Lab 6: Classification Using R and Bioconductor

June 4, 2003

We continue our extended example involving the dataset from **?**. In this lab, we will use some of the different R routines.

We will concentrate on the following list

- knn
- lda
- svm
- neural networks

The R packages you will need to carry out this tutorial are

- Biobase, annotate, genefilter
- e1071
- nnet, MASS (from the VR bundle)

```
> library(Biobase)
> library(annotate)
> library(golubEsets)
> library(genefilter)
> library(nnet)
> library(mva)
> library(e1071)
> library(class)
> library(bioclabs)
> library(MASS)
```

We will set up the data from scratch once again.

```
> mmfilt <- function(r = 5, d = 500, na.rm = TRUE) {
+     function(x) {
+         minval <- min(2^x, na.rm = na.rm)
+         maxval <- max(2^x, na.rm = na.rm)
+         (maxval/minval > r) && (maxval - minval > d)
+     }
+ }
> GolubTrans <- function(eSet) {
+     X <- exprs(eSet)
+     X[X < 100] <- 100
+     X[X > 16000] <- 16000
+     X <- log2(X)
+     eSet@exprs <- X
+     eSet
+ }
> data(golubTrain)
> data(golubMerge)
> data(golubTest)
> gTrn <- GolubTrans(golubTrain)
> gTest <- GolubTrans(golubTest)
> gMerge <- GolubTrans(golubMerge)
> mmfun <- mmfilt()
> ffun <- filterfun(mmfun)
> sub <- genefilter(gTrn, ffun)
> sub[c(2401, 3398, 4168)] <- FALSE
> sum(sub)

[1] 3051

> gTrnS <- gTrn[sub, ]
> gTestS <- gTest[sub, ]
> gMergeS <- gMerge[sub, ]
> Ytr <- paste(golubTrain$ALL.AML, golubTrain$T.B.cell)
> Ytr <- sub("NA", "", Ytr)
> Ytest <- paste(golubTest$ALL.AML, golubTest$T.B.cell)
> Ytest <- sub("NA", "", Ytest)
> Ymerge <- paste(golubMerge$ALL.AML, golubMerge$T.B.cell)
> Ymerge <- sub("NA", "", Ymerge)
```

For the most part we will concentrate on developing supervised learning models for the `ALL.AML` categories. We will rely on the training set to build a model and then use the test set to establish its prediction error.

Since this is reasonably wasteful of the data resources, we will, if time permits, explore the use of the whole data set and cross-validation.

In this tutorial we make the assumption that all the data have been obtained, processed (expression levels estimated and normalization has been carried out). Performing those tasks is the subject of a separate tutorial.

In some sense the purpose of classification is to develop a model and to estimate all of its parameters so that when a new case is provided its class can be predicted. One problem that will arise with microarray data (and many other types of high throughput data) is that comparison of microarrays relies on co-normalization and none of the normalization methods in common use let you normalize a new array with out access to all arrays. This new normalization generally alters the estimated expression values on all arrays (although typically by only a small amount).

We now outline the four steps that need to be carried out for classification.

1. Feature selection: includes transformation.
2. Model selection: select a distance, model etc.
3. Model fitting: use the training set to determine the model parameters.
4. Model assessment: use the test set to estimate the error rate.

In all cases much more can be learned by referring to the manual pages and other resources such as **?**. For Bioconductor packages there are additional documentation resources in the form of vignettes and HowTo's that are supplied with each of the packages or from the Bioconductor web site (`www.bioconductor.org`).

1 Gene filtering

One of the first tasks that must be carried out when analysing expression data is to filter out those genes that are unlikely to be of interest. The Affymetrix U68 gene chips have measurements on 7129 different expressed sequence tags (ESTS). Some of these ESTs map to the same genes and others are there for quality control purposes. However, the chips provide estimates of the expression of mRNA for about 6000 different human genes.

In any particular tissue (for these data the tissue of interest is either blood or bone marrow) it has been estimated that about 40 percent of the genome is expressed. There are also certain genes that are expressed at more or less constant levels in all samples, these genes are sometimes referred to as house–keeping genes. Our first step is to remove the unexpressed genes and the house keeping genes so that the computational burden is reduced and we have a better chance of finding relevant information.

Once the filtering has been done the vector `sub` is a logical vector (it contains `TRUE` or `FALSE`) indicating which ESTs have passed the tests. The datasets can then be subset using this vector and we can proceed with our analysis.

Feature Selection

We still have far too many genes to do much with and so we must filter them down a bit. There are many different ways that this can be done.

You could for example use *edd* to find those genes that look like they are mixtures and use only those genes. You could select genes that are used in a particular pathway, genes that show much more variation in one group than in the other. There a very, very many different ways to select genes.

In the interest of expediency we will use the genes selected in Lab 2 by Anova filtering.

```
> data(gfaF)
> gTrA <- gTrnS[gfaF, ]
> gTeA <- gTestS[gfaF, ]
> gMeA <- gMergeS[gfaF, ]
> whBad1 <- (apply(gTrA@exprs, 1, mad) == 0)
> whBad2 <- (apply(gTeA@exprs, 1, mad) == 0)
> whBad <- whBad1 | whBad2
> sum(whBad)
```

```
[1] 37
```

```
> gTrA <- gTrA[!whBad, ]
> gTeA <- gTeA[!whBad, ]
> gMeA <- gMeA[!whBad, ]
```

A problem with the methods used to filter genes now surfaces. We have a number of genes that really are not showing much variation and we will have to remove them to proceed with the analysis. Our standardization procedure will be to subtract the median and divide by the MAD. So we remove all genes for which the MAD is 0. We are now down to 150 genes.

```
> star <- function(x) (x - median(x))/mad(x)
> TrExprs <- t(apply(exprs(gTrA), 1, star))
> TeExprs <- t(apply(exprs(gTeA), 1, star))
> MeExprs <- t(apply(exprs(gMeA), 1, star))
```

This transoformation makes all the genes comparable. This is often reasonable since we have no *a priori* reason to favor one gene over another. Other choices may be appropriate for other situations.

Classification

We now consider some of the different classification tools that are available.

First we consider the discriminant analysis functions.

Discriminant Analysis

```
> gTr.lda <- lda(t(TrExprs), Ytr)
> plot(gTr.lda)
> preds.lda <- predict(gTr.lda, t(TeExprs))
> table(preds.lda$class, Ytest)

            Ytest
             ALL B-cell ALL T-cell AML
  ALL B-cell 12          0          1
  ALL T-cell  7          1          1
  AML         0          0         12
```

An alternative to linear discriminant analysis is logistic discrimination.

```
> library(nnet)
> gTr.mult <- multinom(factor(Ytr) ~ ., data = data.frame(t(TrExprs)),
+     maxit = 250)

# weights:  456 (302 variable)
initial  value 41.747267
iter  10 value 0.007824
iter  20 value 0.000461
iter  30 value 0.000127
final  value 0.000097
converged

> tEdf <- data.frame(t(TeExprs))
> log.preds <- predict(gTr.mult, data.frame(t(TeExprs)))
> table(log.preds, Ytest)

            Ytest
log.preds    ALL B-cell ALL T-cell AML
  ALL B-cell  6          0          1
  ALL T-cell 10          1          1
  AML         3          0         12
```

2 Non-parametric rules

Among the most popular of the non-parametric methods is k-nearest neighbors. The idea is to classify each point according the majority vote of its k nearest neighbors.

Small values of k make the classifier quite local. Large values of k make it more global. The selection of k could be made via cross-validation.

All distances are Euclidean (and there is currently no option to allow you you set it).

So even though most people cluster genomic data using one minus the correlation much of the classification is done on Euclidean distances. This is mainly due to a lack of software that is flexible enough to allow the user to select any distance that they feel is appropriate.

```
> knn1 <- knn(t(TrExprs), t(TeExprs), factor(Ytr), k = 1)
> table(knn1, Ytest)

            Ytest
knn1         ALL B-cell ALL T-cell AML
  ALL B-cell  8          0          1
  ALL T-cell  7          1          1
  AML         4          0         12

> knn3 <- knn(t(TrExprs), t(TeExprs), factor(Ytr), k = 3)
> table(knn3, Ytest)

            Ytest
knn3         ALL B-cell ALL T-cell AML
  ALL B-cell 6          0          3
  ALL T-cell 9          1          2
  AML        4          0          9

> knn5 <- knn(t(TrExprs), t(TeExprs), factor(Ytr), k = 5)
> table(knn5, Ytest)

            Ytest
knn5         ALL B-cell ALL T-cell AML
  ALL B-cell  2          0          1
  ALL T-cell 13          1          3
  AML         4          0         10
```

Cross-validation

We now use the expression values from the merged data set and the `knn.cv` function from the *class* library. This function leaves out each observation in turn and predicts its class on the basis of distances to those samples retained.

```
> knn1.cvpreds <- knn.cv(t(MeExprs), factor(Ymerge), k = 1)
> table(knn1.cvpreds, Ymerge)
```

```
            Ymerge
knn1.cvpreds ALL B-cell ALL T-cell AML
  ALL B-cell 35          1          1
  ALL T-cell  0          8          0
  AML         3          0         24
```

```
> knn3.cvpreds <- knn.cv(t(MeExprs), factor(Ymerge), k = 3)
> table(knn3.cvpreds, Ymerge)
```

```
            Ymerge
knn3.cvpreds ALL B-cell ALL T-cell AML
  ALL B-cell 36          1          3
  ALL T-cell  0          8          0
  AML         2          0         22
```

```
> knn5.cvpreds <- knn.cv(t(MeExprs), factor(Ymerge), k = 5)
> table(knn5.cvpreds, Ymerge)
```

```
            Ymerge
knn5.cvpreds ALL B-cell ALL T-cell AML
  ALL B-cell 36          0          2
  ALL T-cell  0          8          0
  AML         2          1         23
```