Preliminaries

In this lab we explore the use of support vector machines (svm) for classification in microarray experiments. The lab and some of the experiments described here were based on "Microarray Analaysis and Classification by SVM and PAM", by R. Spang and F. Markowetz.

The data is that reported by **?** and contained in the package *golubEsets*. The software for fitting the svms comes from the package *e1071*.

The first step in our process of analysing these data is to perform the basic transformations reported in **?**. The transformations are Windsorizing the data (setting the minimum expression values to 100 and the maximum to 16000).

```
> X <- exprs(golubTrain)
> Wlow <- 100
> Whigh <- 16000
> X[X < Wlow] <- Wlow
> X[X > Whigh] <- Whigh
```

We then filter the test set genes.

```
> mmfilt <- function(r = 5, d = 500, na.rm = TRUE) {
+     function(x) {
+         minval <- min(x, na.rm = na.rm)
+         maxval <- max(x, na.rm = na.rm)
+         (maxval/minval > r) && (maxval - minval > d)
+     }
+ }
> mmfun <- mmfilt()
> ffun <- filterfun(mmfun)
> sub <- genefilter(X, ffun)
> sum(sub)

[1] 3051
```

The filtering process has selected 3051 genes that seem worthy (according to the criteria imposed) of further investigation. The filtering was not especially robust in the sense that it was based on the minimum or maximum expression value between subjects. Using extreme deciles or some other measures would probably be better.

This is a non-specific filter. The genes were selected according to their variability not with respect to their ability to classify any particular set of samples.

```
> X <- X[sub, ]
> dim(X)

[1] 3051   38

> golubTrainSub <- golubTrain[sub, ]
> golubTrainSub@exprs <- X
```

```
> Y <- golubTrainSub$ALL.AML
> Y <- paste(golubTrain$ALL.AML, golubTrain$T.B.cell)
> Y <- sub(" NA", "", Y)
```

In order to make the test set comparable we must select the same set of genes and apply the same transformations to that data set.

```
> Xt <- exprs(golubTest)
> Xt[Xt < Wlow] <- Wlow
> Xt[Xt > Whigh] <- Whigh
> golubTestSub <- golubTest[sub, ]
> golubTestSub@exprs <- Xt[sub, ]
> Ytest <- golubTestSub$ALL.AML
> Ytest <- paste(golubTest$ALL.AML, golubTest$T.B.cell)
> Ytest <- sub(" NA", "", Ytest)
```

In this analysis the genes were selected according to their behavior in the test set. If the same selection criteria is applied to the training set a different set of genes would be selected in the training set. Since we want to compare and combine the analyses that approach cannot be used.

Support Vector Machines

We will make use of the interface to `svm` that requires the specification of a data matrix, `x`, and a response, `y`. The labels in `y` correspond to the rows of `x`.

```
> Xm <- t(exprs(golubTrainSub))
> resp <- golubTrainSub$ALL
> svm1 <- svm(Xm, resp, type = "C-classification", kernel = "linear")
```

We can now explore the prediction training error.

```
> trpred <- predict(svm1, Xm)
> sum(trpred != resp)

[1] 0

> table(trpred, resp)

      resp
trpred ALL AML
   ALL 27   0
   AML  0  11
```

As anticipated there are no errors in the prediction on the training set. We can also employ cross-validation to see what the cross-validated training set error rate is.

```
> trcv <- svm(Xm, resp, type = "C-classification", kernel = "linear",
+     cross = 10)
> summary(trcv)

Call:
 svm.default(x = Xm, y = resp, type = "C-classification", kernel = "linear",      cross = 10)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.0003277614

Number of Support Vectors:  30

 ( 21 9 )


Number of Classes:  2

Levels:
 ALL AML

10-fold cross-validation on training data:

Total Accuracy: 92.10526
Single Accuracies:
 66.66667 75 100 100 100 100 100 100 75 100
```

Of course we also have a test data set and we can use the svm based on the training set to predict the class of samples in the test set.

```
> Xmtr <- t(exprs(golubTestSub))
> tepred <- predict(svm1, Xmtr)
> sum(tepred != golubTestSub$ALL)

[1] 1

> table(tepred, golubTestSub$ALL)

tepred ALL AML
   ALL 20   1
   AML  0  13
```

We see that only one sample is incorrectly predicted. This suggests that svm is a reasonably good classifier.

The cross-validation error rate was estimated at about 5% and on the test set the error rate was 1 in 34 or about 3%. These seem to be in reasonable agreement.

As an exercise you could reverse the rolls of the two data sets, the test set could be treated as the training data set and the training data set could be treated as the test data set.

One group of participants should reverse the rolls and repeat the entire development given above using the training set as the test set and vice versa. Determine how much of the analysis changes.

A second group of participants should follow the following line of argument. If both the test set and the training set represent samples drawn from the same population (which they need to if we are going to use one of them to assess the performance of the other), then we should be able to consider a larger experiment. The two data sets combined simply represent a larger sample. The labels, training set and test set are irrelevant and one should be able to draw (or create) very many pseudo-test and training sets.

An easy way to start exploring that particular setting is to combine the data into one large dataset. This was done to create `golubMerge`. Take `golubMerge` and perform the various processing steps on it. Determine what (if any differences result). Use the merged data set and svm's built in cross-validation procedure to explore the error rates obtained on the combined data set.

For those who get through either (or both) of these exercises quite quickly consider the following

How to efficiently do the exercises

The easiest way to do the exercises that are given here is to use `Sweave` and `Stangle`.

1. Set the working directory to the location of the directory that contains the file `SVM.Rnw`.
2. Run the command, `Stangle("SVM.Rnw", driver=Rtangle())`

This will create file called `SVM.R` which contains all the R commands. You can now open this in your favorite editor and change the names as appropriate.

How good is SVM really?

Spang and Markowitz suggest the following experiment. Suppose that we take the training data set. There are 27 ALL patients and 11 AML patients. In the previous exercises we selected genes that were good predictors of the different classes. Suppose instead we make up some class labels that are not associated with any biologically meaningful variables (that we know of).

```
> newlabs <- sample(c(rep("A", 27), rep("B", 11)), 38)
> table(newlabs, golubTrain$ALL)

newlabs ALL AML
      A 19   8
      B  8   3
```

```
> funnysvm <- svm(Xm, newlabs, type = "C-classification", kernel = "linear")
> fpred <- predict(funnysvm, Xm)
> sum(fpred != newlabs)

[1] 0

> table(fpred, newlabs)

     newlabs
fpred A  B
    A 27  0
    B  0 11
```

Wow, we can predict perfectly! And as for cross-validation, well,

```
> trfcv <- svm(Xm, newlabs, type = "C-classification", kernel = "linear",
+     cross = 10)
> summary(trfcv)

Call:
 svm.default(x = Xm, y = newlabs, type = "C-classification", kernel = "linear",      cross =

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.0003277614

Number of Support Vectors:  34

 ( 23 11 )


Number of Classes:  2

Levels:
 A B

10-fold cross-validation on training data:

Total Accuracy: 63.1579
Single Accuracies:
 100 75 50 25 50 33.33333 75 75 100 50
```

now we see that the error rate is a bit higher than for the correct categories.

However, this does not always turn out to be the case.

So, what is going on? Basically classifiers such as svm, randomForests and neural networks are very flexible and very good at what they are doing. The ability to correctly classifiy (on the training set) is no evidence of anything more than the capability of the classifier and the richness of the feature space.

Random Forests

In this part of the laboratory exercise we will use the random forests (**?**) and the *randomForest* package to further explore the data in the *golubEsets* package.

```
> library(randomForest)
```

Basic use of the random forest technology is fairly straightforward. The only parameter that seems to be very important is `mtry`. This controls the number of features that are selected for each split. The default value is the square root of the number of features but often a smaller value tends to have better performance.

```
> set.seed(123)
> rf1 <- randomForest(t(X), golubTrainSub$ALL, ntree = 2000, mtry = 55,
+     importance = TRUE)
> rf1

Call:
 randomForest.default(x = t(X), y = golubTrainSub$ALL, ntree = 2000,      mtry = 55, importan
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 55

        OOB estimate of  error rate: 7.89%
Confusion matrix:
    ALL AML class.error
ALL  27   0   0.0000000
AML   3   8   0.2727273

> rf2 <- randomForest(t(X), golubTrainSub$ALL, ntree = 2000, mtry = 35,
+     importance = TRUE)
> rf2

Call:
 randomForest.default(x = t(X), y = golubTrainSub$ALL, ntree = 2000,      mtry = 35, importan
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 35

        OOB estimate of  error rate: 5.26%
Confusion matrix:
```

```
    ALL AML class.error
ALL  27   0   0.0000000
AML   2   9   0.1818182
```

Notice that the predictive capabilities of random forests do not seem to be as good as those of svm. Random forests seems to have some difficulties when the sizes of the groups are not approximately equal. There is a `weight` argument that can be given to the random forest function but it appears to have little or no effect.

We can use the prediction function to assess the ability of these two forests to predict the class for the test set.

```
> p1 <- predict(rf1, t(Xt), prox = TRUE)
> table(p1$pred)

ALL AML
 34   0

> p2 <- predict(rf2, t(Xt), prox = TRUE)
> table(p2$pred)

ALL AML
 34   0
```

In both cases the prediction function performs poorly. All samples from the test set were classified as ALL. This seems to be an artifact of the lack of balance between the samples.

```
> gT2 <- golubTestSub[, 7:34]
> rfx <- randomForest(t(exprs(gT2)), gT2$ALL, ntree = 2000, mtry = 55,
+     importance = TRUE)
> rfx

Call:
 randomForest.default(x = t(exprs(gT2)), y = gT2$ALL, ntree = 2000,      mtry = 55, importanc
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 55

        OOB estimate of  error rate: 7.14%
Confusion matrix:
    ALL AML class.error
ALL  13   1  0.07142857
AML   1  13  0.07142857

> rfx2 <- randomForest(t(exprs(gT2)), gT2$ALL, ntree = 2000, mtry = 35,
+     importance = TRUE)
> rfx2
```

```
Call:
 randomForest.default(x = t(exprs(gT2)), y = gT2$ALL, ntree = 2000,      mtry = 35, importanc
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 35

        OOB estimate of  error rate: 7.14%
Confusion matrix:
    ALL AML class.error
ALL  13   1  0.07142857
AML   1  13  0.07142857

> px2 <- predict(rfx2, t(X))
> table(px2, golubTrainSub$ALL)

px2   ALL AML
  ALL 24   0
  AML  3  11
```

So we see that the problem does not seem to be with the technology per se, but rather with its implementation. This is, however, worrying. And one must be quite cautious with all machine learning programs. They have not, in general, been tested under a wide variety of conditions and with a wide variety of inputs.

Feature Selection

One of the nice things about the random forest technology is that it provides some indication of which variables were most important in the classification process. These features can be compared to those selected by t-test or other means.

The current version of *randomForest* produces four different variable importance statistics. Breiman has recently recommended that only two of those be considered (the other two are too unstable). The ones to concentrate on are measures two and four.

In the next code chunk a small function is defined that can be used to extract the most important variables (those with the highest scores).

```
> var.imp.plot(rf1, n.var = 15)
> var.imp.plot(rf2, n.var = 15)
> impvars <- function(x, which = 2, k = 10) {
+     v1 <- order(x$importance[, which])
+     l1 <- length(v1)
+     x$importance[v1[(l1 - k + 1):l1], which]
+ }
> iv.rf1 <- impvars(rf1, k = 25)
> library(hu6800)
> library(annotate)
> isyms <- getSYMBOL(names(iv.rf1), data = "hu6800")
```

In **?** the authors identified 50 genes that were most highly related to the classes. We could compare the genes selected by random forests with those listed by Golub.

0.1 Exercises

Again a number of interesting exercises present themselves.

1. Reverse the role of the test set and the training set and see how the estimated prediction errors change.

2. Use the combined data set (`golubMerge`) to build a random forest (probably select a subset with equal numbers of AML and ALL). How well does it do?

3. Compare the variable importance measures obtained using the balanced subset from the training set with those selected using the whole data set. Do the important variables change very much?