From: KDD-96 Proceedings. Copyright © 1996, AAAI (www.aaai.org). All rights reserved.

Linear-Time Rule Induction

Pedro Domingos
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717, U.S.A.
pedrod@ics.uci.edu
http://www.ics.uci.edu/~pedrod

Abstract

The recent emergence of data mining as a major application of machine learning has led to increased interest in fast rule induction algorithms. These are able to efficiently process large numbers of examples, under the constraint of still achieving good accuracy. If e is the number of examples, many rule learners have $O(e^4)$ asymptotic time complexity in noisy domains, and C4.5RULES has been empirically observed to sometimes require $O(e^3)$. Recent advances have brought this bound down to $O(e \log^2 e)$, while maintaining accuracy at the level of C4.5RULES's. In this paper we present CWS, a new algorithm with guaranteed $O(e)$ complexity, and verify that it outperforms C4.5RULES and CN2 in time, accuracy and output size on two large datasets. For example, on NASA's space shuttle database, running time is reduced from over a month (for C4.5RULES) to a few hours, with a slight gain in accuracy. CWS is based on interleaving the induction of all the rules and evaluating performance globally instead of locally (i.e., it uses a "conquering without separating" strategy as opposed to a "separate and conquer" one). Its bias is appropriate to domains where the underlying concept is simple and the data is plentiful but noisy.

Introduction and Previous Work

Very large datasets pose special problems for machine learning algorithms. A recent large-scale study found that most algorithms cannot handle such datasets in a reasonable time with a reasonable accuracy (Michie, Spiegelhalter, & Taylor 1994). However, in many areas—including astronomy, molecular biology, finance, retail, health care, etc.—large databases are now the norm, and discovering patterns in them is a potentially very productive enterprise, in which interest is rapidly growing (Fayyad & Uthurusamy 1995). Designing learning algorithms appropriate for such problems has thus become an important research problem.

In these "data mining" applications, the main consideration is typically not to maximize accuracy, but to extract useful knowledge from a database. The learner's output should still represent the database's contents with reasonable fidelity, but it is also important that it be comprehensible to users without machine learning expertise. "If ... then ..." rules are perhaps the most easily understood of all representations currently in use, and they are the focus of this paper.

A major problem in data mining is that the data is often very noisy. Besides making the extraction of accurate rules more difficult, this can have a disastrous effect on the running time of rule learners. In C4.5RULES (Quinlan 1993), a system that induces rules via decision trees, noise can cause running time to become cubic in e, the number of examples (Cohen 1995). When there are no numeric attributes, C4.5, the component that induces decision trees, has complexity $O(ea^2)$, where a is the number of attributes (Utgoff 1989), but its running time in noisy domains is dwarfed by that of the conversion-to-rules phase (Cohen 1995). Outputting trees directly has the disadvantage that they are typically much larger and less comprehensible than the corresponding rule sets. Noise also has a large negative impact on windowing, a technique often used to speed up C4.5/C4.5RULES for large datasets (Catlett 1991).

In algorithms that use reduced error pruning as the simplification technique (Brunk & Pazzani 1991), the presence of noise causes running time to become $O(e^4 \log e)$ (Cohen 1993). Fürnkranz and Widmer (1994) have proposed *incremental reduced error pruning (IREP)*, an algorithm that prunes each rule immediately after it is grown, instead of waiting until the whole rule set has been induced. Assuming the final rule set is of constant size, IREP reduces running time to $O(e \log^2 e)$, but its accuracy is often lower than C4.5RULES's (Cohen 1995). Cohen introduced a number of modifications to IREP, and verified empirically that RIPPERk, the resulting algorithm, is competitive with C4.5RULES in accuracy, while retaining an average running time similar to IREP's (Cohen 1995).

Catlett (Catlett 1991) has done much work in making decision tree learners scale to large datasets. A preliminary empirical study of his peepholing technique shows that it greatly reduces C4.5's running time without significantly affecting its accuracy.[1] To the best of our knowledge, peepholing has not been evaluated on any large real-world datasets, and has not been applied to rule learners.

A number of algorithms achieve running time linear in e by forgoing the greedy search method used by the learners above, in favor of exhaustive or pruned near-exhaustive search (e.g., (Weiss, Galen, & Tadepalli 1987; Smyth, Goodman, & Higgins 1990; Segal & Etzioni 1994)). However, this causes running time to become exponential in a, leading to a very high cost per example, and making application of those algorithms to large databases difficult. Holte's 1R algorithm (Holte 1993) outputs a single tree node, and is linear in a and $O(e \log e)$, but its accuracy is often much lower than C4.5's.

Ideally, we would like to have an algorithm capable of inducing accurate rules in time linear in e, without becoming too expensive in other factors. This paper describes such an algorithm and its empirical evaluation. The algorithm is presented in the next section, which also derives its worst-case time complexity. A comprehensive empirical evaluation of the algorithm is then reported and discussed.

The CWS Algorithm

Most rule induction algorithms employ a "separate and conquer" method, inducing each rule to its full length before going on to the next one. They also evaluate each rule by itself, without regard to the effect of other rules. This is a potentially inefficient approach: rules may be grown further than they need to be, only to be pruned back afterwards, when the whole rule set has already been induced. An alternative is to interleave the construction of all rules, evaluating each rule in the context of the current rule set. This can be termed a "conquering without separating" approach, by contrast with the earlier method, and has been implemented in the CWS algorithm.

In CWS, each example is a *vector* of *attribute-value pairs*, together with a specification of the *class* to which it belongs; attributes can be either *nominal* (symbolic) or *numeric*. Each rule consists of a conjunction of *antecedents* (the *body*) and a *predicted class* (the *head*). Each antecedent is a condition on a single attribute. Conditions on nominal attributes are equality tests of the form $a_i = v_{ij}$, where a_i is the attribute and v_{ij} is one of its legal values. Conditions on numeric attributes are inequalities of the form $a_i > v_{ij}$ or $a_i < v_{ij}$. Each rule in CWS is also associated with a vector of class probabilities computed from the examples it covers; the predicted class is the one with the highest probability. For class C_i, $P_r(C_i)$ is estimated by n_{ri}/n_r, where n_r is the total number of examples covered by rule r, and n_{ri} is the number of examples of the ith class among them. When a test example is covered by more than one rule, the class probability vectors of all the rules covering it are summed, and the class with the highest sum is chosen as the winner. This is similar to the approach followed in CN2 (Clark & Boswell 1991), with the difference that probabilities are used instead of frequencies. In a system like CN2, this could give undue weight to rules covering very few examples (the "small disjuncts" (Holte, Acker, & Porter 1989)), but we have verified empirically that in CWS this problem is largely avoided. Examples not covered by any rules are assigned to the class with the most examples in the training set.

[1] Due to the small number of data points (3) reported for the single real-world domain used, it is difficult to determine the exact form of the resulting time growth (linear, log-linear, etc.).

Table 1: The CWS algorithm.

```
Procedure CWS

Let RS = ∅.
Repeat
   Add one active rule with empty body to RS.
   For each active rule R in RS,
      For each possible antecedent AV,
         Let R' = R with AV conjoined to its body.
         Compute class probs. and pred. class for R'.
         Let RS' = RS with R replaced by R'.
         If Acc(RS') > Acc(RS) then let RS = RS'.
      If RS is unchanged then deactivate R.
Until all rules are inactive.
Return RS.
```

CWS is outlined in pseudo-code in Table 1. Initially the rule set is empty, and all examples are assigned to the majority class. In each cycle a new rule with empty body is tentatively added to the set, and each of the rules already there is specialized by one additional antecedent. Thus induction of the second rule starts immediately after the first one is begun, etc., and induction of all rules proceeds in step. At the end of each cycle, if a rule has not been specialized, it is deactivated, meaning that no further specialization of it will be attempted. If the rule with empty body is deactivated, it is also deleted from the rule set. A rule with empty body predicts the default class, but this is irrelevant, because a rule only starts to take part in the classification of examples once it has at least one antecedent, and it will then predict the class that most training examples satisfying that antecedent belong to. A rule's predicted class may change as more antecedents are added to it. $Acc(RS)$ is the accuracy of the rule set RS on the training set (i.e., the fraction of examples that RS classifies correctly). Most rule induction algorithms evaluate only the accuracy (or entropy, etc.) of the changed rule on the examples that it still covers. This ignores the effect of any other rules that cover those examples, and also the effect of uncovering some examples by specializing the rule, and leads to a tendency for overspecialization that has to be countered by pruning. CWS avoids this through its global evaluation procedure and interleaved rule induction.

Let e be the number of examples, a the number of attributes, v the average number of values per attribute, c the number of classes, and r the number of rules produced. The basic step of the algorithm involves adding an antecedent to a rule and recomputing $Acc(RS')$. This requires matching all rules with all training examples, and for each example summing the class probabilities of the rules covering it, implying a time cost of $O[re(a+c)]$. Since there are $O(av)$ possible antecedents, the cost of the inner loop ("For each AV", see Table 1) is $O[avre(a+c)]$. However, this cost can be much reduced by avoiding the extensive redundancy present in the repeated computation of $Acc(RS')$. The key to this optimization is to avoid rematching all the rules that remain unchanged when attempting to specialize a given rule, and to match the unchanged antecedents of this rule with each example only once. Recomputing $Acc(RS')$ when a new antecedent AV is attempted now involves only checking whether each example already covered by the rule also satisfies that antecedent, at a cost of $O(e)$, and updating its class probabilities if it does, at a cost of $O(ec)$. The latter term dominates, and the cost of recomputing the accuracy is thus reduced to $O(ec)$, leading to a cost of $O(eavc)$ for the "For each AV" loop.

In more detail, the optimized procedure is as follows. Let $Cprobs(R)$ denote the vector of class probabilities for rule R, and $Cscores(E)$ denote the sum of the class probability vectors for all rules covering example E. $Cscores(E)$ is maintained for each example throughout. Let R be the rule whose specialization is going to be attempted. Before the "For each AV" loop begins, R is matched to all examples and those which satisfy it are selected, and, for each such example E, $Cprobs(R)$ is subtracted from $Cscores(E)$. $Cscores(E)$ now reflects the net effect of all other rules on the example. Each possible antecedent AV is now conjoined to the rule in turn, leading to a changed rule R' (or $R'(AV)$, to highlight that it is a function of AV). New class probabilities for R' are computed by finding which examples E' among the previously-selected ones satisfy AV. These probabilities are now added to $Cscores(E')$ for the still-covered examples E'. Examples that were uncovered by the specialization already have the correct values of $Cscores(E)$, since the original rule's $Cprobs(R)$ were subtracted from them beforehand. All that remains is to find the new winning class for each example E. If the example was previously misclassified and is now correctly classified, there is a change of $+1/e$ in the accuracy of the rule set. If it was previously correctly classified, the change is $-1/e$. Otherwise there is no change. Summing this for all the examples yields the global change in accuracy. As successive antecedents are attempted, the best antecedent and maximum global change in accuracy are remembered. At the end of the loop the best antecedent is permanently added to the rule, if the corresponding change in accuracy is positive. This simply involves repeating the procedure above, this time with permanent effects. If no antecedent produces a positive change in accuracy, the rule's original class probabilities $Cprobs(R)$ are simply re-added to the $Cscores(E)$ of all the examples that it covers, leaving everything as before. This procedure is shown in pseudo-code in Table 2. Note that the optimized version produces exactly the same output as the non-optimized one; conceptually, the much simpler Table 1 is an exact description of the CWS algorithm.

Table 2: The optimized CWS algorithm.

```
Procedure CWS

Let RS = ∅.
Let Cscores(E) = 0 for all E, C.
Repeat
   Add one active rule R_n with empty body to RS.
   Let Cprobs(R_n) = 0 for all C.
   For each active rule R in RS,
      For each example E covered by R,
         Subtract Cprobs(R) from Cscores(E).
      For each possible antecedent AV,
         Let ΔAcc(AV) = 0.
         Let R' = R with AV conjoined to it.
         Compute Cprobs(R') and its pred. class.
         For each example E' covered by R'
            Add Cprobs(R') to Cscores(E').
         For each example E covered by R
            Assign E to class with max. Cscore(E).
            Compute ΔAcc_E(AV) (-1/e, 0 or +1/e).
            Add ΔAcc_E(AV) to ΔAcc(AV).
      Pick AV with max. ΔAcc(AV).
      If ΔAcc(AV) > 0 then R = R'(AV),
         else deactivate R.
      For each ex. E covered by R (R = R' or not)
         Add Cprobs(R) to Cscores(E).
Until all rules are inactive.
Return RS.
```

The total asymptotic time complexity of the algorithm is obtained by multiplying $O(eavc)$ by the maximum number of times that the double outer loop ("Repeat ... For each R in RS ...") can run. Let s be the output size, measured as the total number of antecedents effectively added to the rule set. Then the double outer loop runs at most $O(s)$, since each computation within it (the "For each AV" loop) adds at most one antecedent. Thus the total asymptotic time complexity of CWS is $O(eavcs)$.

The factor s is also present in the complexity of other rule induction algorithms (CN2, IREP, RIPPERk, etc.), where it can typically grow to $O(ea)$. It can become a significant problem if the dataset is noisy. However, in CWS it cannot grow beyond $O(e)$, because each computation within the double outer loop ("Repeat ... For ...") either produces an improvement in accuracy or is the last one for that rule, and in a dataset with e examples at most e improvements in accuracy are possible. Ideally, s should be independent of e, and this is the assumption made in (Fürnkranz & Widmer 1994) and (Cohen 1995), and verified below for CWS.

CWS incorporates three methods for handling numeric values, selectable by the user. The default method discretizes each attribute into equal-sized intervals, and has no effect on the asymptotic time complexity of the algorithm. Discretization can also be performed using a method similar to Catlett's (Catlett 1991), repeatedly choosing the partition that minimizes entropy until one of several termination conditions is met. This causes learning time to become $O(e \log e)$, but may improve accuracy in some situations. Finally, numeric attributes can be handled directly by testing a condition of each type ($a_i > v_{ij}$ and $a_i < v_{ij}$) at each value v_{ij}. This does not change the asymptotic time complexity, but may cause v to become very large. Each of the last two methods may improve accuracy in some situations, at the cost of additional running time. However, uniform discretization is surprisingly robust (see (Dougherty, Kohavi, & Sahami 1995)), and can result in higher accuracy by helping to avoid overfitting.

Missing values are treated by letting them match any condition on the respective attribute, during both learning and classification.

Empirical Evaluation

This section describes an empirical study comparing CWS with C4.5RULES and CN2 along three variables: running time, accuracy, and comprehensibility of the output. All running times were obtained on a Sun 670 computer. Output size was taken as a rough measure of comprehensibility, counting one unit for each antecedent and consequent in each rule (including the default rule, with 0 antecedents and 1 consequent). This measure is imperfect for two reasons. First, for each system the meaning of a rule is not necessarily transparent: in CWS and CN2 overlapping rules are probabilistically combined to yield a class prediction, and in C4.5RULES each rule's antecedent side is implicitly conjoined with the negation of the antecedents of all preceding rules of different classes. Second, output simplicity is not the only factor in comprehensibility, which is ultimately subjective. However, it is an acceptable and frequently used approximation, especially when the systems being compared have similar output, as here (see (Catlett 1991) for further discussion).

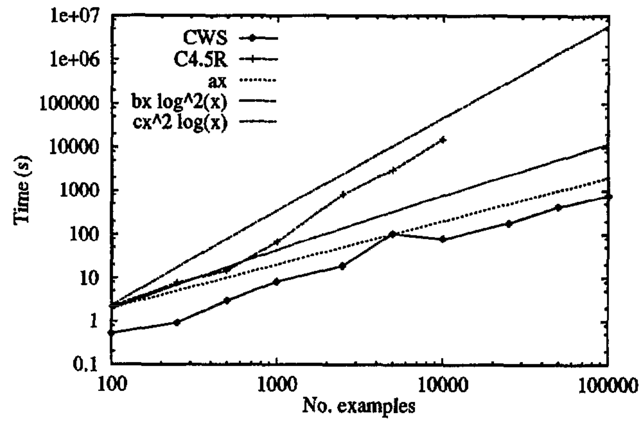

Figure 1: Learning times for the concept $abc \vee def$.

A preliminary study was conducted using the Boolean concept $abc \vee def$ as the learning target, with each disjunct having a probability of appearing in the data of 0.25, with 13 irrelevant attributes, and with 20% class noise (i.e., each class label has a probability of 0.2 of being flipped). Figure 1 shows the evolution of learning time with the number of examples for CWS and C4.5RULES, on a log-log scale. Recall that on this type of scale the slope of a straight line corresponds to the exponent of the function being plotted. Canonical functions approximating each curve are also shown, as well as $e \log^2 e$, the running time observed by Cohen (Cohen 1995) for RIPPERk and IREP.[2] CWS's running time grows linearly with the number of examples, as expected, while C4.5RULES's is $O(e^2 \log e)$. CWS is also much faster than IREP and RIPPERk (note that, even though the log-log plot shown does not make this evident, the difference between e and $e \log^2 e$ is much larger than e).

CWS is also more accurate than C4.5RULES for each number of examples, converging to within 0.6% of the Bayes optimum (80%) for only 500 examples, and reaching it with 2500, while C4.5RULES never rises above 75%. CWS's output size stabilizes at 9, while C4.5RULES's increases from 17 for 100 examples to over 2300 for 10000. Without noise, both systems learn the concept easily. Thus these results indicate that CWS is more robust with respect to noise, at least in this simple domain. CN2's behavior is similar to C4.5RULES's in time and accuracy, but it produces larger rule sets.

The relationship between the theoretical bound of $O(eavcs)$ and CWS's actual average running time was investigated by running the system on 28 datasets from the UCI repository[3] (Murphy & Aha 1995). Figure 2

[2] The constants a, b and c were chosen so as to make the respective curves fit conveniently in the graph.

[3] Audiology, annealing, breast cancer (Ljubljana), credit

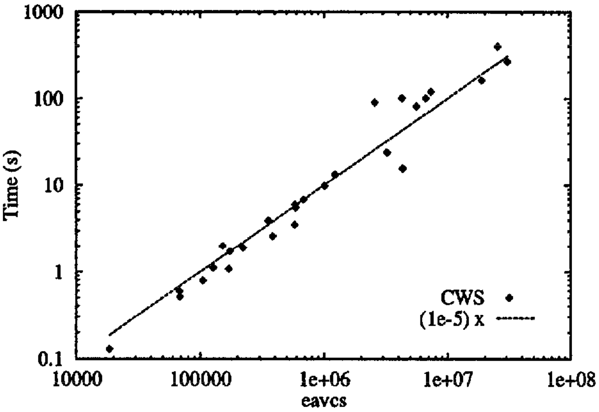

Figure 2: Relationship of empirical and theoretical learning times.

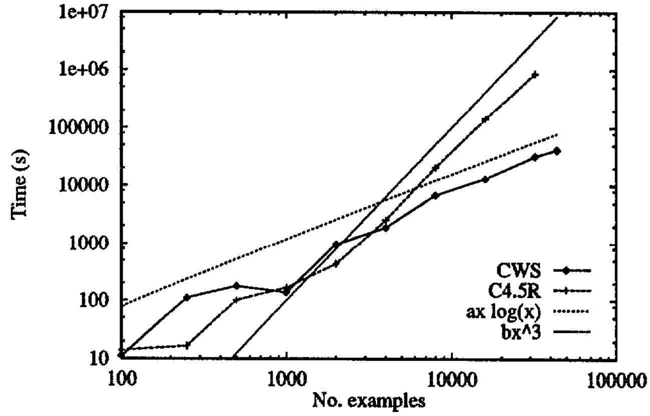

Figure 3: Learning times for the shuttle database.

plots CPU time against the product *eavcs*. Linear regression yields the line $time = 1.1 \times 10^5\ eavcs + 5.1$, with a correlation of 0.93 ($R^2 = 0.87$). Thus *eavcs* explains almost all the observed variation in CPU time, confirming the prediction of a linear bound.[4]

Experiments were also conducted on NASA's space shuttle database. This database contains 43500 training examples from one shuttle flight, and 14500 testing examples from a different flight. Each example is described by nine numeric attributes obtained from sensor readings, and there are seven possible classes, corresponding to states of the shuttle's radiators (Catlett 1991). The goal is to predict these states with very high accuracy (99–99.9%), using rules that can be taught to a human operator. The data is known to be relatively noise-free; since our interest is in induction algorithms for large noisy databases, 20% class noise was added to the training data following a procedure similar to Catlett's (each class has a 20% probability of being changed to a random class, including itself).

The evolution of learning time with the number of training examples for CWS and C4.5RULES is shown in Figure 3 on a log-log scale, with approximate asymptotes also shown, as before. CWS's curve is approximately log-linear, with the logarithmic factor attributable to the direct treatment of numeric values that was employed. (Uniform discretization resulted in linear time, but did not yield the requisite very high accuracies.) C4.5RULES's curve is roughly cubic. Extrapolating from it, C4.5RULES's learning time for the full database would be well over a month, while CWS takes 11 hours.

Learning curves are shown in Figure 4. CWS's accuracy is higher than C4.5RULES's for most points, and generally increases with the number of examples, showing that there is gain in using the larger samples, up to the full dataset. Figure 5 shows the evolution of output size. CWS's is low and almost constant, while C4.5RULES's grows to more than 500 by $e = 32000$. Up to 8000 examples, CN2's running time is similar to C4.5RULES's, but its output size grows to over 1700, and its accuracy never rises above 94%.[5] In summary, in this domain CWS outperforms C4.5RULES and CN2 in running time, accuracy and output size.

Compared to the noise-free case, the degradation in CWS's accuracy is less than 0.2% after $e = 100$, the rule set size is similar, and learning time is only degraded by a constant factor (of a few percent, on average). Thus CWS is again verified to be quite robust with respect to noise.

Conclusions and Future Work

This paper introduced CWS, a rule induction algorithm that employs a "conquering without separating" strategy instead of the more common "separate-and-conquer" one. CWS interleaves the induction of all rules and evaluates proposed induction steps globally. Its asymptotic time complexity is linear in the number of examples. Empirical study shows that it can be used to advantage when the underlying concept is simple and the data is plentiful but noisy.

Directions for future work include exploring ways of boosting CWS's accuracy (or, conversely, broadening the set of concepts it can learn effectively) without affecting its asymptotic time complexity, and applying it to larger databases and problems in different areas.

screening (Australian), chess endgames (kr-vs-kp), Pima diabetes, echocardiogram, glass, heart disease (Cleveland), hepatitis, horse colic, hypothyroid, iris, labor negotiations, lung cancer, liver disease, lymphography, mushroom, post-operative, promoters, primary tumor, solar flare, sonar, soybean (small), splice junctions, voting records, wine, and zoology.

[4]Also, there is no correlation between the number of examples e and the output size s ($R^2 = 0.0004$).

[5]For $e > 8000$ the program crashed due to lack of memory. This may be due to other jobs running concurrently.

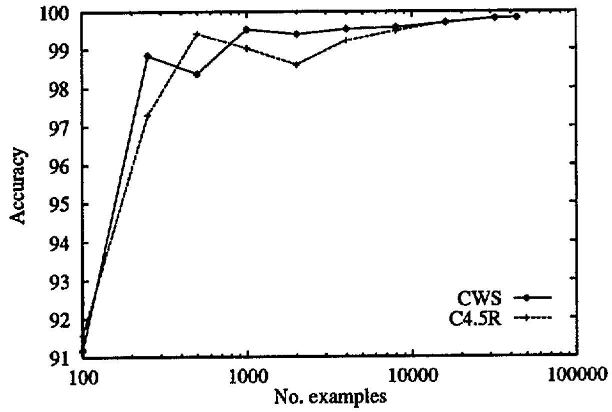

Figure 4: Learning curves for the shuttle database.

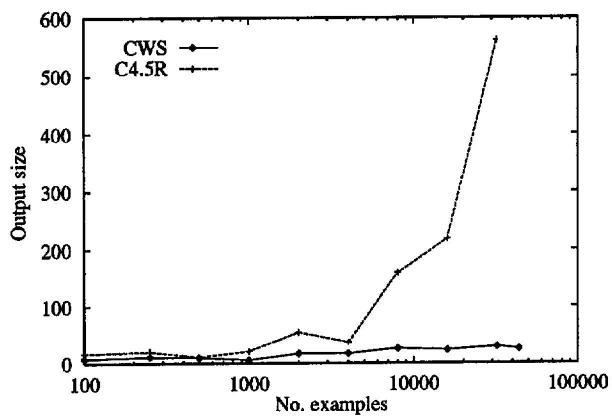

Figure 5: Output size growth for the shuttle database.

Acknowledgments

This work was partly supported by a JNICT/PRAXIS XXI scholarship. The author is grateful to Dennis Kibler for many helpful comments and suggestions, and to all those who provided the datasets used in the empirical study.

References

Brunk, C., and Pazzani, M. J. 1991. An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of the Eighth International Workshop on Machine Learning*, 389–393. Evanston, IL: Morgan Kaufmann.

Catlett, J. 1991. *Megainduction: Machine Learning on Very Large Databases.* Ph.D. Dissertation, Basser Department of Computer Science, University of Sydney, Sydney, Australia.

Clark, P., and Boswell, R. 1991. Rule induction with CN2: Some recent improvements. In *Proceedings of the Sixth European Working Session on Learning*, 151–163. Porto, Portugal: Springer-Verlag.

Cohen, W. W. 1993. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 988–994. Chambery, France: Morgan Kaufmann.

Cohen, W. W. 1995. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 115–123. Tahoe City, CA: Morgan Kaufmann.

Dougherty, J.; Kohavi, R.; and Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*, 194–202. Tahoe City, CA: Morgan Kaufmann.

Fayyad, U. M., and Uthurusamy, R., eds. 1995. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining.* Montréal, Canada: AAAI Press.

Fürnkranz, J., and Widmer, G. 1994. Incremental reduced error pruning. In *Proceedings of the Eleventh International Conference on Machine Learning*, 70–77. New Brunswick, NJ: Morgan Kaufmann.

Holte, R. C.; Acker, L. E.; and Porter, B. W. 1989. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 813–818. Detroit, MI: Morgan Kaufmann.

Holte, R. C. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning* 11:63–91.

Michie, D.; Spiegelhalter, D. J.; and Taylor, C. C., eds. 1994. *Machine Learning, Neural and Statistical Classification.* New York: Ellis Horwood.

Murphy, P. M., and Aha, D. W. 1995. UCI repository of machine learning databases. Machine-readable data repository, Department of Information and Computer Science, University of California at Irvine, Irvine, CA.

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

Segal, R., and Etzioni, O. 1994. Learning decision lists using homogeneous rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 619–625. Seattle, WA: AAAI Press.

Smyth, P.; Goodman, R. M.; and Higgins, C. 1990. A hybrid rule-based/Bayesian classifier. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, 610–615. Stockholm, Sweden: Pitman.

Utgoff, P. E. 1989. Incremental induction of decision trees. *Machine Learning* 4:161–186.

Weiss, S. M.; Galen, R. M.; and Tadepalli, P. V. 1987. Optimizing the predictive value of diagnostic decision rules. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 521–526. Seattle, WA: AAAI Press.