

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

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

14 de novembre de 2018

Índice

1. Introduction	1
1.1. The CMA package	1
1.2. The data for the examples	6
2. Building and validating the predictors	7
2.1. Creating the learning datasets	7
2.2. Selecting genes	8
2.3. Hyperparameter tuning	13
2.4. Classification	13
2.5. Classifiers comparison	19

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 al. ([3]) or by [2] and summarized in the following 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

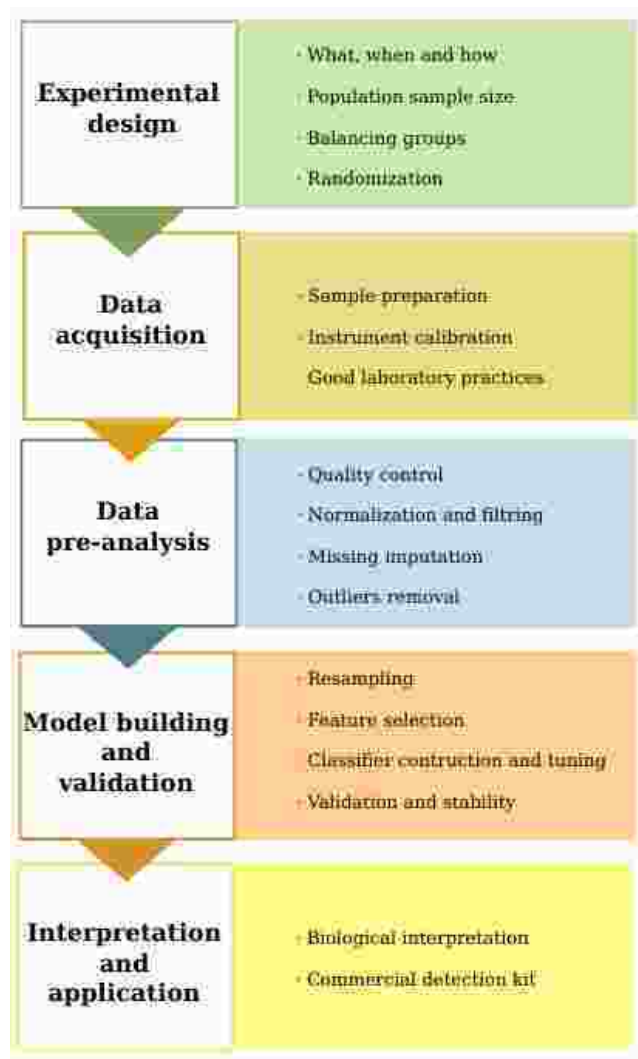


Figura 1: Basics steps in biomarker discovery

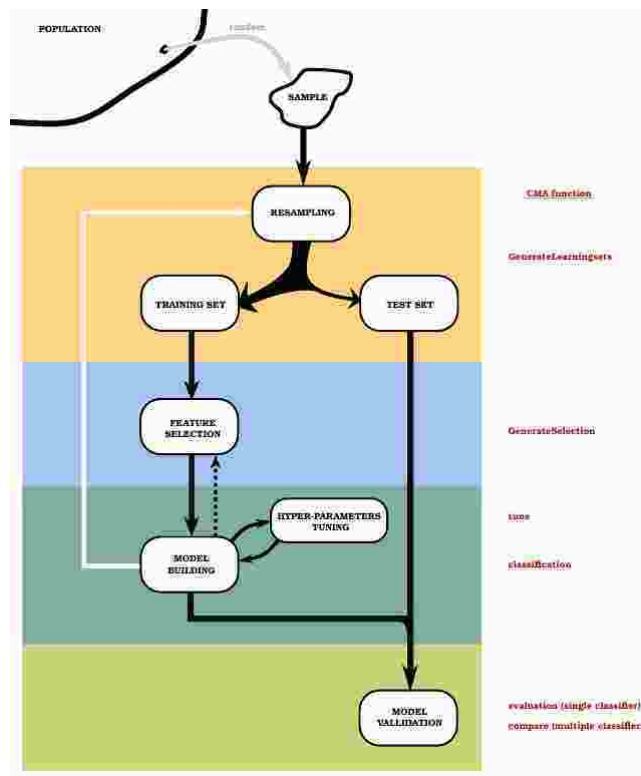


Figura 2: Basic steps of cross-validation

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: e1071
## Loading required package: glmnet
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-16
## Loading required package: randomForest
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
## 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:randomForest':
##
##   combine
## The following objects are masked from 'package:Matrix':
##
##   colMeans, colSums, rowMeans, rowSums, which
## The following objects are masked from 'package:stats':
##
##   IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##   anyDuplicated, append, as.data.frame, basename, cbind,
##   colMeans,
##   colnames, colSums, dirname, do.call, duplicated, eval,
##   evalq,
##   Filter, Find, get, grep, grepl, intersect, is.unsorted,
##   lapply,
##   lengths, Map, mapply, match, mget, order, paste, pmax,
##   pmax.int,
##   pmin, pmin.int, Position, rank, rbind, Reduce, rowMeans,
```

```
rownames,
##   rowSums, sapply, setdiff, sort, table, tapply, union,
unique,
##   unsplit, which, which.max, which.min
## Welcome to Bioconductor
##
##   Vignettes contain introductory material; view with
##   'browseVignettes()'. To cite Bioconductor, see
##   'citation("Biobase)"', and for packages 'citation("pkgname)"'.
##
## Attaching package: 'CMA'
## The following object is masked from 'package:el071':
##
##   tune
```

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. an 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 methods¹.
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.

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

- 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 simultaneously comprehensive (represents well different philosophies) 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, clearly add noise.

CMA expects the data to consist of:

- a numerical matrix²,
- a character/factor vector with the labels describing to which group each individual belongs.

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)
```

²Indeed 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.

```
## [1] 63 2309

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

##      khanY      X2      X3      X4      X5
## 37    RMS 1.24231 0.251148 -1.092133 -2.41575
## 23    EWS 0.83824 -1.833207 -0.547144 -0.68082
## 32    RMS 0.43243 -2.700082 -0.402121 0.13077
## 33    RMS 0.89294 0.026837 -0.062620 -2.21641
## 11    EWS 1.16393 -2.686778 0.044973 -2.08506
## 22    EWS 0.76593 -2.337177 -0.694748 -1.50869

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 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 <- 100
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 classifier 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)
```



```

# Això no cal fer-ho perquè 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))

```

```

###
### 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      187      211.741
## 2     1955      134.787
## 3     1003      116.126
## 4       509      108.949
## 5     1389      107.964
## 6     1799       98.948
## 7     1954       93.190
## 8       246       86.308
## 9     1645       83.357
## 10    2046       83.096
## 11    2050       81.151
## 12    1319       77.193
## 13    1194       76.488
## 14    1207       65.361
## 15     545       62.017

```

```

## 16 1723      56.373
## 17 1888      50.902
## 18 1074      50.467
## 19 2146      47.738
## 20 1911      47.368
## 21  129      41.815
## 22  229      40.125
## 23 1327      39.663
## 24 1924      38.854
## 25  910      36.693
## top 25 genes for iteration 1
##
##      index importance
## 1   1955  0.0102445
## 2   1003  0.0091650
## 3    509  0.0089284
## 4    187  0.0081571
## 5   1645  0.0074473
## 6   1723  0.0070011
## 7   1799  0.0066217
## 8   1389  0.0066003
## 9    338  0.0065883
## 10  1194  0.0058161
## 11   747  0.0057300
## 12  1319  0.0051658
## 13  2046  0.0049282
## 14   372  0.0044197
## 15  1822  0.0044119
## 16  1994  0.0043991
## 17  1954  0.0043937
## 18  2050  0.0043916
## 19   246  0.0043753
## 20   229  0.0042388
## 21   373  0.0036993
## 22   910  0.0036975
## 23  1924  0.0036749
## 24  2146  0.0033670
## 25   554  0.0032937
## top 25 genes for iteration 1
##
##      index importance
## 1   1955   139.405
## 2    246   104.987
## 3   1954    98.442
## 4    187    79.923

```

```

## 5    1799    78.228
## 6    1003    74.929
## 7    1319    66.630
## 8    2046    66.244
## 9    2050    64.142
## 10   1389    62.535
## 11   1207    55.350
## 12    509    55.185
## 13    229    50.456
## 14    545    49.614
## 15   1911    48.623
## 16   1723    48.535
## 17   1194    45.034
## 18   1645    42.675
## 19   1055    40.801
## 20   2146    40.550
## 21    910    37.459
## 22    129    36.840
## 23   1074    35.724
## 24   1888    33.589
## 25   1093    31.088
## top 25 genes for iteration 1
##
##      index importance
## 1    1003  0.0155981
## 2    1954  0.0115053
## 3    1799  0.0083869
## 4     187  0.0082061
## 5     229  0.0066356
## 6     246  0.0059944
## 7    1389  0.0058111
## 8     545  0.0050828
## 9    1207  0.0049359
## 10   1319  0.0044566
## 11   1955  0.0043709
## 12   1911  0.0043636
## 13   1980  0.0042606
## 14     509  0.0040058
## 15   1888  0.0039683
## 16   2050  0.0038088
## 17   2247  0.0036606
## 18     982  0.0036556
## 19     910  0.0036427
## 20   1194  0.0035713
## 21   2046  0.0035273

```

```
## 22 1158 0.0035219
## 23 1074 0.0034313
## 24 1110 0.0033778
## 25 566 0.0031581

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

## res.fctest
## 129 187 229 246 509 545 910 1003 1055 1074 1093 1194 1207 1319 1327 13
## 2 2 2 2 2 2 2 2 1 2 1 2 2 2 1
## 1645 1723 1799 1888 1911 1924 1954 1955 2046 2050 2146
## 2 2 2 2 2 1 2 2 2 2 2

table(res.rf)

## res.rf
## 187 229 246 338 372 373 509 545 554 566 747 910 982 1003 1074 11
## 2 2 2 1 1 1 2 1 1 1 1 2 1 2 1
## 1158 1194 1207 1319 1389 1645 1723 1799 1822 1888 1911 1924 1954 1955 1980 19
## 1 2 1 2 2 1 1 2 1 1 1 1 2 2 1
## 2046 2050 2146 2247
## 2 2 1 1

table(res.all)

## res.all
## 129 187 229 246 338 372 373 509 545 554 566 747 910 982 1003 10
## 2 4 4 4 1 1 1 4 3 1 1 1 4 1 4
## 1074 1093 1110 1158 1194 1207 1319 1327 1389 1645 1723 1799 1822 1888 1911 19
## 3 1 1 1 4 3 4 1 4 3 3 4 1 3 3
## 1954 1955 1980 1994 2046 2050 2146 2247
## 4 4 1 1 4 4 3 1

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.100.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 training 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 built. 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,
                               tuner=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
  }
}
}
)

## 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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.

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

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

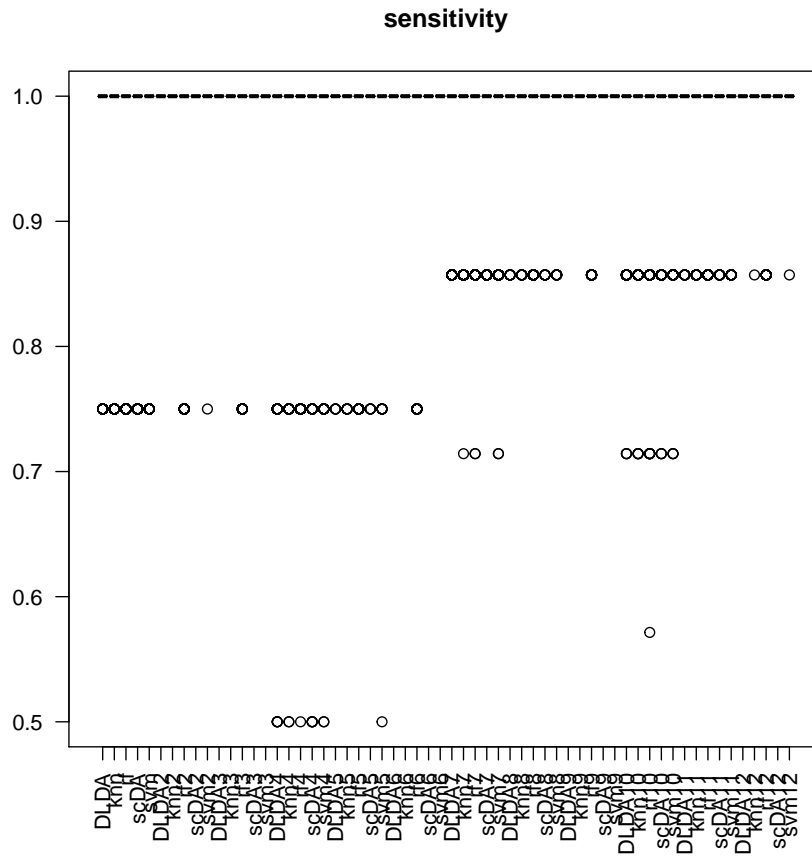
2.5. Classifiers comparison

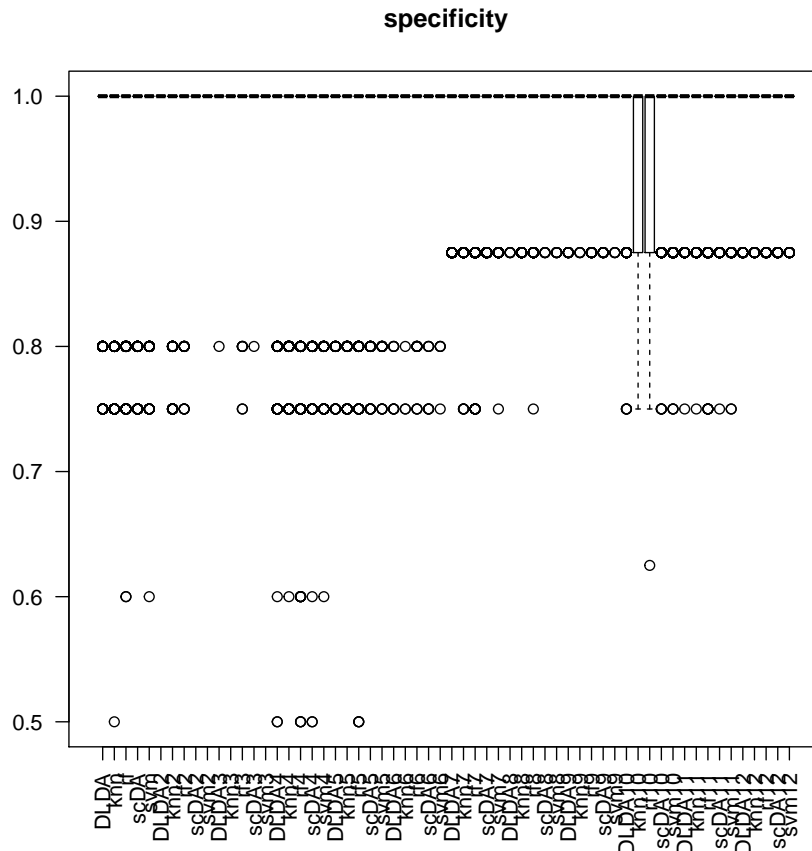
The loop performed in the previous section builds and tests many different classifiers. In order to evaluate and compare them they can be processed using the CMA function compare which analyzes the performance of each classifier based on standard measures such as "misclassification probability", "sensitivity",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



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

## Warning in data.frame(CrossVal = s1, VarSel = s2, numGenes
## = s3, Classif = s4, : row names were found from a short variable
## and have been discarded

write.csv2(resultsClassif,
            file=file.path(resultsDir,
                            paste("resultsClassif",
                                    numIter, "csv", sep=".")))

```

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.