— Molecular Diagnosis —
Tumor classification by PAM and Random Forest

Florian Markowetz, Markus Ruschhaupt, and Rainer Spang

Practical DNA Microarray Analysis, Heidelberg, 2008 March 03–06
`http://compdiag.molgen.mpg.de/ngfn/pma2008mar.php`

Abstract. *This tutorial refers to the practical session on day three of the course in Practical DNA Microarray Analysis. The topic is Molecular Diagnosis. You will learn how to apply Nearest Shrunken Centroids and Random Forest to microarray datasets. Important topics will be feature selection, cross validation and the selection bias.*

We consider expression data from patients with acute lymphoblastic leukemia (ALL) that were investigated using HGU95AV2 Affymetrix GeneChip arrays [Chiaretti et al., 2004]. The data were normalized using quantile normalization and expression estimates were computed using RMA. The preprocessed version of the data is available in the Bioconductor data package `ALL` from `http://www.bioconductor.org`. Type in the following code chunks to reproduce our analysis. At each ">" a new command starts, the "+" indicates that one command spans over more than one line.

```
> library(ALL)
> data(ALL)
> ALL
```

The most pronounced distinction in the data is between B- and T-cells. We concentrate on B-cells. Of particular interest is the comparison of samples with the BCR/ABL fusion gene resulting from a chromosomal translocation (9;22) with samples that are cytogenetically normal. We select samples of interest by first finding the B-cell samples, then finding all BCR/ABL and negative samples, and finally choosing the intersection of both sets:

```
> Bcell <- grep("^B", as.character(ALL$BT))
> trans <- which(as.character(ALL$mol.biol) %in% c("BCR/ABL",
+     "NEG"))
> select <- intersect(Bcell, trans)
> eset <- ALL[, select]
> pData(eset)[, "mol.biol"] <- factor(as.character(eset$mol.biol))
```

Exercise: Why is the last line necessary? What will be different if you remove the last line and then use the data for training the classifier?

The usual setting is diagnosis of patients we have not seen before. To simulate this situation we randomly hide six patients to be diagnosed later by a classifier trained on the rest. Of course the results will depend on this choice, so your results will naturally differ from those shown in this tutorial.

```
> hide <- sample(79, 6)
> known.patients <- eset[, -hide]
> new.patients <- eset[, hide]
> labels <- known.patients$mol.biol
```

When writing this tutorial, we ended up with 73 patients in the training set, 34 labeled as BCR/ABL and 39 labeled as NEG.

1 Nearest Shrunken Centroids

We first load the package pamr and then organize the data in a list `train.data` with tags for the expression matrix (`x`), the labels (`y`), names of genes and patients, and gene IDs. We get the gene IDs with the help of the annotation package `hgu95av2`.

```
> library(pamr)
> library(hgu95av2)
> dat <- exprs(known.patients)
> gI <- featureNames(known.patients)
> gN <- sapply(gI, get, hgu95av2SYMBOL)
> sI <- known.patients$cod
> train.dat <- list(x = dat, y = labels, genenames = gN, geneid = gI,
+     sampleid = sI)
```

1.1 Training PAM

In the lecture on PAM this morning you learned about gene selection by shrinkage. Shrinkage is controlled by a parameter which was called Δ in the lecture and is called *threshold* in the software. By default PAM fits the model for 30 different threshold values (that's the line of numbers you see when `pamr.train` runs):

```
> model <- pamr.train(train.dat)
> model
```

You get a table with 3 columns and 30 rows. The rows correspond to threshold values (first column). For each threshold you see the number of surviving genes (second column) and the number of misclassifications on the training set (third column).

Exercise: Explain why the number of surviving genes decreases when the size of the threshold increases.

1.2 10-fold cross validation

A more reliable error estimate than the number of misclassifications on the training set is the cross validation error. We compute and plot it for 30 different values of the threshold parameter.

```
> model.cv <- pamr.cv(model, train.dat, nfold = 10)
> model.cv
> pamr.plotcv(model.cv)
```

Exercise: Explain why the cross validation error (the third column when printing `model.cv`) is bigger than the training error (third column of `model`). Explain the upper figure in terms of bias-variance trade-off. Explain the behaviour at the right tail of the lower figure.

1.3 Plots for different threshold values

Using the results of cross validation, choose a threshold value `Delta` as a tradeoff between a small number of genes and a good generalization accuracy.

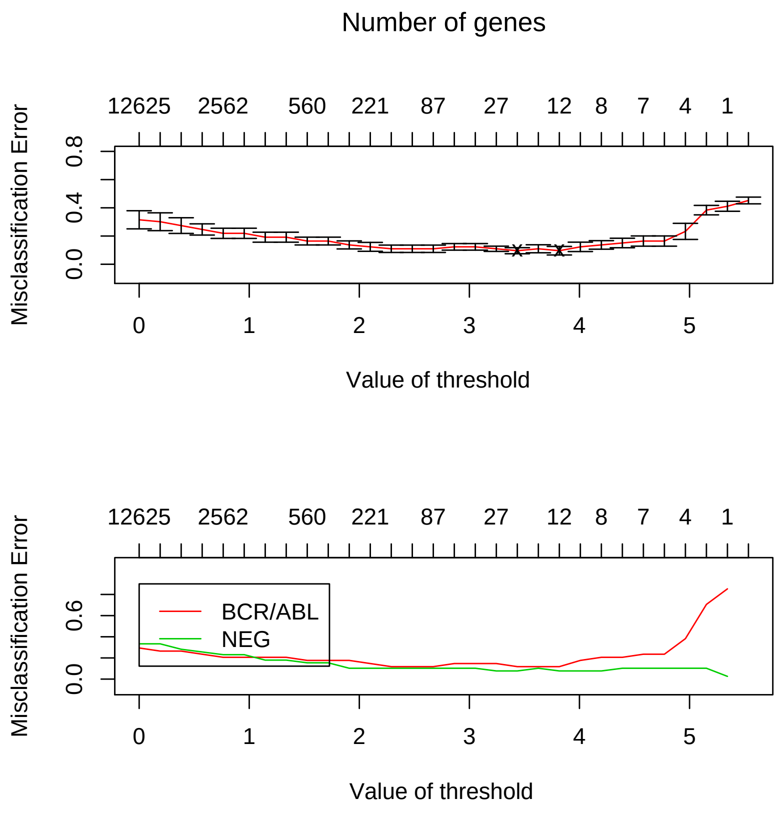

Figure 1: *Result of `pamr.plotcv()`. In both figures, the x-axis represents different values of threshold (corresponding to different numbers of nonzero genes as shown on top of each figure) and the y-axis shows the number of misclassifications. The upper figure describes the whole dataset, the lower one describes each class individually.*

```
> Delta = 3.5
```

This is just an arbitrary example. In the next steps, vary `Delta` through a range of values and observe how the plots and figures change.

• The function `pamr.plotcen()` plots the shrunken class centroids for each class, for genes surviving the threshold for at least one class.

```
> pamr.plotcen(model, train.dat, Delta)
```

Unfortunately, one can hardly read the gene names in the R plotting result. If you are interested in them, print the active graphic window using one of the following commands and then use Ghostview or AcrobatReader to view it in more detail.

```
> dev.print(file = "MYcentroids.ps")
> dev.print(device = pdf, file = "MYcentroids.pdf")
```

• The next function prints a 2×2 *confusion table*, which tells us how many samples in each class were predicted correctly.

```
> pamr.confusion(model.cv, Delta)
```

```
        BCR/ABL NEG Class Error rate
BCR/ABL      30   4        0.1176471
NEG           4  35        0.1025641
Overall error rate= 0.109
```

• To get a visual impression of how clearly the two classes are separated by PAM, we plot the cross validated sample probabilities:

```
> pamr.plotcvprob(model.cv, train.dat, Delta)
```

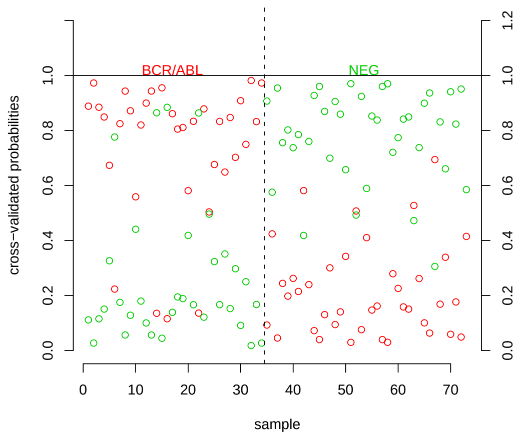

Figure 2: *Result of* `pamr.plotcvprob()`*. The 73 samples (x-axis) are plotted against the probabilities to belong to either class BCR/ABL (green) or NEG (red). For each sample you see two small circles: the red one shows the probability that this sample belongs to NEG and the green one that it belongs to BCR/ABL. A sample is put into that class for which probability exceeds* 0.5.

• The following command plots for each gene surviving the threshold a figure showing the expression levels of this gene over the whole set of samples. You will see which genes are up or down regulated and how variable they are.

```
> pamr.geneplot(model, train.dat, Delta)
```

Sometimes you get an error message "`Error in plot.new(): Figure margins too large`", because there is not enough space to plot all the genes. To mend this problem increase the threshold – which will decrease the number of genes.

More information about the genes used for the classification is given by `pamr.listgenes`. The output lists the Affymetrix ID and the name of the gene. In the last two columns you see a score indicating whether the gene is up or down regulated in the two classes of samples:

```
> pamr.listgenes(model, train.dat, Delta, genenames = TRUE)
```

```
      id        name     BCR/ABL-score NEG-score
 [1,] 1636_g_at ABL1     0.2551        -0.2224
 [2,] 39730_at  ABL1     0.2297        -0.2002
 [3,] 1674_at   YES1     0.2126        -0.1853
 [4,] 32434_at  MARCKS   0.1864        -0.1625
 [5,] 40202_at  KLF9     0.1595        -0.1391
 [6,] 40504_at  PON2     0.1586        -0.1383
 [7,] 1635_at   ABL1     0.1584        -0.1381
 [8,] 37363_at  MTSS1    0.0939        -0.0818
 [9,] 37015_at  ALDH1A1  0.0857        -0.0747
[10,] 37027_at  AHNAK    0.0611        -0.0532
[11,] 37403_at  ANXA1    0.0591        -0.0516
[12,] 37014_at  MX1      -0.0559       0.0487
[13,] 33362_at  CDC42EP3 0.0349        -0.0304
[14,] 34472_at  FZD6     0.0327        -0.0285
[15,] 33774_at  CASP8    0.0168        -0.0146
[16,] 33440_at  ZEB1     0.0049        -0.0043
[17,] 36591_at  TUBA4A   0.0026        -0.0023
```

Exercise: Create a new data set by removing the genes you got in this list from the original data set. Repeat the classification with PAM. What do you get? What does this tell you about the importance of the first list?

1.4 Computational Diagnosis

Now we use the trained PAM classifier to diagnose the new patients:

```
> pamr.predict(model, exprs(new.patients), Delta)
```

```
[1] NEG     BCR/ABL NEG     NEG     NEG     BCR/ABL
Levels: BCR/ABL NEG
```

But PAM does not only classify, if you use type="posterior" it also tells you how sure it is about its decision. The following table presents posterior class probabilities:

```
> pamr.predict(model, exprs(new.patients), Delta, type = "posterior")
```

```
         BCR/ABL        NEG
28021 0.38834783 0.61165217
24011 0.73103904 0.26896096
57001 0.49459198 0.50540802
68001 0.02872809 0.97127191
04008 0.04360260 0.95639740
26003 0.93320184 0.06679816
attr(,"scaled:center")

  BCR/ABL       NEG
0.9153791 0.7418747
```

Exercise: Which patients are clearly recognized, which predictions are doubtful? How does this change when you vary the threshold value?

1.5 Size does matter

Classification in high dimensions is a adventurous task, even if we have many training examples. Here we show what happens when sample size is really small. We use the function `rnorm()` to generate $N(0,1)$ data ("white noise") for 10 patients and 10000 genes. Due to sample variance, PAM sometimes "successfully" learns in a situation of pure noise with no signal at all. (Thanks to Benedikt for this instructive example.)

```
> nrG <- 10000
> nrP <- 10
> chance <- matrix(rnorm(nrG * nrP), ncol = nrP)
> class <- c(rep(0, nrP/2), rep(1, nrP/2))
> noise <- list(x = chance, y = class)
> model <- pamr.train(noise)
> pamr.plotcv(pamr.cv(model, noise, nfold = 5))
```

Exercise: Repeat this experiment a few times and watch out how often you oberserve a cross validation error of zero. Explain what happens! Why is this a disaster?

2 Random Forest

The Random Forest software is included in package `randomForest`. It needs the data in a slightly different format than PAM.

```
> library(randomForest)
> dat <- t(exprs(known.patients))
```

The command `t()` transposes a expression matrix ("mirrors it at the main diagonal"). The labels are already factors, so we keep them. To speed up computations, we will reduce the number of genes in the data. Let's have a look at the natural variation of the genes by computing the variance for each gene over all patients:

```
> v <- apply(dat, 2, var)
> hist(v, 100)
```

You will see that the activity of most genes does not differ much between the samples. We will discard the static genes and select the 2000 genes with highest variance.

```
> sel <- order(-v)[1:2000]
> dat <- dat[, sel]
```

2.1 Training error and test error

We will start with the training of the classifier:

```
> model <- randomForest(dat, labels)
> model

Call:
 randomForest(x = dat, y = labels)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 44

        OOB estimate of  error rate: 12.33%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      28   6  0.17647059
NEG           3  36  0.07692308
```

You get an overview of the trained forest. One thing that you will get is the OOB (out-of-bag) estimate of the test error. To see the difference between training error and test error we apply the model to the data that was used fitting the model:

```
> predicted <- predict(model, dat)
> table(true = labels, pred = predicted)

         pred
true       BCR/ABL NEG
  BCR/ABL      34   0
  NEG           0  39
```

2.2 Tuning the forest

There are several parameters that can be changed when growing a forest. For example you can alter the number of trees that are grown (`ntree`), the number of variables that are selected at each node (`mtry`), or the minimal node size for each leaf of any tree. The parameter that has the biggest effect when growing the forest is `mtry`. The default value for this parameter is the first integer that is lower than the square root of the number of variables.

```
> default <- floor(sqrt(ncol(dat)))
```

Maybe this is not a good choice. To check this we will vary the parameter and build a forest for every parameter. Then we will choose the model with the “best” parameter. To this end we will look at the OOB- error rate. This is a good estimate for the test error. The whole preocedure may take some time. If you do not want to wait that much, you can reduce the number of repetitions (parameter `nr.runs`).

```
> nr.runs <- 15
> mtry.range <- default * 2^c(0:4)
> res <- list()
> for (i in 1:length(mtry.range)) {
+     m <- mtry.range[i]
+     res[[i]] <- replicate(nr.runs, rev(randomForest(dat,
+         labels, mtry = m, ntree = 150)$err.rate[, "OOB"])[1])
+ }
> M <- ""
> yL <- "OOB error estimate"
> boxplot(data.frame(sapply(res, c)), names = mtry.range, main = M,
+     ylab = yL)
```

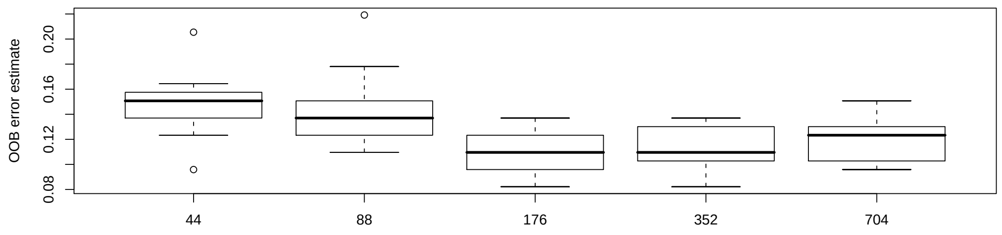

Figure 3: *Result of tuning randomForest with different choices for the number of parameters at each split.*

Although there is high variance in the OOB estimate, we see that a higher number of possible parameters gives slightly better results. So the default parameter seems to be inappropriate, and we set the parameter $mtry$ to be 200 and build our final forest.

```
> model <- randomForest(dat, labels, mtry = 200, ntree = 1000)
```

You can also use the function `tune.randomForest` (package `e01071`) to optimize the parameter `mtry`. The function uses a cross validation estimate of the test error instead of the OOB estimate. This might result in a slightly different result.

```
> library(e1071)
> default <- floor(sqrt(ncol(dat)))
> mtry.range <- default * 2^c(0:4)
> tune.result <- tune.randomForest(dat, labels, mtry = mtry.range)
> plot(tune.result)
```

Exercise: Read the manual to find out what the function `replicate` is doing exactly.

Exercise: Another parameter that can be changed is the number of trees that should be grown. Use `tune.randomForest` to compare forest with different number of trees. Does the change affect the classification result?

Exercise: (only if you have a lot of time). Use the function `tune.randomForest` to vary both parameters, the number of trees and the number of parameters at each node, simultaniously. Because this takes a while to compute, use only a few values for both parameters. Have a look at the plot function. What do you see?

Exercise: Read the help text of `tune.randomForets` and use `tune.control` to evaluate models by bootstrapping instead of cross validation.

2.3 Computational diagnosis

By analyzing the training error and the out-of-bag error we have seen that Random Forest is quite good at learning the difference between BCR/ABL and NEG. What does it tell us about the new patients?

```
> Predicted <- predict(model, t(exprs(new.patients))[, sel])
> Predicted

[1] NEG     BCR/ABL NEG     NEG     NEG     BCR/ABL
Levels: BCR/ABL NEG
```

The object `t(exprs(new.patients))[,sel]` in the command above is the expression matrix of the new patients – transposed and containing only high-variance genes. You can check whether the results are correct by comparing them to the true classes of the patients. If they do not correspond to the predictions, maybe we should have done the tuning more carefully ...

```
> data.frame(Status = new.patients$mol, Predicted = Predicted)

   Status Predicted
1 BCR/ABL       NEG
2 BCR/ABL   BCR/ABL
3     NEG       NEG
4     NEG       NEG
5     NEG       NEG
6 BCR/ABL   BCR/ABL
```

2.4 Zero training error does not guarantee good prediction!

In high dimensions our intuitions break down. The `labels` we used until now describe a biologically meaningful class distinction of the cancer samples. We demonstrate now that Random Forest can separate two classes of samples even if the labels are randomly permuted – destroying any biological information in the data.

```
> labels.rand <- sample(labels, 73)
> model.rand <- randomForest(dat, labels.rand)
> predicted <- predict(model.rand, dat)
> table(true = labels.rand, pred = predicted)
```

```
         pred
true      BCR/ABL NEG
  BCR/ABL      34   0
  NEG           0  39
```

Zero training error! Wow! Now have a look at the OOB error estimate. Compare the OOB error on the biological and on the randomized data. The OOB error for random labels will be very very high. This means: even with zero training error, we are bad at predicting new things.

Why is this observation important for medical diagnosis? Whenever you have expression levels from two kinds of patients, you will ALWAYS find differences in their gene expression - no matter how the groups are defined, no matter if there is any biological meaning. And these differences will not always be predictive.

2.5 How to select informative genes

We use a simple t-statistic to select the genes with the most impact on classification. The function `mt.teststat` from the library `multtest` provides a convenient way to calculate test statistics for each row of a data frame or matrix. As input it needs a matrix with rows corresponding to genes and columns to experiments. The class labels are supposed to be integers 0 or 1.

```
> library(multtest)
> dat <- t(exprs(known.patients))
> labels2 <- as.integer(labels) - 1
> tscores <- mt.teststat(t(dat), labels2, test = "t")
```

The vector `tscores` contains for each gene the value of the t-statistic. These values measure how well a gene separates the two classes. We select the 500 genes with highest t-statistic.

```
> selection <- order(-abs(tscores))[1:500]
> dat.sel <- dat[, selection]
```

The matrix `data.sel` contains 73 rows (samples) and 500 columns (the selected genes). Train Random Forest on this reduced dataset and compare the results to those obtained with all genes. You may vary the parameter `mtry` as well. Have a look at the OOB error. Vary the number of selected genes. How many genes do you need to still get a reasonable OOB error?

Exercise: As before write a loop to vary the number of selected genes and store the OOB estimates in a vector. Plot the result.

Exercise: Use the syntax `fsel <- function(x,y,n){.. commands in here ..}` to write a function for feature selection with the t-statistic that takes as inputs a data matrix `x`, a vector of labels `y`, and a number `n` of genes to be selected. The output should be a vector of indices of informative genes.

3 Model assessment

This section teaches you how to detect and avoid cheating in cross validation. We show you some widespread errors you can find in many papers, and their corrections. The content applies to any classification scheme, we will exemplify it using Random Forest.

3.1 The selection bias

There has been a conceptual flaw in the way we combined the cross validation with gene selection in the last section.

The idea of cross validation is this: Split your dataset in e.g. 10 subsets and take one subset out as a test set. Train your classifier on the remaining samples and assess its predictive power by applying it to the test set. Do this for all of the 10 subsets. This procedure is only sensible if no information about the test set is used while training the classifier. The test set has to remain 'unseen'.

What went wrong in the last section? We did a feature selection on the whole dataset, i.e. we selected the 12625 genes that are most informative given all 73 samples. Then we did a cross validation using these genes. Thus, the test set in each cross validation step had already been used for the feature selection. We had already seen it before training the classifier. Selecting genes outside the cross validation loop introduces a bias towards good results [Ambroise and McLachlan 2002, Simon *et al.* 2003]. The bias is more pronounced the fewer genes you select and the fewer samples you have.

What is the right way to use feature selection in cross validation? Do a selection of important genes in every step of cross validation anew! Do the selection only on the training set and never on the test set!

Exercise: Explain why selecting genes with high variance over all the samples does not result in the same distortion of cross validation as feature selection by t-scores.

Exercise: In the following we will guide you to write your own function for 10-fold cross validation with "in-loop" feature selection. Before starting, make sure that you understand the problem of selection bias, i.e. "out of the loop" feature selection.

If you have not done so yet to keep a log, open some texteditor and write your commands in an empty file.

```
CrossVal <- function(x, y, k=10, n) # x=data, y=labels, k=steps, n=genes
            {
            ... put the following commands all in here ...
            }
```

The input to `CrossVal` is the data (`x`), the labels (`y`), the number of cross validation steps (`k`) and the number of genes to select in each step (`n`). By default we set `k=10`.

At the beginning, we divide the data into several heaps such that the labels of both classes are balanced in each chunk. This can be easily done by the function `balanced.folds` from package `pamr` (which is not exported, so we have to use `get()`):

```
> balanced.folds <- get("balanced.folds", en = asNamespace("pamr"))
> help(balanced.folds)
> heaps <- balanced.folds(labels, k)
```

The list `heaps` has k entries, each containing the indices of one chunk. The data in the ith heap are reserved for testing. Training and feature selection are only performed on the remaining data. A cross validation is then just one `for`-loop:

`for (i in 1:k){`

1. Do a feature selection for `x` (without the test data!) using the function `fsel()` from the last section.
2. Train Random Forest on the pruned training set.
3. Then predict the labels of the test data (using only the selected genes).
4. Compute the accuracy (the number of correctly predicted test cases).

`}`

Collect the single CV accuracies obtained in each step. The total accuracy is the mean of it. Your function should return a list containing the total accuracy and the vector of single accuracies.

Exercise: Try cross validation with out-of-the-loop feature selection and with in-loop feature selection at least 100 times, gather the results in two vectors and compare the distribution of results in a boxplot and by `t.test()`. Explain why the variance of results is bigger in cross validation with in-loop feature selection.

More advanced: The function `errorest` in package `ipred` implements a unified interface to several resampling bases estimators. You can combine model fitting with gene selection by writting a wrapper function which can then be fed to `errorest`.

3.2 Nested-loop Cross validation

In section 3.1 we said: "The idea of cross validation is this: Split your dataset in e.g. 10 subsets and take one subset out as a test set. *Train your classifier* on the remaining samples and assess its predictive power by applying it to the test set. Do this for all of the 10 subsets." Now, what does "train your classifier" mean? It means: select the best parameters for the classifier given the training data in this CV step. But how do we select the best parameters? A natural way is to do cross validation again!

The last section taught us to do feature selection inside every cross validation loop. Here we see that also model selection should be done inside the cross validation loops. This results in two nested CV loops: the outer one for model assessment, the inner one for model selection.

If you want to implement nested-loop CV on your own, you will have to adapt the function `CrossVal()` you did in the last section by including an inner `for`-loop which selects the parameters to train the SVM. But luckily there is already a package doing it: `MCRestimate` by Ruschhaupt *et al.* (2004). It computes an estimate of the *misclassfication error rate* (MCR).

Inputs to the function are the *expression set* (`known.patients`), the name of the column with *class labels* (`"mol.biol"`), the *classification function* (here a wrapper for Random Forest), the *possible parameters* (here we vary the number of chosen parameters at each split *mtry*), and *how many outer and inner loops* cross validation should have and *how often to repeat* the whole process.

```
> library(MCRestimate)
> NestedCV.rf <- MCRestimate(known.patients, "mol.biol", classification.fun = "RF.wrap",
+     variableSel.fun = "varSel.highest.var", poss.parameters = list(var.numbers = c(100),
+         mtry = c(10, 50)), cross.outer = 5, cross.inner = 3,
+     cross.repeat = 3)
```

Let's consider an overview of the cross validation result:

```
> NestedCV.rf


Result of MCRestimate with 3 repetitions of 5-fold cross-validation

Preprocessing function 1 : varSel.highest.var
Classification function  : RF.wrap

The confusion table:
        BCR/ABL NEG class error
BCR/ABL      29   5       0.147
NEG           5  34       0.128
```

`MCRestimate` counts how often samples are misclassified in cross validation. The resulting vote matrix is visualized in Fig. 4.

```
> plot(NestedCV.rf, main = "Nested Cross validation with Random Forest")
```

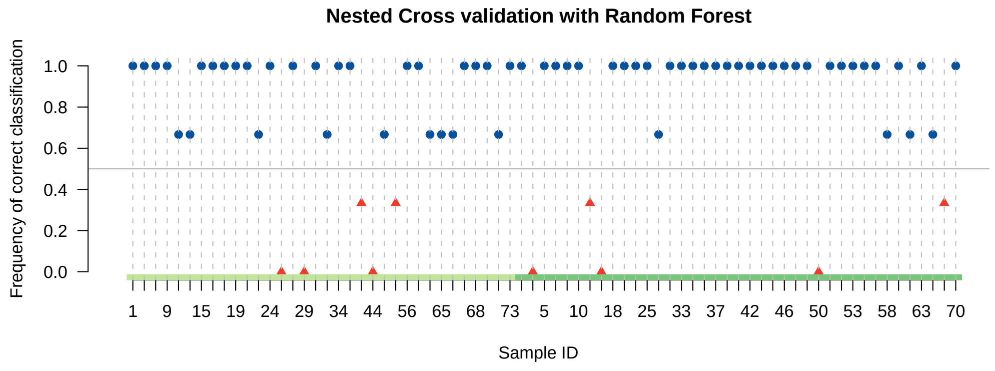

Figure 4: *Visualization of the vote matrix. Samples that are misclassificed most of the time are plotted in red triangles, the others in blue dots. The light green and dark green horizontal bars represent the two disease groups.*

Exercise: Use the function `ClassifierBuild` to classify `new.patients`.

Exercise: Evaluate Nearest Shrunken Centroids with MCRestimate. You find an instruction in the paper by Ruschhaupt *et al.* (2004). The package also contains wrappers for Support Vector Machines, Penalized Logistic Regression and Bayesian Binary Prediction Tree Models. If you feel like it and still have time left you could try out any of them.

4 Summary

Scan through this paper once again and identify in each section the main message you have learned. Some of the most important points are:

- You have learned how to apply the Nearest Shrunken Centroids method and Random Forest to microarray data.
- You have learned how to do feature selection using the t-statistic.
- You have experienced: It is quite easy to separate the training set without errors (even with randomly labeled samples), but this does not guarantee a good generalization performance on unseen test data.
- In cross validation: do all the data manipulation (like feature selection) inside the CV loop and not before. Else: cheating!
- For honest model assessment use nested-loop cross validation.

If you have any comments, criticisms or suggestions on our lectures, please contact us:
`rainer.spang@molgen.mpg.de` and `m.ruschhaupt@dkfz.de`

5 Solutions to selected exercises

Feature selection

```
> fsel

function (x, y, n)
{
    dat <- t(x)
    labels <- as.integer(y) - 1
    tscores <- mt.teststat(dat, labels, test = "t")
    sel <- order(-abs(tscores))[1:n]
    return(sel)
}
```

Feature selection inside the cross validation loop

```
> CrossVal

function (x, y, k = 10, n = 20)
{
    heaps <- get("balanced.folds", en = asNamespace("pamr"))(y,
        k)
    acc <- numeric(k)
    for (i in 1:k) {
        test <- heaps[[i]]
        sel <- fsel(x[-test, ], y[-test], n)
        RF <- randomForest(x[-test, sel], y[-test])
        p <- predict(RF, x[test, sel])
        acc[i] <- 100 * sum(p == y[test])/length(test)
    }
    return(list(tot.accuracy = mean(acc), single.accuracies = round(acc)))
}
```

Comparison of in-loop and out-of-loop feature selection

```
> nr.runs <- 20
> k <- 10
> n <- 20
> dat.sel <- dat[, fsel(dat, labels, n)]
> outloop <- 100 * (1 - replicate(nr.runs, rev(randomForest(dat.sel,
+     labels)$err.rate[, "OOB"])[1]))
> inloop <- replicate(nr.runs, CrossVal(dat, labels, k, n)$tot.accuracy)
> N <- c("out-of-loop feature selection", "in-loop feature selection")
> M <- "Out-of-loop feature selection is cheating!"
> yL <- "cross validation accuracy"
> t.test(outloop, inloop)$p.value
```

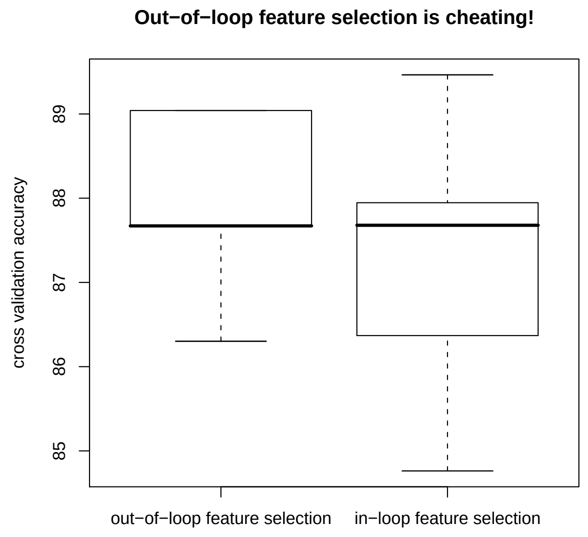

Figure 5: *Comparison of in-loop and out-of-loop feature selection. Selecting genes on the whole dataset introduces a bias on cross validation results.*

6 References

Ambroise C, McLachlan GJ. Selection bias in gene extraction on the basis of microarray gene-expression data. Proc Natl Acad Sci U S A. 2002 May 14;99(10):6562-6.

Chiaretti S, Li X, Gentleman R, Vitale A, Vignetti M, Mandelli F, Ritz J, Foa R. Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. Blood. 2004 Apr 1;103(7):2771-8.

Ruschhaupt M, Huber W, Poustka A, Mansmann U. A Compendium to Ensure Computational Reproducibility in High-Dimensional Classification Tasks. Statistical Applications in Genetics and Molecular Biology (2004) Vol. 3: No. 1, Article 37. http://www.bepress.com/sagmb/vol3/iss1/art37

Simon R, Radmacher MD, Dobbin K, McShane LM. Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification. J Natl Cancer Inst. 2003 Jan 1;95(1):14-8.

Tibshirani R, Hastie T, Narasimhan B, Chu G. Diagnosis of multiple cancer types by shrunken centroids of gene expression. Proc Natl Acad Sci U S A. 2002 May 14;99(10):6567-72.

7 PAMR patches

There are some ways to fix bugs in your own installation of the `pamr` package. The command

```
> file.path(.path.package("pamr"), "R")
```

gives you the path to where the R-code of package `pamr` is stored in your installation. In there you find the textfile `pamr`. Open it in a texteditor to do some little changes.

• If you install the package you can only do 10-fold cross validation with `pamr.cv`. Changing parameter `nfold` is fruitless. The problem lies in function `nsccv`. We patch it by substituting

```
folds <-balanced.folds(y)
```

with

```
if(is.null(nfold)) folds <-balanced.folds(y)
else folds <-balanced.folds(y, nfold=nfold)
```

• Using `pamr.geneplot` to plot a single gene results in an error message. The problem is that for a single gene the datamatrix becomes a vector and drops its dimension attributes. To prevent this, substitute two lines:

```
d    <- (cen - fit$centroid.overall)[aa, ]/fit$sd[aa]            # OLD
d    <- (cen - fit$centroid.overall)[aa, ,drop=FALSE]/fit$sd[aa] # NEW
... AND ...
xx   <- x[aa, o]             # OLD
xx   <- x[aa, o, drop=FALSE] # NEW
```

• Additionally, you could add a `cat("\n")` to the end of `pamr.train`.

Versions of R-packages

```
> sessionInfo()


R version 2.6.0 (2007-10-03)
x86_64-unknown-linux-gnu

locale:
C

attached base packages:
[1] splines   tools     stats     graphics  grDevices datasets  utils
[8] methods   base

other attached packages:
 [1] MCRestimate_1.11.6   ROC_1.12.0          gpls_1.10.0
 [4] arrayMagic_1.16.1    genefilter_1.16.0    vsn_3.2.1
 [7] limma_2.12.0         affy_1.16.0          preprocessCore_1.0.0
[10] affyio_1.6.1         golubEsets_1.4.4     RColorBrewer_1.0-2
[13] e1071_1.5-17         class_7.2-39         multtest_1.18.0
[16] randomForest_4.5-18  hgu95av2_1.99.10     pamr_1.36.0
[19] survival_2.32        cluster_1.11.9       ALL_1.4.3
[22] Biobase_1.16.2

loaded via a namespace (and not attached):
[1] AnnotationDbi_1.0.6 DBI_0.2-4          RSQLite_0.6-3
[4] annotate_1.16.1     grid_2.6.0         lattice_0.16-5
[7] rcompgen_0.1-15
```