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

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

July 22, 2018

Contents

This document provides detailed procedures for testing the SLEMI package, description of numerical test examples as well as of the experimental data set from manuscript

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

Following tutorial is partly redundant with User Manual and was prepared to allow validation of the package in the review process of the manuscript. R scripts for testing are provided in the folder 'testing_procedures' of the package.

The testing procedure is divided into three components, with approximate running time on a single core given in brackets,

1. Quick guide for setting up the package (approx. time: 5-20 minutes),
2. Numerical test examples (approx. time: 1 hour).

Codes are provided in the file `testing_procedures.R`,

3. Replication of the analysis of the NFkB data set presented in the main paper (approx. time: at least 4 hours).

Codes are provided in files `paper_MP.R` and `paper_SI.R`.

Running the following testing procedure requires the following hardware an 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 a base installation of R, SLEMI requires several additional R packages, as described in the User Manual. Are these packages are not found, they will be installed automatically by the installation procedure described below.

3 Installation

The package can be directly installed from GitHub. For installation, 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/SLEMI")
```

4 Key functions of the package

Estimation of the channel capacity is the core functionality of the package and is performed by the function`capacity_logreg_main()`. Estimation of the mutual information is performed by the function `mi_logreg_main()`, whereas probabilities of correct discrimination by the function `prob_discr_pairwise()`. Each of these function takes, as the input, experimental data of the form described in Section 1.1 of the Supplementary Information. The testing procedures below are focused on the function `capacity_logreg_main()` which is the main contribution of this package. The use of the function `mi_logreg_main()` is analogous, as described in the user manual. The use of the function `prob_discr_pairwise()` that serves to calculate probabilities of correct discrimination is presented in one of the sections below pertaining to replication of the Figures of the main paper.

5 Input data

Each of the above function takes, as the input, experimental data of the form described in Section 1.1 of the Supplementary Information. Precisely, single cell responses are assumed to follow a probability distribution $P(Y|X = x_i)$ and, hence, the collected dataset can be conceptually represented as

$$P(Y|X = x_i) \quad \sim \quad \left(y^i_1,\ y^i_2,\ \ldots,\ y^i_{n_i}\right), \tag{1}$$

Therefore, an experimental dataset is typically represented as a table

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

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

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

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

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

signal	respons	e_0 respons	e_3 respons	e_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

It can be shown 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))
```

Each of the numerical examples below takes as the input a data set of the form described above.

6 Numerical example 1

To demonstrate how to use SLEMI, we will generate a synthetic dataset of a channel, for which the conditional output distribution, $Y|X$, is Log-normal. 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, $Y|X = x$, is a one-dimensional Log-normal distribution $\exp\{\mathcal{N}(10 \cdot \frac{x}{1+x}, 1)\}$;
iii) for each x, there are 1000 observations.

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

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 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.

7. 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
```

8. Create output directory

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

9. Generating synthetic data

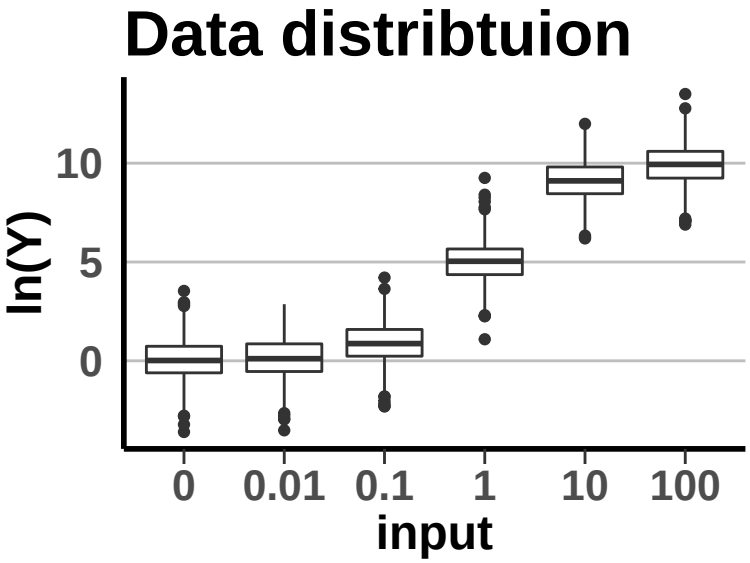
```
# concentration of input; spans from 0 to saturation.
xx <-
  signif(
c(0,
  exp(
    seq(
      from = log(0.01),
      to = log(100),
      length.out = input_num-1))),
digits = 2)
# mean of the dose-response relation; Michaelis-Menten assumed
example_means <- 10*(xx/(1+xx))
example_sds <- rep(dist_sd,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



6.3 Running algorithm

The estimation of channel capacity can be performed by using `capacity_logreg_main` function. In the most 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 can be obtained by

11. Set required parameters for the algorithm

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

12. Estimate 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 visualisation implemented within the package (graphs created in output directory)

13. 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.): 5.31"
```

14. 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]))
```

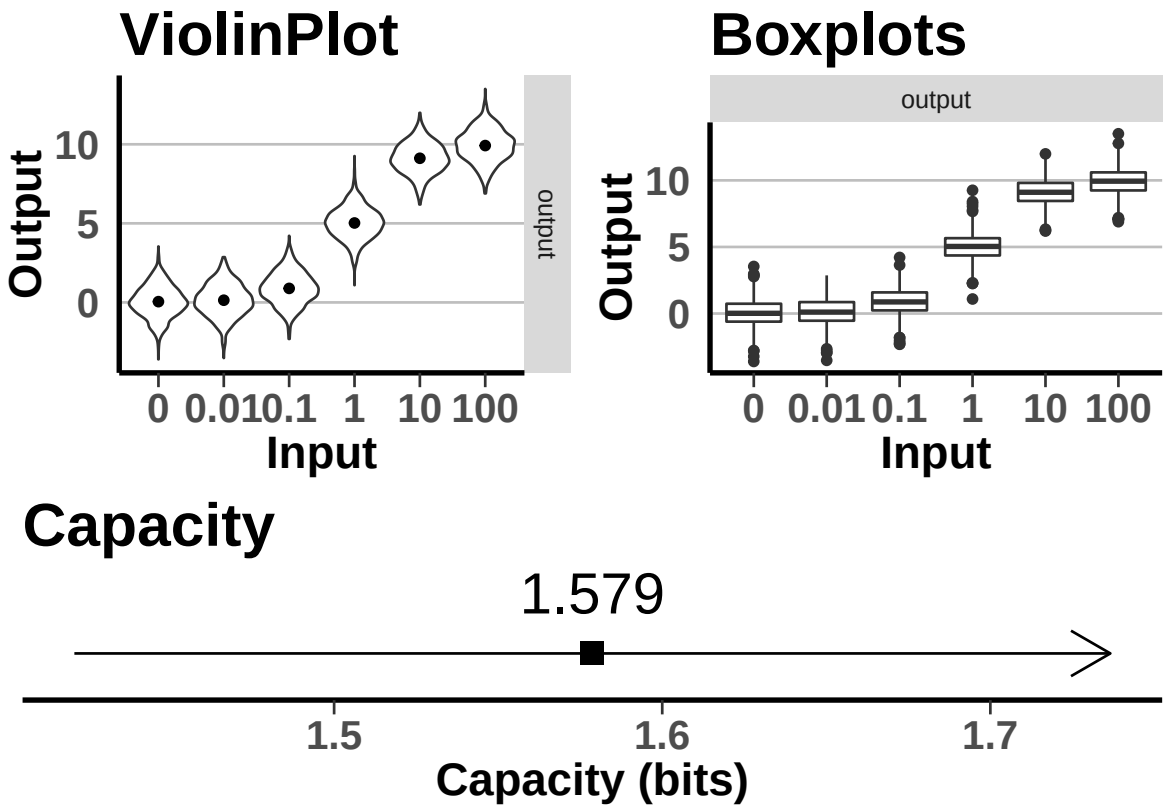
```
## [1] "Accuracy of classification assertion: TRUE"
```

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

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

15. 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). 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. 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
)
```

17. 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
)
```

18. 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
)
```

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 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, those diagnostic 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. 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
)
```

21. 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)"
```

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

```
## [1] "Time of computations (sec.): 38.12"
```

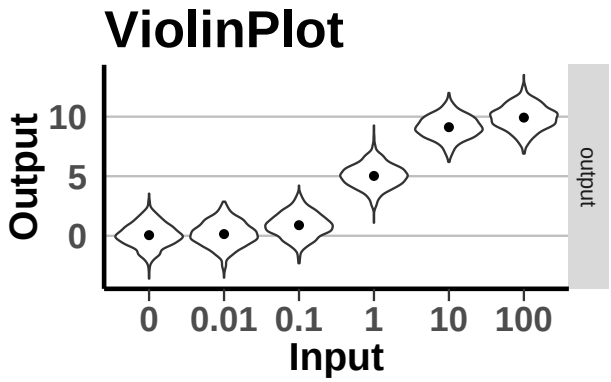
22. See the visualisation in the output directory - MainPlot.pdf. For each diagnostic test, there is a corresponding histogram of calculated capacities. This graph is also obtainable from the returned list by a command

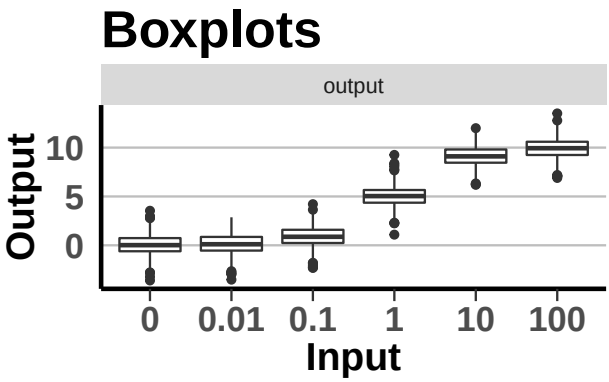
```
if(display_plots){
  grid.arrange(tempoutput$logGraphs[[9]])
}
```

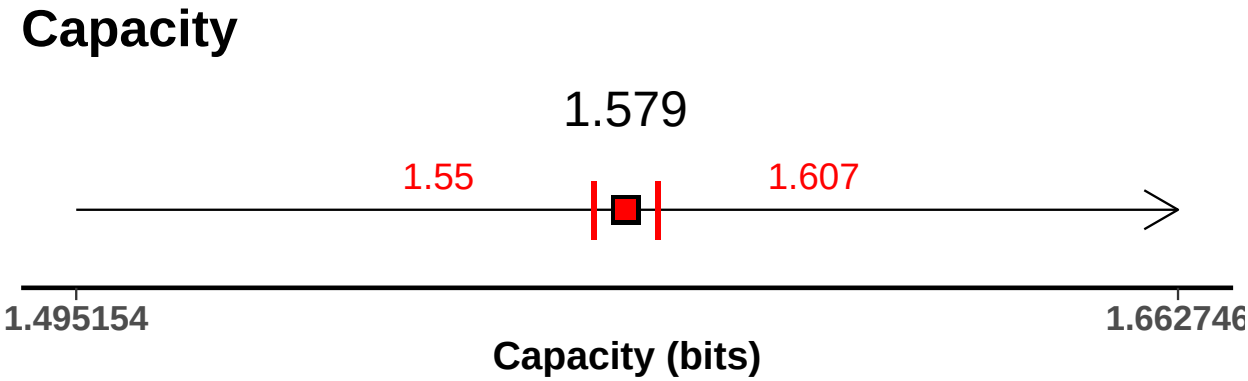
23. For each diagnostic test, we provide left- and right-tailed empirical p-values of the obtained channel capacity. They indicated if the regime of data bootstrapping or dividing data into training and testing sample influence the calculation of capacity in a significant way. A small p-value in any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

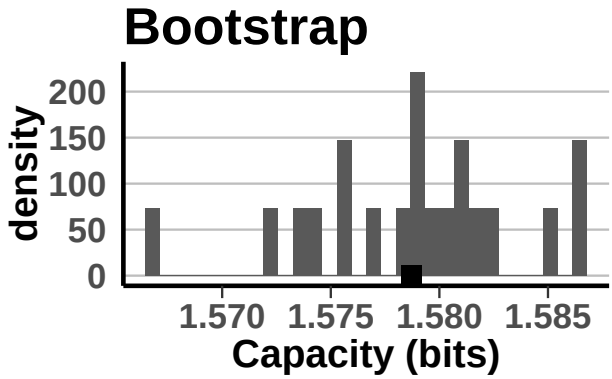
```
print(paste("P-values:",
            tempoutput$testing_pv$bootstrap[1],
            " (left-tailed); ",
            tempoutput$testing_pv$bootstrap[2],
            " (right-tailed) ",
            sep = ""))
```

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



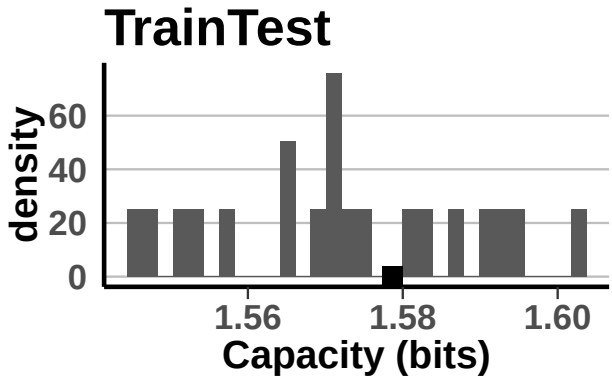






PV−left: 0.4

PV−right: 0.6



PV−left: 0.65

PV−right: 0.35

7 Replicating results from the main paper and the supplementary information

Instructions how to replicate results presented in the i) main paper (MP) and ii) supplementary information (SI) are provided in files `paper_MP.R` and `paper_SI.R`, respectively. Reproducing all figures in detail 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 ~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.

7.1 Main Paper - NfkB analysis

Here, we replicate Fig. 1 of the MP. Single-cell time series in a data.frame object are attached to the package and can be accessed using `data_nfkb` variable.

The code is given in `paper_MP.R` file and 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 A-B (lines 79-180)
3) Probabilities of discrimination - replicates Fig.1 C-D (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"
```

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: `ggplot2`, `gridExtra`, `mvtnorm`, and `corrplot`.

7.2 Supplementary Methods - validation and comparison

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

Therefore 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 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"
```

Please follow instructions and run codes in `paper_SI.R` to replicate the results from SI. Installation of additional packages is needed: `ggplot2`, `gridExtra`, `mvtnorm`, `nloptr`, `FNN`, `DEoptimR`, `TDA` and `corrplot`.