Cubist Models For Regression

Max Kuhn (max.kuhn@pfizer.com)
Steve Weston
Chris Keefer

June 21, 2011

1 Introduction

Cubist is an R port of the Cubist GPL C code released by RuleQuest at

http://rulequest.com/cubist-info.html

See the last section of this document for information on the porting. The other parts describes the functionality of the R package.

2 Model Trees

Cubist is a rule–based model that is an extension of Quinlan's M5 model tree. A tree is grown where the terminal leaves contain linear regression models. These models are based on the predictors used in previous splits. Also, there are intermediate linear models at each step of the tree. A prediction is made using the linear regression model at the terminal node of the tree, but is "smoothed" by taking into account the prediction from the linear model in the previous node of the tree (which also occurs recursively up the tree). The tree is reduced to a set of rules, which initially are paths from the top of the tree to the bottom. Rules are eliminated via pruning and/or combined for simplification.

This is explained better in Quinlan (1992). Wang and Witten (1997) attempted to recreate this model using a "rational reconstruction" of Quinlan (1992) that is the basis for the `M5P` model in Weka (and the R package RWeka).

An example of a model tree can be illustrated using the Boston Housing data in the mlbench package.

```
> library(Cubist)
> library(mlbench)
> data(BostonHousing)
> BostonHousing$chas <- as.numeric(BostonHousing$chas) - 1
> set.seed(1)
> inTrain <- sample(1:nrow(BostonHousing), floor(.8*nrow(BostonHousing)))
> trainingPredictors <- BostonHousing[ inTrain, -14]
> testPredictors     <- BostonHousing[-inTrain, -14]
> trainingOutcome <- BostonHousing$medv[ inTrain]
> testOutcome     <- BostonHousing$medv[-inTrain]
> modelTree <- cubist(x = trainingPredictors, y = trainingOutcome)
> modelTree

Call:
cubist.default(x = trainingPredictors, y = trainingOutcome)

Number of samples: 404
Number of predictors: 13

Number of committees: 1
Number of rules: 4


> summary(modelTree)

Call:
cubist.default(x = trainingPredictors, y = trainingOutcome)


Cubist [Release 2.07 GPL Edition]  Tue Jun 21 13:00:22 2011
---------------------------------

    Target attribute `outcome'

Read 404 cases (14 attributes) from undefined.data

Model:

  Rule 1: [88 cases, mean 13.81, range 5 to 27.5, est err 2.10]

    if
        nox > 0.668
    then
        outcome = 2.07 + 3.14 dis - 0.35 lstat + 18.8 nox + 0.007 b
                  - 0.12 ptratio - 0.008 age - 0.02 crim

  Rule 2: [153 cases, mean 19.54, range 8.1 to 31, est err 2.16]

    if
        nox <= 0.668
        lstat > 9.59
```

```
    then
        outcome = 34.81 - 1 dis - 0.72 ptratio - 0.056 age - 0.19 lstat + 1.5 rm
                  - 0.11 indus + 0.004 b

  Rule 3: [39 cases, mean 24.10, range 11.9 to 50, est err 2.73]

    if
        rm <= 6.23
        lstat <= 9.59
    then
        outcome = 11.89 + 3.69 crim - 1.25 lstat + 3.9 rm - 0.0045 tax
                  - 0.16 ptratio

  Rule 4: [128 cases, mean 31.31, range 16.5 to 50, est err 2.95]

    if
        rm > 6.23
        lstat <= 9.59
    then
        outcome = -1.13 + 1.6 crim - 0.93 lstat + 8.6 rm - 0.0141 tax
                  - 0.83 ptratio - 0.47 dis - 0.019 age - 1.1 nox


Evaluation on training data (404 cases):

    Average  |error|               2.27
    Relative |error|               0.34
    Correlation coefficient        0.94


        Attribute usage:
          Conds  Model

           78%   100%    lstat
           59%    53%    nox
           41%    78%    rm
                 100%    ptratio
                  90%    age
                  90%    dis
                  62%    crim
                  59%    b
                  41%    tax
                  38%    indus


Time: 0.0 secs
```

There is no formula method for `cubist`; the predictors are specified as matrix or data frame and the outcome is a numeric vector.

There is a predict method for the model:

```
> mtPred <- predict(modelTree, testPredictors)
> ## Test set RMSE
> sqrt(mean((mtPred - testOutcome)^2))

[1] 3.337924

> ## Test set R^2
> cor(mtPred, testOutcome)^2

[1] 0.8573504
```

3 Boosting

The Cubist model can also use a boosting–like scheme called *committees* where iterative model trees are created in sequence. The first tree follows the procedure described in the last section. Subsequent trees are created using adjusted versions to the training set outcome: if the model over–predicted a value, the response is adjusted downward for the next model (and so on). Unlike traditional boosting, stage weights for each committee are not used to average the predictions from each model tree; the final prediction is a simple average of the predictions from each model tree.

The `committee` option can be used to control number of model trees:

```
> set.seed(1)
> committeeModel <- cubist(x = trainingPredictors, y = trainingOutcome,
+                          committees = 5)
> summary(committeeModel)

Call:
cubist.default(x = trainingPredictors, y = trainingOutcome, committees = 5)


Cubist [Release 2.07 GPL Edition]  Tue Jun 21 13:00:23 2011
-------------------------------

    Target attribute `outcome'

Read 404 cases (14 attributes) from undefined.data

Model 1:

  Rule 1/1: [88 cases, mean 13.81, range 5 to 27.5, est err 2.10]

    if
```

```
      nox > 0.668
    then
      outcome = 2.07 + 3.14 dis - 0.35 lstat + 18.8 nox + 0.007 b
                - 0.12 ptratio - 0.008 age - 0.02 crim

  Rule 1/2: [153 cases, mean 19.54, range 8.1 to 31, est err 2.16]

    if
      nox <= 0.668
      lstat > 9.59
    then
      outcome = 34.81 - 1 dis - 0.72 ptratio - 0.056 age - 0.19 lstat + 1.5 rm
                - 0.11 indus + 0.004 b

  Rule 1/3: [39 cases, mean 24.10, range 11.9 to 50, est err 2.73]

    if
      rm <= 6.23
      lstat <= 9.59
    then
      outcome = 11.89 + 3.69 crim - 1.25 lstat + 3.9 rm - 0.0045 tax
                - 0.16 ptratio

  Rule 1/4: [128 cases, mean 31.31, range 16.5 to 50, est err 2.95]

    if
      rm > 6.23
      lstat <= 9.59
    then
      outcome = -1.13 + 1.6 crim - 0.93 lstat + 8.6 rm - 0.0141 tax
                - 0.83 ptratio - 0.47 dis - 0.019 age - 1.1 nox

Model 2:

  Rule 2/1: [71 cases, mean 13.41, range 5 to 27.5, est err 2.66]

    if
      crim > 5.69175
      dis > 1.4254
    then
      outcome = 42.13 + 2.45 dis - 0.47 lstat - 0.71 ptratio - 1.8 rm

  Rule 2/2: [84 cases, mean 18.75, range 8.1 to 27.5, est err 2.25]

    if
      crim <= 5.69175
      nox > 0.532
      dis > 1.4254
      tax > 222
      ptratio > 17
    then
```

```
      outcome = 44.08 + 1.19 crim - 0.43 lstat - 1.05 ptratio - 0.011 age

Rule 2/3: [15 cases, mean 23.43, range 5 to 50, est err 5.62]

  if
      dis <= 1.4254
      ptratio > 17
  then
      outcome = 174.86 - 100.95 dis - 1.07 lstat - 0.09 ptratio

Rule 2/4: [77 cases, mean 23.90, range 11.8 to 50, est err 2.37]

  if
      ptratio <= 17
      lstat > 5.12
  then
      outcome = -3.3 + 8.3 rm - 0.0238 tax - 1.66 dis - 0.063 age - 0.1 lstat
                - 0.21 ptratio - 3.8 nox + 0.007 zn

Rule 2/5: [128 cases, mean 25.56, range 14.4 to 50, est err 3.12]

  if
      crim <= 5.69175
      nox <= 0.532
      ptratio > 17
  then
      outcome = -15.58 + 2.43 crim + 7.1 rm - 0.075 age + 0.24 lstat
                - 0.41 dis - 0.16 ptratio

Rule 2/6: [16 cases, mean 27.91, range 15.7 to 39.8, est err 5.25]

  if
      tax <= 222
      lstat > 5.12
  then
      outcome = 274.62 - 12.31 ptratio - 0.212 age - 0.03 lstat

Rule 2/7: [18 cases, mean 30.49, range 22.5 to 50, est err 3.69]

  if
      rm <= 6.861
      lstat <= 5.12
  then
      outcome = -58.03 + 10.96 crim + 13.3 rm - 0.03 lstat - 0.08 dis
                - 0.06 ptratio - 1.1 nox

Rule 2/8: [19 cases, mean 41.54, range 31.2 to 50, est err 3.63]

  if
      rm > 6.861
      age <= 71
```

```
        lstat <= 5.12
    then
        outcome = -56.93 + 14.2 rm - 0.07 lstat - 0.2 dis - 2.6 nox
                  - 0.13 ptratio + 0.006 zn

  Rule 2/9: [14 cases, mean 43.48, range 22.8 to 50, est err 5.55]

    if
        age > 71
        lstat <= 5.12
    then
        outcome = -24.48 + 1.99 crim + 0.467 age + 3.5 rm

Model 3:

  Rule 3/1: [88 cases, mean 13.81, range 5 to 27.5, est err 2.32]

    if
        nox > 0.668
    then
        outcome = -9 + 5.5 dis + 19.4 nox + 0.014 b - 0.12 lstat - 0.16 ptratio
                  - 0.04 crim

  Rule 3/2: [10 cases, mean 17.64, range 11.7 to 27.5, est err 11.68]

    if
        nox <= 0.668
        b <= 179.36
    then
        outcome = -2.07 + 0.149 b + 0.77 lstat

  Rule 3/3: [156 cases, mean 19.68, range 8.1 to 33.8, est err 2.23]

    if
        nox <= 0.668
        lstat > 9.53
    then
        outcome = 28.56 - 1.09 dis - 0.27 lstat - 0.068 age + 2.6 rm
                  - 0.6 ptratio

  Rule 3/4: [164 cases, mean 29.68, range 11.9 to 50, est err 3.44]

    if
        lstat <= 9.53
    then
        outcome = 6.57 + 4.08 crim - 0.75 lstat + 7.6 rm - 0.0301 tax
                  - 0.79 ptratio - 0.15 dis - 2.2 nox + 0.001 b

Model 4:

  Rule 4/1: [335 cases, mean 19.44, range 5 to 50, est err 2.69]
```

```
  if
      rm <= 7.079
      lstat > 5.12
  then
      outcome = 45.08 - 0.4 lstat + 0.27 rad - 0.0124 tax - 0.2 crim
                - 0.6 ptratio - 8.5 nox - 0.36 dis - 0.04 indus

Rule 4/2: [19 cases, mean 20.96, range 5 to 50, est err 6.81]

  if
      rm <= 7.079
      dis <= 1.4261
  then
      outcome = 163.2 - 85.4 dis - 1.21 lstat - 0.15 crim

Rule 4/3: [111 cases, mean 23.01, range 14.4 to 32, est err 1.92]

  if
      nox <= 0.51
      rm <= 7.079
      tax > 193
      lstat > 5.12
  then
      outcome = 9.18 + 12.12 crim + 2.8 rm - 0.031 age - 0.05 lstat + 0.04 rad
                - 0.002 tax - 0.1 ptratio - 0.1 dis - 1.6 nox

Rule 4/4: [9 cases, mean 24.33, range 15.7 to 36.2, est err 7.38]

  if
      rm <= 7.079
      tax <= 193
      lstat > 5.12
  then
      outcome = 22.72

Rule 4/5: [18 cases, mean 30.49, range 22.5 to 50, est err 4.91]

  if
      rm <= 6.861
      lstat <= 5.12
  then
      outcome = 20.95 + 8.16 crim - 0.54 lstat + 0.23 rad + 1.3 rm

Rule 4/6: [35 cases, mean 36.15, range 22.5 to 50, est err 3.61]

  if
      age <= 71
      lstat <= 5.12
  then
      outcome = -67.4 + 15.9 rm - 1.05 rad - 0.005 b - 0.05 lstat
```

```
  Rule 4/7: [43 cases, mean 39.37, range 15 to 50, est err 6.37]

    if
        rm > 7.079
    then
        outcome = -123.73 + 0.308 b + 8.8 rm - 0.45 rad - 1.38 ptratio
                  - 0.04 lstat - 0.0016 tax - 0.1 dis - 1.2 nox - 0.02 indus
                  - 0.01 crim

  Rule 4/8: [14 cases, mean 43.48, range 22.8 to 50, est err 5.14]

    if
        age > 71
        lstat <= 5.12
    then
        outcome = -34.28 + 0.598 age - 0.75 lstat + 6.1 rm - 0.047 b + 0.16 rad

Model 5:

  Rule 5/1: [88 cases, mean 13.81, range 5 to 27.5, est err 2.73]

    if
        nox > 0.668
    then
        outcome = -35.12 + 8.59 dis + 38.7 nox + 0.017 b - 0.04 lstat
                  - 0.07 ptratio + 0.01 rad + 0.1 rm

  Rule 5/2: [156 cases, mean 19.68, range 8.1 to 33.8, est err 2.53]

    if
        nox <= 0.668
        lstat > 9.53
    then
        outcome = 44.88 - 1.48 dis - 0.076 age - 0.28 lstat - 0.8 ptratio
                  + 0.012 b + 0.1 rad + 0.3 rm - 1.6 nox - 0.0007 tax

  Rule 5/3: [189 cases, mean 24.76, range 12.7 to 50, est err 2.41]

    if
        dis > 3.3175
    then
        outcome = -24.62 + 1.13 crim + 10.4 rm - 0.0183 tax - 0.69 dis
                  - 0.19 lstat - 0.043 age - 0.26 ptratio + 0.022 zn

  Rule 5/4: [44 cases, mean 35.04, range 11.9 to 50, est err 6.37]

    if
        dis <= 3.3175
        lstat <= 9.53
    then
```

```
        outcome = 32.74 + 6.34 crim - 0.0468 tax - 0.87 lstat + 5.5 rm
                  - 1.16 ptratio


Evaluation on training data (404 cases):

    Average  |error|               1.91
    Relative |error|               0.29
    Correlation coefficient        0.96


        Attribute usage:
          Conds  Model

           65%    99%    lstat
           46%    56%    nox
           32%    71%    rm
           18%    88%    dis
           13%    95%    ptratio
           12%    65%    crim
            9%    55%    tax
            4%    56%    age
                  36%    b
                  34%    rad
                  23%    indus
                  12%    zn


Time: 0.1 secs
```

For this model:

```
> cmPred <- predict(committeeModel, testPredictors)
> ## RMSE
> sqrt(mean((cmPred - testOutcome)^2))

[1] 2.870122

> ## R^2
> cor(cmPred, testOutcome)^2

[1] 0.8955536
```

4 Instance–Based Corrections

Another innovation in Cubist using nearest–neighbors to adjust the predictions from the rule–based model. First, a model tree (with or without committees) is created. Once a sample is predicted by

this model, Cubist can find it's nearest neighbors and determine the average of these training set points. See Quinlan (1993a) for the details of the adjustment.

The development of rules and committees is independent of the choice of using instances. The original C code allowed the program to choose whether to use instances, not use them or let the program decide. Our approach is to build a model with the `cubist` function that is ignorant to the decision about instances. When samples are predicted, the argument `neighbors` can be used to adjust the rule–based model predictions (or not).

We can add instances to the previously fit committee model:

```
> instancePred <- predict(committeeModel, testPredictors, neighbors = 5)
> ## RMSE
> sqrt(mean((instancePred - testOutcome)^2))

[1] 2.693565

> ## R^2
> cor(instancePred, testOutcome)^2

[1] 0.9111374
```

Note that the previous models used the implicit default of `neighbors = 0` for their predictions.

To tune the model over different values of `neighbors` and `committees`, the `train` function in the caret package can be used to optimize these parameters. For example:

```
> library(caret)
> set.seed(1)
> cTune <- train(x = trainingPredictors, y = trainingOutcome,
+                "cubist",
+                tuneGrid = expand.grid(.committees = c(1, 10, 50, 100),
+                                       .neighbors = c(0, 1, 5, 9)),
+                trControl = trainControl(method = "cv"))

Fitting: committees=1, neighbors=9
Fitting: committees=10, neighbors=9
Fitting: committees=50, neighbors=9
Fitting: committees=100, neighbors=9
Aggregating results
Selecting tuning parameters
Fitting model on full training set

> cTune

404 samples
 13 predictors

Pre-processing: None
```

```
Resampling: Cross-Validation (10 fold)

Summary of sample sizes: 364, 364, 361, 364, 364, 364, ...

Resampling results across tuning parameters:

  committees  neighbors  RMSE  Rsquared  RMSE SD  Rsquared SD
  1           0          4.63  0.757     1.44     0.159
  1           1          4.59  0.753     1.8      0.18
  1           5          4.3   0.78      1.56     0.15
  1           9          4.36  0.776     1.58     0.157
  10          0          3.54  0.838     1.52     0.153
  10          1          3.62  0.835     1.57     0.151
  10          5          3.34  0.854     1.56     0.143
  10          9          3.37  0.851     1.61     0.152
  50          0          3.47  0.845     1.45     0.143
  50          1          3.52  0.844     1.49     0.141
  50          5          3.27  0.859     1.51     0.136
  50          9          3.29  0.857     1.55     0.144
  100         0          3.44  0.848     1.43     0.139
  100         1          3.45  0.848     1.47     0.137
  100         5          3.23  0.863     1.48     0.132
  100         9          3.25  0.86      1.53     0.14

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were committees = 100 and neighbors = 5.
```

Figure 1 shows the profiles of the tuning parameters produced using `plot(cTune)`.

It may also be useful to see how different models fit a single predictor:

```
> lstat <- trainingPredictors[, "lstat", drop = FALSE]
> justRules <- cubist(lstat, trainingOutcome)
> andCommittees <- cubist(lstat, trainingOutcome, committees = 100)
```

Figure 2 shows the model fits for the test data. For these data, there doesn't appear to be much of a improvement when committees or instances are added to the based rules.

5 Variable Importance

The `modelTree` method for Cubist shows the usage of each variable in either the rule conditions or the (terminal) linear model. In actuality, many more linear models are used in prediction that are shown in the output. Because of this, the variable usage statistics shown at the end of the output of the `summary` function will probably be inconsistent with the rules also shown in the output. At each split of the tree, Cubist saves a linear model (after feature selection) that is allowed to have terms for each variable used in the current split or any split above it. Quinlan (1992) discusses a

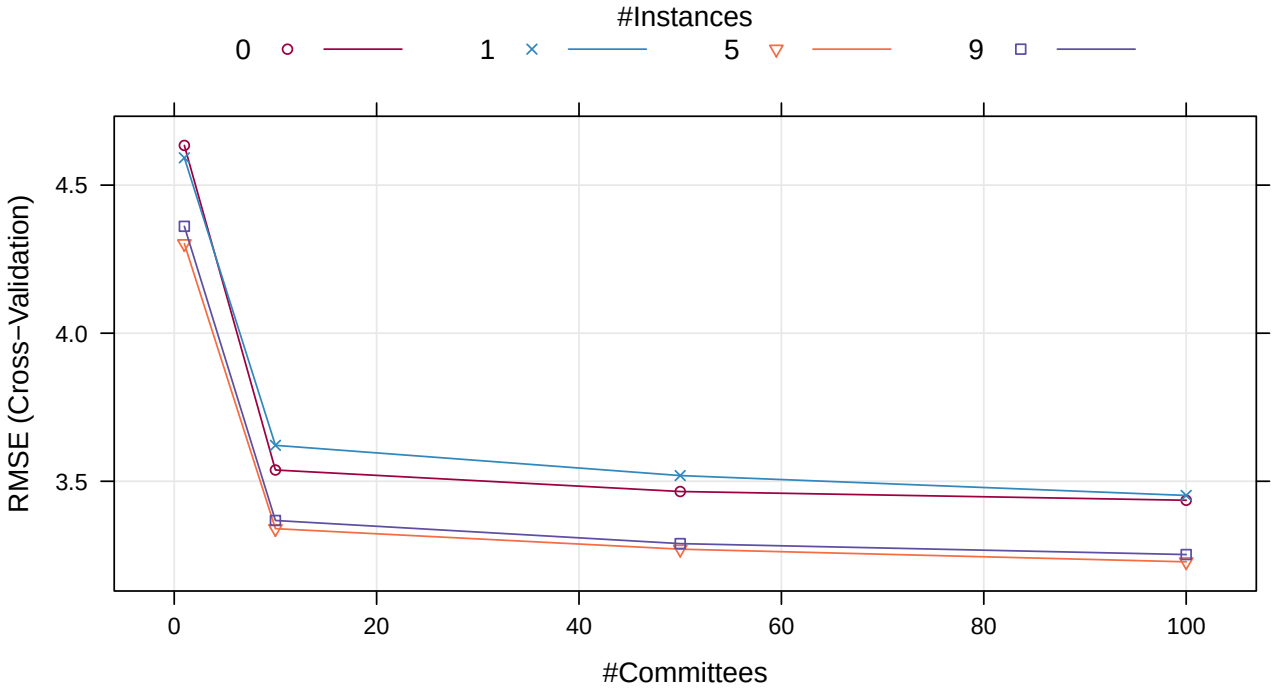

Figure 1: The relationship between performance and the two tuning parameters, as estimated using cross–validation.

smoothing algorithm where each model prediction is a linear combination of the parent and child model along the tree. As such, the final prediction is a function of all the linear models from the initial node to the terminal node. The percentages shown in the Cubist output reflects all the models involved in prediction (as opposed to the terminal models shown in the output).

The raw usage statistics are contained in a data frame called `usage` in the `cubist` object.

The caret package has a general variable importance method `varImp`. When using this function on a `cubist` argument, the variable importance is a linear combination of the usage in the rule conditions and the model.

For example:

```
> summary(modelTree)

Call:
cubist.default(x = trainingPredictors, y = trainingOutcome)


Cubist [Release 2.07 GPL Edition]  Tue Jun 21 13:00:22 2011
---------------------------------
```

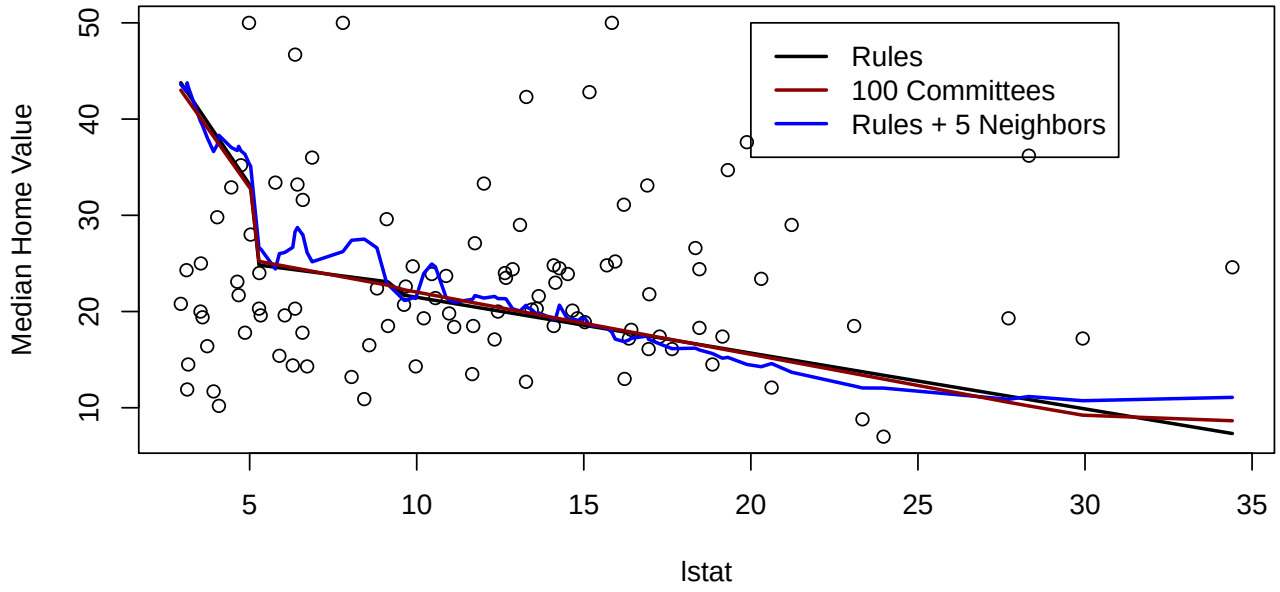

Figure 2: Different Cubist models for a single predictor.

```
    Target attribute `outcome'

Read 404 cases (14 attributes) from undefined.data

Model:

  Rule 1: [88 cases, mean 13.81, range 5 to 27.5, est err 2.10]

    if
        nox > 0.668
    then
        outcome = 2.07 + 3.14 dis - 0.35 lstat + 18.8 nox + 0.007 b
                  - 0.12 ptratio - 0.008 age - 0.02 crim

  Rule 2: [153 cases, mean 19.54, range 8.1 to 31, est err 2.16]

    if
        nox <= 0.668
        lstat > 9.59
    then
        outcome = 34.81 - 1 dis - 0.72 ptratio - 0.056 age - 0.19 lstat + 1.5 rm
```

```
                - 0.11 indus + 0.004 b

  Rule 3: [39 cases, mean 24.10, range 11.9 to 50, est err 2.73]

    if
        rm <= 6.23
        lstat <= 9.59
    then
        outcome = 11.89 + 3.69 crim - 1.25 lstat + 3.9 rm - 0.0045 tax
                - 0.16 ptratio

  Rule 4: [128 cases, mean 31.31, range 16.5 to 50, est err 2.95]

    if
        rm > 6.23
        lstat <= 9.59
    then
        outcome = -1.13 + 1.6 crim - 0.93 lstat + 8.6 rm - 0.0141 tax
                - 0.83 ptratio - 0.47 dis - 0.019 age - 1.1 nox


Evaluation on training data (404 cases):

    Average  |error|               2.27
    Relative |error|               0.34
    Correlation coefficient        0.94


        Attribute usage:
          Conds  Model

           78%   100%    lstat
           59%    53%    nox
           41%    78%    rm
                 100%    ptratio
                  90%    age
                  90%    dis
                  62%    crim
                  59%    b
                  41%    tax
                  38%    indus


Time: 0.0 secs
```

```
> modelTree$usage
```

```
   Conditions Model Variable
1          78   100    lstat
2          59    53      nox
3          41    78       rm
```

```
4           0   100  ptratio
5           0    90      age
6           0    90      dis
7           0    62     crim
8           0    59        b
9           0    41      tax
10          0    38    indus
11          0     0       zn
12          0     0     chas
13          0     0      rad
```

```
> library(caret)
> varImp(modelTree)
```

```
        Overall
lstat      89.0
nox        56.0
rm         59.5
ptratio    50.0
age        45.0
dis        45.0
crim       31.0
b          29.5
tax        20.5
indus      19.0
zn          0.0
chas        0.0
rad         0.0
```

It should be noted that this variable importance measure does not capture the influence of the predictors when using the instance–based correction.

6 Exporting the Model

As previously mentioned, this code is a port of the command–line C code. To run the C code, the training set data must be converted to a specific file format as detailed on the RuleQuest website. Two files are created. The `file.data` file is a header–less, comma delimited version of the data (the `file` part is a name given by the user). The `file.names` file provides information about the columns (eg. levels for categorical data and so on). After running the C program, another text file called `file.models`, which contains the information needed for prediction.

Once a model has been built with the R cubist package, the `exportCubistFiles` can be used to create the `.data`, `.names` and `.model` files so that the same model can be run at the command–line.

7 Current Limitations

There are a few features in the C code that are not yet operational in the R package:

- variable usage/importance haven't been ported into R objects
- only continuous and categorical predictors can be used (the C allows for other data types)
- there is an option to let the C code decide on using instances or not. The choice is more explicit in this package
- non–standard predictor names are not currently checked/fixed
- the C code supports binning of predictors

Many of these features will be implemented in the future.

8 About the Cubist C Code and Our Approach

This section may be interesting or important to those of you who care about the implementation (if you exist at all).

The cubist sources are written to take specific data files from the file system, pull them into memory, run the computations, then write the results to a text file that is also saved to the file system. The code makes use of a lot of global variables (especially for the data). The code has been around for a while and, after reading it, one can tell that the author put in a lot of time to catch many special cases. At Pfizer, we have pushed millions of samples through the non–GPL code without any substantive errors.

So the approach here is to pass in the training data as strings that mimic the formats that one would use with the command line version and get back the textual representation that would be saved to the `.model` file also as a string. The prediction function would then pass the model text string (and the data text string if instances are used) to the C code for prediction.

We did this for a few reasons. First, this approach would require us to re–write `main()` and touch as little of the original code as possible (otherwise we would have to write a parser for the data and try to get it into the global variable structure with complete fidelity). Second, most modeling functions implicitly assume that the data matrix is all numeric, thus factors are converted to dummy variables etc. Cubist doesn't want categorical data split into dummy variables based on how it does splits. Thus, we would have to pass in the numeric and categorical predictors separately unless we want to get really fancy.

9 Session Information

- R version 2.11.1 (2010-05-31), `x86_64-apple-darwin9.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/C/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: caret 4.91, cluster 1.12.3, Cubist 0.0.8, lattice 0.18-8, mlbench 2.1-0, plyr 1.2.1, reshape 0.8.3, reshape2 1.1
- Loaded via a namespace (and not attached): grid 2.11.1, stringr 0.4

10 References

Quinlan. Learning with continuous classes. Proceedings of the 5th Australian Joint Conference On Artificial Intelligence (1992) pp. 343-348

Quinlan. Combining instance-based and model-based learning. Proceedings of the Tenth International Conference on Machine Learning (1993a) pp. 236-243

Quinlan. *C4.5: Programs For Machine Learning* (1993b) Morgan Kaufmann Publishers Inc. San Francisco, CA

Wang and Witten. Inducing model trees for continuous classes. Proceedings of the Ninth European Conference on Machine Learning (1997) pp. 128-137

http://rulequest.com/cubist-info.html