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 ExpressionSets. 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 ?? 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.

```r
> 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 description:
    Samples: Sample index
    ALL.AML: Factor, indicating ALL or AML
      ...: ...
    Source: Source of sample
      (11 total)
featureData
  featureNames: D13627_at, D13628_at, ..., D16469_at (60 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
  pubMedIds: 10521349
Annotation: hu6800
```

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

```r
> krun = MLearn(ALL.AML ~ ., smallG, knnI(k = 1), 1:40)
> krun

MLInterfaces classification output container
The call was:
MLearn(formula = ALL.AML ~ ., data = smallG, method = knnI(k = 1),
trainInd = 1:40)
Predicted outcome distribution for test set:

<table>
<thead>
<tr>
<th></th>
<th>ALL</th>
<th>AML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>10</td>
</tr>
</tbody>
</table>

Summary of scores on test set (use testScores() method for details):

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The `confuMat` method computes the confusion matrix resulting from applying the trained model to the reserved test data:

```r
> 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)

```r
> set.seed(1234)
> nns <- MLearn(ALL.AML ~ ., smallG[1:10, ], nnetI, 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

MLInterfaces classification output container

The call was:
MLearn(formula = ALL.AML ~ ., data = smallG[1:10, ], method = nnetI,
       trainInd = 1:40, size = 2, decay = 0.01, maxit = 250)

Predicted outcome distribution for test set:

<table>
<thead>
<tr>
<th></th>
<th>ALL</th>
<th>AML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>24</td>
<td>8</td>
</tr>
</tbody>
</table>

Summary of scores on test set (use testScores() method for details):
[1] 0.2690627

> confuMat(nns)

<table>
<thead>
<tr>
<th></th>
<th>ALL</th>
<th>AML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

## 2 Usage

The basic call sequence for supervised learning for ExpressionSets is

MLearn(formula, data, learnerSchema, trainInd, ...)

The parameter `formula` is a standard R formula, with `y ~ x + z` indicating that `x` and `z` are predictors of response `y`. If `data` is a data.frame instance, then the formula has the usual interpretation for R. If `data` is an ExpressionSet instance, then it is assumed that the dependent variable is present in the pData component of phenoData, and the variables on the RHS are found in the exprs component of assayData. If . is used on the RHS, then all features in the exprs component are used as predictors. The `learnerSchema` parameter is bound by instances of the learnerSchema class. Many examples are provided with MLInterfaces, see the page from help(MLearn) for a complete list. Parameter `trainInd` is a numeric sequence isolating the samples to be used for training; it may also be bound by an instance of xvalSpec to define a cross-validation of a learning process (see section ??).

For unsupervised learning, there are a number of methods of the form

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

<table>
<thead>
<tr>
<th></th>
<th>b-&gt;h1</th>
<th>i1-&gt;h1</th>
<th>i2-&gt;h1</th>
<th>i3-&gt;h1</th>
<th>i4-&gt;h1</th>
<th>i5-&gt;h1</th>
<th>i6-&gt;h1</th>
<th>i7-&gt;h1</th>
<th>i8-&gt;h1</th>
<th>i9-&gt;h1</th>
<th>i10-&gt;h1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00</td>
<td>0.02</td>
<td>-0.17</td>
<td>0.07</td>
<td>0.03</td>
<td>-0.02</td>
<td>-0.05</td>
<td>-0.06</td>
<td>-0.07</td>
<td>0.01</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>b-&gt;h2</th>
<th>i1-&gt;h2</th>
<th>i2-&gt;h2</th>
<th>i3-&gt;h2</th>
<th>i4-&gt;h2</th>
<th>i5-&gt;h2</th>
<th>i6-&gt;h2</th>
<th>i7-&gt;h2</th>
<th>i8-&gt;h2</th>
<th>i9-&gt;h2</th>
<th>i10-&gt;h2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00</td>
<td>-0.10</td>
<td>0.12</td>
<td>-0.10</td>
<td>0.16</td>
<td>0.08</td>
<td>0.21</td>
<td>0.00</td>
<td>-0.10</td>
<td>0.00</td>
<td>0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>b-&gt;o</th>
<th>h1-&gt;o</th>
<th>h2-&gt;o</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.59</td>
<td>-3.64</td>
<td>5.94</td>
</tr>
</tbody>
</table>

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)
am of cluster::agnes(x = dat, metric = metric, stand = stand, meth:
Dendrogram of keep.diss = keep.diss, keep.data = keep.dat):

3 Classes

For input to MLearn, to define the procedure to be used, two major classes are defined:
learnerSchema, and xvalSpec.

> getClass("learnerSchema")

Slots:

Name: packageName mlFunName converter
Class: character character function

Known Subclasses: "nonstandardLearnerSchema"

> getClass("xvalSpec")
Slots:

Name: type niter partitionFunc fsFun  
Class: character numeric function function

For output, we have only the classifierOutput class:

> getClass("classifierOutput")

Slots:

Name: testOutcomes testPredictions testScores trainOutcomes  
Class: factor factor ANY factor

Name: trainPredictions trainScores fsHistory ROBject  
Class: factor ANY list ANY

Name: call embeddedCV  
Class: call logical

Some legacy classes for output structures exist but are too complex and will be abandoned after a deprecation cycle. The clustOutput class will be replaced by a clusteringOutput class:

> getClass("clustOutput")

Slots:

Name: clustIndices clustScores method ROBject call  
Class: MLabel MLScore character ANY call

Name: distMat  
Class: dist

Extends: "MLOutput"

4 Cross-validation

Instances of the xvalSpec class are bound to the trainInd parameter of MLearn to perform cross-validation. The constructor xvalSpec can be used in line. It has parameters type (only relevant to select "LOO", for leave-one out), niter (number of partitions to use), partitionFunc (function that returns indices of members of partitions), fsFunc (function that performs feature selection and returns a formula with selected features on right-hand side).
The partitionFunc must take parameters data, clab, iternum. data is the usual data frame to be supplied to the learner function. clab must be the name of a column in data. Values of the variable in that column are balanced across cross-validation partitions. iternum is used to select the partition elements as we iterate through cross validation.

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

```r
> library(golubEsets)
> data(Golub_Merge)
> smallG <- Golub_Merge[200:250, ]
> lk1 <- MLearn(ALL.AML ~ ., smallG, knnI(k = 1, l = 0), xvalSpec("LOO"))
> confuMat(lk1)
```

<table>
<thead>
<tr>
<th>predicted</th>
<th>given</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AML</td>
</tr>
<tr>
<td>ALL</td>
<td>10</td>
</tr>
<tr>
<td>AML</td>
<td>15</td>
</tr>
</tbody>
</table>

- Now do a random 8-fold cross-validation.

```r
> ranpart = function(K, data) {
+   N = nrow(data)
+   cu = as.numeric(cut(1:N, K))
+   sample(cu, size = N, replace = FALSE)
+ }
> ranPartition = function(K) function(data, clab, iternum) {
+   p = ranpart(K, data)
+   which(p == iternum)
+ }
> lkran <- MLearn(ALL.AML ~ ., smallG, knnI(k = 1, l = 0), xvalSpec("LOG", + 8, partitionFunc = ranPartition(8)))
> confuMat(lkran)
```

<table>
<thead>
<tr>
<th>predicted</th>
<th>given</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALL</td>
</tr>
<tr>
<td>ALL</td>
<td>25</td>
</tr>
<tr>
<td>AML</td>
<td>7</td>
</tr>
</tbody>
</table>

- Now do an 8-fold cross-validation with approximate balance among groups with respect to frequency of ALL and AML. The utility function balKfold.xvspec helps for this.
> lk3 <- MLearn(ALL.AML ~ ., smallG, knnI(k = 1, l = 0), xvalSpec("LOG", + 8, partitionFunc = balKfold.xvspec(8)))
> confuMat(lk3)

    predicted
given ALL AML
       ALL  39  8
       AML  8 17

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 MLearn/nnetI 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(testScores(nns), 3)

We save the true labels and the predicted labels.

> truth <- as.character(smallG$ALL.AML[-c(1:40)])
> simpPred <- as.character(testPredictions(nns))
We create a closure that allows boundaries of class probabilities to be specified for assertion of “doubt”:

```r
> 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).

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

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

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

The resulting modified predictions are in the fourth column:

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

<table>
<thead>
<tr>
<th>truth</th>
<th>simpPred</th>
<th>douPred</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>&quot;0.643&quot;</td>
<td>&quot;AML&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>11</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>12</td>
<td>&quot;0.94&quot;</td>
<td>&quot;AML&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>14</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>15</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>16</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>17</td>
<td>&quot;0.999&quot;</td>
<td>&quot;AML&quot;</td>
</tr>
<tr>
<td>18</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>19</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>20</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>21</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>22</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>23</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>24</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>25</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>26</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>27</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>28</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>29</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>30</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>31</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>32</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>33</td>
<td>&quot;0.045&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>34</td>
<td>&quot;0.999&quot;</td>
<td>&quot;AML&quot;</td>
</tr>
<tr>
<td>35</td>
<td>&quot;0.999&quot;</td>
<td>&quot;AML&quot;</td>
</tr>
</tbody>
</table>
> 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