M5PrimeLab

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

ver. 1.2.1

Gints Jekabsons

E-mail: gints.jekabsons@rtu.lv
http://www.cs.rtu.lv/jekabsons/

User's manual

October, 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) as well as building ensembles of trees using Bagging (Breiman, 1996) and Random Forests (Breiman, 2001, Breiman, 2002). With this toolbox you can build trees and ensembles of trees, test them on separate test sets or using Cross-Validation, use them for prediction, assess variable importance, as well as print or plot tree structure. M5PrimeLab accepts input variables to be continuous, binary, and categorical, as well as manages missing values.

The user's 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.

Feedback

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 user's manual.

Citing the M5PrimeLab toolbox

Jekabsons G., M5PrimeLab: M5' regression tree, model tree, and tree ensemble 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, model tree, or ensemble of trees;
- `m5pparams` – creates configuration for building M5' trees;
- `m5pparamsensemble` – creates configuration for building ensembles of trees;
- `m5ppredict` – makes predictions using M5' tree or ensemble of trees;
- `m5ptest` – tests M5' tree or ensemble of trees on a test data set;
- `m5pcv` – tests M5' performance using Cross-Validation;
- `m5pout` – prints or plots M5' tree in a human-readable form.

2.1. Function `m5pbuild`

Purpose:

Builds M5' regression tree, model tree, or ensemble of trees.

Call:

```
   [model, time, ensembleResults] = m5pbuild(Xtr, Ytr, trainParams, isBinCat,
trainParamsEnsemble, verbose)
```

All the input arguments, except the first two, 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. Input variables can be continuous, binary, as well as categorical (indicate using `isBinCat`). All values must be numeric. 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).
`isBinCat`	: A vector of flags indicating type of each input variable – either continuous (`false`) or categorical with any number of categories, including binary (`true`). The vector should be of the same length as the number of columns in `Xtr`. `m5pbuild` then detects all the actually possible values for categorical variables from the training data. Any new values detected later, e.g., in the test data, will be treated as `NaN`. By default, the vector is created with all values equal to `false`, meaning that all the variables are treated as continuous.
`trainParamsEnsemble`	: A structure of parameters for building ensemble of trees. If not provided, a single tree is built. See function `m5pparamsensemble` for details. This can also be useful for variable importance assessment. See user's manual for examples of usage. Note that the ensemble building algorithm employs random number generator for which you can set seed before calling `m5pbuild`.
`verbose`	: Whether to output additional information to console. (default value = `true`)

Output:

`model`	: A single M5' tree or a cell array of M5' trees if an ensemble is built. A structure defining one tree has the following fields:
`binCat`	: Information regarding original (continuous / binary / categorical) variables, transformed (synthetic binary) variables, possible values for categorical variables, and lowest values for all the variables.
`trainParams`	: A structure of training parameters for the algorithm (updated if any values are chosen automatically).
`tree`	: A structure defining the built M5' tree.
`time`	: Algorithm execution time (in seconds).
`ensembleResults`	: A structure of results if ensemble of tree is built. The structures has the following fields:
`OOBError`	: Out-of-bag estimate of prediction Mean Squared Error of the ensemble after each new tree is built. The number of rows is equal to the number of trees built. `OOBError` is available only if `getOOBError` in `trainParamsEnsemble` is set to `true`.
`varImportance`	: Variable importance assessment. Calculated when out-of-bag data of a variable is permuted. A matrix with four rows and as many columns as there are columns in `Xtr`. First row is the average increase of out-of-bag Mean Absolute Error (MAE), second row is standard deviation of the average increase of MAE, third row is the average increase of out-of-bag Mean Squared Error (MSE), fourth row is standard deviation of the average increase of MSE. The final variable importance estimate is usually calculated by dividing each MAE or MSE by the corresponding standard deviation. Bigger values then indicate bigger importance of the corresponding variable. See user's manual for example of usage. `varImportance` is available only if `getVarImportance` in `trainParamsEnsemble` is > 0.

Remarks:

M5' builds a tree in two phases: growing phase and pruning phase. In the first phase the algorithm starts with one leaf node and recursively tries to split each leaf node 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.

In M5PrimeLab, smoothing is done in `m5pbuild`, right after the pruning phase (instead of doing it only at the moment of prediction, i.e., in `m5ppredict`), by incorporating regression models of the interior nodes into regression models of each leaf. Smoothing can substantially increase accuracy of predictions but it also makes the trees more difficult to interpret, as smoothed trees have more complex models at their leaves (that can even look as if dropping of terms never occurred).

2.2. Function `m5pparams`

Purpose:

Creates configuration for building M5' trees. The structure is for further use with `m5pbuild` and `m5pcv` functions.

Call:

```
trainParams = m5pparams(modelTree, minNumCases, prune, smoothingK, splitThreshold, aggressivePruning)
```

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

It is quite possible that the default values for `minNumCases` and `smoothingK` will be far from optimal for your data.

For a typical configuration for trees in Random Forests, set `modelTree` = `false`, `minNumCases` = 5, `prune` = `false`, `smoothingK` = 0 with all the other parameters left to their defaults.

Input:

`modelTree`	: Whether to build a model tree (`true`) or a regression tree (`false`) (default value = `true`). For model trees, each leaf node will contain a linear regression model. If pruning is enabled, all the models will be reduced in size using sequential backward selection algorithm by greedily dropping terms.
`minNumCases`	: The minimum number of training observations one node may represent. Values lower than 2 are not allowed (default value = 4). If built trees are too large or overfit the data, consider increasing the number – this will result in smaller trees that are less sensitive to noise (but can also be underfitted).
`prune`	: Whether to prune the tree. (default value = `true`). First, all models are reduced in size by greedily dropping terms and then the tree itself is pruned by eliminating leaves and subtrees if doing so improves the estimated error.
`smoothingK`	: Smoothing parameter. Larger values mean more smoothing. Usually not recommended for regression trees but can be useful for model trees. Set to 0 to disable. Default value = 15 (Quinlan, 1992; Wang & Witten, 1997). Smoothing tries to compensate for sharp discontinuities occurring between adjacent nodes of the tree. In case studies by Quinlan, 1992, as well as Wang & Witten, 1997, this almost always had a positive effect on results. Smoothing is performed after building and pruning therefore this parameter does not influence those processes. Unfortunately, smoothed trees are harder to interpret.
`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 response variable for the entire training data (default value = 0.05 (i.e., 5%) (Wang & Witten, 1997)). The results are usually not very sensitive to the exact choice of the threshold (Wang & Witten, 1997).
`aggressivePruning`	: By default pruning is done as proposed by Quinlan, 1992, and Wang & Witten, 1997, but you can also employ more aggressive pruning, the one that is implemented in Weka's version of M5' (Hall et al., 2009). Simply put, in the aggressive version, while estimating error of a subtree, one penalizes not only the number of parameters of regression models at its leaves but also its total number of splits. Aggressive pruning will produce significantly smaller trees that are easier to interpret but it can also cause underfitting. (default value = `false`)

Output:

`trainParams`	: A structure of parameters for further use with `m5pbuild` and `m5pcv` functions containing the provided values (or default ones, if not provided).

2.3. Function `m5pparamsensemble`

Purpose:

Creates configuration for building ensembles of trees using Bagging or Random Forests. The structure is for further use with `m5pbuild` and `m5pcv` functions.

Call:

```
trainParamsEnsemble = m5pparamsensemble(numTrees, numVarsTry, withReplacement, inBagFraction, getOOBError, getVarImportance, verboseNumIter)
```

All the input arguments of this function are optional. Empty values are also accepted (the corresponding default values will be used). The default values are prepared for building Random Forests.

Remember to configure how individual trees are built for the ensemble (see description of `m5pparams`). See user's manual for examples of usage.

Input:

`numTrees`	: Number of trees to build (default value = 100). Should be set so that every data observation gets predicted at least a few times.
`numVarsTry`	: Number of input variables randomly sampled as candidates at each split in a tree. Set to -1 (default) to automatically sample one third of the variables (typical starting value for Random Forests in regression). Set to 0 to just use all variables (typical for Bagging). Set to a positive integer if you want some other number of variables to sample. To select a good value for `numVarsTry`, Leo Breiman suggests trying the default value and trying a value twice as high and half as low (Breiman, 2002). Note that while using this parameter, function `aresbuild` takes the total number of input variables directly from supplied training data set, not after synthetic binary variables are already made, if any.
`withReplacement`	: Should sampling of in-bag observations for each tree be done with (`true`) or without (`false`) replacement? Both, Bagging and Random Forests typically use sampling with replacement. (default value = `true`)
`inBagFraction`	: The fraction of the total number of observations to be sampled for in-bag set. Default value = 1, i.e., the in-bag set will be the same size as the original data set. This is the typical setting for both, Bagging and Random Forests. Note that for sampling without replacement `inBagFraction` should be lower than 1 so that out-of-bag set is not empty.
`getOOBError`	: Whether to perform out-of-bag error calculation to estimate prediction error of the ensemble (default value = `true`). Disable for speed.
`getVarImportance`	: Whether to assess importance of input variables (by calculating the average increase in error when out-of-bag data of a variable is permuted) and how many times the data is permuted per tree for the assessment (default value = 1). Set to 0 to disable and gain some speed. Number larger than 1 can give slightly more stable estimate, but the process is even slower.
`verboseNumIter`	: Set to some positive integer to print progress every `verboseNumIter` trees. Set to 0 to disable. (default value = 50)

Output:

`trainParamsEnsemble`	: A structure of parameters for further use with `m5pbuild` and `m5pcv` functions containing the provided values (or default ones, if not provided).

2.4. Function `m5ppredict`

Purpose:

Predicts response values for the given query points `Xq` using M5' tree or ensemble of trees.

Call:

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

Input:

`model`	: M5' model or a cell array of M5' models, if ensemble of trees is to be used.
`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. If `model` is an ensemble, `Yq` is a matrix whose rows correspond to `Xq` rows (i.e., observations) and columns correspond to each ensemble size (i.e., the increasing number of trees), the values in the very last column being the values for a full ensemble.

Remarks:

1. If the data contains categorical variables with more than two categories, they are transformed in a number of synthetic binary variables in exactly the same way as `m5pbuild` does it.
2. Every previously unseen value of a binary or categorical variable is treated as `NaN`.

2.5. Function `m5ptest`

Purpose:

Tests M5' tree or ensemble of trees on a test data set (`Xtst`, `Ytst`).

Call:

```
results = m5ptest(model, Xtst, Ytst)
```

Input:

`model`	: M5' model or a cell array of M5' models (if ensemble of trees is to be tested).
`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:

`results`	: A structure of different error measures calculated on the test data set. The structure has the following fields (if the `model` is an ensemble, the fields are column vectors with one value for each ensemble size, the very last value being error for a full ensemble):
`MAE`	: Mean Absolute Error.
`MSE`	: Mean Squared Error.
`RMSE`	: Root Mean Squared Error.

`RRMSE`	: Relative Root Mean Squared Error.
`R2`	: Coefficient of Determination.

2.6. Function `m5pcv`

Purpose:

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

Call:

```
[results, residuals] = m5pcv(X, Y, trainParams, isBinCat, k, shuffle, trainParamsEnsemble, verbose)
```

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

For more stable results, call `m5pcv` a few times and average the results.

Note that, if parameter `shuffle` is set to `true`, this function employs random number generator for which you can set seed before calling the function, if you require it.

Input:

`X, Y`	: Observations. Missing values in `X` must be indicated as `NaN`. (see function m5pbuild for details)
`trainParams`	: A structure of training parameters. If not provided, default values will be used (see function `m5pparams` for details).
`isBinCat`	: 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 observations before performing Cross-Validation. (default value = `true`)
`trainParamsEnsemble`	: A structure of parameters for building ensembles of trees. If not provided, a single tree is built. See function `m5pparamsensemble` for details.
`verbose`	: Whether to output additional information to console. (default value = `true`)

Output:

`resultsTotal`	: A structure of results averaged over Cross-Validation folds. For tree ensembles, the structure contains fields that are column vectors with one value for each ensemble size, the very last value being value for a full ensemble.
`resultsFolds`	: A structure of row vectors of results for each Cross-Validation fold. For tree ensembles, the structure contains matrices whose rows correspond to Cross-Validation folds while columns correspond to each ensemble size, the very last value being a value for a full ensemble.

Both structures have the following fields:

`MAE`	: Mean Absolute Error.
`MSE`	: Mean Squared Error.
`RMSE`	: Root Mean Squared Error.
`RRMSE`	: Relative Root Mean Squared Error. Not reported for Leave-One-Out Cross-Validation.
`R2`	: Coefficient of Determination. Not reported for Leave-One-Out Cross-Validation.

`nRules`	: Number of rules in tree. For ensembles of trees this field is omitted.
`nVars`	: Number of input variables included in tree. This counts original variables (not synthetic ones that are automatically made). For ensembles of trees this field is omitted.

2.7. Function `m5pout`

Purpose:

Prints or plots M5' tree in a human-readable form. Does not work with ensembles.

Call:

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

All the input arguments, except the first one, 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, left child of a node corresponds to outcome 'true' and right child to 'false'.
`plotFontSize`	: Font size for text in the plot (default value = 9).
`dealWithNaN`	: Whether to display how the tree deals with unknown values (`NaN`, displayed as '?'). (default value = `false`)

Remarks:

1. For smoothed trees, the smoothing process is already done in `m5pbuild`, therefore if you want to see unsmoothed versions (which are usually easier to interpret) you should build trees with smoothing disabled.
2. If the training data has categorical variables with more than two categories, the corresponding synthetic binary variables are shown.

3. EXAMPLES OF USAGE

3.1. Growing regression trees and model trees

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

```
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 let's try to grow a model tree. We will turn off smoothing because our data has sharp discontinuities and we don't want to loose them. All the other parameters will be left to their defaults. We will supply `isBinCat` vector indicating that the first input variable is continuous, the second is binary, and the third is categorical with four categories (detected automatically). M5' tree is grown by calling `m5pbuild`.

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

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

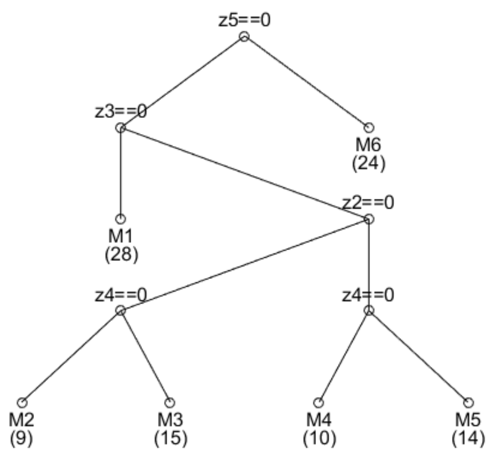
If the tree is too large or overfits consider increasing `minNumCases` parameter. You can also use `m5pcv` to test different configurations.

```
m5pout(model, true, 5);

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 = -0.0039976 -2.0036*z1 (28)
 else
  if z2 == 0
   if z4 == 0
    y = -0.0055303 +1.0162*z1 (9)
   else
    y = -0.0012792 (15)
  else
   if z4 == 0
    y = -0.0083598 +1.0228*z1 (10)
   else
    y = 1.0004 (14)
else
 y = 2.9962 (24)
Number of rules in the tree: 6
Number of original input variables in the tree: 3 (x1, x2, x3)
```

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

```
m5pout(model, true, 5, true)
```



```
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
Models:
M1 = -0.0039976 -2.0036*z1
M2 = -0.0055303 +1.0162*z1
M3 = -0.0012792
M4 = -0.0083598 +1.0228*z1
M5 = 1.0004
M6 = 2.9962
```

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

```
rng(1);
results = m5pcv(X, Y, params, [false true true])

results =
       MAE: 0.0209
       MSE: 0.0022
      RMSE: 0.0314
     RRMSE: 0.0217
        R2: 0.9990
    nRules: 5.9000
     nVars: 3
```

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

```
params = m5pparams(false, [], [], 0);
model = m5pbuild(X, Y, params, [false true true]);
m5pout(model, true, 5);

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 <= 0.54819
   if z1 <= 0.24359
    y = -0.28138 (4)
   else
    if z1 <= 0.3558
     y = -0.6216 (5)
    else
     y = -0.83133 (4)
  else
   if z1 <= 0.77846
    if z1 <= 0.67075
     y = -1.269 (4)
    else
     y = -1.4235 (4)
   else
```

```
      y = -1.7839 (7)
   else
    if z2 == 0
     if z4 == 0
      if z1 <= 0.55017
       y = 0.35085 (4)
      else
       y = 0.80781 (5)
     else
      y = -0.0012792 (15)
    else
     if z4 == 0
      if z1 <= 0.19898
       y = 0.11908 (6)
      else
       y = 0.51882 (4)
     else
      y = 1.0004 (14)
  else
   y = 2.9962 (24)
  Number of rules in the tree: 13
  Number of original input variables in the tree: 3 (x1, x2, x3)

  rng(1);
  results = m5pcv(X, Y, params, [false true true])

  results =
         MAE: 0.0838
         MSE: 0.0223
        RMSE: 0.1416
       RRMSE: 0.0985
          R2: 0.9894
      nRules: 11.5000
       nVars: 3
```

3.2. Growing ensembles of trees

For this example we will use Housing dataset available at the UCI repository. The data has 506 observations and 13 input variables. One input variable is binary, all others are continuous.

First, we must create a configuration for growing individual trees in the ensemble. We will create a typical configuration for Random Forests in regression problems: we will grow regression trees, the maximum number of observations in one node will be 5, the trees will not be pruned, as well as no smoothing will be applied.

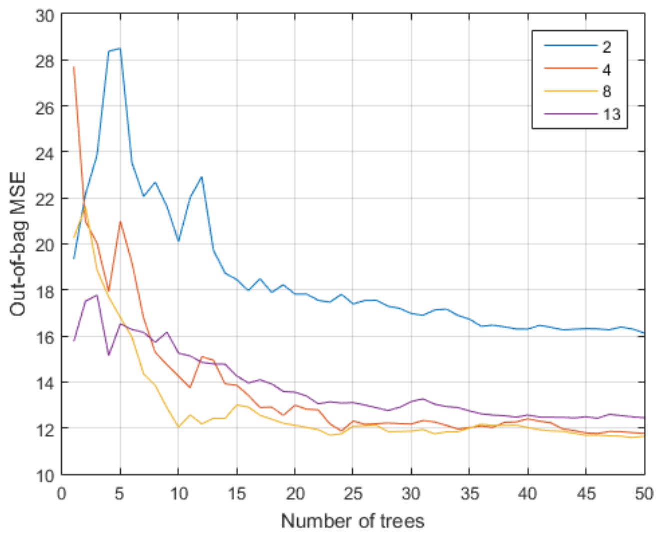
```
params = m5pparams(false, 5, false, 0);
```

Next, we must create a configuration for growing the ensemble. This is done using function `m5pparamsensemble`. The default parameters in this function are already prepared to grow Random Forests but let's try to find a better value for `numVarsTry` (this is the number of input variables randomly sampled as candidates at each split while growing a tree). By default the value is -1 which means that `m5pbuild` will automatically set it to one third of the number of input variables (a typical starting value for Random Forests in regression problems). For our data, the value is floor(13 / 3) = 4. But let's try also a value twice as high and half as low (as suggested by Breiman, 2002) as well as all variables (which is the value for Bagging), i.e., 2, 4, 8, and 13. For faster processing, we will grow only 50 trees. Later, when the “best” value is found, we will grow bigger ensemble.

We will also enable `getOOBError` because we need out-of-bag error estimates and disable `getVarImportance` because we don't yet need variable importance estimates.

An ensemble is grown by calling `m5pbuild`. We will supply `isBinCat` vector indicating that one variable is binary and the rest are continuous. By supplying the fifth argument (`paramsEnsemble`) we indicate that an ensemble should be created instead of just one tree.

```
paramsEnsemble = m5pparamsensemble(50, [], [], [], true, 0);
isBinCat = [false(1,3) true false(1,9)];
numVarsTry = [2 4 8 13];
figure;
for i = 1:4
    paramsEnsemble.numVarsTry = numVarsTry(i);
    [model, time, ensembleResults] = ...
        m5pbuild(X, Y, params, isBinCat, paramsEnsemble);
    plot(1:size(ensembleResults.OOBError,1), ensembleResults.OOBError(:,1));
    hold on;
end
grid on;
xlabel('Number of trees');
ylabel('Out-of-bag MSE');
legend({'2' '4' '8' '13'}, 'Location', 'NorthEast');
hold off;
```

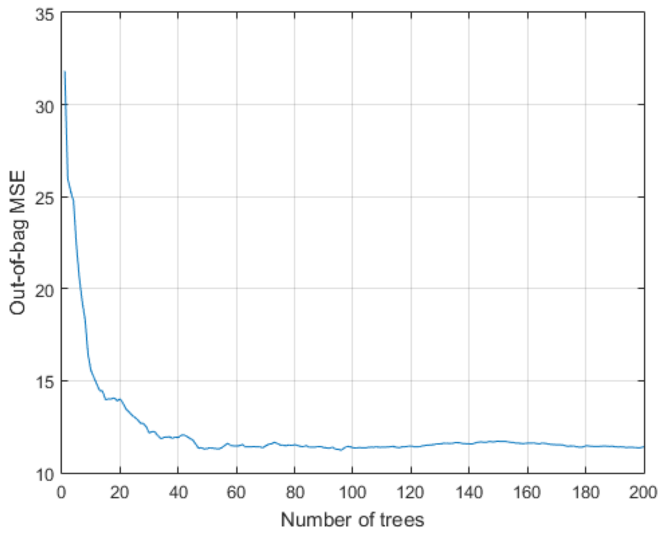


We can see that curves for values 4, 8, and 13 are very similar and are better than curve for 2. So let's say we choose `numVarsTry` to be the default, 4. Now we can build a larger ensemble consisting of 200 trees and, while we're at it, use it to estimate input variable importance (set `getVarImportance` to 1).

```
paramsEnsemble = m5pparamsensemble(200, 4, [], [], true, 1);
[model, time, ensembleResults] = m5pbuild(X, Y, params, isBinCat, paramsEnsemble);
```

Now we can inspect the out-of-bag error curve again.

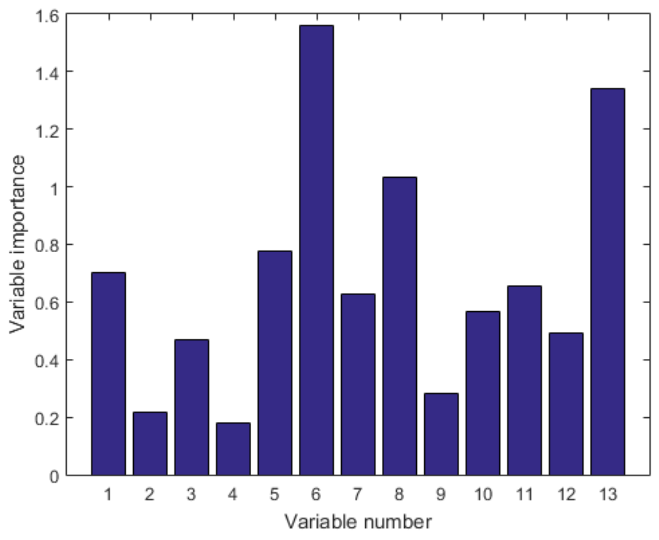
```
figure;
plot(1:size(ensembleResults.OOBError,1), ensembleResults.OOBError(:,1));
grid on;
xlabel('Number of trees');
ylabel('Out-of-bag MSE');
```



We can see that the prediction error estimate becomes quite stable. The values in last row of `ensembleResults.OOBError` show us that the ensemble of 200 trees estimates its prediction error to be MSE = 11.4.

Now let's plot variable importances. For that we will use the third and the forth row of `ensembleResults.varImportance`. The third row is the average increase of out-of-bag MSE when out-of-bag data of a variable is permuted. The fourth row is standard deviation of the average increase of the MSE. The importance estimate is usually calculated by dividing the increase by its standard deviation. Bigger values then indicate bigger importance of the corresponding variable (we could also express these values as percent of the maximum importance).

```
figure;
bar(1:size(ensembleResults.varImportance,2), ...
    ensembleResults.varImportance(3,:) ./ ensembleResults.varImportance(4,:));
xlabel('Variable number');
ylabel('Variable importance');
```

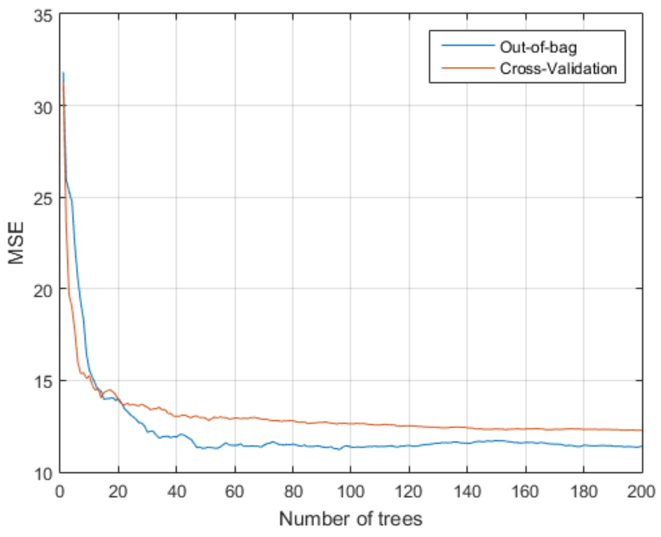


We can see that the 6th and 13th variables seem to be the most important ones while the 2nd and 4th seem to be the least important ones.

We can also evaluate the ensemble using Cross-Validation or a separate test data set. This is done using `m5pcv` and `m5ptest`. The output arguments of both these functions contain vectors with error calculated at each ensemble size (as well as, in case of `m5pcv`, at each fold).

```
rng(1);
resultsCV = m5pcv(X, Y, params, isBinCat, 10, [], paramsEnsemble);

figure;
plot(1:size(ensembleResults.OOBError,1), ensembleResults.OOBError(:,1));
hold on;
plot(1:size(resultsCV.MSE,1), resultsCV.MSE);
grid on;
xlabel('Number of trees');
ylabel('MSE');
legend({'Out-of-bag' 'Cross-Validation'}, 'Location', 'NorthEast');
hold off;
```



After doing Cross-Validation, we can see that in our case it gives us error estimate that is very similar to the previously calculated out-of-bag estimate.

4. REFERENCES

1. Breiman L. Bagging predictors. Machine Learning 24 (2), 1996, pp. 123-140.
2. Breiman L. Random forests. Machine Learning, 45 (1), 2001, pp. 5-32.
3. Breiman L. Manual on setting up, using, and understanding random forests v4.0. Statistics Department University of California Berkeley, CA, USA, 2002
4. Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., Witten I.H. The WEKA data mining software: an update, SIGKDD Explorations, 11 (1), 2009
5. Quinlan J.R. Learning with continuous classes. Proceedings of 5th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1992, pp. 343-348.
6. Wang Y. & Witten I.H. Induction of model trees for predicting continuous classes. Proceedings of the 9th European Conference on Machine Learning Poster Papers, Prague, 1997, pp. 128-137.