

M5PrimeLab

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

ver. 1.0.1

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

September, 2010

CONTENTS

1. INTRODUCTION	3
2. AVAILABLE FUNCTIONS	4
2.1. Function m5pbuild	4
2.2. Function m5pparams	5
2.3. Function m5ppredict	6
2.4. Function m5ptest	6
2.5. Function m5pcv	6
2.6. Function m5pout	7
3. EXAMPLES OF USAGE	8
4. REFERENCES.....	10

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). The toolbox allows building the trees using different settings, testing them on a separate test set or using k-fold Cross-Validation, using them for prediction, outputting them in a user-readable way etc. M5PrimeLab accepts input variables to be continuous, binary, and categorical, as well as manages missing values.

M5PrimeLab is written entirely in Matlab/Octave. I tried to implement the functionality of M5' method as close to the description in the Wang & Witten's paper (Wang & Witten 1997) as possible.

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.

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, 2010, available at <http://www.cs.rtu.lv/jekabsons/>

2. AVAILABLE FUNCTIONS

M5PrimeLab toolbox provides the following list of functions:

- `m5pbuild` – builds a M5' regression tree or model tree;
- `m5pparams` – creates a configuration for M5' tree building algorithm for further use with `m5pbuild` or `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 the built M5' tree in a user-readable way.

2.1. Function `m5pbuild`

Purpose:

Builds a 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` : Training data cases ($x_{tr}(i,:)$, $y_{tr}(i)$), $i = 1, \dots, n$. Missing values in `Xtr` must be indicated as NaNs.

`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 NaNs).
(default value = vector of all zeroes, meaning that all the variables by default are treated as continuous)

`verbose` : Set to `false` for no verbose. (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` or `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:

For most applications, it can be expected that the most attention should be paid to the following parameters: `modelTree`, `smoothing`, and maybe `smoothing_k` or `minNumCases`.

<code>modelTree</code>	: Whether to build a model tree (<code>true</code>) or a regression tree (<code>false</code>). (default value = <code>true</code>)
<code>minNumCases</code>	: The minimum number of training data cases one node may represent. Values lower than 2 are not allowed. (default value = 4)
<code>prune</code>	: Whether to prune the tree. (default value = <code>true</code>)
<code>smoothing</code>	: Whether to perform smoothing when the tree is used for prediction (in <code>m5ppredict</code>). (default value = <code>false</code>)
<code>smoothing_k</code>	: 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 data cases) 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))
<code>splitThreshold</code>	: A node is not splitted if the standard deviation of the output variable values at the node is less than <code>splitThreshold</code> of the standard deviation of the output variable values of the entire original data set. (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:

<code>trainParams</code>	: A structure of training parameters for <code>m5pbuild</code> function containing the provided values of the parameters (or default ones, if not provided).
--------------------------	--

2.3. Function `m5ppredict`

Purpose:

Predicts output values for the given query points x_q using an M5' model.

Call:

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

Input:

`model` : M5' model
`Xq` : Inputs of query data points ($x_{q(i,:)}), i = 1, \dots, nq$. Missing values in x_q must be indicated as NaNs.

Output:

`Yq` : Predicted outputs of the query data points ($y_{q(i)}), i = 1, \dots, nq$

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 model on a test data set (x_{tst}, y_{tst}).

Call:

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

Input:

`model` : M5' model
`Xtst, Ytst` : Test data cases ($x_{tst(i,:)}, y_{tst(i)}), i = 1, \dots, ntst$. Missing values in x_{tst} must be indicated as NaNs.

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` : Data cases $(x(i,:), Y(i)), i = 1, \dots, n$. Missing values in `x` must be indicated as NaNs.

`trainParams` : See function `m5pbuild`.

`binCat` : See 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 data cases before performing Cross-Validation. Note that the random seed value can be controlled externally before calling `m5pcv`. (default value = true)

`verbose` : Set to `false` for no verbose. (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 the built M5' tree in a user-readable way.

Call:

`m5pout(model, showNumCases, precision)`

Input:

`model` : M5' model

`showNumCases` : Whether to show the number of training data cases corresponding to each leaf.

`precision` : Number of digits in the model coefficients, split values etc.

Remarks:

1. If the training data contained 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 cases.

```
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) + 0.05*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 `m5pbuid`.

```
model = m5pbuid(X, Y, [], [0 2 3]);
```

As the building process ends, we can examine the structure of the built model tree using function `m5pout`.

```
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 = 0.017929802 -2.0348163*z1 (23)
  else
    if z2 = 0
      if z4 = 0
        y = -0.0015175334 +0.97185955*z1 (10)
      else
        y = 0.012205123 (15)
    else
      if z4 = 0
        y = -0.07698552 +1.0910505*z1 (11)
      else
        y = 1.0122469 (16)
    else
      y = 2.9886087 (25)
Number of rules in the tree: 6
```

We evaluate predictive performance of this 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.

```
params = m5pparams(false);
model = m5pbuid(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 <= 0.40191
      if z1 <= 0.16103
        y = -0.11600635 (4)
      else
        if z1 <= 0.23678
          y = -0.43892861 (4)
        else
          y = -0.55706674 (4)
    else
      if z1 <= 0.68595
        y = -1.0703288 (5)
      else
        y = -1.7644304 (6)
  else
    if z2 = 0
      if z4 = 0
        if z1 <= 0.61442
          y = 0.22152349 (6)
        else
          y = 0.8139559 (4)
      else
        y = 0.012205123 (15)
    else
      if z4 = 0
        if z1 <= 0.35529
          y = 0.056672188 (4)
        else
          y = 0.71278659 (7)
      else
        y = 1.0122469 (16)
  else
    y = 2.9886087 (25)
```

Number of rules in the tree: 12

```
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. Induction of model trees for predicting continuous classes. Proceedings European Conference on Machine Learning, Prague, 1997, pp. 128-137.