A machine learning tutorial: applications of the Bioconductor MLInterfaces package to gene expression data

VJ Carey

April 30, 2018

Contents

1 Overview

The term *machine learning* refers to a family of computational methods for analyzing multivariate datasets. Each data point has a vector of *features* in a shared *feature space*, and may have a *class label* from some fixed finite set.

Supervised learning refers to processes that help articulate rules that map *feature vectors* to *class labels*. The class labels are known and function as supervisory information to guide rule construction. *Unsupervised learning* refers to processes that discover structure in collections of feature vectors. Typically the structure consists of a grouping of objects into clusters.

This practical introduction to machine learning will begin with a survey of a low-dimensional dataset to fix concepts, and will then address problems coming from genomic data analysis, using RNA expression and chromatin state data.

Some basic points to consider at the start:

- Distinguish predictive modeling from inference on model parameters. Typical work in epidemiology focuses on estimation of relative risks, and random samples are not required. Typical work with machine learning tools targets estimation (and minimization) of the misclassification rate. Representative samples are required for this task.

- "Two cultures": model fitters vs. algorithmic predictors. If statistical models are correct, parameter estimation based on the mass of data can yield optimal discriminators (e.g., LDA). Algorithmic discriminators tend to prefer to identify boundary cases and downweight the mass of data (e.g., boosting, svm).

- Different learning tools have different capabilities. There is little *a priori* guidance on matching learning algorithms to aspects of problems. While it is convenient to sift through a variety of approaches, one must pay a price for the model search.

- Data and model/learner visualization are important, but visualization of higher dimensional data structures is hard. Dynamic graphics can help; look at ggobi and Rggobi for this.

- These notes provide very little mathematical background on the methods; see for example Ripley (*Pattern recognition and neural networks*, 1995), Duda, Hart, Stork (*Pattern classification*), Hastie, Tibshirani and Friedman (2003, *Elements of statistical learning*) for copious background.

2 Getting acquainted with machine learning via the crabs data

2.1 Attaching and checking the data

The following steps bring the crabs data into scope and illustrate aspects of its structure.

```
> library("MASS")
> data("crabs")
> dim(crabs)

[1] 200   8

> crabs[1:4,]

  sp sex index  FL  RW   CL   CW  BD
1  B   M     1 8.1 6.7 16.1 19.0 7.0
2  B   M     2 8.8 7.7 18.1 20.8 7.4
3  B   M     3 9.2 7.8 19.0 22.4 7.7
4  B   M     4 9.6 7.9 20.1 23.1 8.2

> table(crabs$sex)

  F   M
100 100

> library("lattice")
> print(bwplot(RW~sp|sex, data=crabs))
```

The plot is shown in Figure 1.

We will regard these data as providing five quantitative features (FL, RW, CL, CW, BD)[1] and a pair of class labels (sex, sp=species). We may regard this as a four class problem, or as two two class problems.

2.2 A simple classifier derived by human reasoning

Our first problem does not involve any computations. If you want to write R code to solve the problem, do so, but use prose first.

- *Question 1.* On the basis of the boxplots in Figure 1, comment on the prospects for predicting species on the basis of RW. State a rule for computing the predictions. Describe how to assess the performance of your rule.

[1] You may consult the manual page of `crabs` for an explanation of these abbreviations.

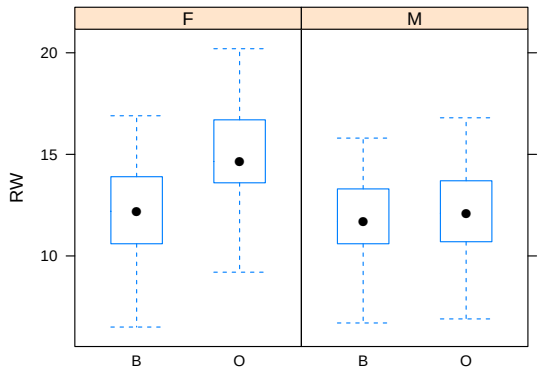


Figure 1: Boxplots of RW, the rear width in mm, stratified by species (”B” or ”O” for blue or orange) and sex (”F” and ”M”).

2.3 Prediction via logistic regression

A simple approach to prediction involves logistic regression.

```
> m1 = glm(sp~RW, data=crabs, family=binomial)
> summary(m1)

Call:
glm(formula = sp ~ RW, family = binomial, data = crabs)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.67807  -1.08840  -0.04168   1.07160   1.88030

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.44908    0.82210  -4.195 2.72e-05 ***
RW           0.27080    0.06349   4.265 2.00e-05 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 277.26  on 199  degrees of freedom
Residual deviance: 256.35  on 198  degrees of freedom
```

```
AIC: 260.35

Number of Fisher Scoring iterations: 4
```

- *Question 2.* Write down the statistical model corresponding to the R expression above. How can we derive a classifier from this model?

- *Question 3.* Perform the following computations. Discuss their interpretation. What are the estimated error rates of the two models? Is the second model, on the subset, better?

```
> plot(predict(m1,type="response"), crabs$sp)
> table(predict(m1,type="response")>.5, crabs$sp)
> m2 = update(m1, subset=(sex=="F"))
> table(predict(m2,type="response")>.5, crabs$sp[crabs$sex=="F"])
```

2.4 The cross-validation concept

Cross-validation is a technique that is widely used for reducing bias in the estimation of predictive accuracy. If no precautions are taken, bias can be caused by *overfitting* a classification algorithm to a particular dataset; the algorithm learns the classification "by heart", but performs poorly when asked to generalise it to new, unseen examples. Briefly, in cross-validation the dataset is deterministically partitioned into a series of training and test sets. The model is built for each training set and evaluated on the test set. The accuracy measures are averaged over this series of fits. Leave-one-out cross-validation consists of N fits, with N training sets of size N-1 and N test sets of size 1.

First let us use `MLearn` from the *MLInterfaces* package to fit a single logistic model. `MLearn` requires you to specify an index set for training. We use `c(1:30, 51:80)` to choose a training set of size 60, balanced between two species (because we know the ordering of records). This procedure also requires you to specify a probability threshold for classification. We use a typical default of 0.5. If the predicted probability of being "O" exceeds 0.5, we classify to "O", otherwise to "B".

```
> library(MLInterfaces)
> fcrabs = crabs[crabs$sex == "F", ]
> ml1 = MLearn( sp~RW, fcrabs, glmI.logistic(thresh=.5), c(1:30, 51:80),
+               family=binomial)
> ml1

MLInterfaces classification output container
The call was:
MLearn(formula = sp ~ RW, data = fcrabs, .method = glmI.logistic(thresh = 0.5),
```

```
    trainInd = c(1:30, 51:80), family = binomial)
Predicted outcome distribution for test set:
 O
40
Summary of scores on test set (use testScores() method for details):
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7553  0.8861  0.9803  0.9355  0.9917  0.9997
```

```
> confuMat(ml1)
```

```
  B  O NA
B 0 20  0
O 0 20  0
```

- *Question 4.* What does the report on `ml1` tell you about predictions with this model? Can you reconcile this with the results in model `m2`? [Hint – non-randomness of the selection of the training set is a problem.]
- *Question 5.* Modify the MLearn call to obtain a predictor that is more successful on the test set.

Now we will illustrate cross-validation. First, we scramble the order of records in the `ExpressionSet` so that sequentially formed groups are approximately random samples.

```
> set.seed(123)
> sfcrabs = fcrabs[ sample(nrow(fcrabs)),  ]
```

We invoke the `MLearn` method in two ways – first specifying a training index set, then specifying a five-fold cross-validation.

```
> sml1 = MLearn( sp~RW, sfcrabs, glmI.logistic(thresh=.5),
+   c(1:30, 51:80),
+   family=binomial)
> confuMat(sml1)
```

```
     predicted
given  B  O
    B 13 10
    O  3 14
```

```
> smx1 = MLearn( sp~RW, sfcrabs, glmI.logistic(thresh=.5),
+   xvalSpec("LOG", 5, function(data, clab, iternum) {
+     which(rep(1:5, each=20) == iternum) }),
+   family=binomial)
> confuMat(smx1)
```

```
     predicted
given  B  O
    B 34 16
    O 12 38
```

- *Question 6.* Define clearly the difference between models sml1 and smx1 and state the misclassification rate estimates associated with each model.

2.5 Exploratory multivariate analysis

2.5.1 Scatterplots

- *Question 7.* Interpret the following code, whose result is shown in Figure 2. Modify it to depict the pairwise configurations with different colors for crab genders.

```
> pairs(crabs[,-c(1:3)], col=ifelse(crabs$sp=="B", "blue", "orange"))
```

2.5.2 Principal components; biplot

Principal components analysis transforms the multivariate data X into a new coordinate system. If the original variables are X1, ..., Xp, then the variables in the new representation are denoted PC1, ..., PCp. These new variables have the properties that PC1 is the linear combination of the X1, ..., Xp having maximal variance, PC2 is the variance-maximizing linear combination of residuals of X after projecting into the hyperplane normal to PC1, and so on. If most of the variation in $X_{n\times p}$ can be captured in a low dimensional linear subspace of the space spanned by the columns of X, then the scatterplots of the first few principal components give a good representation of the structure in the data.

Formally, we can compute the PC using the singular value decomposition of X, in which $X = UDV^t$, where $U_{n\times p}$ and $V_{p\times p}$ are orthonormal, and D is a diagonal matrix of p nonnegative singular values. The principal components transformation is $XV = UD$, and if D is structured so that $D_{ii} \geq D_{jj}$ whenever $i > j$, then column i of XV is PCi. Note also that $D_{ii} = \sqrt{n-1}\,\mathrm{sd}(\mathrm{PCi})$.

```
> pc1 = prcomp( crabs[,-c(1:3)] )

> pairs(pc1$x, col=ifelse(crabs$sp=="B", "blue", "orange"))
```

The plot is shown in Figure 3.

The biplot, Figure 4, shows the data in PC space and also shows the relative contributions of the original variables in composing the transformation.

```
> biplot(pc1, choices=2:3, col=c("#80808080", "red"))
```

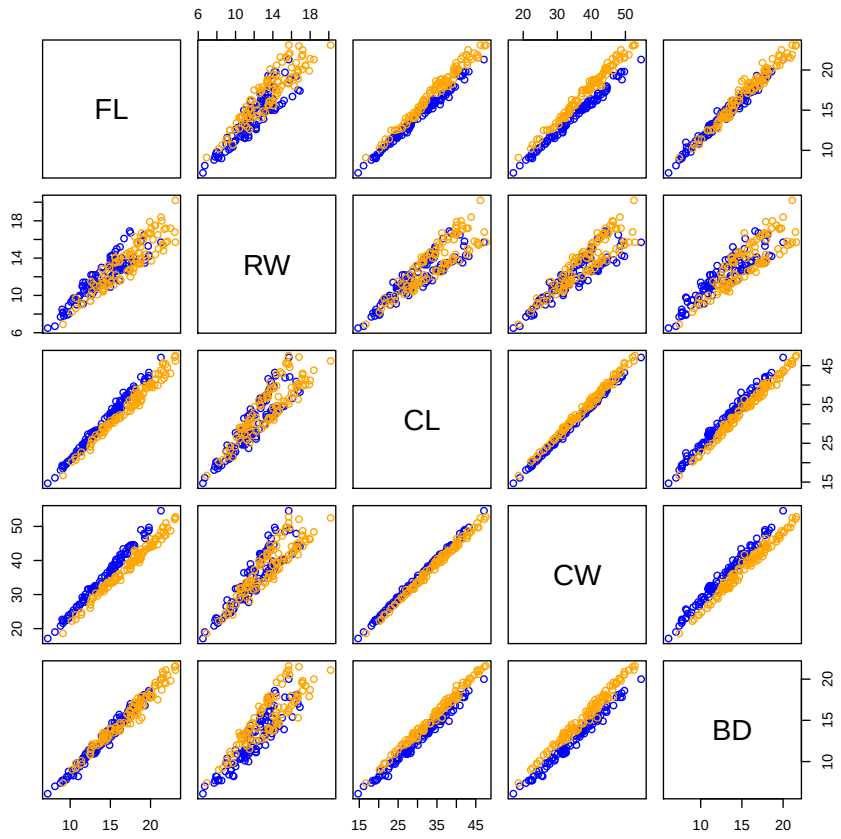


Figure 2: Pairs plot of the 5 quantitative features of the `crabs` data. Points are colored by species.

2.5.3 Clustering

A familiar technique for displaying multivariate data in high-throughput biology is called the heatmap. In this display, samples are clustered as columns, and features as rows. The clustering technique used by default is R `hclust`. This procedure builds a clustering tree for the data as follows. Distances are computed between each pair of feature vectors for all N observations. The two closest pair is joined and regarded as a new object, so there are $N - 1$ objects (clusters) at this point. This process is repeated until 1 cluster is formed; the clustering tree shows the process by which clusters are created via this agglomeration process.

The most crucial choice when applying this method is the initial choice of the distance

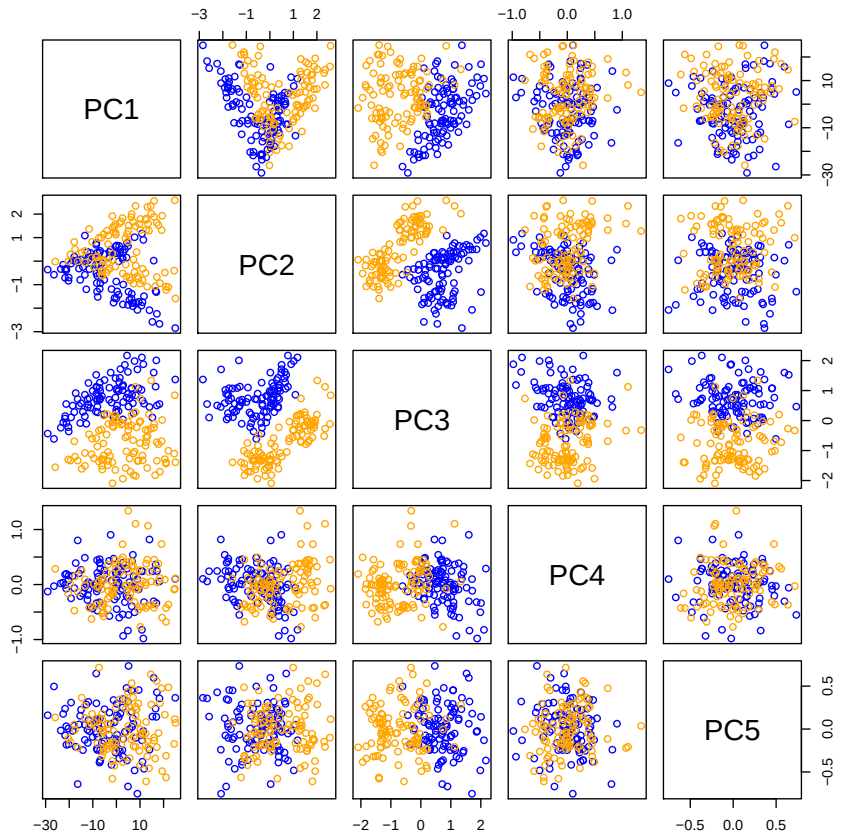


Figure 3: Pairs plot of the `crabs` data in principal component coordinates.

metric between the features.

Once clusters are being formed, there are several ways to measure distances between them, based on the initial between-feature distances. Single-linkage clustering takes the distance between two clusters to be the shortest distance between any two members of the different clusters; average linkage averages all the distances between members; complete-linkage uses hte maximum distance between any two members of the different clusters. Other methods are also available in `hclust`.

Figure 5 shows cluster trees for samples and features. The default color choice is not great, thus we specify own using the `col` argument. A tiled display at the top, defined via the argument `ColSideColors` shows the species codes for the samples. An important choice to be made when calling `heatmap` is the value of the argument `scale`, whose default

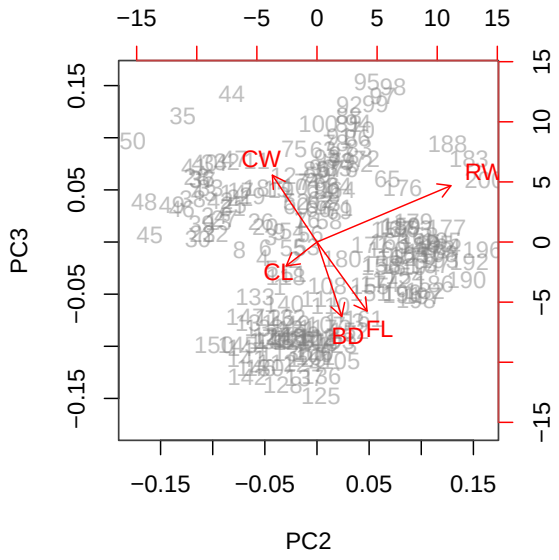


Figure 4: Biplot of the principal component analysis of the `crabs` data.

setting is to scale the rows, but not the columns.

```
> X = data.matrix(crabs[,-c(1:3)])
> heatmap(t(X),
+     ColSideColors=ifelse(crabs$sp=="O", "orange", "blue"),
+     col = colorRampPalette(c("blue", "white", "red"))(255))
```

Typically clustering is done in the absence of labels – it is an example of unsupervised machine learning. We can ask whether the clustering provided is a 'good' one using the measurement of a quantity called the *silhouette*. This is defined in R documentation as follows:

> For each observation i, the _silhouette width_ s(i) is defined as follows:
> Put a(i) = average dissimilarity between i and all other points of the cluster to which i belongs (if i is the _only_ observation in its cluster, s(i) := 0 without further calculations). For all _other_ clusters C, put d(i,C) = average dissimilarity of i to all observations of C. The smallest of these d(i,C) is b(i) := min_C d(i,C), and can be seen as the dissimilarity between i and its

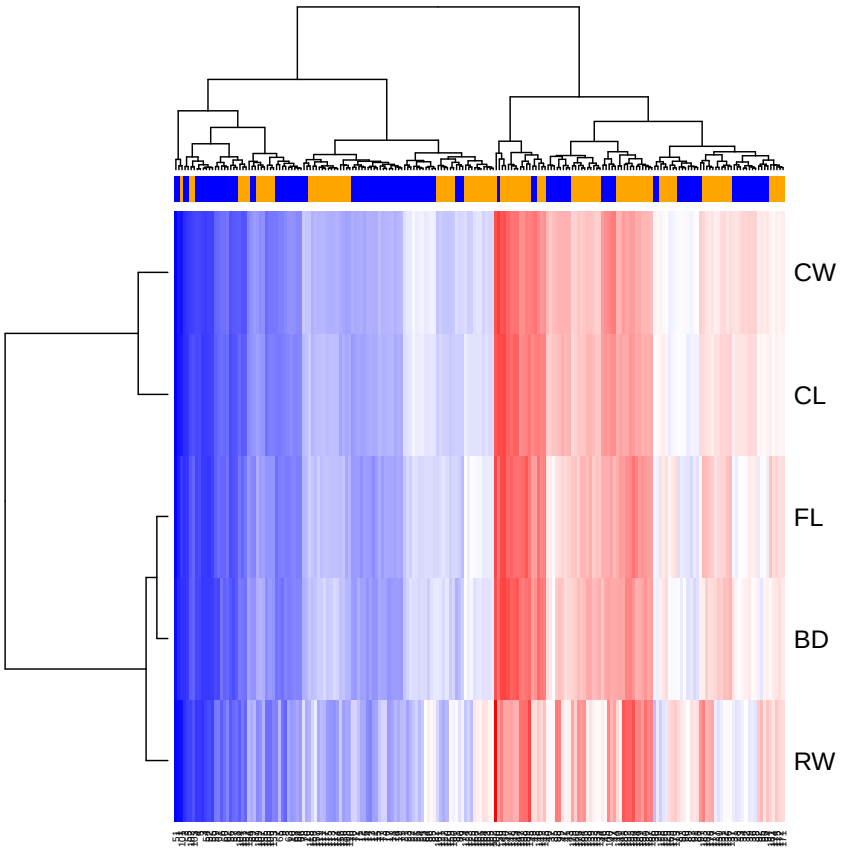


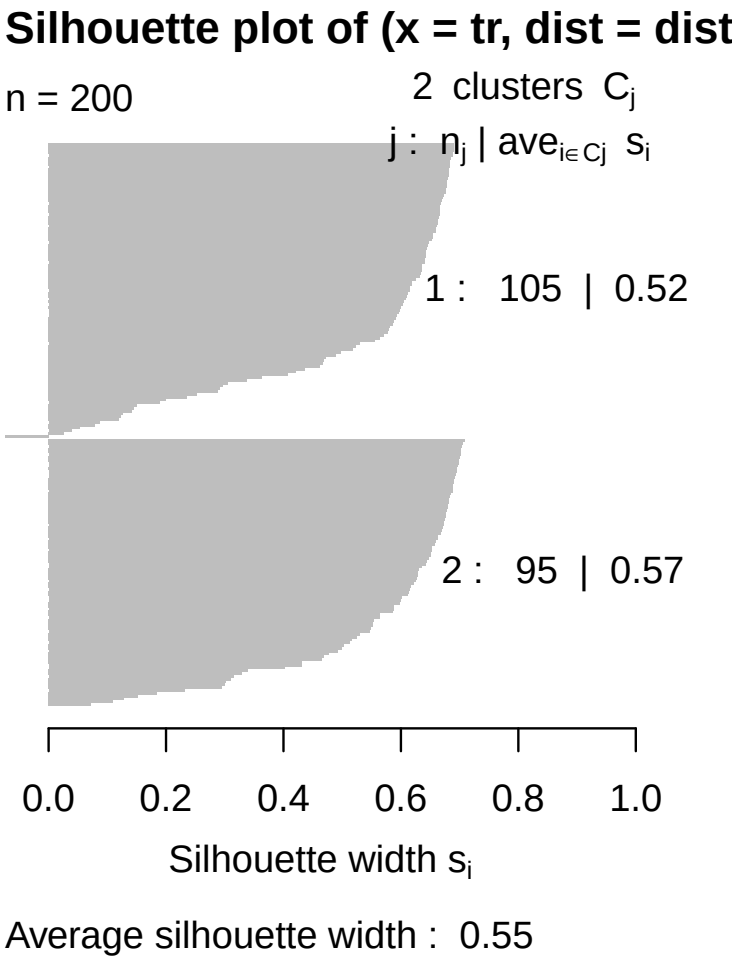
Figure 5: Heatmap plot of the `crabs` data, including dendrograms representing hierarchical clustering of the rows and columns.

```
"neighbor" cluster, i.e., the nearest one to which it does _not_
belong. Finally,

        s(i) := ( b(i) - a(i) ) / max( a(i), b(i) ).
```

We can compute the silhouette for any partition of a dataset, and can use the hierarchical clustering result to define a partition as follows:

```
> cl = hclust(dist(X))
> tr = cutree(cl,2)
> table(tr)
```

```
tr
  1   2
105  95

> library(cluster)
> sil = silhouette( tr, dist(X) )
> plot(sil)
```



- *Question 8.* In the preceding, we have used default `dist`, and default clustering algorithm for the heatmap. Investigate the impact of altering the choice of distance and clustering method on the clustering performance, both in relation to capacity to recover groups defined by species and in relation to the silhouette distribution.

- *Question 9.* The PCA shows that the data configuration in PC2 and PC3 is at least bifurcated. Apply hierarchical and K-means clustering to the two-dimensional data in this subspace, and compare results with respect to capturing the species × gender labels, and with respect to silhouette values. For example, load the exprs slot of crES [see just below for the definition of this structure] with the PCA reexpression of the features, call the result pcrES, and then:

```
> ff = kmeansB(pcrES[2:3,], k=4)
> table(ff@clustIndices, crES$spsex)
```

2.6 Supervised learning

In this section we will examine procedures for polychotomous prediction. We want to be able to use the measurements to predict both species and sex of the crab. Again we would like to use the MLInterfaces infrastructure, so an ExpressionSet container will be useful.

```
> feat2 = t(data.matrix(crabs[, -c(1:3)]))
> pd2 = new("AnnotatedDataFrame", crabs[,1:2])
> crES = new("ExpressionSet", exprs=feat2, phenoData=pd2)
> crES$spsex = paste(crES$sp, crES$sex, sep=":")
> table(crES$spsex)

B:F B:M O:F O:M
 50  50  50  50
```

We will permute the samples so that simple selections for training set indices are random samples.

```
> set.seed(1234)
> crES = crES[ , sample(1:200, size=200, replace=FALSE)]
```

2.6.1 RPART

A classic procedure is recursive partitioning.

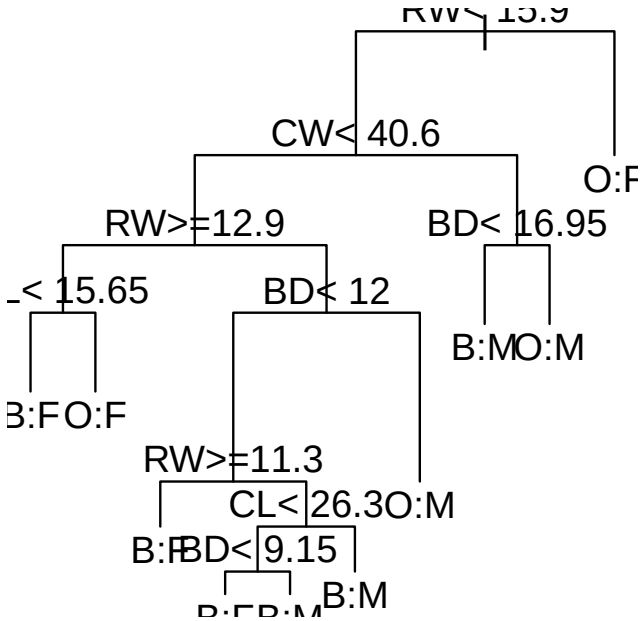
```
> library(rpart)
> tr1 = MLearn(spsex~., crES, rpartI, 1:140)
> tr1

MLInterfaces classification output container
The call was:
MLearn(formula = spsex ~ ., data = crES, .method = rpartI, trainInd = 1:140)
Predicted outcome distribution for test set:
```

```
B:F B:M O:F O:M
 17  11  16  16
Summary of scores on test set (use testScores() method for details):
      B:F       B:M       O:F       O:M
0.3013154 0.2050270 0.2892524 0.2044052
```

```
> confuMat(tr1)

     predicted
given B:F B:M O:F O:M
  B:F  10   4   3   0
  B:M   2   4   0   4
  O:F   3   1  11   3
  O:M   2   2   2   9
```

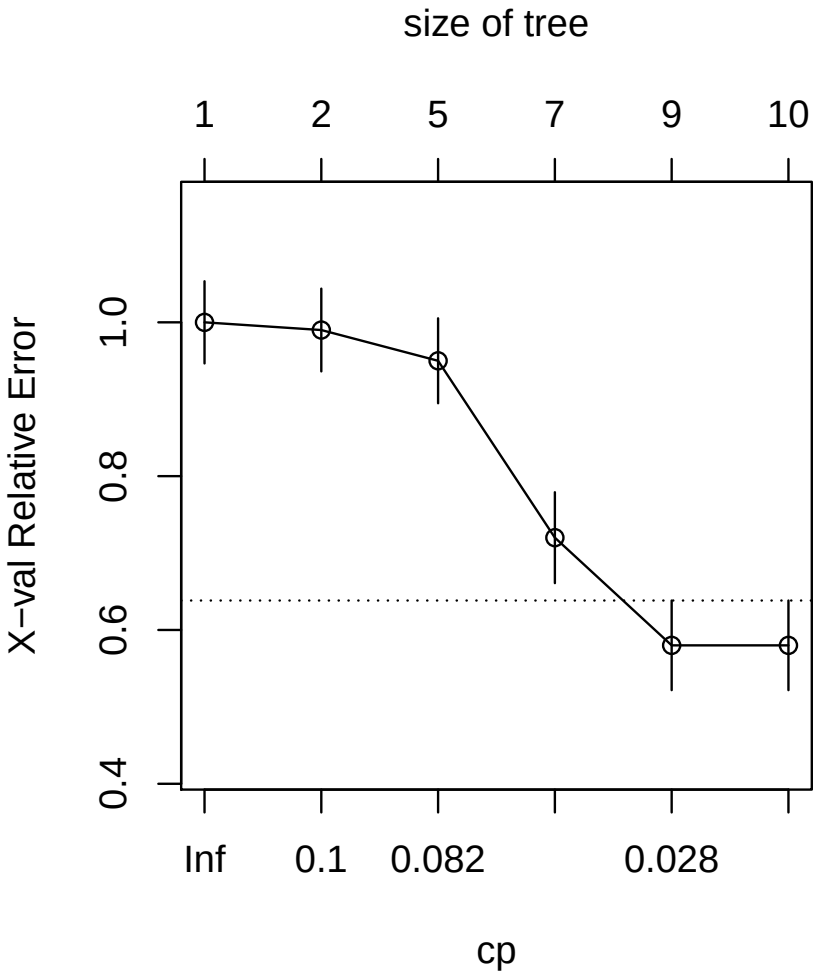
The actual tree is

```
> plot(RObject(tr1))
> text(RObject(tr1))
```



This procedure includes a diagnostic tool called the cost-complexity plot:

```
> plotcp(RObject(tr1))
```



2.6.2 Random forests

A generalization of recursive partitioning is obtained by creating a collection of trees by bootstrap-sampling cases and randomly sampling from features available for splitting at nodes.

```
> set.seed(124)
> library(randomForest)
> rf1 = MLearn(spsex~., crES, randomForestI, 1:140 )
> rf1
```

```
MLInterfaces classification output container
The call was:
```

```
MLearn(formula = spsex ~ ., data = crES, .method = randomForestI,
    trainInd = 1:140)
Predicted outcome distribution for test set:

B:F B:M O:F O:M
 17  12  18  13
Summary of scores on test set (use testScores() method for details):
      B:F       B:M       O:F       O:M
0.2922667 0.2255667 0.2602333 0.2219333
```

```
> cm = confuMat(rf1)
> cm

     predicted
given B:F B:M O:F O:M
  B:F  14   2   1   0
  B:M   1   9   0   0
  O:F   1   0  15   2
  O:M   1   1   2  11
```

The single split error rate is estimated at 18%.

- *Question 10.* What is the out-of-bag error rate for rf1? Obtain a cross-validated estimate of misclassification error using randomForest with an xvalSpec().

2.6.3 Linear discriminants

```
> ld1 = MLearn(spsex~., crES, ldaI, 1:140 )
> ld1

MLInterfaces classification output container
The call was:
MLearn(formula = spsex ~ ., data = crES, .method = ldaI, trainInd = 1:140)
Predicted outcome distribution for test set:

B:F B:M O:F O:M
 18   9  18  15

> confuMat(ld1)

     predicted
given B:F B:M O:F O:M
```

```
  B:F  17   0   0   0
  B:M   1   9   0   0
  O:F   0   0  18   0
  O:M   0   0   0  15
```

```
> xvld = MLearn( spsex~., crES, ldaI, xvalSpec("LOG", 5, balKfold.xvspec(5)))
> confuMat(xvld)
```

```
     predicted
given B:F B:M O:F O:M
  B:F  48   2   0   0
  B:M   6  44   0   0
  O:F   0   0  47   3
  O:M   0   0   0  50
```

- *Question 11.* Use the balKfold function to generate an index set for partitions that is balanced with respect to class distribution. Check the balance and repeat the cross validation.

2.6.4 Neural net

```
> nn1 = MLearn(spsex~., crES, nnetI, 1:140, size=3, decay=.1)
```

```
# weights:  34
initial  value 201.317011
iter  10 value 193.558660
iter  20 value 183.341858
iter  30 value 126.978224
iter  40 value 105.225773
iter  50 value 57.452804
iter  60 value 47.299132
iter  70 value 38.464400
iter  80 value 36.120882
iter  90 value 35.569938
iter 100 value 35.497034
final  value 35.497034
stopped after 100 iterations
```

```
> nn1
```

```
MLInterfaces classification output container
The call was:
```

```
MLearn(formula = spsex ~ ., data = crES, .method = nnetI, trainInd = 1:140,
    size = 3, decay = 0.1)
Predicted outcome distribution for test set:

B:F B:M O:F O:M
 18   9  18  15
Summary of scores on test set (use testScores() method for details):
      B:F       B:M       O:F       O:M
0.2808466 0.1724775 0.3048809 0.2417950

> RObject(nn1)

a 5-3-4 network with 34 weights
inputs: FL RW CL CW BD
output(s): spsex
options were - softmax modelling  decay=0.1

> confuMat(nn1)

     predicted
given B:F B:M O:F O:M
  B:F  17   0   0   0
  B:M   1   9   0   0
  O:F   0   0  18   0
  O:M   0   0   0  15

> xvnnBAD = MLearn( spsex~., crES, nnetI,
+     xvalSpec("LOG", 5, function(data, clab, iternum) {
+       which( rep(1:5,each=40) == iternum ) }),
+       size=3, decay=.1 )
> xvnnGOOD = MLearn( spsex~., crES, nnetI,
+     xvalSpec("LOG", 5, balKfold.xvspec(5) ),
+       size=3, decay=.1 )

> confuMat(xvnnBAD)

     predicted
given B:F B:M O:F O:M
  B:F  48   2   0   0
  B:M   6  44   0   0
  O:F   0   0  48   2
  O:M   0   0   4  46
```

```
> confuMat(xvnnGOOD)

     predicted
given B:F B:M O:F O:M
  B:F  49   1   0   0
  B:M   5  45   0   0
  O:F   0   0  49   1
  O:M   0   0   1  49
```

2.6.5 SVM

```
> sv1 = MLearn(spsex~., crES, svmI, 1:140)

Called from: .method@converter(ans, data, trainInd)
debug: teData = data[-trainInd, ]
debug: trData = data[trainInd, ]
debug: tepr = predict(obj, teData, decision.values = TRUE, probability = TRUE)
debug: trpr = predict(obj, trData, decision.values = TRUE, probability = TRUE)
debug: new("classifierOutput", testPredictions = factor(tepr[seq_len(length(tepr))]),
    trainPredictions = factor(trpr[seq_len(length(trpr))]), testScores = attr(tepr,
        "probabilities"), trainScores = attr(trpr, "probabilities"),
    RObject = obj)

> sv1

MLInterfaces classification output container
The call was:
MLearn(formula = spsex ~ ., data = crES, .method = svmI, trainInd = 1:140)
Predicted outcome distribution for test set:

B:F B:M O:F O:M
 23   7  16  14
Summary of scores on test set (use testScores() method for details):
      B:M       O:M       O:F       B:F
0.1817453 0.2258914 0.2922744 0.3000889

> RObject(sv1)

Call:
svm(formula = formula, data = data, probability = probability)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.2

Number of Support Vectors:  131
```

```
> confuMat(sv1)
```

```
     predicted
given B:F B:M O:F O:M
  B:F  16   0   1   0
  B:M   3   7   0   0
  O:F   2   0  15   1
  O:M   2   0   0  13
```

```
> xvsv = MLearn( spsex~., crES, svmI, xvalSpec("LOG", 5,
+     balKfold.xvspec(5)))
```

```
> confuMat(xvsv)
```

```
     predicted
given B:F B:M O:F O:M
  B:F  44   1   5   0
  B:M   9  36   1   4
  O:F   8   0  38   4
  O:M   3   3   0  44
```

3 Learning with expression arrays

Here we will concentrate on ALL: acute lymphocytic leukemia, B-cell type.

3.1 Phenotype reduction

We will identify expression patterns that discriminate individuals with BCR/ABL fusion in B-cell leukemia.

```
> library("ALL")
> data("ALL")
> bALL = ALL[, substr(ALL$BT,1,1) == "B"]
> fus = bALL[, bALL$mol.biol %in% c("BCR/ABL", "NEG")]
> fus$mol.biol = factor(fus$mol.biol)
> fus
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 12625 features, 79 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 01005 01010 ... 84004 (79 total)
  varLabels: cod diagnosis ... date last seen (21 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
  pubMedIds: 14684422 16243790
Annotation: hgu95av2
```

3.2 Nonspecific filtering

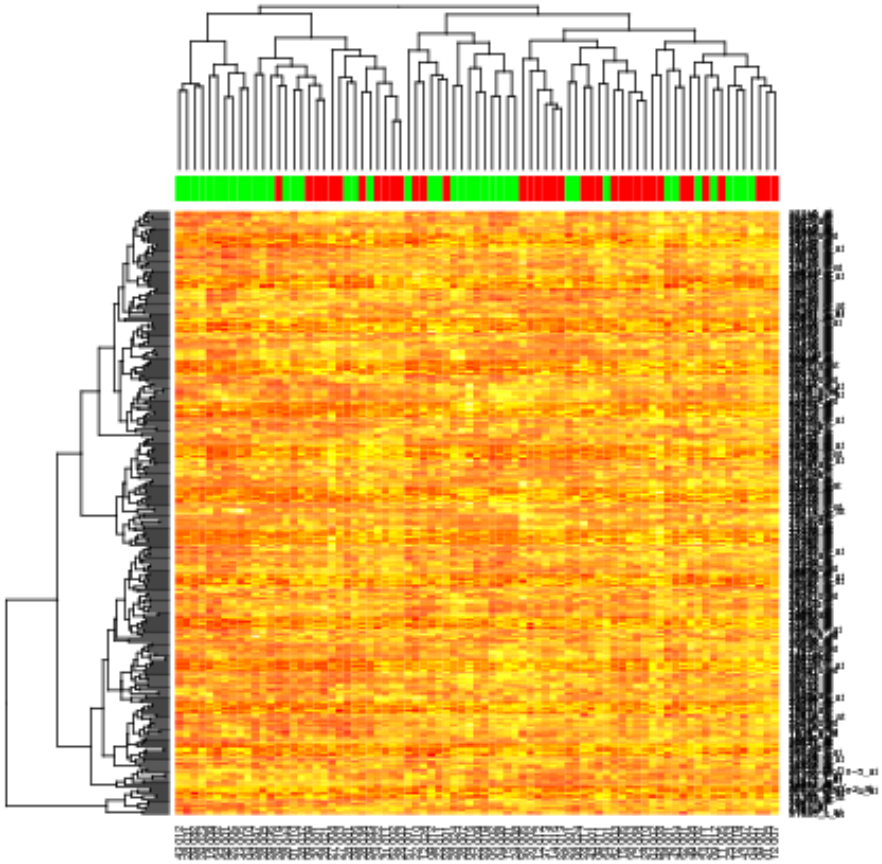
We can nonspecifically filter to 300 genes (to save computing time) with largest measures of robust variation across all samples:

```
> mads = apply(exprs(fus),1,mad)
> fusk = fus[ mads > sort(mads,decr=TRUE)[300], ]
> fcol = ifelse(fusk$mol.biol=="NEG", "green", "red")
```

3.3 Exploratory work

For exploratory data analysis, a heatmap is customary.

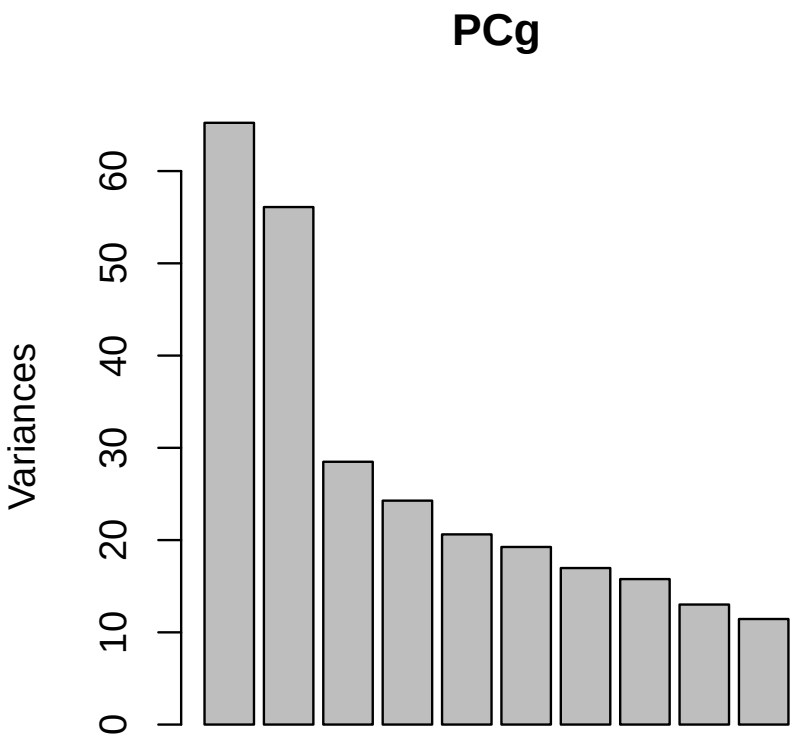
```
> heatmap(exprs(fusk), ColSideColors=fcol)
```



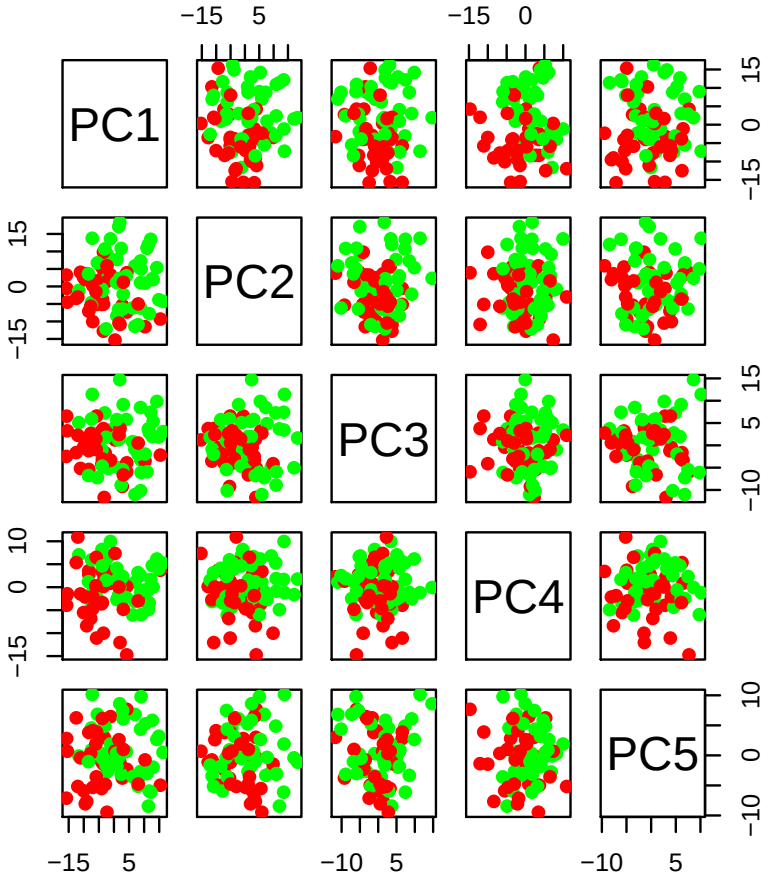
Principal components and a biplot may be more revealing. How many principal components are likely to be important?

```
> PCg = prcomp(t(exprs(fusk)))
```

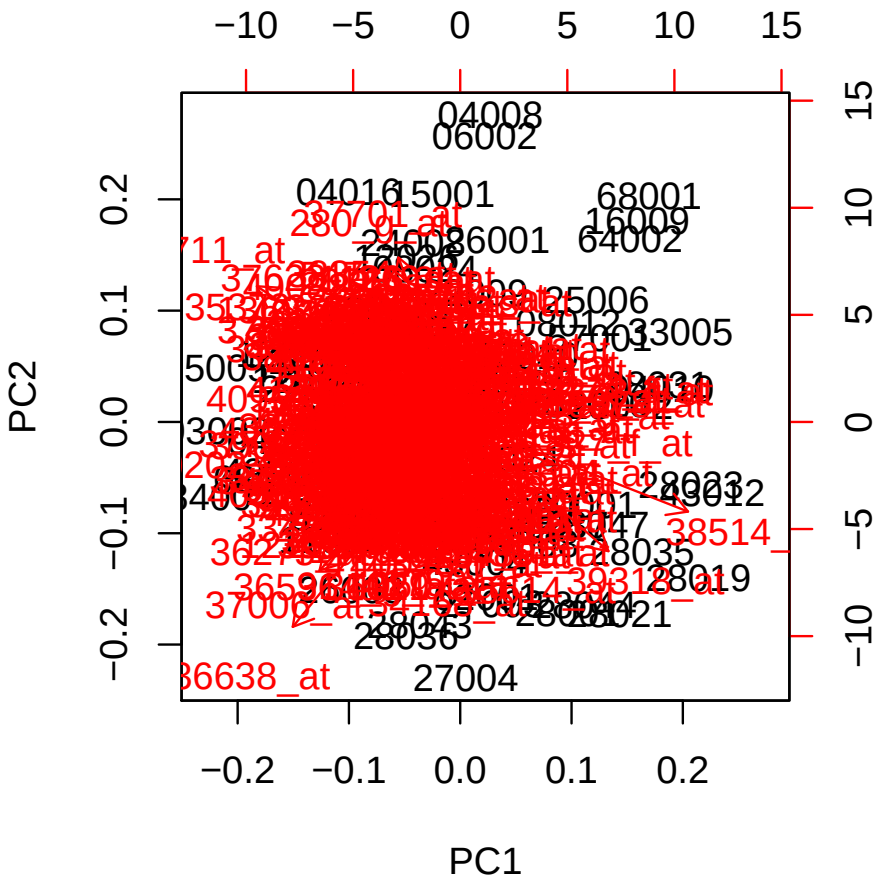
```
> plot(PCg)
```




```
> pairs(PCg$x[,1:5],col=fcol,pch=19)
```


PC1
PC2
PC3
PC4
PC5
−15 5
−15 0
−15 0 15
−15 0 15
−10 5 15
−15 0 10
−10 0 10
−15 5
−10 5
−10 5

```
> biplot(PCg)
```



- *Question 12.* Modify the biplot so that instead of plotting sample ID, the symbol "O" is plotted for a NEG sample and "+" is plotted for a BCR/ABL sample.

- *Question 13.* Consider the following code

```
chkT = function (x, eset=fusk) {
t.test(exprs(eset)[x, eset$mol.b == "NEG"], exprs(eset)[x, eset$mol.b ==
    "BCR/ABL"]) }
```

  Use it in conjunction with the biplot to interpret expression patterns of genes that appear to be important in defining the PCs.

3.4 Classifier construction

3.4.1 Demonstrations

Diagonal LDA has a good reputation. Let's try it first, followed by neural net and random forests. We will not attend to tuning the latter two, defaults or guesses for key parameters are used.

```
> dld1 = MLearn( mol.biol~., fusk, dldaI, 1:40 )

> dld1

MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fusk, .method = dldaI,
    trainInd = 1:40)
Predicted outcome distribution for test set:

BCR/ABL     NEG
     27      12

> confuMat(dld1)

         predicted
given     BCR/ABL NEG
  BCR/ABL      15   1
  NEG          12  11

> nnALL = MLearn( mol.biol~., fusk, nnetI, 1:40, size=5, decay=.01,
+   MaxNWts=2000 )

# weights:  1506
initial  value 32.308250
iter  10 value 25.621164
iter  20 value 14.891037
iter  30 value 9.342362
iter  40 value 6.206645
iter  50 value 4.449391
iter  60 value 3.476369
iter  70 value 2.310562
iter  80 value 1.307912
iter  90 value 1.205184
iter 100 value 0.880492
final  value 0.880492
stopped after 100 iterations
```

```
> confuMat(nnALL)

         predicted
given     BCR/ABL NEG
  BCR/ABL      14   2
  NEG           8  15

> rfALL = MLearn( mol.biol~., fusk, randomForestI, 1:40 )

> rfALL

MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fusk, .method = randomForestI,
    trainInd = 1:40)
Predicted outcome distribution for test set:

BCR/ABL     NEG
     24      15
Summary of scores on test set (use testScores() method for details):
BCR/ABL     NEG
  0.522   0.478

> confuMat(rfALL)

         predicted
given     BCR/ABL NEG
  BCR/ABL      15   1
  NEG           9  14
```

None of these are extremely impressive, but the problem may just be very hard. An interesting proposal is RDA, regularized discriminant analysis (package rda, Guo, Hastie, Tibshirani 2007 Biostatistics). This algorithm recognizes the fact that covariance matrix estimation in high dimensional data is very inaccurate, so the estimator is shrunk towards the identity. We have a rudimentary interface, in which the key parameters are chosen by native cross-validation (in rda.cv) and then applied once to get the final object.

```
> set.seed(1234)
> rdaALL = MLearn( mol.biol~., fusk, rdacvI, 1:40 )

> rdaALL
```

```
MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fusk, .method = rdacvI,
    trainInd = 1:40)
Predicted outcome distribution for test set:

BCR/ABL     NEG
     21      18
```

```
> confuMat(rdaALL)

         predicted
given     BCR/ABL NEG
  BCR/ABL      14   2
  NEG           7  16
```

A by-product of the algorithm is a set of retained genes that were found to play a role in discrimination. This can be established as follows:

```
> library(hgu95av2.db)
> psid = RObject(rdaALL)$keptFeatures
> psid = gsub("^X", "", psid) # make.names is run inopportunely
> mget(psid, hgu95av2GENENAME, ifnotfound=NA)[1:5]

$`106_at`
[1] "runt related transcription factor 3"

$`1077_at`
[1] "recombination activating 1"

$`1113_at`
[1] "bone morphogenetic protein 2"

$`1178_at`
[1] "dihydrofolate reductase"

$`1211_s_at`
[1] "CASP2 and RIPK1 domain containing adaptor with death domain"
```

- *Question 14.* How can we assess the relative impacts of regularization (expanding the covariance model beyond that of DLDA, which was shown above to do poorly, but without relying on the full covariance) and implicit feature selection conducted in RDA?

3.4.2 Gene set appraisal

- *Question 15.* We can assess the predictive capacity of a set of genes by restricting the ExpressionSet to that set and using the best classifier appropriate to the problem. We can also assess the incremental effect of combining gene sets, relative to using them separately.

  One collection of gene sets that is straightforward to use and interpret is provided by the keggorthology package (see also GSEABase). Here's how we can define the ExpressionSets for genes annotated by KEGG to Environmental (Genetic) Information Processing:

```
> library(keggorthology)
> data(KOgraph)
> adj(KOgraph, nodes(KOgraph)[1])

$KO.Feb10root
[1] "Metabolism"
[2] "Genetic Information Processing"
[3] "Environmental Information Processing"
[4] "Cellular Processes"
[5] "Organismal Systems"
[6] "Human Diseases"

> EIP = getKOprobes("Environmental Information Processing")
> GIP = getKOprobes("Genetic Information Processing")
> length(intersect(EIP, GIP))

[1] 40

> EIPi = setdiff(EIP, GIP)
> GIP = setdiff(GIP, EIP)
> EIP = EIPi
> Efusk = fusk[ featureNames(fusk) %in% EIP, ]
> Gfusk = fusk[ featureNames(fusk) %in% GIP, ]
```

  Obtain and assess the predictive capacity of the genes annotated to "Cell Growth and Death".

- *Question 16.* How many of the genes identified by RDA as important for discriminating fusion are annotated to Genetic Information Processing in the KEGG orthology?

4 Embedding features selection in cross-validation

We provide helper functions to conduct several kinds of feature selection in cross-validation, see `help(fs.absT)`. Here we pick the top 30 features (ranked by absolute t statistic) for each cross-validation partition.

```
> dldFS = MLearn( mol.biol~., fusk, dldaI, xvalSpec("LOG", 5,
+    balKfold.xvspec(5), fs.absT(30) ))
> dldFS

MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fusk, .method = dldaI,
    trainInd = xvalSpec("LOG", 5, balKfold.xvspec(5), fs.absT(30)))
Predicted outcome distribution for test set:

BCR/ABL     NEG
     42      37
history of feature selection in cross-validation available; use fsHistory()

> confuMat(dld1)

         predicted
given     BCR/ABL NEG
  BCR/ABL      15   1
  NEG          12  11

> confuMat(dldFS)

         predicted
given     BCR/ABL NEG
  BCR/ABL      34   3
  NEG           8  34
```

5 Session information

```
> sessionInfo()

R version 3.5.0 (2018-04-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.4 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] keggorthology_2.32.0 graph_1.58.0         hgu95av2.db_3.2.3
 [4] org.Hs.eg.db_3.6.0   ALL_1.21.0           randomForest_4.6-14
 [7] rpart_4.1-13         lattice_0.20-35      MASS_7.3-50
[10] golubEsets_1.21.0    MLInterfaces_1.60.0  cluster_2.0.7-1
[13] annotate_1.58.0      XML_3.98-1.11        AnnotationDbi_1.42.0
[16] IRanges_2.14.0       S4Vectors_0.18.0     Biobase_2.40.0
[19] BiocGenerics_0.26.0

loaded via a namespace (and not attached):
 [1] sfsmisc_1.1-2      bit64_0.9-7        splines_3.5.0      gtools_3.5.0
 [5] threejs_0.3.1      shiny_1.0.5        assertthat_0.2.0   blob_1.1.1
 [9] robustbase_0.93-0  pillar_1.2.2       RSQLite_2.1.0      glue_1.2.0
[13] digest_0.6.15      RColorBrewer_1.1-2 promises_1.0.1     gbm_2.1.3
[17] htmltools_0.3.6    httpuv_1.4.1       Matrix_1.2-14      rda_1.0.2-2
[21] mlbench_2.1-1      pkgconfig_2.0.1    genefilter_1.62.0  xtable_1.8-2
[25] mvtnorm_1.0-7      gdata_2.18.0       later_0.7.1        tibble_1.4.2
```

```
[29] ggvis_0.4.3         nnet_7.3-12         ada_2.0-5           survival_2.42-3
[33] magrittr_1.5        mime_0.5            mclust_5.4          memoise_1.1.0
[37] hwriter_1.3.2       class_7.3-14        tools_3.5.0         trimcluster_0.1-2
[41] kernlab_0.9-26      fpc_2.1-11          bindrcpp_0.2.2      e1071_1.6-8
[45] pls_2.6-0           compiler_3.5.0      rlang_0.2.0         grid_3.5.0
[49] RCurl_1.95-4.10     htmlwidgets_1.2     crosstalk_1.0.0     igraph_1.2.1
[53] bitops_1.0-6        base64enc_0.1-3     flexmix_2.3-14      DBI_0.8
[57] R6_2.2.2            prabclus_2.2-6      dplyr_0.7.4         bit_1.1-12
[61] bindr_0.1.1         modeltools_0.2-21   Rcpp_0.12.16        DEoptimR_1.0-8
[65] diptest_0.75-7
```