Rmagpie 1.9.0 User Manual

Camille Maumet

October 17, 2016

Contents

Chapter 1

Introduction

This package provides classes (data structures) and methods (functions) to train classifiers and to estimate their predictive error rate using external one-layer and two-layer cross-validation. These two techniques of cross-validation have been presented respectively in [**?**] and [**?**], [**?**], [**?**]. One-layer cross-validation can be used to determine a nearly unbiased estimate of the error rate which can then be used for feature selection. Feature selection may be used to select a near-optimal subset of the features and the error rate can be estimated for each subset of features using cross-validation. However, if the user wants to choose the classifer whose feature subset produced 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.

To load the Rmagpie package with the following R command and start using Rmagpie:

Chapter 2

Quick Start

2.1 Introduction

This section presents a quick review of the package functionality by giving an example. The Rmagpie package does not aim to pre-process your micro-array data. Other packages already provide this functionality and we assume that we are working on pre-processed data.

2.2 Define your experiment

To specify the options of our classification task, we need to create three objects. An object of class `ExpressionSet` 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.

2.2.1 Load your dataset

First of all, your data should be stored in an `ExpressionSet` in order to be usable from Rmagpie package. For more information on how to create and load data into an `ExpressionSet` please refer to the relevant documentation available on Bioconductor's website (http://www.bioconductor.org).

2.2.2 Store your feature selection options

Two feature selection methods are currently available, the Recursive Feature Elimination (RFE) for the Support Vector Machine (SVM), as presented in [**?**] and the Nearest Shrunken Centroid (NSC) as described in [**?**]. 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 consider subsets of size one up to the number total of features and all integer powers of two in between. For example, if the total number of features were 35, then subsets of size 1, 2, 4, 8, 16, 32 and 35 would be considered. If you wish to use this default `geneSubsets`, you can ignore the current section and go directly to section **??**.

Another solution is to define the size of the largest 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 [**?**], that the largest subset is considered first, then a subset of size equal to the greatest power below this and then decreasing by 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:
```

Alternatively, you can specify all the subset sizes that the software must try. For example the following command requests subsets of size 1, 2, 3, 5, 9, 10, 15 and 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 subset size larger than the number of genes in your dataset. If you do so, an error message similar to that shown below will be generated when you try to incorporate this object into your `experiment`.

```
Error in validObject(.Object) :
  invalid class "assessment" 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` stores the thresholds that must be considered by the nearest shrunken algorithm to determine which one is best. The threshold chosen determines how many genes are selected. Basically, you must specify the thresholds to 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 deatils 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 **??**.

Alternatively, you can specify all the thresholds that must be considered by the software. For example the following command requests the 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))
```

2.2.3 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 what options to use for the experiment. We begin by creating an object of class `assessment`. The argument must be specified as follows:

- `dataset`, ExpressionSet object that we created in section **??**
- `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 and for SVM as a classifier: 'linear' (linear kernel), 'radial' (radial basis kernel) or 'polynomial' (polynomial kernel).
- `noOfRepeats`, K-fold cross-validation allocates observations randomly to folds, so repeating the process is likely to give different estimates of error rate. The final results are then averaged over the repeats. As mentioned in [**?**], 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 **??** or `missing` if you want to use the default values

In the next sections, we will work with the dataset `vV70genes` available in the Rmagpie package. This is a version of data drawn from [**?**], in which the original number of genes measurements (for around 25000 genes) has been reduced to data for 70 genes. That process has biased any possible results, but it does provide a useful platform for testing. Before using it, you must make the following call to load the dataset.

```
> data('vV70genesDataset')
```

We will use the default `geneSubsets`.

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

```
> myAssessment <- new ( "assessment",
+                         dataset = vV70genes,
+                         noFolds1stLayer = 9,
+                         noFolds2ndLayer = 10,
+                         classifierName = "svm",
```

```
+                          featureSelectionMethod = 'rfe',
+                          typeFoldCreation = "original",
+                          svmKernel = "linear",
+                          noOfRepeats = 3)
> myAssessment

assessment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
featureSelectionMethod:rfe
featureSelectionOptions
   optionValues: 1 2 4 8 16 32 64 70  (maxSubsetSize: 70, speed:high, noOfOptions:8)
typeFoldCreation:original
noOfRepeats:3
dataset
: use 'getDataset(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 an NSC classifier, a cross-validation with 10 folds in the outer layer, 9 folds in the inner layer performed 10 times, we would use:

```
> myAssessment2 <- new ( "assessment",
+                           dataset = vV70genes,
+                           noFolds1stLayer = 9,
+                           noFolds2ndLayer = 10,
+                           classifierName = "nsc",
+                           featureSelectionMethod = 'nsc',
+                           typeFoldCreation = "original",
+                           noOfRepeats = 2)

12345678910111213141516171819202122324252627282930

> myAssessment2

assessment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:nscfeatureSelectionMethod:nsc
featureSelectionOptions
    optionValues:0 0.0907303 0.1814605 0.2721908 0.362921 0.4536513 0.5443815 0.6351118 0.72
typeFoldCreation:original
noOfRepeats:2
dataset
: use 'getDataset(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 assessment object, the thresholds have been successfully updated.

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

There are two methods (functions) which can help use run our assessment: `runOneLayerExtCV` and `runTwoLayerExtCV` for, respectively, computing an external one-layer or an external two-layer cross-validation including feature selection.

2.3.1 External One-Layer Cross Validation

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 an error rate estimate 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.

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

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

assessment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
featureSelectionMethod:rfe
featureSelectionOptions
   optionValues: 1 2 4 8 16 32 64 70  (maxSubsetSize: 70, speed:high, noOfOptions:8)
typeFoldCreation:original
noOfRepeats:3
dataset
: use 'getDataset(object)'

resultRepeated1LayerCV
   original1LayerCV: 3  combined 1 layer 10-folds CV: use 'getOriginal1LayerCV(object)'
   summaryFrequencyTopGenes: use 'getFrequencyTopGenes(object)'
   summaryErrorRate:
     cvErrorRate:
       0.3632479 0.3717949 0.3162393 0.2649573 0.2521368 0.2008547 0.2051282 0.2136752
     seErrorRate:
       0.0174141 0.0222094 0.0245958 0.029513 0.0401672 0.0365962 0.0361366 0.035697
```

```
    classErrorRates:
      goodPronosis: 0.2651515 0.3257576 0.280303 0.2272727 0.2045455 0.1969697 0.1666667 0.
      poorPronosis: 0.4901961 0.4313725 0.3627451 0.3137255 0.3137255 0.2058824 0.254902 0.
  => bestOptionValue: 32 (Est. Error rate: 0.2008547)
  executionTime: 1.790142s

No Results for 2 layers external CV

Final Classifier has not been computed yet
```

From this display we can see the key results of the one-layer cross-validation. For more details on how to get the complete results of this cross-validation, see section **??**. Here we can infer that the best subset size is 32 with an error rate of 0.2008547. The cross-validated error rates for subsets of size 1, 2, 4, 8, 16, 32, 64 and 70 are respectively 0.3632479, 0.3717949, 0.3162393, 0.2649573, 0.2521368, 0.2008547, 0.2051282, 0.2136752. The standard errors of these cross-validated error rates are respectively 0.0174141 0.0222094 0.0245958 0.029513 0.0401672 0.0365962 0.0361366 and 0.035697. 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 [**?**], [**?**] and [**?**], the best error rate is biased and two-layer of cross-validation must be computed to get a unbiased estimate.

2.3.2 Two-layer Cross Validation

External two-layer cross-validation aims to assess the error rate of a classifier after the best feature subset has been chosen using the results of one-layer cross-validtion with each feature subset size. Since all the options have already been chosen via the `experiment` object, it is easy to start two-layer cross-validation.

We can look again at our experiment. The result of two-layer cross-validation will have been updated.

```
> myAssessment <- runTwoLayerExtCV(myAssessment)
> myAssessment

assessment
noFolds1stLayer:9
noFolds2ndLayer:10
classifierName:svm (svmKernel: linear)
featureSelectionMethod:rfe
featureSelectionOptions
   optionValues: 1 2 4 8 16 32 64 70  (maxSubsetSize: 70, speed:high, noOfOptions:8)
typeFoldCreation:original
noOfRepeats:3
dataset
: use 'getDataset(object)'

resultRepeated1LayerCV
   original1LayerCV: 3  combined 1 layer 10-folds CV: use 'getOriginal1LayerCV(object)'
   summaryFrequencyTopGenes: use 'getFrequencyTopGenes(object)'
   summaryErrorRate:
     cvErrorRate:
```

```
      0.3632479 0.3717949 0.3162393 0.2649573 0.2521368 0.2008547 0.2051282 0.2136752
    seErrorRate:
      0.0174141 0.0222094 0.0245958 0.029513 0.0401672 0.0365962 0.0361366 0.035697
    classErrorRates:
      goodPronosis: 0.2651515 0.3257576 0.280303 0.2272727 0.2045455 0.1969697 0.1666667 0.
      poorPronosis: 0.4901961 0.4313725 0.3627451 0.3137255 0.3137255 0.2058824 0.254902 0.
  => bestOptionValue: 32 (Est. Error rate: 0.2008547)
  executionTime: 1.790142s

resultRepeated2LayerCV
  original2LayerCV: 3  combined 2 layer 10-folds CV: use 'getOriginal2LayerCV(object)'
  summaryErrorRate:
    finalErrorRate:
      0.2125899
    seFinalErrorRate:
      0.0376517
    classErrorRates:
      goodPronosis: 0.1818182
      poorPronosis: 0.254902
  => avgBestOptionValue: 36.26667 (Est. Error rate: 0.2125899)
  executionTime: 20.36138s

Final Classifier has not been computed yet
```

This command has given the key results of the two-layer cross-validation. For more details on how to get the complete results of cross-validation, see section **??**. Here we can infer that the estimate of the error rate after choosing the optimal subset size was 0.21 with the optimal subset size averaging 36.3 genes.

2.4 Classify new observations

Another simple method allow us to classify new based on our dataset. The final classifier is trained on the whole dataset. By default, it uses 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 or more observations. First, you must pre-process your raw data to produce a file containing the gene expression values in which each column corresponds to an observation and each line to a gene. The first row must contain the names of the new observations 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`.

```
> myAssessment <- findFinalClassifier(myAssessment)

[1] "no. features= 70"
[1] "no. features= 64"
[1] "no. features= 32"
[1] "no. features= 16"
[1] "no. features= 8"
```

```
[1] "no. features= 4"
[1] "no. features= 2"
```

Once the final classifier has been trained we can use it to predict the class of new observations. We will use the following filename `vV_NewSamples.txt`, which contains the gene expression values of four new observations.

	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`.

```
> newSamplesFile <- system.file(package="Rmagpie","extdata", "vV_newSamples.txt")
> res <- classifyNewSamples( myAssessment,newSamplesFile)

> library(xtable)
> goodPrognosis <-names(res)[res=="goodPronosis"]
> goodPrognosis

 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11" "V12"
[13] "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V24"
[25] "V25" "V26" "V27" "V28" "V29" "V30" "V31" "V32" "V33" "V34" "V35" "V36"
[37] "V37" "V38" "V39" "V40" "V41" "V42" "V43" "V44"

> poorPrognosis <-names(res)[res=="poorPronosis"]
> poorPrognosis

 [1] "V45" "V46" "V47" "V48" "V49" "V50" "V51" "V52" "V53" "V54" "V55" "V56"
[13] "V57" "V58" "V59" "V60" "V61" "V62" "V63" "V64" "V65" "V66" "V67" "V68"
[25] "V69" "V70" "V71" "V72" "V73" "V74" "V75" "V76" "V77" "V78"
```

```
> newSamplesFile <- system.file(package="Rmagpie","extdata", "vV_newSamples.txt")
> res <- classifyNewSamples( myAssessment,newSamplesFile,optionValue=1)

> goodPrognosis <-names(res)[res=="goodPronosis"]
> goodPrognosis

 [1] "V1"  "V3"  "V5"  "V6"  "V7"  "V8"  "V9"  "V11" "V12" "V13" "V14" "V15"
[13] "V16" "V17" "V18" "V19" "V21" "V22" "V23" "V25" "V27" "V28" "V29" "V30"
[25] "V31" "V32" "V33" "V34" "V36" "V37" "V38" "V39" "V40" "V41" "V42" "V46"
[37] "V52" "V54" "V55" "V60" "V61" "V62" "V64" "V70" "V76" "V78"

> poorPrognosis <-names(res)[res=="poorPronosis"]
```

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

Chapter 3

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

3.1 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 obtain more details on your run. This is possible by calling method `getResults`. This method has been designed to be a user-friendly interface to the complex class structure which stores the results of one-layer and two-layer cross-validation.

3.2 Argument of the method getResults

The method `getResults` has two main arguments: `layer` which specifies which layer of cross-validation is we are concerned with and `topic` which specifies what information is needed. There are also two optional arguments `errorType` and `genesType` that describe the scope of the `topic` arguments.

The argument `layer` can take on any of 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 specified layer of cross-validation. The optional argument errorType can be used in conjunction with this topic.
- 'genesSelected': Access to the genes selected in the specified 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 threshold for NSC) in the specified layer. This value can be an average.
- 'executionTime': Access to the time in seconds that was used to compute the specified layer.

3.3 Error rates

3.3.1 Optional argument errorType

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

- 'all' or missing: Access to all the following values.
- 'cv': Access to the cross-validated error rate.
- 'se': Access to the standard error of 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 repeated one-layer cross-validation gives 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 3.1: 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

3.3.2 Examples

```
> # All the information on error rates for the repeated one-layer CV
> getResults(myAssessment, 1, topic='errorRate')

cvErrorRate:
  0.3632479 0.3717949 0.3162393 0.2649573 0.2521368 0.2008547 0.2051282 0.2136752
seErrorRate:
  0.0174141 0.0222094 0.0245958 0.029513 0.0401672 0.0365962 0.0361366 0.035697
classErrorRates:
  goodPronosis: 0.2651515 0.3257576 0.280303 0.2272727 0.2045455 0.1969697 0.1666667 0.18181
  poorPronosis: 0.4901961 0.4313725 0.3627451 0.3137255 0.3137255 0.2058824 0.254902 0.25490
```

```
> # Cross-validated error rates for the repeated one-layer CV: Une value
> # per size of subset
> getResults(myAssessment, 1, topic='errorRate', errorType='cv')
```

```
[1] 0.3632479 0.3717949 0.3162393 0.2649573 0.2521368 0.2008547 0.2051282
[8] 0.2136752
```

```
> # Cross-validated error rates for the repeated two-layer CV: Une value
> # only corresponding to the best error rate
> getResults(myAssessment, 2, topic='errorRate', errorType='cv')
```

```
[1] 0.2125899
```

3.4 Genes selected

3.4.1 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 3.2: 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

3.4.2 Examples

```
> # Frequency of the genes selected among the folds and repeats
> # of the one-layer CV
> res <- getResults(myAssessment, 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(myAssessment, c(1,2), topic='genesSelected', genesType='fold')[[3]][[2]]
```

```
               gene1    gene13    gene58    gene51
rfe-scores 0.4836447 0.2064432 0.2236418 0.2779342
```

3.5 Best value of option

3.5.1 Overview

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

3.5.2 Examples

```
> # Best number of genes in one-layer CV
> getResults(myAssessment, 1, topic='bestOptionValue')

[1] 32

> # Best number of genes in the third repeat of one-layer CV
> getResults(myAssessment, c(1,3), topic='bestOptionValue')

[1] 64

> # Average (over the folds), best number of genes in the two-layer CV
> getResults(myAssessment, 2, topic='bestOptionValue')

[1] 36.26667

> # Average (over the folds), best number of genes in the
> # third repeat of the two-layer CV
> getResults(myAssessment, c(2,3), topic='bestOptionValue')

[1] 38.4
```

3.6 Execution time

3.6.1 Overview

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

3.6.2 Examples

```
> # Execution time to compute the repeated one-layer CV
> getResults(myAssessment, 1, topic='executionTime')

[1] 1.790142

> # Execution time to compute the third repeat of the repeated one-layer CV
> getResults(myAssessment, c(1,3), topic='executionTime')

[1] 0.5838621

> # Execution time to compute the repeated two-layer CV
> getResults(myAssessment, 2, topic='executionTime')
```

```
[1] 20.36138
```

```
> # Execution time to compute the second repeat of the repeated two-layer CV
> getResults(myAssessment, c(2,2), topic='executionTime')
```

```
[1] 6.530155
```

Chapter 4

Plots and graphics

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

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

4.1.1 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 is given below.

```
> png("plotErrorsSummaryOneLayerCV.png")
> plotErrorsSummaryOneLayerCV(myAssessment)
> dev.off()
```

4.1.2 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 is given below.

```
> png("plotSummaryErrorRate.png")
> plotErrorsRepeatedOneLayerCV(myAssessment)
> dev.off()
```

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

4.2.1 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 is given below.

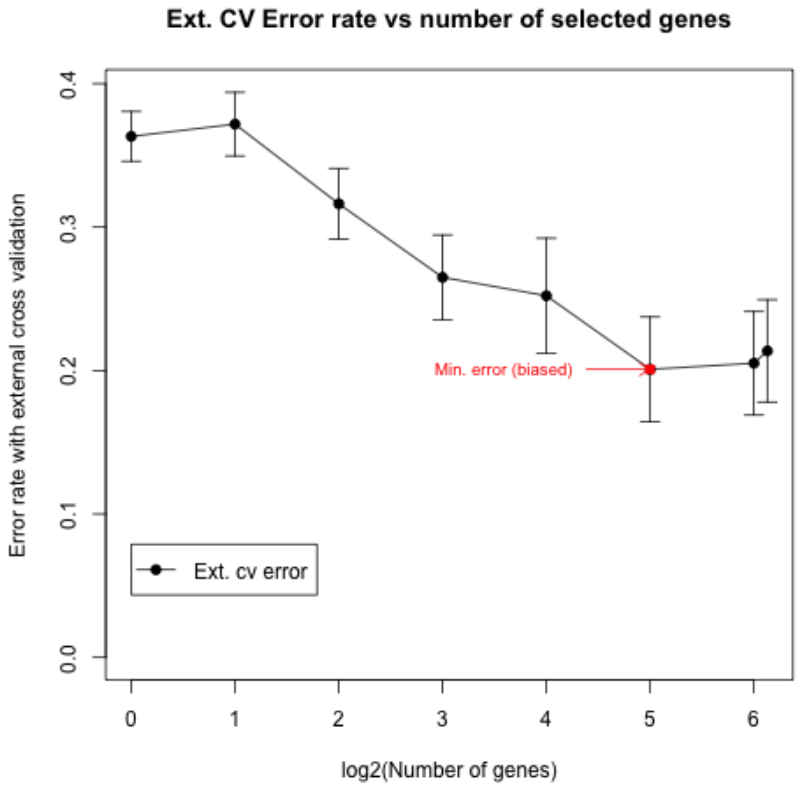

Figure 4.1: Cross-validated error rates of one layer cross validation.

```
> png("twoLayerCrossValidation.png")
> plotErrorsFoldTwoLayerCV(myAssessment)
> dev.off()
```

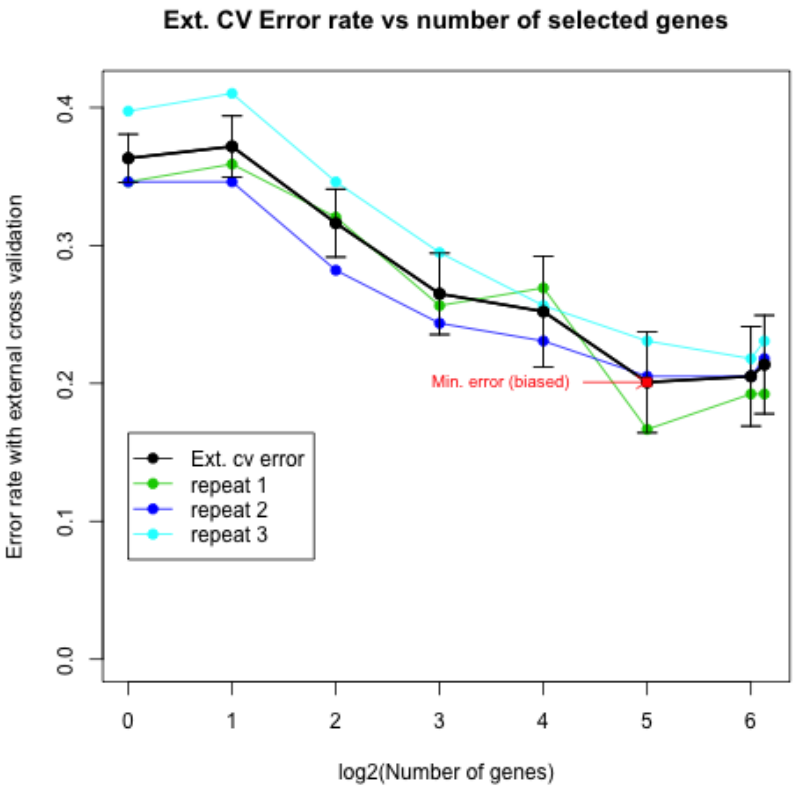

Figure 4.2: Cross-validated error rate averaged over the repeats .

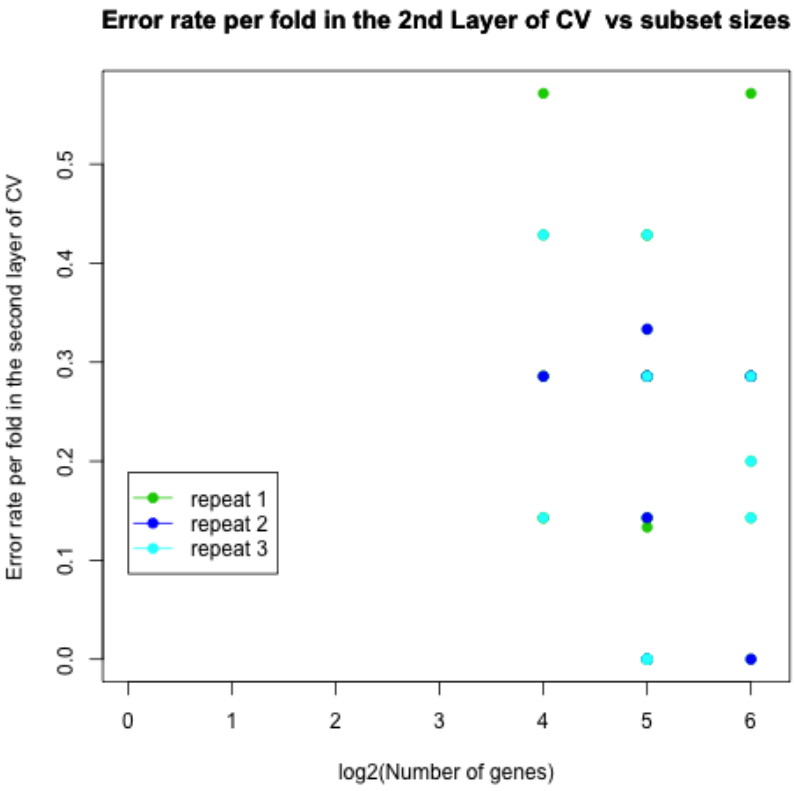

Figure 4.3: Error rates per fold in the second layer .