In Proceedings AI'92 (Adams & Sterling, Eds), 343-348, Singapore: World Scientific, 1992

LEARNING WITH CONTINUOUS CLASSES

J. R. QUINLAN
Basser Department of Computer Science, University of Sydney
Sydney, Australia 2006

ABSTRACT: Some empirical learning tasks are concerned with predicting values rather than the more familiar categories. This paper describes a new system, M5, that constructs tree-based piecewise linear models. Four case studies are presented in which M5 is compared to other methods.

1. Introduction

One branch of machine learning, *empirical learning*, is concerned with building or revising models in the light of large numbers of exemplary cases, taking into account typical problems such as missing data and noise. Many such models involve classification and, for these, learning algorithms that generate decision trees are efficient, robust, and relatively simple[1,7].

Other tasks, however, require the learned model to predict a numeric value associated with a case rather than the class to which the case belongs. For instance, a case might be the composition of an alloy and the associated value the temperature at which it melts. Some researchers have attempted to use decision tree methods for value prediction by dividing the value's range into small "categories" such as 0-4%, 5-9%, etc., then using systems that build classification models. These attempts often fail, partly because algorithms for building decision trees cannot make use of the implicit ordering of such classes.

Several more effective learning methods are available for predicting real values. The CART program[1] builds *regression trees* that differ from decision trees only in having values rather than classes at the leaves. MARS[3], a recent and elegant system, constructs models whose basis functions are splines. Many classical statistical methods such as multiple linear regression address the same task, postulating a simple form for the model and then finding parameter values that maximise its fit to the training data[6].

This paper describes M5, a new system for learning models that predict values. Like CART, M5 builds tree-based models but, whereas regression trees have values at their leaves, the trees constructed by M5 can have multivariate linear models; these *model trees* are thus analogous to piecewise linear functions. M5 learns efficiently and can tackle tasks with very high dimensionality – up to hundreds of attributes. This ability sets M5 (and CART) apart from MARS, whose computational requirements grow very rapidly with dimensionality, effectively limiting its applicability to tasks with no more than 20 or so attributes. The advantage of M5 over CART is that model trees are generally much smaller than regression trees and have proven more accurate in the tasks investigated.

This paper gives an overview of the algorithm for constructing model trees, then reports the performance of M5 on four learning tasks.

2. Constructing Model Trees

We suppose that we have a collection T of training cases. Each case is specified by its value of a fixed set of attributes, either discrete or numeric, and has an associated a target value. The aim is to construct a model that relates the target values of the training cases to their values of the other attributes. The worth of the model will generally be measured by the accuracy with which it predicts the target values of unseen cases.

Tree-based models are constructed by the divide-and-conquer method. The set T is either associated with a leaf, or some test is chosen that splits T into subsets corresponding to the test outcomes and the same process is applied recursively to the subsets. This relentless division often produces over-elaborate structures that must be pruned back, for instance by replacing a subtree with a leaf.

The first step in building a model tree is to compute the standard deviation of the target values of cases in T. Unless T contains very few cases or their values vary only slightly, T is split on the outcomes of a test. Every potential test is evaluated by determining the subset of cases associated with each outcome; let T_i denote the subset of cases that have the ith outcome of the potential test. If we treat the standard deviation $sd(T_i)$ of the target values of cases in T_i as a measure of error, the expected reduction in error as a result of this test can be written

$$\Delta error = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i).$$

After examining all possible tests, M5 choose one that maximises this expected error reduction. (For comparison, CART chooses a test to give the greatest expected reduction in either variance or absolute deviation.)

The major innovations of M5 come into play after the initial tree has been grown. A detailed discussion is precluded by the length of this paper, but the main ideas are:

Error estimates: M5 often needs to estimate the accuracy of a model on unseen cases. First, the *residual* of a model on a case is just the absolute difference between the actual target value of the case and the value predicted by the model. To estimate the error of a model derived from a set of training cases, M5 first determines the average residual of the model on these cases. This will generally underestimate the error on unseen cases, so M5 multiplies the value by $(n+\nu)/(n-\nu)$, where n is the number of training cases and ν is the number of parameters in the model. The effect is to increase the estimated error of models with many parameters constructed from small numbers of cases.

Linear models: A multivariate linear model is constructed for the cases at each node of the model tree using standard regression techniques[6]. However, instead of using all attributes, this model is restricted to the attributes that are referenced by tests or linear models somewhere in the subtree at this node. As M5 will compare the accuracy of a linear model with the accuracy of a subtree, this ensures a level playing field in which the two types of models use the same information.

Simplification of linear models: After each linear model is obtained as above, it is simplified by eliminating parameters to minimise its estimated error. Even though the elimination of parameters generally causes the average residual to increase, it also reduces the multiplicative factor above, so the estimated error can decrease. M5 uses a greedy search to remove variables that contribute little to the model; in some cases, M5 removes all variables, leaving only a constant.

Pruning: Each non-leaf node of the model tree is examined, starting near the bottom. M5 selects as the final model for this node either the simplified linear model above or the model subtree, depending on which has the lower estimated error. If the linear model is chosen, the subtree at this node is pruned to a leaf.

Smoothing: Pregibon[5] observes that the prediction accuracy of tree-based models can be improved by a smoothing process. When the value of a case is predicted by a model tree, the value given by the model at the appropriate leaf is adjusted to reflect the predicted values at nodes along the path from the root to that leaf. The form of smoothing used by M5 differs from that developed by Pregibon, but the motivation is similar. M5's smoothed predicted value is backed up from the leaf to the root as follows:

- The predicted value at the leaf is the value computed by the model at that leaf.
- If the case follows branch S_i of subtree S, let n_i be the number of training cases at S_i, $PV(S_i)$ the predicted value at S_i, and $M(S)$ the value given by the model at S. The predicted value backed up to S is

$$PV(S) = \frac{n_i \times PV(S_i) + k \times M(S)}{n_i + k}$$

  where k is a smoothing constant (default 15).

Smoothing has most effect on a case when the models along the path predict very different values and when some models were constructed from few training cases.

3. Case Studies

M5 has been evaluated on several learning tasks for which results of other methods are also available. When comparing approaches, a common measure of performance is *relative error*, the ratio of the variance of the residuals to the variance of the target values themselves. Another useful statistic is the *correlation* between actual and predicted values (although a correlation coefficient of 1 indicates only that there is a linear relationship between actual and predicted values). Some authors report *percentage deviation*, the average over the cases of the ratio of the residual to the target value – this is not useful if the target values include 0!

In most trials, M5's performance was assessed in a 10-way cross-validation in which the available data was divided into ten equal-sized blocks. For each block in turn, a model was constructed using only cases in the remaining nine blocks, then tested on cases in the "hold-out" block. Each case was thus tested once using a model constructed without reference to the case.

All M5 results reported below were obtained using the same default values of parameters such as the minimum number of cases that can be split and the smoothing constant. To show the effects of forming models at the leaves and of smoothing, results are also given with these features disabled.

CHMIN $\leq$ 7 : RM0
CHMIN $>$ 7 :
- MMAX $>$ 24000 : RM1
- MMAX $\leq$ 24000 :
  - CACH $\leq$ 48 : RM2
  - CACH $>$ 48 : average 217.5

Model:	RM0	RM1	RM2
	11.5	-101.4	11.9
Cycle time			
Min mem		0.030	
Max mem	0.003		0.008
Cache size	0.902		
Min chans			
Max chans	0.518	4.686	

Figure 1: Model tree for CPU performance

	Original Attributes		Transformed Attributes	
	Correlation	Percentage Deviation	Correlation	Percentage Deviation
Ein–Dor (retrial)	–	–	.966	33.9%
IBP	–	35.0%	–	33.0%
M5	.921	34.9%	.956	34.0%
M5 (no smoothing)	.908	37.2%	.957	33.9%
M5 (no models)	.803	49.9%	.853	48.6%

Table 1: CPU performance data

3.1 CPU Performance

Ein-Dor and Feldmesser[2] carried out experiments to predict the relative performance of new CPUs as a function of six easily-determined variables: cache size, cycle time, minimum and maximum channels, and minimum and maximum memory. They did not use the attributes directly, but developed from them three composite attributes. Then, using 209 cases, they constructed a linear model of the square root of performance as a function of the three transformed attributes.

Kibler, Aha and Albert[4] studied this same data using an instance-based approach. IBP predicts a case's target value by interpolating from known values of similar cases. Results were reported for both the six original and three transformed attributes. IBP gave better predictive accuracy than Ein-Dor and Feldmesser's model without the need to resort to indirect methods such as predicting the square root of the target value.

When M5 was run on the 209 cases using the original attributes, it produced the compact model tree of Figure 1. All results with the original and transformed datasets are shown in Table 1. Note that reported accuracies for Ein-Dor and Feldmesser are for *retrial* performance on the training data from which the model was derived – their model would presumably give worse results on unseen cases. Predictive accuracies for IBP and M5 shown in the table were obtained by cross-validation – n-way for IBP and 10-way for M5. The accuracy of the models developed by M5 was comparable to that of IBP, but performance degraded on the original data when smoothing was disabled, and (markedly) on both datasets when the system was prevented from forming models at leaves.

	Correlation	Percentage Deviation	Relative Error
IBP	–	11.8%	–
M5	.916	12.7%	16.1%
M5 (no smoothing)	.885	14.2%	23.2%
M5 (no models)	.857	15.2%	27.6%

Table 2: Car price data

3.2 Car Prices

Kibler, Aha and Albert also evaluated IBP on a collection of 159 car descriptions. Each case has values for 16 attributes such as number of doors and engine size, with the target value being the car price. This task is considerably more challenging than the previous one, since it has higher dimensionality and yet there are fewer training cases. Table 2 shows results for both IBP and M5 on this data; M5 gave a higher percentage deviation than IBP, but the contribution of smoothing and linear models are both apparent.

3.3 An Artificial Dataset

Breiman *et al*[1] reported regression tree results for a small dataset affected by noise. In this study there are ten variables, X_1, ..., X_{10}; X_1 takes values -1 or 1 and the others -1, 0, or 1, all values being equiprobable. The generating function is

$$\text{target value} = \begin{cases} 3 + 3X_2 + 2X_3 + X_4 + \xi & \text{if } X_1 = 1 \\ -3 + 3X_5 + 2X_6 + X_7 + \xi & \text{if } X_1 = -1 \end{cases}$$

where ξ is a random Gaussian variable with variance 2, representing noise. From 200 training cases, CART constructed a regression tree with 13 leaves. When given a similar dataset, M5 produced a model tree with just two leaves (as we would hope). Table 3 compares published results from CART with experimental results from M5, both obtained by a 10-way cross-validation. The better performance of M5 on this data demonstrates clearly the benefit of allowing linear models, rather than just averages, at leaves.

3.4 Activity of Drugs

The last example concerns a very high-dimensional real-world task: predicting the

	Correlation	Relative Error
CART	–	17%
M5	.951	9.5%
M5 (no smoothing)	.955	8.9%
M5 (no models)	.914	16.6%

Table 3: An artificial task

	Correlation	Relative Error
M5	.887	21.4%
M5 (no smoothing)	.872	24.1%
M5 (no models)	.856	27.0%
linear regression	.805	38.5%

Table 4: Drug activity data

activity levels of LHRH peptides (strings of 10 amino acids). Data provided by Arris Pharmaceutical consists of 563 cases, each of which records the values of 128 attributes and a measured activity.

On a cross-validation trial, the model trees constructed by M5, with an average of 9.6 leaves, were still quite simple. The results in Table 4 show the contribution of both smoothing and linear models. For comparison, the Table also shows the performance of a standard multivariate linear model using the same cross-validation runs.

4. Conclusion

Model trees, like regression trees, can be learned efficiently from large datasets – the model tree for the car price data was constructed in less than a second, while that for the LHRH data (563 cases $\times$ 128 attributes) still required less than a minute on my DECstation 5000. However, model trees have advantages over regression trees in both compactness and prediction accuracy, attributable to the ability of model trees to exploit local linearity in the data. There is one other noteworthy difference – regression trees will never give a predicted value lying outside the range observed in the training cases, whereas model trees can extrapolate. (Whether this last is an advantage or a cause for concern is still an open question!)

M5 could be strengthened in many respects. I am exploring alternative heuristics to guide the selection of tests at internal nodes and different smoothing algorithms. Further, the models at the leaves could be generalised to allow non-linear functions of the attributes, albeit with an increase in computation.

Acknowledgements

Thanks to Arris Pharmaceutical of San Francisco for permission to use the LHRH data. This research was supported by a grant from the Australian Research Council and by a research agreement with Digital Equipment Corporation.

References

1. Breiman, L, Friedman, J.H., Olshen, R.A., and Stone, C.J., *Classification and Regression Trees* (Wadsworth, Belmont CA, 1984).
2. Ein-Dor, P. and Feldmesser, J., Attributes of the performance of central processing units: a relative performance prediction model, *Communications ACM 30, 4* (1987) 308-317.
3. Friedman, J.H., Multivariate Adaptive Regression Splines, Technical Report 102, Laboratory for Computational Statistics, Stanford University, Stanford CA (1988).
4. Kibler, D., Aha, D.W., and Albert, M.K., Instance-based prediction of real-valued attributes, Technical Report 88-07, ICS, University of California, Irvine CA (1988).
5. Pregibon, D., private communications, 1989, 1992.
6. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes in C* (Cambridge University Press, 1988).
7. Quinlan, J.R., *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Mateo CA, 1992).