

MLInterfaces: towards uniform behavior of machine learning tools in R

VJ Carey, J Mar, R Gentleman

February 2, 2005

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 `exprSets`. The present implementation addresses the following concerns:

- simplify the selection of the predictand from `exprSet` 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.

Several other concerns will be addressed as the project matures. Among these are:

- generic interfaces for cross-validation (exploiting native resources when available)
- simplified specification of criteria for outlier and doubt predictions
- appropriate visualizations.

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)
```

```
Welcome to Bioconductor
```

```
Vignettes contain introductory material. To view,
  simply type: openVignette()
For details on reading vignettes, see
  the openVignette help page.
```

```
> library(golubEsets)
> data(golubMerge)
> smallG <- golubMerge[200:259, ]
> smallG
```

```
Expression Set (exprSet) with
```

```
60 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
```

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

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

```
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

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)

```
> nns <- nnetB(smallG[1:10, ], "ALL.AML", trainInd = 1:40, size = 4,
+             decay = 0.01)
```

```
# weights: 49
```

```
initial value 26.225598
```

```
iter 10 value 24.733370
```

```
iter 20 value 23.689542
```

```
iter 30 value 19.161957
```

```
iter 40 value 15.249217
```

```
iter 50 value 13.755607
```

```
iter 60 value 11.413292
```

```
iter 70 value 10.982218
```

```
iter 80 value 10.947865
```

```
iter 90 value 10.940764
```

```
iter 100 value 10.297149
```

```
final value 10.297149
```

```
stopped after 100 iterations
```

```
> nns
```

```
MLOutput instance, method= nnet
```

```
a 10-4-1 network with 49 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 = 4, decay = 0.01)
predicted class distribution:
ALL AML
 23    9
summary of class membership probabilities:
      [,1]
Min.    0.0007404
1st Qu. 0.0375600
Median  0.2236000
Mean    0.3514000
3rd Qu. 0.8018000
Max.    0.9993000

```

2 Usage

The basic call sequence for supervised learning for `exprSets` 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 10-4-1 network with 49 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    0.03   -0.02    0.14   -0.01    0.04    0.06   -0.07    0.03    0.01
i10->h1
 -0.04
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
 -0.01   -0.06    0.09    0.04   -0.08    0.05    0.02    0.14    0.06    0.04
i10->h2
 -0.04
  b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3

```

```

0.00    0.13   -0.04   -0.03    0.00    0.01    0.00    0.07    0.07   -0.01
i10->h3
-0.03
b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4
0.01    0.03    0.11    0.01    0.01    0.02    0.00    0.02   -0.01    0.05
i10->h4
0.10
b->o h1->o h2->o h3->o h4->o
0.64 -3.93  5.96 -4.57  0.65

```

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

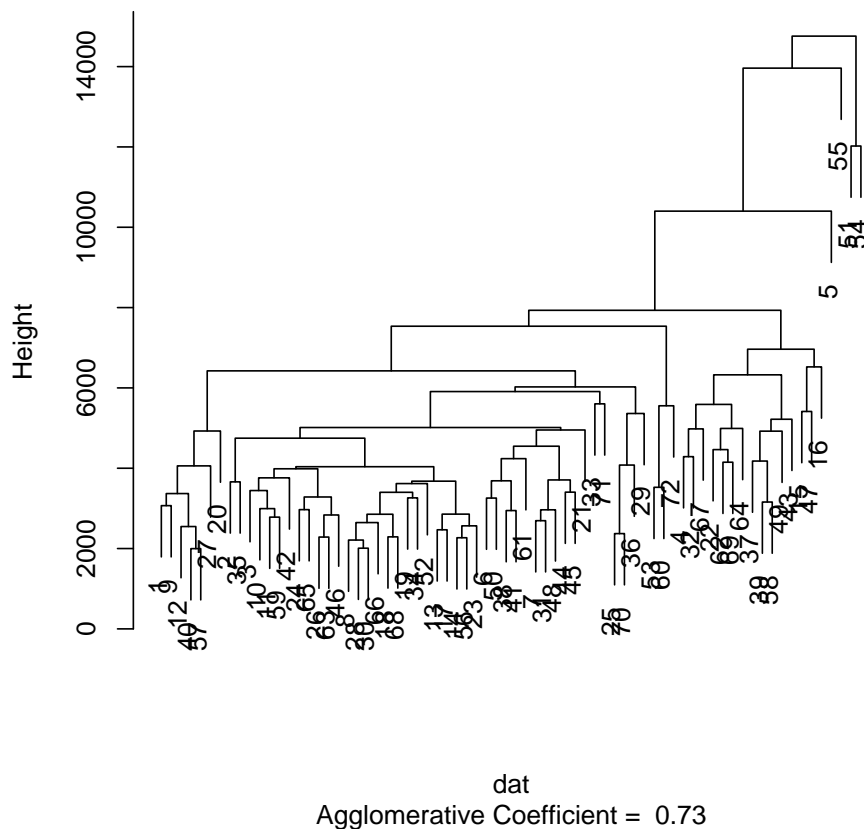
```

> 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

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 cluster group 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 "list"

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 "list"

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

3.2 classifOutput

Container for classification results.

```
> getClass("classifOutput")
```

Slots:

Name:	predLabels	predScores	trainInds	allClass	method	RObject
Class:	MLabel	MLScore	integer	character	character	ANY

Name:	call	distMat
Class:	call	dist

Extends: "MLOutput"

3.3 clustOutput

Container for clustering results.

```
> getClass("clustOutput")
```

Slots:

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

Name:	distMat
Class:	dist

Extends: "MLOutput"