MLInterfaces: towards uniform behavior of machine learning tools in R

VJ Carey, J Mar, R Gentleman

August 1, 2007

1 Introduction

We define machine learning methods as data based algorithms for prediction. Given data D, a generic machine learning procedure MLP produces a function ML = MLP(D). For data D' with structure comparable to D, ML(D') is a set of predictions about elements of D'.

To be slightly more precise, a dataset D is a set of records. Each record has the same structure, consisting of a set of features (predictors) and one or more predictands (classes or responses of interest, to be predicted). MLP uses features, predictands, and tuning parameter settings to construct the function ML. ML is to be a function from features only to predictands.

There are many packages and functions in R that provide machine learning procedures. They conform to the abstract setup described above, but with great diversity in the details of implementation and use. The input requirements and the output objects differ from procedure to procedure.

Our objective in *MLInterfaces* is to simplify the use and evaluation of machine learning methods by providing specifications and implementations for a uniform interface. (The `tune` procedures in *e1071* also pursue more uniform interface to machine learning procedures.) At present, we want to simplify use of machine learning with microarray data, assumed to take the form of `ExpressionSet`s. The present implementation addresses the following concerns:

- simplify the selection of the predictand from `ExpressionSet` structure;
- simplify (in fact, require) decomposition of input data into training and test set, with output emphasizing test set results;
- provide a uniform output structure.

The output structures currently supported are subclasses of a general class *MLOutput*, described in Section 3.1 below.

To give a flavor of the current implementation, we perform a few runs with different machine learning tools. We will use 60 genes drawn arbitrarily from Golub's data.

```
> library(MLInterfaces)
> library(golubEsets)

> data(Golub_Merge)
> smallG <- Golub_Merge[200:259, ]
> smallG

ExpressionSet (storageMode: lockedEnvironment)
assayData: 60 features, 72 samples
  element names: exprs
phenoData
  sampleNames: 39, 40, ..., 33 (72 total)
  varLabels and varMetadata:
    Samples: Sample index
    ALL.AML: Factor, indicating ALL or AML
    ...: ...
    Source: Source of sample
    (11 total)
featureData
  rowNames: D13627_at, D13628_at, ..., D16469_at (60 total)
  varLabels and varMetadata: none
experimentData: use 'experimentData(object)'
  pubMedIds: 10521349
Annotation [1] "hu6800"
```

Here is how k-nearest neighbors is used to get predictions of ALL status, using the first 40 records as the training set:

```
> krun <- knnB(smallG, "ALL.AML", trainInd = 1:40)
> krun

MLOutput instance, method= knn
Call:
 knnB(exprObj = smallG, classifLab = "ALL.AML", trainInd = 1:40)
predicted class distribution:
ALL AML
 22  10
summary of class assignment quality scores:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

The `confuMat` method computes the confusion matrix resulting from applying the trained model to the reserved test data:

```
> confuMat(krun)

     predicted
given ALL AML
  ALL  18   3
  AML   4   7
```

Additional parameters can be supplied as accepted by the target procedure in package *class*. To use a neural net in the same context (with fewer genes to simplify the summary below)

```
> set.seed(1234)
> nns <- nnetB(smallG[1:10, ], "ALL.AML", trainInd = 1:40, size = 2,
+     decay = 0.01, maxit = 250)

# weights:  25
initial  value 27.327352
iter  10 value 25.018790
iter  20 value 21.799662
iter  30 value 19.267699
iter  40 value 14.859899
iter  50 value 10.836288
iter  60 value 8.196922
iter  70 value 8.182311
iter  80 value 7.381528
iter  90 value 7.291806
iter 100 value 7.204848
iter 110 value 7.185043
iter 120 value 7.159077
iter 130 value 6.803327
iter 140 value 6.779275
iter 150 value 6.778553
iter 160 value 6.778405
iter 170 value 6.778377
iter 180 value 6.778356
iter 190 value 6.778342
iter 190 value 6.778342
iter 190 value 6.778342
final  value 6.778342
converged
```

```
> nns

MLOutput instance, method= nnet
a 10-2-1 network with 25 weights
inputs: D13627_at D13628_at D13630_at D13633_at D13634_at D13635_at D13636_at D13637_at
output(s): sampLab
options were - entropy fitting  decay=0.01
Call:
 nnetB(exprObj = smallG[1:10, ], classifLab = "ALL.AML", trainInd = 1:40,
    size = 2, decay = 0.01, maxit = 250)
predicted class distribution:
ALL AML
 24   8
summary of class membership probabilities:
           [,1]
Min.    0.04531
1st Qu. 0.04531
Median  0.04531
Mean    0.26910
3rd Qu. 0.19470
Max.    0.99850

> confuMat(nns)

     predicted
given ALL AML
  ALL  18   3
  AML   6   5
```

2 Usage

The basic call sequence for supervised learning for `ExpressionSet`s is

```
methB(eset, tag, trainInd, ...)
```

The parameter `tag` is the name of the `phenoData` element to be used as predictand. Parameter `trainInd` is an integer sequence isolating the samples to be used for training. For unsupervised learning,

```
methB(eset, k, height, ...)
```

The idea here is that one may specify a number `k` of clusters, or a height of a clustering tree that will be cut to form clusters from the `eset` samples. Note that there is no training/test dichotomy for clustering at this stage.

The `RObject` method will access the fit object from the basic procedure. Thus, returning to the `nnetB` invocation above, we have

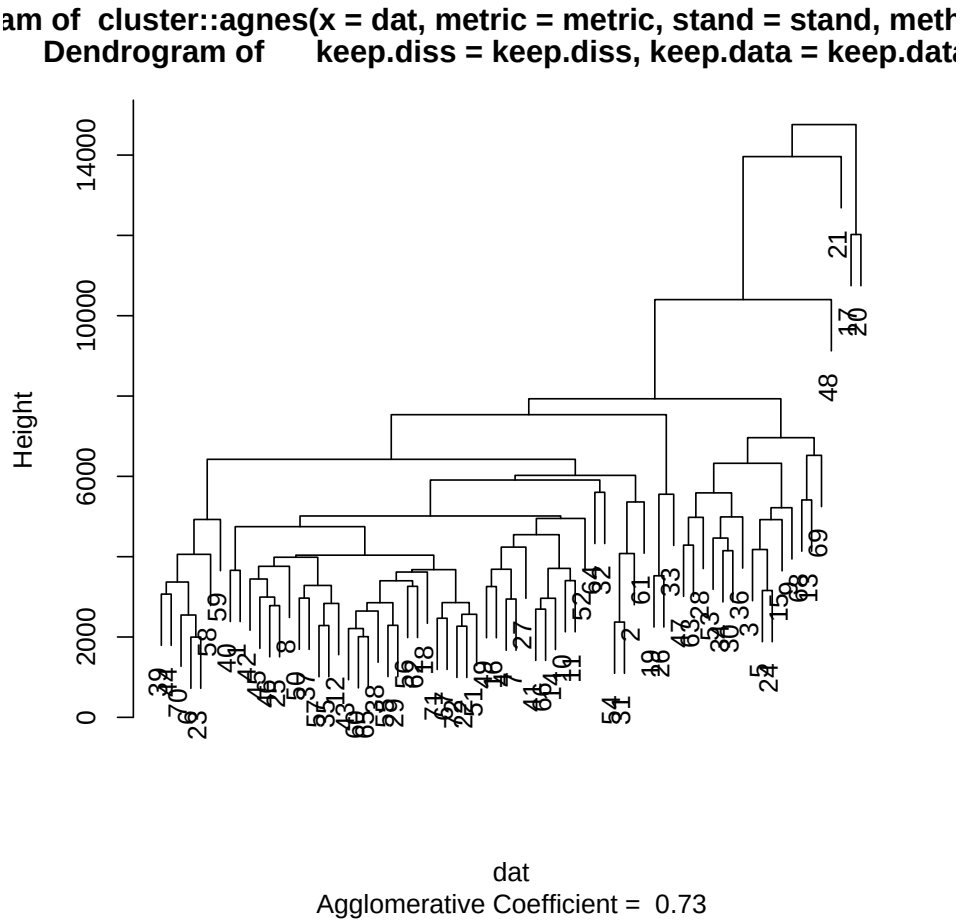
```
> summary(RObject(nns))

a 10-2-1 network with 25 weights
options were - entropy fitting  decay=0.01
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1
   0.00    0.02   -0.17    0.07    0.03   -0.02   -0.05   -0.06   -0.07    0.01
i10->h1
   0.00
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
   0.00   -0.10    0.12   -0.10    0.16    0.08    0.21    0.00   -0.10    0.00
i10->h2
   0.03
 b->o h1->o h2->o
 0.59 -3.64  5.94
```

which is the customary `nnet` summary. This also gives access to visualization.

```
> ags <- agnesB(smallG, k = 4, height = 0, stand = FALSE)

> plot(RObject(ags), which.plot = 2)
```



3 Classes

The S4 class structure is based on a few observations. First, there are two basic types of task covered in machine learning, 'supervised' (MLP uses known classes and ML returns predictand instances) and 'unsupervised' (MLP groups data based on features, no predictand data assumed; ML can tell which group D' is closest to). Second, there is an important concept of 'quality' of prediction or clustering events, and it will be important to allow flexible representation of different approaches to quality measurement by machine learning procedures. Third, there are various things that one always wants access to, regardless of the underlying MLP (call information, R object constituting the 'fit' to the training data). This leads to general classes `MLOutput`, which collects the most general information of interest, `MLLabel`, which represents predictand or clustergroup information, and `MLScore`, which represents quality information. Classes `classifOutput` and `clustOutput` manage the results of supervised and unsupervised learning respectively.

3.1 MLOutput

Extended by all machine learning output containers.

```
> getClass("MLOutput")

Slots:

Name:     method   RObject      call   distMat
Class: character       ANY      call      dist

Known Subclasses: "classifOutput", "clustOutput"
```

3.1.1 MLLabel

Identifies the results of machine learning labeling events, either in the form of class labels or integer cluster indices.

```
> getClass("MLLabel")

Virtual Class

No Slots, prototype of class "S4"

Known Subclasses: "predClass", "groupIndex"
```

3.1.2 MLScore

Identifies quality information about classification or clustering events. This can range from a scalar (agglomeration coefficient, not yet used) to a vector (vote proportions in knn) to a matrix (posterior probabilities of class assignments) to an array (pamr thresholded posteriors).

```
> getClass("MLScore")

Virtual Class

No Slots, prototype of class "S4"

Known Subclasses: "probMat", "probArray", "membMat", "qualScore", "silhouetteVec"
```

3.2 classifOutput

Container for classification results.

```
> getClass("classifOutput")

Slots:

Name:    predLabels   predScores predLabelsTr    trainInds     allClass
Class:      MLLabel      MLScore      MLLabel      integer    character

Name:        method      RObject         call      distMat
Class:    character          ANY         call         dist

Extends: "MLOutput"
```

3.3 clustOutput

Container for clustering results.

```
> getClass("clustOutput")

Slots:

Name:  clustIndices  clustScores       method      RObject         call
Class:      MLLabel      MLScore    character          ANY         call

Name:       distMat
Class:         dist

Extends: "MLOutput"
```

4 Cross-validation

The prevalent interfaces make it difficult to rely on resubstitution to evaluate performance: the [f]B interfaces demand that the user supply an index set for a training data subset, and `confuMat` computes the confusion matrix on the complementary test subset.

Calls to [f]B interfaces can be wrapped to facilitate cross-validation. Several illustrations follow.

- straight leave-one-out (LOO) – note the group parameter must be integer; it is irrelevant for the LOO method.

```
> library(golubEsets)
> data(Golub_Merge)
> smallG <- Golub_Merge[200:250, ]
> lk1 <- xval(smallG, "ALL.AML", knnB, xvalMethod = "LOO", group = as.integer(0)
> table(lk1, smallG$ALL.AML)

lk1   ALL AML
  ALL  37  10
  AML  10  15
```

- Now do an 8-fold cross-validation using the sequence of records to define groups:

```
> lk2 <- xval(smallG, "ALL.AML", knnB, xvalMethod = "LOG", group = as.integer(rep
+     each = 9)))
> table(lk2, smallG$ALL.AML)

lk2   ALL AML
  ALL  35  11
  AML  12  14
```

- A helper function `balKfold` allows the user to create partitions that are approximately balanced with respect to class representation.

```
> balKfold

function (K)
function(data, clab, iternum) {
    clabs <- pData(data)[[clab]]
    narr <- nrow(pData(data))
    cnames <- unique(clabs)
    ilist <- list()
    for (i in 1:length(cnames)) ilist[[cnames[i]]] <- which(clabs ==
        cnames[i])
    clens <- lapply(ilist, length)
    nrep <- lapply(clens, function(x) ceiling(x/K))
    grpinds <- list()
    for (i in 1:length(nrep)) grpinds[[i]] <- rep(1:K, nrep[[i]])[1:clens[[i]]]
    (1:narr)[-which(unlist(grpinds) == iternum)]
}

> lk3 <- xval(smallG, "ALL.AML", knnB, xvalMethod = "FUN", 0:0,
+     indFun = balKfold(5), niter = 5)
> table(lk3, smallG$ALL.AML)
```

```
lk3   ALL AML
  ALL  30  19
  AML  17   6
```

- A function can be supplied to define allocation of records to partitions. Here we supply a function that implements LOO:

```
> LOO2 <- xval(smallG, "ALL.AML", knnB, "FUN", 0:0, function(x,
+     y, i) {
+     (1:ncol(exprs(x)))[-i]
+ }, niter = 72)
> table(lk1, LOO2)

     LOO2
lk1   ALL AML
  ALL  47   0
  AML   0  25
```

5 Cross validation with feature selection

Stephen Henderson of UC London supplied infrastructure to allow embedding of feature selection in the cross-validation process. See the manual page on xval for more details.

```
> t.fun <- function(data, fac) {
+     require(genefilter)
+     xd <- matrix(as.double(exprs(data)), nrow = nrow(exprs(data)))
+     return(abs(rowttests(xd, pData(data)[[fac]], tstatOnly = FALSE)$statistic))
+ }
> lk3f <- xval(smallG, "ALL.AML", knnB, xvalMethod = "LOO", 0:0,
+     fsFun = t.fun)
> table(lk3f$out, smallG$ALL.AML)

     ALL AML
  ALL  41  12
  AML   6  13
```

6 A sketch of a 'doubt' computation

The `nnet` function returns a structure encoding predicted probabilities of class occupancy. We will use this to enrich the `nnetB` output to include a "doubt" outcome. As written this code will handle a two-class outcome; additional structure emerges with more than two classes and some changes will be needed for such cases.

First we obtain the predicted probabilities (for the test set) and round these for display purposes.

```
> predProb <- round(nns@predScores, 3)
```

We save the true labels and the predicted labels.

```
> truth <- as.character(smallG$ALL.AML[-c(1:40)])
> simpPred <- predict(RObject(nns), newdata = data.frame(t(exprs(smallG)[1:10,
+     -c(1:40)])), type = "class")
```

We create a closure that allows boundaries of class probabilities to be specified for assertion of "doubt":

```
> douClo <- function(pprob) function(lo, hi) pprob > lo & pprob <
+     hi
```

Evaluate the closure on the predicted probabilities, yielding a function of two arguments (lo, hi).

```
> smallDou <- douClo(predProb)
```

Now replace the labels for those predictions that are very close to .5.

```
> douPred <- simpPred
> douPred[smallDou(0.48, 0.52)] <- "doubt"
```

The resulting modified predictions are in the fourth column:

```
> mm <- cbind(predProb, truth, simpPred, douPred)
> mm
```

```
           truth simpPred douPred
7  "0.643" "ALL" "AML"    "AML"
8  "0.045" "ALL" "ALL"    "ALL"
9  "0.045" "ALL" "ALL"    "ALL"
10 "0.045" "ALL" "ALL"    "ALL"
11 "0.045" "ALL" "ALL"    "ALL"
12 "0.94"  "ALL" "AML"    "AML"
13 "0.045" "ALL" "ALL"    "ALL"
14 "0.045" "ALL" "ALL"    "ALL"
15 "0.045" "ALL" "ALL"    "ALL"
16 "0.045" "ALL" "ALL"    "ALL"
17 "0.999" "ALL" "AML"    "AML"
18 "0.045" "ALL" "ALL"    "ALL"
19 "0.045" "ALL" "ALL"    "ALL"
```

```
20 "0.045" "ALL" "ALL"    "ALL"
21 "0.045" "ALL" "ALL"    "ALL"
22 "0.045" "ALL" "ALL"    "ALL"
23 "0.045" "ALL" "ALL"    "ALL"
24 "0.045" "ALL" "ALL"    "ALL"
25 "0.045" "ALL" "ALL"    "ALL"
26 "0.045" "ALL" "ALL"    "ALL"
27 "0.045" "ALL" "ALL"    "ALL"
34 "0.999" "AML" "AML"    "AML"
35 "0.999" "AML" "AML"    "AML"
36 "0.045" "AML" "ALL"    "ALL"
37 "0.045" "AML" "ALL"    "ALL"
38 "0.045" "AML" "ALL"    "ALL"
28 "0.045" "AML" "ALL"    "ALL"
29 "0.045" "AML" "ALL"    "ALL"
30 "0.045" "AML" "ALL"    "ALL"
31 "0.999" "AML" "AML"    "AML"
32 "0.999" "AML" "AML"    "AML"
33 "0.947" "AML" "AML"    "AML"
```

```
> table(mm[, "truth"], mm[, "simpPred"])

      ALL AML
  ALL  18   3
  AML   6   5
```

```
> table(mm[, "truth"], mm[, "douPred"])

      ALL AML
  ALL  18   3
  AML   6   5
```