

Magpie R package 0.2.0 User Manual

Camille Maumet

July 14, 2008

Contents

1	Introduction	3
2	Installation	4
3	Quick Start	5
3.1	Introduction	5
3.2	Pre-processing	6
3.2.1	Format the output classes file	6
3.2.2	Format the genes expression file	6
3.3	Define your experiment	7
3.3.1	Load your dataset	7
3.3.2	Store your feature selection options	8
3.3.3	Store the options related to your experiment	9
3.4	Run one-layer and two-layer cross-validation	11
3.4.1	External One-Layer Cross Validation	11
3.4.2	two-layer Cross Validation	12
3.5	Classify new samples	13
4	Access the results of one-layer and two-layer cross-validation	16
4.1	Introduction	16
4.2	Argument of the method getResults	16
4.3	Error rates	17
4.3.1	Optional argument errorType	17
4.3.2	Examples	17
4.4	Genes selected	17
4.4.1	Optional argument genesType	17
4.4.2	Examples	18
4.5	Best value of option	18
4.5.1	Overview	18
4.5.2	Examples	18
4.6	Execution time	18
4.6.1	Overview	18
4.6.2	Examples	18
5	Plots and graphics	19
5.1	Plot the cross-validated error rates of one-layer cross-validation	19
5.1.1	Plot the summary error rate only	19
5.1.2	Plot the summary error rate only	19
5.2	Plot the fold error rates of two-layer cross-validation	20
5.2.1	Plot the summary error rate only	20
	Bibliography	20

Chapter 1

Introduction

This package provides classes and methods to train classifiers and to estimate the predictive error rate of classifiers using external one-layer cross-validation and two-layer cross-validation . These two techniques of cross-validation have been presented respectively in [1] and [5], [8], [7]. One-layer cross-validation can be used to determine a nearly unbiased estimate of the error rate in a context of feature selection. The feature selection is performed for different sizes of subsets of genes and the corresponding error rate is estimated by cross-validation. As an output of this one-layer cross-validation, the user gets a cross-validated error rate per size of subset. However, if the user wants to know the smallest estimated error rate over all the subsets considered, then a second layer of cross-validation is required to estimate the effect of this choice.

Chapter 2

Installation

The Magpie package will be soon available online but, at the moment it has to be installed manually. To this aim, once you have got a copy of the windows zip file containing the package, you must open R and in the menu **Packages**, click on **Install package(s) from Local zip file...** Select the zip file containing the package. If the installation fails, you might have to install manually the following packages before:

- BioConductor using the following command lines in R:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- The MLInterfaces and Biobase packages using the following command lines in R:

```
biocLite("MLInterfaces")
biocLite("Biobase")
```

- The R packages: 'e1071', 'sma', 'pamr', 'kernlab' using the following command lines in R:

```
install.packages(c('e1071', 'sma', 'pamr', 'kernlab'))
```

Then, try again to install the magpie zip file from the menu. If it still does not work please do not hesitate to report the error.

You are now ready to load the magpie package with the following R command and start using Magpie:

```
library('magpie')
```

Chapter 3

Quick Start

3.1 Introduction

This section presents a quick review of the package functionality by giving an example. We will use two files `raw_geneExpr.txt` and `raw_classes.txt` that contain the gene expression levels of 20 genes and the output classes of 10 samples. The samples, labeled S1 to S10, come from two classes labeled A and B. The first column of the gene expression file contains the name of the genes.

```
raw_classes.txt
```

```
S1 A
S2 B
S3 A
S4 A
S5 B
S6 B
S7 A
S8 A
S9 A
S10 B
```

```
raw_geneExpr.txt
```

```
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
211316_x_at 193.7 131 287 187.8 201.9 314.9 501.1 340.1 580.7 333.1
201947_s_at 905.1 1268 633.6 623.2 988.6 813.7 647.7 808.5 703.9 473.5
208018_s_at 77.8 48.1 275.2 84.1 57.9 78.5 145.6 38.2 102.6 131.7
208884_s_at 164.3 157.1 280.7 297.8 238.1 171.7 243.9 171.6 168.3 73.7
218251_at 251.2 250.4 205.1 129.8 256.7 189.9 199.2 329.2 316.7 542.1
220712_at 165.8 112.9 49.7 113.4 103 123 143.9 111.6 69.2 159.5
34764_at 46.9 56.2 36.4 74.6 81.1 100.1 43.3 74 56.4 68.9
217754_at 269.2 176.1 314 137.5 269.6 177.7 107.6 176.7 98.4 146
221938_x_at 203.9 197.8 189.6 141.2 108 221.3 205.7 180.1 112.4 278
209492_x_at 711.7 787.6 652.4 921.1 863.6 1129.2 789.1 999.6 805.9 1201.3
211596_s_at 142.5 211.8 147.6 214.8 132.2 135 90.2 116.1 208.4 163.9
221925_s_at 78.5 103 130 157.3 116.4 137 133.2 85.7 148.9 29.6
200804_at 3544.6 3120.1 2430.5 2761.5 2721.3 2328.2 2212.7 2403.2 2397 1655.6
206529_x_at 17.3 22.5 27.4 31.9 1.3 1.8 336.8 16.3 68.3 33
213224_s_at 37.7 60 48.3 83.9 104.6 12 71 28.3 77.6 7.5
215628_x_at 315.7 370.5 396.1 334 321.9 361.3 331.8 380.9 342.2 558.3
211362_s_at 56 56.4 84.3 19.7 146.9 63.8 34.1 50.2 35.3 69
221058_s_at 64.2 79.6 97.8 89.4 135.4 49.5 81 126.8 39 31.2
210381_s_at 72.8 65 31.8 74.4 64.7 124.3 32.4 15.5 80.9 84.6
```

216989_at 7.1 36.3 24.1 25.3 45.6 54.4 7.8 54.6 5.5 16.8

3.2 Pre-processing

The aim of the Magpie package is not to provide a way of pre-processing your micro-array data. Other packages already provide this functionality and we assume that we are working on pre-processed data. The pre-processing step that is undertaken here aims to convert the data files to the right format to be able to use them in Magpie and provides the possibility of normalizing over the samples and genes. This format is, by the way, very simple and you might not need to use the following function. In this case, you can go directly to section 3.3. The two functions `formatClasses` and `formatGenesExpr` can be used as helpers to convert respectively the output classes file and the gene expression file.

3.2.1 Format the output classes file

The following command is used to format our output classes file.

```
> formatClasses( classDataFile="pathToFile/raw_classes.txt",
                 outClassDataFile="pathToFile/formated_classes.txt",
                 sampleNames=TRUE,
                 vertical=TRUE)
```

This command creates a new file situated at `outClassDataFile`, in our case in the folder `pathToFile` with the name `formated_classes.txt`. `sampleNames` is set to `TRUE` which means that the names of the samples are available in the entry file. And `vertical` is `TRUE` since the output classes are presented in a column. By default, the function assumes that the class labels are separated by a blank character (blanks or tabs, etc) but you can choose your own separator, say a comma, by adding another parameter to the function: `separator=","`. The output file is as follows.

formated_classes

```
"S1" "S2" "S3" "S4" "S5" "S6" "S7" "S8" "S9" "S10"
"type" "A" "B" "A" "A" "B" "B" "A" "A" "A" "B"
```

3.2.2 Format the genes expression file

The following command is used to format our gene expression file.

```
> formatGenesExpr( geneExprDataFile="pathToFile/raw_geneExpr.txt",
                   outGeneExprDataFile="pathToFile/formated_geneExpr.txt",
                   rowNames=TRUE, colNames=TRUE,
                   transpose=FALSE,
                   normalize=TRUE, lineNorm=TRUE, colNorm=TRUE,
                   firstLineNorm=FALSE)
```

This command creates a new file situated at `outGeneExprDataFile`, in our case in the folder `pathToFile` with the name `formated_geneExpr.txt`. `rowNames` is set to `TRUE` which means that names are available on the first row of the entry file (in our case the names of the samples). `colNames` is set to `TRUE` which means that names are available on the first column of the entry file (in our case the names of the genes). `transpose` is `FALSE` which indicates that each row corresponds to a gene and each column to a sample. By default, the function assumes that the gene expression values are separated by any blank character (blank, tab, horizontal tab...) but you can choose your own separator, say a comma, by adding another parameter to the function: `separator=","`. `normalize` is set to `TRUE` which indicates that we want to normalize the gene expression over the rows (`lineNorm` set to `TRUE`) and the columns (`colNorm` set to `TRUE`), starting with the column normalization (`firstLineNorm` set to `FALSE`). The output, considering only three decimal places, is as follows.

```
> round(read.table("pathToFile/formated_geneExpr.txt"), digits=3)
S1    S2    S3    S4    S5    S6    S7    S8    S9    S10
211316_x_at 0.131 0.095 0.270 0.157 0.166 0.287 0.496 0.308 0.538 0.369
201947_s_at 0.281 0.424 0.275 0.241 0.376 0.342 0.295 0.337 0.300 0.242
208018_s_at 0.143 0.095 0.703 0.191 0.130 0.195 0.391 0.094 0.258 0.396
208884_s_at 0.192 0.197 0.457 0.432 0.340 0.271 0.418 0.269 0.270 0.141
218251_at   0.195 0.209 0.221 0.125 0.243 0.199 0.226 0.342 0.337 0.689
220712_at   0.328 0.241 0.137 0.279 0.249 0.329 0.418 0.297 0.188 0.518
34764_at     0.168 0.217 0.182 0.332 0.355 0.485 0.228 0.356 0.277 0.405
217754_at   0.333 0.235 0.542 0.212 0.408 0.298 0.196 0.294 0.167 0.297
221938_x_at 0.248 0.259 0.321 0.213 0.161 0.364 0.367 0.294 0.188 0.555
209492_x_at 0.183 0.218 0.234 0.295 0.272 0.394 0.298 0.346 0.285 0.508
211596_s_at 0.215 0.343 0.310 0.402 0.243 0.275 0.199 0.235 0.431 0.405
221925_s_at 0.160 0.226 0.370 0.399 0.291 0.379 0.399 0.235 0.418 0.099
200804_at   0.342 0.324 0.327 0.331 0.321 0.304 0.314 0.311 0.318 0.262
206529_x_at 0.034 0.047 0.075 0.078 0.003 0.005 0.969 0.043 0.184 0.106
213224_s_at 0.152 0.261 0.272 0.421 0.516 0.066 0.421 0.153 0.430 0.050
215628_x_at 0.193 0.243 0.337 0.253 0.240 0.298 0.297 0.312 0.287 0.559
211362_s_at 0.194 0.211 0.408 0.085 0.623 0.299 0.174 0.234 0.168 0.393
221058_s_at 0.183 0.244 0.389 0.317 0.472 0.191 0.339 0.485 0.153 0.146
210381_s_at 0.243 0.234 0.148 0.309 0.264 0.562 0.159 0.070 0.371 0.464
216989_at   0.052 0.287 0.247 0.231 0.409 0.541 0.084 0.538 0.055 0.203
```

3.3 Define your experiment

Now that our data is ready, we can go further and specify the options of our classification task. To this end, we create three objects. An object of class **dataset** to store the microarray data, an object of class **featureSelectionOptions** to store the options relative to the feature selection process. And finally an **experiment** object which stores all the information needed before starting the classification task.

3.3.1 Load your dataset

Since we have already created the data files in the previous section the creation of the **dataset** object is fairly simple. The following command creates the dataset and loads the data from the files.

```
> myDataset <- new( "dataset",
                    dataId="exampleData",
                    geneExprFile="formated_geneExpr.txt",
                    classesFile="formated_classes.txt",
                    dataPath=file.path("pathToFile"))
```

The creation of the dataset object is undertaken by calling the function **new("dataset")** with the following arguments:

- **dataId**, an id for your dataset
- **geneExprFile**, name of the file in which the gene expression values are stored
- **classesFile**, name of the file in which the output classes are stored
- **dataPath**, the path to the folder where the gene expression file and the output file are stored

If you have a look at your dataset, you will notice that the slot **eset** is **NULL**, you have to load the data manually before inserting it in the **experiment** object.

```
dataId
exampleData
dataPath
```

```

pathToFile
geneExprFile
  formatted_geneExpr.txt
classesFile
  formatted_classes.txt
eset: use 'getEset(object)'

```

3.3.2 Store your feature selection options

Two feature selection methods are currently available, the Recursive Feature Elimination (RFE) based on Support Vector Machine (SVM), as presented in [3] and the Nearest Shrunken Centroid (NSC) as described in [6]. The former can be used by creating an object of class `geneSubsets` and the latter by creating an object of class `thresholds`.

RFE-SVM as a method of feature selection

The object of class `geneSubsets` is meant to store the information relative to the subsets of genes that must be considered during forward selection by the RFE. Basically, you must specify the sizes of the subsets that should be considered. There are three easy ways to reach this goal.

The easiest way is to keep the default values which corresponds to subsets of size one to the number total of features by powers of two. If you want to use this default `geneSubsets`, you can ignore the current section and go directly to section 3.3.3.

Another solution is to define the size of the biggest subset to be considered and the speed of the RFE: `high` or `slow`. By default the speed is set to `high`. This means, as proposed in [3], that the biggest subset is considered first, then a subset of size equal to the greatest power of two smaller than the biggest size and then decreasing by a powers of two until reaching a single feature. With a `slow` value for speed the size of the subsets decreases by one at each step. This methods can produce better results but is highly computationally intensive.

```

> geneSubsets <- new("geneSubsets", speed="high", maxSubsetSize=20)
> geneSubsets
optionValues: 1 2 4 8 16 20 (maxSubsetSize: 20, speed:high, noOfOptions:6)

> geneSubsets <- new("geneSubsets", speed="slow", maxSubsetSize=20)
> geneSubsets
optionValues: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 (maxSubsetSize: 20,
speed:slow, noOfOptions:20)

```

Alternatively, you can also give all the subset sizes that the software must try. For example the following command ask for subsets of size 1,2,3,5,9,10,15,20.

```

> geneSubsets <- new("geneSubsets", speed="high", optionValues=c(1,2,3,5,9,10,15,20))
> geneSubsets
optionValues: 1 2 3 5 9 10 15 20 (maxSubsetSize: 20, speed:high, noOfOptions:8)

```

Be careful not to give a number larger than the actual number of genes in your dataset or an error, with a message similar to the one presented above, will be generated when you will try to incorporate this object into your `experiment`.

```

Error in validObject(.Object) :
  invalid class "experiment" object: The maximum of genes in 'geneSubsets'(70) must not
  be greater than the number of features in 'dataset'(20)

```


NSC as a method of feature selection

The object of class `thresholds` is meant to store the thresholds that must be considered by the nearest shrunken algorithm to determine which one is the best. Basically, you must specify the thresholds that should be considered. There are two easy ways to reach this goal.

The easiest way is to keep the default values which corresponds to the thresholds generated by the function `pamr.train` on the whole dataset, for more details please refer to the `pamr` package documentation. If you want to use this default `thresholds`, you can ignore the current section and go directly to section 3.3.3.

Alternatively, you can specify all the thresholds that must be considered by the software. For example the following command ask for thresholds 0,0.1,0.2,0.3,0.4,0.5,1,2.

```
> thresholds <- new("thresholds", optionValues=c(0,0.1,0.2,0.3,0.4,0.5,1,2))
```

3.3.3 Store the options related to your experiment

The last step in the definition of your experiment is to integrate your dataset, your feature selection options and decide of the options related to the experiment. This aim is reached by creating an object of class `experiment`. The argument must be specified as follows:

- `dataset`, dataset object that we created in section 3.3.1
- `noFolds1stLayer`, number of folds to be created in the inner layer of two-layer cross-validation. 1 corresponds to leave-one-out
- `noFolds2ndLayer`, number of folds to be created in the outer layer of two-layer cross-validation and for the one-layer cross-validation. 1 corresponds to leave-one-out
- `classifierName`, name of the classifier to be used 'svm' (Support Vector Machine), or 'nsc' (Nearest Shrunken Centroid).
- `featureSelectionMethod`, name of the feature selection method: 'rfe' (Recursive Feature Elimination) or 'nsc' (Nearest Shrunken Centroid).
- `typeFoldCreation`, name of the method to be used to generate the folds: 'naive', 'balanced' or 'original'.
- `svmKernel`, name of the feature kernel used both for the SVM as a feature selection method in RFE or for SVM as a classifier: 'linear' (Linear Kernel), 'radial' (Radial Kernel) or 'polynomial' (Polynomial Kernel).
- `noOfRepeats`, cross-validation allocates observations randomly to folds, unless it is leave-one-out cross-validation, repeating the process is likely to give a different result. The final results are then averaged over the repeats. As mentioned in [2], this is believed to improve the accuracy of estimates. `noOfRepeats` is the number of repeats to be done, both for one-layer and two-layer of cross-validation.
- `featureSelectionOptions`, `geneSubsets` or `thresholds` object that we created in section 3.3.2 or `missing` if you want to use the default values

In the next sections, we will work with the dataset `vV70genes` available in the `Magpie` package. Before using it, you must call `data('vV70genesDataset')`, we will use the default `geneSubsets`.

For example, if we want to set the feature selection method as RFE-SVM, with an SVM as classifier, a cross-validation with 10 folds in the outer layer, 9 folds in the inner layer, we can have:

```
> myExperiment <- new ( "experiment",  
                        dataset = vV70genes,  
                        noFolds1stLayer = 9,  
                        noFolds2ndLayer = 10,
```

```

        classifierName = "svm",
        featureSelectionMethod = 'rfe',
        typeFoldCreation = "original",
        svmKernel = "linear",
        noOfRepeats = 3)
> experiment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
typeFoldCreation:original
noOfRepeats:3
featureSelectionOptions
  optionValues: 1 2 4 8 16 32 64 70 (maxSubsetSize: 70, speed:high, noOfOptions:8)
dataset
  dataId
    vantVeer_70
  dataPath: use 'getDataPath(object)'
  geneExprFile: use 'getGeneExprFile(object)'
  classesFile: use 'getClassesFile(object)'
  eset: use 'getEset(object)'

```

No Results for external CV (1 layer)

No Results for 2 layers external CV

Final Classifier has not been computed yet

Similarly, if we want to set the feature selection method as NSC, with a NSC classifier, a cross-validation with 10 folds in the outer layer, 9 folds in the inner layer performed 10 times, we can have:

```

> myExperiment2 <- new ( "experiment",
        dataset = vV70genes,
        noFolds1stLayer = 9,
        noFolds2ndLayer = 10,
        classifierName = "nsc",
        featureSelectionMethod = 'nsc',
        typeFoldCreation = "original",
        noOfRepeats = 2)
> myExperiment2
experiment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:nsc
typeFoldCreation:original
noOfRepeats:2
featureSelectionOptions
  optionValues:0 0.0907303 0.1814605 0.2721908 0.362921 0.4536513 0.5443815 0.6351118 0.725842 0.8
dataset
  dataId
    vantVeer_70
  dataPath: use 'getDataPath(object)'
  geneExprFile: use 'getGeneExprFile(object)'
  classesFile: use 'getClassesFile(object)'
  eset: use 'getEset(object)'

```

No Results for external CV (1 layer)

No Results for 2 layers external CV

Final Classifier has not been computed yet

As we can see from the display of the experiment, the thresholds have been successfully updated.

3.4 Run one-layer and two-layer cross-validation

That was easy, wasn't it? It's now time to run our experiment. Two methods are here to help us in doing this important step: `runOneLayerExtCV` and `runTwoLayerExtCV` for, respectively, computing an external one-layer or an external two-layer cross-validation including feature selection.

3.4.1 External One-Layer Cross Validation

As a reminder, let's just state that the external one-layer cross-validation aims to assess the error rate of a classifier using feature selection in an appropriate manner. At the end of this step we will get a cross-validated error-rate for each size of subset considered. Since all the options have already been chosen via the `experiment` object, the command to start the cross-validation is trivial.

```
> # Necessary to find the same results
> set.seed(234)
> myExperiment <- runOneLayerExtCV(myExperiment)
```

Once the previous command has been run, we can look again at our experiment, the result of one-layer cross-validation has been updated.

```
> myExperiment
experiment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
typeFoldCreation:original
noOfRepeats:3
featureSelectionOptions
  optionValues: 1 2 4 8 16 32 64 70 (maxSubsetSize: 70, speed:high, noOfOptions:8)
dataset
  dataId
    vantVeer_70
  dataPath: use 'getDataPath(object)'
  geneExprFile: use 'getGeneExprFile(object)'
  classesFile: use 'getClassesFile(object)'
  eset: use 'getEset(object)'

resultRepeated1LayerCV
  original1LayerCV: 3 combined 1 layer 10-folds CV: use 'getOriginal1LayerCV(object)'
  summaryFrequencyTopGenes: use 'getFrequencyTopGenes(object)'
  summaryErrorRate:
    cvErrorRate:
      0.3504274 0.3760684 0.3290598 0.2820513 0.2393162 0.1965812 0.2179487 0.2136752
    seErrorRate:
      0.0359811 0.0350716 0.0335795 0.031384 0.02832 0.0261317 0.0278346 0.0277808
    classErrorRates:
      goodPronosis: 0.3106061 0.3712121 0.3106061 0.2575758 0.2045455 0.1742424 0.1969697 0.1969697
      poorPronosis: 0.4019608 0.3823529 0.3529412 0.3137255 0.2843137 0.2254902 0.245098 0.2352941
=> bestOptionValue: 32 (Est. Error rate: 0.1965812)
executionTime: 11.422s
```

No Results for 2 layers external CV

Final Classifier has not been computed yet

From this display we get the key results of the one-layer cross-validation. For more details on how to get the complete results of this cross-validation, you can have a look at section 4. Here we can infer that the best size of subset is 8 with an error rate of 0.4. The cross-validated error rates for subsets of size 1,2,4,8,16,20 are respectively 0.7 0.6 0.6 0.4 0.6 0.6. The standard errors of these cross-validated error rates are respectively 0.0460566 0.0492366 0.0492366 0.0492366 0.0492366 0.0492366. These are calculated by treating the cross-validation error rate as the average of the error rates on each fold. This display also provide the error rate for each class. As we know from [5], [8] and [7], the best error rate is biased and two-layer of cross-validation must be computed to get a unbiased estimate.

3.4.2 two-layer Cross Validation

As a reminder, let's just state that two-layer cross-validation aims to assess the best error rate of a classifier using feature selection in an appropriate manner. At the end of this step we will get an estimate of the best error rate that we can compute using our test set. Since all the options have already been chosen via the `experiment` object, the command to start the two-layer cross-validation is trivial.

```
> myExperiment <- runTwoLayerExtCV(myExperiment)
```

One the previous command has been run, we can look again at our experiment, the result of two-layer cross-validation has been updated.

```
> experiment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
typeFoldCreation:original
noOfRepeats:3
featureSelectionOptions
  optionValues: 1 2 4 8 16 32 64 70 (maxSubsetSize: 70, speed:high, noOfOptions:8)
dataset
  dataId
    vantVeer_70
  dataPath: use 'getDataPath(object)'
  geneExprFile: use 'getGeneExprFile(object)'
  classesFile: use 'getClassesFile(object)'
  eset: use 'getEset(object)'

resultRepeated1LayerCV
  original1LayerCV: 3 combined 1 layer 10-folds CV: use 'getOriginal1LayerCV(object)'
  summaryFrequencyTopGenes: use 'getFrequencyTopGenes(object)'
  summaryErrorRate:
    cvErrorRate:
      0.3504274 0.3760684 0.3290598 0.2820513 0.2393162 0.1965812 0.2179487 0.2136752
    seErrorRate:
      0.0359811 0.0350716 0.0335795 0.031384 0.02832 0.0261317 0.0278346 0.0277808
    classErrorRates:
      goodPronosis: 0.3106061 0.3712121 0.3106061 0.2575758 0.2045455 0.1742424 0.1969697 0.1969697
      poorPronosis: 0.4019608 0.3823529 0.3529412 0.3137255 0.2843137 0.2254902 0.245098 0.2352941
  => bestOptionValue: 32 (Est. Error rate: 0.1965812)
  executionTime: 11.422s
```

```
resultRepeated2LayerCV
  original2LayerCV: 3 combined 2 layer 10-folds CV: use 'getOriginal2LayerCV(object)'
```

```

summaryErrorRate:
  finalErrorRate:
    0.2013024
  seFinalErrorRate:
    0.0323111
  classErrorRates:
    goodPronosis: 0.1515152
    poorPronosis: 0.2941176
=> avgBestOptionValue: 37.86667 (Est. Error rate: 0.2013024)
executionTime: 111.692s

```

Final Classifier has not been computed yet

From this display we get the key results of the two-layer cross-validation. For more details on how to get the complete results of this cross-validation, you can have a look at section 4. Here we can infer that the estimate of the best error rate that we can get is 0.7 for an average value of subset size of 5.7.

3.5 Classify new samples

Another simple method allow us to classify new samples based on our dataset. The final classifier is trained on the whole dataset. By default, it is inferred by considering only the genes obtained by feature selection with the best value of option (size of subset for RFE-SVM or threshold for NSC) found in one-layer cross-validation. You can instead select your favorite option (number of genes or threshold) by specifying it in the arguments. Three steps are involved for the classification of one more more samples. First, you must pre-process your raw data as in section 3.2.2 to get a file containing the gene expression values in which each column corresponds to a sample and each line to a gene. The first row must contain the names of the new samples and the first column the names of the genes. Second, the final classifier must be trained on the whole dataset using only the relevant genes by calling the method `findFinalClassifier`.

```
> myExperiment <- findFinalClassifier(myExperiment)
```

Once the final classifier has been trained we can try it on new samples. Let's use the following file names `vv_NewSamples.txt`, that contains the gene expression values of four new samples.

```

S1new S2new S3new S4new
211316_x_at 0.238549585 0.309818611 0.039801616 0.185127978
201947_s_at 0.062913738 0.348206391 0.049101993 0.018549661
208018_s_at 0.214811858 0.310358253 0.90570071 0.117063616
208884_s_at 0.116130479 0.105216261 0.413862903 0.364270183
218251_at 0.110915852 0.082847135 0.05732792 0.250266178
220712_at 0.156955443 0.018847956 0.22558974 0.058140084
34764_at 0.163686223 0.066543884 0.136281185 0.164491474
217754_at 0.166883529 0.030785639 0.583919806 0.076886967
221938_x_at 0.003795356 0.017734243 0.142946003 0.073167907
209492_x_at 0.13303862 0.063216618 0.031262267 0.089941499
211596_s_at 0.070772013 0.186176953 0.119515381 0.30368565
221925_s_at 0.179151366 0.047201722 0.240400082 0.298616043
200804_at 0.184900824 0.148434291 0.154408075 0.162802706
206529_x_at 0.432168595 0.405113542 0.350297917 0.344634651
213224_s_at 0.195566057 0.021122683 0.04348983 0.341574279
215628_x_at 0.114695588 0.013643621 0.173638361 0.006354456
211362_s_at 0.111345205 0.078990291 0.315260021 0.330167423
221058_s_at 0.133462748 0.011197787 0.278062554 0.134014777
210381_s_at 0.013173383 0.032487425 0.203678868 0.118008774
216989_at 0.395635336 0.073903352 0.006366366 0.038050583

```

The classification task is started by calling `classifyNewSamples`. If the argument

```
> classifyNewSamples( myExperiment,
newSamplesFile="pathToFile/vV_NewSamples.txt")
      V1      V2      V3      V4      V5      V6
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V7      V8      V9     V10     V11     V12
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V13     V14     V15     V16     V17     V18
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V19     V20     V21     V22     V23     V24
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V25     V26     V27     V28     V29     V30
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V31     V32     V33     V34     V35     V36
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V37     V38     V39     V40     V41     V42
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V43     V44     V45     V46     V47     V48
goodPronosis goodPronosis poorPronosis poorPronosis poorPronosis poorPronosis
      V49     V50     V51     V52     V53     V54
poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis
      V55     V56     V57     V58     V59     V60
poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis
      V61     V62     V63     V64     V65     V66
poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis
      V67     V68     V69     V70     V71     V72
poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis
      V73     V74     V75     V76     V77     V78
poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis poorPronosis
Levels: goodPronosis poorPronosis
> classifyNewSamples( myExperiment,
newSamplesFile="pathToFile/vV_NewSamples.txt",
optionValue=1)
      V1      V2      V3      V4      V5      V6
goodPronosis poorPronosis goodPronosis poorPronosis goodPronosis goodPronosis
      V7      V8      V9     V10     V11     V12
goodPronosis goodPronosis goodPronosis poorPronosis goodPronosis goodPronosis
      V13     V14     V15     V16     V17     V18
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V19     V20     V21     V22     V23     V24
goodPronosis poorPronosis goodPronosis goodPronosis goodPronosis poorPronosis
      V25     V26     V27     V28     V29     V30
goodPronosis poorPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V31     V32     V33     V34     V35     V36
goodPronosis goodPronosis goodPronosis goodPronosis poorPronosis goodPronosis
      V37     V38     V39     V40     V41     V42
goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis goodPronosis
      V43     V44     V45     V46     V47     V48
poorPronosis poorPronosis poorPronosis goodPronosis poorPronosis poorPronosis
      V49     V50     V51     V52     V53     V54
poorPronosis poorPronosis poorPronosis goodPronosis poorPronosis goodPronosis
      V55     V56     V57     V58     V59     V60
goodPronosis poorPronosis poorPronosis poorPronosis poorPronosis goodPronosis
      V61     V62     V63     V64     V65     V66
goodPronosis goodPronosis poorPronosis goodPronosis poorPronosis poorPronosis
```

	V67	V68	V69	V70	V71	V72
poorPronosis	poorPronosis	poorPronosis	poorPronosis	goodPronosis	poorPronosis	poorPronosis
	V73	V74	V75	V76	V77	V78
poorPronosis	poorPronosis	poorPronosis	poorPronosis	goodPronosis	poorPronosis	goodPronosis

Levels: goodPronosis poorPronosis

The vector returned contains the predicted class for each new sample.

Chapter 4

Access the results of one-layer and two-layer cross-validation

4.1 Introduction

When a one-layer or a two-layer cross-validation is run, the key results are printed out on screen. However, you might want to get more details about your run. This is possible via the call of the method `getResults`. This method has been designed to be a user-friendly interface to the complex class structure which assure the storage of the results of one-layer and two-layer cross-validation.

4.2 Argument of the method `getResults`

The method `getResults` has two main arguments: `layer` which specifies which layer of cross-validation is concerned and `topic` which specifies which piece of information is needed. There are also two optional arguments `errorType` and `genesType` that precise the scope of the `topic` arguments. The argument `layer` can take the following values:

- 1: Access to the one-layer external cross-validation
- 1,i: Access to the ith repeat of the one-layer external cross-validation
- 2: Access to the two-layer external cross-validation
- 2,i: Access to the ith repeat of the two-layer external cross-validation
- 2,i,j: Access to the jth inner one-layer cross-validation of the ith repeat of the two-layer external cross-validation
- 2,i,j,k: Access to the kth repeat of the jth inner one-layer cross-validation of the ith repeat of the two-layer external cross-validation

The argument `topic` can take the following values:

- ‘errorRate’: Access to the error rates related to the selected layer of cross-validation. The optional argument `errorType` can be used in conjunction with this topic.
- ‘genesSelected’: Access to the genes selected in to the selected layer of cross-validation. The optional argument `genesType` can be used in conjunction with this topic.
- ‘bestOptionValue’: Access to the best option value (best number of genes for RFE-SVM or best thresholds for NSC) in the selected layer. This value can be an average.
- ‘executionTime’: Access to the time in second that was necessary to compute the selected layer.

4.3 Error rates

4.3.1 Optional argument `errorType`

Different information on the error rates are available and can be specified with the arguments `errorType`:

- ‘all’: Access to all the following values.
- ‘cv’: Access to the cross-validated error rate.
- ‘se’: Access to the standard error on the cross-validated error rate.
- ‘fold’: Access to the error rate in each fold.
- ‘noSamplesPerFold’: Access to the number of samples per fold.
- ‘class’: Access to the error rate in each class.

The previous options are not available for all the types of layers. For instance, Since the repeated one-layer cross-validation is a summary of several repeats of one-layer cross-validation, we don’t have a fold error rate. The following table describes which option is available for each kind of layer.

Table 4.1: genesType available for each type of layer

layer argument	‘all’	‘cv’	‘se’	‘fold’	‘noSamplesPerFold’	‘class’
1	Yes	Yes	Yes	No	No	Yes
1,i	Yes	Yes	Yes	Yes	Yes	Yes
2	Yes	Yes	Yes	No	No	Yes
2,i	Yes	Yes	Yes	Yes	Yes	Yes
2,i,j	Yes	Yes	Yes	No	No	Yes
2,i,j,k	Yes	Yes	Yes	Yes	Yes	Yes

4.3.2 Examples

```
# All the information on error rates for the repeated one-layer CV
getResults(myExperiment, 1, topic='errorRate')
# Cross-validated error rates for the repeated one-layer CV: One value
# per size of subset
getResults(myExperiment, 1, topic='errorRate', errorType='cv')
# Cross-validated error rates for the repeated two-layer CV: One value
# only corresponding to the best error rate
getResults(myExperiment, 2, topic='errorRate', errorType='cv')
```

4.4 Genes selected

4.4.1 Optional argument `genesType`

Different information on the genes selected are available and can be specified with the arguments `genesType`:

- missing: Access to one of the following values (by default ‘frequ’ if available).
- ‘fold’: Access to the list of genes selected in each fold (and for each size of subset or threshold if relevant).
- ‘frequ’: Access to the genes selected order by their frequency along the folds and the repeats.

The previous options are not available for all the types of layers. For instance, Since the repeated one-layer cross-validation is a summary of several repeats of one-layer cross-validation, we don’t have the genes selected in each fold. The following table describes which option is available for each kind of layer.

Table 4.2: errorType available for each type of layer

layer argument	'fold'	'frequ'
1	No	Yes
1,i	Yes	Yes
2	No	Yes
2,i	Yes	Yes
2,i,j	No	Yes
2,i,j,k	Yes	Yes

4.4.2 Examples

```
# Frequency of the genes selected among the folds and repeats
# of the one-layer CV
getResults(myExperiment, c(1,1), topic='genesSelected', genesType='frequ')
# Genes selected for the 3rd size of subset in the 2nd fold of the
# second repeat of one-layer external CV
getResults(myExperiment, c(1,2), topic='genesSelected', genesType='fold')[[3]][[2]]
```

4.5 Best value of option

4.5.1 Overview

This topic gives access to the best values of option (best size of subset or best threshold) for a given layer.

4.5.2 Examples

```
# Best number of genes in one-layer CV
getResults(myExperiment, 1, topic='bestOptionValue')
# Best number of genes in the third repeat of one-layer CV
getResults(myExperiment, c(1,3), topic='bestOptionValue')
# Average (over the folds), best number of genes in the two-layer CV
getResults(myExperiment, 2, topic='bestOptionValue')
# Average (over the folds), best number of genes in the
# third repeat of the two-layer CV
getResults(myExperiment, c(2,3), topic='bestOptionValue')
```

4.6 Execution time

4.6.1 Overview

This topic gives access to the execution time needed to compute a given layer.

4.6.2 Examples

```
# Execution time to compute the repeated one-layer CV
getResults(myExperiment, 1, topic='executionTime')
# Execution time to compute the third repeat of the repeated one-layer CV
getResults(myExperiment, c(1,3), topic='executionTime')
# Execution time to compute the repeated two-layer CV
getResults(myExperiment, 2, topic='executionTime')
# Execution time to compute the second repeat of the repeated two-layer CV
getResults(myExperiment, c(2,2), topic='executionTime')
```

Chapter 5

Plots and graphics

This package also provide three methods to plot the results of the one-layer and two-layer cross-validation. `plotErrorsSummaryOneLayerCV` and `plotErrorsRepeatedOneLayerCV` plot the cross-validated error rate obtained during the one-layer cross-validation and `plotErrorsFoldTwoLayerCV` plot the fold error rates obtained in the second layer of two-layer cross-validation.

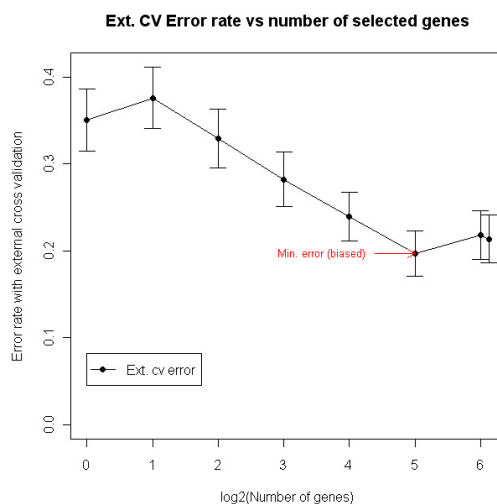
5.1 Plot the cross-validated error rates of one-layer cross-validation

5.1.1 Plot the summary error rate only

Concerning the one-layer cross-validation, the method `plotErrorsSummaryOneLayerCV` plots the cross-validated error rate averaged over the repeats versus the number of genes (for SVM-RFE) or the value of the thresholds (for NSC). An example of code is given below and figure 5.1.1 presents the corresponding plot.

```
plotErrorsSummaryOneLayerCV(myExperiment)
```

Figure 5.1: One-layer cross-validation: plot of the summary cross-validated error rate



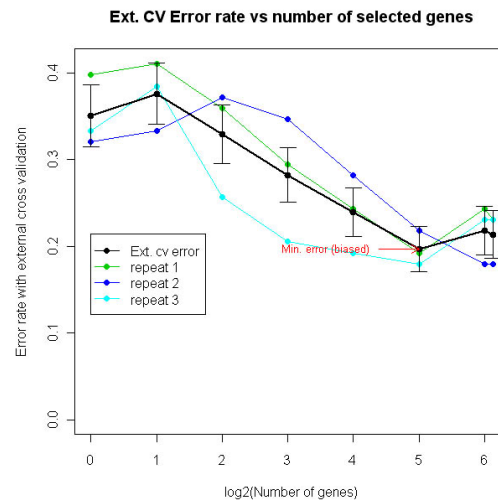
5.1.2 Plot the summary error rate only

Concerning the one-layer cross-validation, the method `plotErrorsRepeatedOneLayerCV` plots the cross-validated error rate averaged over the repeats and the cross-validated error rate obtained for each repeat

versus the number of genes (for SVM-RFE) or the value of the thresholds (for NSC). An example of code is given below and figure 5.1.2 presents the corresponding plot.

```
plotErrorsRepeatedOneLayerCV(myExperiment)
```

Figure 5.2: One-layer cross-validation: plot of the summary cross-validated error rate and the cross-validated error rate in each repeat



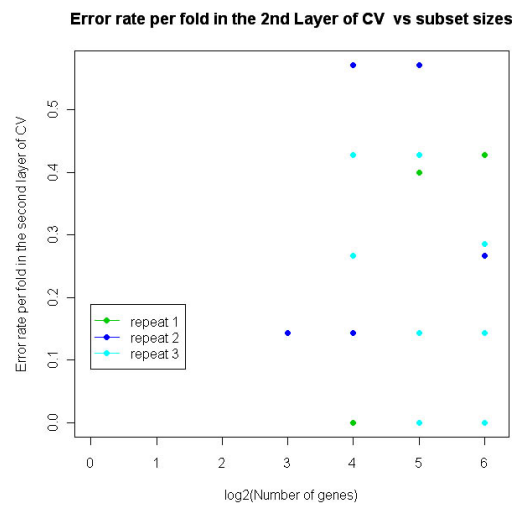
5.2 Plot the fold error rates of two-layer cross-validation

5.2.1 Plot the summary error rate only

Concerning the two-layer cross-validation, the method `plotErrorsFoldTwoLayerCV` plots the fold error rates in the second layer versus the number of genes (for SVM-RFE) or the value of the thresholds (for NSC). An example of code is given below and figure 5.2.1 presents the corresponding plot.

```
plotErrorsFoldTwoLayerCV(myExperiment)
```

Figure 5.3: Two-layer cross-validation: plot of fold error rates



Bibliography

- [1] C. Ambroise and G.J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6567–6572, 2002.
- [2] P. Burman. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514, 1989.
- [3] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.
- [4] G.J. McLachlan, J. Chevelu, and J. Zhu. Correcting for selection bias via cross-validation in the classification of microarray data. *Beyond Parametrics in Interdisciplinary Research: Festschrift in Honour of Professor Pranab K. Sen, N. Balakrishnan, E. Pena, and M.J. Silvapulle (Eds.)*. Hayward, California: IMS Collections, 1:383–395, 2008.
- [5] M. Stone. Cross-validatory choice and assessment of statistical predictions. *J. R. Stat. Soc. Ser.*, B(36):111–147, 1974.
- [6] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6567–6572, 2002.
- [7] I.A. Wood, P.M. Visscher, and K.L. Mengersen. Classification based upon gene expression data: bias and precision of error rates. *Bioinformatics*, 23(11):1363–1370, 2007.
- [8] J.X. Zhu, G.J. McLachlan, L. Ben-Tovim, and I. Wood. On selection biases with prediction rules formed from gene expression data. *Journal of Statistical Planning and Inference*, 38:374–386, 2008.