Machine Learning Lab

R. Gentleman, V. Carey, W. Huber

June 11, 2004

Contents

1 Preliminaries

In this lab we explore the use of trees, nnets, support vector machines (SVM) and random forests for classification in microarray experiments. The lab and some of the experiments described here were based on the exercises "Microarray Analaysis and Classification by SVM and PAM" by R. Spang and F. Markowetz from the NGFN 1 microarray courses.

The data is that reported by Chiaretti *et al* and contained in the package *ALL*. 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; we reduce the data to a comparison of BCR/ABL and NEG.

```
> data("ALL")
> sel1 <- grep("^B", as.character(ALL$BT))
> sel2 <- which(as.character(ALL$mol) %in% c("BCR/ABL", "NEG"))
> eset <- ALL[, intersect(sel1, sel2)]
```

We first apply a non-specific filtering step to remove those genes which show low values of expression or which show little variation across samples. Note that the non-specific filtering step does **not** select genes with respect to their ability to classify any particular set of samples.

Note however that, if you want to use cross-validation to estimate the performance of a classifier (e.g. mis-classification rate), the variable selection step should be included in the cross-validation (3).

```
> library(genefilter)
> f1 <- pOverA(0.25, log2(200))
> f2 <- function(x) (IQR(x) > 0.5)
> ff <- filterfun(f1, f2)
> selected <- genefilter(eset, ff)
> sum(selected)

[1] 1439

> esetSub <- eset[selected, ]
```

The filtering has selected 1439 genes that seem worthy (according to the criteria imposed) of further investigation.

2 An analysis using rpart

First, let's set up a response variable `Y` and explanatory variables (genes) `X`:

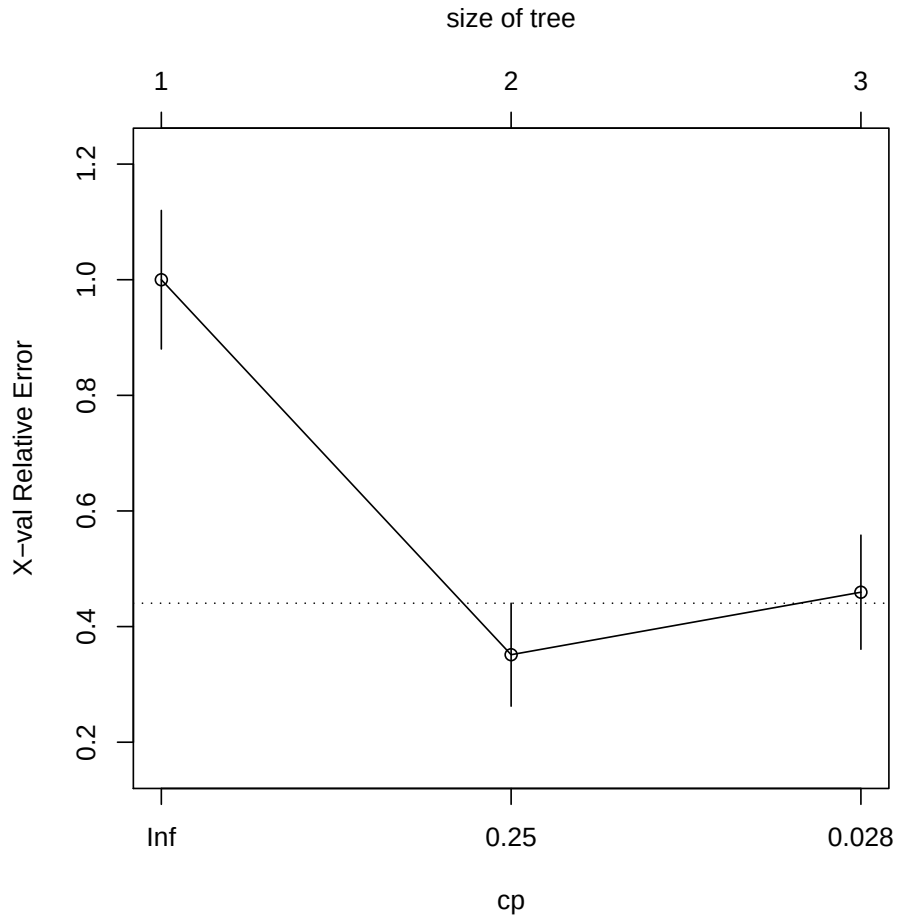
```
> Y <- factor(esetSub$mol.biol)
> table(Y)

Y
BCR/ABL     NEG
     37      42

> X <- t(exprs(esetSub))

> library(rpart)
> df <- data.frame(Y = Y, X = X)
> tr <- rpart(Y ~ ., data = df)

> plotcp(tr)
```



This seems strange. The best cross-validated ("X-val") relative error is obtained with a *two node* tree.

```
> print(tr)

n= 79

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 79 37 NEG (0.4683544 0.5316456)
  2) X.1636_g_at>=9.35504 32  2 BCR/ABL (0.9375000 0.0625000) *
  3) X.1636_g_at< 9.35504 47  7 NEG (0.1489362 0.8510638)
    6) X.2025_s_at< 8.166946 7  2 BCR/ABL (0.7142857 0.2857143) *
    7) X.2025_s_at>=8.166946 40  2 NEG (0.0500000 0.9500000) *

> print(summary(tr))

Call:
rpart(formula = Y ~ ., data = df)
```

```
  n= 79

         CP nsplit rel error    xerror       xstd
1 0.75675676      0 1.0000000 1.0000000 0.11986993
2 0.08108108      1 0.2432432 0.3513514 0.08906937
3 0.01000000      2 0.1621622 0.4594595 0.09871997

Node number 1: 79 observations,    complexity param=0.7567568
  predicted class=NEG      expected loss=0.4683544
    class counts:    37    42
   probabilities: 0.468 0.532
  left son=2 (32 obs) right son=3 (47 obs)
  Primary splits:
      X.1636_g_at < 9.35504  to the right, improve=23.67688, (0 missing)
      X.39730_at  < 9.207515 to the right, improve=22.26650, (0 missing)
      X.1635_at   < 8.073684 to the right, improve=19.29742, (0 missing)
      X.40202_at  < 8.95228  to the right, improve=17.82382, (0 missing)
      X.37027_at  < 9.146563 to the right, improve=12.88599, (0 missing)
  Surrogate splits:
      X.39730_at < 9.220421 to the right, agree=0.975, adj=0.938, (0 split)
      X.1635_at  < 8.073684 to the right, agree=0.937, adj=0.844, (0 split)
      X.40202_at < 8.95228  to the right, agree=0.797, adj=0.500, (0 split)
      X.33924_at < 8.063734 to the right, agree=0.772, adj=0.438, (0 split)
      X.31536_at < 8.694725 to the right, agree=0.759, adj=0.406, (0 split)

Node number 2: 32 observations
  predicted class=BCR/ABL  expected loss=0.0625
    class counts:    30     2
   probabilities: 0.938 0.062

Node number 3: 47 observations,    complexity param=0.08108108
  predicted class=NEG      expected loss=0.1489362
    class counts:     7    40
   probabilities: 0.149 0.851
  left son=6 (7 obs) right son=7 (40 obs)
  Primary splits:
      X.2025_s_at  < 8.166946 to the left,  improve=5.257751, (0 missing)
      X.36125_s_at < 6.682295 to the left,  improve=5.168948, (0 missing)
      X.37027_at   < 9.168073 to the right, improve=4.515904, (0 missing)
      X.1403_s_at  < 8.042473 to the right, improve=4.448227, (0 missing)
      X.1288_s_at  < 12.31291 to the left,  improve=4.370022, (0 missing)
  Surrogate splits:
      X.33425_at   < 8.93569  to the left,  agree=0.979, adj=0.857, (0 split)
      X.36125_s_at < 6.632967 to the left,  agree=0.979, adj=0.857, (0 split)
      X.37329_at   < 7.395774 to the left,  agree=0.979, adj=0.857, (0 split)
```

```
      X.37364_at   < 8.401382 to the left,  agree=0.979, adj=0.857, (0 split)
      X.39110_at   < 8.633084 to the left,  agree=0.979, adj=0.857, (0 split)

Node number 6: 7 observations
  predicted class=BCR/ABL  expected loss=0.2857143
    class counts:     5     2
   probabilities: 0.714 0.286

Node number 7: 40 observations
  predicted class=NEG      expected loss=0.05
    class counts:     2    38
   probabilities: 0.050 0.950

n= 79

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 79 37 NEG (0.4683544 0.5316456)
  2) X.1636_g_at>=9.35504 32  2 BCR/ABL (0.9375000 0.0625000) *
  3) X.1636_g_at< 9.35504 47  7 NEG (0.1489362 0.8510638)
    6) X.2025_s_at< 8.166946 7  2 BCR/ABL (0.7142857 0.2857143) *
    7) X.2025_s_at>=8.166946 40  2 NEG (0.0500000 0.9500000) *
```

Now we see what is happening. The chosen gene creates two nearly pure nodes. And several other genes do this.

How strongly conditioned is this result on this particular dataset? To answer that question you can randomly split your data into a test set and a training set. You fit a model on the test set and assess it on the training set.

3 Test and training set

So, let us randomly divide the data into two groups, so that we can use them for test and training purposes.

```
> a1 = which(as.character(Y) == "NEG")
> a2 = which(as.character(Y) == "BCR/ABL")
> set.seed(100)
> sub1N = sample(a1, 21)
> sub1B = sample(a2, 19)
> sub2N = a1[!(a1 %in% sub1N)]
> sub2B = a2[!(a2 %in% sub1B)]
> testSet = esetSub[, c(sub1N, sub1B)]
> testY = Y[c(sub1N, sub1B)]
> trainSet = esetSub[, c(sub2N, sub2B)]
> trainY = Y[c(sub2N, sub2B)]
```

Exercise 1

- *Write a function that will create a data set divided into two groups at at given proportion (e.g. 75% training and 25% test set). It should take an* `exprSet` *object and the name of a two class variable in its* `pData` *slot on which to balance.*

Exercise 2

Now we can return to the classification tree from Section 2.

1. *Fit a model using the genes in* `trainSet`*. Assess the fit of that model.*
2. *Now use those genes to fit a model using those best performing genes in your test data set.*
3. *Carry out a different split of your data into test and training sets and repeat.*

4 Neural network

Unfortunately, we cannot use all the genes as inputs to the current nnet implementation, at least on a modestly endowed machine. We'll use a randomly reduced set of genes.

```
> library(nnet)
> df <- data.frame(Y = Y, X = X[, sample(ncol(X), 150)])
> nn1 <- nnet(Y ~ ., data = df, size = 5, decay = 0.01, MaxNWts = 5000)

> print(nn1)

a 150-5-1 network with 761 weights
inputs: X.34891_at X.41138_at X.40103_at X.40634_at X.38251_at X.36979_at X.40108_at X.41187_
output(s): Y
options were - entropy fitting  decay=0.01

> print(table(predict(nn1, type = "class"), Y))

         Y
          BCR/ABL NEG
  BCR/ABL 37       0
  NEG      0      42
```

Exercise 3

1. *Is this a reproducible analysis? How can it be made so?*
2. *Does the number of iterations need to be increased?*
3. *Use the tune.nnet procedure of package e1071 to choose effective size and decay parameters that may lead to a more parsimonious model.*
4. *Apply this procedure to your test set and training set.*

5 k nearest neighbors

The user interface of this procedure is a bit unusual: it asks for the matrix of 'training records', a matrix of 'test records', the true classes for the training records, and then parameters of the procedure. Here's a simple example.

```
> library(class)
> knn1 <- knn(t(exprs(trainSet)), t(exprs(testSet)), trainY, k = 1)
> table(knn1, testY)
```

```
         testY
knn1      BCR/ABL NEG
  BCR/ABL 15       7
  NEG      4      14
```

Exercise 4

1. *Repeat the exercise with different values of k and l.*
2. *Use knn.cv on the original data. What does it do? How does its performance compare to that of the separate test and training sets.*

6 Support Vector Machines

The function `svm` asks for the specification of a data matrix `x` and a response `y`. The labels in `y` correspond to the rows of `x`.

```
> Xm <- t(exprs(trainSet))
> svm1 <- svm(Xm, trainY, type = "C-classification", kernel = "linear")
```

We can now explore the prediction training error.

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

```
[1] 0
```

```
> table(trpred, trainY)
```

```
         trainY
trpred    BCR/ABL NEG
  BCR/ABL 18       0
  NEG      0      21
```

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, trainY, type = "C-classification", kernel = "linear",
+     cross = 10)
> summary(trcv)
```

```
Call:
 svm.default(x = Xm, y = trainY, type = "C-classification", kernel = "linear",      cross = 1

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.000694927

Number of Support Vectors:  36

 ( 18 18 )


Number of Classes:  2

Levels:
 BCR/ABL NEG

10-fold cross-validation on training data:

Total Accuracy: 79.48718
Single Accuracies:
 66.66667 50 75 100 75 100 75 75 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(testSet))
> tepred <- predict(svm1, Xmtr)
> sum(tepred != testY)

[1] 6

> table(tepred, testY)

         testY
tepred    BCR/ABL NEG
  BCR/ABL 15       2
  NEG      4      19
```

Exercise 5

1. *How many samples are misclassified?*
2. *Does* `svm` *seem to work better or worse than* `knn`*?*
3. *Compare the test and training set error rates.*

4. *Reverse the roles of the test and training data sets.*

5. *Repeat the cross-validation part using the whole data set.*

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.

6.1 How good is SVM really?

Spang and Markowitz suggest the following experiment. Suppose that we take the training data set. In our training set there are 18 samples with BCR/ABL and 21 NEG samples. 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("B", 18), rep("N", 21)), 39)
> table(newlabs, trainY)

       trainY
newlabs BCR/ABL NEG
      B 11       7
      N  7      14

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

[1] 0

> table(fpred, newlabs)

     newlabs
fpred B  N
    B 18  0
    N  0 21
```

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:  0.000694927

Number of Support Vectors:  38

 ( 20 18 )


Number of Classes:  2

Levels:
 B N

10-fold cross-validation on training data:

Total Accuracy: 43.58974
Single Accuracies:
 33.33333 25 0 50 75 25 50 50 75 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 classify (on the training set) is no evidence of anything more than the capability of the classifier and the richness of the feature space.

7 Random Forests

In this part of the laboratory exercise we will use the random forests (1; 1) 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(Xm, trainY, ntree = 2000, mtry = 55, importance = TRUE)
> rf1
```

```
Call:
 randomForest.default(x = Xm, y = trainY, ntree = 2000, mtry = 55,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 55

        OOB estimate of  error rate: 15.38%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      15   3   0.1666667
NEG           3  18   0.1428571
```

```
> rf2 <- randomForest(Xm, trainY, ntree = 2000, mtry = 35, importance = TRUE)
> rf2

Call:
 randomForest.default(x = Xm, y = trainY, ntree = 2000, mtry = 35,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 35

        OOB estimate of  error rate: 23.08%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      14   4   0.2222222
NEG           5  16   0.2380952
```

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, Xm, prox = TRUE)
> table(trainY, p1$pred)

trainY    BCR/ABL NEG
  BCR/ABL 18       0
  NEG      0      21
```

```
> p2 <- predict(rf2, Xm, prox = TRUE)
> table(trainY, p2$pred)

trainY    BCR/ABL NEG
  BCR/ABL 18       0
  NEG      0      21
```

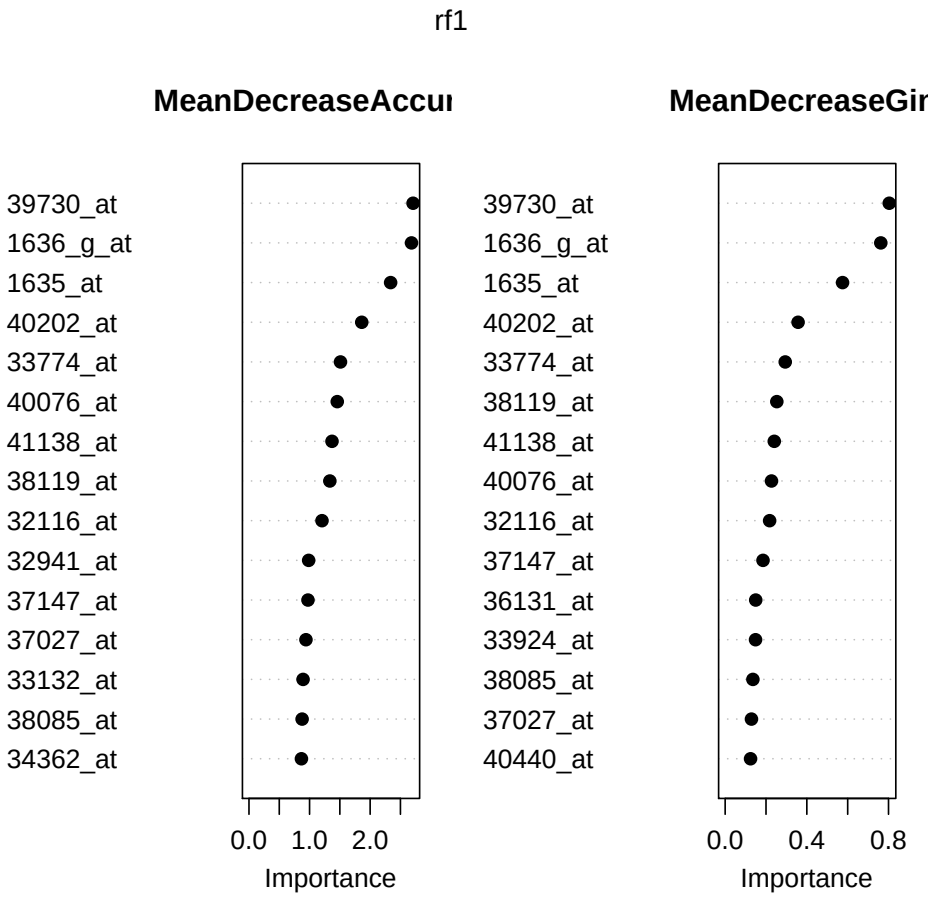
7.1 Feature Selection

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.

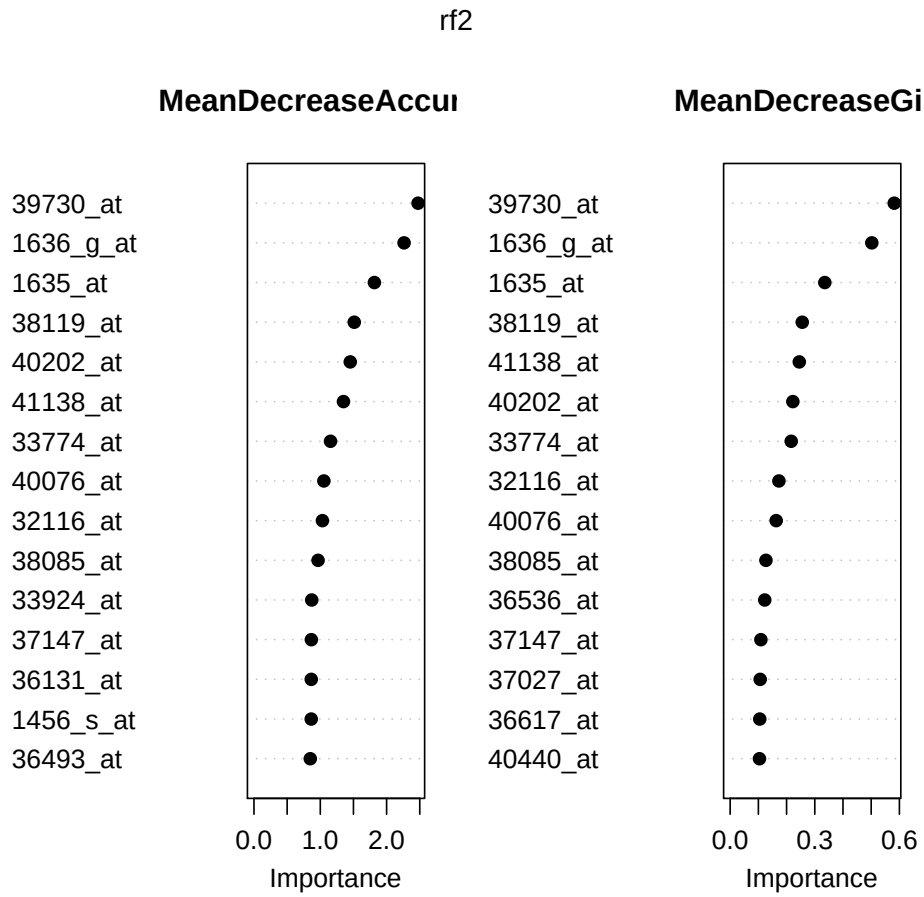
The current version of *randomForest* produces four different variable importance statistics. Breiman has recently recommended that only two of those be considered (the other two are too unstable). The ones to concentrate on are measures two and four.

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("hgu95av2")
> library(annotate)
> isyms <- getSYMBOL(names(iv.rf1), data = "hgu95av2")
```

7.2 More exercises

Again a number of interesting exercises present themselves.

Exercise 6

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

2. *Use the whole data set to build a random forest. How well does it do?*

There are very many other classifiers available to you.

- Linear discriminant analysis `lda`, from the *MASS* package.
- Logistic discrimination, `multinom`, in *nnet*.
- Generalized partial least squares, *gpls*.
- So-called "Prediction Analysis for Microarrays", *rpam*.

References

[1] Breiman L. Random forests - random features. *Preprint Department of Statistics, U.C. Berkeley, no. 567* (1999).

[2] Breiman L. Notes on setting up, using, and understanding random forests V3.0 (2001).

[3] Ambroise C and McLachlan GJ. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proc Natl Acad Sci* 99(10): 6562-6 (2002).