HOW TO use MCRestimate

Markus Ruschhaupt

April 24, 2017

Estimation the misclassification error

Every classification task starts with data. Here we choose the well known ALL/AML data set from T.Golub available in the package golubEsets. Furthermore, we specify the name of the phenodata column that should be used for the classification.

```
> library(MCRestimate)
> library(randomForest)
> library(golubEsets)
> data(Golub_Train)
> class.colum <- "ALL.AML"
```

Cross-validation is an appropriate instrument to estimate the misclassification rate of an algorithm that should be used for classification. Often a variable selection or aggregation is applied to the data set before the main classification procedure is started. But these steps must also be part of the cross-validation. In our example we want to perform a variable selection and only take the genes with the highest variance across all samples. Because we don’t know exactly how many genes we want to have, we give two possible values (250 and 1000). The described methods is implemented in the functions g.red.highest.var. Further preprocessing functions are part of the package MCRestimate and also new functions can be implemented.

```
> Preprocessingfunctions <- c("varSel.highest.var")
> list.of.poss.parameter <- list(var.numbers=c(250,1000))
```

To use MCRestimate with a classification procedure a wrapper for this method must be available. The package MCRestimate includes wrapper for the following classification functions.

**RF.wrap** wrapper for random forest (based on the package randomForest)

**PAM.wrap** wrapper for PAM (based on the package pamr)
PLR.wrap wrapper for the penalised logistic regression (based on the package MCRestimate)

SVM.wrap wrapper for support vector machines (based on the package e1071)

GPLS.wrap wrapper for generalised partial least squares (based on the package gpls)

It is easy to write a wrapper for a new classification method. Here we want to use random forest for our classification task.

```r
> class.function <- "RF.wrap"
```

**Parameter to optimize:** Most classification and preprocessing methods have parameters that must be chosen and/or optimized. In our example we choose two possible number of clusters for our preprocessing method. In each cross-validation step MCRestimate will choose the value that have the lowest misclassification error on the training set. This will be estimated through a second (inner) cross-validation with the function tune available in the package e1071.

**Parameter for the plot of the results (see man page for details):** We want to have the sample names as labels in the resulting plot. Samples is the name of the phenoData column the sample names are stored in.

```r
> plot.label <- "Samples"
```

**The Cross-validation parameter (see man page for details):** For time reasons here we choose very small values. Normally the values for cross.outer and cross.inner should be at least 5.

```r
> cross.outer <- 2
> cross.repeat <- 3
> cross.inner <- 2
```

Now we have specified all parameter and can run the function

```r
> RF.estimate <- MCRestimate(Golub_Train,
+    class.colum,
+    classification.fun="RF.wrap",
+    thePreprocessingMethods=Preprocessingfunctions,
+    poss.parameters=list.of.poss.parameter,
+    cross.outer=cross.outer,
+    cross.inner=cross.inner,
+    cross.repeat=cross.repeat,
+    plot.label=plot.label)
```

The result is an element of class MCRestimate
\texttt{> class(RF.estimate)}

[1] "MC\textquoteleft R\textquoteleft Estimate"

For each group we see how many samples have been misclassified most of the time. We can also visualize the result. The plot shows for each sample the number of times it has been classified correctly.

\texttt{> plot(RF.estimate,rownames.from.object=TRUE, main="Random Forest")}

\textbf{New data}

We have estimated the misclassification rate of our complete algorithm. Because this seems to be a quit good result, we want to use this algorithm to classify new data. The function \texttt{ClassifierBuild} is used to build a classifier that can be used for new data.
> RF.classifier <- ClassifierBuild (Golub_Train,
+ class.colum,
+ classification.fun="RF.wrap",
+ thePreprocessingMethods=Preprocessingfunctions,
+ poss.parameters=list.of.poss.parameter,
+ cross.inner=cross.inner)

The result of the function is a list with various arguments.

> names(RF.classifier)
[1] "classifier.for.matrix"
[2] "classifier.for.exprSet"
[3] "classes"
[4] "parameter"
[5] "thePreprocessingMethods"
[6] "class.method"
[7] "cross.inner"
[8] "information"

The most important elements of this list are 
classifier.for.matrix and classifier.for.exprSet.
These can now be used to classify new data. The first one can be used if the new data
is given by a matrix and the second will be used if the new data is given by a exprSet.
Here we want to classify the data from the exprSet golubTest, that is also part of the
package golubEsets.

> data(Golub_Test)
> RF.classifier$classifier.for.exprSet(Golub_Test)

39 40 42 47 48 49 41 43 44 45 46 70
ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
71 72 68 69 67 55 56 59 52 53 51 50
ALL ALL ALL ALL ALL ALL ALL AML AML AML AML
54 57 58 60 61 65 66 63 64 62
ALL AML AML ALL ALL AML ALL AML AML AML
Levels: ALL AML

This work was supported by NGFN (Nationales Genomforschungsnetz).

References

Loh ML, Downing JR, Caligiuri MA, Bloomfield CD, Lander ES. Molecular classification
of cancer: class discovery and class prediction by gene expression monitoring