An introduction to rScudo

***Matteo Ciciani*[*1], *Thomas Cantore*[1], *and Mario Lauria*[2,3]**

[1]Centre for Integrative Biology (CIBIO), University of Trento, Italy
[2]Department of Mathematics, University of Trento, Italy
[3]The Microsoft Research-University of Trento Centre for Computational and Systems Biology (COSBI), Rovereto, Italy

*matteo.ciciani@gmail.com

2019-10-29

Contents

1 Introduction

This package implements in R the SCUDO rank-based signature identification method described in [1] and [2]. SCUDO (Signature-based Clustering for Diagnostic Purposes) is a method for the analysis and classification of gene expression profiles for diagnostic and classification purposes. The `rScudo` package implements the very same algorithm that participated in the SBV IMPROVER Diagnostic Signature Challenge, an open international competition designed to assess and verify computational approaches for classifying clinical samples based on gene expression. SCUDO earned second place overall in the competition, and first in the Multiple Sclerosis sub-challenge, out of 54 submissions [3].

The method is based on the identification of sample-specific gene signatures and their subsequent analysis using a measure of signature-to-signature similarity. The computation of a similarity matrix is then used to draw a map of the signatures in the form of a graph, where each node corresponds to a sample and a connecting edge, if any, encodes the level of similarity between the connected nodes (short edge = high similarity; no edge = negligible similarity). The expected result is the emergence of a partitioning of the set of samples in separate and homogeneous clusters on the basis of signature similarity (clusters are also sometimes referred to as communities).

The package has been designed with the double purpose of facilitating experimentation on different aspects of the SCUDO approach to classification, and enabling performance comparisons with other methods. Given the novelty of the method, a lot of work remain to be done in order to fully optimize it, and to fully characterize its classification performance. For this purpose the package includes features that allow the user to implement his/her own signature similarity function, and/or clustering and classification methods. It also adds functions to implement steps that were previously performed manually, such as determining optimal signature length and computing classification performance indices, in order to facilitate the application and the evaluation of the method.

2 Method in brief

Starting from gene expression data, the functions `scudoTrain` and `scudoNetwork` perform the basic SCUDO pipeline, which can be summarized in 4 steps:

1. First, fold-changes are computed for each gene. Then, a feature selection step is performed. The user can specify whether to use a parametric or a non parametric test. The test used also depends on the number of groups present in the dataset. This step can be optionally skipped.
2. The subsequent operations include single sample gene ranking and the extraction of signatures formed by up-regulated and down-regulated genes. The length of the signatures are customizable. Consensus signtures are then computed, both for up- and down-regulated genes and for each group. The computation of consensus signatures is performed aggregating the ranks of the genes in each sample and ranking again the genes.
3. An all-to-all distance matrix is then computed using a distance similar to the GSEA (Gene Set Enrichment Analysis) [4]: the distance between two samples is computed as the mean of the enrichment scores (ES) of the signatures of each sample in the expression profile of the other sample. The distance function used is customizable.

4. Finally, a user-defined threshold N is used to generate a network of samples. The distance matrix is treated as an adjacency matrix, but only the distances that fall below the N^{th} quantile of distances are used to draw edges in the network. This is performed by the function `scudoNetwork`. The network can then be displayed in R or using Cytoscape.

The function `scudoTrain` returns an object of class `scudoResults`, which contains sample-specific gene signatures, consensus gene signatures for each group and the sample distance matrix.

After the identification of a list of genes that can be used to partition the samples in separated communities, the same procedure can be applied to a testing dataset. The function `scudoTest` performs steps 2 and 3 on a testing dataset, taking into account only the genes selected in the training phase.

Alteranatively, the function `scudoClassify` can be used to perform supervised classification. This function takes as input a training set, containing samples with known classification, and a testing set of samples with unknown classification. For each sample in the testing set, the function computes a network formed by all the samples in the training set and a single sample from the training set. Then, classification scores are computed for each sample in the testing set looking at the neighbors of that sample in the network. See the documentation of the function for a detailed description of the computation of the classification scores.

3 Example workflow of rScudo

3.1 Data preparation

In this example we will use the *ALL* dataset, containing gene expression data from T- and B-cells acute lymphoblastic leukemia patients. In this first part, we are interested in distinguishing B-cells and T-cells samples, based on gene expression profiles. We begin by loading relevant libraries and subsetting the dataset, dividing it in a training and a testing set, using the function `createDataPartition` from the package *caret*.

```
library(rScudo)
library(ALL)
data(ALL)

bt <- as.factor(stringr::str_extract(pData(ALL)$BT, "^."))

set.seed(123)
inTrain <- caret::createDataPartition(bt, list = FALSE)
trainData <- ALL[, inTrain]
testData <- ALL[, -inTrain]
```

3.2 Analysis of the training set

We start by analyzing the training set. We first run `scudoTrain`, which returns an object of class `ScudoResults`.

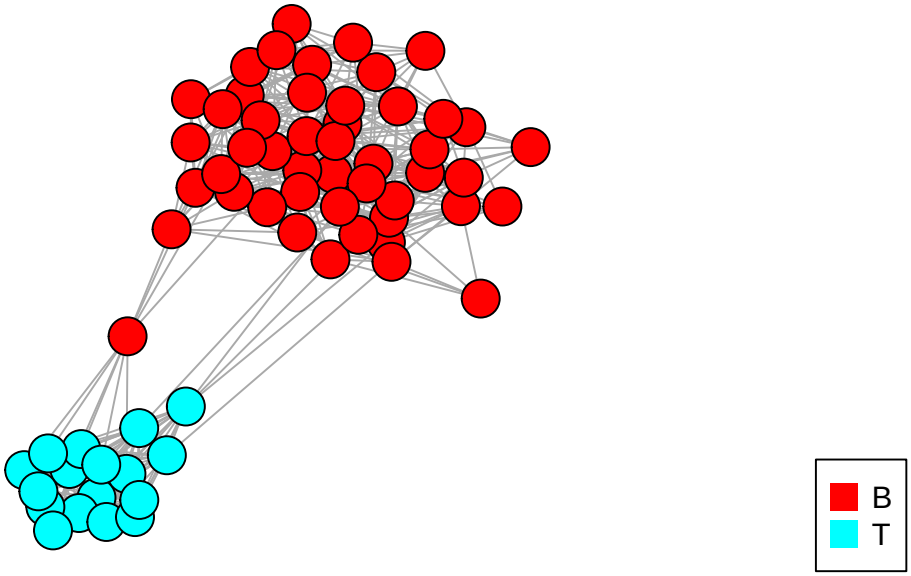
```
trainRes <- scudoTrain(trainData, groups = bt[inTrain], nTop = 100,
    nBottom = 100, alpha = 0.1)
trainRes
#> Object of class ScudoResults
#> Result of scudoTrain
#>
#> Number of samples      :  65
#> Number of groups       :  2
#>     B :  48 samples
#>     T :  17 samples
#> upSignatures length    :  100
#> downSignatures length  :  100
#> Fold-changes           :  computed
#>     grouped            :  No
#> Feature selection      :  performed
#>     Test               :  Wilcoxon rank sum test
#>     p-value cutoff     :  0.1
#>     p.adjust method    :  none
#>     Selected features  :  4286
```

From this object we can extract the signatures for each sample and the consensus signatures for each group.

```
upSignatures(trainRes)[1:5,1:5]
#>      04007      04010    04016     06002    08012
#> 1 36638_at 33273_f_at 36575_at  38355_at 38604_at
#> 2 34362_at 33274_f_at 40511_at  37283_at  1857_at
#> 3 37006_at   38514_at 37623_at  40456_at 878_s_at
#> 4  1113_at   39318_at 547_s_at  41273_at 38355_at
#> 5 40367_at 35530_f_at 37187_at 2036_s_at 37921_at
consensusUpSignatures(trainRes)[1:5, ]
#>            B         T
#> 1   37039_at  38319_at
#> 2   35016_at  33238_at
#> 3   39839_at  38147_at
#> 4 38095_i_at  37078_at
#> 5 38096_f_at 2059_s_at
```

The function `scudoNetwork` can be used to generate a network of samples from the object `trainRes`. This function returns an *igraph* object. The parameter `N` controls the percentage of edges to keep in the network. We can plot this network using the function `scudoPlot`.

```
trainNet <- scudoNetwork(trainRes, N = 0.25)
scudoPlot(trainNet, vertex.label = NA)
```



You can also render the network in Cytoscape, using the function `scudoCytoscape`. Note that Cytoscape has to be open when running this function.

```
scudoCytoscape(trainNet)
```

Since we obtained a very good separation of the two groups, we proceed to analyze the testing set.

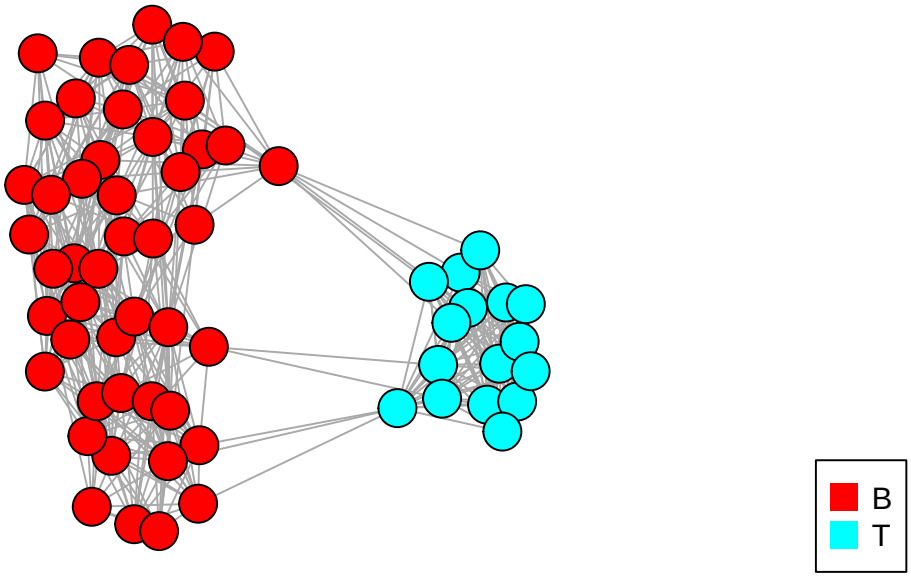
3.3 Analysis of the testing set

We can use a `ScudoResults` object and the function `scudoTest` to analyze the testing set. The feature selection is not performed in the testing set. Instead, only the features selected in the training step are used in the analysis of the testing set.

```
testRes <- scudoTest(trainRes, testData, bt[-inTrain], nTop = 100,
    nBottom = 100)
testRes
#> Object of class ScudoResults
#> Result of scudoTest
#>
#> Number of samples      :  63
#> Number of groups       :  2
#>     B :  47 samples
#>     T :  16 samples
#> upSignatures length    :  100
#> downSignatures length  :  100
#> Fold-changes           :  computed
#>     grouped            :  No
```

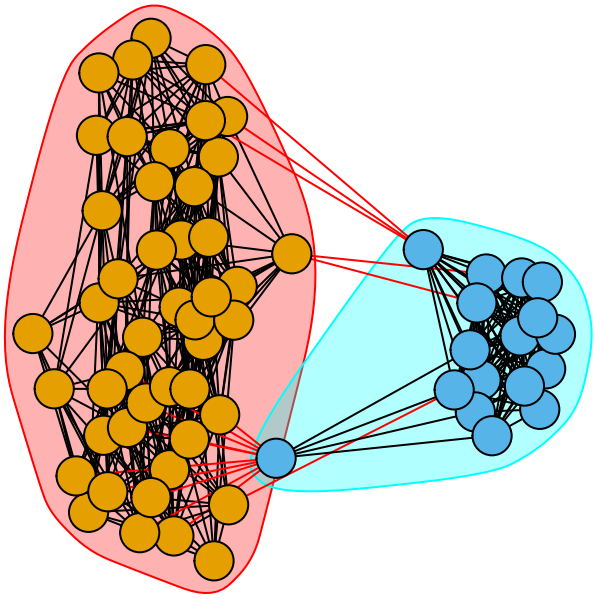
We can generate a network of samples and plot it.

```
testNet <- scudoNetwork(testRes, N = 0.25)
scudoPlot(testNet, vertex.label = NA)
```



We can use a community clustering algorithm to identify clusters of samples. In the following example we use the function `cluster_spinglass` from the package *igraph* to perform clustering of our network. In Cytoscape we can perform a similar analysis using clustering functions from the clusterMaker app.

```
testClust <- igraph::cluster_spinglass(testNet, spins = 2)
plot(testClust, testNet, vertex.label = NA)
```



3.3.1 Supervised classification

`scudoClassify` performs supervised classification of sample in a testing set using a model built from samples in a training set. It uses a method based on neighbors in the graph to assign a class label to each sample in the testing set. We suggest to use the same `N`, `nTop`, `nBottom` and `alpha` that were used in the training step.

```
classRes <- scudoClassify(trainData, testData, N = 0.25, nTop = 100,
    nBottom = 100, trainGroups = bt[inTrain], alpha = 0.1)
```

Classification performances can be explored using the `confusionMatrix` function from *caret*.

```
caret::confusionMatrix(classRes$predicted, bt[-inTrain])
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  B  T
#>          B 47  0
#>          T  0 16
#>
#>                Accuracy : 1
#>                  95% CI : (0.9431, 1)
#>     No Information Rate : 0.746
#>     P-Value [Acc > NIR] : 9.632e-09
#>
#>                   Kappa : 1
#>
#>  Mcnemar's Test P-Value : NA
#>
#>             Sensitivity : 1.000
#>             Specificity : 1.000
#>          Pos Pred Value : 1.000
#>          Neg Pred Value : 1.000
#>              Prevalence : 0.746
#>          Detection Rate : 0.746
#>    Detection Prevalence : 0.746
#>       Balanced Accuracy : 1.000
#>
#>        'Positive' Class : B
#>
```

3.4 Example of multigroup analysis

The analysis can also be performed on more than two groups. In this section, we try to predict the stage of B-cells ALL using gene expression data. We focus only on stages B1, B2 and B3, since they have a suitable sample size.

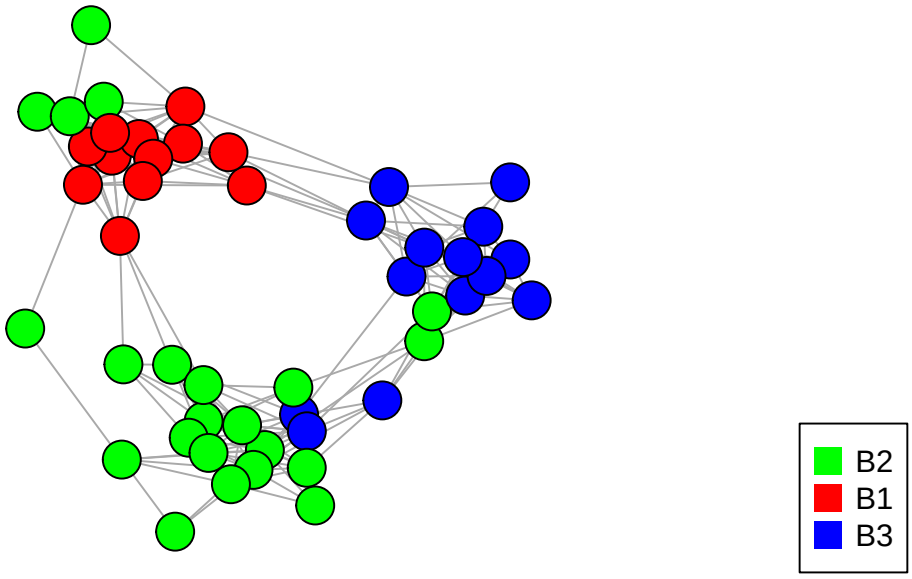
```
isB <- which(as.character(ALL$BT) %in% c("B1", "B2", "B3"))
ALLB <- ALL[, isB]
stage <- ALLB$BT[, drop = TRUE]
table(stage)
```

```
#> stage
#> B1 B2 B3
#> 19 36 23
```

We divide the dataset in a training and a testing set and we apply `scudoTrain`, identifying suitable parameter values. Then, we perform supervised classification of the samples in the testing set using the function `scudoClassify`.

```
inTrain <- as.vector(caret::createDataPartition(stage, p = 0.6, list = FALSE))

stageRes <- scudoTrain(ALLB[, inTrain], stage[inTrain], 100, 100, 0.01)
stageNet <- scudoNetwork(stageRes, 0.2)
scudoPlot(stageNet, vertex.label = NA)
```




```
classStage <- scudoClassify(ALLB[, inTrain], ALLB[, -inTrain], 0.25, 100, 100,
    stage[inTrain], alpha = 0.01)
caret::confusionMatrix(classStage$predicted, stage[-inTrain])
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction B1 B2 B3
#>         B1  6  3  1
#>         B2  1 10  1
#>         B3  0  1  7
#>
#> Overall Statistics
#>
#>                Accuracy : 0.7667
#>                  95% CI : (0.5772, 0.9007)
#>     No Information Rate : 0.4667
#>     P-Value [Acc > NIR] : 0.0008038
#>
```

```
#>                  Kappa : 0.6441
#>
#>  Mcnemar's Test P-Value : 0.5724067
#>
#> Statistics by Class:
#>
#>                      Class: B1 Class: B2 Class: B3
#> Sensitivity             0.8571    0.7143    0.7778
#> Specificity             0.8261    0.8750    0.9524
#> Pos Pred Value          0.6000    0.8333    0.8750
#> Neg Pred Value          0.9500    0.7778    0.9091
#> Prevalence              0.2333    0.4667    0.3000
#> Detection Rate          0.2000    0.3333    0.2333
#> Detection Prevalence    0.3333    0.4000    0.2667
#> Balanced Accuracy       0.8416    0.7946    0.8651
```

3.5 Increasing performance through parameter tuning

Parameters such as `nTop` and `nBottom` can be optimally tuned using techniques such as cross-validation. The package *caret* offers a framework to perform grid search for parameters tuning. Here we report an example of cross-validation, in the context of the multigroup analysis previously performed. Since feature selection represents a performance bottleneck, we perform it before the cross-validation. Notice that we also transpose the dataset, since functions in *caret* expect features on columns and samples on rows.

```
trainData <- exprs(ALLB[, inTrain])
virtControl <- rowMeans(trainData)
trainDataNorm <- trainData / virtControl
pVals <- apply(trainDataNorm, 1, function(x) {
    stats::kruskal.test(x, stage[inTrain])$p.value})
trainDataNorm <- t(trainDataNorm[pVals <= 0.01, ])
```

We use the function `scudoModel` to generate a suitable input model for `train`. `scudoModel` takes as input the parameter values that have to be explored and generates all possible parameter combinations. We then call the function `trainControl` to specify control parameters for the training procedure and perform it using `train`. Then we run `scudoClassify` on the testing set using the best tuning parameters found by the cross-validation. We use parallelization to speed up the cross-validation.

```
cl <- parallel::makePSOCKcluster(2)
doParallel::registerDoParallel(cl)

model <- scudoModel(nTop = (2:6)*20, nBottom = (2:6)*20, N = 0.25)
control <- caret::trainControl(method = "cv", number = 5,
    summaryFunction = caret::multiClassSummary)
cvRes <- caret::train(x = trainDataNorm, y = stage[inTrain], method = model,
    trControl = control)

parallel::stopCluster(cl)
```

```
classStage <- scudoClassify(ALLB[, inTrain], ALLB[, -inTrain], 0.25,
    cvRes$bestTune$nTop, cvRes$bestTune$nBottom, stage[inTrain], alpha = 0.01)
caret::confusionMatrix(classStage$predicted, stage[-inTrain])
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction B1 B2 B3
#>         B1  6  4  1
#>         B2  1  9  0
#>         B3  0  1  8
#>
#> Overall Statistics
#>
#>                Accuracy : 0.7667
#>                  95% CI : (0.5772, 0.9007)
#>     No Information Rate : 0.4667
#>     P-Value [Acc > NIR] : 0.0008038
#>
#>                   Kappa : 0.6512
#>
#>  Mcnemar's Test P-Value : 0.2838861
#>
#> Statistics by Class:
#>
#>                      Class: B1 Class: B2 Class: B3
#> Sensitivity             0.8571    0.6429    0.8889
#> Specificity             0.7826    0.9375    0.9524
#> Pos Pred Value          0.5455    0.9000    0.8889
#> Neg Pred Value          0.9474    0.7500    0.9524
#> Prevalence              0.2333    0.4667    0.3000
#> Detection Rate          0.2000    0.3000    0.2667
#> Detection Prevalence    0.3667    0.3333    0.3000
#> Balanced Accuracy       0.8199    0.7902    0.9206
```

4 Session info

```
sessionInfo()
#> R version 3.6.1 (2019-07-05)
#> Platform: x86_64-apple-darwin15.6.0 (64-bit)
#> Running under: OS X El Capitan 10.11.6
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
```

```
#> attached base packages:
#> [1] parallel  stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#> [1] ALL_1.27.0          Biobase_2.46.0      BiocGenerics_0.32.0
#> [4] rScudo_1.2.0        BiocStyle_2.14.0
#>
#> loaded via a namespace (and not attached):
#>  [1] tidyselect_0.2.5   xfun_0.10          reshape2_1.4.3
#>  [4] purrr_0.3.3        splines_3.6.1      lattice_0.20-38
#>  [7] colorspace_1.4-1   generics_0.0.2     stats4_3.6.1
#> [10] htmltools_0.4.0    yaml_2.2.0         prodlim_2018.04.18
#> [13] survival_2.44-1.1  rlang_0.4.1        e1071_1.7-2
#> [16] ModelMetrics_1.2.2 pillar_1.4.2       withr_2.1.2
#> [19] glue_1.3.1         foreach_1.4.7      plyr_1.8.4
#> [22] lava_1.6.6         stringr_1.4.0      timeDate_3043.102
#> [25] munsell_0.5.0      gtable_0.3.0       recipes_0.1.7
#> [28] codetools_0.2-16   evaluate_0.14      knitr_1.25
#> [31] doParallel_1.0.15  caret_6.0-84       class_7.3-15
#> [34] Rcpp_1.0.2         scales_1.0.0       BiocManager_1.30.9
#> [37] S4Vectors_0.24.0   ipred_0.9-9        ggplot2_3.2.1
#> [40] digest_0.6.22      stringi_1.4.3      bookdown_0.14
#> [43] dplyr_0.8.3        grid_3.6.1         tools_3.6.1
#> [46] magrittr_1.5       lazyeval_0.2.2     tibble_2.1.3
#> [49] crayon_1.3.4       pkgconfig_2.0.3    MASS_7.3-51.4
#> [52] Matrix_1.2-17      data.table_1.12.6  lubridate_1.7.4
#> [55] gower_0.2.1        assertthat_0.2.1   rmarkdown_1.16
#> [58] iterators_1.0.12   R6_2.4.0           rpart_4.1-15
#> [61] igraph_1.2.4.1     nnet_7.3-12        nlme_3.1-141
#> [64] compiler_3.6.1
```

References

[1] Lauria M. Rank-based transcriptional signatures. Systems Biomedicine. 2013; 1(4):228-239.

[2] Lauria M, Moyseos P, Priami C. SCUDO: a tool for signature-based clustering of expression profiles. Nucleic Acids Research. 2015; 43(W1):W188-92.

[3] Tarca AL, Lauria M, Unger M, Bilal E, Boue S, Kumar Dey K, Hoeng J, Koeppl H, Martin F, Meyer P, et al. IMPROVER DSC Collaborators. Strengths and limitations of microarray-based phenotype prediction: lessons learned from the IMPROVER Diagnostic Signature Challenge. Bioinformatics. 2013; 29:2892–2899.

[4] Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, Paulovich A, Pomeroy SL, Golub TR, Lander ES, Mesirov JP. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. PNAS. 2005; 102(43):15545-15550.