BioC 2008 practical: Machine learning with genome-scale data

©2008 VJ Carey stvjc at channing.harvard.edu

July 24, 2008

Contents

1 Introduction

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.

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.

- All prediction or clustering algorithms, like all modeling procedures, rely on a choice of distance metric that permits quantitative evaluation of similarity and dissimilarity among objects. The choice can affect results and there is typically no *a priori* basis for selecting the distance function. A one-minus-correlation distance can be very different from euclidean distance for a given pair of genes.

- "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 in 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 extensive background.

2 Data structures

The representation of genome-scale data has impacts on many aspects of data analysis. For microarray measures of mRNA abundance (expression arrays) we use the `ExpressionSet` to unify data and metadata on a set of arrays. Let G denote the number of genes probed on the array, and N denote the number of samples which will be assumed independent (familial data structures not directly considered). The key points for an `ExpressionSet` instance `X` are

- `exprs(X)` is a $G \times N$ `matrix` of expression values (typically on the log scale)
- `pData(X)` is a $N \times R$ `data.frame` of sample-level variables
- `X$v` is an N-vector of values on the sample-level variable named `v`
- `X[G, S]` is a new `ExpressionSet` instance with genes restricted according to predicate `G` and samples restricted according to predicate `S`

Our first example is the Chiaretti et al. dataset on acute lymphocytic leukemia.

```
> library(ALL)
> data(ALL)
> ALL
```

We will focus on the molecular classification of leukemia-type within B-cell leukemias, and create a subset of B-cell ALL samples that are either positive or negative for BCR/ABL gene fusion

```
> table(ALL$BT, ALL$mol.biol)

     ALL1/AF4 BCR/ABL E2A/PBX1 NEG NUP-98 p15/p16
  B         0       2        1   2      0       0
  B1       10       1        0   8      0       0
  B2        0      19        0  16      0       1
  B3        0       8        1  14      0       0
  B4        0       7        3   2      0       0
  T         0       0        0   5      0       0
  T1        0       0        0   1      0       0
  T2        0       0        0  15      0       0
  T3        0       0        0   9      1       0
  T4        0       0        0   2      0       0
```

```
> bALL = ALL[, substr(ALL$BT, 1, 1) == "B"]
> fbALL = bALL[, bALL$mol.biol %in% c("BCR/ABL", "NEG")]
> fbALL$mol.biol = factor(fbALL$mol.biol, levels = c("NEG", "BCR/ABL"))
> fbALL$binFus = 1 * (fbALL$mol.biol == "BCR/ABL")
```

In the last assignment, we make a 0-1 representation of the mol.biol factor.

3 Logistic regression and linear discriminant analysis

A standard analysis of a problem with a two-class outcome focuses on modeling the probability of class membership. With our representation, we consider the probability that a sample is positive for BCR/ABL fusion (this event is denoted $F = 1$, F a binary random variable) conditional on the level of expression of a selected gene. A linear logistic model takes the form

$$\text{logit } \Pr(F = 1|x) = \alpha + x\beta$$

3.1 Manual fit of a logistic regression

This can be fit manually using R as follows. We use the third gene on the array as x.

```
> lr1 = glm(formula = fbALL$binFus ~ exprs(fbALL)[3, ], family = binomial)
> summary(lr1)

Call:
glm(formula = fbALL$binFus ~ exprs(fbALL)[3, ], family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5067  -1.0964  -0.8908   1.1876   1.5081

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)          7.345      4.850   1.514    0.130
exprs(fbALL)[3, ]   -1.931      1.253  -1.541    0.123

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 109.20  on 78  degrees of freedom
Residual deviance: 106.73  on 77  degrees of freedom
AIC: 110.73

Number of Fisher Scoring iterations: 4
```

Exercises

- Visualize the class-specific distributions of the modeled gene.
- What is the name of the gene used in this analysis?

3.2 Using MLInterfaces

3.2.1 A simple application: single gene logistic regression

Here we use a generic method that operates on `ExpressionSet` instances and formulae to carry out the same logistic regression analysis, but with cross-validation. In this species of cross-validation, the dataset is partitioned into five subsets, each of which is formed to have approximately equal representation of the outcome classes.

```
> library(MLInterfaces)
> lr2 = MLearn(mol.biol~., fbALL[3,], glmI.logistic(thresh=.5),
+    xvalSpec("LOG", 5, balKfold.xvspec(5)),
+    family=binomial)
> lr2
```

```
MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fbALL[3, ], method = glmI.logistic(thresh = 0.5),
    trainInd = xvalSpec("LOG", 5, balKfold.xvspec(5)), family = binomial)
Predicted outcome distribution for test set:

 0  1
53 26
history of feature selection in cross-validation available; use fsHistory()
```

Notice that the result of this call is not a table of coefficients, but an object. That object has the same formal structure for any successful call to `MLearn`.

We can get the table of coefficients through the following:

```
> summary(RObject(RObject(lr2)[[1]]$mlans))
```

```
Call:
lfun(formula = formula, family = ..1, data = trdata)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.434  -1.046  -0.877   1.230   1.568

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    7.055      5.238   1.347    0.178
X1002_f_at    -1.894      1.356  -1.397    0.163

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 84.915  on 61  degrees of freedom
Residual deviance: 82.886  on 60  degrees of freedom
AIC: 86.886

Number of Fisher Scoring iterations: 4
```

The access path to the coefficients is somewhat convoluted – two levels of storage must be traversed. At the top level (interior call to `RObject`, applied to `lr2`), we are looking into the result of five cross-validation iterations. We pick the first using `[[1]]`. This returns a list. The element named `mlans` holds the table of coefficients (actually the glm model fit), retrieved using the outer call to `RObject`, rendered using `summary`.

The `MLearn` method is tailored to predictive applications. In the use of logistic modeling, a threshold is specified in the third argument to MLearn. The predicted probability of fusion for each sample is computed according to the fitted model and if it exceeds the threshold parameter, the sample is predicted to be in the fusion class; otherwise it is predicted to be negative for fusion.

The confusion matrix for the cross-validated prediction exercise cross-tabulates known class vs predicted class for all samples. The proportion of off-diagonal entries is an estimate of the misclassification rate.

Exercises

- If we relax the threshold for classifying to fusion to 40%, what happens to the misclassification rate for cross-validated, single-gene, logistic regression-based prediction?

3.2.2 Prediction with many genes

We will now illustrate linear discriminant analysis (LDA). We use a collection of genes, denoted x_i to characterize sample i, and compute the multivariate mean for each class (denoted μ_F and μ_N for fusion and negative respectively) and the common covariance matrix Σ for all the observations. If π_F and π_N are the proportions of fusion and negative samples in the dataset, The classification procedure is to allocate the sample with gene 'signature' x_i to class F when

$$LD(x_i) = (\mu_F - \mu_N)^t \Sigma^{-1}(x_i - .5(\mu_F + \mu_N)) > \log(\pi_F/\pi_N).$$

For this to be feasible on modest hardware, we need to filter the genes in use. We use genefilter's `nsFilter` procedure to eliminate genes with relatively low variability across samples.

```
> library(genefilter)
> ffbALL = nsFilter(fbALL, var.func=var, var.cutoff=.9)
> ffbALL[[2]] # check exclusion events
```

```
$numLowVar
[1] 8104

$numDupsRemoved
[1] 3100

$feature.exclude
[1] 19

$numRemoved.ENTREZID
[1] 501

> ffbALL = ffbALL[[1]] # keep only ExpressionSet
```

Now we construct a cross-validated linear discriminant analysis using all filtered genes.

```
> lda1 = MLearn(mol.biol ~ ., ffbALL, ldaI, xvalSpec("LOG", 5,
+     balKfold.xvspec(5)))

> mm = confuMat(lda1)
> mcr = (sum(mm) - sum(diag(mm)))/sum(mm)
> mcr

[1] 0.1898734
```

Is this a legitimate application? Is 0.19 a good estimate of the misclassification rate of the procedure? Perhaps not, because feature selection was conducted outside the cross-validation procedure. The set of genes filtered away will differ from iteration to iteration of the cross-validation. This process can be factored into our cross-validation using the `fsFun` parameter of `xvalSpec`. Ideally we would apply this feature selection process to the full fbALL ExpressionSet but on a small computer this seems to cause problems. We illustrate with only 2500 genes:

```
> lda2 = MLearn(mol.biol ~ ., fbALL[1:2500, ], ldaI, xvalSpec("LOG",
+     5, balKfold.xvspec(5), fsFun = fs.topVariance(0.9)))

> lda2

MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = fbALL[1:2500, ], method = ldaI,
    trainInd = xvalSpec("LOG", 5, balKfold.xvspec(5), fsFun = fs.topVariance(0.9)))
Predicted outcome distribution for test set:
    NEG BCR/ABL
     40      39
history of feature selection in cross-validation available; use fsHistory()
```

```
> confuMat(lda2)
```

```
         predicted
given     NEG BCR/ABL
  NEG      36       6
  BCR/ABL   4      33
```

```
> length(fsHistory(lda2)[[1]])
```

```
[1] 250
```

3.3 Summary

- ExpressionSets store assay and sample-level data from microarray experiments;
- Manual application of standard statistical modeling tools to test specific gene effects is feasible, but variation in calling sequence and return values reduces efficiency of application and interpretation;
- MLInterfaces MLearn supports direct application of supervised learning methods to ExpressionSet instances
  - standard formula interface may refer to any phenoData variable for response
  - learnerSchema instances select the algorithm to be used; we looked at glmI.logistic and ldaI
  - cross-validation is supported through the xvalSpec object
  - algorithmic feature selection can be embedded in cross-validation
- MLearn returns a structured object responding to `confuMat`, `RObject`, and, when relevant, `fsHistory`

4 Some technical details of MLInterfaces

4.1 The signature of MLearn

```
> showMethods("MLearn")
```

```
Function: MLearn (package MLInterfaces)
formula="formula", data="ExpressionSet", method="character", trainInd="numeric", mlSpec
formula="formula", data="ExpressionSet", method="character", trainInd="numeric", mlSpec
formula="formula", data="ExpressionSet", method="learnerSchema", trainInd="integer", ml
    (inherited from: formula="formula", data="ExpressionSet", method="learnerSchema", t
formula="formula", data="ExpressionSet", method="learnerSchema", trainInd="numeric", ml
formula="formula", data="ExpressionSet", method="learnerSchema", trainInd="xvalSpec", m
formula="formula", data="data.frame", method="character", trainInd="numeric", mlSpecial
formula="formula", data="data.frame", method="learnerSchema", trainInd="integer", mlSpe
    (inherited from: formula="formula", data="data.frame", method="learnerSchema", trai
formula="formula", data="data.frame", method="learnerSchema", trainInd="numeric", mlSpe
formula="formula", data="data.frame", method="learnerSchema", trainInd="xvalSpec", mlSp
```

4.2 Available learnerSchema instances

```
> grep(".*I($|\\.)", ls("package:MLInterfaces"), value = TRUE)
```

```
 [1] "RABI"           "adaI"           "baggingI"       "dldaI"
 [5] "glmI.logistic"  "knn.cvI"        "knnI"           "ksvmI"
 [9] "ldaI"           "ldaI.predParms" "lvqI"           "naiveBayesI"
[13] "nnetI"          "qdaI"           "randomForestI"  "rdaI"
[17] "rdacvI"         "rpartI"         "sldaI"          "svmI"
```

You can add your own schemata for new learning functions. See the vignette MLint_devel.

4.3 Tuning

You can set additional parameters in the ... argument place to MLearn. Eventually a tuningSpec object will be defined to control this.

5 Supervised learning: Additional illustrations

5.1 CART

Decision trees are attractive models for certain investigations. If the process under study has a hierarchical structure, so that some features decompose the population at a high level, and others operate within lower level components, a tree-structured model may be useful. Classification And Regression Trees (CART) denotes a family of algorithms that aggressively sift through features in a recursive series of splits of the data. At the first stage, all the data live in a root tree node. All features are dichotomized in all possible ways and the node is split using the feature that leads to two nodes that are most pure in distribution of the class label according to some user-selected metric such as the Gini index or the deviance. The process recurses in the new nodes. The tree construction proceeds until nodes reach some minimal size, and then it may be pruned back. Details can be found in Ripley, Pattern Recognition and Neural Networks, 1995.

```
> rp1 = MLearn(mol.biol ~ ., ffbALL, rpartI, xvalSpec("LOG", 5,
+     balKfold.xvspec(5)))
```

```
> confuMat(rp1)
```

```
         predicted
given     NEG BCR/ABL
  NEG      30      12
  BCR/ABL   5      32
```

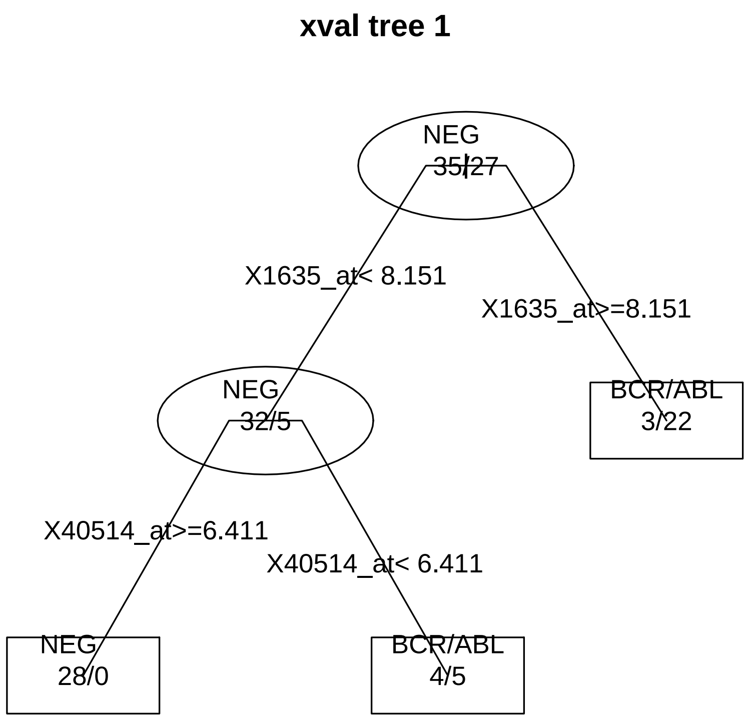
Each fit yields an extensive summary.

```
> summary(RObject(RObject(rp1)[[1]]$mlans))
```

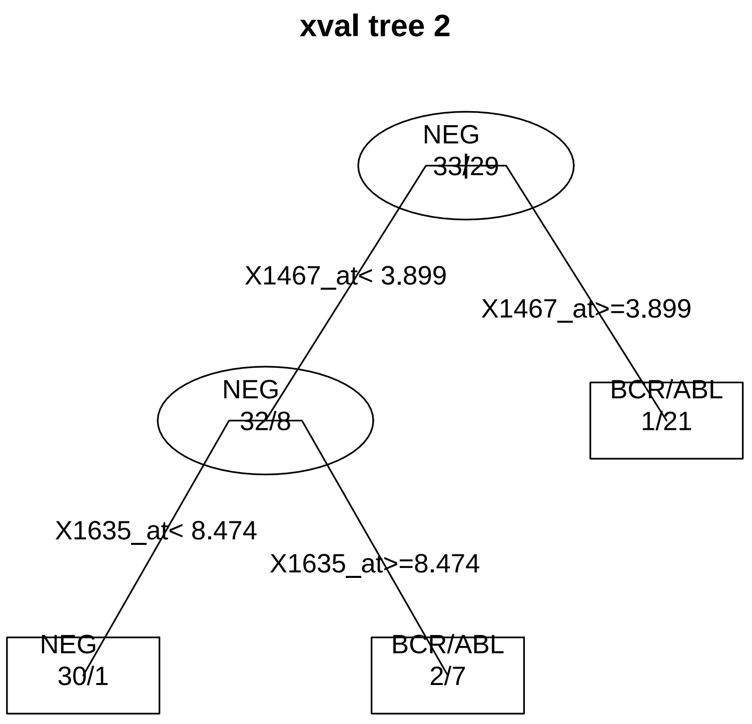
```
Call:
lfun(formula = formula, data = trdata)
  n= 62

          CP nsplit rel error    xerror      xstd
1 0.70370370      0 1.0000000 1.0000000 0.1445960
2 0.03703704      1 0.2962963 0.5925926 0.1276084
3 0.01000000      2 0.2592593 0.6666667 0.1323741

Node number 1: 62 observations,    complexity param=0.7037037
  predicted class=NEG      expected loss=0.4354839
    class counts:    35    27
   probabilities: 0.565 0.435
  left son=2 (37 obs) right son=3 (25 obs)
  Primary splits:
      X1635_at  < 8.150509 to the left,  improve=16.55522, (0 missing)
      X40202_at < 8.95228  to the left,  improve=15.44637, (0 missing)
      X1467_at  < 3.920108 to the left,  improve=13.98677, (0 missing)
      X37015_at < 4.427861 to the left,  improve=13.14044, (0 missing)
      X36591_at < 8.994071 to the left,  improve=12.14615, (0 missing)
  Surrogate splits:
      X41138_at < 11.11763 to the left,  agree=0.806, adj=0.52, (0 split)
      X37351_at < 6.812907 to the left,  agree=0.806, adj=0.52, (0 split)
      X1674_at  < 5.811727 to the left,  agree=0.806, adj=0.52, (0 split)
      X37015_at < 4.427861 to the left,  agree=0.790, adj=0.48, (0 split)
    X32542_at < 8.25133  to the left,  agree=0.790, adj=0.48, (0 split)

Node number 2: 37 observations,    complexity param=0.03703704
  predicted class=NEG      expected loss=0.1351351
    class counts:    32     5
   probabilities: 0.865 0.135 
  left son=4 (28 obs) right son=5 (9 obs)
  Primary splits:
      X40514_at  < 6.410764 to the right, improve=4.204204, (0 missing)
      X1581_s_at < 5.273067 to the right, improve=4.204204, (0 missing)
      X1161_at   < 9.433677 to the right, improve=3.648649, (0 missing)
      X37283_at  < 5.556465 to the right, improve=3.648649, (0 missing)
      X1403_s_at < 8.043665 to the left,  improve=3.648649, (0 missing)
  Surrogate splits:
      X1467_at   < 3.998326 to the left,  agree=0.919, adj=0.667, (0 split)
      X37747_at  < 5.928225 to the left,  agree=0.892, adj=0.556, (0 split)
      X1161_at   < 9.215001 to the right, agree=0.892, adj=0.556, (0 split)
      X1984_s_at < 7.415567 to the right, agree=0.892, adj=0.556, (0 split)
      X37506_at  < 5.215513 to the right, agree=0.892, adj=0.556, (0 split)

Node number 3: 25 observations
  predicted class=BCR/ABL  expected loss=0.12
    class counts:     3    22
   probabilities: 0.120 0.880 

Node number 4: 28 observations
  predicted class=NEG      expected loss=0
    class counts:    28     0
   probabilities: 1.000 0.000 

Node number 5: 9 observations
  predicted class=BCR/ABL  expected loss=0.4444444
    class counts:     4     5
   probabilities: 0.444 0.556 
```

We have cross-validated and can inspect the tree for each iteration:

```
> tr1 = RObject(RObject(rp1)[[1]]$mlans)
> plot(tr1, uniform = TRUE, branch = 0.2, compress = TRUE, margin = 0.1,
+     main = paste("xval tree", 1))
> text(tr1, all = TRUE, use.n = TRUE, fancy = TRUE, pretty = TRUE)
```



```
> tr2 = RObject(RObject(rp1)[[2]]$mlans)
> plot(tr2, uniform = TRUE, branch = 0.2, compress = TRUE, margin = 0.1,
+     main = paste("xval tree", 2))
> text(tr2, all = TRUE, use.n = TRUE, fancy = TRUE, pretty = TRUE)
```



Exercises

- The trees visualized here are not as informative as they would be if gene symbols were used for probe sets. Alter ffbALL so that the featureNames are symbols and generate a better plot.

  You could use code like

  ```
  > X = featureNames(ffbALL)
  > library(hgu95av2.db)
  > SX = mget(X, hgu95av2SYMBOL)
  > any(duplicated(unlist(SX)))
  > featureNames(ffbALL) = SSX
  > rrp1 = MLearn(mol.biol ~ ., ffbALL, rpartI, xvalSpec("LOG", 5,
  +     balKfold.xvspec(5)))
  ```

- Create a deeper set of trees by specifying an option defined in rpart.control. For example, set minsplit=3. Use `plotcp` on the resulting tree objects and interpret.

5.2 Random forests and variable importance assessment

Leo Breiman extended the tree structured modeling approach by integrating random feature and case selection over a long sequence of tree fits. Voting over the tree sequence is used to create the classifier.

According to wikipedia, "Each tree is constructed using the following algorithm:

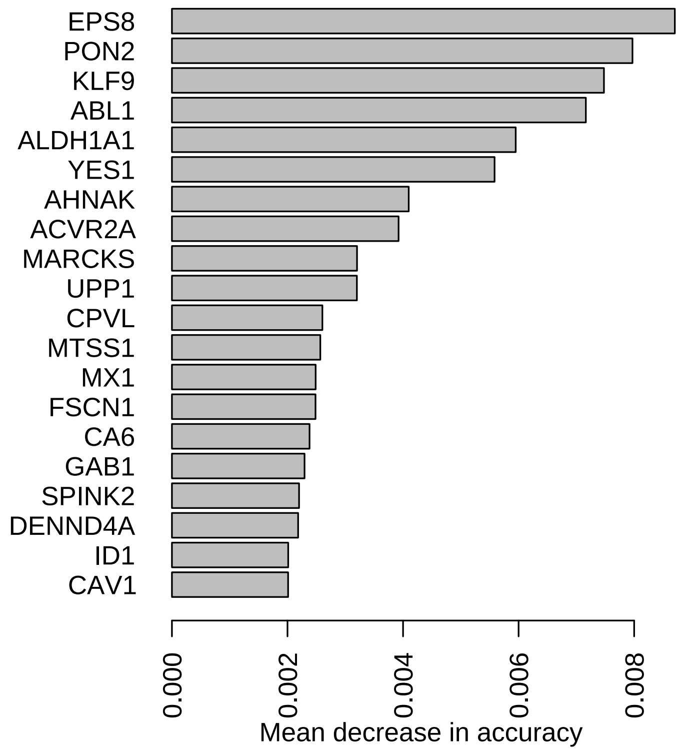
- Let the number of training cases be N, and the number of variables in the classifier be M.
- We are told the number m of input variables to be used to determine the decision at a node of the tree; m should be much less than M.
- Choose a training set for this tree by choosing N times with replacement from all N available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
- For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.
- Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

This is easy to use with MLearn. Because of the internal resampling, we do not need to cross-validate (unless we really want to).

```
> set.seed(12345)

> rf1 = MLearn(mol.biol ~ ., ffbALL, randomForestI, xvalSpec("NOTEST"),
+     importance = TRUE)

> confuMat(rf1, "train")

         predicted
given     NEG BCR/ABL
  NEG      42       0
  BCR/ABL   0      37
```

The variable importance measure can be rendered as follows:

```
> par(las = 2, mar = c(5, 9, 5, 5))
> plot(getVarImp(rf1, TRUE), plat = "hgu95av2", toktype = "SYMBOL")
```



Exercises

- Generate a textual report on the relative importance measures, using `getVarImp`.
- Compare the top genes identified via randomForest to those identified via limma. Comment on the added learning provided by the machine learning algorithm.

5.3 Regularized discriminant analysis

This is an interesting algorithm in which LDA is generalized in two directions. First, the covariance matrix of the data is modeled as a linear combination of an identity matrix and the sample covariance matrix. Second, features are dropped according to their distance from sample centroids. See the 2007 Biostatistics paper of Guo and Tibshirani for details.

The code distributed by Guo et al has internal cross-validation by which parameters $\alpha \in [0, 1]$ (weight on the sample covariance matrix, to which $1 - \alpha \times I$ is added to get the effective covariance to be used in discriminant computation) and δ (parameter dictating how features are dropped depending on their distance from data centroids) are selected. We therefore use RDA in two stages, first to inspect the results of internal cross-validation, and then to compute the predictors for the 'optimal' α and δ. (There are typically a range of attractive values for these parameters.)

```
> set.seed(12345)
```

```
> rda1 = MLearn(mol.biol ~ ., ffbALL, rdacvI, xvalSpec("NOTEST"))
```

We now retrieve the summary of cross-validation results:

```
> attr(RObject(rda1), "xvalAns")
```

```
Call:
rda.cv(fit = run1, x = x, y = resp)
$nonzero
      delta
alpha    0 0.333 0.667   1 1.333 1.667   2 2.333 2.667   3
  0    901   130    12   0     0     0   0     0     0   0
  0.11 901    30     0   0     0     0   0     0     0   0
  0.22 901    35     1   0     0     0   0     0     0   0
  0.33 901    53     2   0     0     0   0     0     0   0
  0.44 901    83     6   0     0     0   0     0     0   0
  0.55 901   139    16   2     0     0   0     0     0   0
  0.66 901   240    46   8     1     0   0     0     0   0
  0.77 901   395   132  43    13     4   1     0     0   0
  0.88 901   614   380 220   120    66  38    20    10   5
  0.99 901   880   855 831   806   779 755   732   707 685

$cv.err
      delta
alpha   0 0.333 0.667  1 1.333 1.667  2 2.333 2.667  3
  0    16    16    18 37    37    37 37    37    37 37
  0.11  9     9    37 37    37    37 37    37    37 37
  0.22 11     9    36 37    37    37 37    37    37 37
  0.33 13    10    29 37    37    37 37    37    37 37
  0.44 14    11    19 37    37    37 37    37    37 37
  0.55 14    14    13 32    37    37 37    37    37 37
  0.66 14    13    12 17    31    37 37    37    37 37
  0.77 14    13    13 12    15    22 32    36    37 37
  0.88 14    14    13 13    13    12 11    14    17 22
  0.99 14    14    14 14    14    14 14    14    14 14
```

We see that values of α around .11 and δ around .33 lead to small numbers of errors in cross-validation. Thus:

```
> rda2 = MLearn(mol.biol ~ ., ffbALL, rdaI, xvalSpec("NOTEST"),
+     alpha = 0.11, delta = 0.333)
> confuMat(rda2, "train")

         predicted
given      BCR/ABL NEG
  NEG            1  41
  BCR/ABL       35   2
```

Exercises

- There is a problem with the rendering of the confusion matrix above. Describe how to avoid it.
- Obtain the list of 'retained genes' by inspecting the rda2 object. Compare to limma top table.

5.4 Support vector machine

An important variation on LDA is the support vector machine (SVM) algorithm. The basic ideas can be gleaned from a paper by Kristin Bennett to be distributed at the course.

```
> svm1 = MLearn(mol.biol ~ ., ffbALL, svmI, xvalSpec("LOG", 5,
+     balKfold.xvspec(5)), kernel = "linear")
> confuMat(svm1)

         predicted
given      NEG BCR/ABL
  NEG       35       7
  BCR/ABL    8      29
```

Exercises

- Examine the `tune` infrastructure of package e1071 and consider whether a selection of parameters for tuning the svm can improve performance in this case. To use tune you will have to extract response and predictor data and convert to appropriate formats.

5.5 Boosting

Random forests uses an ensemble of trees to generate predictions; boosting uses very simple trees generated along a sequence of data reweighting steps. At each iteration, data that are easy to classify are downweighted. We use the implementation in adaboost package; there are others.

This seems to be a very intensive algorithm and we run it for a very short while for feasibility. You may examine the effects of reducing the feature set.

```
> ada1 = MLearn(mol.biol ~ ., ffbALL, adaI, 1:40, type = "discrete",
+     iter = 20)
> confuMat(ada1)
```

6 Unsupervised learning in brief

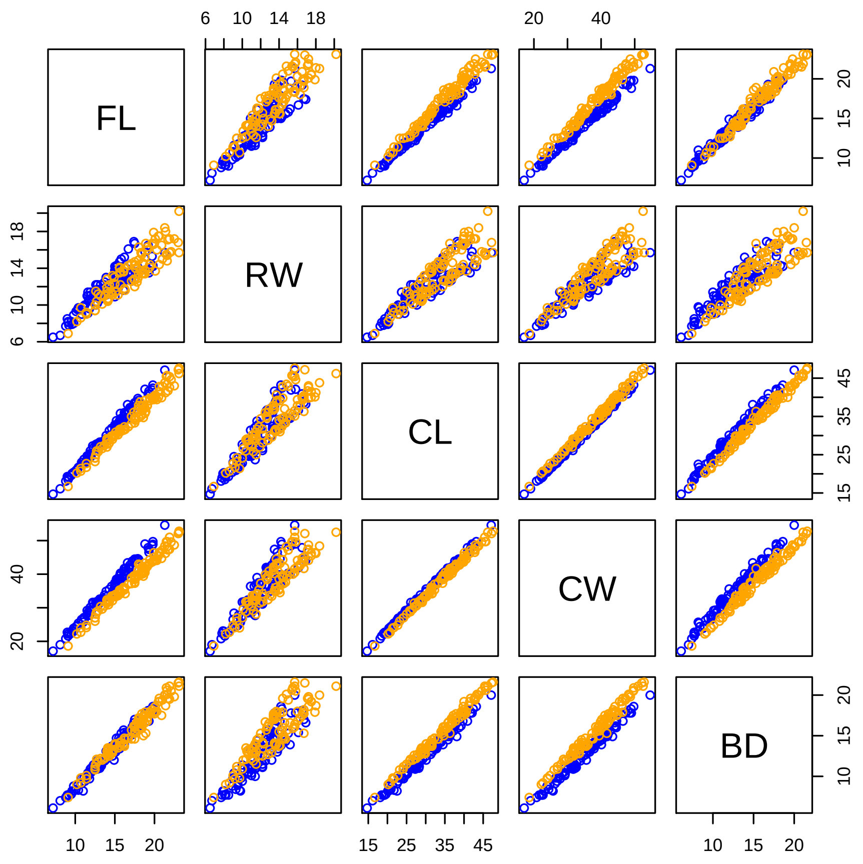
Unsupervised learning occurs in the absence of class labels. In essence, one is trying to learn both the class labels and the rules for attaching them to objects yet unseen.

Cluster analysis is widely used, is propagated through heatmap displays, and has substantial conceptual defects. We will not discuss it unless there is extra time.

6.1 PCA and biplots

To discuss principal components and dimension reduction, we use a simpler data set: the crabs data set in the MASS package. Consider the following display:

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



Various crab body measurements are plotted against each other. Clearly there are high correlations between certain variables and it would be useful to focus on measures that are independently informative on the relationship between crab size and species. Such independently informative measures can be constructed using linear combinations of the raw measurements.

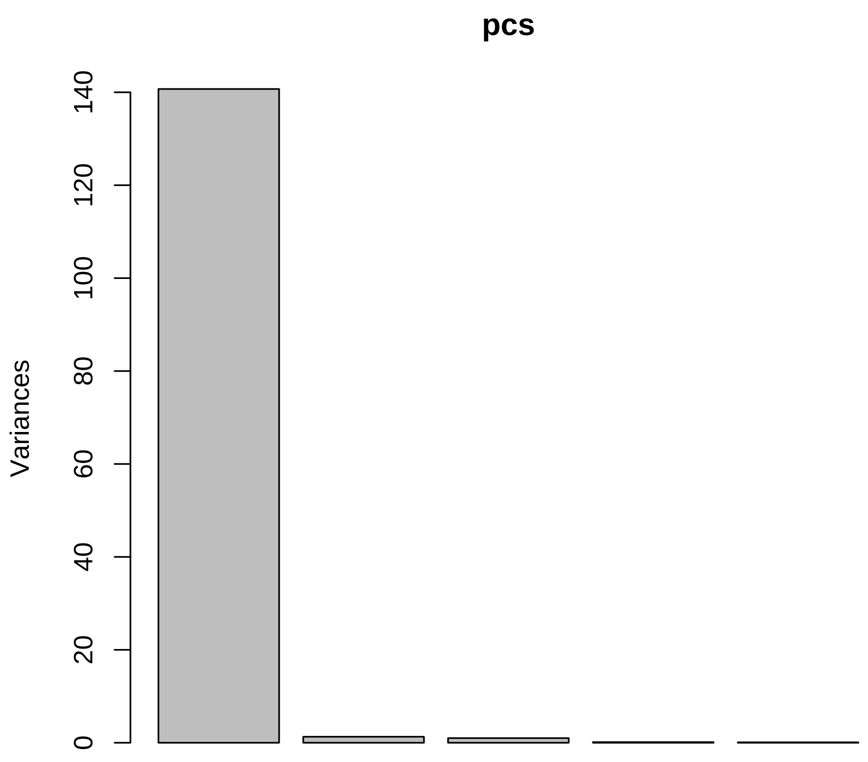
6.1.1 PCA defined and illustrated

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 X having maximal variance, PC2 is the variance-maximizing linear combination of residuals of X after projecting on 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, scatterplots of the first few principal components will depict this.

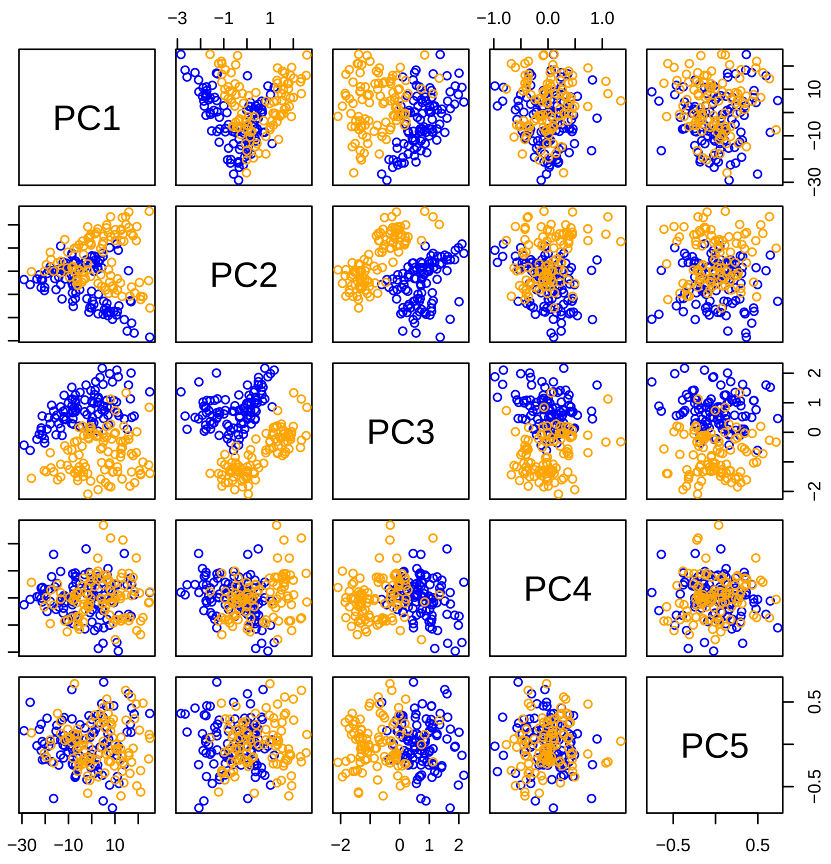
Formally, we can compute the PC using the singular value decomposition of X, in which $X = UDV^t$, where $U_{n\times p}$ and $V_{p\times p}$ are orthonormal, and D is a diagonal matrix of

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 $D_{ii} = \sqrt{n-1}$SD PCi.

```
> library(MASS)
> data(crabs)
> pcs = prcomp(crabs[, -c(1:3)])

> plot(pcs)
```



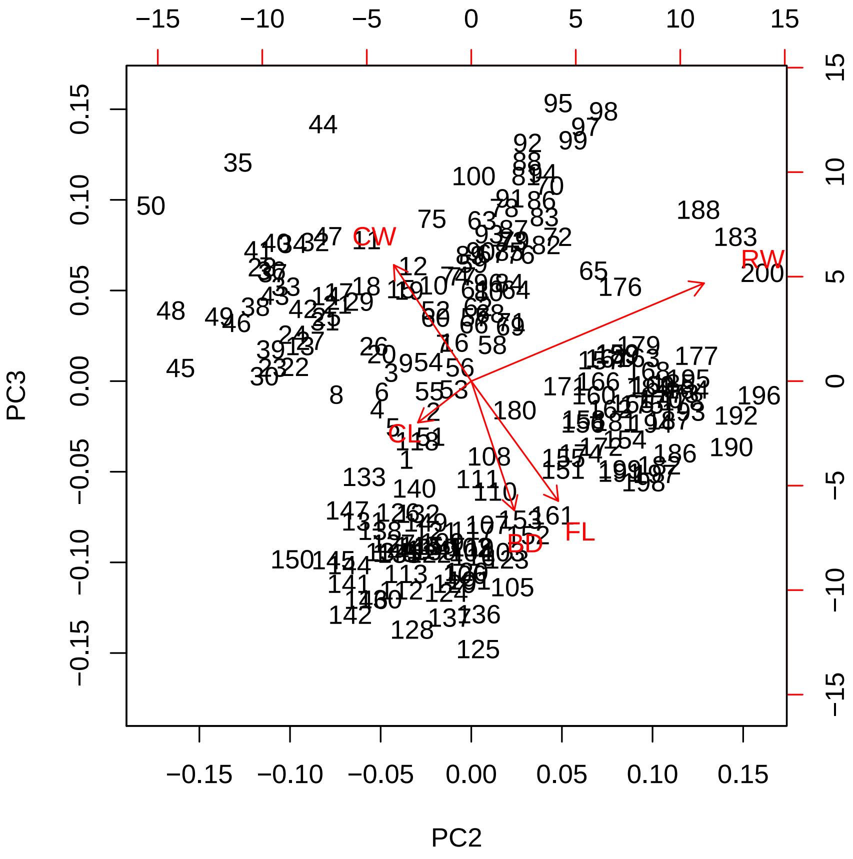
```
> pairs(pcs$x, col = ifelse(crabs$sp == "B", "blue", "orange"))
```



6.1.2 Biplots: superimposing samples and variables after dimension reduction

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

```
> biplot(pcs, choices = 2:3)
```



In addition to the sample representation in PC1-PC2, we have a representation of the original variables and their roles in defining the principal components transformation. The right-hand and top axes measure the first and second components of the eigenvectors corresponding to the original variables.

A formal definition of the biplot procedure is given in Venables and Ripley, and is worth reviewing. Rows of X are observations and columns are variables. A rank-2 approximation to X is obtained via singular value decomposition, setting all but the largest two singular values to zero. Now

$$X \approx [u_1 u_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} v_1^t \\ v_2^t \end{bmatrix} = GH^t$$

and various approaches can be entertained for absorption of the eigenvalues λ_i into G and H. A two-parameter system for doing this is

$$G = n^{\alpha/2}[u_1 u_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}^{1-\theta}, \quad H = n^{\alpha/2}[v_1 v_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}^{\theta}.$$

The "principal component biplot" sets $\alpha = \theta = 1$, and consists in plotting rows of G and H with distinguished symbols. Euclidean distances between rows of G represent

Mahalanobis distances between observations; inner products betwen rows of H represent covariances between variables.

Exercise

Create a biplot using the gene expression data from ALL, focusing on genes found to be predictive by randomForests. Interpret.