*In Proceedings ML'93 (Utgoff, Ed), San Mateo, CA: Morgan Kaufmann, 1993*

# Combining Instance-Based and Model-Based Learning

**J. R. Quinlan**
Basser Department of Computer Science
University of Sydney
Sydney Australia 2006
quinlan@cs.su.oz.au

## Abstract

This paper concerns learning tasks that require the prediction of a continuous value rather than a discrete class. A general method is presented that allows predictions to use both instance-based and model-based learning. Results with three approaches to constructing models and with eight datasets demonstrate improvements due to the composite method.

**Keywords:** learning with continuous classes, instance-based learning, model-based learning, empirical evaluation.

## 1 INTRODUCTION

Among the many approaches to supervised learning, instance-based and model-based methods belong to two extremes. Instance-based approaches represent what has been learned as a collection of *prototypes* couched in the same language as that used to describe training data. A prototype might be just one of the training cases, or some hypothetical case computed from one or more of them (such as the weighted average of a set of cases). An unseen case is classified by finding similar prototypes and using their classes in some way to form a prediction. Different instance-based systems vary in how they assess the similarity of two cases, specify the number of similar prototypes to be used, and combine predictions from individual prototypes. Stanfill and Waltz [1986] and Aha, Kibler, and Albert [1991] provide examples of these mechanisms and discussions of the merits of instance-based learning.

Model-based approaches, on the other hand, represent what has been learned in some theory language that is richer than the language used to describe data. Learning methods of this kind construct explicit generalizations of the training cases, rather than allowing generalization to flow implicitly from the similarity measure above. When a task domain has many irrelevant attributes, an explicit generalization expressed by some model can be more intelligible than a set of prototype cases and can also lead to better predictions. Breiman, Friedman, Olshen and Stone [1984] and Quinlan [1993a] present arguments in favor of learning by constructing symbolic models.

This paper considers tasks of learning to predict numerical values, a form of supervised learning in which the classes are continuous rather than discrete. A general method for combining instance-based and model-based learning for such tasks is proposed. This method, which can be used with any form of model, is illustrated here using three of them – familiar *linear regression* [Press, Flannery, Teukolsky, and Vetterling, 1988], *model trees* [Quinlan, 1992], and *neural networks* [Hinton, 1986; McClelland and Rumelhart, 1988; Hinton, 1992]. Over a representative collection of datasets, the composite methods often produce better predictions than either instance-based or model-based approaches.

## 2 USING MODELS AND INSTANCES

We assume some set of training cases $T$. Each case consists of an attribute-value vector and a known class value, here a number rather than a category. Using $T$, some instance-based method assembles a set of prototypes $P$ and a model-based method constructs a predictive model $M$. For a case or prototype $C$, we will write $V(C)$ to represent the class value of $C$ and $M(C)$ to represent the value predicted for $C$ by $M$. (Note that, since prototypes are expressed in the same language as cases, we can apply the model to both cases and prototypes.)

Suppose now that we are to predict the class value for some unseen case $U$. Using the model-based approach we would get $M(U)$. If the instance-based approach were employed, a subset $\{P_1, P_2, ..., P_k\}$ of the prototype points would first be identified as being similar to $U$; the values $\{V(P_1), V(P_2), ..., V(P_k)\}$ would then be combined in some way to give the predicted value of $U$.

Consider one of the selected prototypes, $P_i$ say. The rationale for making use of $V(P_i)$ in computing a predicted value for $U$ is that $P_i$ is similar to $U$, so $V(P_i)$ should be similar to $V(U)$. Prototype $P_i$ will not generally be identical to $U$, but the nature of the difference between them is not taken into account, even if some measure of their similarity is used to determine the weight of $V(P_i)$ in the combined prediction.

The model $M$, however, provides a way of taking explicit account of the difference between the unseen case and the prototype. Specifically, the model $M$ predicts the difference between the class values of $P_i$ and $U$ to be

$$M(P_i) - M(U).$$

If the model is correct, the adjusted value

$$V(P_i) - (M(P_i) - M(U))$$

should be a better predictor of the class value of $U$ than the quantity $V(P_i)$ alone. The composite method differs from the instance-based method only in carrying out this adjustment to the class values of selected prototypes before they are combined to give a prediction.

## 3 EMPIRICAL EVALUATION

This straightforward idea for combining instance-based and model-based learning has been evaluated using a simple instance-based paradigm, three different forms of model-based learning, and eight learning tasks. Short descriptions of each follow.

### 3.1 INSTANCE-BASED LEARNING

The instance-based approach used here is based on IBL [Kibler, Aha, and Albert, 1988]. Prototypes consist of all training cases without any modification. The dissimilarity of two cases is computed by summing their normalized differences over each attribute, defined as

- for discrete attributes: 0 if the attribute values are the same, 1 otherwise; and
- for continuous-valued attributes: the ratio of the absolute difference between the values divided by value range of the attribute.

To classify an unknown case, the three most similar prototypes are found and their associated class values averaged. (When the instance-based approach is combined with a model-based approach, the class values are adjusted before their mean is computed.)

### 3.2 REGRESSION

The first of the methods for constructing predictive models is the ubiquitous multivariate linear regression [Press *et al*, 1988] that is often used as the starting point for statistical data analysis. Let $C_i$ denote the value of the $i$th attribute for case $C$, or the rank of this value if the $i$th attribute is discrete (nominal). We assume a model of the form

$$M(C) = \alpha_0 + \sum_i \alpha_i \times C_i$$

and find the coefficients $\{\alpha_i\}$ to minimize the sum of the squares of the differences between the actual and predicted values for the training cases. These coefficients can be found by inverting a matrix, with some minor complications when the matrix is singular.

### 3.3 MODEL TREES

The second is a system called M5 that uses recursive partitioning to build a piecewise linear model in the form of a model tree [Quinlan, 1992]. The idea is to split the training cases in much the same way as when growing a decision tree, using a criterion of minimizing intra-subset variation of class values rather than maximizing information gain. Whereas a leaf of a decision tree contains just a class name, the corresponding leaf of a model tree is a linear model relating the class values of the training cases to their attribute values. *Regression trees* [Breiman *et al*, 1984] are based on a similar divide-and-conquer strategy, but have values rather than linear models at the leaves.

Consider a set $T$ of training cases for which a model tree is to be constructed. Unless $T$ contains few cases or their values vary only slightly, it is split according to 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 $i$th 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 chooses one that maximizes this expected error reduction. (For comparison, CART [Breiman *et al*, 1984] 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:

*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, M5 first determines the average residual of the model on the training cases used to construct it. 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 $\nu$ 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 linear model is constructed for the cases at each node of the model tree using standard regression techniques. This model, however, is restricted to the attributes that are referenced by some test or linear model 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*: Each linear model is then simplified by eliminating parameters so as to minimize 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 internal node of the tree now has both a simplified model and a model subtree. The one of these with lower estimate error is chosen; if this is the linear model, the subtree at this node has been pruned to a leaf.

*Smoothing*: Pregibon [private communications, 1989, 1992] 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 returned by the model at the appropriate leaf is adjusted to take account of models 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. The predicted value is backed up from the leaf to the root as follows:

- The predicted value at the leaf is unchanged.
- 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[1].

[1]The default value of 15 was used in all experiments.

Smoothing has most effect when leaf models are constructed from few training cases and do not agree with models higher in the tree.

## 3.4 NEURAL NETS

The third form of model is the neural network. Geoffrey Hinton very kindly agreed to develop the network models used in these experiments, as described below:

"The neural networks all contained one layer of hidden units that used the logistic function to convert their combined input into their output. The single output unit of each network was linear and the error function was the sum of the squared differences between the actual and desired outputs. All input and output values for each training set were normalized to have zero mean and unit variance and the same normalization (based on the training set) was used for the test set.

"To reduce overfitting, the number of hidden units was kept small and, in addition, a penalty was imposed on the squared values of the weights. So the overall cost function that was minimized during training was the sum of the error function and a penalty coefficient times the sum of the squared weights.

"The precise number of hidden units and the coefficient of the penalty term were different for each task. To decide the values of these two parameters, one third of one of the training sets was held-out as a validation set. The remaining two thirds of the training set was then used to train many different networks containing different numbers of hidden units and different penalty coefficients. For each task, the largest number of hidden units explored was the number that gave about 2 training cases per connection, and the smallest number explored was the one that gave about 5 training cases per connection. Within this range, up to 6 different numbers of hidden units were tried. The penalty coefficients explored were 0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08, 0.1, 0.12, 0.16, and 0.2. For each setting of these two parameters, the network was trained 3 times starting with different random weights uniformly distributed in the range -0.3 to +0.3. The parameters that gave the smallest average error on the validation set were selected. Ideally, this whole procedure should have been repeated for each of the 10 training sets used for each task, but to save time, the parameters were determined from the first training set of each task.

"The training was performed using the default conjugate gradient method of the Xerion neural network simulator developed at the University of Toronto. The user does not need to specify any parameters for this method. Training is terminated when the error decreases by less than $10^{-6}$." [Hinton, private communication, 1993]

## 3.5 DATASETS

The four "pure" methods and three composites were tested on eight datasets drawn from seven domains that were selected to provide a range of learning challenges:

- *pw-linear:* (200 cases)
  This is a piecewise linear function defined by Breiman *et al* [1984]. There are 10 attributes, $X_1$ having equiprobable values -1 and 1, and $X_2$ to $X_{10}$ having equiprobable values -1, 0, and 1. The generating function is
  $$V(X) = \begin{cases} 3X_2 + 2X_3 + X_4 + 3 + Z & \text{if } X_1{=}1 \\ 3X_5 + 2X_6 + X_7 - 3 + Z & \text{if } X_1{=}{-1} \end{cases}$$
  where $Z$ is a random Gaussian noise term with variance 2. Attributes $X_8$, $X_9$ and $X_{10}$ have no bearing on the class value.
- *housing:* (506 cases)
  This dataset, obtained from the Statistics library maintained by Carnegie Mellon University, concerns housing values in suburbs of Boston. There are 12 continuous attributes and one binary-valued attribute.
- *cpu:* (209 cases)
  Ein-Dor and Feldmesser [1987] published this dataset that relates the measured performance of CPUs to six continuous-valued and one multi-valued discrete attribute (the vendor). Although the paper also develops a new set of derived features, the attributes used in these experiments are the original parameters of the cpu such as minimum and maximum memory size.
- *auto-price:* (159 cases)
  The class value for this dataset is the 1985 list price of common automobiles. There are 16 continuous attributes covering quantities such as the automobile's size, weight and engine capacity.
- *auto-mpg:* (398 cases)
  This is another dataset from the CMU Statistics library. The data concern city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multi-valued discrete and 5 continuous attributes. (The dataset was used as the testbed for graphical analysis packages at the 1983 American Statistical Association Exposition.)
- *servo:* (167 cases)
  This interesting collection of data, provided by Karl Ulrich, refers to an extremely non-linear phenomenon – predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages.
- *lhrh:* (526 cases)
  This final domain concerns a very difficult task – predicting the biomedical activity of LHRH peptides that are chains of exactly 10 amino acids. The information was obtained from Arris Pharmaceutical Corporation of San Francisco, who used an extensive table compiled by Dr Marvin Karten of the U.S. National Institutes of Health. This domain supplies two datasets:
  - *lhrh-att:* Each peptide is described in terms of 48 binary and 80 continuous attributes that measure properties of the individual amino acids and properties of the peptide as a whole. This dataset provides a very high-dimensional test of all the methods.
  - *lhrh-def*: Each peptide is described only by the names of its constituent amino acids. There are thus 10 discrete-valued attributes, all of which have very many values – about 400 amino acids occur at one or more places in the 526 peptides. Recall that linear models and neural networks use the rank of discrete attribute values. Since the amino acid names are ordered randomly, this dataset provides little joy to model-based learning systems!

Results have previously been reported for at least four of these domains. Breiman *et al* [1984] give performance figures for CART on *pw-linear* and *housing*; Ein-Dor and Feldmesser [1987] and Kibler *et al* [1988] describe experiments with *cpu*; and the latter paper also discusses *auto-price*.

## 3.6 EXPERIMENTS AND RESULTS

Each dataset was analysed by each method using the same 10-way cross-validation. The dataset was divided into ten blocks of near-equal size and distribution of class values. For every block in turn, each method was trained on the remaining blocks and tested on the hold-out block. Results, averaged over all test blocks, thus reflect predictive performance on unseen cases.

The first measure of performance is average error magnitude (or residual) on unseen cases, summarized in Table 1. Each column is associated with one dataset, each row with one method. The first row refers to the simple *default* method that always predicts the average value of the training set; this serves as a reference point for the underlying difficulty of the prediction tasks. The next rows show the results for instances alone and then for pairs of pure models and composite methods.

The same information is presented graphically in Figure 1, where the average error magnitudes have been normalized by dividing by the corresponding magnitude given by the default procedure.

Because all methods were evaluated with the same cross-validation partitions, each trial using exactly the same training and test cases, the sensitive one-tailed paired test of significance can be used. For every dataset, we are interested in seeing how each composite

**Table 1:** Average error on unseen cases

| | pw-linear | housing | cpu | auto-price | auto-mpg | servo | lhrh-att | lhrh-def |
|---|---|---|---|---|---|---|---|---|
| default procedure | 3.77 | 6.65 | 96.0 | 4578 | 6.53 | 1.15 | 2.28 | 2.28 |
| instances alone | 1.73 | 2.90 | 34.0 | 1689 | 2.72 | 0.52 | 0.91 | 0.85 |
| regression | 2.13 | 3.29 | 35.5 | 1848 | 2.61 | 0.86 | 1.10 | 2.01 |
| regression + instances | 1.56 | 2.45 | 30.0 | 1430 | 2.37 | 0.48 | 0.97 | 1.22 |
| model trees | 1.10 | 2.45 | 28.9 | 1562 | 2.11 | 0.45 | 0.96 | 1.29 |
| model trees + instances | 1.21 | 2.32 | 28.1 | 1386 | 2.18 | 0.30 | 0.88 | 0.97 |
| neural nets | 1.14 | 2.29 | 28.7 | 1833 | 2.02 | 0.30 | 1.07 | 1.16 |
| neural nets + instances | 1.29 | 2.23 | 29.0 | 1677 | 2.06 | 0.29 | 0.99 | 1.04 |

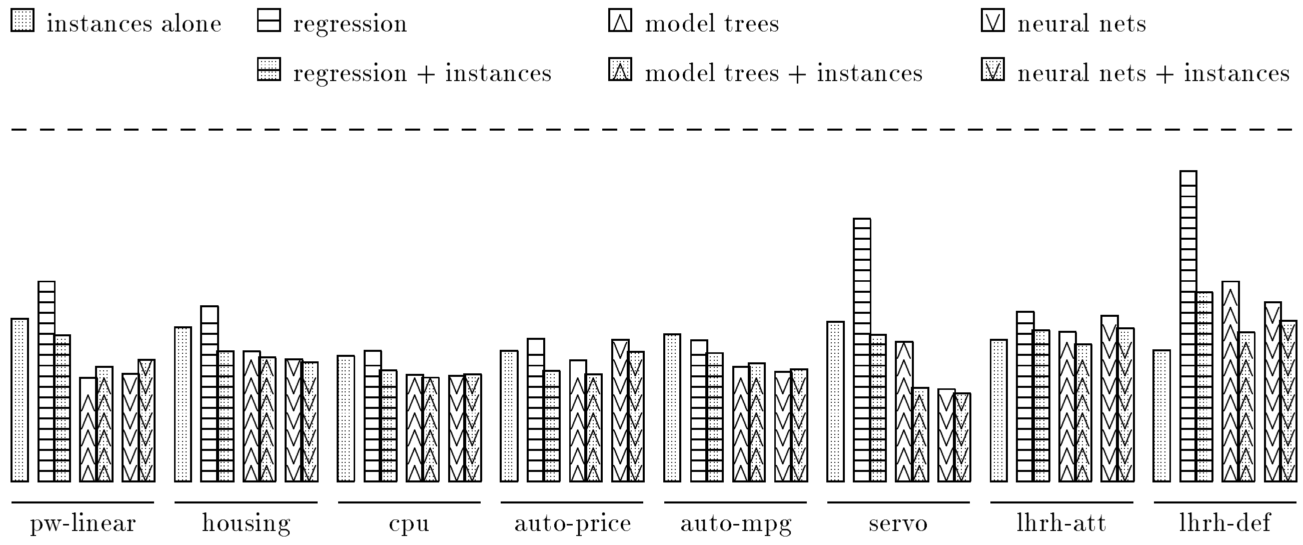

**Figure 1:** Average error on unseen cases

method compares to the two pure methods from which it is constituted, e.g. how *model trees + instances* compares with *instances* alone and with *model trees* alone. This gives six comparisons for each dataset, or 48 comparisons in all.

At the 5% significance level, the composite methods are significantly better than the constituent pure methods in 31 comparisons, about the same in twelve comparisons, and significantly worse in only five comparisons. The five cases in which composite models are significantly worse are:

- In the *pw-linear* task, model trees are superior to *model trees + instances* and neural networks are superior to *neural nets + instances*. Both the model trees and neural nets find close approximations to the true underlying function (minus the noise term). The use of prototypes with a near-perfect model causes a slight increase in error, since the noise is absent from the model but present in the prototype cases.
- The other three cases all concern the *lhrh-def* task, instances alone being superior to all three composite methods. For this dataset, where attribute values are just the names of amino acids, all models are (understandably) very weak and their use to adjust prototype estimators introduces substantial errors.

On the positive side, the composite methods often score big wins. The improved accuracy of the *regression + instances* composite over simple regression is uniformly (and dramatically) apparent on all datasets. The *servo* dataset also highlights the effectiveness of

**Table 2:** Relative error (%) on unseen cases

| | pw-linear | housing | cpu | auto-price | auto-mpg | servo | lhrh-att | lhrh-def |
|---|---|---|---|---|---|---|---|---|
| instances alone | 21.7 | 22.7 | 20.0 | 26.0 | 23.0 | 25.6 | 20.7 | 17.5 |
| regression | 33.5 | 29.4 | 19.7 | 20.1 | 19.4 | 49.2 | 31.5 | 84.5 |
| regression + instances | 18.3 | 16.8 | 11.8 | 14.8 | 17.3 | 19.7 | 26.9 | 37.8 |
| model trees | 9.5 | 18.6 | 17.2 | 16.2 | 14.7 | 28.7 | 23.6 | 39.6 |
| model trees + instances | 11.7 | 16.5 | 12.0 | 12.8 | 16.0 | 16.5 | 21.0 | 25.4 |
| neural nets | 10.1 | 13.6 | 11.0 | 28.5 | 12.5 | 11.4 | 28.3 | 32.4 |
| neural nets + instances | 12.6 | 12.9 | 11.1 | 25.7 | 13.4 | 10.6 | 25.4 | 27.5 |

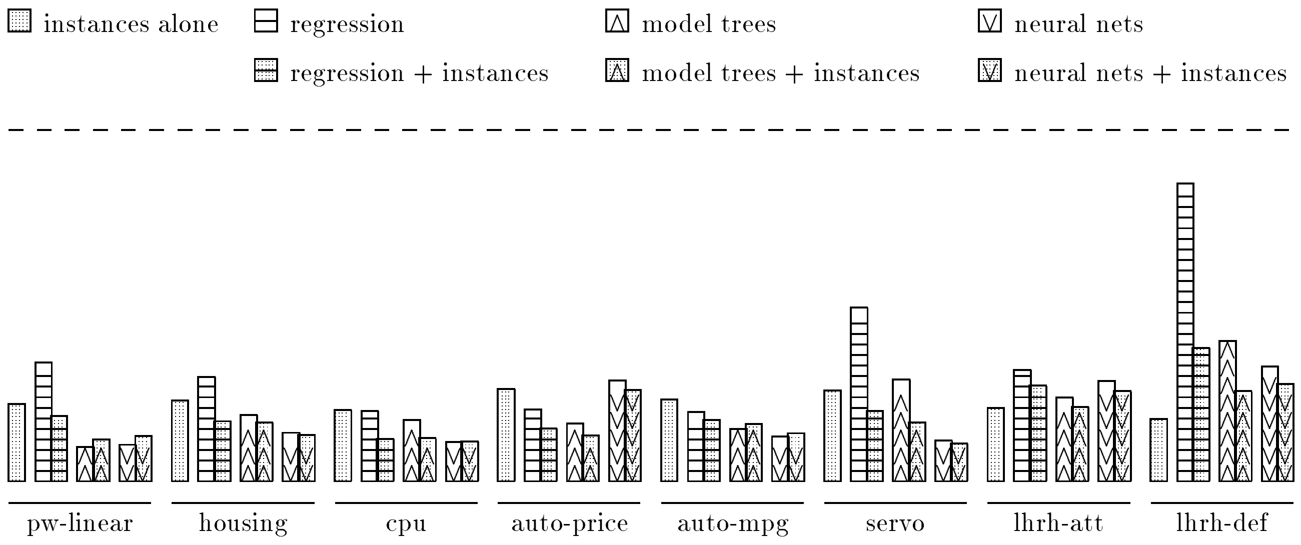

**Figure 2:** Relative error on unseen cases

the composite methods: average error is nearly halved in the case of regression, is reduced by 33% for model trees, and is reduced slightly even for networks (where the pure method is already doing extremely well). In the few cases for which the composite models are inferior to one or other of their constituent models, the difference in accuracy is usually small.

The second performance measure investigated is *relative error* [Breiman *et al*, 1984], defined as the variance of the residuals on unseen cases, divided by the variance of the class values themselves. (Despite its name, this statistic thus measures squared error.) As with the normalized metric used in Figure 1, the value is 100% for any method that always predicts the mean value. Relative errors are presented in Table 2 and repeated in graph form in Figure 2. The results show a similar pattern, although the error-squared measure accentuates differences in some domains.

**Table 3**: Average relative error (%) across tasks

| | |
|---|---|
| instances alone | 22.2 |
| regression | 35.9 |
| regression + instances | 20.4 |
| model trees | 21.0 |
| model trees + instances | 16.5 |
| neural nets | 18.5 |
| neural nets + instances | 17.4 |

As a rough indicator of overall performance, the relative errors for each method were averaged over all eight tasks (Table 3).

Finally, there might be some concern that the use of an unsophisticated instance-based paradigm could have distorted these rather positive results. However, per-

**Table 4:** Comparison of relative error

| | model trees + instances | CART | IBL |
|---|---|---|---|
| *pw-linear* | 12% | 17% | |
| *housing* | 16% | 22% | |
| *cpu* | 11% | | 23% |
| *auto-price* | 11% | | 19% |

formance of the various methods compares favorably with results from other systems. For example, Table 4 shows relative error of the *model trees + instances* composite compared with published results from CART on the first two domains and with computed results from IBL on the next two. The composite fares well on this comparison, and might achieve even better accuracy if it were changed to incorporate more advanced instance-based methods such as those described by Aha *et al* [1991].

## 4 CONCLUSION

This paper has described a simple method for combining instance-based and model-based learning for tasks that involve the prediction of values. The advantages of the approach have been illustrated using three disparate model-learning methods and a variety of domains. The approach is quite general and should be applicable to any instance-based method and any form of learned model.

The method achieves its power through using the same training data to provide local information (in the form of prototypes) and a global model. Both kinds of knowledge are brought to bear when the class value of an unseen case is predicted. The method seems to provide robust improvement, with two exceptions:

- If the model is extremely weak, as was the case with the *lhrh-def* dataset, it is advisable to use the instances alone.
- If the model is near-perfect but there is unavoidable noise, as with model trees for the *pw-linear* domain, the use of local information might introduce additional error.

I have also experimented with a similar composite approach in which the model attempts to predict differences directly [Quinlan, 1993b]. For domains in which such a difference model can be found, including *servo* and *lhrh-def* among the present datasets, this approach produces predictors that are better still.

**Acknowledgements**

This research has been made possible by grants from the Australian Research Council and assisted by a research agreement with Digital Equipment Corporation. I am extremely grateful to Geoff Hinton, who designed, trained, and evaluated the 80 neural nets used in these experiments. Donald Michie made several helpful suggestions, especially the idea of trying composite regression models. The paper was improved as a result of comments by Mike Cameron-Jones.

Thanks to Dr Marvin Karten, National Institutes of Health, Bethesda MD, and to Arris Pharmaceutical Corporation, San Francisco, for access to the LHRH peptide data, and to Karl Ulrich for the servomechanism dataset. I also gratefully acknowledge using datasets from the UCI Data Repository and StatLib at CMU.

## References

Aha, D.W., Kibler, D., and Albert, M.K. (1991), Instance-based learning algorithms, *Machine Learning 6*, 1, 37-66.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984), *Classification and regression trees*, Belmont, California: Wadsworth.

Ein-Dor, P. and Feldmesser, J. (1987), Attributes of the performance of central processing units: a relative performance prediction model, *Communications of the ACM 30*, 4, 308-317.

Hinton, G.E. (1986), Learning distributed representations of concepts, *Proceedings Eighth Annual Conference of the Cognitive Science Society*, Amherst. Reprinted in R.G.M. Morris (Ed), *Parallel Distributed Processing: Implications for Psychology and Neurobiology*, Oxford University Press.

Hinton, G.E. (1992), How neural networks learn from experience, *Scientific American 267*, 3, (September 1992), 144-151.

Kibler, D., Aha, D.W., and Albert, M.K. (1988), Instance-based prediction of real-valued attributes, Technical Report 88-07, ICS, University of California, Irvine.

McClelland, J.L., and Rumelhart, D.E. (1988), *Explorations in Parallel Distributed Processing*, Cambridge: MIT Press.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. (1988), *Numerical Recipes in C*, Cambridge University Press.

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

Quinlan, J.R. (1993a), *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann.

Quinlan, J.R. (1993b), A case study in machine learning, *Proceedings 16th Australian Computer Science Conference*, Brisbane, 731-737.

Stanfill, C. and Waltz, D. (1986), Toward memory-based reasoning, *Communications of the ACM 29*, 12, 1213-1228.