Inducing Model Trees for Continuous Classes

Yong Wang and Ian H. Witten

Department of Computer Science, University of Waikato, New Zealand
email {yongwang,ihw}@cs.waikato.ac.nz

Abstract. Many problems encountered when applying machine learning in practice involve predicting a "class" that takes on a continuous numeric value, yet few machine learning schemes are able to do this. This paper describes a "rational reconstruction" of M5, a method developed by Quinlan (1992) for inducing trees of regression models. In order to accommodate data typically encountered in practice it is necessary to deal effectively with enumerated attributes and with missing values, and techniques devised by Breiman et al. (1984) are adapted for this purpose. The resulting system seems to outperform M5, based on the scanty published data that is available.

1 Introduction

In our experience of applying machine learning to agricultural situations (Garner et al. 1995), many problems encountered in practice involve predicting a "class" that takes on a continuous numeric value, rather than a discrete category into which an example falls. Classical decision-tree and decision-rule learning methods have developed in an environment in which class values, and originally attribute values too, are discrete. Over the last decade it has become commonplace to extend induction techniques to deal with numerically-valued attributes by choosing a threshold at each node of the tree, or for each conjunct of a rule, and testing the value against that threshold. However, decision-tree and decision-rule learners are not commonly extended to situations where the class value itself is numeric.

There are, of course, several learning techniques that do predict numeric values. These techniques include standard regression, neural nets, instance-based learning, regression trees, and prediction by pre-discretization. But all have serious weaknesses. Standard regression is not a very potent way of representing an induced function because it imposes a linear relationship on the data. Neural nets and instance-based learning are more powerful but suffer from opacity: the model does not reveal anything about the structure of the function that it represents. Our experience is that most end users of machine learning are keenly interested in any light that the learning technique sheds on the structure of the data, sometimes more than in making good predictions on new data. Regression trees, which are adopted by the well-known CART system (Breiman et al. 1984), approximate a non-linear function by a piecewise constant one. Prediction by discretizing simply maps the problem into a categorical-class one.

A new technique for dealing with continuous-class learning problems, the "model tree," has been developed by Quinlan (1992) and embodied in a learning algorithm called M5. Model trees combine a conventional decision tree with the possibility of linear regression functions at the leaves. This representation is relatively perspicuous because the decision structure is clear and the regression functions do not normally involve many variables.

This paper describes a new implementation of a model-tree inducer, based on Quinlan's pioneering work. However, details of that work are not readily available, and solutions to some small design decisions had to be worked out from scratch. We also adopted a method from the CART system (Breiman et al. 1984) for dealing with enumerated attributes, and adapted a method for treating missing values—both of which feature strongly in the real-world datasets that we have encountered in practice. We find that the resulting model tree inducer—a public-domain "rational reconstruction" of the original system—which we call M5′, performs somewhat better than the original algorithm on the standard datasets for which results have been published. We also introduce a modification which allows the tree size to be reduced dramatically with only a small penalty in prediction performance, leading to much more comprehensible models. Finally, we present results which test the method used for dealing with missing values.

2 Inducing Model Trees

The basic idea behind building a model tree is quite straightforward. In the first stage, a decision-tree induction algorithm is used to build a tree. Instead of maximizing the information gain at each interior node, a splitting criterion is used that minimizes the intra-subset variation in the class values down each branch. In the second stage, consideration is given to pruning the tree back from each leaf, a technique that was pioneered independently by Breiman et al. (1984) and Quinlan (1986), and has become standard in decision-tree induction. The only difference is that when pruning to an interior node, consideration is given to replacing that node by a regression plane instead of a constant value. And the attributes that serve to define that regression are precisely those that participate in decisions in nodes subordinate to the current one.

2.1 Building the Initial Tree

The splitting criterion is based on treating the standard deviation of the class values that reach a node as a measure of the error at that node, and calculating the expected reduction in error as a result of testing each attribute at that node. The attribute which maximizes the expected error reduction is chosen. The standard deviation reduction (SDR) is calculated by the formula

$$\mathrm{SDR} = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \ , \tag{1}$$

where T is the set of examples that reach the node and T_1, T_2, ... are the sets that result from splitting the node according to the chosen attribute. The tree-growing procedure is similar to that of CART (Breiman et al. 1984), except that

CART chooses the attribute that gives the greatest expected reduction of either variance or absolute deviation, a choice that is made at run-time as a command-line option. Our experience is that results are quite insensitive to which of these three criteria is chosen: similar trees are obtained in each case. Splitting in M5 ceases when the class values of all the instances that reach a node vary very slightly, or only a few instances remain.

2.2 Pruning the Tree

The pruning procedure makes use of an estimate of the expected error that will be experienced at each node for test data. First, the absolute difference between the predicted value and the actual class value is averaged for each of the training examples that reach that node. This average will underestimate the expected error for unseen cases, of course, and to compensate, it is multiplied by the factor $(n + \nu)/(n - \nu)$, where n is the number of training examples that reach the node and ν is the number of parameters in the model that represents the class value at that node.

M5 computes a linear model for each interior node of the unpruned tree. The model is calculated using standard regression, using only the attributes that are tested in the subtree below this node. The resulting linear model is simplified by dropping terms to minimize the estimated error calculated using the above multiplication factor—dropping a term decreases the multiplication factor, which may be enough to offset the inevitable increase in average error over the training examples. Terms are dropped one by one, greedily, so long as the error estimate decreases. Finally, once a linear model is in place for each interior node, the tree is pruned back from the leaves, so long as the expected estimated error decreases.

2.3 Smoothing

A final stage is to use a smoothing process to compensate for the sharp discontinuities that will inevitably occur between adjacent linear models at the leaves of the pruned tree, particularly for some models constructed from a small number of training instances. The smoothing procedure described by Quinlan (1992) first uses the leaf model to compute the predicted value, and then filters that value along the path back to the root, smoothing it at each node by combining it with the value predicted by the linear model for that node. The calculation is

$$p' = \frac{np + kq}{n + k} \ , \tag{2}$$

where p' is the prediction passed up to the next higher node, p is the prediction passed to this node from below, q is the value predicted by the model at this node, n is the number of training instances that reach the node below, and k is a constant (default value 15). As we see below, smoothing substantially increases the accuracy of predictions.

3 Implementing M5′

The M5′ algorithm is based closely on Quinlan's ideas as summarized above. Some details are not completely resolved in the publications describing M5; we identify these in the first section below and give our interpretation. Moreover, it is not clear how enumerated attributes and missing values should be handled. Since these features are of vital importance for the real-world data sets that we have encountered in our practical work, we have incorporated ways of dealing with them which are described in the next two sections. Finally, we give pseudo-code for the M5′ algorithm.

3.1 Further Details of the Original Algorithm

Here we discuss four details of the original M5 and clarify the approach of M5′.

First, the form of the linear models is quite clear. For a particular node in the tree with k attributes, say a_1, a_2, $\ldots$, a_k, which are tested in splittings below that node, we use $k+1$-parameter model that includes a constant term. Because this makes the compensation factor $(n+\nu)/(n-\nu)$ used in calculating the expected error infinite at leaves having one example (since $n = \nu = 1$), we never allow a split that creates a leaf with fewer than two training examples.

Second, during the initial splitting procedure, M5 does not split a node if it represents very few examples or their values vary only slightly. Since our leaves cannot contain fewer than two training examples, we do not split nodes if they represent three examples or less. Neither do we split them if the standard deviation of the class values of the examples at the node is less than 5% of the standard deviation of the class values of the entire original set of examples. Experiments show that the results are not very sensitive to the exact choice of threshold.

Third, it is not clear exactly what attributes are used in the linear models. Recall that, during pruning, attributes are dropped from a model when their effect is so small that it actually increases the estimated error. Do these attributes participate in higher-level models or not? After some experimentation, we decided to leave them in; sometimes lower-level models seem to discard attributes that higher-level models can in fact use effectively. Tests showed that although the error rate is not very sensitive to this decision, much smaller trees were sometimes obtained by leaving the attributes in.

Fourth, when making the decision whether or not to prune a subtree, it is necessary to compare the estimated expected error for the linear model at that node with the estimated expected error from the subtree. To calculate the latter, the expected error from each branch is combined into a single overall value for the node using a linear sum in which each branch is weighted by the proportion of the training examples that go down it.

3.2 Enumerated Attributes

Before constructing a model tree, all enumerated attributes are transformed into binary variables. For each enumerated attribute, the average class value corresponding to each possible value in the enumeration is calculated from the training examples, and the values in the enumeration are sorted according to these averages. Then, if the enumerated attribute has k possible values, it is replaced by $k-1$ synthetic binary attributes, the ith being 0 if the value is one of the first i in the ordering and 1 otherwise. Thus in M5$'$ all splits are binary: they involve either a continuous-valued attribute or a synthetic binary one.

This technique is based on a development by Breiman et al. (1984 p. 274), who prove that the best split at a node for an enumerated variable with k values is one of the $k-1$ positions obtained by ordering the average class values for each enumerated value. CART performs this operation at each node for every enumerated attribute. However, M5$'$ does it only once before starting to build a model tree. Although this has the advantage of speed, it does assume that the optimal ordering of different values of each attribute at nodes in the tree is, in general, the same as the optimal ordering for the entire data set, an assumption that may be invalid when data sets are not uniformly sampled. On the other hand, CART's method may suffer from the inevitable increase in noise due to small numbers of examples at lower nodes in the tree—and in some cases nodes may not represent all values for some attributes.

As with regular decision tree induction, the problem arises that enumerated attributes having a large number of different values are automatically favored. For instance, an attribute that has a different value for each example will automatically be chosen first for splitting, despite the fact that it has no predictive power. C4.5 deals with this problem by defining the information gain in a way that takes account of the number of subnodes created by the split (Quinlan 1993b), but this solution does not transfer easily to the situation where a k-valued enumerated attribute is transformed into $k-1$ binary attributes.

Our solution in M5$'$ is to multiply the SDR value by a factor β that is unity for a binary split and decays exponentially as the number of values increases.

3.3 Missing Values

To take account of missing values, the SDR is further modified to

$$\mathrm{SDR} = \frac{m}{|T|} \times \beta(i) \times \left[sd(T) - \sum_{j \in \{L,R\}} \frac{|T_j|}{|T|} \times sd(T_j) \right]. \tag{3}$$

m is the number of examples without missing values for that attribute, and T is the set of examples that reach this node. $\beta(i)$ is the correction factor mentioned in the previous section, calculated for the original attribute to which this synthetic attribute corresponds. T_L, T_R are sets that result from splitting on this attribute—for all attributes are now binary.

Once an attribute is selected for splitting, it is necessary to divide the examples into subsets according to their value for this attribute. An obvious problem arises when the value is missing. CART uses an interesting technique called "surrogate splitting" to handle this situation. Essentially, it finds another attribute to split on in place of the original one and uses it instead. The attribute is chosen as the one most highly correlated with the original attribute. However, this technique is both complex to implement and time-consuming to execute, and in M5′ we are experimenting with a simpler heuristic.

During training, we use the class value as the surrogate attribute, in the belief that, a priori, this is the attribute most likely to be correlated with the one being used for splitting. We first deal with all examples for which the value of the splitting attribute is known. If it is continuous, we determine a numeric threshold for splitting in the usual way, by sorting the examples according to its value and, for each possible split point, calculating the SDR according to the above formula, choosing the split point that yields the greatest reduction in error. Only the examples for which the value of the splitting attribute is known are used to determine the split point.

Then we divide these examples into the two sets L and R according to the test. In the case of an enumerated attribute, we determine whether the examples in L or R have the greater average class value, and we calculate the average of these two averages. Then, an example for which this attribute value is unknown is placed into L or R according to whether its class value exceeds this overall average or not—if it does, it goes into whichever of L and R has the greater average class value, otherwise it goes into the one with the smaller average class value. In the case of a continuous attribute we sort on attribute value and then use the same procedure, except that we just use a few examples each side of the threshold—the top three examples in L and the bottom three in R—to determine the average class value to be used for splitting the examples with missing values. When the splitting stops, all the missing values will be replaced by the average values of the corresponding attributes of the training examples reaching the leaves.

So much for processing the training set. Suppose a node is encountered when processing a test example that specifies a test on an attribute whose value is unknown. We cannot use the class value as above, because it too is unknown. Consequently, we simply replace the unknown attribute value by the average value of that attribute for the training examples that reach the node—which has the effect, for a binary attribute, of choosing the most populous subnode. This simple approach seems to work very well in practice.

3.4 Pseudo-Code for M5′

Figure 1 gives pseudo-code for M5′. The two main parts are creating a tree by successively splitting nodes, performed by `split`, and pruning it from the leaves upwards, performed by `prune`. The `node` data structure contains a type flag saying whether it is an internal node or a leaf, pointers to the left and right child, the set of examples that reach that node, the attribute that is used for splitting at that node, and a structure representing the linear model for the node.

The `sd` function called at the beginning of the main program and again at the beginning of `split` calculates the standard deviation of the class values of a set of examples. The procedure for obtaining synthetic binary attributes that

```
M5'(examples)
{
  SD = sd(examples)
  for each k-valued enumerated attribute
    convert into k-1 synthetic binary attributes
  root = new_node
  root.examples = examples
  split(root)
  prune(root)
  print_tree(root)
}
split(node)
{
  if sizeof(node.examples) < 4 or
    sd(node.examples) < 0.05*SD
    node.type = LEAF
  else
    node.type = INTERIOR
    for each continuous and binary attribute
      for all possible split positions
        calculate the attribute's SDR
    node.attribute = attribute with max SDR
    split(node.left)
    split(node.right)
}
prune(node)
{
  if node = INTERIOR then
    prune(node.left_child)
    prune(node.right_child)
    node.model = linear_regression(node)
    if subtree_error(node) > error(node) then
      node.type = LEAF
}
subtree_error(node)
{
  l = node.left; r = node.right
  if node = INTERIOR then
    return (sizeof(l.examples)*subtree_error(l) +
            sizeof(r.examples)*subtree_error(r))/
            sizeof(node.examples)
  else return error(node)
}
```

Fig. 1. Pseudo-code for the M5′ algorithm

follows was discussed in Sect. 3.2. Standard procedures for creating new nodes and printing the final tree are not shown. In `split`, `sizeof` returns the number of elements in a set. Missing attribute values are dealt with as described in Sect. 3.3. The SDR is calculated according to (3). Although not shown in the code, it is set to infinity if splitting on the attribute would create a leaf with less than two examples. In `prune`, the `linear-regression` routine recursively descends the subtree collecting attributes, performs a linear regression on the examples at that node as a function of those attributes, and then greedily drops terms if doing so improves the error estimate, as described in Sect. 2.2. Finally, the `error` function returns

$$\frac{(n+\nu)}{(n-\nu)} \times \frac{\sum_{examples} |\text{deviation from predicted class value}|}{n} \tag{4}$$

where n is the number of examples at the node and ν the number of parameters in the node's linear model.

4 Empirical Results

We performed empirical testing of the implementation on the six data sets described by Quinlan (1993a), taken from the UCI Repository of Machine Learning Databases. The performance measure we adopt is the relative error as defined by Breiman et al. (1984, p. 224). All results given below are calculated using 10-fold cross validation, repeated twenty times and averaged.

The datasets *pw-linear* and *auto-price* contain continuous attributes (10 and 15 respectively); they do not have any enumerated attributes. *Housing* contains 12 continuous and one enumerated attribute, but the latter is binary so our treatment has no special effect. *Cpu* has six continuous and one 30-valued enumerated attribute. *Auto-mpg* contains five continuous and two enumerated attributes, the latter having five and three possible values respectively. Finally, *servo* contains two continuous and two five-valued enumerated attributes; it is the one for which the enumerated attributes have the most prominent effect.

4.1 Comparing M5′ with M5

Table 1 shows the error rates of standard linear regression, the results Quinlan (1993a) gives for M5, and the results of M5′, both smoothed and unsmoothed. Our experimental results are written in the form "mean ± standard deviation," calculated from the twenty individual cross-validations. They all use synthetic binary attributes.

Quinlan (1993a) also gives the results of standard linear regression, although it is unclear what approach he takes to enumerated attributes. Our regression results resemble Quinlan's except in the case of the *auto-mpg* data set, when (for reasons that we do not understand) he obtains a much higher relative error.

Comparing the second and third rows, that is, the published results of M5 with those of M5′, we see similar performance for *pw-linear*, *housing*and *auto-price*, slightly better results for *cpu* and *auto-mpg*, and dramatically better results for *servo*. Since the first three datasets have either no enumerated attributes or only binary-valued enumerated attributes, we conclude that the policy of creating synthetic binary attributes for each multi-valued enumerated attribute has a significant positive effect on performance.

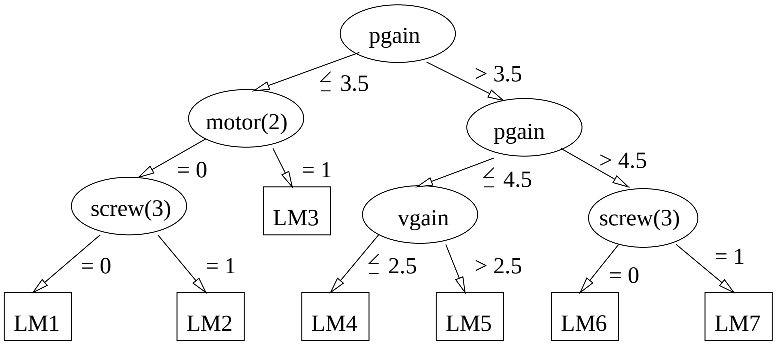
Comparing the final two rows of Table 1 shows that smoothing does indeed have an appreciable effect on results for most of the datasets.

	pw-linear	housing	cpu	auto-price	auto-mpg	servo
regression	31.7±1.0	27.6±0.4	18.4±2.1	19.6±1.0	14.2±0.3	48.1±1.5
M5	9.5	18.6	17.2	16.2	14.7	28.7
M5′ (smoothed)	9.4±0.3	17.8±2.4	14.6±2.4	14.9±1.2	13.0±0.5	15.8±1.7
M5′ (unsmoothed)	9.3±0.3	19.9±2.5	16.3±2.8	18.2±1.8	13.6±0.5	18.3±2.2

Table 1. Relative errors (%) on test cases (results for M5 from Quinlan, 1993)

4.2 A model for the *servo* Dataset

It is instructive to take a closer look at the results for the *servo* dataset, which has two continuous and two discrete attributes. The discrete attributes play important roles. Figure 2 shows the structure of the model tree found by M5′. Four synthetic binary attributes have been created for each of the five-valued enumerated attributes *motor* and *screw*, and are each shown in the Table in terms of the two sets of values to which they correspond.



Model	LM1	LM2	LM3	LM4	LM5	LM6	LM7
constant term	−0.44	2.6	3.5	0.18	0.52	0.36	0.23
pgain							
vgain	0.82		0.42				0.06
motor = D *vs* E, C, B, A		3.3		0.24	0.42		
motor = D, E *vs* C, B, A	1.8			−0.16		0.15	0.22
motor = D, E, C *vs* B, A				0.1	0.09		0.07
motor = D, E, C, B *vs* A			0.18				
screw = D *vs* E, C, B, A							
screw = D, E *vs* C, B, A	0.47						
screw = D, E, C *vs* B, A	0.63		0.28	0.34			
screw = D, E, C, B *vs* A			0.9	0.16	0.14		

Fig. 2. Model tree and linear models for data set *servo*

4.3 Tests on Missing Values

Finally, Table 2 shows some results for versions of the standard datasets into which missing values have been artificially introduced. For all the datasets, 5%, 10%, and 25% of the non-class attributes were removed at random. Results for CART on two of the datasets appear in Breiman et al. (1984 pp. 249–250) and are shown in the Table; in every case M5′ performs better, usually significantly better. All the results in Table 2 are obtained when the pruning factor α is 1.0.

Data missing	pw-linear		housing		cpu	auto-price	auto-mpg	servo
	M5′	CART	M5′	CART				
0 %	9.4	17	17.8	22	14.6	14.9	13.0	15.8
5 %	16.7	27	22.3	22	20.5	20.3	17.4	25.3
10 %	23.2	36	23.0	31	24.8	22.1	21.7	33.2
25 %	43.6	65	34.6	35	35.6	29.9	32.1	53.4

Table 2. Relative error (%) for different amounts of missing data

5 Conclusion

This paper has described M5′, a reconstruction of a machine learning method for inducing model trees from empirical data. This system fills a notable gap by providing a public-domain scheme[1] for inducing models from data that involves continuous classes. It deals effectively with both enumerated attributes and missing attribute values. Its performance is not generally quite as high as the best of the methods described by Quinlan (1993a), which involve neural nets and instance-based learning (although its performance overall is arguably superior to either of these methods individually). However, it has the advantage that the models it generates are compact and relatively comprehensible.

There are many productive areas for future work. A number of design decisions and parameters in M5′ seem *ad hoc* and arbitrary; it is possible that a minimum description length formulation will be able to provide a design that is more principled. The pruning factor controls a tradeoff between prediction and tree size that deserves further investigation. And we have yet to return, armed with our method, to the agricultural problems that originally motivated this work.

References

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont CA. (1984)

Garner, S.R., Cunningham, S.J., Holmes, G., Nevill-Manning, C.G. and Witten, I.H.: Applying a machine learning workbench. Proc Machine Learning in Practice Workshop, Machine Learning Conference, Tahoe City, CA. (1995) 14–21

Quinlan, J.R.: Simplifying decision trees. Proc Workshop on Knowledge Acquisition for Knowledge-based Systems, Banff, Canada.(1986)

Quinlan, J.R.: Learning with continuous classes. Proceedings 5th Australian Joint Conference on Artificial Intelligence. World Scientific, Singapore. (1992) 343–348

Quinlan, J.R.: Combining instance-based and model-based learning. International Conference on Machine Learning. (1993) 236–243

Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann. (1993)

[1] The code for M5′ is available from `http://www.cs.waikato.ac.nz/~ml`