images/biomarkerDiscoveryWorkflow

images/biomarkerValidationScheme

```
opts_chunk$set(fig.path='images/grafic', tidy=FALSE, dev='pdf')
```

```
workingDir <- getwd()
dataDir <- file.path(workingDir, "dades")
resultsDir <- file.path(workingDir, "results")
codeDir <-file.path(workingDir, "Rcode")
setwd(workingDir)

options(width=80)
options(digits=5)

toPNG <- FALSE
```

concordance=TRUE

1. Introduction

A common task in bioinformatics is tu build and validate a predictor to distinguish between two or more biological classes based on a series of biological features such as genes, microRNA, metabolites (or, more recently, a combination of some of these).

Our goal here is to illustrate the *standard process* such as described by Sanchez et alt. ([**?**]) or by Serra Cayuela ([**?**]) and summarized in the follwing two figures.

Figure 1 illustrates the basics blocks on which the biomarker discovery process can be arbitrarily divided. Figure 1 illustrates how to use cross-validation to build and validate a biomarker in such a way that unbiased estimates of generalization error can be obtained.

1.1. The CMA package

There are many packages in **R** to apply each of the many available classification methods. Some of them such as `caret` (**??**), `CMA` (**??**) or `MLtools` (**??**) also support the process of building and validating a classifier based on testing a set of different approaches on a set of samples using an appropriate cross-validation approach.

Here we rely on the Bioconductor `CMA` ("Classification for MicroArrays") package which has been specifically designed with microarray data in mind, but can also be used with any high throughput data.

```r
# Compte: cal carregar abans el paquet e1071 perque si no es fa així pot donar e

if (!require(e1071)){
  install.packages("e1071", dep=TRUE)
}
require(e1071)

if (!require(glmnet)){
  install.packages("glmnet", dep=TRUE)
}
require(glmnet)

if (!require(randomForest)){
  install.packages("randomForest", dep=TRUE)
}
require(randomForest)


if (!(require(CMA))){
  source("http://Bioconductor.org/biocLite.R")
  biocLite("CMA")
}
require(CMA)
```

According to its manual the aim of the package is *to provide a user-friendly environment for the evaluation of classification methods using gene expression data*. A strong focus is on combined variable selection, hyperparameter tuning, evaluation, visualization and comparison of (up to now) 21 classification methods from three main fields: Discriminant Analysis, Neural Networks and Machine Learning.

Using this package a (not-so-)simple workflow for building and validating a classifier can be built. The main steps for this workflow are:

1. Start with a high-throughput dataset (e.g. en expression matrix) and a vector of labels assigning each column of the dataset to a group.
2. Generate a given number of evaluation datasets using `GenerateLearningsets`.
3. (Optionally): Perform variable selection using `GeneSelection`.
4. (Optionally): Peform hyperparameter tuning using `tune`.
5. Perform classification using 1.-3.
6. Repeat steps 3–5 based on the learning sets generated in step 2 for an appropriate (wisely chosen) subset of all available methods[1].

[1] compBoostCMA, dldaCMA, ElasticNetCMA, fdaCMA, flexdaCMA, gbmCMA, knnCMA, ldaCMA, LassoCMA, nnetCMA, pknnCMA, plrCMA, pls_ldaCMA, pls_lrCMA, pls_rfCMA, pnnCMA, qdaCMA, rfCMA, scdaCMA, shrinkldaCMA, svmCMA

7. Evaluate the results from 6 using `evaluation` and/or compare the different results using the `compare` function.

In practice in order to implement the workflow a series of decisions must be taken. This means that one has to decide:

- Which methods to use for building learning sets.
- Which methods to use for selecting variables with best discriminating power.
- How many variables to use when building those classifiers that cannot decide this by themselves.
- Which classifiers to build so that the set of classifiers tested is simultaneosuly comprehensive (represents well different philosopies) and non-redundant (excludes equivalent or very similar methods).

This can be done using a nested loop that applies each gene selection method for each sample size each learning set and each classifier.

1.2. The data for the analysis

Although the goal of this document is to be as general as possible it is good to recall that not all datasets are suitable for classification analysis, due mainly to the fact that problems in the data can badly affect the performance of predictors obtained.

That is if the data to be used shows batch effects, too many variables or outliers, this has to be dealt with before attempting to build and compare the predictors.

Some typical preprocessing that may have to be dealt with are:

1. remove outliers,
2. keep only groups of samples where there were individuals of both classes analyzed and
3. remove batch effects attributable to technical questions or to experimental design
4. filter the data to retain only genes with a "minimum"variability (that is remove "flat"features).

Although a Bioconductor package CMA uses no Bioconductor structures that is, it expects the data to be in a numerical matrix and the labels to be in a character (or factor) vector.

Indeed for coherence with many standard **R** packages CMA assumes that variables are in columns and samples in rows, that is, it works with what would be the transposed of standard expression matrices.

In order to keep this document as much general as possible the data used to illustrate the following sections are one of those contained in the CMA package, the `khan` dataset. It consists of expression values obtained from small blue round cell tumour which comprises 65 samples from four tumour classes.

```
require(CMA)
```

```
## Loading required package: CMA
## Loading required package: Biobase
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB
## The following object is masked from 'package:stats':
##
##    xtabs
## The following objects are masked from 'package:base':
##
##    anyDuplicated, append, as.data.frame, as.vector, cbind,
colnames,
##    do.call, duplicated, eval, evalq, Filter, Find, get,
intersect,
##    is.unsorted, lapply, Map, mapply, match, mget, order,
paste, pmax,
##    pmax.int, pmin, pmin.int, Position, rank, rbind, Reduce,
rep.int,
##    rownames, sapply, setdiff, sort, table, tapply, union,
unique,
##    unlist
## Welcome to Bioconductor
##
##    Vignettes contain introductory material; view with
##    'browseVignettes()'. To cite Bioconductor, see
##    'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
data(khan)
nabX <- as.matrix(khan[,-1])
dim(nabX)
```

```
## [1]   63 2308
```

```
nabY <- labs<-as.factor(khan[,1])
table(nabY)
```

```
## nabY
##  BL EWS  NB RMS
##   8  23  12  20
```

2. Building and validating the predictors

2.1. Creating the learning datasets

Ideally in classification problems one should have a completely independent test set where the classifier could be checked once it has been built on the train data.

Given that it is not usually possible an alternative approach is to perform *cross-validation* which consists of creating a certain number of splits of the data (that is different divisions of the original dataset into a train and test subset) which can later be used to train/test the predictor and whose results can be aggregated.

There are different schemes for splitting the samples into test and train subsets. Here we use the "five-fold.ªnd "Monte Carlo Cross-Validation.ªpproaches, described in [**?**].

```
#### NOTA: AL FER EL CÀLCUL DEFINITIU CONVE AUGMENTAR EL NOMBRE D'ITERACIONS. (p
numIter <- 3
numFold <- 5
learnSetNames <- c("fiveFold", "MCCV")
set.seed(1234567)
```

2.2. Selecting genes

The first important step in the process of building a classifier is *variable selection*. It is very important, however, not to confound variable selection with classification. Gene selection -or variable selection- is "only" intended to select appropriate variables, but tells nothing about classifiers. Classification relies on variables that distinguish well samples but it looks for a more complex information, that is the ability to classify new individuals into either groups.

Following the recommendations in [**?**] it is a good idea to use several, different, variable selection methods that rely on different approaches, Here we try three gene selection methods: *T-test*, *Random forest* and the *Lasso* corresponding to three different approaches (...).

The code below shows how to perform gene selection on the different learning sets and how to annotate compare them.

This provides relevant information, but it is important to recall that it does not provide us with what is the goal of this document: a good classifier.

```
# selMethodNames <- c("t.test", "rfe", "lasso") # VALID PER DOS GRUPS
# selScheme <- "pairwise"                       # VALID PER DOS GRUPS
selMethodNames <- c("f.test", "rf")           # ADIENT PER MES DE 2 GRUPS
schemeName <- "multiclass"                     # ADIENT PER MES DE 2 GRUPS
numGenes2Sel <- c(2,5,10, 25)
```

```
# Això no cal fer-ho perque es fa al loop principal
geneSels<- list()
for (i in 1:length(learningSets)){
  for (j in 1:length(selMethodNames)){
    selected  <- GeneSelection(nabX, nabY, learningsets = learningSets[[i]],
                               method = selMethodNames[j], scheme=schemeName)
    itemName<- paste(learnSetNames[i], selMethodNames[j], sep=".")
    geneSels[[itemName]]<-selected
  }
}
```

```
## GeneSelection: iteration 1
## GeneSelection: iteration 2
## GeneSelection: iteration 3
## GeneSelection: iteration 4
## GeneSelection: iteration 5
## GeneSelection: iteration 6
## GeneSelection: iteration 7
## GeneSelection: iteration 8
## GeneSelection: iteration 9
## GeneSelection: iteration 10
## GeneSelection: iteration 11
## GeneSelection: iteration 12
## GeneSelection: iteration 13
## GeneSelection: iteration 14
## GeneSelection: iteration 15
## GeneSelection: iteration 1

## Warning in library(package, lib.loc = lib.loc, character.only
= TRUE, logical.return = TRUE, : there is no package called
'randomForest'
## Error in selfun(X, y, learnind = learnmatrix[i, ], ...):
could not find function "randomForest"
```

```
selectedGenesFileName <- paste("selectedGenes",numIter,"iter.Rda", sep="")
save(geneSels, file=file.path(resultsDir,selectedGenesFileName))
```

```
# Si no s'ha fet lo de dalt no tes sentit
if (!(exists("learningSets"))) load(file=file.path(resultsDir, learningSetsFileN
if (!(exists("geneSels"))) load(file=file.path(resultsDir, selectedGenesFileName
topLists <- lapply(geneSels, toplist, 25)
```

```
## top  25  genes for iteration  1
##
##    index importance
```

```
## 1   1389     61.486
## 2    246     61.435
## 3    545     50.657
## 4   1955     50.308
## 5    842     42.971
## 6   1954     39.999
## 7   2050     38.351
## 8   1003     37.919
## 9    107     36.631
## 10  1194     36.296
## 11  1319     35.269
## 12   129     33.405
## 13   509     33.352
## 14   187     32.890
## 15  1066     32.807
## 16  1158     32.435
## 17   255     30.925
## 18  2046     30.462
## 19   742     30.039
## 20     1     30.013
## 21  1427     29.658
## 22  1980     29.240
## 23  1207     28.783
## 24  1645     27.718
## 25  1708     27.107
```

```
res25<- as.data.frame(topLists)
colnames(res25)
```

```
## [1] "fiveFold.f.test.index"      "fiveFold.f.test.importance"
```

```
res.ftest <- c(res25[,1], res25[,5])
```

```
## Error in `[.data.frame`(res25, , 5): undefined columns selected
```

```
res.rf <-  c(res25[,3], res25[,7])
```

```
## Error in `[.data.frame`(res25, , 3): undefined columns selected
```

```
res.all <- c(res.ftest, res.rf)
```

```
## Error in eval(expr, envir, enclos): object 'res.ftest' not
found
```

```
table(res.ftest)
```

```
## Error in eval(expr, envir, enclos): object 'res.ftest' not
found
```

```
table(res.rf)
```

```
## Error in eval(expr, envir, enclos): object 'res.rf' not
found
```

```
table(res.all)
```

```
## Error in eval(expr, envir, enclos): object 'res.all' not
found
```

```
x<-as.data.frame(sort(table(res.all), decreasing=TRUE))
```

```
## Error in as.data.frame(sort(table(res.all), decreasing =
TRUE)): error in evaluating the argument 'x' in selecting a
method for function 'as.data.frame': Error in sort(table(res.all),
decreasing = TRUE) :
##  error in evaluating the argument 'x' in selecting a method
for function 'sort': Error in eval(expr, envir, enclos) : object
'res.all' not found
## Calls: table ->standardGeneric ->eval ->eval ->eval
```

```
# require(annotate)
# require("hgu133plus2.db")
# gname <- function(x) unlist(mget(x, hgu133plus2GENENAME))
# gsymb <- function(x) getSYMBOL(x, "hgu133plus2.db")
# genIdxs <- rownames(x)
# geneNames <- colnames(nabX)
# myGeneNames <- geneNames[as.integer(genIdxs)]
# myGeneSymbols <- gsymb(myGeneNames)
# myGeneDesc <- gname(myGeneNames)
# selectedTable <- cbind(Gene=myGeneSymbols, as.integer(x[,1]), Desc=myGeneDesc)
colnames(x) <- timesSelected
```

```
## Error in eval(expr, envir, enclos): object 'timesSelected'
not found
```

```
selectedTable <- x
```

```
## Error in eval(expr, envir, enclos): object 'x' not found
```

```
biomarkersFileName <- paste("candidate.biomarkers", numIter, "text", sep=".")
write.table(selectedTable, file=file.path(resultsDir, biomarkersFileName), sep="
```

```
## Error in is.data.frame(x): object 'selectedTable' not found
```

File candidate.biomarkers.3.text contains a table with genes selected by the different methods and the number of times they have been selected.

2.3. Hyperparameter tuning

Some methods require -it is recommended- that a tuning of their parameters is performed to yield their best performance. To avoid overfitting this tuning is performed inside the cross-validation loop created to test each classifier on each set of (selected) variables and each set of randomly selected trainig samples.

2.4. Classification

Once all the elements are ready that is: cross-validation scheme, gene selection methods and hyperparameter tunings needed known a global cross-validation loop implementing the process can be builit. The CMA package has some functions that strongly facilitate this process as shown in the code below.

```
load(file=file.path(resultsDir, learningSetsFileName))
# load(file=file.path(resultsDir, selectedGenesFileName)) NO CAL: Es recalcula

classifierNames <- c("dldaCMA", "knnCMA", "rfCMA", "scdaCMA", "svmCMA")
isTunable <- c(FALSE, TRUE, FALSE, TRUE, TRUE)

#classifierNames <- c("dldaCMA", "knnCMA")
#isTunable <- c(FALSE, TRUE)

classifs <- list()

st <- system.time(
for (i in 1:length(learningSets)){
  for (j in 1:length(selMethodNames)){
      selected  <- GeneSelection(nabX, nabY, learningsets = learningSets[[i]],
                                 method = selMethodNames[j])
      for (numGenes in numGenes2Sel){ # Opcional : Un altre nivell d'iteració
        for (k in 1:length(classifierNames)){
          myClassifier <- eval(parse(text=classifierNames[k]))
          if(isTunable[k]){
               tuneVals <- tune (X=nabX, y=nabY, learningsets= learningSets[[i]]
                                 genesel=selected, nbgene=numGenes,
                                classifier =myClassifier,  grids=list())
               classif <- classification(X = nabX, y=nabY, learningsets = learni
                                         genesel=selected, nbgene=numGenes,
                                         classifier=myClassifier,
                                         tuneres=tuneVals)
             }else{
               classif <- classification(X = nabX, y=nabY, learningsets = learni
                                         genesel=selected, nbgene=numGenes,
                                         classifier=myClassifier)
             }
```

```
          itemName<- paste(learnSetNames[i], selMethodNames[j], numGenes, classi
          classifs[[itemName]]<- classif
        }
      }
    }
}
)

## Warning in tune(X, y = as.numeric(y) - 1, learningsets =
learningsets, genesel = genesel, : Combination of feature selection
and hyperparameter tuning
##                 is subject to pessimistic bias and will be
fixed in a future
##                 package version.
## Warning: package 'class' was built under R version 3.1.3
## Warning in library(package, lib.loc = lib.loc, character.only
= TRUE, logical.return = TRUE, : there is no package called
'randomForest'
## Error in classifier(models = FALSE, X = structure(c(1.183075979,
0.545111102, : could not find function "randomForest"

cat("Time consumed: ", st, "\n")

## Error in cat("Time consumed: ", st, "\n"): object 'st' not
found

classifsFileName <- paste("classifs",numIter,"iter.Rda", sep="")
save(classifs, file=file.path(resultsDir,classifsFileName))
```

2.5. Classifiers comparison

The loop performed in the previous section builds and tests many different classifiers. In order to evaluate and compare them they can be processed using the CMA function `compare` which analyzes the performance of each classifier based on standard measures such as "misclassification probability", "sensitivity", "specificity.ºr the "auc" the area under the curve.

Classification results can be plotted or stored in a file for further exploration

```
compMeasures <-  c("misclassification", "sensitivity", "specificity")
                 # , "average probability", "auc")
s1 <- c(rep("fiveF", 60), rep("mccv", 60))
s2<- c(rep("tttest",20), rep ("rfe",20) ,rep("lasso",20),
       rep("tttest",20), rep("rfe",20), rep("lasso",20))
s3 <- rep(c(rep(2,5), rep(5,5),rep(10,5), rep(25,5)),6)
s4 <- rep(classifierNames, 24)
s <- paste(s1,s2,s3,s4, sep=".")
```

```
compClassifs <- compare(classifs,  measure = compMeasures)

## Error in evaluation(clresultlist[[j]], measure = measure[i]):
'sensitivity', 'specificity' or 'auc' are only computed for
binary classification

resultsClassif <- data.frame(CrossVal=s1, VarSel=s2, numGenes=s3, Classif=s4,com

## Error in data.frame(CrossVal = s1, VarSel = s2, numGenes
= s3, Classif = s4, : object 'compClassifs' not found

write.csv2(resultsClassif, file=file.path(resultsDir, paste("resultsClassif", nu

## Error in is.data.frame(x): object 'resultsClassif' not found
```

Referencias