Machine learning in Bioconductor: Practical exercises

©2005, VJ Carey <stvjc@channing.harvard.edu>

July 30, 2006

Contents

1 Introduction

Excerpt from the Monograph chapter: Machine learning refers to computational and statistical inference processes employed to create, on the basis of observational data, reusable algorithms for prediction. The term *machine* is introduced to reflect the view that the creation of the predictive algorithm should occur with minimal human intervention, and the predictions for future observations should occur with no human intervention. (End excerpt.)

Let $\mathbb{X}$ denote a p-dimensional feature space, and let T_n denote a set of n examples available for inspection. Two basic species of machine learning have evolved.

- In unsupervised learning, T_n is used to identify intrinsic patterns or configurations of features. The predictive use of this procedure can take various forms. In one simple approach, future observations are grouped together on the basis of the patterns discovered in T_n.

  Cluster analysis is the primary exemplar of unsupervised machine learning. Many different clustering algorithms are available in R.

- In supervised learning, each example consists of an element of $\mathbb{X}$ associated with a class label $c \in \mathbb{C} = \{1, \ldots, C, \mathbb{D}, \mathbb{O}\}$, where C is the number of proper classes of objects about which learning is to be conducted, $\mathbb{D}$ is a special class label denoting 'doubt' and $\mathbb{O}$ is a special class label denoting 'outlier'. The problem of machine learning is to use T_n to identify or build a function from $\mathbb{X}$ to $\mathbb{C}$ that will correctly classify future objects on the basis of their features.

Many different approaches to supervised machine learning have evolved in the twentieth century. See the monograph chapter for relevant background references.

2 Prologue on Bioconductor and the R computing environment

Software implementing machine learning methods is widely available. The Bioconductor project focuses on the R environment for a variety of reasons – chief among them is the recognition that many statistical researchers use R to develop, test, and distribute new inferential tools.

It is important to appreciate the structure of the model fitting framework in R.

- **function/object paradigm:** Activities in R primarily take the form of function evaluation. In R, generically, `y=f(x,...)` creates an object called `y` by evaluating `f` on input `x` and other inputs. `f` must be an R function, but `x` can be anything, and `y` is completely defined by the function `f` as evaluated on its arguments. There is no function typing, so the type or class of `y` may be vary across invocations of `f`, depending on the classes and types of arguments and on the specific details of `f`. However, all entities definable in R are in certain specific respects self-describing, so it will generally be possible to interrogate `y` (and `f` can interrogate its arguments) to learn about its structure.

  The statistical modeling paradigm in R takes the form of evaluating a function (a model fitting function) on some objects to create some new object. The new object can be operated upon by other functions to obtain reports, predictions, figures of merit, graphics, etc. Certain very common procedures (coded as R generics) such as `summary`, `plot`, `residuals`, `predict` are defined by most modeling procedures to work as R methods on outputs of specific modeling functions.

- **data.frame, formula:** Given a rectangular data structure in the `data.frame` format, a `formula` specifies (typically up to the value of an (unknown) parameter vector) relationships among variables defining a model. The formula `y ~ x1+x2` specifies that `y` is the dependent variable and `x1` and `x2` are predictor variables. In the classical frameworks of ANOVA, linear regression, and generalized linear models, the Wilkinson Rogers formalism can be used to specify interactions concisely. The same syntax can be used to specify elements of neural network models, support vector machine models, mixed effects models, etc., although additional structures (e.g., parameter settings or random effects models) are needed to complete the specification.

- **predict:** A modeling procedure returns an R object with a particular structure. It may be a list, it may be a formal object in some object-oriented paradigm such as S4. In most cases, a `predict` method is available that will use the fitted model to produce predictions on new assignments to predictor variables. This has the form of a "black box" – the user only needs to supply the `newdata` data frame instance, and predictions will be made. Note that some modeling tools most naturally compute predictions in the form of posterior probabilities of events, even if the dependent variable has a categorical form. Typically a `type` parameter can be used to distinguish different prediction formats.

- **figures of merit:** Most modeling methods optimize some objective function of the data (e.g., minimize the sum of squared residuals), but estimating predictive accuracy based on a single optimization on the data at hand is unsound. *Cross-validation* refers to a family of procedures of iteratively refitting the model based on partitions of the data, fitting to one part and predicting on the "left out" part.

Some machine learning procedures in R have cross-validation built in (e.g., rpart, svm in e1071), and the *MLInterfaces* package provides an xval wrapper.

- **software brokering:** A basic expense encountered in using distributed methods for machine learning is the requirement to "recode" data in a specific format to allow execution of the code. This occurs even in R, where various packages and functions require various types of input. An initiative called PMML (predictive modeling markup language) pursues a generic markup for specification of models and exposure of data for fitting. The MLInterfaces package attempts to help users in bioinformatics in two ways:
  - uniform input idiom based on ExpressionSet class
  - uniform output idiom based on MLOutput class

  These features will be illustrated in the lab

Summary. One can step from a microarray experiment to a machine learning tool, extracting a matrix of numbers from the experimental output and throwing the matrix to a software tool that computes a report. This is not the way Bioconductor is supposed to work, even though some packages allow this pattern. Instead,

- the experimental data should be bound with metadata about the experiment and about the samples, in the form of an ExpressionSet
- the relationships of interest should be expressible in a formula
- the modeling software tool should be able to
  - use the ExpressionSet to obtain the data to be modeled,
  - produce an object that facilitates convenient comparison of fits across different learning procedures
  - be used in the context of different approaches to cross validation, including embedded feature selection

The MLInterfaces system endeavors to support these enhancements to the modeling patterns available in R.

3 A non-genomic refresher with the crabs data

3.1 Introduction

In this section we review multivariate displays and some aspects of unsupervised learning, using a demonstration multivariate dataset of modest dimensions.

The *MASS* package includes a dataset derived from

```
Campbell, N.A. and Mahon, R.J. (1974) A multivariate study of
variation in two species of rock crab of genus _Leptograpsus._
_Australian Journal of  Zoology_ *22*, 417-425.
```

Ensure that the *MASS* package is installed with your installation of R. If `library(MASS)` fails, install the package.

To access the crabs data, use

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

Then `help(crabs)` will provide some descriptive information. You can see the first few records using

```
> crabs[1:3, ]
```

```
  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
```

The variables are

- `sp`, a species code, O for orange, B for blue
- `sex`, a gender code
- five anatomical measurements recorded in millimeters: FL (frontal lobe width), RW (rear width), CL (carapace length), CW (carapace width), BD (body depth).

Our basic objectives are to describe the relationships among the anatomic features, and to understand how anatomical measurements associate with species and gender.

3.2 Workspace management

Use the operating system to start a folder called "crabs1" on your computer. For windows users it might be `C:/crabs1`; for others it might be `$HOME/crabs1`. Assign the absolute path name of your chosen folder to the variable `crabfolder` and then use `setwd` to relocate R's operating directory there. For example:

```
> crabfolder <- "C:/crabs1"
> setwd(crabfolder)
```

You should save your image and save your history in this directory before shutting down.

```
> boxplot(crabs[, -c(1, 2, 3)])
```

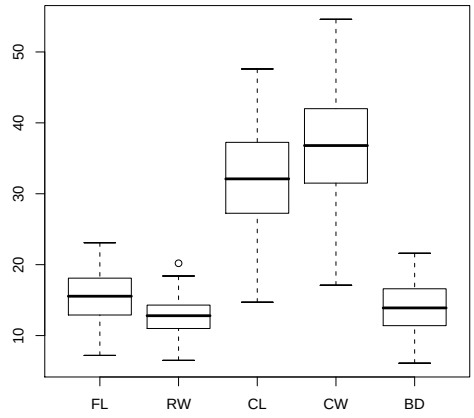


Figure 1: Crab measure univariate distributional summaries.

3.3 Simple graphics

First we would like to assess location and spread of the anatomical measures over the whole sample. R's boxplot command takes care of this very simply. We omit the first three columns for now; see Figure 1.

To assess the correlations among the measures, we use the pairs function. See Figure 2.

Marginal normality of frontal lobe measures can be assessed as in Figure 3.

3.4 Stratified graphics

The *lattice* package is useful for displaying multivariate data. With the data in its standard form, we can assess the effect of gender on carapace width within species; see Figure 4.

```
> pairs(crabs[, -c(1, 2, 3)], col = ifelse(crabs$sex == "M", "blue",
+     "gold"), pch = 15)
```

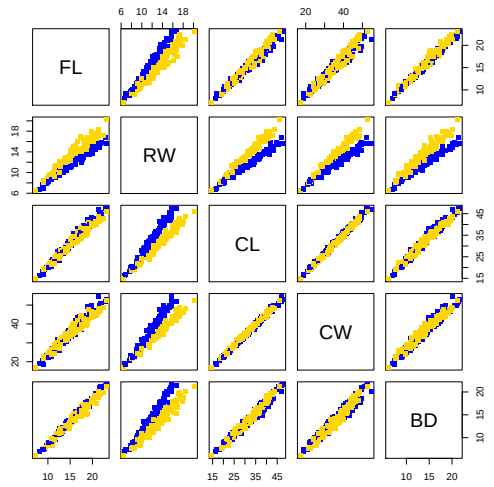


Figure 2: Crab measure bivariate scatterplots.

```
> qqnorm(crabs[, "FL"])
```

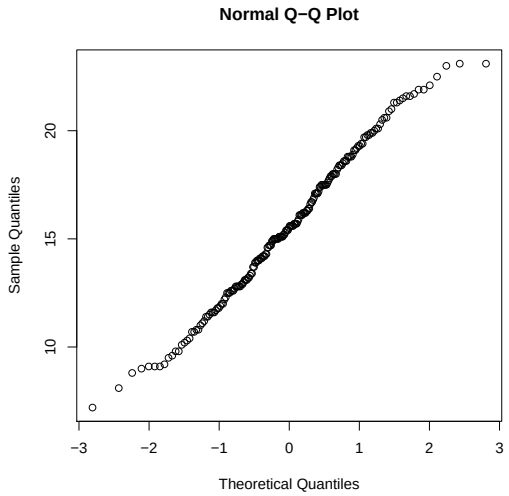


Figure 3: Crab frontal lobe gaussianity assessment.

```
> library(lattice)
> print(bwplot(sex ~ CW | sp, data = crabs, layout = c(1, 2)))
```

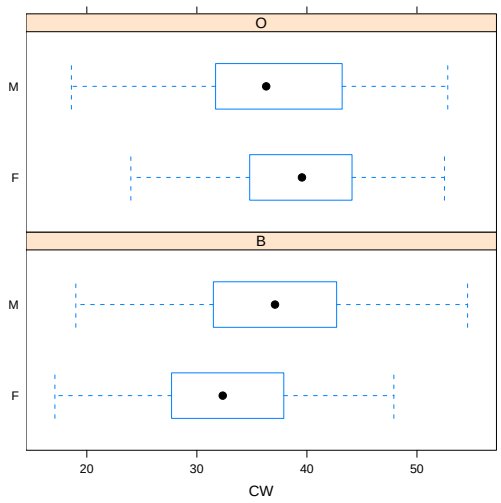


Figure 4: Stratified carapace width distributional summaries.

3.5 Altering data structure

Data structure can affect the ease with which stratified graphics are specified. The structure in use has one record per crab. It is possible to restructure the data with one record per feature of each crab.

One approach to this is as follows; see figure 5.

```
> cfeat <- data.matrix(crabs[, -c(1, 2, 3)])
> fvec <- as.numeric(t(cfeat))
> fnames <- factor(rep(names(crabs[, -c(1, 2, 3)]), 200))
> newsp <- rep(crabs$sp, each = 5)
> newcfeat <- data.frame(feat = fnames, val = fvec, sp = newsp)
> newcfeat[1:10, ]

   feat  val sp
1    FL  8.1  B
2    RW  6.7  B
3    CL 16.1  B
4    CW 19.0  B
5    BD  7.0  B
6    FL  8.8  B
7    RW  7.7  B
8    CL 18.1  B
9    CW 20.8  B
10   BD  7.4  B

> cfeat[1:2, ]

   FL  RW   CL   CW  BD
1 8.1 6.7 16.1 19.0 7.0
2 8.8 7.7 18.1 20.8 7.4
```

```
> print(bwplot(sp ~ val | feat, data = newcfeat, layout = c(1,
+     5)))
```

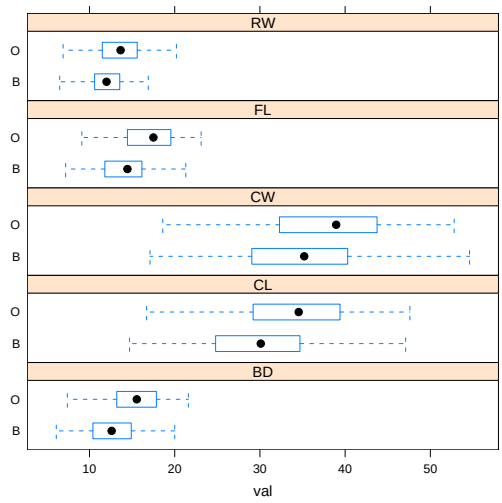


Figure 5: Stratified distributional summaries for all measures.

3.6 Distances

We will confine attention to 40 observations, using

```
> kpind <- c(1:10, 51:60, 101:110, 151:160)
> kpsp <- crabs$sp[kpind]
> kpsex <- crabs$sex[kpind]
> kfeat <- cfeat[kpind, ]
```

If x_1 and x_2 are two points in p-dimensional Euclidean space, the distance between them, $d_E(x_1, x_2)$, is

$$d_E(x_1, x_2) = \sqrt{\sum_{i=1}^{p} (x_{1i} - x_{2i})^2}$$

We can look at the distances among the features:

```
> dist(t(kfeat))
```

```
       FL     RW     CL     CW
RW  11.67
CL  73.63  84.57
CW  93.53 104.39  20.09
BD   9.12   5.17  82.45 102.37
```

A different definition of distance is 'Canberra':

$$d_C(x_1, x_2) = \sum_i \frac{|x_{1i} - x_{2i}|}{|x_{1i} + x_{2i}|}$$

We can impose the use of this measure through the following:

```
> dist(t(kfeat), method = "canberra")
```

```
      FL    RW    CL    CW
RW  3.27
CL 13.55 16.36
CW 15.79 18.47  2.59
BD  2.76  1.33 15.93 18.05
```

3.7 Clustering

A variety of clustering methods are available in R. The main *stats* package includes `hclust`, which computes hierarchical clustering trees according to certain agglomeration criteria, and `kmeans`, which computes a partition of the feature space into k regions (k selected by the user). Clusters are defined by inclusion in the regions, and "all cluster

centres are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster centre)" [from the man page]. The *cluster* package includes `pam`, a partitioning procedure like `kmeans`, which constrains the cluster centers to be chosen among actual feature vectors in the data; geometrically, the centers are "medoids".

The following code compares and plots (in Figure 6 various approaches to clustering.

```
> par(mfrow = c(2, 2))
> hc1 <- hclust(dist(kfeat))
> plot(hc1, main = "default hclust")
> hc2 <- hclust(dist(kfeat), method = "single")
> plot(hc2, main = "single linkage")
> hc3 <- hclust(dist(kfeat, method = "canberra"))
> library(cluster)
> hc4 <- pam(dist(kfeat), 4)
> clusplot(hc4)
> plot(silhouette(hc4))
> par(mfrow = c(1, 1))
```

The final plot is a "silhouette" plot, which displays measures of within-cluster to between-cluster separation.

```
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
"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) ).


'silhouette.default()' is now based on C code donated by Romain
Francois (the R version being still available as
'cluster:::silhouette.default.R').

Observations with a large s(i) (almost 1) are very well clustered,
a small s(i) (around 0) means that the observation lies between
two clusters, and observations with a negative s(i) are probably
placed in the wrong cluster.
```

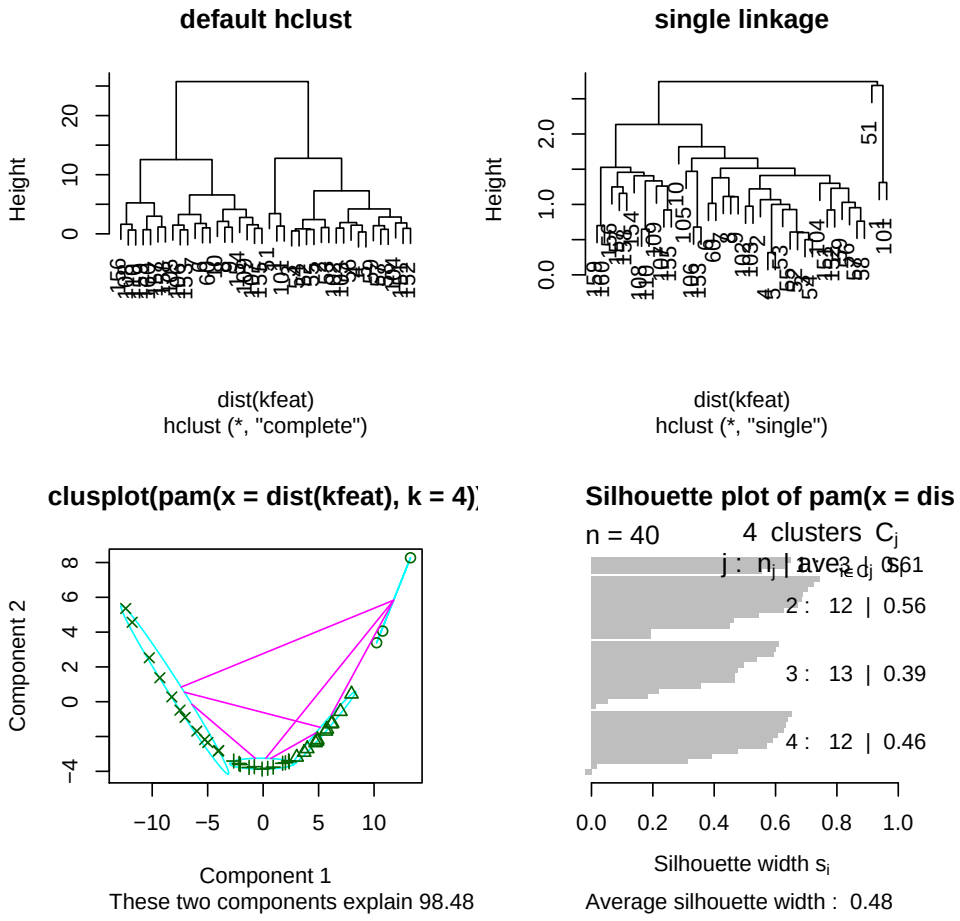


Figure 6: Displays of clustering procedures and evaluation using silhouette measures.

To convert a clustering tree, as created with `hclust`, into a set of cluster labels, the `cutree` function is used.

```
> c1 <- cutree(hc1, k = 4)
> table(c1)

c1
 1  2  3  4
 3 17 13  7

> c2 <- cutree(hc2, k = 4)
> table(c1, c2)

   c2
c1   1  2  3  4
  1  2  0  1  0
  2  0 17  0  0
  3  0  9  0  4
  4  0  0  0  7
```

Partition-based methods such as `pam` return a clustering labeling component.

```
> hc4$clustering

  1   2   3   4   5   6   7   8   9  10  51  52  53  54  55  56  57  58  59  60
  1   2   2   2   2   3   3   3   3   4   1   2   2   2   2   2   2   3   3   3
101 102 103 104 105 106 107 108 109 110 151 152 153 154 155 156 157 158 159 160
  1   2   2   3   3   3   4   4   4   4   3   3   3   4   4   4   4   4   4   4

> table(hc4$clustering, c1)

   c1
     1  2  3  4
  1  3  0  0  0
  2  0 12  0  0
  3  0  5  8  0
  4  0  0  5  7
```

Two clusterings agree if the cross tabulation of label frequencies as displayed above can be expressed as a matrix product DP, where D is a diagonal matrix with whole number entries, and P is a permutation matrix.

In this dataset, we have a way of labeling crabs according to gender and species.

```
> ss <- paste(as.character(kpsp), as.character(kpsex), sep = "")
> table(ss, hc4$clustering)
```

```
ss   1 2 3 4
  BF 1 6 3 0
  BM 1 4 4 1
  OF 0 0 3 7
  OM 1 2 3 4
```

We see that the agreement between gender-species labeling and clustering returned by PAM is not very good.

3.8 Principal components

The principal components analysis of a p-dimensional multivariate dataset is a re-expression of the data so that intrinsic structure (such as separation of groups) may be exhibited in a simpler space with fewer than p dimensions. Here (Figure 7) we compute the principal components and plot the first two. We use colors to discriminate species, but in a truly unsupervised context we would have no way of doing this.

```
> pp <- prcomp(crabs[, -c(1, 2, 3)])
> pp

Standard deviations:
[1] 11.862  1.139  1.000  0.368  0.279

Rotation:
     PC1    PC2    PC3    PC4    PC5
FL 0.289  0.323 -0.507  0.734  0.125
RW 0.197  0.865  0.414 -0.148 -0.141
CL 0.599 -0.198 -0.175 -0.144 -0.742
CW 0.662 -0.288  0.491  0.126  0.471
BD 0.284  0.160 -0.547 -0.634  0.439
```

.

A qualitative interpretation is that most of the variation in the data can be captured by a weighted sum of features, with most of the weight given to carapace dimensions. Much of the remaining variability is expressible through the difference between the carapace and non-carapace dimensions.

We can examine a three-dimensional perspective plot of the data, discarding data on two dimensions (Figure 8); Figure 9 employs the principal components reexpression, and uses different glyphs for species and gender. .

.

The interpretation is somewhat difficult, but this display has the advantage of being in the raw units of measurement. There are many decisions to be made regarding angle of

```
> plot(pp$x[, 1], pp$x[, 2], col = ifelse(crabs$sp == "O", "orange",
+     "blue"), pch = 16, xlab = "PC.1", ylab = "PC.2")
```

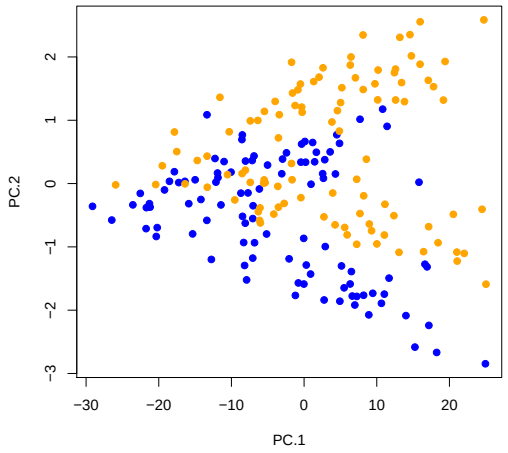


Figure 7: Principal components analysis of crab data: first two principal components.

```
> attach(crabs)
> library(scatterplot3d)
> scatterplot3d(CL, CW, RW, color = ifelse(crabs$sp == "O", "orange",
+     "blue"))
```

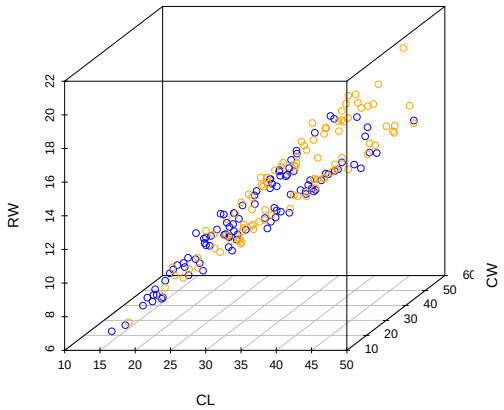


Figure 8: Three variables in a 3D display.

```
> scatterplot3d(pp$x[, 1], pp$x[, 2], pp$x[, 3], color = ifelse(crabs$sp ==
+     "O", "orange", "blue"), pch = ifelse(crabs$sex == "M", 1,
+     15))
```

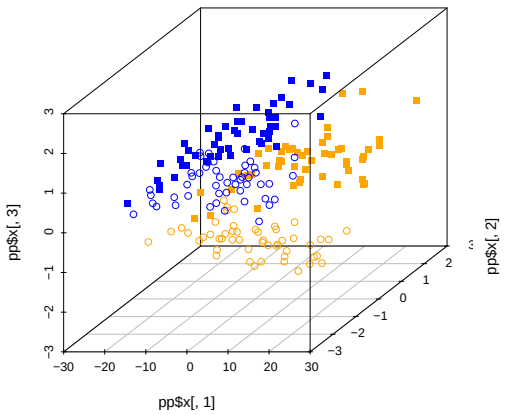


Figure 9: First 3 PC in a 3D display.

presentation and choice of glyph and shading. Dynamic high-dimensional visualizations are sometimes valuable; see www.ggobi.org for an open source tool that works with R.

Note the configuration of points in the pairwise display of all principal components, Figure 10.

Interpret each of the principal components informally. (See Ripley, chapter 9).

```
> pairs(pp$x, col = ifelse(crabs$sp == "O", "orange", "blue"),
+       pch = ifelse(crabs$sex == "M", 1, 16))
```

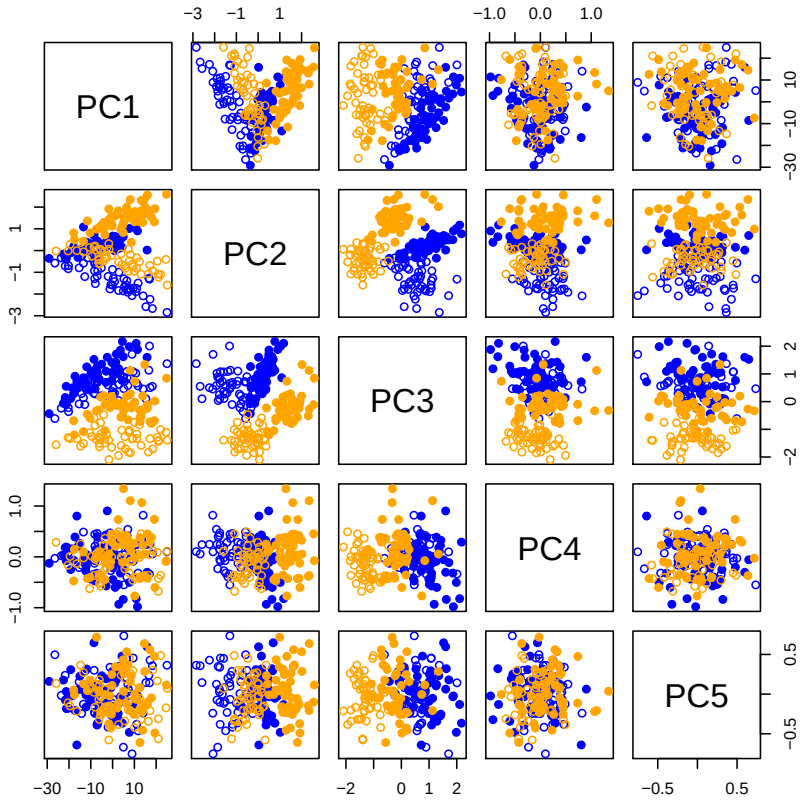


Figure 10: All PC.

3.9 Biplots

Presentation of data on both cases and variables construed as multivariate objects is accomplished by the biplot function. Here we use all the crabs data in a slightly reformatted matrix. We look at the second and third principal components.

```
> CM <- data.matrix(crabs[, -c(1, 2, 3)])
> rownames(CM) <- as.character(crabs$sp)


> biplot(prcomp(CM), choice = 2:3)
```

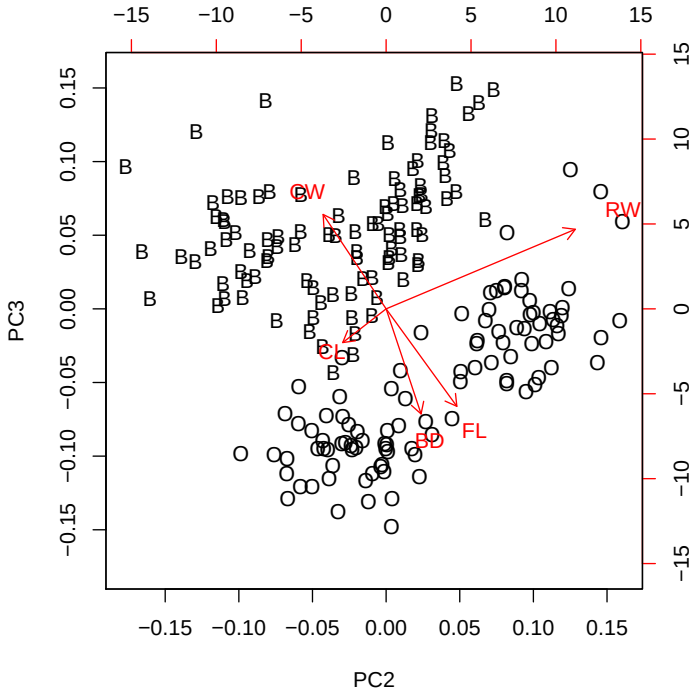


Figure 11: Biplot using PC2 and PC3.

3.10 Some supervised learning examples

3.10.1 Tree-based model evaluation and tuning

Recursive partitioning (also known as CART, classification and regression trees) is simple to use in R.

```
> library(rpart)
> tr1 <- rpart(sp ~ FL + RW + CL + CW + BD, data = crabs)
> tr1

n= 200

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

 1) root 200 100 B (0.5000 0.5000)
   2) FL< 17.4 135  47 B (0.6519 0.3481)
     4) CW>=36.2 40   4 B (0.9000 0.1000) *
     5) CW< 36.2 95  43 B (0.5474 0.4526)
      10) BD< 12.1 62  13 B (0.7903 0.2097)
        20) CW>=29.9 21   0 B (1.0000 0.0000) *
        21) CW< 29.9 41  13 B (0.6829 0.3171)
          42) FL< 12.2 34   6 B (0.8235 0.1765) *
          43) FL>=12.2 7   0 O (0.0000 1.0000) *
      11) BD>=12.1 33   3 O (0.0909 0.9091) *
   3) FL>=17.4 65  12 O (0.1846 0.8154)
     6) CW>=44.3 33  12 O (0.3636 0.6364)
      12) FL< 19.9 11   0 B (1.0000 0.0000) *
      13) FL>=19.9 22   1 O (0.0455 0.9545) *
     7) CW< 44.3 32   0 O (0.0000 1.0000) *
```

Plotting and annotation are straightforward (see Figure 12); a "post" method can be used for detailed graphics.

An important appraisal tool is the cost-complexity plot. A cross-validation is conducted as part of the model fitting procedure, and estimates of classification error rates (relative to those obtained by cross-validation with a single node) are summarized as a function of tree node count.

The parameters by which a tree-fitting procedure are controlled are specified in a control list:

```
> unlist(rpart.control())

      minsplit      minbucket             cp     maxcompete   maxsurrogate
         20.00           7.00           0.01           4.00           5.00
```

```
  usesurrogate surrogatestyle       maxdepth           xval
          2.00           0.00          30.00          10.00
```

We can cause the elaboration of the tree using the generic `update` method, see Figure 14.

```
> tr2 <- update(tr1, minsplit = 5)
```

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

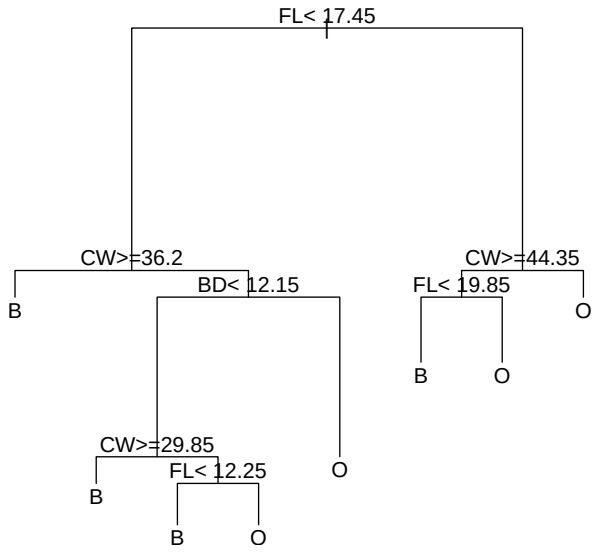


Figure 12: Default tree model fit to crabs data.

```
> plotcp(tr1)
```

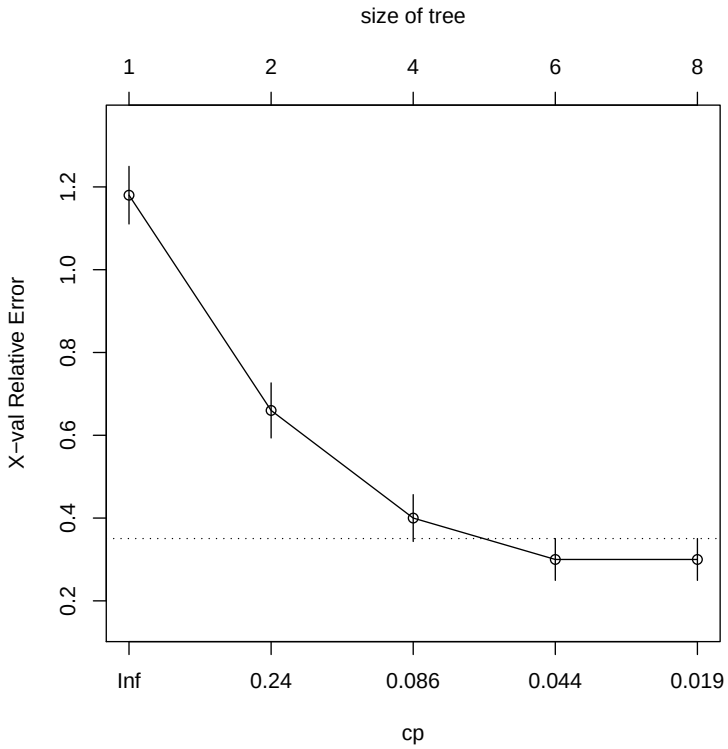


Figure 13: Cost-complexity plot for crabs data.

```
> plot(tr2)
> text(tr2)
```

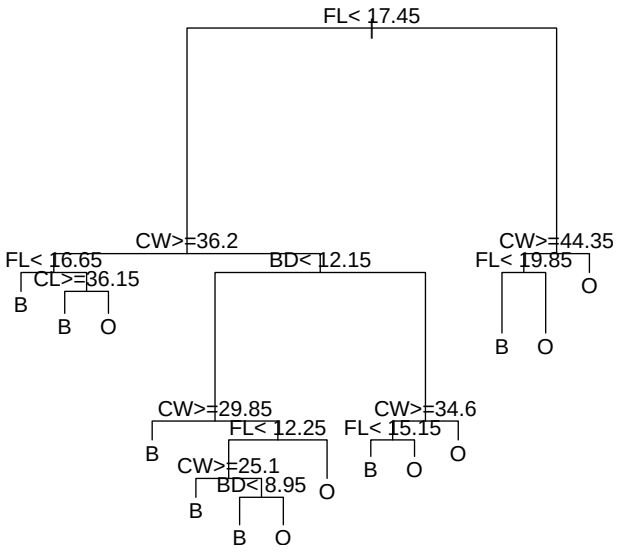


Figure 14: Elaborated tree model fit to crabs data.

3.10.2 Neural networks

Single layer feed-forward neural networks are easy to fit.

```
> library(nnet)
> nn1 <- nnet(sp ~ CL + CW + FL + RW + BD, data = crabs, size = 3)

# weights:  22
initial  value 160.265075
iter  10 value 131.082520
iter  20 value 0.058116
iter  30 value 0.000722
final  value 0.000001
converged

> nn1

a 5-3-1 network with 22 weights
inputs: CL CW FL RW BD
output(s): sp
options were - entropy fitting

> table(predict(nn1, type = "class"), crabs$sp)

     B   O
  B 100   0
  O   0 100
```

Selection of tuning parameters can be performed using tools in package *e1071*.

```
> library(e1071)
> library(Biobase)
> tnn <- cache("tnn", tune(nnet, sp ~ CL + CW + FL + RW + BD, data = crabs,
+     ranges = list(size = c(2, 3, 4, 5), decay = c(0.001, 0.01,
+         0.02, 0.03, 0.05))))

> tnn

Parameter tuning of `nnet':

- sampling method: 10-fold cross validation

- best parameters:
 size decay
    2 0.001

- best performance: 1
```

3.11 Some novel procedures

3.11.1 Association rule mining

Consider the tertize function:

```
> tertize = function(df) data.frame(lapply(df, function(x) cut(x,
+     br = quantile(x, c(0, 0.33, 0.66, 1)), include.lowest = TRUE,
+     labels = c("low", "mid", "hi"))))
> tertize(crabs[, -c(1, 2, 3)])[c(1:2, 20:21, 40:41), ]

    FL  RW  CL  CW  BD
1  low low low low low
2  low low low low low
20 low low mid mid low
21 mid mid mid mid mid
40  hi mid  hi  hi  hi
41  hi mid  hi  hi  hi

> sapply(tertize(crabs[, -c(1, 2, 3)]), table)

    FL RW CL CW BD
low 67 66 67 66 66
mid 65 69 66 66 66
hi  68 65 67 68 68

> tc = tertize(crabs[, -c(1, 2, 3)])
> dtc = data.frame(tc, sp = crabs$sp)
```

The data frame here has the form of a collection of itemsets, where each crab feature has a categorical measurement, and the crab species is also provided. This can be used to obtain a transactional analysis using the apriori algorithm:

```
> library(arules)

Loading required package: stats4
Loading required package: Matrix

> tdtc = as(dtc, "transactions")
> ap = apriori(tdtc)

parameter specification:
 confidence minval smax arem  aval originalSupport support minlen maxlen target
        0.8    0.1    1 none FALSE            TRUE     0.1      1      5  rules
   ext
```

```
 FALSE

algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[17 item(s), 200 transaction(s)] done [0.00s].
sorting and recoding items ... [17 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [351 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
> sap = subset(ap, subset = rhs %in% c("sp=0", "sp=B"))
> inspect(sap)
```

```
  lhs         rhs    support confidence lift
1 {RW=hi,
    BD=hi} => {sp=0}   0.190      0.809 1.62
2 {FL=hi,
    RW=hi} => {sp=0}   0.205      0.820 1.64
3 {FL=hi,
    RW=hi,
    BD=hi} => {sp=0}   0.190      0.809 1.62
```

The support $supp(X)$ of an itemset X is the proportion of transactions containing the itemset. The confidence of rule $X \rightarrow Y$ is $conf(X \rightarrow Y) = supp(X \cup Y)/supp(X)$. The apriori algorithm is an efficient rule search that can identify rules satisfying conditions on antecedent support and rule confidence. By default, support must exceed .1 and confidence must exceed .8 to be kept.

3.11.2 Weka

There are a variety of machine learning algorithms collected together in a package called RWeka. An interface to a tree visualization paradigm based on ATT graphviz is also available.

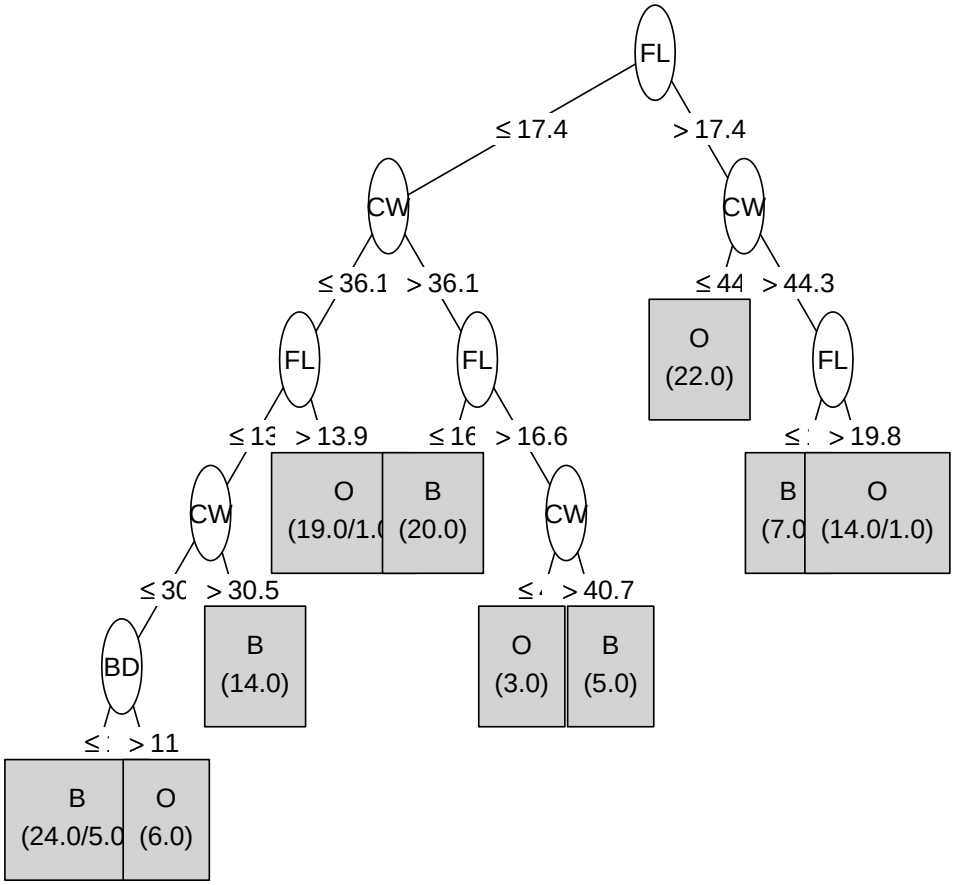
```
> library(RWeka)
```

```
Loading required package: rJava
Loading required package: grid
```

```
> zz = J48(sp ~ CW + CL + RW + BD + FL, data = crabs, control = "-R")
> library(party)

Loading required package: survival
Loading required package: splines
Loading required package: modeltools
Loading required package: coin
Loading required package: mvtnorm
Loading required package: zoo
Loading required package: sandwich
Loading required package: strucchange

Attaching package: 'strucchange'


        The following object(s) are masked from package:MASS :

         SP500

> write_to_dot(zz)

digraph J48Tree {
N0 [label="FL" ]
N0->N1 [label="<= 17.4"]
N1 [label="CW" ]
N1->N2 [label="<= 36.1"]
N2 [label="FL" ]
N2->N3 [label="<= 13.9"]
N3 [label="CW" ]
N3->N4 [label="<= 30.5"]
N4 [label="BD" ]
N4->N5 [label="<= 11"]
N5 [label="B (24.0/5.0)" shape=box style=filled ]
N4->N6 [label="> 11"]
N6 [label="O (6.0)" shape=box style=filled ]
N3->N7 [label="> 30.5"]
N7 [label="B (14.0)" shape=box style=filled ]
N2->N8 [label="> 13.9"]
N8 [label="O (19.0/1.0)" shape=box style=filled ]
N1->N9 [label="> 36.1"]
N9 [label="FL" ]
N9->N10 [label="<= 16.6"]
N10 [label="B (20.0)" shape=box style=filled ]
```

```
N9->N11 [label="> 16.6"]
N11 [label="CW" ]
N11->N12 [label="<= 40.7"]
N12 [label="O (3.0)" shape=box style=filled ]
N11->N13 [label="> 40.7"]
N13 [label="B (5.0)" shape=box style=filled ]
N0->N14 [label="> 17.4"]
N14 [label="CW" ]
N14->N15 [label="<= 44.3"]
N15 [label="O (22.0)" shape=box style=filled ]
N14->N16 [label="> 44.3"]
N16 [label="FL" ]
N16->N17 [label="<= 19.8"]
N17 [label="B (7.0)" shape=box style=filled ]
N16->N18 [label="> 19.8"]
N18 [label="O (14.0/1.0)" shape=box style=filled ]
}
```

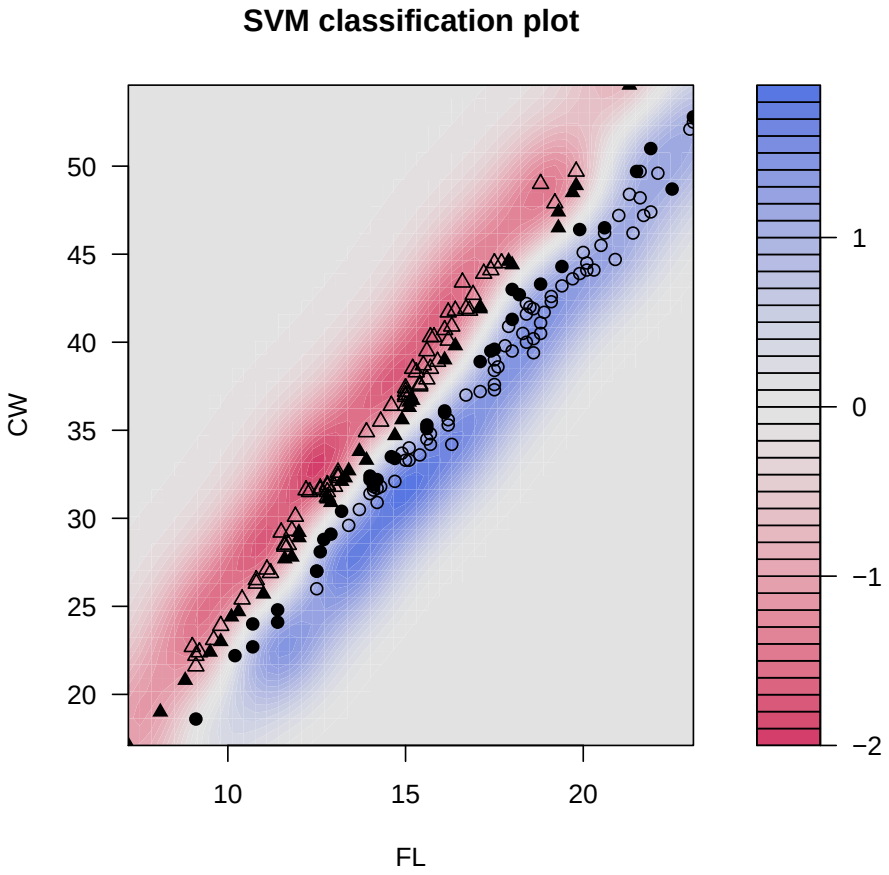
> plot(zz)



3.11.3 kernlab

A rich collection of methods involving kernel transformations is available in the *kernlab* package.

```
> library(kernlab)
> zz = ksvm(sp ~ CW + FL, data = crabs)

Using automatic sigma estimation (sigest) for RBF or laplace kernel

> table(crabs$sp, predict(zz))

     B   O
  B 100   0
  O   1  99

> plot(zz, data = crabs)
```



4 The *MLInterfaces* package

Bioconductor's *MLInterfaces* package aims to simplify use of machine learning methods in statistical genomics applications, and to focus infrastructure development to further the simplification process. Table 1 indicates the scope of the current interface.

The fact that a function is not covered in *MLInterfaces* does not mean that one cannot use it with genomic data. However, *MLInterfaces* simplifies the use of machine learning tools with *exprSet* instances, and provides uniform output containers for machine learning algorithms.

	Package	Functions covered
1	*class*	`knn, knn1, knn.cv, lvq1, lvq2, lvq3, olvq1, som SOM`
2	*cluster*	`pam, agnes, clara, diana, fanny, silhouette`
3	*e1071*	`bclust, cmeans, cshell, lca naiveBayes, svm`
4	*gbm*	`gbm`
5	*ipred*	`bagging, ipredknn, slda, cv`
6	*MASS*	`isoMDS, lda, qda`
7	*nnet*	`nnet`
8	*pamr*	`pamr`
9	*randomForest*	`randomForest`
10	*rpart*	`rpart`
11	*stats*	`kmeans, hclust`

Table 1: Packages and functions covered by *MLInterfaces*.

4.1 *exprSet* refresher

Golub's leukemia data are often used for illustrative purposes. These are derived from the Affymetrix(tm) HU6800 chip. We store them in an *exprSet* instance.

```
> library(golubEsets)
> data(golubMerge)
> golubMerge

Expression Set (exprSet) with
        7129 genes
        72 samples
        phenoData object with 11 variables and 72 cases
        varLabels
                Samples: Sample index
                ALL.AML: Factor, indicating ALL or AML
                BM.PB: Factor, sample from marrow or peripheral blood
                T.B.cell: Factor, T cell or B cell leuk.
                FAB: Factor, FAB classification
                Date: Date sample obtained
                Gender: Factor, gender of patient
                pctBlasts: pct of cells that are blasts
                Treatment: response to treatment
                PS: Prediction strength
                Source: Source of sample
```

Some of the tasks supported by exprSet infrastructure include

- extraction of expression measures as a matrix

  ```
  > dim(exprs(golubMerge))

  [1] 7129   72

  > exprs(golubMerge)[1:4, 1:4]

                 [,1] [,2] [,3] [,4]
  AFFX-BioB-5_at -342  -87   22 -243
  AFFX-BioB-M_at -200 -248 -153 -218
  AFFX-BioB-3_at   41  262   17 -163
  AFFX-BioC-5_at  328  295  276  182
  ```

- extraction of sample-level data as a data.frame

  ```
  > dim(pData(golubMerge))
  ```

```
[1] 72 11
```

- extraction of vectors of sample-level data as list components

```
> table(golubMerge$ALL.AML)

ALL AML
 47  25
```

- closure of *exprSet* object class under gene or sample subsetting

```
> dim(exprs(golubMerge[1:5, 1:5]))

[1] 5 5
```

4.2 MLInterfaces and two-gene machines

First we consider qualitative differences between supervised machine learning tools. The `planarPlot` function is helpful for qualitative assessment.

We take two genes from Golub's dataset and form the sub-exprSet:

```
> litG <- golubMerge[7000:7001, ]
```

We now compute four learning machine outputs, using CART, random forests, k nearest neighbors, and a support vector machine.

```
> library(MLInterfaces)
> ld1 <- ldaB(litG, "ALL.AML", 1:35)
> kn1 <- knnB(litG, "ALL.AML", 1:35)
> rf1 <- randomForestB(litG, "ALL.AML", 1:35)
> svm1 <- svmB(litG, "ALL.AML", 1:35)
```

Note that the calling sequences for default applications of the learning models are identical, and are quite simple. The parameters used are the name of the exprSet, the name of the phenoData variable to be used for classification, and the indices of the training set. MLInterfaces does not easily support computation of resubstitution-based assessments of learners. Simple training-test data decompositions or variations on cross-validation are readily carried out, as will be illustrated below.

Once an MLInterfaces learner has been computed, one can obtain a brief report:

```
> rf1

MLOutput instance, method= randomForest
Call:
 randomForestB(exprObj = litG, classifLab = "ALL.AML", trainInd = 1:35)
predicted class distribution:
ALL AML
 12  25
```

```
> svm1

MLOutput instance, method= svm
Call:
 svmB(exprObj = litG, classifLab = "ALL.AML", trainInd = 1:35)
predicted class distribution:
ALL AML
 21  16
```

The computation of the confusion matrix using the test data (data not used to teach the machine) is very easy:

```
> confuMat(ld1)

     predicted
given ALL AML
  ALL  16  10
  AML   6   5

> confuMat(kn1)

     predicted
given ALL AML
  ALL  12  14
  AML   2   9
```

To see the prediction regions implied by the construction of the machines on the training data, use planarPlot, as in Figure 15:

```
> par(mfrow = c(2, 2))
> planarPlot(ld1, litG, "ALL.AML")
> title("LDA")
> planarPlot(kn1, litG, "ALL.AML")
> title("K-NN")
> planarPlot(rf1, litG, "ALL.AML")
> title("random forest")
> planarPlot(svm1, litG, "ALL.AML")
> title("svm")
> par(mfrow = c(1, 1))
```

The R structure created by the R function implementing the machine learning method can be obtained using the `RObject` accessor method:

```
> RObject(svm1)
```

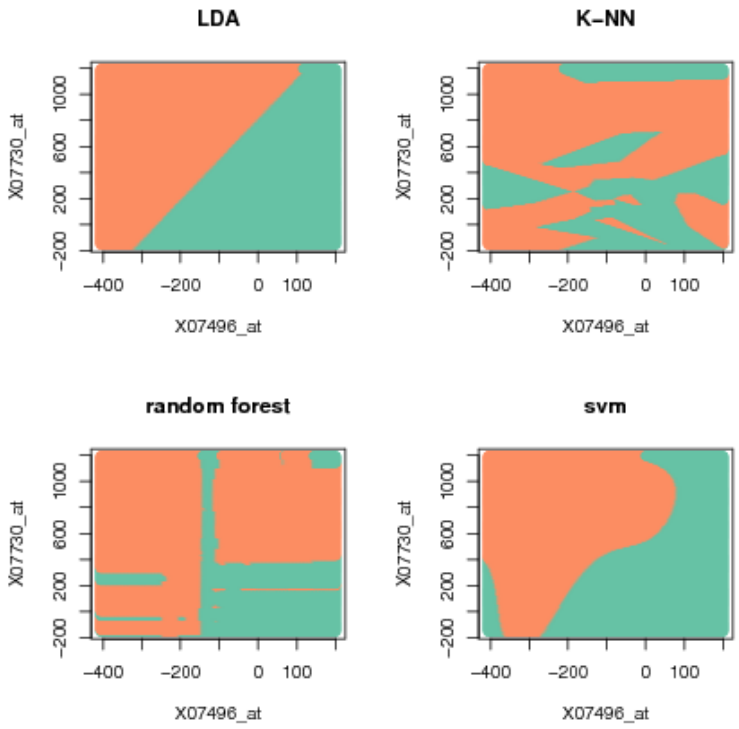


Figure 15: Decision regions computed by two-gene machines.

```
Call:
svm.default(x = trainDat, y = cl, scale = scale, type = type, kernel = kernel,
    degree = degree, gamma = gamma, coef0 = coef0, cost = cost, nu = nu,
    class.weights = class.weights, cachesize = cachesize, tolerance = tolerance,
    epsilon = epsilon, shrinking = shrinking, cross = cross, fitted = fitted,
    subset = subset, na.action = na.action)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  28

> RObject(ld1)

Call:
lda(trainDat, grouping = cl, prior = prior, tol = tol, method = method,
    CV = CV, nu = nu)

Prior probabilities of groups:
ALL AML
0.6 0.4

Group means:
    X07496_at X07730_at
ALL     -60.8       238
AML    -123.1       423

Coefficients of linear discriminants:
               LD1
X07496_at -0.00678
X07730_at  0.00218
```

4.3 Do we need to filter?

Machine learning methods differ with respect to computational resources required on a given set of data. For the merged Golub data, we find that a reasonably equipped PowerBook can perform LDA without filtering.

Estimation of required times can be conducted as follows:

```
> cache <- function(name, expr) {
+     cachefile <- paste("tmp-", name, ".RData", sep = "")
+     if (file.exists(cachefile)) {
+         load(cachefile)
+     }
+     else {
+         assign(name, expr)
+         save(list = name, file = cachefile)
+     }
+     get(name)
+ }
> ut <- cache("ut", unix.time(ldaB(golubMerge[1:1000, ], "ALL.AML",
+     1:35)))
> ut

[1] 2.960 0.363 4.065 0.000 0.000

> ut2 <- cache("ut2", unix.time(ldaB(golubMerge[1:2000, ], "ALL.AML",
+     1:35)))
> ut2

[1]  6.96  1.21 12.00  0.00  0.00

> ut3 <- cache("ut3", unix.time(ldaB(golubMerge[1:3000, ], "ALL.AML",
+     1:35)))
> ut3

[1] 13.42  2.62 26.19  0.00  0.00
```

4.3.1 Non-specific filtering

If a method of interest seems to outstrip available resources, gene filtering can be performed. The *genefilter* package has a variety of tools that can be used for distribution based or non-specific filtering. These methods ignore the annotation of the genes. We will illustrate use of coefficient-of-variation (CV) filtering.

First let's assess the distribution of CV over genes.

```
> CV <- function(x) sd(x)/abs(mean(x))
> ACV <- apply(exprs(golubMerge), 1, CV)
> summary(ACV)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
9.66e-02 5.45e-01 8.15e-01 3.55e+00 1.55e+00 2.60e+03
```

If we want to eliminate the 50% of genes with the smallest coefficient of variation, we create and run the filter as follows:

```
> library(genefilter)
> cvf <- cv(0.82)
> cvff <- filterfun(cvf)
> docv <- genefilter(golubMerge, cvff)
> summary(apply(exprs(golubMerge[docv, ]), 1, CV))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.82    1.09    1.56    6.58    2.97 2600.00
```

The output of genefilter is a logical variable:

```
> sum(docv)

[1] 3544
```

We use this to construct a filtered exprSet:

```
> LG <- golubMerge[docv, ]
> confuMat(ldaB(LG, "ALL.AML", 1:35))

     predicted
given ALL AML
  ALL  26   0
  AML   2   9
```

Some approaches to filtering are not yet handled in *genefilter*. To filter away genes with small median absolute deviations, you could do the following:

```
> mads <- apply(exprs(golubMerge), 1, mad)
> topm <- golubMerge[mads > 142, ]
> flda <- ldaB(topm, "ALL.AML", 1:35)
> confuMat(flda)

     predicted
given ALL AML
  ALL  26   0
  AML   1  10
```

4.3.2 Substantive filtering

Suppose we want to attempt learning in the context of a specific biological process. For example, we may want to know if different genes associated with apoptosis support discrimination of ALL and AML in different ways.

There are various ways to search the annotation resource of Bioconductor; one function that may be useful is jotted down here. You can enter this manually into your workspace.

```
> revgrep <- function(x, env) {
+     el <- as.list(env)
+     vals <- unlist(el)
+     tags <- names(vals)
+     ans <- tags[gg <- grep(x, vals)]
+     if (length(ans) > 0)
+         names(ans) <- vals[gg]
+     ans
+ }
```

We'll search the gene name environment for references to a substring of 'apoptosis'.

```
> library(hu6800)
> app <- revgrep("apop", hu6800GENENAME)
> length(app)

[1] 16

> app[1:2]

caspase 1, apoptosis-related cysteine peptidase (interleukin 1, beta, convertase)
                                                                    "M87507_s_at"
                                caspase 6, apoptosis-related cysteine peptidase
                                                                    "U20536_s_at"

> substr(names(app), 1, 36)

 [1] "caspase 1, apoptosis-related cystein"
 [2] "caspase 6, apoptosis-related cystein"
 [3] "caspase 4, apoptosis-related cystein"
 [4] "CASP8 and FADD-like apoptosis regula"
 [5] "caspase 8, apoptosis-related cystein"
 [6] "caspase 8, apoptosis-related cystein"
 [7] "caspase 10, apoptosis-related cystei"
 [8] "caspase 2, apoptosis-related cystein"
 [9] "caspase 2, apoptosis-related cystein"
```

```
[10] "BCL2-interacting killer (apoptosis-i"
[11] "caspase 7, apoptosis-related cystein"
[12] "caspase 5, apoptosis-related cystein"
[13] "caspase 9, apoptosis-related cystein"
[14] "caspase 10, apoptosis-related cystei"
[15] "caspase 8, apoptosis-related cystein"
[16] "caspase 3, apoptosis-related cystein"

> apmrg <- golubMerge[app, ]
> apmrg

Expression Set (exprSet) with
        16 genes
        72 samples
        phenoData object with 11 variables and 72 cases
        varLabels
                Samples: Sample index
                ALL.AML: Factor, indicating ALL or AML
                BM.PB: Factor, sample from marrow or peripheral blood
                T.B.cell: Factor, T cell or B cell leuk.
                FAB: Factor, FAB classification
                Date: Date sample obtained
                Gender: Factor, gender of patient
                pctBlasts: pct of cells that are blasts
                Treatment: response to treatment
                PS: Prediction strength
                Source: Source of sample
```

4.4 Tuning a learner

The *e1071* package includes tools to navigate the tuning parameter space. This is not yet integrated with *MLInterfaces*, so one must create the matrix and factor arguments `train.x` and `train.y`.

```
> set.seed(1234)
> tn1 <- tune(svm, train.x = t(exprs(apmrg)), train.y = apmrg$ALL.AML,
+     ranges = list(cost = c(0.5, 1, 1.5, 2), gamma = seq(0.01,
+         0.1, 0.01)))
> tn1

Parameter tuning of `svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
 cost gamma
    2  0.08

- best performance: 0.248
```

```
> confuMat(svmB(apmrg, "ALL.AML", 1:35, cost = 2, gamma = 0.1))

     predicted
given ALL AML
  ALL  21   5
  AML   4   7
```

4.5 Measures of variable importance

Two procedures handled by *MLInterfaces* compute measures of variable importance. We can visualize relative importance with a plot method (Figure 16).

```
> aprf <- randomForestB(apmrg, "ALL.AML", 1:35, importance = TRUE,
+     ntrees = 5000)
```

4.6 Cross-validation

The `xval` method can be used to simplify cross-validation tasks. Case-based and group-based partitions are supported, along with more general approaches to division of data between training and test components. Extensions supplied by S. Henderson of Imperial College allow incorporation of feature selection.

```
> lda1 <- ldaB(apmrg, "ALL.AML", 1:35)
> confuMat(lda1)

     predicted
given ALL AML
  ALL  17   9
  AML   4   7
```

```
> xv1 <- xval(apmrg, "ALL.AML", ldaB, "LOO", 0:0)
> table(xv1, apmrg$ALL.AML)

xv1   ALL AML
  ALL  37  13
  AML  10  12
```

```
> par(las = 2, mar = c(5, 5, 3, 3))
> plot(getVarImp(aprf), n = 10, resolveenv = hu6800SYMBOL)
```

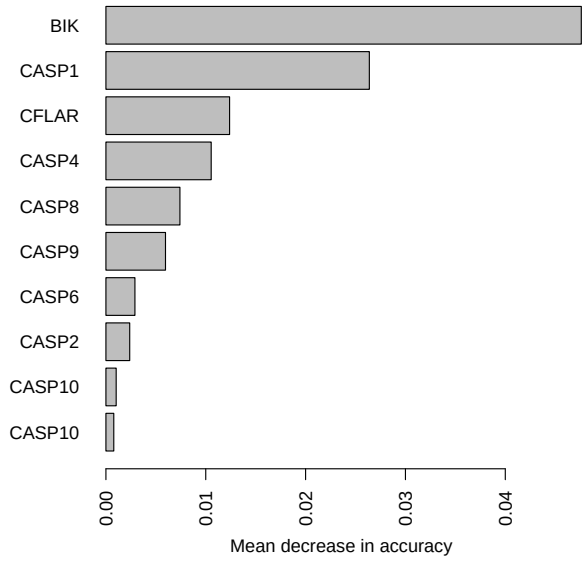


Figure 16: Variable importance in the apoptosis subset of genes.

To use the feature selection procedure, we add a function that returns a score for each feature.

```
> t.fun <- function(data, fac) {
+     require(genefilter)
+     xd <- matrix(as.double(exprs(data)), nrow = nrow(exprs(data)))
+     return(abs(rowttests(xd, data[[fac]], tstatOnly = FALSE)$statistic))
+ }
```

Then we pass this as the fsFun parameter.

```
> xv2 <- xval(apmrg, "ALL.AML", ldaB, "LOO", 0:0, fsFun = t.fun,
+     fsNum = 5)
> if (is.list(xv2)) table(xv2$out, apmrg$ALL.AML) else table(xv2,
+     apmrg$ALL.AML)

     ALL AML
  ALL  41  12
  AML   6  13

> xv3 <- xval(apmrg, "ALL.AML", ldaB, "LOO", 0:0, fsFun = t.fun,
+     fsNum = 10)
> if (is.list(xv3)) table(xv3$out, apmrg$ALL.AML) else table(xv3,
+     apmrg$ALL.AML)

     ALL AML
  ALL  39  13
  AML   8  12
```

5 Problems

You will use the bcmlpack package

1. The mystRMA data in bcmlpack are arrays from 10 cell lines studied in the Novartis symatlas. There are five pairs of assays with two biological replicates in each pair. Labels were lost on five of the replicates. Use machine learning tools to pair the samples with known cell type to their partners:

   ```
   > library(bcmlpack)
   > data(mystRMA)
   > sampleNames(mystRMA)

    [1] "ee.cd71.cel"   "lostA.cel"     "lostW.cel"     "lostX.cel"
    [5] "lostY.cel"     "lostZ.cel"     "lym.rajiA.cel" "nk.cd56.cel"
    [9] "t.cd4.cel"     "t.cd8.cel"
   ```

In other words, draw the lines connecting samples listed in column I to their partners in column II:

```
column I          column II
  A                 nk.cd56
  W                 t.cd4
  X                 t.cd8
  Y                 lym.raji
  Z                 ee.cd71
```

Describe your choice of learning tool and state your confidence in the solution.

2. Once the pairings have been completed, use biplots to compare the expression patterns of the CD4 and CD8 T cells. Name some genes that are useful to discriminate these two families of cells. Use limma to do a standard differential expression analysis, and list the top ten discriminating genes.

3. run and explain the following code:

```
library(bcmlpack)
library(Biobase)
data(mystRMA)
tlym = 1:10 %in% c(3,5,9,10)
islym = factor(1*tlym)
pData(mystRMA)$islym = islym
phenoData(mystRMA)@varLabels$islym = "t4 or t8 lymphocyte"
library(MLInterfaces)
ff = rpartB(mystRMA[1:500,], "islym", 1:5 , minsplit=1 )
plotcp(RObject(ff))
plot(RObject(ff))
text(RObject(ff))
library(annotate)
library(hgu133a)
lookUp("1007_s_at", "hgu133a", "GENENAME")
gg = randomForestB( mystRMA[1:500,], "islym", 1:5, importance=TRUE)
par(mar=c(5,19,5,5))
plot(getVarImp(gg), n=10, resolveenv=hgu133aGENENAME)
confuMat(gg)
```

What are the arbitrary parts of the steps taken above, and how can the analysis be made more thorough?

4. The bcmlpack package includes an exprSet "fuse" related to the ALL package. In this problem we will analyze the relationship between gene expression and BCR/ABL fusion, encoded in the mol.biol phenoData variable.

(a) create a subset of genes related to interleukins

```
library(annotate)
library(GO)
library(hgu95av2)
lt = sapply(as.list(GOTERM), Term)
gotags = names(lt)[grep("interleukin", lt)]
allil = as.character(na.omit(unlist(lookUp(gotags,
    "hgu95av2", "GO2ALLPROBES"))))
library(bcmlpack)
data(fuse)
lit = fuse[ geneNames(fuse) %in% allil, ]
```

(b) use limma to assess whether these genes are differentially expressed in BCR/ABL

(c) use various tools in MLInterfaces or other machine learning packages to measure "variable importance" and predictability of BCR/ABL fusion on the basis of interleukin-related genes – try out various forms of cross-validation

(d) substitute other gene families/functions for "interleukin" and extend your analysis to try to explain the BCR/ABL fusion phenotype on the basis of expression.