

Statistical Learning based Estimation of the Mutual Information (SLEMI) - R package

Testing procedures

T. Jetka, K. Nienaltowski, T. Winarski, S. Błński, M. Komorowski

Version: August 9, 2018

Abstract

This document provides testing procedures for 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 single-cell signaling responses using SLEMI*, hereafter, referred to as *Main Paper* (MP). The Supplemental Information of the paper will be referred to as **SI**.

Contents

1	Requirements - Hardware	2
2	Requirements - Software	2
3	Installation	2
4	Key functions of the package	3
5	Input data	3
6	Numerical example	4
6.1	Initialization	4
6.2	Generation of the synthetic dataset	5
6.3	Calculation of the information capacity	6
6.4	Visualisation of results	6
6.5	Calculation of the mutual information	9
6.6	Calculation of the probabilities of correct discrimination	9
6.7	Diagnostics	10
7	Replicating results of the main paper and the supplementary information	15
7.1	Main paper - NfκB analysis	15
7.2	Supplementary Methods - validation and comparison	15

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 the **SI**, specifically of Fig. S1 and S3 (implemented in the script `paper_SI.R`; approx. running time: at least 2 hours);

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

Execution of the testing procedures has 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 in case 'devtools' package is 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. The use of the functions `mi_logreg_main()` and `prob_discr_pairwise()` is largely analogous and, therefore, described with more brevity.

5 Input data

Functions `mi_logreg_main()`, `capacity_logreg_main()`, `prob_discr_pairwise()` require data in form of the object `data.frame` with a specific structure of rows and columns. As described in detail in Section 1 of the **SI**, single cell responses y_j^i are assumed to be measured for a finite set of stimuli levels x_1, x_2, \dots, x_m . The responses y_j^i can be multidimensional. Usually, experimental dataset is represented as a table with rows and columns organized as below

input	output 1	output 2	output 3	...
$n_1 \left\{ \begin{array}{l} x_1 \\ \vdots \\ x_1 \end{array} \right.$	$y_1^1(1)$ \vdots $y_{n_1}^1(1)$	$y_1^1(2)$ \vdots $y_{n_1}^1(2)$	$y_1^1(3)$ \vdots $y_{n_1}^1(3)$	
$n_2 \left\{ \begin{array}{l} x_2 \\ \vdots \\ x_2 \end{array} \right.$	$y_1^2(1)$ \vdots $y_{n_2}^2(1)$	$y_1^2(2)$ \vdots $y_{n_2}^2(2)$	$y_1^2(3)$ \vdots $y_{n_2}^2(3)$	
\vdots	\vdots	\vdots	\vdots	...
$n_m \left\{ \begin{array}{l} x_m \\ \vdots \\ x_m \end{array} \right.$	$y_1^m(1)$ \vdots $y_{n_m}^m(1)$	$y_1^m(2)$ \vdots $y_{n_m}^m(2)$	$y_1^m(3)$ \vdots $y_{n_m}^m(3)$	

Therefore, the input data frame is expected to have the form represet by the above table. 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. The first column represents the input value. The subsequent columns represent the experimental measurements of the output.

This data structure is exemplified by the NfκB 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

The testing procedures are largely based on a numerical example, which involves generation of a synthetic data set. Precisely, we consider the conditional output probability, $P(Y|X = x_i)$, given by the log-normal distribution. In addition, 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 the capacity, mutual information and the probabilities of correct discrimination. In addition, the code exemplifies how to visualize results, and how to asses the quality of information capacity estimates i.e., perform diagnostics.

6.1 Initialization

1. Open `testing_procedures.R` script in RStudio (you can also follow comments in the script)
2. Set a working directory to where output and figures should 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 of computations

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

5. Set number of cores during computations

```
cores_num <- 8
```

6. Define whether to display plots with the results of the computations

```
display_plots <- TRUE
```

6.2 Generation of the synthetic dataset

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

7. Set parameter values

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

8. Create the output directory

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

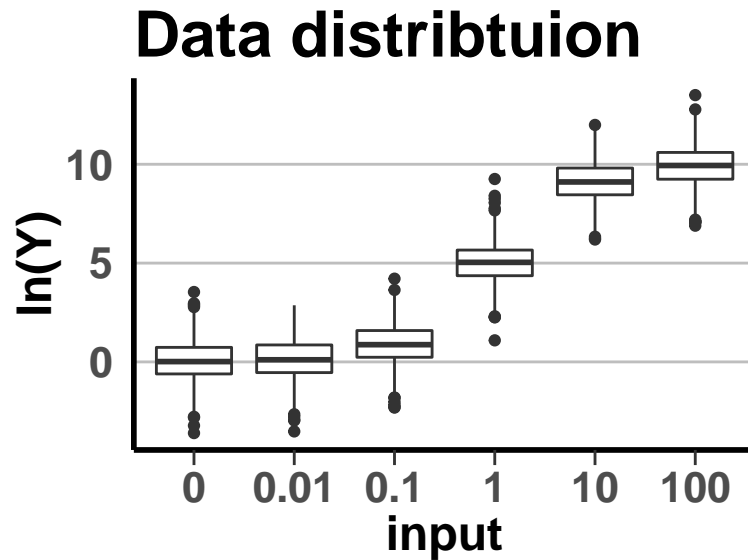
9. Generate the synthetic data

```
# 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 the distribution of the generated 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 default setting it expects four arguments: 1) `dataRaw` - data frame with experimental data, 2) `signal` - name of the input column, 3) `response` - names of the output columns and 4) `output_path` - a path to the output directory. It is executed by the following commands

11. Set required parameters for the algorithm

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

12. Estimate 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

Results can be accessed by either exploring the list returned by the function or by inspecting the graphs created in the output directory.

13. Displaying results of the estimation in the console - the value of estimated channel capacity and the optimal input distribution, together with the overall time of computations. These are also saved in object `output.rds` in the specified output directory.

```
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 = " " ))
```

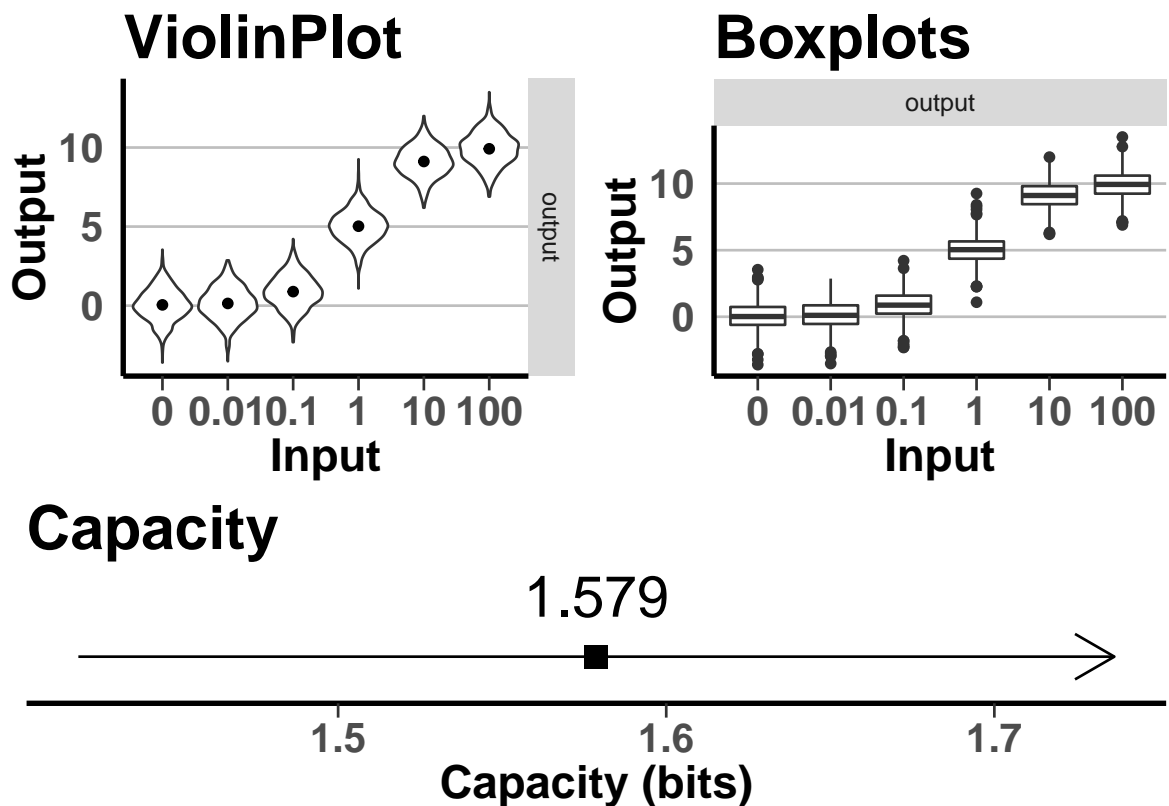
```
## [1] "Optimal input probabilities, x_i = 0 : 0.196414629740068"
## [2] "Optimal input probabilities, x_i = 0.01 : 0.0213765104176143"
## [3] "Optimal input probabilities, x_i = 0.1 : 0.12955652377174"
## [4] "Optimal input probabilities, x_i = 1 : 0.30577294887575"
## [5] "Optimal input probabilities, x_i = 10 : 0.141269433599551"
## [6] "Optimal input probabilities, x_i = 100 : 0.205609953595277"
```

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

```
## [1] "Time of computations (sec.): 6.89"
```

14. See pdf files in the folder 'output_path/' for the visualization of the data and of the capacity estimation. In the "MainPlot.pdf" a summary of the performed analysis is presented: mean input-output relation; distributions of output and channel capacity (see below). These 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]])
}
```



15. Change the size of generated graphs. The commands below will repeat the computations of the step 12

and the results will be saved in plots of different sizes, which is set by the parameters `plot_width` and `plot_height`

```
# specify parameters `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 computations of the step 12 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. Re-run step 12 with a minimal output by setting `graphs`, `dataout`, `model_out` to `FALSE`. These settings prevents from creating graphs, as well as does not include data and regression model in the returned list. This setting is useful 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,
  dataout = data_save,
  model_out = model_save
)
```


6.5 Calculation of the mutual information

The use of the function `mi_logreg_main()` is analogous to the use of the function `capacity_logreg_main()` with the exception that the input distribution must be specified by the user. A numeric vector with values of respective probabilities should be passed by an additional argument of `mi_logreg_main()` function - `pinput`.

18. The code below computes mutual information assuming uniform distribution of input values

```
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,
pinput=rep(1/6,6))
```

19. Mutual information can be compared with the channel capacity by executing the following

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

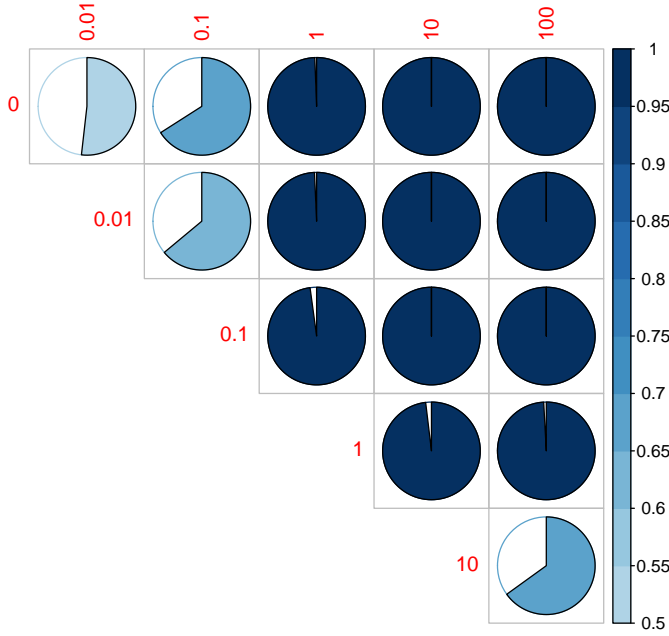
```
## [1] "Mutual Information: 1.48 bits      Channel Capacity: 1.58 bits"
```

6.6 Calculation of the probabilities of correct discrimination

20. The probabilities of correct discrimination between pairs of input values are computed as follows

```
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)
```

The above generates the following graph in the output directory. Pie charts represent probabilities of correct discrimination as described in the Section 4.5 of the SI.



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

6.7 Diagnostics

In addition to the sole calculation of the information capacity, the function `capacity_logreg_main()` can also be used to assess accuracy of the channel capacity estimates resulting from potentially insufficient sample size and potential over-fitting of the regression model. Two tests are implemented. Precisely, the function can perform

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

In order to perform diagnostic tests, that by default are turned off, user must set the value of the input argument

- `testing = TRUE` (default=FALSE)

In addition, settings of the diagnostic test can be altered with the following parameters

- 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

In the example considered above, the diagnostic tests can be performed by executing the following

21. 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)
```

22. Run the estimation of the information capacity with full diagnostics by using the above parameters and setting 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
)
```

23. Inspect the results of diagnostic tests. These are stored 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)"
```

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

```
## [1] "Time of computations (sec.): 42.52"
```

24. For each diagnostic test, the function provides also 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

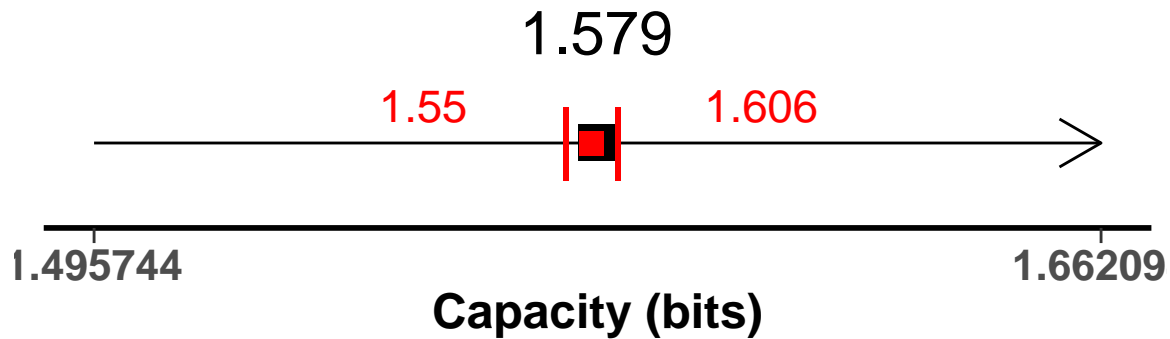
```
print(paste("P-values:",
tempoutput_diag$testing_pv$bootstrap[1],
" (left-tailed); ",
tempoutput_diag$testing_pv$bootstrap[2],
" (right-tailed) ",
sep = ""))
```

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

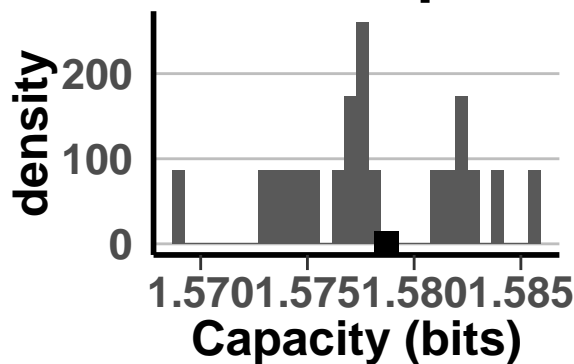
25. Visualize results of the diagnostic test. Histogram of calculated capacities are provided in the file capacity.pdf in the output directory. The graph is also obtainable from the returned list by a command

```
if(display_plots){
grid.arrange(tempoutput_diag$logGraphs[[7]])
}
```

Capacity



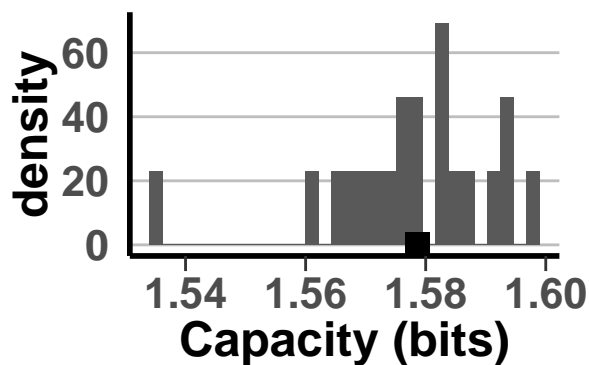
Bootstrap



PV-left: 0.65

PV-right: 0.35

TrainTest



PV-left: 0.55

PV-right: 0.45

The top diagram shows the value of the capacity estimate (in black) obtained from the complete dataset and the mean value of bootstrap repetitions with indicated \pm standard deviation (in red). Plots that follow show histograms of calculated capacities for different diagnostic regimes. The black dot represents the estimate of the channel capacity based on the complete dataset. In addition, corresponding empirical p-values of both tests (left- and right-sided) are calculated to assess the randomness of obtained results (PV in the plots).

A reliable estimation of the information capacity should yield the following results of the bootstrap and overfitting tests.

- The bootstrap test should yield distribution of the capacity estimates with small variance. In addition, the capacity estimated based on the complete dataset should not be an outlier ($p\text{-value} > 0.05$). Otherwise, it would indicate that the sample size is too low for an accurate estimation of the channel capacity.

- The over-fitting test should provide similar results. The capacity estimate obtained based on the complete dataset should lie within the distribution of capacities generated in the test. In the opposite case, it could mean that the logistic regression model does not fully grasp the essential aspects of input-output dependencies in the data.

7 Replicating results of 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 - NfκB analysis

The code provided in the file `paper_MP.R` implements computations to replicate Fig. 1 of the MP. The code is divided into three sections that should be run consecutively. Execution of `paper_MP.R` requires installation of following packages: `ggplot2`, `gridExtra`, `mvtnorm`, and `corrplot`.

- 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 repetitions 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, follow instructions and run the corresponding section of the code in `paper_MP.R`.

7.2 Supplementary Methods - validation and comparison

The script `paper_SI.R` implements codes to replicate examples presented in the SI. 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. Execution of `paper_SI.R` requires installation of following packages: `ggplot2`, `gridExtra`, `mvtnorm`, `nloptr`, `FNN`, `DEoptimR`, `TDA` and `corrplot`.

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"
```