Users' Guide

MultiML

Leandro Balzano Nogueira (*leobalzano@ufl.edu*)
Version 0.1.0 (October 9, 2019)

Content

Introduction

Although differential expression methods are needed to perform an initial variable selection step and hence it is key to be aware of how statistical power of such methods applied on our data can impact the results, an integrative multi-omic model is the final goal of multi-omic studies in many cases. Dimension reduction techniques or machine learning approaches are a popular choice in these cases and the biological problem to solve will determine if the selected methodology is unsupervised or supervised, and for either classification or prediction. For any of these approaches, the optimal size required to optimize the performance of the chosen method is still an open question in the field and, up to our knowledge, no tools exist to estimate sample sizes in a systematic way. On top of that, the classical definition of statistical power is not appropriate for machine learning algorithms where a significance level has to be performed *a posteriori*. Thus, in the particular case of supervised approaches, the performance can be measured in terms of classification or prediction error rates.
The MultiML method employs a multi-omic pilot data to estimate the minimum sample size required to keep the error rate below the desired threshold. This MultiML version uses Partial Least Squares-Discriminant Analysis (PLS-DA) or Random Forest classification methods to depict K subsamples of different sample sizes (n_k) from the available observations, obtaining a classifier for each case, and measuring the corresponding error rate (e_k). On these (n_k, e_k) values, a first order-smoothed penalized P-spline regression is performed to estimate the learning curve that will allow the prediction of error rates for larger sample sizes. The MultiML method was created to help userd to determine optimal sample size required for an accurate classification error rate, when using one or more Omics datasets.

Getting Started

The MultiML method is available in https://bitbucket.org/ConesaLab/multiml/.

Required R libraries:
These are the libraries needed from the CRAN repository that can be installed with `install.packages()` function and then loaded with `library()`.

- plotly
- ggplot2
- gridExtra
- grid
- ggpmisc
- RColorBrewer
- dplyr
- glmnet
- reshape2
- rslurm
- whisker
- randomForest
- caret
- gtable
- pls

Required libraries from Bioconductor repository that must be installed following the Bioconductor instructions at:
https://bioconductor.org/packages/release/bioc/html/RnaSeqSampleSize.html

- mixOmics
- Biobase

Finally, there are some basic packages that do not need to be updated but they need to be loaded:

- parallel
- splines
- graphics

After installing all packages, it is necessary to load them all:

```
library("plotly")
library("ggplot2")
library ("gridExtra")
library("grid")
library("ggpmisc")
library("RColorBrewer")
library("mixOmics")
library("dplyr")
library ("glmnet")
library ("Biobase")
library("reshape2")
library("parallel")
library("rslurm")
library("whisker")
library("randomForest")
library("caret")
library("gtable")
library("splines")
library("graphics")
library("pls")
```

Functions:
A set of functions developed to perform the MultiML predictive analysis are required and it is necessary to upload them as well.

```
source ("FoMPredictiveFunctionsV5.R")
```

Prediction of the number of samples required to reach a particular classification error rate

MultiML method requires some input for each omic data type in order to compute a model to understand and predict the adequate number of samples with which the user can reach reasonably low classification errors. Utilizing Omics data results from a pilot study, the user can then extrapolate how many more samples they need to reach an acceptable error of classification.
Next a sequence of the functions present in the method will be explained, in terms of inputs required and outputs obtained:

Input data
As unique input data required for MultiML to operate is a list of as many Omics datasets as the user wants and the Response matrix that can be binary or multifactorial.
The names of the omics should be the names of each table in the list. All tables must be arranged in the same way, being in rows, the variables (genes, proteins, etc.) and in columns, the different samples (treatments and/or controls). Each element in this list must be a raw count data frame.

```
        TCGA-02-0432-01 TCGA-08-0245-01 TCGA-28-1750-01 TCGA-06-1084-01 TCGA-02-0064-01
FSTL1          6.732094        9.172518       10.003722        9.818185       11.002899
AACS           6.121826        6.348193        7.067599        6.822424        6.676739
RPS11         10.783535       10.533063       10.438774       11.006742       10.662374
CREB3L1        4.487414        4.620867        4.339173        4.283061        4.711638
ELMO2          7.770408        6.158398        7.035239        5.751987        7.152336
PNMA1          9.921098        9.556290        9.143960        7.626237        9.838729
```

The response matrix must indicate a category for each sample.

```
        TCGA-02-0432-01 TCGA-08-0245-01 TCGA-28-1750-01 TCGA-06-1084-01 TCGA-02-0064-01
Type          Proneural       Proneural      Mesenchymal     Mesenchymal     Mesenchymal
```

Estimation of the time required for a particular study:
Since MultiML is a computationally expensive, time consuming strategy, it is recommendable to perform an estimation of the required time necessary to analyze a particular study.
For this, a function `RequiredtimeTest()` was created. Its arguments are described in detail next:

Input
Predictors: A list of the different Omics Datasets and the Response matrix.
Response: A number indicating the position of the response matrix included in "Predictors" object.
Comps: Number of components to be calculated after each iteration. Just applicable to PLSDA approach.

Function: Modular function used to calculate the error rate. It can be PLSDA.MP or Random.Forest.MP to indicate the approach to be used.
crosval: Type of cross validation to be applied, Leave-One-Out (LOOCV) or ten fold change (TenF).
Ticks: Number of segments (groups) of samples to evaluate.
ERIterations: Number of iterations in which the error rate (ER) will be calculated.
LassoIterations: Number of iterations of the Lasso selection per each error rate analysis.
cpus_per_node: Number of CPUs that will be used in the analysis.
...: Arguments to be passed to methods.

Example:

```
EstTime<-RequiredtimeTest (Predictors=tcgadata[c(4,6)],Response=2,
                           cpus_per_node=1,
                           Ticks = 50, Function = PLSDA.MP,Comps = 10,
                           crosval = "LOOCV",ERIterations = 50,
                           LassoIterations=50)
```

Output:

Summary: A table indicating the conditions of the analysis, established by the user.
EstimatedTime: A table of the estimated time that the process will last showed in different metrics (seconds, minutes, hours, or days).

Classification Error Rate calculus

This is calculated through the `ER_Calculator()` function which is the main function in the MultiML method.

Input:

Predictors: A list of different Omics Datasets and the Response matrix.
Response: A number indicating the response matrix included in "Predictors" object.
Previous_CER: A previous result of class "`ClassificationErrorRate`" (CER) to be fused to the one that is going to be calculated. This is useful for merging a pilot study with a posterior analysis.
Ticks: Number of segments (groups) of samples to evaluate. If NULL, the calculation is made on its own considering the "`TheoreticER`".
WhichTicks: Vector of numbers of ticks the user wants to analyze. If NULL, a random selection between the minimum and maximum number of samples is calculated. It results useful in posterior rounds of analysis.
Function: Modular function used to calculate the error rate. It can be PLSDA.MP or Random.Forest.MP to indicate the approach to be used.
Comps: Number of components to be calculated after each iteration. Just applicable to PLSDA approach.
crosval: Type of cross validation to be applied, Leave-One-Out (LOOCV) or ten fold change (TenF).
ERIterations: Number of iterations in which the error rate (ER) will be calculated.
LassoIterations: Number of iterations of the Lasso selection per each error rate analysis.

Example:

```
ProtsRF<-ER_Calculator(Predictors=tcgadata[c(4,6)],
                       Response=2,Previous_CER=NULL,Ticks=NULL,
                       WhichTicks=NULL,Function=Random.Forest.MP,
                       Comps=10,crosval = "LOOCV",ERIterations=2,
                       LassoIterations=2, TheoreticER=0.02)
```

Output:

TestedTicks: A vector of the number of evaluated number of samples.
Omics: An indicator of the Omics evaluated.
Minimums: A list of the minimum value of error rate, balanced (BER) or not (ER) obtained per each ten-component analysis. This is measured through three distance metrics to evaluate the classification performance of the model. Maximal distance “max.dist”, distance to centroids “centroids.dist” or Mahalanobis distance “mahalanobis.dist”. Thus, each table contains the results per each iteration at different subsets of samples.
TablebyTick: A data frame with information related to the classification error rate obtained by tick ± standard error of the mean “SEM” and the characteristics of the analysis selected by the user.
Prediction_table: A table with the information related to de predicted number of samples required to reach a particular ER ± a margin of error “MOE”.

CompWinner: A list of the number of components in which the minimum value of error rate, balanced (BER) or not (ER) was obtained per each iteration. This only applies for PLSDA module and it is measured through the three mentioned distance metrics, Maximal distance, distance to centroids or Mahalanobis distance, to evaluate the classification performance of the model.

Elaboration of files and folder for running analyzes in a high-performance computer (HPC)

Often, the users need to evaluate the data thoroughly, with analyzes among two or more Omics datasets, or determine if an Omics type is necessary or not for a particular study.
To overcome the fact that the method is computationally demanding, we offer a set of functions that allows the data arrangement and the creation of procedure commands, so the user can be able to run them into an HPC, as long as the HPC can be run through the Slurm workload manager job scheduler.

Step 1: Determining the comparisons to be evaluated in the dataset.
This can be performed with the `CombinationsFunction()` function. Here, the user can create the combination of datasets, so they can be included in the HPC.

Input:
OmicsData: The data as list of predictor datasets including also the response (Y) matrix as the last element of the list.
AnalysisType: Here we offer three types of analyzes that can be called. "exploratory", where all predictor matrices are going to be compared vs response matrix (Y) separately, as well as all omics dataset vs Y. "complete", where simply all combinations are created, even though this is not recommended because of the amount of time that a job of these characteristics implies. "detailed", where the user can choose by levels indicating if two datasets vs Y or three vs Y is required.
levels: A vector indicating the levels to be analyzed. If "AnalysisType" = "detailed", if 2, then two predictors datasets will be contrasted against Y and so on.

Example:
```
ExplTest<-CombinationsFunction (OmicsData=tcgadata,
                                AnalysisType="detailed",
                                levels=c(2))
```

Output:
Combinations: A list of the combinations created by the user.
Data: All required lists of datasets to be used as input for the error rate analysis.

Step 2: Creating the files and folders to upload into an HPC.
For this, the user can simply follow a procedure to create an "SlurmFunction" as follows:
```
SlurmFunction <- function (X) {
                           ER_Calculator(Predictors=X,
                           Response=length(X), Previous_CER=NULL,
                           Ticks=NULL, WhichTicks=NULL,
                           Function=Random.Forest.MP, Comps=10,
                           crosval = "LOOCV", ERIterations=2,
                           LassoIterations=2, TheoreticER=0.02)
                           }
```

Then, the user can create their “SlurmJob” using the `Slurm_Creator()` function.
This function creates the files to be uploaded into a cluster to calculate the results of all desired combinations of predictor datasets.

Input:
Function: An object class “SlurmFunction” that will calculate the error rates.
Parameters: All required lists of datasets to be used as input for the error rate calculation. These parameters are the ones obtained in step 1.
jobname: The desired name of the Slurm job. If NA, a random name of the form "slr####" will be assigned.
nodes: The maximum number of nodes to be used in the cluster.
cpus_per_node: The number of CPUs per node in the cluster. It determines how many processes are run in parallel per node.
pkgs: A character vector containing the names of packages that must be loaded on each cluster node. By default, it includes all packages loaded by the user when slurm_apply is called.
time: Time in days to run the data in the cluster. By default is 30 days, but according to the user HPC-availability this can be different.

Example:
```
My_Slurm_job <- Slurm_Creator(Function=SlurmFunction,
                              Parameters=ExplTest$Data,
                              jobname = 'my_Job', nodes = 8,
                              cpus_per_node = 100, time = 30,
                              pkgs = rev(.packages()) )
```

Output:
Four elements to be uploaded into cluster are created:
Function.RDS: A .RDS file with the Slurm Function that will be calculated.
Parameters.RDS: A .RDS file with all lists of datasets that will be used for the error rate calculation.
SlurmCreator_Run.R: A .R file with the parameters to run the job.
SlurmCreator_submit_sh: A .sh file that indicates all parameters that the cluster requires to perform the job.

Step 3: Upload all four files into an HPC and run.
Step 4: Once the job is done, download all files from the HPC.
Step 5: Plot results.

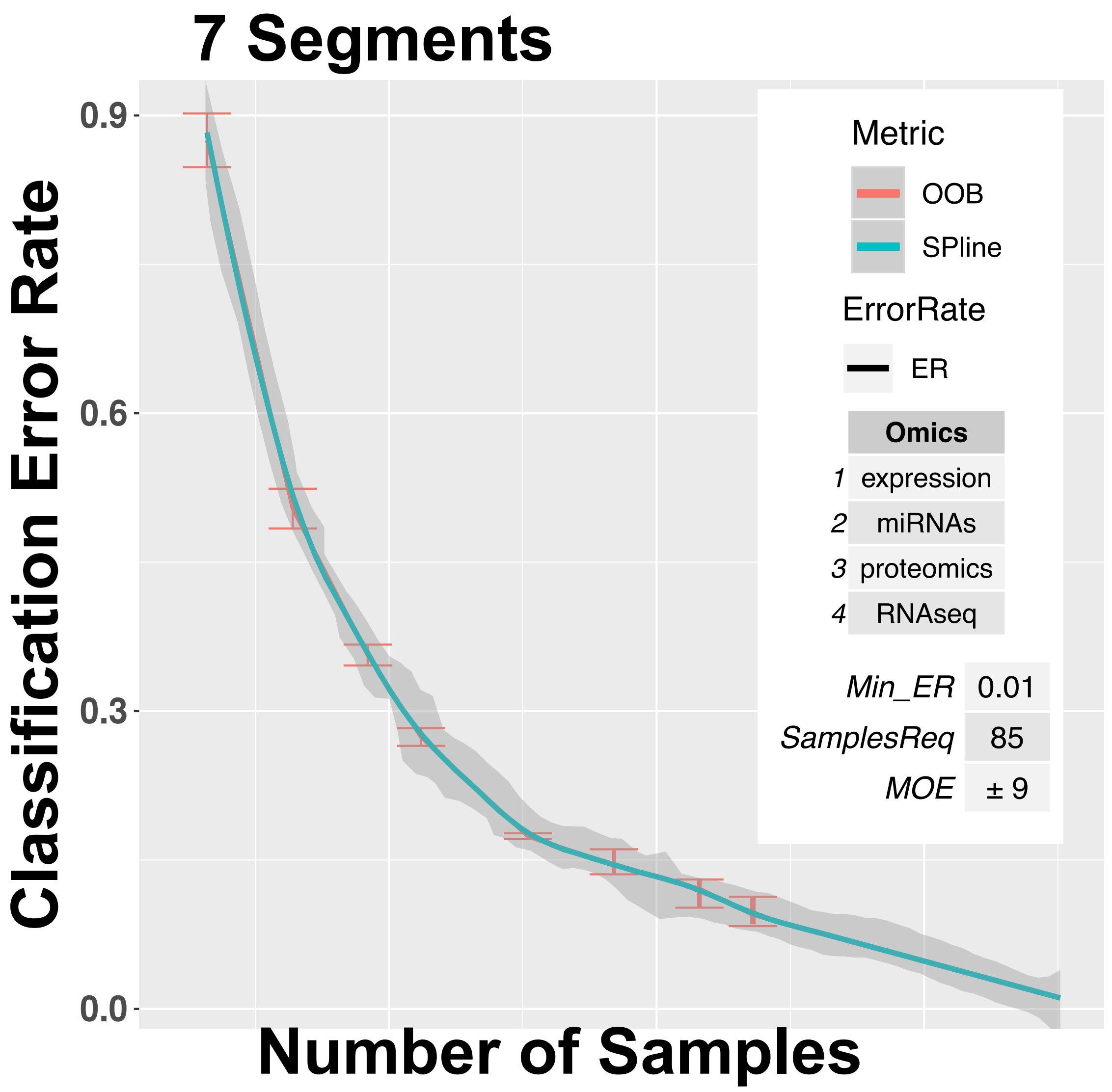
Plotting the results

Predictive plot.
The predictive plots are generated throughout the function called `ErrorRateplot()`. This function can not only use the information calculated in the previous step but also, it can calculate the samples required for ER values even lower than the ones reached in the pilot experiment.

Input:
x: An object of class ER_calculator
ErrorRateType: Character to indicate which error rate to plot. It can be error rate "ER", balanced error rate "BER" or "Both".
MetricsType: Character to indicate the metrics to be plotted. This only applies for PLSDA module. It can be Maximal distance "max.dist", distance to centroids "centroids.dist", Mahalanobis distance "mahalanobis.dist" or "All"
DoF: It can be either NULL to indicate that the degrees of freedom of the Spline model are ticks-1, or a value so the model is created with a different degree of freedom. The chosen DoF affects the samples required so the user must be careful.
Projection: Character to indicate if the user needs for the Spline model to be projected until the minimum error rate Min_ER (TRUE) or not (FALSE).
Spline: Character to indicate if the user needs to plot the Spline model (TRUE) or not (FALSE).
TheoreticER: Character to indicate the minimum value of ER in order to calculate the adequate number of samples.
ConfInt: Character to indicate the confidence interval for the calculation of margin of error and plot of the confidence area of the Spline model. The user can use 0.90, 0.95 or 0.99.

Example:
```
plot<-ErrorRateplot(x=ProtsRF,Projection=TRUE, Spline=TRUE,
                    DoF = NULL,ErrorRateType = "ER",
                    MetricsType ="mahalanobis.dist",TheoreticER=NULL,
                    ConfInt=0.95 )
```

Output:
A plot of “number of samples” versus “classification error rate”, indicating also the metric used, the Omics evaluated, and the samples required to reach a particular error rate ± a margin of error “MOE”.



Comparative plot.
The user can also create a comparative plot of all Omics combinations to determine the best contributing Omics to an accurate classification by using `Comparative_ERPlot()` function.

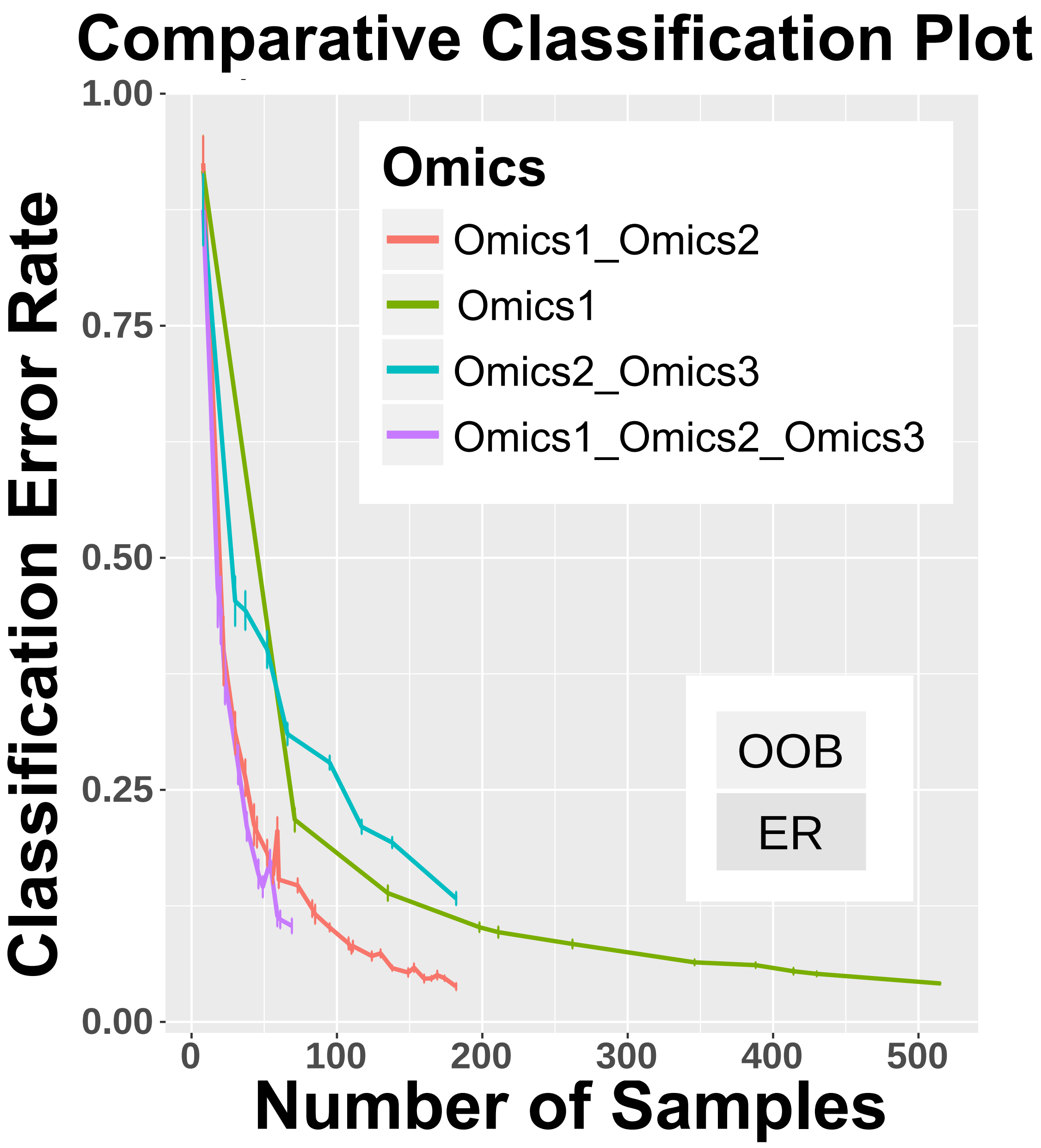
Input:
L: list of different Omics-integration results obtained with `ER_Calculator()` function.
ErrorRateType: Character to indicate which error rate to plot. It can be error rate "ER", balanced error rate "BER" if the PLSDA approach was the one calculated.
MetricsType: Character to indicate the metrics to be plotted. This only applies for PLSDA module. It can be Maximal distance "max.dist", distance to centroids "centroids.dist" or Mahalanobis distance "mahalanobis.dist".

Example:
First, it is necessary to simply create a list of the results you want to plot together as follows:

```
Allplots<-list(
        My_Omics1_Result= My_Omics1 _Result,
        My_Omics2_Result= My_Omics2_Result,
        My_Omics3_Result= My_Omics3_Result,
        My_Omics1_Omics2_Result= My_Omics1_Omics2_Result,
        My_Omics1_Omics3_Result= My_Omics1_Omics3_Result,
        My_Omics1_Omics2_Omics3_Result=My_Omics1_Omics2_Omics3_Result,
        My_Omics2_Omics3_Result= My_Omics2_Omics3_Result
       )

Then, simply use the function
Plot<- Comparative_ERPlot(L=Allplots,
                          ErrorRateType = "ER",
                          MetricsType ="mahalanobis.dist")
```

Output:
A plot of “number of samples” versus “classification error rate” of all different analyzes.


Comparative Classification Plot
Classification Error Rate
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
Number of Samples
Omics
Omics1_Omics2
Omics1
Omics2_Omics3
Omics1_Omics2_Omics3
OOB
ER

How to cite MultiML

Soon in bioRxiv !!

References

[1] Hastie, T., James, G., Witten, D., Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer. New York. pp. 316–321.

[2] Meyer, M. C. (2008). Inference using shape-restricted regression splines. *The Annals of Applied Statistics*, *2*(3), 1013-1033.

[3] Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical science*, *3*(4), 425-441.