— Molecular Diagnosis —
Tumor classification by SVM and PAM

Florian Markowetz and Rainer Spang

Practical DNA Microarray Analysis, Munich, 2006 May 08–11
http://compdiag.molgen.mpg.de/ngfn/pma2006may.shtml

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 Support Vector Machines 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)
> show(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))
```

The usual setting is diagnosis of patients we have not seen before. To simulate this situation we randomly hide three 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, 3)
> known.patients <- eset[, -hide]
> new.patients <- eset[, hide]
> labels <- known.patients$mol.biol
```

When writing this tutorial, we ended up with 76 patients in the training set, 35 labeled as BCR/ABL and 41 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:

```
> library(pamr)
> library(hgu95av2)
> dat <- exprs(known.patients)
> gN <- sapply(geneNames(ALL), get, hgu95av2SYMBOL)
> gI <- geneNames(known.patients)
> 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.

```
> Delta = 4
```

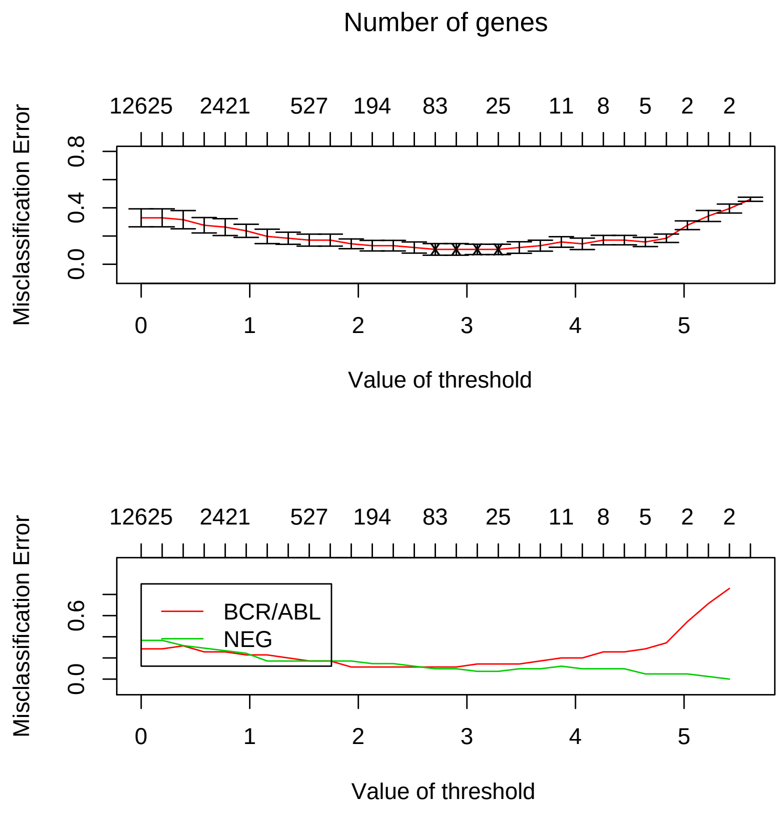

Figure 1: *Result of `pamr.plotcv()`. In both figures, the x-axis represents different values of threshold (corresponding to different numbers of nonzero genes as shown on top of each figure) and the y-axis shows the number of misclassifications. The upper figure describes the whole dataset, the lower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

This is just an arbitray 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      28   7       0.20000000
NEG           4  37       0.09756098
Overall error rate= 0.144
```

• 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, train.dat, Delta)
```

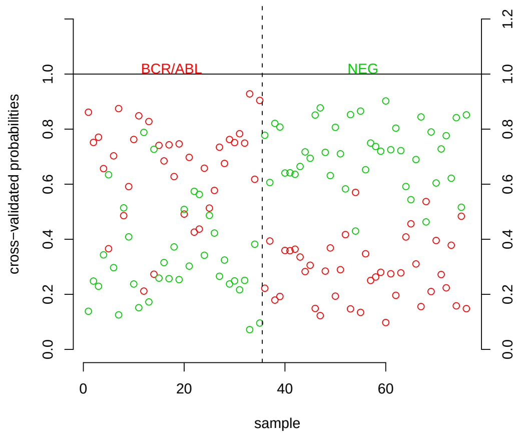

Figure 2: *Result of* `pamr.plotcvprob()`*. The 76 samples (x-axis) are plotted against the probabilities to belong to either class BCR/ABL (green) or NEG (red). For each sample you see two small circles: the red one shows the probability that this sample belongs to NEG and the green one that it belongs 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 downregulated 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    0.2           -0.1708
 [2,] 39730_at   ABL1    0.181         -0.1545
 [3,] 1635_at    ABL1    0.1234        -0.1053
 [4,] 1674_at    YES1    0.1121        -0.0957
 [5,] 40202_at   KLF9    0.086         -0.0734
 [6,] 40504_at   PON2    0.058         -0.0495
 [7,] 37015_at   ALDH1A1 0.0537        -0.0458
 [8,] 32434_at   MARCKS  0.0479        -0.0409
 [9,] 37027_at   AHNAK   0.011         -0.0094
[10,] 37014_at   MX1     -0.0087       0.0075
[11,] 37363_at   MTSS1   0.0024        -0.002
```

1.4 Computational Diagnosis

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

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

```
[1] BCR/ABL BCR/ABL NEG
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
03002 0.8423036 0.1576964
12012 0.8359386 0.1640614
28031 0.3536885 0.6463115
attr(,"scaled:center")

  BCR/ABL       NEG
0.8337321 0.6596805
```

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 Support Vector Machines (SVM)

The SVM software is included in package `e1071` – which is named after the Austrian code-number for the Vienna statistics department. It needs the data in a slightly different format than PAM.

```
> library(e1071)
> 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 1000 genes with highest variance

```
> sel <- order(-v)[1:1000]
> dat <- dat[, sel]
```

2.1 Training and cross-validation

We will begin with the simple linear kernel:

```
> model <- svm(dat, labels, kernel = "linear")
```

If `class(labels)` is not `factor`, the svm software does regression instead of classification. Use `summary(model)` to get an overview of the trained SVM and check that we did the right thing. Let's compute the training error by applying the model to the data on which it was trained:

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

```
         pred
true      BCR/ABL NEG
  BCR/ABL      35   0
  NEG           0  41
```

The linear kernel separates the training set without errors! But how is its ability to predict samples, which it has not seen before? We investigate by 10-fold cross validation.

```
> model.cv <- svm(dat, labels, kernel = "linear", cross = 10)
```

Use `summary(model.cv)` to get an overview over the SVM model. Do cross validation with the linear kernel several times. Why does the line "Single Accuracies" change each time? How does the total accuracy change?

2.2 Tuning the machine

In the last section we saw the cross-validation error for linear SVMs. Maybe classifiers with different complexity improve the result. We investigate by varying the width gamma of a radial basis kernel. The default value in the software is gamma=1/ncol(dat). We will look at the results for different multiples of this default value:

```
> g <- 1/ncol(dat)
> gamma.range <- 2^(-3:2) * g
> tune.result <- tune.svm(dat, labels, gamma = gamma.range, kernel = "radial")
> plot(tune.result)
```

```
> plot(tune.result)
> points(tune.result$performances[4, ], col = 2, pch = 19, lwd = 6)
```

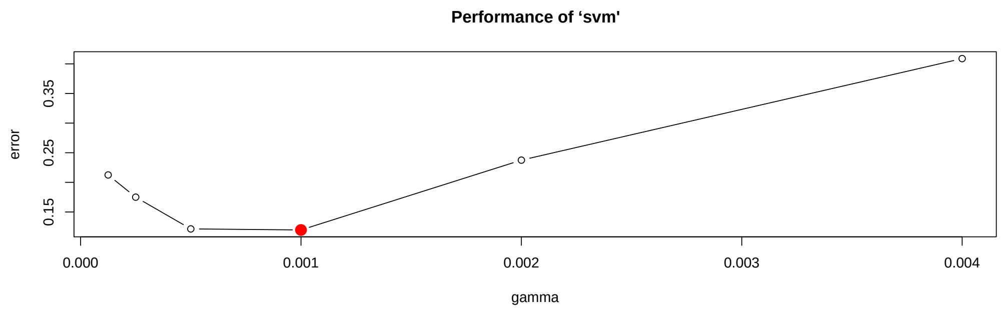

Figure 3: *Result of tuning SVMs with radial basis kernels of different with. The fat red dot corresponds to the default value. Performance can be improved slightly by using a smaller value of gamma – corresponding to a less complex SVM.*

Once we know the best parameter setting, we can use it to learn an SVM. These two steps (finding best parameter and then training an SVM) are combined in the function best.svm, which returns the SVM with the best parameters.

```
> model <- best.svm(dat, labels,
                    gamma = gamma.range,
                    kernel = "radial",
                    tunecontrol = tune.control()
                   )
```

Exercise: Use tune.svm to compare polynomial SVMs of degree 1 to 5. Do complex classifiers improve the classification result?

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

Advanced SVMing: The package svmpath allows to fit the entire regularization path (i.e. for all cost values) with essentially the same computational cost as fitting one SVM model.

2.3 Computational diagnosis

By analyzing the training error and the cross-validation error we have seen that a SVM is quite good at learning the difference between BCR/ABL and NEG. What does it tell us about the three new patients?

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

[1] BCR/ABL BCR/ABL NEG
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 to the true classes of the patients. If they do not agree to the predictions, maybe we should have done the tuning more carefully ...

```
> new.patients$mol

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

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 SVM can separate two classes of samples even if the labels are randomly permuted – destroying any biological information in the data.

```
> labels.rand <- sample(labels, 76)
> model.rand <- svm(dat, labels.rand, kernel = "linear")
> predicted <- predict(model.rand, dat)
> table(true = labels.rand, pred = predicted)

         pred
true      BCR/ABL NEG
  BCR/ABL      35   0
  NEG           0  41
```

Zero training error! Wow! Now train the SVM again with cross validation. Compare the CV error on the biological and on the randomized data. The CV 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)
> 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 100 genes with highest t-statistic.

```
> selection <- order(-abs(tscores))[1:100]
> dat.sel <- dat[, selection]
```

The matrix data.sel contains 76 rows (samples) and 1000 columns (the selected genes). Train a SVM on this reduced dataset with different kernels and parameters and compare the results to those obtained with all genes. Do cross-validation on the reduced dataset.

Vary the number of selected genes. How many genes do you need to still get a reasonable CV error?

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 SVMs.

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 1000 genes that are most informative given all 76 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 already 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 a SVM 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 accurarcies.

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 the 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 would 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 SVMs), the *possible parameters* (here we vary kernel width γ and the misclassification cost), and *how many outer and inner loops* cross validation should have and *how often to repeat* the whole process.

```
> library(MCRestimate)
> NestedCV.svm <- MCRestimate(known.patients, "mol.biol",
                              classification.fun = "SVM.wrap",
                              variableSel.fun = "varSel.highest.var",
                              poss.parameters = list(gamma = 2^(-2:2),
                              cost = 2^(-2:2)),
                              cross.outer = 5,
                              cross.inner = 5,
                              cross.repeat = 5)
```

Let's see an overview of the cross-validation result:

```
> NestedCV.svm
```

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

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

The confusion table:
        BCR/ABL NEG class error
BCR/ABL      26   9       0.257
NEG           6  35       0.146
```

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

```
> plot(NestedCV.svm, main = "Nested Cross-validation with Support Vector Machines")
```

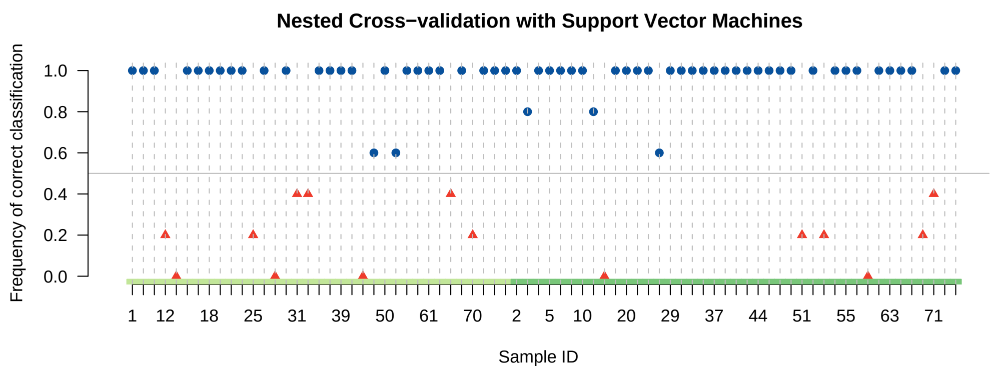

Figure 4: *Visualization of the vote matrix. Samples that are misclassificed most of the time are plotted in red triangles, the others in blue dots. The light green and dark green horizontal bars represent the two disease groups.*

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 Random Forests, 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 Support Vector Machines 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 `florian.markowetz@molgen.mpg.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){     # x=data, y=labels, k=nr. of splits, n
heaps <- balanced.folds(y,k)          # create k balanced folds
acc   <- numeric(k)                   # initialize list of accuracies
for (i in 1:k){                       # run through all k folds
  test<- heaps[[i]]                   # indices of test data
  sel <- fsel(x[-test,],y[-test],n)   # select genes on train data
  SVM <- svm(x[-test,sel], y[-test], kernel="linear") # train SVM on train set
  p   <- predict(SVM,x[test,sel])     # predict labels of TEST set
  acc[i] <- 100*sum(p == y[test])/length(test) # compute accuracy
  }
return(list(tot.accuracy=mean(acc), single.accuracies=round(acc)))
}
```

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

```
> nr.runs <- 1000
> k <- 10
> n <- 20
> dat.sel <- dat[, fsel(dat, labels, n)]
> outloop <- replicate(nr.runs, svm(dat.sel, labels, kernel = "linear", cross = k)$tot.accuracy)
> inloop <- replicate(nr.runs, CrossVal(dat, labels, k, n)$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"
> boxplot(data.frame(cbind(outloop, inloop)), names = N, main = M, ylab = yL)
> t.test(outloop, inloop)$p.value

[1] 2.063535e-236
```

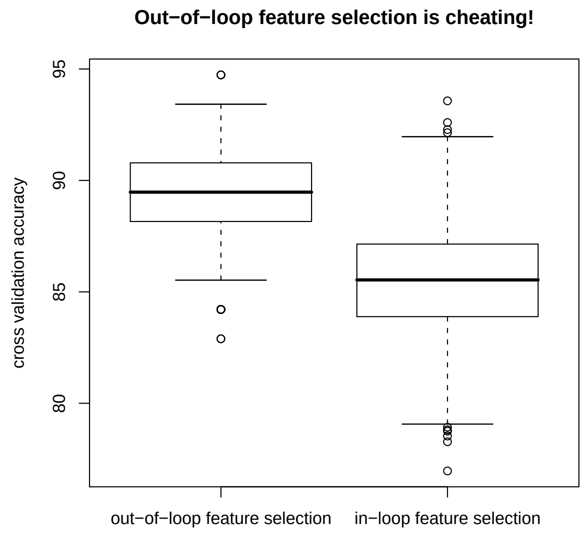

Figure 5: *Comparison of in-loop and out-of-loop feature selection. Selecting genes on the whole dataset introduces a bias on cross-validation results.*

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

	used.version
xtable	1.2−5
MCRestimate	1.4.0
RColorBrewer	0.2−3
randomForest	4.5−12
multtest	1.8.0
survival	2.18
splines	2.2.0
e1071	1.5−9
class	7.2−17
hgu95av2	1.10.0
pamr	1.27
Biobase	1.8.0
tools	2.2.0
ALL	1.0.2
methods	2.2.0
stats	2.2.0
graphics	2.2.0
grDevices	2.2.0
utils	2.2.0
datasets	2.2.0
base	2.2.0