
- probabilities of discrmination between input values based on output measurements

Those two aspects can be investigated using functions `mi_logreg_main()` and `prob_discr_pairwise()`, which assume very similar arguments as function `capacity_logreg_main()`.

24. Estimate mutual information between X and Y, assuming distribution of input as in experimental data (proportional to the number of observations in each class). Required parameters and syntax is analogous as in the previous codes

```
signal_name="signal"
response_name="output"
i_type="testing_mi_simpleGauss"
path_output_main=paste('output/',i_type,'/',sep="")
dir.create(path_output_main,recursive = TRUE)
tempoutput_mi  <- mi_logreg_main(dataRaw=tempdata, signal=signal_name,
response=response_name, output_path=path_output_main)
```

25. Compare it with channel capacity estimate

```
print(paste("Mutual Information: ",round(tempoutput_mi$mi,digits=2)," bits","     ",
            "Channel Capacity: ",round(tempoutput$cc,digits=2)," bits",sep=""))
```

```
## [1] "Mutual Information: 1.48 bits     Channel Capacity: 1.58 bits"
```

and probabilities of the optimal input distribution

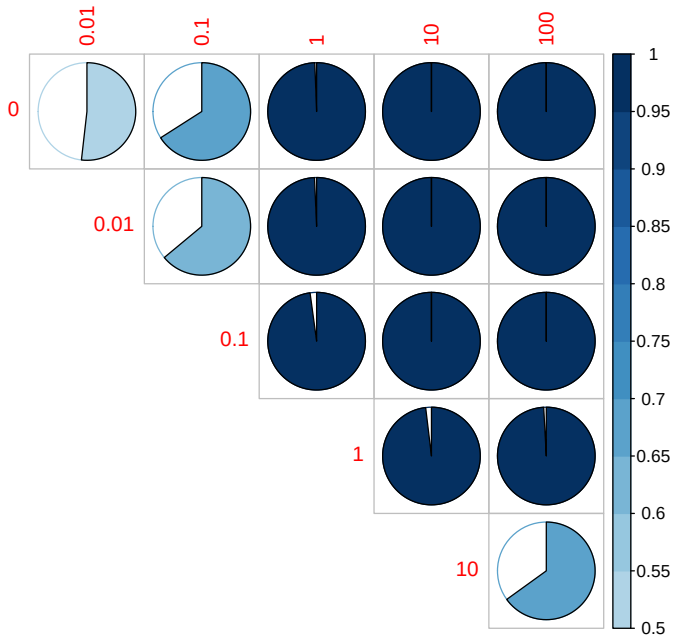
```
df=data.frame(rbind(round(tempoutput_mi$pinput,digits=2),
round(tempoutput$p_opt,digits=2)))
colnames(df)=paste("X = ",c(0,0.01,0.1,1,10,100),sep=" ")
df[["Input distribution"]]=c("uniform (mutual information)","optimal (channel capacity)")
df=df[,c(7,1:6)]
df
```

Input distribution	X = 0	X = 0.01	X = 0.1	X = 1	X = 10	X = 100
uniform (mutual information)	0.17	0.17	0.17	0.17	0.17	0.17
optimal (channel capacity)	0.20	0.02	0.13	0.31	0.14	0.21

26. Explore the probabilities of correct discrimination between pairs of input values

```
i_type="testing_probs_simpleGauss"
path_output_main=paste('output/',i_type,'/',sep="")
dir.create(path_output_main,recursive = TRUE)
tempoutput_probs  <- prob_discr_pairwise(dataRaw=tempdata, signal=signal_name,
response=response_name,  output_path=path_output_main)
```

what generates the following graph in the output directory, where each pie chart represents a probability of correct discriminating between a pair of corresponding input values as computed by fitting a logistic regression model



Those probabilities are also stored in `prob_matr` element of a returned list:

```
tempoutput_probs$prob_matr
```

```
##           0   0.01    0.1      1     10    100
## 0    1.0000 0.5180 0.6595 0.9940 1.0000 1.0000
## 0.01 0.5180 1.0000 0.6395 0.9930 1.0000 1.0000
## 0.1  0.6595 0.6395 1.0000 0.9785 1.0000 1.0000
## 1    0.9940 0.9930 0.9785 1.0000 0.9810 0.9915
## 10   1.0000 1.0000 1.0000 0.9810 1.0000 0.6505
## 100  1.0000 1.0000 1.0000 0.9915 0.6505 1.0000
```

7 Replicating results from the main paper and the supplementary information

Instructions on how to replicate results presented in the main paper and supplementary information are provided in the files `paper_MP.R` and `paper_SI.R`, respectively. Reproducing all figures in detail can last up to 24 hours (using a single core). Therefore, the scripts, by default, implement a simplified workflow, which requires ~4 hours of computations. Simplification of the analysis did not have a significant impact on the qualitative aspects of results. The full analysis can be obtained by uncommenting specified lines of the scripts.

7.1 Main Paper - NfkB analysis

The code provided in the file `paper_MP.R` implements computations that replicate Fig. 1 of the MP. The code is divided into three sections that should be run consecutively.

1) Preliminary - setting up packages and working environment. (lines 60-77)
2) Capacity - replicates Fig.1 B-C (lines 79-180)
3) Probabilities of correct discrimination - replicates Fig.1 D-E (lines 184-299)

In the default mode, i.e., 5 repetition of bootstrap, running Capacity section takes approx. 2 hours on a single core. Set number of cores for parallel processing in line 83. For graphs exactly like in the main paper, set line 84 to

```
analysis_type="long"
```

Calculating the probabilities of correct discrimination takes about 3 minutes. Please follow instructions and run code in `paper_MP.R` to replicate the results from the MP.

Execution of `paper_MP.R` requires installation of following packages: `ggplot2`, `gridExtra`, `mvtnorm`, and `corrplot`.

7.2 Supplementary Methods - validation and comparison

The script `paper_SI.R` implements codes to replicate examples presented in the Supplementary Information. Primarily, the validation of our method is shown for several simple examples and secondly we compare the performance of our method with the KNN method.

The code is divided into three sections that should be run consecutively

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 the 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 76. For graphs exactly like in SI, set line 77 to

```
analysis_type="long"
```

Execution of `paper_SI.R` requires installation of following packages: `ggplot2`, `gridExtra`, `mvtnorm`, `nloptr`, `FNN`, `DEoptimR`, `TDA` and `corrplot`.