# 1

# Machine Learning, Part I

Robert Gentleman, Wolfgang Huber, Vince Carey, Raphael Irizarry

Abstract

In this lab we will cover some of the basic principles of machine learning. We will use the ALL data set and will work on two different problems. For one of them it is relatively easy to classify the samples and for the other, it is harder. You will be introduced to some of the basic concepts in machine learning such as the distance function, supervised and unsupervised machine learning, as well as the so-called *confusion* matrix.

## 1.1 Introduction

Fundamental to the task of machine learning is selecting a distance. In many cases it is more important than the choice of classification method (you might want to try some different choices for distances in the problems below and see what changes). Feature selection is also an important problem. We suggest that you take a simple approach and use genes which are differentially expressed between the phenotypes under study. In some cases this can be improved on, but in general it seems to be a reasonable approach. In most cases we have no *a priori* reason to believe that any one gene should get more weighting than another. If that is true, then we must standardize the genes before carrying out machine learning. If we do not standardize them (for each gene, subtract some measure of the center and divide by some measure of the variability, across samples), then many machine learning algorithms (and distances) will treat different genes quite differently, typically depending on their observed mean expression level and its variation across samples. So, standardization is recommended, however, it raises an important prerequisite. If you decide to standardize your expression data you will need to perform some sort of non-specific filtering to remove genes that have low variability, for example because they are not expressed, or because the microarray experiment did not work for these genes due to low labeling or hybridization efficiencies. The reason you must do this is that we do not want to amplify what is essentially *noise* by the operation of standardization.

### *1.1.1 Machine Learning Check List*

1. Filter out features (genes) which show little variation across samples, or which are known not to be of interest. If appropriate transform features to all be on the same scale.
2. Select a distance measure. What does it mean for two genes to be close? Make sure that the selected distance embodies your notion of similarity.
3. Feature selection: select features to be used for machine learning.
4. Select the algorithm: which of the very many machine learning algorithms do you want to use?
5. Assess the performance of your analysis. If performing supervised machine learning performance is often assessed using cross-validation. For unsupervised machine learning (or clustering) it is more difficult to determine how well the algorithm has performed.

## Non-specific filtering

First load the **Biobase** and **ALL** packages and then use the *data* function to load the `ALL` data. Since the data in `ALL` are large and phenotypically quite diverse, we reduce the cases down to a reasonable two group comparison. We will return to a multigroup comparison later.

```
> library("Biobase")
> library("ALL")
> data(ALL, package = "ALL")
> ALLBs = ALL[, grep("^B", as.character(ALL$BT))]
> ALLBCRNEG = ALLBs[, ALLBs$mol == "BCR/ABL" | ALLBs$mol == "NEG"]
> ALLBCRNEG$mol.biol = factor(ALLBCRNEG$mol.biol)
> numBN = length(ALLBCRNEG$mol.biol)
> ALLBCRALL1 = ALLBs[, ALLBs$mol == "BCR/ABL" | ALLBs$mol == "ALL1/AF4"]
> ALLBCRALL1$mol.biol = factor(ALLBCRALL1$mol.biol)
> numBA = length(ALLBCRALL1$mol.biol)
```

**Question 1**
*How many samples are in the BCR/ABL-NEG subset? How many are in the BCR/ABL-ALL1/AF4 subset?*

You now have two data sets to work with. Most of the code for carrying out machine learning can easily be applied to either data set. The comparison of BCR/ABL to NEG is difficult, and the error rates are typically quite high. On the other hand, the comparison of BCR/ABL to ALL1/AF4 is rather easy, and the error rates should be small. In this lab we will first select some genes to use as features for the rest of the lab. Next we will use those features to do some machine learning, in particular we will make use of cross-validation to select parameters of the classification model and see how to assess the model itself. Many of the details can be explored in much more detail, and some suggestions are made.

## Preprocessing

First carry out non-specific filtering, as described in the Differential Expression Lab. You should remove those genes that you think are not sufficiently informative to be considered further. One recommendation is to filter on variability. Here, we take the simplistic approach of using the $75^{th}$ percentile of the interquartile range (IQR) as the cut-off point. We do this because we want to have relatively few genes to deal with so the examples will run quickly on laptops. Finding the IQR can be done either by applying the `IQR` function over all rows of the *ExpressionSet*, or by manually computing quantiles using the very fast function `rowQ` (there are some slight, hardly relevant numerical differences).

```
> lowQ = rowQ(ALLBCRNEG, floor(0.25 * numBN))
> upQ = rowQ(ALLBCRNEG, ceiling(0.75 * numBN))
> iqrs = upQ - lowQ
> giqr = iqrs > quantile(iqrs, probs = 0.75)
> sum(giqr)
```

```
[1] 3156
```

```
> BNsub = ALLBCRNEG[giqr, ]
```

**Exercise 1**
*What kind of object is* `BNsub`*?*

## 1.2 Selecting a Distance

To some extent your choices here are not always that flexible because many machine learning algorithms have the distance measure fixed in advance. There are a number of different tools that you can use in R to compute the distance between objects. They include the function `dist`, the function `daisy` from the **cluster** package (**?**), and the functions in the **bioDist** package. The **bioDist** package is discussed in Chapter 12 of **?**. Some ideas on visualizing distance measures can be found in Section 10.5 of that same reference.

**Exercise 2**
*What distance measures are availble in the* ***bioDist*** *package? Hint: load the package and then look at the loaded functions, or read the vignette.*

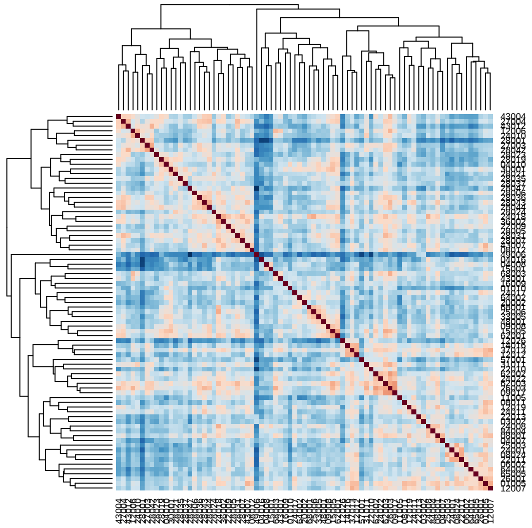


Figure 1.1. A heatmap of the between-sample distances.

To make the computations easier, we take the first sixty genes from the `BNsub` data set and use those for the exercises in this section. The `dist` function computes the distance between rows of an input matrix. Since we want the distances between samples, we transpose the matrix using the function `t`. The return value is an instance of the *dist* class and you should read the manual page carefully to find out more about this class. Since this class is not supported by some R functions we will want to use, we also convert it to a matrix.

```
> dSub <- BNsub[1:60, ]
> eucD <- dist(t(exprs(dSub)))
> eucD@Size

[1] 79

> eucM <- as.matrix(eucD)
```

We can use this as an input to various clustering algorithms and plot the outputs. But for now we want to visualize it as a heatmap.

```
> library("RColorBrewer")
> hmcol <- colorRampPalette(brewer.pal(10, "RdBu"))(256)
> heatmap(eucM, sym = TRUE, col = hmcol, distfun = function(x) as.dist(x))
```

**Question 2**
*What do you notice most about the heatmap? What color is used to encode objects that are similar? What color encodes objects that are dissimilar?*

**Question 3**
*Repeat this analysis using Kendall's tau distance. How much does the heatmap change?*

Since our goal is to introduce you to a number of different distances and to help you understand their effects, visualization is important. We will also create a few helper functions to make it easier to carry out certain transformations and calculations. First we define a function to find the closest neighbor of a particular observation given a distance matrix and a label specifying an observation in the distance matrix.

```
> closestN = function(distM, label) {
+     loc = match(label, row.names(distM))
+     names(which.min(distM[label, -loc]))
+ }
> closestN(eucM, "03002")
```

```
[1] "22013"
```

**Exercise 3**
*Compute the distance between the samples using the* `MIdist` *function from the* ***bioDist*** *package. What distance does this function compute? Which sample is closest to* `"03002"` *in this distance?*

## Feature Selection

Now we are ready to select features. Perhaps the easiest approach to feature selection is to use a $t$-test.

```
> library("genefilter")
> tt1 = rowttests(BNsub, "mol.biol")
> numToSel <- 50
```

Using the $t$-test statistics, we will select the top 50 genes to use for the machine learning questions below.

```
> tt1ord = order(abs(tt1$statistic), decreasing = TRUE)
> top50 = tt1ord[1:numToSel]
> BNsub1 = BNsub[top50, ]
```

**Exercise 4**
*What is the value of the largest* $t$*-statistic? Which gene does it correspond to? What is the corresponding* $p$*-value?*

Next we will standardize all gene expression values. As discussed above, it is important that non-specific filtering has already been applied, otherwise the standardization step will add unnecessary noise to the data. Since we will compute IQR by row many times in the next code chunk, we first write a helper function to compute this for us.

```
> rowIQRs = function(eSet) {
+     numSamp = ncol(eSet)
+     lowQ = rowQ(eSet, floor(0.25 * numSamp))
+     upQ = rowQ(eSet, ceiling(0.75 * numSamp))
+     upQ - lowQ
+ }
```

**Exercise 5**
*Use the* `rowIQRs` *function to repeat the IQR calculation that was carried out previously. Do you get the same values?*

Now we are ready to standardize all genes, which we will do by subtracting the row medians and dividing by the row IQRs. Again, we write a helper function, `standardize`, that will do most of the work.

```
> standardize = function(x) (x - rowMedians(x))/rowIQRs(x)
> exprs(BNsub1) = standardize(exprs(BNsub1))
```

Take a quick look at the data to verify that everything went as intended.

```
> library("RColorBrewer")
> hmcol <- colorRampPalette(brewer.pal(10, "RdBu"))(256)
> spcol <- ifelse(BNsub1$mol.biol == "BCR/ABL", "goldenrod", "skyblue")
> heatmap(exprs(BNsub1), col = hmcol, ColSideColors = spcol)
```

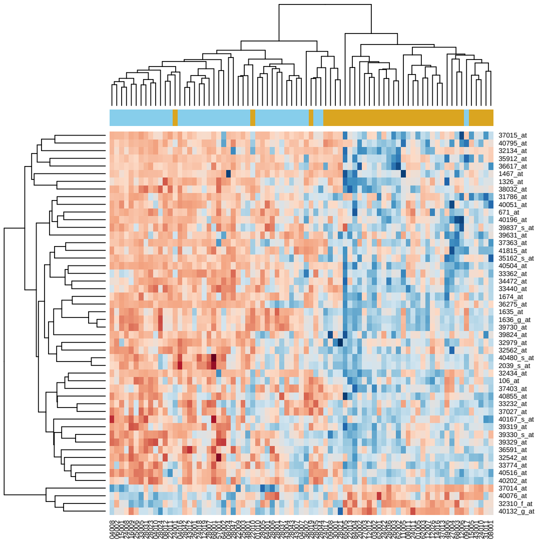


Figure 1.2. Heatmap.

**Exercise 6**

a *What do we expect to see in the heatmap? Do we see that?*

b *What color corresponds to high values of expression?*

c ***Optional:** Repeat the calculations to this point using* `ALLBCRALL1`*.*

d ***Optional**: Use either the **ROC** package or the **edd** package to select genes for the machine learning portion. Alternatively you could use genes in a GO category or a KEGG pathway (but you still want to use only those that passed your non-specific filter).*

Make sure that you standardize the gene expression data once you have selected your set of interesting genes. This standardization insures that all genes have equal weighting in the machine learning exercises below.

## 1.3 Machine Learning

There are many different machine learning algorithms available in R. You may use which ever one you would like, we suggest using $k$ nearest neighbors for this lab since it is conceptually simple and can be used to demonstrate most of the general principles. We also recommend that you use the **MLInterfaces** package. The reason for this suggestion is that this package provides a uniform set of calling parameters and a uniform return value which will make it easier to switch your code from one machine learning algorithm to another. This package does not implement any of the machine learning algorithms, it just provides a set of interfaces and in general the name of the function or method remains the same, but a B is post-pended, so we will use `knnB` and `knn.cvB`.

**Exercise 7**
*Use the knn method to estimate the prediction error rate. If you are ambitious you could try to do this with something more sophisticated than leave-one-out cross-validation.*

Some example code is given below, but you will need to modify it to answer the questions that have been posed.

```
> library("class")
> a1 = knn.cv(t(exprs(BNsub1)), BNsub1$mol.biol)
> ctab1 = table(a1, BNsub1$mol.biol)
> errrate = (ctab1["BCR/ABL", "NEG"] + ctab1["NEG", "BCR/ABL"])/sum(ctab1)
```

**Exercise 8**
*Use cross-validation to estimate $k$, the number of nearest neighbors to use. That is, for each of a number of values of $k$, estimate the cross-validation error, and then select $k$ as that value which yields the smallest error rate.*

Again, the code below is intended solely to get you started, it does not represent a complete solution to the question, you must modify it.

```
> alist = list()
> for (i in 1:4) alist[[i]] = knn.cv(t(exprs(BNsub1)), BNsub1$mol.biol,
+     k = i)
```

```
> sapply(alist, function(x) {
+     ct1 = table(x, BNsub1$mol.biol)
+     (ct1["BCR/ABL", "NEG"] + ct1["NEG", "BCR/ABL"])/sum(ct1)
+ })
```

```
[1] 0.05063291 0.06329114 0.07594937 0.06329114
```

**Exercise 9**

a *What happens when $k$ is even and there is a tie?*

b ***Optional:** Suppose that instead of Euclidean distance you wanted to use some other metric, such as 1-correlation. How might you achieve that?*

c *How might you define outlier and doubt classes? Are there any outliers, or hard to classify samples?*

### 1.3.1 MLInterfaces

We now repeat some of the previous calculations using the **MLInterfaces** package. Load the library and explore its documentation.

```
> library("MLInterfaces")
```

**Exercise 10**

a *Use `library(help=MLInterfaces)`, `?"MLearn-methods"` and `openVignette()` to explore the package.*

b *Try to follow the example at the bottom of the `MLearn-methods` help page. Depending on the packages installed on your computer, you might have luck with the command `example("MLearn-methods")`.*

A key function is `MLearn`. `MLearn` is designed for easy use with expression data. The first argument is the name of variable containing *a priori* classification information, e.g., `mol.biol`. The second argument is an instance of the *ExpressionSet* class, the third argument the name of the machine learning algorithm, and the fourth argument the individuals to be used for training. So to use the $k$ nearest neighbors machine learning algorithm using the first 50 samples for training, do the following:

```
> knnResult <- MLearn(mol.biol ~ ., BNsub1, "knn", 1:50)
> knnResult
```

```
MLOutput instance, method= knn
Call:
 MLearn(formula = formula, data = data, method = method, trainInd = trainInd,
    mlSpecials = mlSpecials)
predicted class distribution:
BCR/ABL     NEG
     14      15
```

**Exercise 11**

a *Interpret each line of the input to `MLearn`.*

(a) *What would you do to change the training set?*
(b) *To use every second sample as the training set?*
(c) *To use all but the last sample for training?*
(d) *To use a training set of 50 individuals, chosen at random from the samples in* `BNsub1` *(hint: use the* `sample` *function).*

b *Interpret the output of* `MLearn`*. In particular, look at the predicted class distribution and check that the right number of samples are being used for testing.*

The confusion matrix compares the known classification of the testing set with the predicted classification based on the tuned machine learning algorithm.

```
> confuMat(knnResult)

         predicted
given     BCR/ABL NEG
  BCR/ABL      12   0
  NEG           2  15
```

**Exercise 12**

a *Interpret the confusion matrix. How well do you think the algorithm is doing? What might you do to improve the classification?*

b *What other information can you extract from the fitted model?*

## Cross-validation

Cross-validation is often used to assess the prediction error of supervised machine learning. In order to get an accurate assessment it is important that all steps that can affect the outcome are included in the cross-validation process. In particular, the selection of features to use in the machine learning algorithm must be included within the cross-validation step. The **MLInterfaces** package has a method for performing cross-validation. The method is called `xval`. Ponder its help page (`?xval`) and think about how you might perform cross-validation of `BNsub1`. From the `xval` help page, it looks like we should be able to perform cross validation with a command like:

```
> knnXval <- xvalML(mol.biol ~ ., data = BNsub1, "knn", xvalMethod = "LOO")
```

The first two arguments should be familiar. The third argument, `knnB`, specifies that we will use the `knn` function. The final argument, *xvalMethod*, indicates the method that will be used for cross-validation. The cryptic *”LOO”* stands for leave-one-out.

**Exercise 13**

a *Describe in words the operation that* `xval` *is performing.*

b *What is the length of* `knnXval`*? Why?*

c *Interpret the meaning of each element in* `knnXval`*.*

d *What information is provided by the following command? How would you use this to assess the performance of this machine learning algorithm?*

```
> table(given = BNsub1$mol.biol, predicted = knnXval)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      35   2
  NEG           2  40
```

Now, let's see what happens when we include feature selection in the cross-validation algorithm.

```
> BNx = BNsub
> exprs(BNx) = standardize(exprs(BNx))
> t.fun <- function(data, fac) {
+     (abs(rowttests(data, data[[fac]], tstatOnly = FALSE)$statistic))
+ }
> lk3f <- xvalML(mol.biol ~ ., data = BNx, "knn", xvalMethod = "LOO",
+     fsFun = t.fun, fsNum = 50)
> table(given = BNx$mol.biol, predicted = lk3f$out)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      33   4
  NEG           4  38
```

**Exercise 14**

a *In the example above we used 50 features for each of the cross-validations. What happens if we use twice as many? What happens if we only use 5? How would you interpret these results?*

b ***Optional: Hard*** *Repeat the exercise above using 10 fold cross-validation. To do this you will need to divide the data into 10 groups and use the group argument to* `xval`.

c *Next, use* `xval` *with a different classifier, such as support vector machines (the function is* `svmB`*).*

## Multi-group machine learning

The part of the exercise described here is **optional**, but it does raise some interesting issues. We briefly consider the application of supervised machine learning methods to a mult-class problem. We will return to our original data, and instead of creating a two class problem, we will create a three class problem.

Instead of treating this as two separate two class problems make one data set that has all three phenotypes. Now use the kNN procedure to make class predictions. Can you estimate the class conditional error rates? Can you control the procedure so that the class-conditional error rates are treated equally?

## 1.4 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)

randomForest 4.5-18
Type rfNews() to see new features/changes/bug fixes.
```

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)
> trainY = BNsub$mol.biol[TrainInd]
> Xm = t(exprs(BNsub)[, TrainInd])
> rf1 <- randomForest(Xm, trainY, ntree = 2000, mtry = 55, importance = TRUE)
> rf1

Call:
 randomForest(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: 12.5%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      17   3        0.15
NEG           2  18        0.10

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

Call:
 randomForest(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: 22.5%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      17   3        0.15
NEG           6  14        0.30

> vcrf1 = MLearn(mol.biol ~ ., data = BNsub, "randomForest", TrainInd,
+     ntree = 2000, mtry = 55, importance = TRUE)
> vcrf1

MLOutput instance, method= randomForest
Call:
 MLearn(formula = formula, data = data, method = method, trainInd = trainInd,
    mlSpecials = mlSpecials, ntree = 2000, mtry = 55, importance = TRUE)
predicted class distribution:
BCR/ABL     NEG
     24      15
```

```
> vcrf2 = MLearn(mol.biol ~ ., data = BNsub, "randomForest", TrainInd,
+     ntree = 2000, mtry = 35, importance = TRUE)
> vcrf2
```

```
MLOutput instance, method= randomForest
Call:
 MLearn(formula = formula, data = data, method = method, trainInd = trainInd,
    mlSpecials = mlSpecials, ntree = 2000, mtry = 35, importance = TRUE)
predicted class distribution:
BCR/ABL     NEG
     22      17
```

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      20   0
  NEG           0  20
```

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

```
trainY    BCR/ABL NEG
  BCR/ABL      20   0
  NEG           0  20
```

```
> confuMat(vcrf1)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      15   2
  NEG           9  13
```

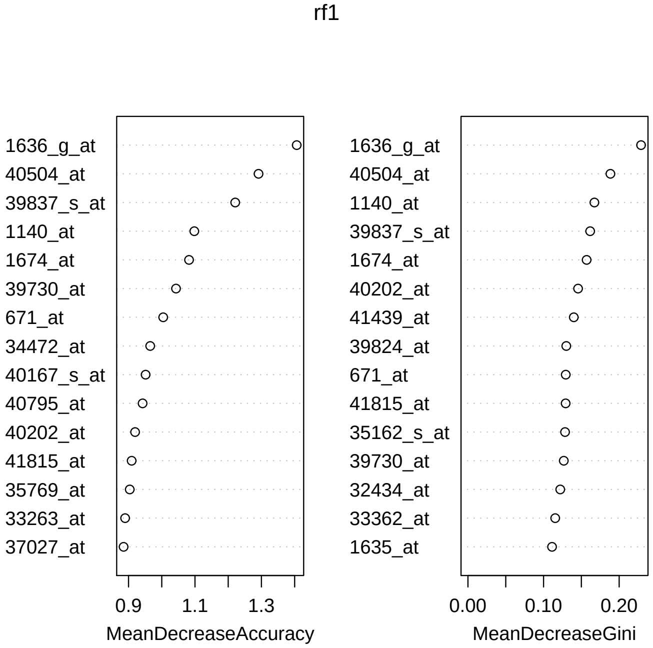
```
> confuMat(vcrf2)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      14   3
  NEG           8  14
```

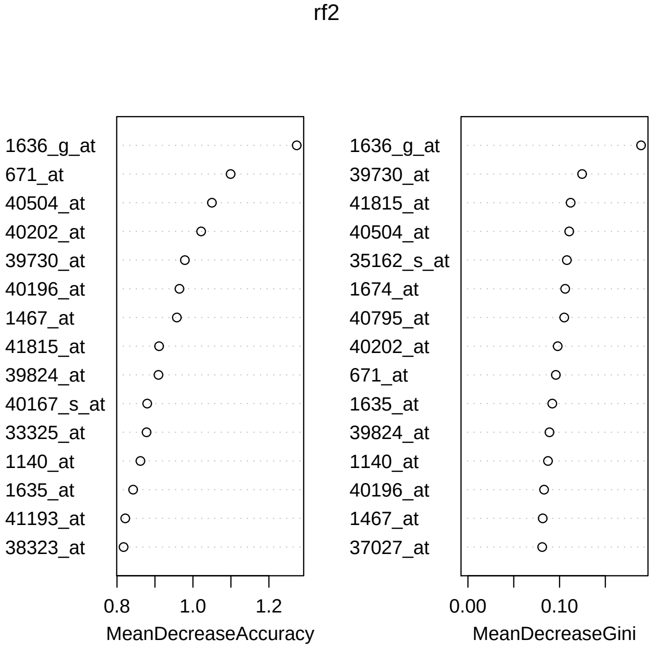
### *1.4.1 Feature Selection*

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).

```
> varImpPlot(rf1, n.var = 15)
```

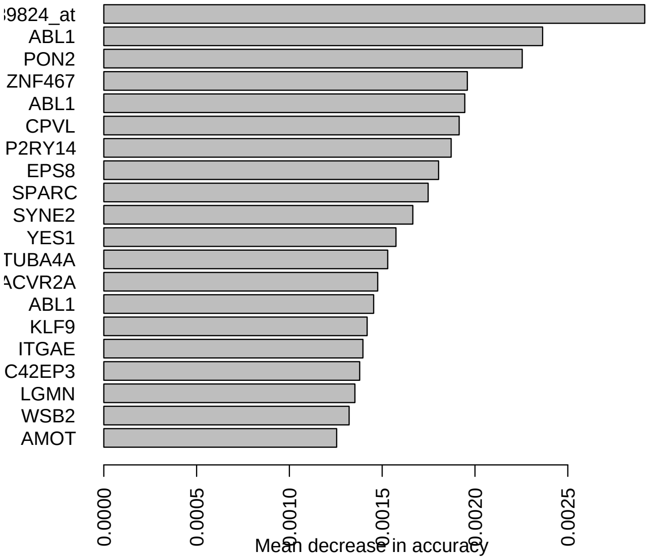


```
> varImpPlot(rf2, n.var = 15)
```



```
> 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")
```

```
> par(las = 2)
> plot(getVarImp(vcrf1), resolveenv = hgu95av2SYMBOL)
```



### *1.4.2 More exercises*

Again a number of interesting exercises present themselves.

**Exercise 15**

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

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

The version number of R and the packages and their versions that were used to generate this document are listed below

```
R version 2.5.0 RC (2007-04-22 r41275)
i386-apple-darwin8.9.1

locale:
C

attached base packages:
[1] "splines"   "tools"     "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

other attached packages:
    annotate randomForest          sma     hgu95av2 MLInterfaces          rda
    "1.14.1"     "4.5-18"     "0.5.15"     "1.16.0"     "1.10.2"        "1.0"
       rpart        class   genefilter     survival      bioDist RColorBrewer
    "3.1-35"     "7.2-34"     "1.15.6"       "2.31"      "1.8.0"      "0.2-3"
         ALL      Biobase       weaver    codetools       digest
     "1.4.2"     "1.14.0"      "1.2.0"      "0.1-1"      "0.3.0"
```