Building and validating predictors using **R** and the CMA package

Alex Sánchez
Estadística i Bioinformàtica
Department d'Estadística. Universitat de Barcelona

14 de noviembre de 2018

Índice

1. Introduction

A common task in bioinformatics is to build and validate predictor -often called a classifier- 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. ([3]) or by [2] and summarized in the follwing two figures.

Figure 1 illustrates the basics blocks on which the biomarker discovery process can be arbitrarily divided. Figure 2 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

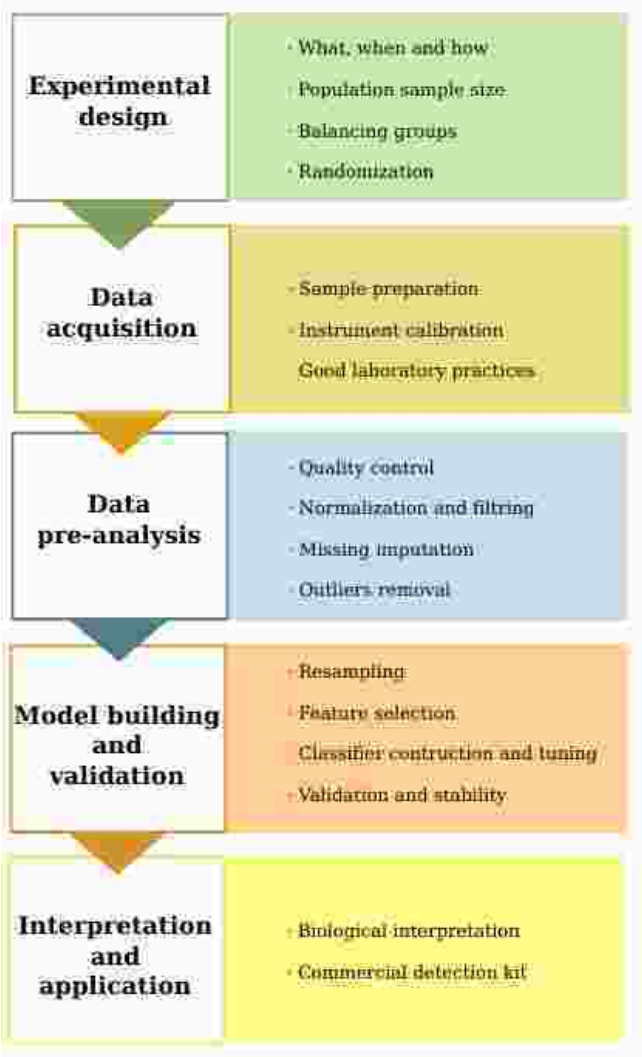

Figura 1: Basics steps in biomarker discovery

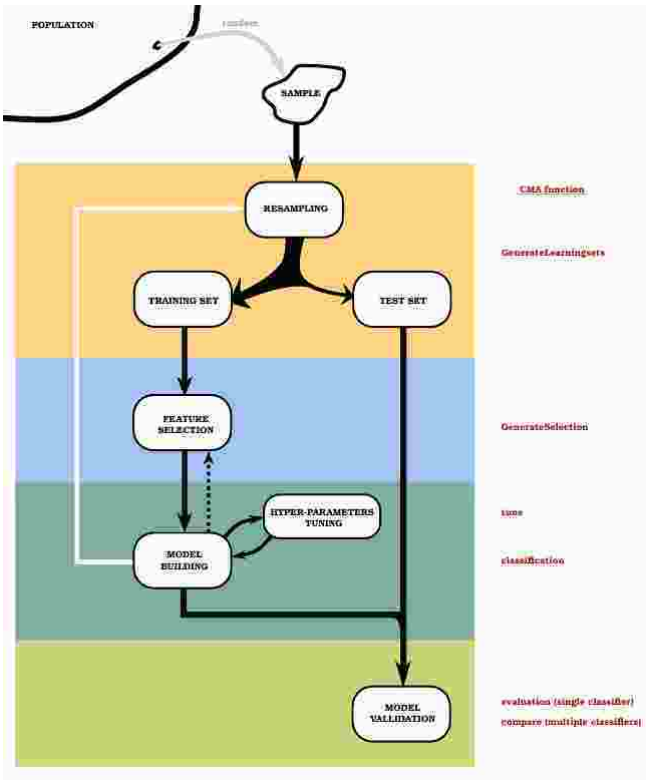

Figura 2: Basic steps of cross-validation

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.

```
## Loading required package: glmnet
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-16
```

The aim of the `CMA` 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 used 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): Do variable selection (using `GeneSelection`).
4. (Optionally): Do 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].
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.

[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

- 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 examples

Although the goal of this document is to be as general as possible it is good to recall that not all datasets are equally 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 has problems such as *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,
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, which cannot reasonably contribute to distinguish classes but which, instead, claerly add noise.

CMA expects the data to consist of:

- a numerical matrix[2],
- a character/factor vector with the labels describing to which group each individual begins.

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.

```
library(CMA)
data(khan)
dim(khan)

## [1]   63 2309
```

[2]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.

```
head(khan[sample(1:nrow(khan),10), 1:5])
```

```
##    khanY          X2          X3          X4         X5
## 21   EWS  1.28550671 -2.24809691 -0.53905364 -1.2743953
## 59    BL -2.68384551 -2.11776666 -0.52323564 -2.1741919
## 3    EWS -0.08446916 -1.64973921 -0.24130752 -2.8752861
## 33   RMS  0.89294022  0.02683665 -0.06262036 -2.2164074
## 28   RMS -0.16193087 -1.02582662  0.03101405  0.2958734
## 49    NB -0.32739357 -2.50470128 -0.44800733  0.4814998
```

```
table(khan$khanY)
```

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

Two-class vs multi-class classifiers Classifiers may be built for problems with two or more than two groups. We start with a simplified example which corresponds to the most common situation, that is when there are two groups. For this we take a subset of the original data, that formed by the "EWS" and the "RMS" individuals.

```
khan2groups <- khan[khan$khanY %in% c("EWS","RMS"),]
nabX <- as.matrix(khan2groups[,-1])
nabY2 <- labs<-as.factor(khan2groups[,1]);
nabY <- droplevels(nabY2 )
table(nabY)
```

```
## nabY
## EWS RMS
##  23  20
```

2. Building and validating the predictors

In the following sections the process of building and validating the predictor, as described in previous section, is applied to the `khan` dataset.

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 in the CMA vignette. Briefly,

- Five-fold Cross Validation divides the data in five parts. Four are used to train the classification model and one is used to test the model. The number of possible decompositions in 4+1 parts depends on the sample size but usually no all possible combinations will be used.
- Monte Carlo Cross-Validation uses a "random splitting" scheme that is the data are randomly divided in two parts of sample sizes 1/5th and 4/5th respectively.

```
#### NOTA: AL FER EL CÀLCUL DEFINITIU CONVE AUGMENTAR EL NOMBRE D'ITERACIONS. (p
numIter <- 10
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 groups.

Following the recommendations in [1] 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*.

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

At the end of the process, once the variables and the method for combining them have been decided the classificator has to be built with all the data.

```
selMethodNames <- c("f.test", "rf") # , "lasso") # VALID FOR TWO GRUPS
selScheme <- "pairwise"                          # VALID FOR TWO GRUPS
schemeName <- "pairwise"
# selMethodNames <- c("f.test", "rf")             # VALID FOR MORE-THAN-TWO GRUPS
# schemeName <- "multiclass"                      # VALID FOR MORE-THAN-TWO GRUPS
numGenes2Sel <- c(5, 10, 25)
```

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

```r
###
### AIXO ES UNA ELABORACIO MANUAL DELS RESULTATS
### S'HAURIA D ECANVIAR PERQUE ES MOLT RIGIDA !!!!!!
###
# Si no s'ha fet lo de dalt no tes sentit
if (!(exists("learningSets")))
  load(file=file.path(resultsDir, learningSetsFileName))
if (!(exists("geneSels")))
  load(file=file.path(resultsDir, selectedGenesFileName))
topLists <- lapply(geneSels, toplist, 25)

## top  25  genes for iteration  1
##
##    index importance
## 1   1389   97.09835
## 2   1955   48.77961
## 3    246   47.04807
## 4    107   42.54279
## 5   1954   41.85440
## 6   1387   41.42712
## 7   1003   39.54160
## 8   1708   37.21509
## 9   1194   36.60658
## 10   129   36.59964
## 11   545   36.25278
## 12  2050   35.62439
## 13     1   35.02198
## 14   783   34.14096
## 15   842   34.10090
```

```
## 16   174   33.06888
## 17   187   31.01996
## 18  1427   29.39010
## 19  2046   29.19195
## 20  1980   27.78957
## 21   509   27.76698
## 22  1645   27.76391
## 23  2162   27.31291
## 24   335   27.20419
## 25  1319   27.20007
## top  25  genes for iteration  1
##
##    index  importance
## 1   1389 0.010982964
## 2    246 0.009659156
## 3   1003 0.009314400
## 4   1772 0.008874875
## 5   1954 0.008581231
## 6   2050 0.007445491
## 7    483 0.006822145
## 8    545 0.006410577
## 9    129 0.006124998
## 10  1319 0.005621616
## 11   976 0.005147524
## 12  1955 0.005116007
## 13  1194 0.004554008
## 14  1911 0.004461585
## 15   187 0.004097203
## 16  2046 0.003906266
## 17  1196 0.003692063
## 18   437 0.003686576
## 19   368 0.003675730
## 20  1055 0.003606028
## 21  1708 0.003498738
## 22  1980 0.003315415
## 23   910 0.003223827
## 24  1110 0.003081059
## 25  1645 0.003066723
## top  25  genes for iteration  1
##
##    index importance
## 1   1389   70.48587
## 2   1194   59.48124
## 3   1003   52.24987
## 4    545   41.07762
```

```
## 5    107   39.81661
## 6   1954   39.56724
## 7   2050   37.37128
## 8   1387   37.29369
## 9   1955   35.87890
## 10     1   35.63930
## 11   842   34.88503
## 12   246   32.49957
## 13  2162   29.91169
## 14  1427   28.85126
## 15  1708   27.06293
## 16   437   27.00688
## 17   248   26.90808
## 18   338   26.01910
## 19  2253   25.33343
## 20   129   25.32331
## 21   607   25.29168
## 22  1207   25.16197
## 23  1886   25.11425
## 24  1206   25.10291
## 25   846   25.09625
## top  25  genes for iteration  1
##
##    index  importance
## 1   2050 0.010724003
## 2   1389 0.010722894
## 3    545 0.010036650
## 4   1003 0.008626669
## 5   1954 0.007666014
## 6    174 0.006249042
## 7    246 0.005972040
## 8    607 0.004718919
## 9   1194 0.004493616
## 10  2046 0.004351099
## 11   509 0.004107805
## 12   742 0.004071967
## 13   338 0.004014911
## 14   187 0.004014555
## 15   842 0.003668231
## 16  2253 0.003265173
## 17  1074 0.003131532
## 18  1207 0.002992262
## 19  1416 0.002989426
## 20   469 0.002972238
## 21   380 0.002928716
```

```
## 22   951 0.002895542
## 23  1911 0.002747894
## 24  1708 0.002669708
## 25   566 0.002658737

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

## [1] "fiveFold.f.test.index"      "fiveFold.f.test.importance"
## [3] "fiveFold.rf.index"          "fiveFold.rf.importance"
## [5] "MCCV.f.test.index"          "MCCV.f.test.importance"
## [7] "MCCV.rf.index"              "MCCV.rf.importance"

res.ftest <- c(res25[,1], res25[,5])
res.rf <-  c(res25[,3], res25[,7])
res.all <- c(res.ftest, res.rf)
table(res.ftest)

## res.ftest
##    1  107  129  174  187  246  248  335  338  437  509  545  607  783  842
##    2    2    2    1    1    2    1    1    1    1    1    2    1    1    2
##  846 1003 1194 1206 1207 1319 1387 1389 1427 1645 1708 1886 1954 1955 1980
##    1    2    2    1    1    1    2    2    2    1    2    1    2    2    1
## 2046 2050 2162 2253
##    1    2    2    1

table(res.rf)

## res.rf
##  129  174  187  246  338  368  380  437  469  483  509  545  566  607  742
##    1    1    2    2    1    1    1    1    1    1    1    2    1    1    1
##  842  910  951  976 1003 1055 1074 1110 1194 1196 1207 1319 1389 1416 1645
##    1    1    1    1    2    1    1    1    2    1    1    1    2    1    1
## 1708 1772 1911 1954 1955 1980 2046 2050 2253
##    2    1    2    2    1    1    2    2    1

table(res.all)

## res.all
##    1  107  129  174  187  246  248  335  338  368  380  437  469  483  509
##    2    2    3    2    3    4    1    1    2    1    1    2    1    1    2
##  545  566  607  742  783  842  846  910  951  976 1003 1055 1074 1110 1194
##    4    1    2    1    1    3    1    1    1    1    4    1    1    1    4
## 1196 1206 1207 1319 1387 1389 1416 1427 1645 1708 1772 1886 1911 1954 1955
##    1    1    2    2    2    4    1    2    2    4    1    1    2    4    3
## 1980 2046 2050 2162 2253
##    2    3    4    2    2
```

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

colnames(x) <- timesSelected

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

selectedTable <- x
biomarkersFileName <- paste("candidate.biomarkers", numIter, "text", sep=".")
write.table(selectedTable,
            file=file.path(resultsDir, biomarkersFileName),
            sep="\t", row.names=FALSE)
```

File candidate.biomarkers.10.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 = learningSets[[i]],
                                         genesel=selected, nbgene=numGenes,
                                         classifier=myClassifier,
                                         tuneres=tuneVals)
             }else{
               classif <- classification(X = nabX, y=nabY,
                                         learningsets = learningSets[[i]],
                                         genesel=selected, nbgene=numGenes,
                                         classifier=myClassifier)
             }
          itemName<- paste(learnSetNames[i], selMethodNames[j],
                           numGenes, classifierNames[k], sep=".")
          classifs[[itemName]]<- classif
        }
      }
    }
}
)
```

```
## Error in model.frame(train.x): argument "train.x" is missing,
with no default
## Timing stopped at: 14.31 0.059 14.38
```

```
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 stan-

dard 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,compClassifs)

## 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",
                                numIter, "csv", sep=".")))

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

Referencias

[1] A.-L. Boulesteix, C. Strobl, T. Augustin, and M. Daumer. Evaluating Microarray-based Classifiers: An Overview. *Cancer Informatics*, 6:77–97, feb 2008.

[2] Darius M. Dziuda. *Data Mining for Genomics and Proteomics: Analysis of Gene and Protein Expression Data*. John Wiley & Sons, jul 2010.

[3] Alex Sánchez-Pla, Ferran Reverter, M Carme Ruíz de Villa, and Manuel Comabella. Transcriptomics: mRNA and alternative splicing. *Journal of neuroimmunology*, 248(1-2):23–31, jul 2012.