M5PrimeLab

M5' regression tree and model tree toolbox for Matlab/Octave

ver. 1.1.0

Gints Jekabsons

Institute of Applied Computer Systems
Riga Technical University
Meza 1/3, LV-1048, Riga, Latvia
E-mail: gints.jekabsons@rtu.lv
URL: http://www.cs.rtu.lv/jekabsons/

Reference manual

July, 2015

CONTENTS

1. INTRODUCTION

What is M5PrimeLab

M5PrimeLab is a Matlab/Octave toolbox for building regression trees and model trees using M5' method (Wang & Witten 1997, Quinlan 1992). With this toolbox you can build trees, test them on separate test sets or using k-fold Cross-Validation, use the trees for prediction, as well as print or plot their structure. M5PrimeLab accepts input variables to be continuous, binary, and categorical, as well as manages missing values.

This reference manual provides overview of the functions available in the M5PrimeLab.

M5PrimeLab can be downloaded at http://www.cs.rtu.lv/jekabsons/.

The toolbox code is licensed under the GNU GPL ver. 3 or any later version.

For any feedback on the toolbox including bug reports feel free to contact me via the email address given on the title page of this reference manual.

Citing the M5PrimeLab toolbox

Please give a reference to the webpage in any publication describing research performed using the toolbox e.g., like this:

Jekabsons G., M5PrimeLab: M5' regression tree and model tree toolbox for Matlab/Octave, 2015, available at http://www.cs.rtu.lv/jekabsons/

2. AVAILABLE FUNCTIONS

M5PrimeLab toolbox provides the following list of functions:

- `m5pbuild` – builds M5' regression tree or model tree;
- `m5pparams` – creates a configuration for M5' algorithm for further use with `m5pbuild` and `m5pcv` functions;
- `m5ppredict` – makes predictions using an M5' tree;
- `m5ptest` – tests an M5' tree on a test data set;
- `m5pcv` – tests M5' performance using k-fold Cross-Validation;
- `m5pout` – outputs M5' tree in a human-readable form.

2.1. Function `m5pbuild`

Purpose:

Builds M5' regression tree or model tree.

Call:

```
[model, time] = m5pbuild(Xtr, Ytr, trainParams, binCat, verbose)
```

All the arguments, except the first two, of this function are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`Xtr, Ytr` : `Xtr` is a matrix with rows corresponding to observations, and columns corresponding to input variables. `Ytr` is a column vector of response values. Missing values in `Xtr` must be indicated as `NaN`.

`trainParams` : A structure of training parameters for the algorithm. If not provided, default values will be used (see function `m5pparams` for details).

`binCat` : A vector indicating type of each input variable (should be of the same length as `Xtr` second dimension). There are three possible choices:
0 = continuous variable (any other value < 2 has the same effect);
2 = binary variable;
3 (or any other value > 2) = categorical variable (the possible values are detected from the training data; any new values detected later e.g., in the test data, will be treated as `NaN`s).
(default value = vector of all zeroes, meaning that all the variables by default are treated as continuous)

`verbose` : Whether to output additional information to console. (default value = `true`)

Output:

`model` : The built M5' model – a structure with the following elements:

- `binCat` : Information regarding original (continuous / binary / categorical) variables, transformed (synthetic binary) variables, possible values for categorical variables, and lowest values all the variables. Note that this `binCat` does not contain the same information as the function argument with the same name.
- `trainParams` : A structure of training parameters for the algorithm (the same as in the input).

`tree` : A structure defining the built M5' tree.
`time` : Algorithm execution time (in seconds)

Remarks:

The M5' tree building algorithm builds a tree in two phases: growing phase and pruning phase. In the first phase the algorithm starts with one node and recursively tries to grow the tree so that intra-subset variation in the output variable's values down each branch is minimized (i.e., Standard Deviation Reduction (SDR) is maximized).

At the end of the first phase we have a large tree which typically overfits the data, and so a pruning phase is engaged. In this phase, the tree is pruned back from each leaf until an estimate of the expected error that will be experienced at each node cannot be reduced any further.

2.2. Function `m5pparams`

Purpose:

Creates a structure of M5' configuration parameter values for further use with `m5pbuild` and `m5pcv` functions.

Call:

```
trainParams = m5pparams(modelTree, minNumCases, prune, smoothing, smoothing_k, splitThreshold)
```

All the arguments of this function are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`modelTree` : Whether to build a model tree (`true`) or a regression tree (`false`) (default value = `true`). For model trees, each leaf will contain a linear regression model reduced in size using sequential backward selection algorithm.

`minNumCases` : The minimum number of training observations one node may represent. Values lower than 2 are not allowed. (default value = 4)

`prune` : Whether to prune the tree. (default value = `true`)

`smoothing` : Whether to perform smoothing when the tree is used for prediction (in `m5ppredict`). (default value = `false`)

`smoothing_k` : This is a kind of smoothing coefficient for the smoothing process. For larger values, more smoothing is applied. For large (relatively to the number of training observations) values, the tree will essentially behave like containing just one leaf (corresponding to the root node). For value 0, no smoothing is applied. (default value = 15 (Wang & Witten 1997))

`splitThreshold` : A node is not split if the standard deviation of the values of output variable at the node is less than splitThreshold of the standard deviation of output variable for the entire training data (default value = 0.05 (Wang & Witten 1997)). The results are usually not very sensitive to the exact choice of the threshold (Wang & Witten 1997).

Output:

`trainParams` : A structure of training parameters for `m5pbuild` function containing the provided values (or default ones, if not provided).

2.3. Function `m5ppredict`

Purpose:

Predicts output values for the given query points `Xq` using an M5' tree.

Call:

```
Yq = m5ppredict(model, Xq)
```

Input:

`model`	: M5' model.
`Xq`	: A matrix of query data points. Missing values in `Xq` must be indicated as `NaN`.

Output:

`Yq`	: A column vector of predicted response values.

Remarks:

1. If the data contains categorical variables, they are transformed in a number of synthetic binary variables (exactly the same way as `m5pbuild` does).
2. Every previously unseen value of a categorical variable is treated as `NaN`.

2.4. Function `m5ptest`

Purpose:

Tests an M5' tree on a test data set (`Xtst`, `Ytst`).

Call:

```
[MSE, RMSE, RRMSE, R2, MAE] = m5ptest(model, Xtst, Ytst)
```

Input:

`model`	: M5' model.
`Xtst, Ytst`	: `Xtst` is a matrix with rows corresponding to testing observations, and columns to corresponding input variables. `Ytst` is a column vector of response values. Missing values in `Xtst` must be indicated as `NaN`.

Output:

`MSE`	: Mean Squared Error
`RMSE`	: Root Mean Squared Error
`RRMSE`	: Relative Root Mean Squared Error
`R2`	: Coefficient of Determination
`MAE`	: Mean Absolute Error

2.5. Function `m5pcv`

Purpose:

Tests M5' performance using k-fold Cross-Validation.

Call:

```
[avgMSE, avgRMSE, avgRRMSE, avgR2, avgMAE, avgTime] = m5pcv(X, Y,
trainParams, binCat, k, shuffle, verbose)
```

All the arguments, except the first two, of this function are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`X, Y`	: Observations. Missing values in `X` must be indicated as `NaN`.
`trainParams`	: Algorithm parameters. See description for function `m5pbuild`.
`binCat`	: See description for function `m5pbuild`.
`k`	: Value of *k* for k-fold Cross-Validation. The typical values are 5 or 10. For Leave-One-Out Cross-Validation set *k* equal to *n*. (default value = 10)
`shuffle`	: Whether to shuffle the order of the observations before performing Cross-Validation. Note that the random seed value can be controlled externally before calling `m5pcv`. (default value = `true`)
`verbose`	: Whether to output additional information to console. (default value = `true`)

Output:

`avgMSE`	: Average Mean Squared Error
`avgRMSE`	: Average Root Mean Squared Error
`avgRRMSE`	: Average Relative Root Mean Squared Error
`avgR2`	: Average Coefficient of Determination
`avgMAE`	: Average Mean Absolute Error
`avgTime`	: Average execution time

2.6. Function `m5pout`

Purpose:

Outputs M5' tree in a human-readable form.

Call:

```
m5pout(model, showNumCases, precision, plotTree, plotFontSize)
```

All the arguments, except the first one, of this function are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`model`	: M5' model.
`showNumCases`	: Whether to show the number of training observations corresponding to each leaf (default value = `true`).
`precision`	: Number of digits in the model coefficients, split values etc. (default value = 15).
`plotTree`	: Whether to plot the tree instead of printing it (default value = `false`). In the plotted tree, upper child of a node corresponds to outcome 'true' and lower child to 'false'.
`plotFontSize`	: Font size for text in the plot (default value = 8).

Remarks:

1. If the training data has categorical variables, the corresponding synthetic variables will be shown.
2. The outputted tree will not reflect smoothing. Smoothing is performed only while predicting output values (in `m5ppredict` function).

3. EXAMPLE OF USAGE

We start by creating a data set using a three-dimensional function with one continuous, one binary, and one categorical variable (with four possible values). The data consists of randomly uniformly distributed 100 observations.

```
clear
X = [rand([100,1]) rand([100,1])>=0.5 floor(rand([100,1])*4)];
Y = X(:,1).*(X(:,3)==0) + X(:,2).*(X(:,3)==1) - ...
    2*X(:,1).*(X(:,3)==2) + 3*(X(:,3)==3) + 0.02*randn([100,1]);
```

First we try to build a model tree leaving all the building parameters to their defaults. We will supply `binCat` vector indicating that the second input variable is binary and the third is categorical. The M5' model is built by calling `m5pbuild`.

```
model = m5pbuild(X, Y, [], [0 2 3]);
```

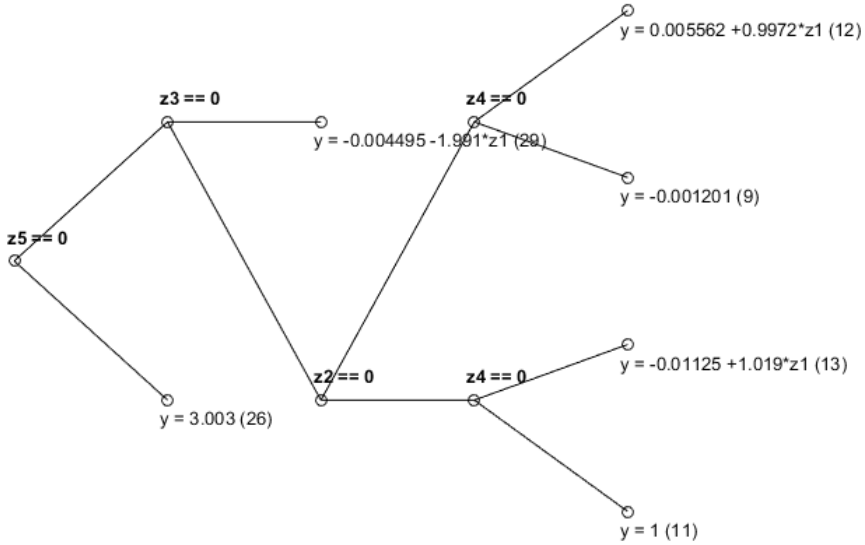
As the building process ends, we can examine the structure of the built model tree using function `m5pout`. First we see synthetic variables (automatically made if the data contains at least one categorical variable) and then the tree itself. Each leaf of the tree contains a linear regression model reduced using sequential backward selection algorithm. Number of training data observations for each leaf is shown in parentheses.

```
m5pout(model, true, 8)

Synthetic variables:
z1 = x1
z2 = x2
z3 = 1 if x3 is in {0, 1, 3} else = 0
z4 = 1 if x3 is in {1, 3} else = 0
z5 = 1 if x3 is in {3} else = 0
The tree:
if z5 == 0
 if z3 == 0
  y = -0.0044948312 -1.9907869*z1 (29)
 else
  if z2 == 0
   if z4 == 0
    y = 0.0055621063 +0.99723893*z1 (12)
   else
    y = -0.0012010096 (9)
  else
   if z4 == 0
    y = -0.011251211 +1.0187034*z1 (13)
   else
    y = 1.0001457 (11)
else
 y = 3.0032404 (26)
Number of rules in the tree: 6
```

Now let’s plot the tree. Upper child of a node corresponds to outcome 'true' and lower child to 'false'.

```
m5pout(model, true, 4, true, 10)
```



We can evaluate predictive performance of this (default) M5' configuration on the data using 10-fold Cross-Validation.

```
rand('state',0);
[avgMSE, avgRMSE, avgRRMSE, avgR2, avgMAE] = m5pcv(X, Y, [], [0 2 3])

avgMSE = 0.0069
avgRMSE = 0.0639
avgRRMSE = 0.0528
avgR2 = 0.9937
avgMAE = 0.0499
```

Let's try doing the same but instead of model tree we will build a regression tree. In a regression tree each leaf predicts the output using just a simple constant value.

```
params = m5pparams(false);
model = m5pbuild(X, Y, params, [0 2 3]);
m5pout(model, true, 8)

Synthetic variables:
z1 = x1
z2 = x2
z3 = 1 if x3 is in {0, 1, 3} else = 0
z4 = 1 if x3 is in {1, 3} else = 0
z5 = 1 if x3 is in {3} else = 0
The tree:
if z5 == 0
 if z3 == 0
  if z1 <= 0.4449
   if z1 <= 0.18334
    if z1 <= 0.089083
     y = -0.11622205 (4)
    else
     y = -0.24954647 (6)
   else
    if z1 <= 0.26445
     y = -0.44362806 (4)
    else
     y = -0.68927169 (6)
  else
   if z1 <= 0.69941
    y = -1.1387276 (4)
```

```
   else
    y = -1.6281793 (5)
 else
  if z2 == 0
   if z4 == 0
    if z1 <= 0.52764
     y = 0.23835163 (7)
    else
     y = 0.71649499 (5)
   else
    y = -0.0012010096 (9)
  else
   if z4 == 0
    if z1 <= 0.49522
     y = 0.30786035 (6)
    else
     y = 0.71409776 (7)
   else
    y = 1.0001457 (11)
else
 y = 3.0032404 (26)
Number of rules in the tree: 13

rand('state',0);
[avgMSE, avgRMSE, avgRRMSE, avgR2, avgMAE] = m5pcv(X, Y, params, [0 2 3])

avgMSE = 0.0204
avgRMSE = 0.1360
avgRRMSE = 0.1032
avgR2 = 0.9868
avgMAE = 0.0994
```

4. REFERENCES

1. Quinlan J.R. Learning with continuous classes. Proceedings of 5th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1992, pp. 343-348.
2. Wang Y. & Witten I.H. Inducing model trees for continuous classes. Proceedings of the 9th European Conference on Machine Learning, Prague, 1997, pp. 128-137.