Microarray Analysis
Classification by SVM and PAM

Rainer Spang and Florian Markowetz

Heidelberg: 2002 Sep 26

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

Abstract

In this practical session 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.

1 Duke dataset

This is a dataset 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 49 breast tumor samples separated into two classes: 25 samples were positive for estrogen receptor (class `ER+`) and 24 were negative (class `ER-`). For each of these samples the expression levels of 7129 genes were measured.

1.1 Nearest Shrunken Centroids

First we load the library containing the PAM software and read in the dataset:

```
library(pamr)
load("DUKE.Rdata")
```

After loading you will find two objects: `duke.pam` and `duke.svm`. Both contain the same dataset but differ in the way they are formatted. We will use `duke.pam` in this section and `duke.svm` in the next.

```
summary(duke.pam)
```

duke.pam$x is a matrix containing the expression profiles. It consists of 49 columns corresponding to samples and 7129 rows corresponding to genes. duke.pam$y contains the class labels of the 49 samples: either ER+ or ER−.

***

```
duke.trained <- pamr.train(duke.pam)
```

With this command you train PAM on the dataset. Typing duke.trained you get an output like this:

```
Call:
pamr.train(data = duke.pam)
   threshold nonzero errors
1  0.000     7129    1
2  0.241     5355    1
.      .        .    .
.      .        .    .
29 6.755        1    3
30 6.996        0    24
```

For 30 different values of the threshold the number of nonzero genes and the number of misclassifications on the training set are listed. Have a look at the last row of this table. Why do we only have 24 errors if we classify with 0 genes? Why not 49?

***

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

```
duke.cv <- pamr.cv(duke.trained, duke.pam)
```

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(duke.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.

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
```

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(duke.trained, duke.pam, 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(duke.cv, t)
```

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

```
    ER- ER+ Class Error rate
ER-  23   1       0.04166667
ER+   3  22       0.12000000
Overall error rate= 0.081
```

24 samples belong to class `ER-`. 23 are classified correctly, and one is misclassified as `ER+`. 25 samples belong to class `ER+`. 22 are classified correctly and 3 are misclassified. This makes an overall error rate of 8.1%. This table will look a bit more exciting in the next part of our session, when we examine a dataset containing six subclasses and not just two.

***

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

```
pamr.plotcvprob(duke.trained, duke.pam, t)
```

The 49 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(duke.trained, duke.pam, 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 classificatin is supplied by

`pamr.listgenes(duke.trained,duke.pam,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"     -0.189     0.1815
[2,] U79293_at "Clone 23948 mRNA sequence" -0.0056    0.0054
```

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

1.2 Support Vector Machines (SVM)

```
library(e1071)     # contains the SVM software
summary(duke.svm)  # same dataset, other format
```

`duke.svm$data` is the gene expression matrix and `duke.svm$labels` are the class labels: `-1` for class `ER-` and `+1` for class `ER+`. Don't get confused with the dimensions of the gene expression matrix. Usually columns are samples and rows are genes. But the SVM software needs it just the other way round: the 49 rows stand for the samples and the 7129 columns for the genes.

1.2.1 Training and cross validation

We will begin with the simple linear kernel:

```
data      <- duke.svm$data
labels    <- duke.svm$labels
svm.model <- svm(data, labels, type="C-classification", kernel="linear")
```

Let's compute the training error:

```
predicted <- predict(svm.model, data) # predict labels of training data
sum(predicted != labels)              # count differences
table(true=labels, 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:

```
svm.cross <- svm(data,labels, type="C-classification",
                 kernel="linear", cross=10)
```

(Note: don't use parentheses like `cross="10"`. Bug!) Typing `svm.cross` gives an overview over your settings and the cross validation results:

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

Total Accuracy: 91.83673
Single Accuracies:
 75 100 60 100 80 100 100 100 100 100
```

Try different kernel functions (`polynomial, radial`) with several parameters (`degree, gamma`) and values of the error weight (`cost`). What is the best result you can get?

1.2.2 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. `duke.svm$labels` 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(c(-1,+1),49, replace=TRUE)
svm.rand    <- svm(data, labels.rand, type="C-classification",
                   kernel="polynomial", degree="2")
```

How many errors on the training set do you find? Does this surprise you?

Now train the SVM again with cross validation! Compare the CV error on the biological and on the randomized data.

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.

1.2.3 How to select the most informative genes

We use a t-statistic to select the genes with the most impact on classification. The t-statistic can be written as

$$t = \frac{|\mu_+ - \mu_-|}{\sqrt{(n_+ - 1)\sigma_+^2 + (n_- - 1)\sigma_-^2}} \times \sqrt{\frac{n_+ n_- (n-2)}{n}}, \tag{1}$$

and is built from the following components:

n : number of samples ($= 49$)
n_+ : number of positive samples ($= 25$)
n_- : number of negative samples ($= 24$)
μ_+ : mean of expression levels in positive samples
μ_- : mean of expression levels in negative samples
σ_+^2 : variance in positive class
σ_-^2 : variance in negative class

The second term in formula (1) is constant. We can neglect it, because we are not interested in the absolute value of the t-statistic but in its relative size for different genes.

Since R is good at matrix computations, you don't have to compute the t-value for each gene individually. It's more efficient to do it like this:

```
ER.pos  <- data[labels==+1,]    # matrix of positive samples
mu.pos  <- apply(ER.pos,2,mean) # compute mean of each column (gene)
var.pos <- apply(ER.pos,2,var)  # compute variance of each column (gene)
```

`mu.pos` and `var.pos` are vectors with 7129 entries. The entry of `mu.pos` at position i is the mean expression level of gene i over all samples with a positive label. Analogous for `var.pos`. Repeat the commands above on the negative samples and get vectors `mu.neg` and `var.neg`.

Use these building blocks to compute formula (1). This will give you a vector of length 7129: each entry is the t-score of the corresponding gene. Call this vector `tscores`. Now we order the vector `tscores` and select the 100 genes with highest t-score:

```
index     <- order(tscores, decreasing=TRUE)
data.sel  <- data[,index[1:100]]
```

The matrix `data.sel` contains 49 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?

1.2.4 The selection bias

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

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 paragraph? We did a feature selection on the whole dataset, i.e. we selected the 100 genes that are most informative given all 49 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 now implement a 10-fold cross-validation with feature selection. If you have not done so already to keep a log, please open some texteditor and write your commands in an empty file. Save this text file as `mycv.r`. To execute it, type

```
source("mycv.r")
```

First, we randomly reorder the dataset.

```
n           <- nrows(data)          # n = 49
permutation <- sample(1:n)
data.perm   <- data[permutation,]   # random permutation of samples
labels.perm <- labels[permutation]  # same permutation of labels
```

A cross validation is just one FOR-loop:

```
k <- 10                             # how many CV steps?
for (i in 1:k){
               ... your commands ...
              }
```

Within the brackets you have to do the following: First, split the data into a training and test set. Since we randomly reordered the data set, you only have to move a window of size n/k over the rows. The samples within are the test set, the samples outside are the training set.

```
win             <- round(n/k)                 # size of window
cv              <- ((i-1)*win+1):min(n,i*win) # move window
CV.test         <- data.perm[cv,]             # samples within window
CV.train        <- data.perm[-cv,]            # samples outside window
CV.labels.test  <- labels.perm[cv]
CV.labels.train <- labels.perm[-cv]
```

The second line looks more difficult than it is. Think about what happens for `i=1` and `i=10`.

(2.) Now you do a feature selection for `CV.train` in 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 `CV.test` (using only the selected genes). Compute the test 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 run `mycv.r` and compare the outcome to the results achieved in the last section.

2 St.Jude dataset

There will be no new excersises in this section. We provide an opportunity for you to apply the methods you have learned to a more complex dataset.

```
load("STJUDE.Rdata")
```

This dataset was published by Yeoh *et al.*, *Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling*, Cancer Cell 1: 133-143, 2002. The original dataset contains 360 samples of ALL patients. We selected 252 samples belonging to 6 well known subclasses: T-ALL, E2A-PBX1, BCR-ABL, TEL-AML1, MLL and hyperdiploid > 50 chromosomes. For each sample the expression levels of 8638 genes are measured.

Data is in the same format as before: use `stjude.pam` with the PAM software and `stjude.svm` with SVMs. To use PAM, just follow the instructions of section 1.1. Training SVM on 252 samples in 6 classes with more than 8000 genes takes a lot of time. To speed things up, we will do a quick feature selection using PAM!

Feature selection by PAM

Train PAM on the St.Jude dataset, do cross validation and choose a threshold `t`:

```
stjude.trained <- pamr.train(stjude.pam)
stjude.cv      <- pamr.cv(stjude.trained, stjude.pam)
t              <- ???
```

Now we use the function `pamr.predict()` with parameter `type="nonzero"` to get the indices of the genes surving the threshold for at least one class:

```
selected <- pamr.predict(trained, stjude.pam$x, t, type="nonzero")
data     <- stjude.svm$data[,selected]
labels   <- stjude.svm$labels
```

Train a SVM on this dataset and compare the CV results to those obtained by PAM using the same number of genes. Try different kernels and parameters.

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 paper will be made available on our homepage:
`http://cmb.molgen.mpg.de/compdiag`