SUITOR: selecting the number of mutational signatures through cross-validation

User Guide

DongHyuk Lee[1], William Wheeler[2] and Bin Zhu[3]

[1] Department of Statistics, Pusan National University, Busan, Korea

[2] Information Management Services (IMS), Inc. Rockville, MD, USA

[3] Division of Cancer Epidemiology and Genetics, National Cancer Institute, National Institutes of Health, Bethesda, MD, USA

Version 1.0.1

March 5, 2022

Contents

1 Introduction

For the *de novo* mutational signature analysis, selecting the correct number of signatures is the crucial starting point; indeed, it would influence each the downstream step, including extraction of signature profiles, estimation of signature activities and classification of tumors based on the estimated activities. Here we present the **R** package *SUITOR*, an unsupervised cross-validation tool to select the optimal number of signatures. This tutorial introduces the usage of two main functions `suitor()` and `suitor_Extract_WH()` to select the number of signatures and to estimate signature profiles and activities, respectively.

2 Installation

To install from Github directly, one can use the devtools **R** package:

```
if (!requireNamespace("devtools", quietly = TRUE))
        install.packages("devtools")
devtools::install_github("binzhulab/SUITOR/source")
```

Alternatively, SUITOR_1.0.1.tar.gz (for Unix) or SUITOR_1.0.1.zip (for Windows, R version >= 4.0) from the Github page[1] is available to be installed, using the following commands:

```
install.packages("SUITOR_1.0.1.tar.gz", repose = NULL, type = "source")
install.packages("SUITOR_1.0.1.zip", repose = NULL, type = "win.binary")
```

Once installed, *SUITOR* can be loaded on **R** by calling

```
> library(SUITOR)
Loading required package: doParallel
Loading required package: foreach
Loading required package: iterators
```

[1] https://github.com/binzhulab/SUITOR

```
Loading required package: parallel
Loading required package: ggplot2
```

3 Example data

For illustrative purpose, we simulated a 96×300 mutation type matrix which contains 300 tumors with respect to 96 single base substitution types. Each element of the matrix is generated from the Poisson distribution (`rpois()` function) with the mean corresponding to each element of **WH**, where **W** is the true signature profile matrix of size 96×8 for 8 signatures and **H** is the signature activity matrix of size 8×300. Specifically, we used the profile of eight COSMIC signatures 4, 6, 7a, 9, 17b, 22, 26, 39 (ref)[2] for **W**. **H** was generated from a uniform distribution between 0 and 100 and some randomly chosen elements of **H** were set to zero to mimic the real data.

```
> data(SimData, package = "SUITOR")
> dim(SimData)
[1]  96 300
> SimData[1:6, 1:6]
          X1  X2  X3  X4  X5  X6
A[C>A]A    0   1   1   6   0   4
A[C>A]C    0   0   0   1   2   4
A[C>A]G    0   0   0   2   1   2
A[C>A]T    1   2   0   3   2   3
C[C>A]A    0   0   0   8   1   7
C[C>A]C    2   0   0  17   0  13
```

4 Selection of the number of mutational signatures

The main function `suitor(data, op)` is to select the number of mutational signatures based on cross-validation. It has two arguments described in Sections

[2] `https://cancer.sanger.ac.uk/cosmic/signatures/SBS`

4.1 and 4.2

4.1 Input data

The first argument of the function `suitor()` is `data` for the mutation type matrix. It could be an **R** `dataframe` or `matrix` of which elements are non-negative counts. Each column of `data` corresponds to a tumor (or sample) while its row represents a mutation type. Although selection of the number of signatures is independent to the order of mutation type, we specify the order of mutation type according to the COSMIC database (SBS signature[3]) for extracting signature profiles using `suitor_Extract_WH()` after estimating the optimal number of signatures (or called rank in this user guide).

4.2 Options

Since SUITOR is based on cross-validation and the Expectation Conditional Maximization (ECM) algorithm, it is necessary to set a list of tuning parameters which control the fitting process.

Table 1: List of options for the function `suitor()`

Name	Description	Default Value
`min.value`	Minimum value of matrix before factorizing	1e-4
`min.rank`	Minimum rank	1
`max.rank`	Maximum rank	10
`k.fold`	Number of folds	10
`em.eps`	ECM algorithm stopping tolerance	1e-5
`max.iter`	Maximum number of iterations in ECM algorithm	2000
`n.seeds`	Number of seeds (starting points)	30
`n.cores`	Number of cores to use	1
`get.summary`	0 or 1 to create summary results	1
`plot`	0 or 1 to produce an error plot	1
`print`	0 or 1 to print info (0=no printing)	1
`seeds`	Vector of seeds (takes precedence over n.seeds)	NULL

[3]`https://cancer.sanger.ac.uk/signatures/sbs/`

The `min.value` is a small number added to the `data` matrix for improving stable computation of non-negative matrix factorization. For a given number of signatures or called rank r (`min.rank` $\leq r \leq$ `max.rank`), the `data` matrix is divided into `k.fold` parts for cross-validation. The default value of the maximal rank `max.rank` is 10 and be changed depending on the cancer type. The default value of the number of fold K (`k.fold`) is 10 and can be modified depending on computer resources.

Since the ECM algorithm may converge to a local saddle point, SUITOR tries multiple initial values for $\mathbf{W}_0$ and $\mathbf{H}_0$ through the number of seeds (`n.seeds`) or a vector of seeds (`seeds`). For example, when setting the number of seeds `n.seeds = 30`, 30 seeds are randomly generated; setting a vector of seeds as `seeds = 1:30` defines the seeds as $1, 2, \cdots, 30$. Note that `seeds` takes precedence over `n.seeds`; in fact, if `seeds = 1:30` is used, `n.seeds = 30` will be ignored. Although the default `n.seeds` is set to 30, it can be increased depending on the size of the `data` matrix and/or computational resources.

For the ECM algorithm, the default value of the maximal iteration `max.iter` is 2000. If the maximal iteration is reached, the function would produce a warning message. In general, we recommend a two-stage approach where the user would run `suitor()` with the default option first and then narrow down the set of plausible ranks (`min.rank` $\leq r \leq$ `max.rank`) with more seeds (`seeds`) and a larger number of maximal iteration (`max.iter`) if necessary.

To ease the computational burden, SUITOR supports parallel computing for Windows and UNIX machines by specifying `n.cores` greater than 1. To check whether the parallel computing is available,

```
> library(SUITOR)
> detectCores()
```

If `detectCores()` returns a value greater than 1, that return value may be used for `n.cores`.

4.3 Running `suitor()` with the default option

We demonstrate an example of the simulated data (`SimData`) in Section 3, using the default option (note that it may take a while).

```
> re <- suitor(SimData)
> str(re)
List of 4
$ rank        : num 8
$ summary     :'data.frame':        20 obs. of  13 variables:
..$ Rank   : num [1:20] 1 1 2 2 3 3 4 4 5 5 ...
..$ Type   : chr [1:20] "Train" "Test" "Train" "Test" ...
..$ MSErr  : num [1:20] 1.066 1.086 0.995 1.052 0.923 ...
..$ fold1  : num [1:20] 29511 3335 25651 3218 22041 ...
..$ fold2  : num [1:20] 29642 3200 25747 3093 22119 ...
..$ fold3  : num [1:20] 29595 3247 25788 3083 22257 ...
..$ fold4  : num [1:20] 29395 3452 25638 3241 22075 ...
..$ fold5  : num [1:20] 29355 3484 25591 3295 22090 ...
..$ fold6  : num [1:20] 29486 3357 25665 3167 22090 ...
..$ fold7  : num [1:20] 29416 3427 25674 3080 22100 ...
..$ fold8  : num [1:20] 29363 3485 25546 3216 22012 ...
..$ fold9  : num [1:20] 29388 3454 25643 3224 22089 ...
..$ fold10: num [1:20] 29362 3496 25571 3278 21968 ...
$ all.results: num [1:3000, 1:6] 1 2 3 4 5 6 7 8 9 10 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : NULL
.. ..$ : chr [1:6] "Rank" "k" "Seed" "Error.Train" ...
$ op          :List of 18
..$ min.rank    : num 1
..$ max.rank    : num 10
..$ k.fold      : num 10
..$ n.seeds     : num 30
..$ max.iter    : num 2000
..$ em.eps      : num 1e-05
..$ plot        : logi TRUE
..$ print       : num 1
..$ min.value   : num 1e-04
..$ get.summary: num 1
..$ n.cores     : num 1
..$ kfold.vec   : int [1:10] 1 2 3 4 5 6 7 8 9 10
..$ seeds       : num [1:30] 61054410 87830287 63003007 1905157 860437 ...
..$ parMat      : num [1:3000, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
..$ parStart    : num 1
..$ parEnd      : num 3000
..$ algorithm   : num 1
..$ type        : chr "PSOCK"
```

By default summary options (`get.summary` with 1), `suitor()` function returns a list, including the selected optimal rank (`re$rank`), a summary matrix (`re$summary`) where cross validation errors are tabulated, as well as the detailed results (`re$all.results`), and options (`re$op`) used by the `suitor()` function.

The following is a detailed list of the `suitor()` outputs of running `suitor()` function for the simulated data. Specifically, `re$all.results` contains the training (`Error.Train`) and testing (`Error.Test`) errors, the total number of ECM updates (`EM.niter`), fold and seed. In `re$summary` table, the `MSErr` column contains the evaluated overall prediction errors for training and test set.

```
> head(re$all.results)
     Rank k     Seed Error.Train Error.Test EM.niter
[1,]    1 1 61054410    29511.08   3335.227        3
[2,]    2 1 61054410    25650.62   3218.025       17
[3,]    3 1 61054410    22041.99   2811.022       28
[4,]    4 1 61054410    19560.17   2538.931       29
[5,]    5 1 61054410    16654.08   2238.370       35
[6,]    6 1 61054410    14291.18   1925.935       40
> re$summary[1:8, 1:7]
  Rank  Type     MSErr     fold1     fold2     fold3     fold4
1    1 Train 1.0659437 29511.082 29641.695 29595.250 29394.926
2    1  Test 1.0855287  3335.227  3199.712  3246.946  3452.472
3    2 Train 0.9948060 25650.619 25747.317 25788.391 25638.213
4    2  Test 1.0523606  3218.024  3092.761  3082.739  3240.887
5    3 Train 0.9230445 22040.905 22119.229 22257.146 22074.585
6    3  Test 0.9824743  2810.477  2738.421  2609.655  2784.612
7    4 Train 0.8590084 19063.701 19164.172 19342.943 19061.705
8    4  Test 0.9207998  2514.880  2375.038  2198.266  2529.894
> summary(re$all.results[, "EM.niter"])
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
3.00    24.00    37.00    40.86    51.00   450.00
> re$rank
[1] 8
```

The number of ECM iteration can be extracted to monitor its convergence. In this example, the median iteration count is 36 (because it was initialized at 1) across all possible combinations of rank, fold and seed. Since the maximum iteration count is 449, the default value of `max.iter` (2000) was not exceeded for the fitting process. Finally, the selected optimal rank is shown by `re$rank`, which is the same as the true number of signatures to generate `SimData` in Section 3. The default `plot` option (set to 1) generates the cross-validation error plot (Figure 1) based on the `MSErr` column.

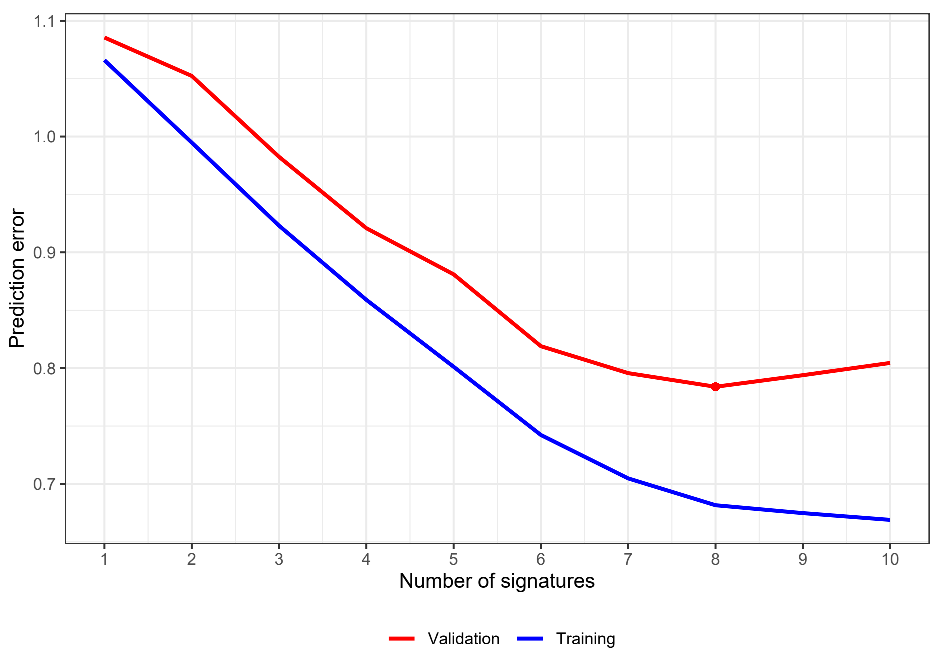

Figure 1: Prediction errors of SUITOR for the training and validation sets of `SimData` in Section 3. The red dot denotes the number of signatures with the minimal prediction error in the validation set. The x-axis corresponds to the values *r* between (`min.rank` $\leq r \leq$ `max.rank`) while the y-axis corresponds to numbers in `MSErr` column in `re$summary` table.

4.4 Running `suitor()` with different option values

From the previous section, the optimal rank is estimated with the default option. To try more initial values, we can change the possible range of ranks from the default [1,10] to [5,13] and increase the number of initial values from 30 to 50. Specifically, we run the function `suitor()` with different option values discussed in Section 4.2, we create a list of options as follows. Note that the name of the option `list` should be matched with elements in Table 1.

```
> OP <- list(min.rank = 5, max.rank = 13, k.fold = 5, n.seeds = 50,
             get.summary = 0)
> re2 <- suitor(data = SimData, op = OP)
> str(re2)
List of 2
$ all.results: num [1:1350, 1:6] 5 6 7 8 9 10 11 12 13 5 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : NULL
.. ..$ : chr [1:6] "Rank" "k" "Seed" "Error.Train" ...
$ op          :List of 19
..$ min.rank   : num 5
..$ max.rank   : num 13
..$ k.fold     : num 5
..$ n.seed     : num 50
..$ get.summary: num 0
..$ n.seeds    : num 30
..$ max.iter   : num 2000
..$ em.eps     : num 1e-05
..$ plot       : logi TRUE
..$ print      : num 1
..$ min.value  : num 1e-04
..$ n.cores    : num 1
..$ kfold.vec  : int [1:5] 1 2 3 4 5
..$ seeds      : num [1:30] 27083050 78500458 69888895 69806671 32878896 ...
..$ parMat     : num [1:1350, 1:3] 5 6 7 8 9 10 11 12 13 5 ...
..$ parStart   : num 1
..$ parEnd     : num 1350
..$ algorithm  : num 1
..$ type       : chr "PSOCK"
```

If `get.summary` is set to 0, `suitor()` only produce a matrix containing all possible results. As shown below, one can use the `getSummary(obj, NC, NR)` function to compute the estimated optimal rank (`Summary1$rank`), where `obj` is the matrix containing all results from `suitor()`, `NC` and `NR` are the numbers of columns and rows respectively in the input `data` for `suitor()`. Please note that `NR` has a default value of 96 for signature analysis of single base substitution.

```
> Summary2 <- getSummary(re2$all.results, ncol(SimData))
> str(Summary2)
List of 3
$ rank        : num 8
$ summary     :'data.frame':        18 obs. of  8 variables:
..$ Rank : num [1:18] 5 5 6 6 7 7 8 8 9 9 ...
..$ Type : chr [1:18] "Train" "Test" "Train" "Test" ...
..$ MSErr: num [1:18] 0.798 0.883 0.738 0.823 0.7 ...
..$ fold1: num [1:18] 14636 4463 12550 3944 11292 ...
..$ fold2: num [1:18] 14722 4434 12609 3851 11352 ...
..$ fold3: num [1:18] 14750 4400 12591 3887 11373 ...
..$ fold4: num [1:18] 14626 4622 12562 3889 11264 ...
..$ fold5: num [1:18] 14564 4549 12504 3940 11219 ...
$ all.results: num [1:1350, 1:6] 5 6 7 8 9 10 11 12 13 5 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : NULL
.. ..$ : chr [1:6] "Rank" "k" "Seed" "Error.Train" ...
> summary(re2$all.results[,"EM.niter"])
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
44.0    67.0   135.0   166.7   248.0   697.0
> Summary2$rank
[1] 8
> plotErrors(Summary2$summary)
```

Again the default value of `max.iter` (2000) is large enough since the maximum iteration count is 696. We confirm that the estimated rank (`Summary2$rank`) here agrees with the true number of signatures as well. In addition, the `plotErrors()` function could be used to draw a cross validation error plot as in Figure 1 (results not shown).

5 Extracting the signature profiles and activities

Once the optimal number of signature or called rank is estimated by `suitor()`, we can extract the signature profiles $\widehat{\mathbf{W}}$ and activities $\widehat{\mathbf{H}}$ with the function `suitor_extract_WH(data, rank, op)`. As in the `suitor()` function, the input mutation type matrix `data` is a data frame or matrix whose elements are non-negative counts. The non-negative integer `rank` is the number of mutational signatures to be extracted. The option values are summarized in the following Table 2 and they can be used in the same manner as `suitor()`.

Table 2: List of options for the function `suitor_extract_WH()`

Name	Description	Default Value
`min.value`	Minimum value of matrix before factorizing	1e-4
`n.cores`	Number of cores to use for parallel computing	1
`n.seeds`	Number of seeds (starting points)	30
`print`	0-1 to print info (0=no printing)	1
`seeds`	Vector of seeds (takes precedence over n.seeds)	NULL

```
> re <- suitor(SimData)
> re$rank
[1] 8
> Extract <- suitor_extract_WH(SimData, re$rank)
seed 30
> head(Extract$W)
          denovo A     denovo B     denovo C     denovo D    denovo E
A[C>A]A 0.04983920 6.864379e-20 9.210142e-04 5.021053e-03 0.010287996
A[C>A]C 0.03619912 2.909597e-03 6.688993e-04 4.062685e-07 0.007367939
A[C>A]G 0.01907898 1.760262e-03 5.568929e-20 5.822657e-20 0.001869262
A[C>A]T 0.03533304 8.483854e-04 5.568929e-20 2.024296e-03 0.008467782
C[C>A]A 0.08706398 7.221585e-03 5.568929e-20 1.323915e-03 0.010022022
C[C>A]C 0.10698556 6.864379e-20 5.568929e-20 2.885607e-12 0.003802372
            denovo F     denovo G     denovo H
A[C>A]A 6.236367e-20 1.121825e-03 5.495331e-20
A[C>A]C 6.236367e-20 1.053258e-03 9.980362e-04
A[C>A]G 6.236367e-20 2.401631e-07 1.528276e-10
```

```
A[C>A]T 4.031693e-07 5.511649e-20 2.594604e-03
C[C>A]A 1.427441e-04 1.903321e-03 5.495331e-20
C[C>A]C 1.287044e-06 5.511649e-20 1.492892e-03

> Extract$H[,1:3]
                 [,1]         [,2]         [,3]
denovo A 2.803011e+00 2.050670e+00 7.926640e-12
denovo B 1.508570e+01 2.105153e+00 1.815314e+01
denovo C 9.011722e+01 1.227970e+01 2.238425e+00
denovo D 8.467579e-13 6.883381e+01 5.620215e+01
denovo E 2.281342e+01 6.971187e-13 2.699901e+00
denovo F 4.592150e+01 4.630038e+01 1.337674e-04
denovo G 1.431012e+01 2.083340e+01 8.982586e+01
denovo H 8.595290e+01 1.260151e+01 1.788483e+01
```

`Extract$W` and `Extract$H` are estimated matrices for the profile $\widehat{\mathbf{W}}$ and the activity $\widehat{\mathbf{H}}$, respectively.

6 BRCA440 example

In this section, we describe the application of *SUITOR* package to the Sanger breast cancer study. The **R** package *SparseSignatures* (Ramazzotti et al., 2021) provides the dataset (Nik-Zainal et al., 2016). Note that we have analyzed 440 of 560 tumors as the validation dataset in the manuscript, removing 110 tumors included in The Pan-Cancer Analysis of Whole Genomes (PCAWG) study. To reproduce the results in the manuscript, we specified the option values for `suitor()` as below. The following codes were executed in a Windows desktop PC with Intel(R) Core(TM) i5-10500 @ 3.10GHz CPU and 8 GB of 3200MHz DDR4 RAM.

```
> V <- read.csv("BRCA440.csv", row.names = 1)
> re <- suitor(data = V, op = list(max.rank = 15, seeds = 1:300,
                 n.cores = 12))
Warning message:
In suitor(data = V, op = list(max.rank = 15, seeds = 1:300, n.cores = 12,   :
The maximal iteration has been reached and the algorithm has not converged;
please increase the maximal iteration R function parameter: max.iter.
```

```
> summary(re$all.results[,"EM.niter"])
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
3.0     52.0   918.0   953.2  1819.0  2001.0
> re$rank
[1] 12
```

It took quite a while to estimate the optimal rank primarily because of a large number (300) of initial values. We set 300 initial values for the research purpose but found that a smaller number of initial values would result in the similar result. Thus in practice, it can be reduced to 50 or 100 and it is feasible for a desktop computer to run `suitor()` for a couple of hundreds tumors. In this example, `suitor()` produces a warning message because ECM algorithm reached the maximum number of iterations (`max.iter`).

```
> re2 <- suitor(data = V, op = list(min.rank = 10, max.rank = 15,
              seeds = 1:300, n.cores = 12, max.iter = 5000))
> summary(re2$all.results[,"EM.niter"])
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
627     1596    1953    1971    2310    4222
> re2$rank
[1] 12
```

To address this issue, we next run `suitor()` with increased `max.iter` value (to 5000), with which all the ECM algorithms converged. The estimated optimal ranks for both cases are with the same (Figure 2).

Once we estimate the optimal rank (`re$rank` or `re2$rank`), the next step is to extract signature profiles and activities by using the `suitor_extract_WH()` function with the estimated rank. Here we show part of the estimated profile matrix (`Ext$W`).

```
> Ext <- suitor_extract_WH(data = V, rank = re$rank,
                          op = list(seeds = 1:300, n.cores = 12))
> head(Ext$W)
denovo A     denovo B     denovo C     denovo D      denovo E
A[C>A]A 0.015398608 0.0013408398 0.012421276 0.004673429 5.043659e-04
```

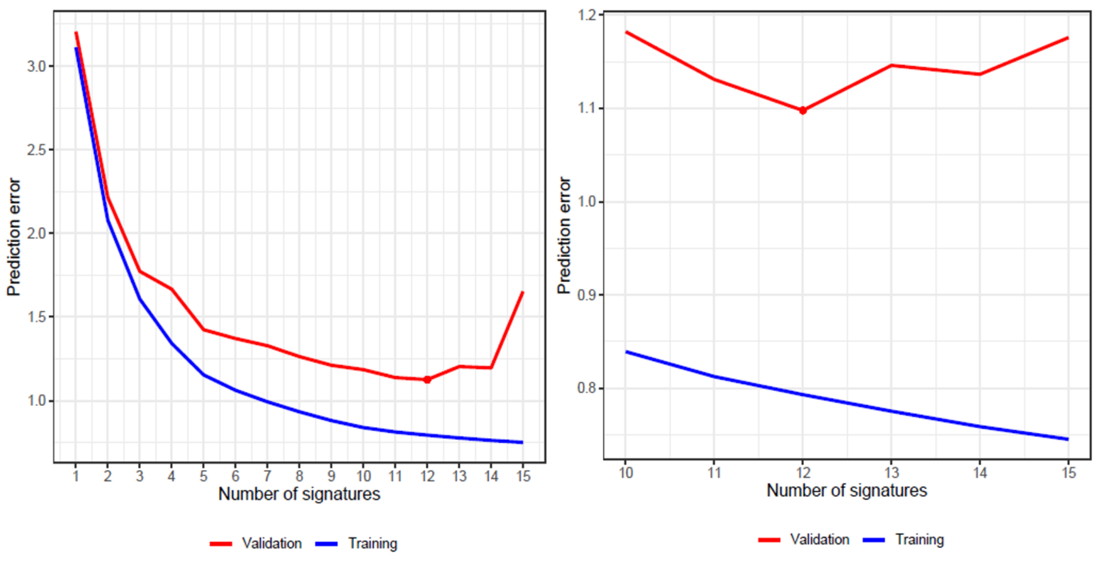

Figure 2: Prediction errors of SUITOR for the training and validation sets of BRCA440 data. The red dot denotes the number of signatures with the minimal prediction error in the validation set. The left panel is obtained with the maximal iteration 2000 while the right panel is obtained with 5000 `max.iter` and range [10, 15].

```
A[C>A]C 0.014403644 0.0013189131 0.008319862 0.003950195 4.509377e-04
A[C>A]G 0.003520653 0.0001585124 0.009031570 0.000299662 5.803079e-13
A[C>A]T 0.005245858 0.0010084542 0.012900031 0.002944834 3.798352e-04
C[C>A]A 0.008168012 0.0027272112 0.015780643 0.003197240 2.078258e-13
C[C>A]C 0.005144998 0.0012440051 0.011297759 0.001853106 5.312907e-04
denovo F      denovo G      denovo H     denovo I     denovo J
A[C>A]A 0.0011285193 0.0091659098 0.0006685698 0.045285236 0.017697343
A[C>A]C 0.0031150846 0.0105639679 0.0005950727 0.021238058 0.015249021
A[C>A]G 0.0001660976 0.0022483645 0.0002263572 0.002522311 0.001124053
A[C>A]T 0.0012826521 0.0036437353 0.0035279329 0.022941440 0.015562899
C[C>A]A 0.0014456343 0.0004147114 0.0103830893 0.047363192 0.021021077
C[C>A]C 0.0029514372 0.0026326900 0.0204591234 0.014492012 0.020779943
denovo K   denovo L
A[C>A]A 1.870085e-02 0.03276727
A[C>A]C 1.168430e-02 0.03964115
A[C>A]G 1.337687e-03 0.00375749
A[C>A]T 1.093196e-02 0.03748351
C[C>A]A 6.386215e-03 0.02832030
C[C>A]C 9.412555e-09 0.03229912
```

The **R** package *MutationalPatterns* (Blokzijl et al., 2021) provides some utility functions to summarize signature profiles. We used its function `plot_96_profile()` to draw the signature profile plot with respect to the 96 trinucleotide categories (Table 3). In addition, we annotated *de novo* signatures with known COSMIC signatures[4] by computing cosine similarities between two profiles, using the function `cos_sim_matrix()` from the *MutationalPatterns* package computes the cosine similarity as follows.

```
> library(MutationalPatterns)
> plot_96_profile(Ext$W, condensed = TRUE, ymax = 0.3)
> CS <- cos_sim_matrix(Ext$W, COSMIC)
> CS[, 1:3]
                SBS1       SBS2      SBS3
denovo A 0.966604695 0.08702249 0.1961778
denovo B 0.006959432 0.24779151 0.3008109
denovo C 0.071124086 0.09355373 0.8434715
denovo D 0.008827082 0.06301161 0.2297678
denovo E 0.056958855 0.98520696 0.1599579
denovo F 0.076298847 0.02962889 0.4541887
denovo G 0.107339862 0.29089700 0.5684496
denovo H 0.653400210 0.08110742 0.4561760
denovo I 0.168841140 0.16482585 0.4874278
denovo J 0.069574828 0.06944552 0.8874014
denovo K 0.205866555 0.09149522 0.6145402
denovo L 0.117290012 0.02210029 0.7832333
> cbind.data.frame(CS = apply(CS, 1, max),
        COSMIC = colnames(CS)[apply(CS, 1, which.max)] )
                CS  COSMIC
denovo A 0.9666047    SBS1
denovo B 0.9680711   SBS13
denovo C 0.8434715    SBS3
denovo D 0.9834026 SBS17b
denovo E 0.9852070    SBS2
denovo F 0.9566436   SBS26
denovo G 0.9332028   SBS30
denovo H 0.8248541    SBS6
```

[4]https://cog.sanger.ac.uk/cosmic-signatures-production/documents/COSMIC_v3.2_SBS_GRCh37.txt

```
denovo I 0.9482479  SBS18
denovo J 0.8883307  SBS39
denovo K 0.7887296  SBS41
denovo L 0.9507982   SBS8
```

References

Ramazzotti D., Lal A., De Sano L., and Sidow A. (2021). SparseSignatures: SparseSignatures. R package version 2.4.0, https://github.com/danro9685/SparseSignatures.

Nik-Zainal, S., Davies, H., Staaf, J. et al. (2016). Landscape of somatic mutations in 560 breast cancer whole-genome sequences. *Nature* **534**, 47—54.

Blokzijl, F., Janssen, R., van Boxtel, R. et al. (2018). MutationalPatterns: comprehensive genome-wide analysis of mutational processes. *Genome Medicine* **10**, 33.

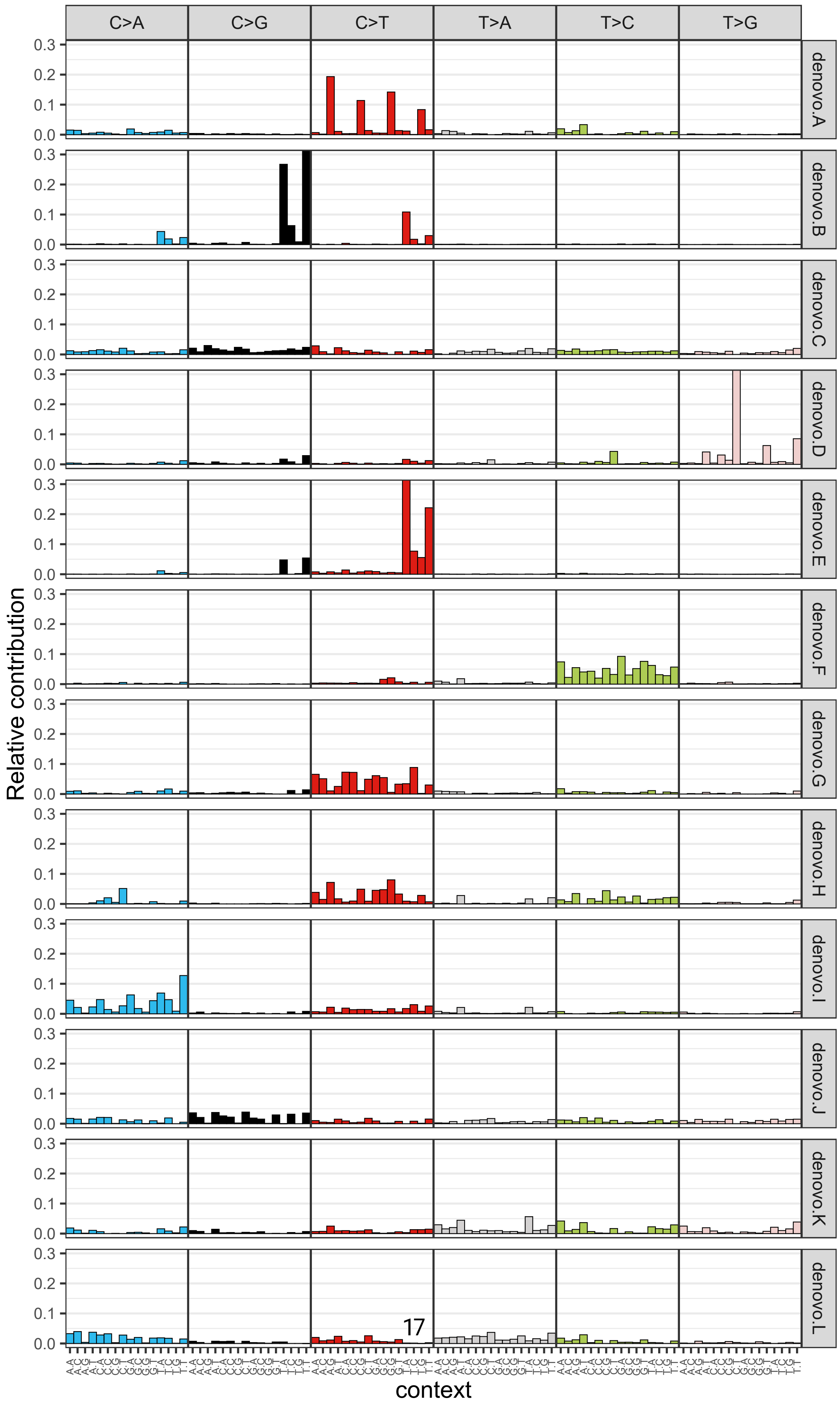

Figure 3: Signature profiles of BRCA440 data