Bagging, Boosting, and C4.5

J. R. Quinlan
University of Sydney
Sydney, Australia 2006
quinlan@cs.su.oz.au

Abstract

Breiman's *bagging* and Freund and Schapire's *boosting* are recent methods for improving the predictive power of classifier learning systems. Both form a set of classifiers that are combined by voting, bagging by generating replicated bootstrap samples of the data, and boosting by adjusting the weights of training instances. This paper reports results of applying both techniques to a system that learns decision trees and testing on a representative collection of datasets. While both approaches substantially improve predictive accuracy, boosting shows the greater benefit. On the other hand, boosting also produces severe degradation on some datasets. A small change to the way that boosting combines the votes of learned classifiers reduces this downside and also leads to slightly better results on most of the datasets considered.

Introduction

Designers of empirical machine learning systems are concerned with such issues as the computational cost of the learning method and the accuracy and intelligibility of the theories that it constructs. Much of the research in learning has tended to focus on improved predictive accuracy, so that the performance of new systems is often reported from this perspective. It is easy to understand why this is so – accuracy is a primary concern in all applications of learning and is easily measured (as opposed to intelligibility, which is more subjective), while the rapid increase in computers' performance/cost ratio has de-emphasized computational issues in most applications.[1]

In the active subarea of learning decision tree classifiers, examples of methods that improve accuracy are:

- Construction of multi-attribute tests using logical combinations (Ragavan and Rendell 1993), arithmetic combinations (Utgoff and Brodley 1990; Heath, Kasif, and Salzberg 1993), and counting operations (Murphy and Pazzani 1991; Zheng 1995).
- Use of error-correcting codes when there are more than two classes (Dietterich and Bakiri 1995).
- Decision trees that incorporate classifiers of other kinds (Brodley 1993; Ting 1994).
- Automatic methods for setting learning system parameters (Kohavi and John 1995).

On typical datasets, all have been shown to lead to more accurate classifiers at the cost of additional computation that ranges from modest to substantial.

There has recently been renewed interest in increasing accuracy by generating and aggregating multiple classifiers. Although the idea of growing multiple trees is not new (see, for instance, (Quinlan 1987; Buntine 1991)), the justification for such methods is often empirical. In contrast, two new approaches for producing and using several classifiers are applicable to a wide variety of learning systems and are based on theoretical analyses of the behavior of the composite classifier.

The data for classifier learning systems consists of attribute-value vectors or *instances*. Both bootstrap aggregating or *bagging* (Breiman 1996) and *boosting* (Freund and Schapire 1996a) manipulate the training data in order to generate different classifiers. Bagging produces replicate training sets by sampling with replacement from the training instances. Boosting uses all instances at each repetition, but maintains a *weight* for each instance in the training set that reflects its importance; adjusting the weights causes the learner to focus on different instances and so leads to different classifiers. In either case, the multiple classifiers are then combined by voting to form a composite classifier. In bagging, each component classifier has the same vote, while boosting assigns different voting strengths to component classifiers on the basis of their accuracy.

This paper examines the application of bagging and boosting to C4.5 (Quinlan 1993), a system that learns decision tree classifiers. After a brief summary of both methods, comparative results on a substantial number of datasets are reported. Although boosting generally increases accuracy, it leads to a deterioration on

[1] For extremely large datasets, however, learning time can remain the dominant issue (Catlett 1991; Chan and Stolfo 1995).

some datasets; further experiments probe the reason for this. A small change to boosting in which the voting strengths of component classifiers are allowed to vary from instance to instance shows still further improvement. The final section summarizes the (sometimes tentative) conclusions reached in this work and outlines directions for further research.

Bagging and Boosting

We assume a given set of N instances, each belonging to one of K classes, and a learning system that constructs a classifier from a training set of instances. Bagging and boosting both construct multiple classifiers from the instances; the number T of repetitions or *trials* will be treated as fixed, although Freund and Schapire (1996a) note that this parameter could be determined automatically by cross-validation. The classifier learned on trial t will be denoted as C^t while C^* is the composite (bagged or boosted) classifier. For any instance x, $C^t(x)$ and $C^*(x)$ are the classes predicted by C^t and C^* respectively.

Bagging

For each trial t = 1,2,...,T, a training set of size N is sampled (with replacement) from the original instances. This training set is the same size as the original data, but some instances may not appear in it while others appear more than once. The learning system generates a classifier C^t from the sample and the final classifier C^* is formed by aggregating the T classifiers from these trials. To classify an instance x, a vote for class k is recorded by every classifier for which $C^t(x) = k$ and $C^*(x)$ is then the class with the most votes (ties being resolved arbitrarily).

Using CART (Breiman, Friedman, Olshen, and Stone 1984) as the learning system, Breiman (1996) reports results of bagging on seven moderate-sized datasets. With the number of replicates T set at 50, the average error of the bagged classifier C^* ranges from 0.57 to 0.94 of the corresponding error when a single classifier is learned. Breiman introduces the concept of an *order-correct* classifier-learning system as one that, over many training sets, tends to predict the correct class of a test instance more frequently than any other class. An order-correct learner may not produce optimal classifiers, but Breiman shows that aggregating classifiers produced by an order-correct learner results in an optimal classifier. Breiman notes:

> "The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy."

Boosting

The version of boosting investigated in this paper is AdaBoost.M1 (Freund and Schapire 1996a). Instead of drawing a succession of independent bootstrap samples from the original instances, boosting maintains a weight for each instance – the higher the weight, the more the instance influences the classifier learned. At each trial, the vector of weights is adjusted to reflect the performance of the corresponding classifier, with the result that the weight of misclassified instances is increased. The final classifier also aggregates the learned classifiers by voting, but each classifier's vote is a function of its accuracy.

Let w^t_x denote the weight of instance x at trial t where, for every x, $w^1_x = 1/N$. At each trial t = 1,2,...,T, a classifier C^t is constructed from the given instances under the distribution w^t (i.e., as if the weight w^t_x of instance x reflects its probability of occurrence). The error ϵ^t of this classifier is also measured with respect to the weights, and consists of the sum of the weights of the instances that it misclassifies. If ϵ^t is greater than 0.5, the trials are terminated and T is altered to t-1. Conversely, if C^t correctly classifies all instances so that ϵ^t is zero, the trials terminate and T becomes t. Otherwise, the weight vector w^{t+1} for the next trial is generated by multiplying the weights of instances that C^t classifies correctly by the factor $\beta^t = \epsilon^t/(1-\epsilon^t)$ and then renormalizing so that $\sum_x w^{t+1}_x$ equals 1. The boosted classifier C^* is obtained by summing the votes of the classifiers C^1,C^2,...,C^T, where the vote for classifier C^t is worth $\log(1/\beta^t)$ units.

Provided that ϵ^t is always less than 0.5, Freund and Schapire prove that the error rate of C^* on the given examples under the original (uniform) distribution w^1 approaches zero exponentially quickly as T increases. A succession of "weak" classifiers $\{C^t\}$ can thus be boosted to a "strong" classifier C^* that is at least as accurate as, and usually much more accurate than, the best weak classifier on the training data, Of course, this gives no guarantee of C^*'s *generalization* performance on unseen instances; Freund and Schapire suggest the use of mechanisms such as Vapnik's (1983) *structural risk minimization* to maximize accuracy on new data.

Requirements for Boosting and Bagging

These two methods for utilizing multiple classifiers make different assumptions about the learning system. As above, bagging requires that the learning system should not be "stable", so that small changes to the training set should lead to different classifiers. Breiman also notes that "poor predictors can be transformed into worse ones" by bagging.

Boosting, on the other hand, does not preclude the use of learning systems that produce poor predictors, provided that their error on the given distribution can be kept below 50%. However, boosting implicitly requires the same instability as bagging; if C^t is the same as C^{t-1}, the weight adjustment scheme has the property that $\epsilon^t = 0.5$. Although Freund and Schapire's specification of AdaBoost.M1 does not force termination when $\epsilon^t = 0.5$, $\beta^t = 1$ in this case so that $w^{t+1} = w^t$ and all classifiers from C^t on have votes with zero

	C4.5	Bagged C4.5 vs C4.5			Boosted C4.5 vs C4.5			Boosting vs Bagging	
	err (%)	err (%)	w-l	ratio	err (%)	w-l	ratio	w-l	ratio
anneal	7.67	6.25	10-0	.814	4.73	10-0	.617	10-0	.758
audiology	22.12	19.29	9-0	.872	15.71	10-0	.710	10-0	.814
auto	17.66	19.66	2-8	1.113	15.22	9-1	.862	9-1	.774
breast-w	5.28	4.23	9-0	.802	4.09	9-0	.775	7-2	.966
chess	8.55	8.33	6-2	.975	4.59	10-0	.537	10-0	.551
colic	14.92	15.19	0-6	1.018	18.83	0-10	1.262	0-10	1.240
credit-a	14.70	14.13	8-2	.962	15.64	1-9	1.064	0-10	1.107
credit-g	28.44	25.81	10-0	.908	29.14	2-8	1.025	0-10	1.129
diabetes	25.39	23.63	9-1	.931	28.18	0-10	1.110	0-10	1.192
glass	32.48	27.01	10-0	.832	23.55	10-0	.725	9-1	.872
heart-c	22.94	21.52	7-2	.938	21.39	8-0	.932	5-4	.994
heart-h	21.53	20.31	8-1	.943	21.05	5-4	.978	3-6	1.037
hepatitis	20.39	18.52	9-0	.908	17.68	10-0	.867	6-1	.955
hypo	.48	.45	7-2	.928	.36	9-1	.746	9-1	.804
iris	4.80	5.13	2-6	1.069	6.53	0-10	1.361	0-8	1.273
labor	19.12	14.39	10-0	.752	13.86	9-1	.725	5-3	.963
letter	11.99	7.51	10-0	.626	4.66	10-0	.389	10-0	.621
lymphography	21.69	20.41	8-2	.941	17.43	10-0	.804	10-0	.854
phoneme	19.44	18.73	10-0	.964	16.36	10-0	.842	10-0	.873
segment	3.21	2.74	9-1	.853	1.87	10-0	.583	10-0	.684
sick	1.34	1.22	7-1	.907	1.05	10-0	.781	9-1	.861
sonar	25.62	23.80	7-1	.929	19.62	10-0	.766	10-0	.824
soybean	7.73	7.58	6-3	.981	7.16	8-2	.926	8-1	.944
splice	5.91	5.58	9-1	.943	5.43	9-0	.919	6-4	.974
vehicle	27.09	25.54	10-0	.943	22.72	10-0	.839	10-0	.889
vote	5.06	4.37	9-0	.864	5.29	3-6	1.046	1-9	1.211
waveform	27.33	19.77	10-0	.723	18.53	10-0	.678	8-2	.938
average	*15.66*	*14.11*		*.905*	*13.36*		*.847*		*.930*

Table 1: Comparison of C4.5 and its bagged and boosted versions.

weight in the final classification. Similarly, an overfitting learner that produces classifiers in total agreement with the training data would cause boosting to terminate at the first trial.

Experiments

C4.5 was modified to produce new versions incorporating bagging and boosting as above. (C4.5's facility to deal with fractional instances, required when some attributes have missing values, is easily adapted to handle the instance weights w_x^t used by boosting.) These versions, referred to below as *bagged C4.5* and *boosted C4.5*, have been evaluated on a representative collection of datasets from the UCI Machine Learning Repository. The 27 datasets, summarized in the Appendix, show considerable diversity in size, number of classes, and number and type of attributes.

The parameter T governing the number of classifiers generated was set at 10 for these experiments. Breiman (1996) notes that most of the improvement from bagging is evident within ten replications, and it is interesting to see the performance improvement that can be bought by a single order of magnitude increase in computation. All C4.5 parameters had their default values, and pruned rather than unpruned trees were used to reduce the chance that boosting would terminate prematurely with ϵ^t equal to zero. Ten complete 10-fold cross-validations were carried out with each dataset.[2]

The results of these trials appear in Table 1. For each dataset, the first column shows C4.5's mean error rate over the ten cross-validations. The second section contains similar results for bagging, i.e., the class of a test instance is determined by voting multiple C4.5 trees, each obtained from a bootstrap sample as above. The next figures are the number of complete cross-validations in which bagging gives better or worse results respectively than C4.5, ties being omitted. This section also shows the ratio of the error rate using bagging to the error rate using C4.5 – a value

[2] In a 10-fold (stratified) cross-validation, the training instances are partitioned into 10 equal-sized blocks with similar class distributions. Each block in turn is then used as test data for the classifier generated from the remaining nine blocks.

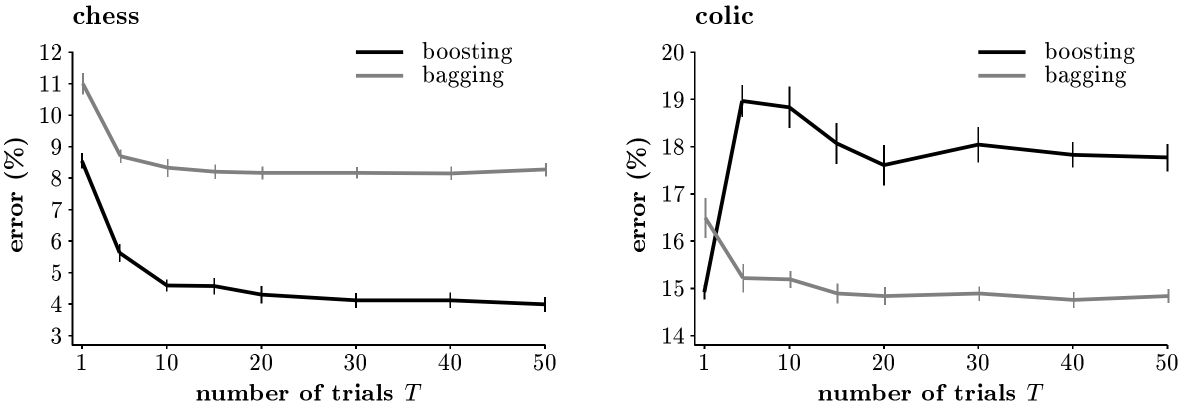

Figure 1: Comparison of bagging and boosting on two datasets

less than 1 represents an improvement due to bagging. Similar results for boosting are compared to C4.5 in the third section and to bagging in the fourth.

It is clear that, over these 27 datasets, both bagging and boosting lead to markedly more accurate classifiers. Bagging reduces C4.5's classification error by approximately 10% on average and is superior to C4.5 on 24 of the 27 datasets. Boosting reduces error by 15%, but improves performance on 21 datasets and degrades performance on six. Using a two-tailed sign test, both bagging and boosting are superior to C4.5 at a significance level better than 1%.

When bagging and boosting are compared head to head, boosting leads to greater reduction in error and is superior to bagging on 20 of the 27 datasets (significant at the 2% level). The effect of boosting is more erratic, however, and leads to a 36% increase in error on the iris dataset and 26% on colic. Bagging is less risky: its worst performance is on the auto dataset, where the error rate of the bagged classifier is 11% higher than that of C4.5.

The difference is highlighted in Figure 1, which compares bagging and boosting on two datasets, chess and colic, as a function of the number of trials T. For T=1, boosting is identical to C4.5 and both are almost always better than bagging – they use all the given instances while bagging employs a sample of them with some omissions and some repetitions. As T increases, the performance of bagging usually improves, but boosting can lead to a rapid degradation (as in the colic dataset).

Why Does Boosting Sometimes Fail?

A further experiment was carried out in order to better understand why boosting sometimes leads to a deterioration in generalization performance. Freund and Schapire (1996a) put this down to overfitting – a large number of trials T allows the composite classifier C^* to become very complex.

As discussed earlier, the objective of boosting is to construct a classifier C^* that performs well on the training data even when its constituent classifiers C^t are weak. A simple alteration attempts to avoid overfitting by keeping T as small as possible without impacting this objective. AdaBoost.M1 stops when the error of any C^t drops to zero, but does not address the possibility that C^* might correctly classify all the training data even though no C^t does. Further trials in this situation would seem to offer no gain – they will increase the complexity of C^* but cannot improve its performance on the training data.

The experiments of the previous section were repeated with T=10 as before, but adding this further condition for stopping before all trials are complete. In many cases, C4.5 requires only three boosted trials to produce a classifier C^* that performs perfectly on the training data; the average number of trials over all datasets is now 4.9. Despite using fewer trials, and thus being less prone to overfitting, C4.5's generalization performance is worse. The overfitting avoidance strategy results in lower cross-validation accuracy on 17 of the datasets, higher on six, and unchanged on four, a degradation significant at better than the 5% level. Average error over the 27 datasets is 13% higher than that reported for boosting in Table 1.

These results suggest that the undeniable benefits of boosting are not attributable just to producing a composite classifier C^* that performs well on the training data. It also calls into question the hypothesis that overfitting is sufficient to explain boosting's failure on some datasets, since much of the benefit realized by boosting seems to be caused by overfitting.

Changing the Voting Weights

Freund and Schapire (1996a) explicitly consider the use by AdaBoost.M1 of confidence estimates provided by some learning systems. When instance x is classified by C^t, let $H^t(x)$ be a number between 0 and 1 that represents some informal measure of the reliability of the prediction $C^t(x)$. Freund and Schapire suggest using this estimate to give a more flexible measure of classifier error.

An alternative use of the confidence estimate H^t is in combining the predictions of the classifiers $\{C^t\}$ to give the final prediction $C^*(x)$ of the class of instance x. Instead of using the fixed weight $\log(1/\beta^t)$ for the vote of classifier C^t, it seems plausible to allow the voting weight of C^t to vary in response to the confidence with which x is classified.

C4.5 can be "tweaked" to yield such a confidence estimate. If a single leaf is used by C^t to classify an instance x as belonging to class $k{=}C^t(x)$, let S denote the set of training instances that are mapped to the leaf, and S_k the subset of them that belong to class k. The confidence of the prediction that instance x belongs to class k can then be estimated by the Laplace ratio

$$H^t(x) = \frac{N \times \sum_{i \in S_k} w_i^t \ + \ 1}{N \times \sum_{i \in S} w_i^t \ + \ 2}.$$

(When x has unknown values for some attributes, C4.5 can use several leaves in making a prediction. A similar confidence estimate can be constructed for such situations.) Note that the confidence measure $H^t(x)$ is still determined relative to the boosted distribution w^t, not to the original uniform distribution of the instances.

The above experiments were repeated with a modified form of boosting, the only change being the use of $H^t(x)$ rather than $\log(1/\beta^t)$ as the voting weight of C^t when classifying instance x. Results show improvement on 25 of the 27 datasets, the same error rate on one dataset, and a higher error rate on only one of the 27 datasets (chess). Average error rate is approximately 3% less than that obtained with the original voting weights.

This modification is necessarily ad-hoc, since the confidence estimate H^t has only an intuitive meaning. However, it will be interesting to experiment with other voting schemes, and to see whether any of them can be used to give error bounds similar to those proved for the original boosting method.

Conclusion

Trials over a diverse collection of datasets have confirmed that boosted and bagged versions of C4.5 produce noticeably more accurate classifiers than the standard version. Boosting and bagging both have a sound theoretical base and also have the advantage that the extra computation they require is known in advance – if T classifiers are generated, then both require T times the computational effort of C4.5. In these experiments, a 10-fold increase in computation buys an average reduction of between 10% and 19% of the classification error. In many applications, improvements of this magnitude would be well worth the computational cost. In some cases the improvement is dramatic – for the largest dataset (letter) with 20,000 instances, modified boosting reduces C4.5's classification error from 12% to 4.5%.

Boosting seems to be more effective than bagging when applied to C4.5, although the performance of the bagged C4.5 is less variable that its boosted counterpart. If the voting weights used to aggregate component classifiers into a boosted classifier are altered to reflect the confidence with which individual instances are classified, better results are obtained on almost all the datasets investigated. This adjustment is decidedly ad-hoc, however, and undermines the theoretical foundations of boosting to some extent.

A better understanding of why boosting sometimes fails is a clear desideratum at this point. Freund and Schapire put this down to overfitting, although the degradation can occur at very low values of T as shown in Figure 1. In some cases in which boosting increases error, I have noticed that the class distributions across the weight vectors w^t become very skewed. With the iris dataset, for example, the initial weights of the three classes are equal, but the weight vector w^5 of the fifth trial has them as setosa=2%, versicolor=75%, and virginica=23%. Such skewed weights seem likely to lead to an undesirable bias towards or against predicting some classes, with a concomitant increase in error on unseen instances. This is especially damaging when, as in this case, the classifier derived from the skewed distribution has a high voting weight. It may be possible to modify the boosting approach and its associated proofs so that weights are adjusted separately within each class without changing overall class weights.

Since this paper was written, Freund and Schapire (1996b) have also applied AdaBoost.M1 and bagging to C4.5 on 27 datasets, 18 of which are used in this paper. Their results confirm that the error rates of boosted and bagged classifiers are significantly lower than those of single classifiers. However, they find bagging much more competitive with boosting, being superior on 11 datasets, equal on four, and inferior on 12. Two important differences between their experiments and those reported here might account for this discrepancy. First, Freund and Schapire use a much higher number $T{=}100$ of boosting and bagging trials than the $T{=}10$ of this paper. Second, they did not modify C4.5 to use weighted instances, instead resampling the training data in a manner analogous to bagging, but using w_x^t as the probability of selecting instance x at each draw on trial t. This resampling negates a major advantage enjoyed by boosting over bagging, viz. that all training instances are used to produce each constituent classifier.

Acknowledgements

Thanks to Manfred Warmuth and Rob Schapire for a stimulating tutorial on Winnow and boosting. This research has been supported by a grant from the Australian Research Council.

Appendix: Description of Datasets

Name	Cases	Classes	Attributes	
			Cont	Discr
anneal	898	6	9	29
audiology	226	6	–	69
auto	205	6	15	10
breast-w	699	2	9	–
chess	551	2	–	39
colic	368	2	10	12
credit-a	690	2	6	9
credit-g	1,000	2	7	13
diabetes	768	2	8	–
glass	214	6	9	–
heart-c	303	2	8	5
heart-h	294	2	8	5
hepatitis	155	2	6	13
hypo	3,772	5	7	22
iris	150	3	4	–
labor	57	2	8	8
letter	20,000	26	16	–
lymph	148	4	–	18
phoneme	5,438	47	–	7
segment	2,310	7	19	–
sick	3,772	2	7	22
sonar	208	2	60	–
soybean	683	19	–	35
splice	3,190	3	–	62
vehicle	846	4	18	–
vote	435	2	–	16
waveform	300	3	21	–

References

Breiman, L. 1996. Bagging predictors. *Machine Learning*, forthcoming.

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

Brodley, C. E. 1993. Addressing the selective superiority problem: automatic algorithm/model class selection. In *Proceedings 10th International Conference on Machine Learning*, 17-24. San Francisco: Morgan Kaufmann.

Buntine, W. L. 1991. Learning classification trees. In Hand, D. J. (ed), *Artificial Intelligence Frontiers in Statistics*, 182-201. London: Chapman & Hall.

Catlett, J. 1991. Megainduction: a test flight. In *Proceedings 8th International Workshop on Machine Learning*, 596-599. San Francisco: Morgan Kaufmann.

Chan, P. K. and Stolfo, S. J. 1995. A comparative evaluation of voting and meta-learning on partitioned data. In *Proceedings 12th International Conference on Machine Learning*, 90-98. San Francisco: Morgan Kaufmann.

Dietterich, T. G., and Bakiri, G. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2: 263-286.

Freund, Y., and Schapire, R. E. 1996a. A decision-theoretic generalization of on-line learning and an application to boosting.Unpublished manuscript, available from the authors' home pages ("http://www.research.att.com/orgs/ssr/people/{yoav,schapire}"). An extended abstract appears in *Computational Learning Theory: Second European Conference, EuroCOLT '95*, 23-27, Springer-Verlag, 1995.

Freund, Y., and Schapire, R. E. 1996b. Experiments with a new boosting algorithm. Unpublished manuscript.

Heath, D., Kasif, S., and Salzberg, S. 1993. Learning oblique decision trees. In *Proceedings 13th International Joint Conference on Artificial Intelligence*, 1002-1007. San Francisco: Morgan Kaufmann.

Kohavi, R., and John, G. H. 1995. Automatic parameter selection by minimizing estimated error. In *Proceedings 12th International Conference on Machine Learning*, 304-311. San Francisco: Morgan Kaufmann,

Murphy, P. M., and Pazzani, M. J. 1991. ID2-of-3: constructive induction of M-of-N concepts for discriminators in decision trees. In *Proceedings 8th International Workshop on Machine Learning*, 183-187. San Francisco: Morgan Kaufmann.

Quinlan, J. R. 1987. Inductive knowledge acquisition: a case study. In Quinlan, J. R. (ed), *Applications of Expert Systems.* Wokingham, UK: Addison Wesley.

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

Ragavan, H., and Rendell, L. 1993. Lookahead feature construction for learning hard concepts. In *Proceedings 10th International Conference on Machine Learning*, 252-259. San Francisco: Morgan Kaufmann.

Ting, K. M. 1994. The problem of small disjuncts: its remedy in decision trees. In *Proceedings 10th Canadian Conference on Artificial Intelligence*, 91-97.

Utgoff, P. E., and Brodley, C. E. 1990. An incremental method for finding multivariate splits for decision trees. In *Proceedings 7th International Conference on Machine Learning*, 58-65. San Francisco: Morgan Kaufmann.

Vapnik, V. 1983. *Estimation of Dependences Based on Empirical Data.* New York: Springer-Verlag.

Zheng, Z. 1995. Constructing nominal X-of-N attributes. In *Proceedings 14th International Joint Conference on Artificial Intelligence*, 1064-1070. San Francisco: Morgan Kaufmann.