— Molecular Diagnosis —
Tumor classification by SVM and PAM

Florian Markowetz and Rainer Spang

Practical DNA Microarray Analysis
Heidelberg, Oct 2003

Max-Planck-Institute for Molecular Genetics
Dept. Computational Molecular Biology
Computational Diagnostics Group
Berlin, Germany
http://compdiag.molgen.mpg.de

Abstract. *This tutorial refers to the practical session on day three of the course in Practical DNA Microarray Analysis, October 2003. 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 will explore data from Duke University and Duke Medical Center, as described in *Predicting the clinical status of human breast cancer by using gene expression profiles* (Mike West *et al.*, PNAS 2001 Sep 25; 98(20):11462-11467). It consists of 46 breast tumor samples separated into two classes: 23 samples were positive for estrogen receptor (class `ER+`) and 23 were negative (class `ER-`). For each of these samples the expression levels of 7129 genes were measured. In addition, you are given the expression profiles of three patients with unknown type of breast cancer. Your task is to learn the difference between `ER+` and `ER-` from the 46 training samples and then propose a diagnosis for the three new patients.

1 Nearest Shrunken Centroids

```
> load("breastcancer.Rdata")
> summary(breastcancer)
> dim(breastcancer$x)
> breastcancer$y
```

`breastcancer$x` is a matrix containing the expression profiles. It consists of 46 columns corresponding to samples and 7129 rows corresponding to genes. `breastcancer$y` contains the class labels of the 46 samples: either `ER+` or `ER-`.

1.1 Training PAM

```
> library(pamr)
> pam.trained <- pamr.train(breastcancer)
```

With this command you train PAM on the dataset. By typing `pam.trained` you get an output like this:

```
Call:
pamr.train(data = breastcancer)
   threshold nonzero errors
1  0.000     7129    0
2  0.236     5343    0
.      .        .    .
.      .        .    .
29 6.600        1    4
30 6.836        0    23
```

For 30 different values of the threshold the number of nonzero genes and the number of misclassifications on the training set are listed. Explain why the number of surviving genes decreases when the size of the threshold increases. Have a look at the last row of this table: why do we only have 23 errors if we classify with 0 genes? Why not 46?

1.2 10-fold cross validation

A more reliable error estimate than the number of misclassifications on the training set is the 10-fold cross validation error:

```
> pam.trained.cv <- pamr.cv(pam.trained, breastcancer)
```

The output of this function looks very similar to the table above. The numbers in the last column are now the summed errors of all 10 cross validation steps. The CV error usually is bigger than the training error. Explain this gap.

The results of cross validation can be visualized by

```
> pamr.plotcv(pam.trained.cv)
```

You will get two figures. In both, 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. Explain the behaviour at the right tail of the lower figure.

1.3 Plots for different threshold values

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. Using the results of cross validation, choose a threshold value `t` as a tradeoff between a small number of genes and a good generalization accuracy. (You cannot choose a value bigger than `5.7`. This seems to be a bug in the software.)

```
> t <- ???  # your choice of a THRESHOLD value
```

In the next steps, vary `t` 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(pam.trained, breastcancer, t)
```

Unfortunately, one cannot read the gene names in this figure. If you are interested in them, print the active graphic window by

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

and then use Ghostview or AcrobatReader to view it in more detail. In addition, the function `pamr.listgenes()` yields a list of gene names and IDs (see below).

***

```
> pamr.confusion(pam.trained.cv, t)
```

This function prints a 2×2 *confusion table* like this:

```
    ER- ER+ Class Error rate
ER-  21   2       0.08695652
ER+   3  20       0.13043478
Overall error rate= 0.108
```

23 samples belong to class `ER-`, 21 are classified correctly, and two are misclassified as `ER+`, 23 samples belong to class `ER+`. 20 are classified correctly and 3 are misclassified. This makes an overall error rate of 10.8%.

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

```
> pamr.plotcvprob(pam.trained, breastcancer, t)
```

The 46 samples (x-axis) are plotted against the probabilities to belong to either class ER+ (green) or ER- (red). For each sample you see two small circles: the red one shows the probability that this sample belongs to ER- and the green one that it belongs to ER+. A sample is put into that class for which probability exceeds 0.5.

***

For each gene surviving the threshold we get a figure showing the expression level of this gene over the whole set of samples by using the following command:

```
> pamr.geneplot(pam.trained, breastcancer, t)
```

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(pam.trained, breastcancer, t, genenames=TRUE)
```

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:

```
     id        name                          ER- score  ER+ score
[1,] X03635_at "ESR Estrogen receptor"     -0.4181    0.4181
[2,] U79293_at "Clone 23948 mRNA sequence" -0.2634    0.2634
```

If you have a biological background: do you know any of the gene names? Which genes in the list would you have expected?

1.4 Computational Diagnosis

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

```
> pam.diagnosis <- pamr.predict(pam.trained, newpatients, t)
```

`pam.diagnosis` is a vector containing your prediction. But PAM does not only classify, it also tells you how sure it is about its decision by computing posterior probabilities:

```
> pamr.predict(pam.trained, newpatients, t, type="posterior")
```

```
           ER-       ER+
V47 0.57961121 0.4203888
V48 0.04988766 0.9501123
V49 0.95703730 0.0429627
```

For patient 2 and 3, PAM is very confident of its decision (more than 95%), but for patient 1 it achieves only a 57% posterior probability for class ER-. And if we compare the result to the true classes (> trueclasses), we see that PAM actually got it wrong for patient 1 =(Let's see if SVMs can do it better!

2 Support Vector Machines (SVM)

```
> library(e1071)     # contains the SVM software
```

The SVM software needs the data in a slightly different format than PAM. The command t() transposes a matrix ("mirrors it at the main diagonal"). The labels are already factors, so we keep them.

```
> train.x <- t(breastcancer$x) # training data
> train.y <- breastcancer$y    # training labels
```

2.1 Training error

We will begin with the simple linear kernel:

```
> svm.trained <- svm(train.x, train.y, kernel="linear")
> summary(svm.trained) # gives an overview of the learned model
```

Because class(train.y) is factor, the svm software does classification by default. Let's compute the training error:

```
> predicted <- predict(svm.trained, train.x) # predict labels of training data
> sum(predicted != train.y)                  # count differences
> table(true=train.y, pred=predicted)        # confusion matrix
```

The linear kernel separates the training set without errors! But how is its ability to predict unseen samples? We investigate by 10-fold cross validation.

2.2 10-fold cross validation

```
> svm.trained.cv <- svm(train.x, train.y, kernel="linear", cross=10)
```

Typing summary(svm.trained.cv) gives an overview over your settings and the cross validation results:

```
10-fold cross-validation on training data:

Total Accuracy: 91.30435
Single Accuracies:
 100 100 100 100 80 75 100 50 100 100
```

Do cross validation with the linear kernel several times. Why does the line "Single Accuracies" change each time? Does the total accuracy change?

Try other kernels: (1.) `kernel="polynomial"` with `degree=2`, and (2.) `kernel="radial"`. The dataset is easy to separate; the complex kernel functions do not give much better results than the linear kernel.

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 `ER+` and `ER-`. What does it tell us about the three new patients?

```
> svm.diagnosis <- predict(svm.trained, t(newpatients))
> svm.diagnosis
```

2.4 Zero training error does not guarantee good prediction!

Maybe we could convince you that SVM are a powerful classification method achieving a very good generalization performance. But before we all get too confident let's do a little experiment. `train.y` describes a biologically meaningful class distinction of the cancer samples. Now we will assign the samples *randomly* into two classes and investigate how SVM will react.

```
> labels.rand <- sample(train.y, 46)
> svm.rand    <- svm(train.x, labels.rand, kernel="linear")
> predicted   <- predict(svm.rand, train.x)
> table(true=labels.rand, pred=predicted)
```

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 the most informative genes

We use a 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)
> data <- t(train.x) # = breastcancer$x
> classlabels <- as.integer(train.y)-1
> tscores <- mt.teststat(data,classlabels,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(tscores, decreasing=TRUE)[1:100]
> data.sel  <- train.x[,selection]
```

The matrix `data.sel` contains 46 rows (samples) and 100 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. How do you explain the differences?

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

2.6 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'.

So what went wrong in the last section? We did a feature selection on the whole dataset, i.e. we selected the 100 genes that are most informative given all 46 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.

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!

As an exercise we will write our own function for 10-fold cross-validation with feature selection. (If you have not done so already to keep a log, open some texteditor and write your commands in an empty file. Save this text file as `CrossVal.r`.)

The input to `CrossVal` is the data, the labels and the number of cross-validation steps `k`. By default `k` is chosen as 10.

```
CrossVal <- function(data, labels, k=10)
            {
            ... put the following commands all in here ...
            }
```

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 the package `pamr`. Look at the help text of this function. The authors state, that `balanced.folds` is an internal function which should not be called by the user. Maybe they want this useful little function to remain a secret ...

```
heaps <- balanced.folds(labels, k)
```

A cross validation is just one `for`-loop:

```
for (i in 1:k){
              ... describe what to do in each step ...
              }
```

(1.) Within the brackets you have to do the following: First, split the data into a training and test set. The samples in the ith heaps are reserved for testing and the model is trained on the remaining data.

```
data.test    <- data[, heaps[[i]]]
data.train   <- data[,-heaps[[i]]]
labels.test  <- labels[ heaps[[i]]]
labels.train <- labels[-heaps[[i]]]
```

(2.) Now you do a feature selection for `data.train` in the same way as you did it for the whole dataset in the last section.

(3.) train a SVM on the training set with the selected genes. Then predict the labels of `data.test` (using only the selected genes). Compute the test error `svm.error`.

(4.) Collect the SVM test error of each step in a vector `CV.error.single` by using the concatenation function `c()` (you have to initialize `CV.error.single <- c()` at the beginning):

```
CV.error.single <- c(CV.error.single, svm.error)
```

(5.) After the `for`-loop has finished the vector `CV.error.single` has `k` entries (one for each step of the cross validation). The overall CV error is the mean of `CV.error.single`.

Implement these steps, then source `CrossVal.r` and compare the outcome to the results achieved in the last section.

```
> source("CrossVal.r")
> CrossVal(train.x,train.y,k=10)
10-fold cross validation achieves 89% accuracy
```

3 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.
- It is quite easy to separate the training set without errors (SVM do this even on 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 and not before.

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`

The datasets used and this tutorial will be made available at the course homepage:
`http://compdiag.molgen.mpg.de/ngfn/pma2003oct.shtml`

4 CrossVal.r

This is how your function `CrossVal` could look like:

```
CrossVal <- function(data, labels, k=10)
{
 CV.error.single <- c()
 heaps <- balanced.folds(labels,k)

 # CV is just a FOR-loop:
 for (i in 1:k)
  {
   # 1. split data for training and testing
   data.test    <- data[, heaps[[i]]]
   data.train   <- data[,-heaps[[i]]]
   labels.test  <- labels[ heaps[[i]]]
   labels.train <- labels[-heaps[[i]]]

   # 2. select genes only on training data
   tscores      <- mt.teststat(data.train, labels.train, test="t")
   selection    <- order(tscores, decreasing=TRUE)[1:100]
   train.sel    <- data.train[selection, ]

   # 3. train the model
   svm.model    <- svm(t(train.sel), as.factor(labels.train), kernel="linear")

   # 4. test the model
   test.sel     <- t(data.test[selection, ])
   predicted    <- predict(svm.model, test.sel)
   svm.error    <- sum(predicted != labels.test)*100/nrow(test.sel)
   CV.error.single <- c(CV.error.single, svm.error)
  }
 CV.error.total <- mean(CV.error.single)
 accuracy       <- 100-CV.error.total
 cat(k,"-fold cross validation achieves ",accuracy,"% accuracy\n",sep="")
}
```