Error-Based and Entropy-Based Discretization of Continuous Features

Ron Kohavi
Data Mining and Visualization
Silicon Graphics, Inc.
2011 N. Shoreline Blvd
Mountain View, CA 94043-1389
ronnyk@sgi.com

Mehran Sahami
Gates Building 1A, Room 126
Computer Science Department
Stanford University
Stanford, CA 94305-9010
sahami@cs.stanford.edu

Abstract

We present a comparison of error-based and entropy-based methods for discretization of continuous features. Our study includes both an extensive empirical comparison as well as an analysis of scenarios where error minimization may be an inappropriate discretization criterion. We present a discretization method based on the C4.5 decision tree algorithm and compare it to an existing entropy-based discretization algorithm, which employs the Minimum Description Length Principle, and a recently proposed error-based technique. We evaluate these discretization methods with respect to C4.5 and Naive-Bayesian classifiers on datasets from the UCI repository and analyze the computational complexity of each method. Our results indicate that the entropy-based MDL heuristic outperforms error minimization on average. We then analyze the shortcomings of error-based approaches in comparison to entropy-based methods.

Introduction

Although real-world classification and data mining tasks often involve continuous features, there exist many algorithms which focus on learning only in nominal feature spaces (Apte & Hong 1996; Cost & Salzberg 1993).

In order to handle continuous features, such algorithms regularly employ simple discretization methods, such as uniform binning of the data, to produce nominal features. Such naive discretization of the data can be potentially disastrous for data mining as critical information may be lost due to the formation of inappropriate bin boundaries. Furthermore, discretization itself may be viewed as a form of knowledge discovery in that critical values in a continuous domain may be revealed. It has also been noted by Catlett (1991) that for very large data sets (as is common in data mining applications), discretizing continuous features can often vastly reduce the time necessary to induce a classifier. As a result, better discretization methods have been developed, but these methods are often not directly compared to each each other, or analyzed for when they may be appropriate to employ.

Dougherty, Kohavi, & Sahami (1995) provided an initial comparison of uniform binning, the discretization method proposed by Holte (1993), and an entropy based method proposed by Fayyad & Irani (1993) using two induction algorithms: C4.5 (Quinlan 1993) and a Naive-Bayesian classifier (Good 1965). Since that study reported that the entropy-based discretization method was the most promising method, we compare that method to two other methods: C4.5-based discretization and error-based discretization.

The C4.5-based discretization is a new entropy-based method that applies C4.5 to each continuous feature separately to determine the number of thresholds and their values. Hence, we still use an entropy-based metric (gain-ratio), but use a different criterion for the number of intervals, *i.e.*, determined by pruning as opposed to Fayyad and Irani's stopping criteria.

The error-based discretization we compare has been proposed by Maass (1994) and used in the T2 algorithm (Auer, Holte, & Maass 1995). Given a number of intervals, k, the method constructs the optimal discretization of a continuous feature with respect to classification error in polynomial time.

We employ the discretization methods listed above in conjunction with C4.5 and Naive-Bayesian classifiers as induction algorithms that are run on the discretized data and show the effectiveness of each discretization method. We also present the computational complexity of each discretization technique. In light of our empirical findings, we analyze situations in which error-based discretization may be inappropriate.

Methods

We briefly describe the induction algorithms and discretization methods we compare.

Induction Algorithms

In our experimental study, we test different discretization methods as applied to C4.5 and Naive-Bayesian classifiers. C4.5 (Quinlan 1993) is a state-of-the-art top-down decision tree induction algorithm. When we discretize features, we declare them nominal, thus C4.5 does a multi-way split on all possible thresholds.

The Naive-Bayesian induction algorithm computes the posterior probability of the classes given the data, assuming independence between the features for each class. The probabilities for nominal features are estimated using counts and a Gaussian distribution is assumed for continuous features (in the undiscretized cases). The Naive-Bayesian classifier used in our experiments is the one implemented in $\mathcal{MLC}$++ (Kohavi *et al.* 1994).

Discretization Algorithms

We focus on two discretization methods using entropy and a recently developed error-based discretization method. These methods are described below. A comprehensive review of the existing discretization literature is found in Dougherty, Kohavi, & Sahami (1995).

Fayyad and Irani's Method First, we consider discretization based on an entropy minimization heuristic proposed by Fayyad & Irani (1993). The method is similar to that of Catlett (1991) but offers a more motivated heuristic for deciding on the number of intervals. This algorithm uses the class information entropy of candidate partitions to select threshold boundaries for discretization. It finds a single threshold that minimizes the entropy function over all possible thresholds; it is then recursively applied to both of the partitions induced. The *Minimal Description Length Principle* (MDLP) is employed to determine a stopping criteria for the recursive discretization strategy. We refer to this algorithm as ***Ent-MDL.***

In our implementation, each split considered in the entropy method takes $O(m \log m)$ time, where m is the number of instances and when we assume a fixed number of classes. If the method chooses k thresholds, then at most $2k+1$ threshold computations are done. Hence, an upper bound on the time complexity is $O(km \log m)$. This bound could be improved using a smarter implementation that would sort only once. If we assume that the thresholds form a balanced tree, then the time to sort the instances at a given level is $O(m \log m)$ and the time bound can be reduced to $O(\log k \cdot m \log m)$. In practice, we expect the behavior to be somewhere between these two bounds. The space complexity of this method is $O(m)$ because only the feature value and label of each instance is stored.

C4.5 Discretization The C4.5 decision tree induction algorithm can also be used as a discretization method. In this sense, C4.5 is first applied to each continuous feature *separately* to build a tree which contains binary splits that only test the single continuous feature. The C4.5 algorithm uses gain-ratio, an entropy-based metric, to determine the partitions for discrete intervals. We refer to this new method as ***C4.5-Disc***.

This method is significantly different from that of Fayyad & Irani (1993) in that the latter employs a top-down stopping criterion based on MDLP, whereas applying C4.5 to a single feature builds a complete tree for that feature and then applies pruning to find an appropriate number of nodes in the tree (*i.e.*, the number of discretization intervals) in a bottom-up approach. After the tree for a single feature is built and pruned using C4.5, we can simply use the threshold values at each node of the induced tree to be the threshold values for a discretization of that continuous feature. We found that C4.5's default pruning confidence (the c parameter) was not pruning enough and forced it to pruned more heavily in order to prevent forming many intervals. To this end, we set the confidence factor to 1 (down from 25). Minor variations of the c value did not have much effect on our experiments. To prevent "overfitting" this value, we did not try to optimize it for our experiments.

The time complexity to discretize features using C4.5 requires that a full single-feature tree be built and then pruned back. The build time dominates the pruning time, but even if only k intervals are finally returned, many more must be constructed. If we assume that at least some constant portion p of the instances (say 10%) are split off each time, then the time bound is $O(\log_{1/(1-p)} m \cdot m \log m)$ because there can be at most $\log_{1/(1-p)} m$ levels in the tree, each taking $O(m \log m)$ time. The space complexity of the C4.5 discretization is $O(m)$ because only the feature value and label of each instance must be stored.

Error-based Discretization Significant work in error-based discretization has only recently been carried out. Maass (1994) developed an algorithm to optimally discretize a continuous feature with respect to error on the training set. This algorithm discretizes a continuous feature by producing an optimal set of k or fewer intervals that results in the minimum error on the training set if the instances were to be classified using only that single feature after discretization. We refer to this algorithm as ***ErrorMin***. The maximum number of intervals k is a user-set parameter.

This method has been implemented as part of the T2 induction algorithm (Auer, Holte, & Maass 1995) which induces one or two level decision trees. T2 circumvented the difficulty of providing a good justification for the value of k by simply setting k to be the number of classes plus one. The algorithm employs a dynamic programming approach to efficiently compute the optimal error discretization thresholds. Under the T2 heuristic the time complexity of the algorithm is $O(m(\log m + k^2))$ and the space complexity is $O(m + k^3)$, where m is the number of training instances.

We used the implementation of this algorithm from the T2 induction system, but tried two different approaches to setting the value for k. The first approach is the one proposed for T2 described above, which we call ***ErrorMin-T2***. The second approach is to set k to be the same number of intervals proposed by running the *Ent-MDL* method, which allows us to compare them for the same k values; we call this method ***ErrorMin-MDL***.

Results

We begin by presenting the experimental results and then analyze them.

Empirical Findings

Table 1 shows the datasets we chose for our comparison. We chose 17 datasets from the UCI repository (Murphy & Aha 1996) such that each had at least one continuous feature. We used 10-fold cross-validation to determine error rates for the application of each discretization and induction method pair to each dataset. It is important to note that in performing cross-validation we *separately* discretized the training set for each fold. Discretizing all the data once before creating the folds for cross-validation allows the discretization method to have access to the testing data, which is known to result in optimistic error rates.

Figure 1 shows the results for C4.5. We report the error rate for each discretization method used in conjunction with C4.5, normalized by the error rate of the original C4.5 run on the data without any prior discretization. Thus, the relative error bars below 1.0 show an improvement over C4.5 without discretization, whereas values above 1.0 show a degradation in classification performance. More generally, lower values are better. Figure 2 shows the analogous table for Naive-Bayes, normalized by the error rate for Naive-Bayes using the normal distribution (Gaussian) for continuous features.

The results for C4.5 show that *Ent-MDL* does better on average than C4.5 run without discretization, lowering error rate in several instances and never significantly increasing it. C4.5 run using *Ent-MDL* sometimes significantly outperforms C4.5 alone because discretization provides a regularization effect (all the data is used to determine the interval boundaries before training, as opposed to during training where the data is fragmented). The absolute average errors were 16.01% and 17.50% respectively, with the following p-values for the significant differences computed using a t-test: Ionosphere improved with p-value = 0.02, Glass2 improved with p-value = 0.03, and Cleve improved with p-value = 0.05. *Ent-MDL* was, on average, also the best performing discretization method of the four methods we tried. This is a noteworthy result given that this method is entropy-based and does not attempt to directly minimize error, our overall objective function. Looking at all the discretization algorithms, error rates increased significantly only in a few cases and in many cases they slightly decreased.

For the *ErrorMin* method, Hypothyroid and Sick-euthyroid degraded significantly. For hypothyroid, the relative difference is significant with p-value < 0.0002. We examined the discretization methods carefully and noted that with only two features: TSH and FTI, the error of C4.5 is almost as good as with all the features. The *ErrorMin* algorithm discretizes the TSH feature into only two intervals (for all ten folds) even though both heuristics (T2 and MDL) recommended three intervals. The reason for this problem is that *ErrorMin* will never create two adjacent intervals with the same majority class. We explore the impact of this phenomenon in an artificial example.

As reported in previous work (Dougherty, Kohavi, & Sahami 1995), any form of discretization produced large improvements over the Naive-Bayesian algorithm with the normality assumption for continuous variables. Discretization allows for the algorithm to better approximate the true distribution for a continuous variable when that distribution is not normal and thus computes a more accurate posterior probability for an instance to be of a particular class. In the rare cases where the continuous features of a domain are in fact normally distributed (as is the case with Iris and six out of eight features in Diabetes), we find that discretization causes a small increase in error rate, but these are much more the exception than the norm. We find that when discretization is applied, error rates are lower for nine domains, relatively unchanged in six domains, and only worse in two domains. All the discretization methods performed approximately the same, but *Ent-MDL* was a slight winner on average. Also, worth noting is that Naive Bayes run using *Ent-MDL* sometimes significantly outperformed C4.5. For example, performance on Anneal, Cleve, and Glass was better with p-values less than 0.01, and performance

	Dataset	Features cont	Features nom	Dataset Size	Majority Error		Dataset	Features cont	Features nom	Dataset Size	Majority Error
1	anneal	6	32	898	23.83	2	australian	6	8	690	44.49
3	breast cancer	10	0	699	34.48	4	cleve	6	7	303	45.62
5	crx	6	9	690	44.49	6	diabetes	8	0	768	34.89
7	german	24	0	1000	30.00	8	glass	9	0	214	64.46
9	glass2	9	0	163	46.73	10	heart	13	0	270	44.44
11	hepatitis	6	13	155	20.83	12	horse-colic	7	15	368	36.91
13	hypothyroid	7	18	3163	4.77	14	ionosphere	34	0	351	35.87
15	iris	4	0	150	76.67	16	sick-euthyroid	7	18	3163	9.26
17	vehicle	18	0	846	77.41						

Table 1: Datasets, the number of continuous features, nominal features, dataset size, and baseline error (majority inducer on the 10 folds).

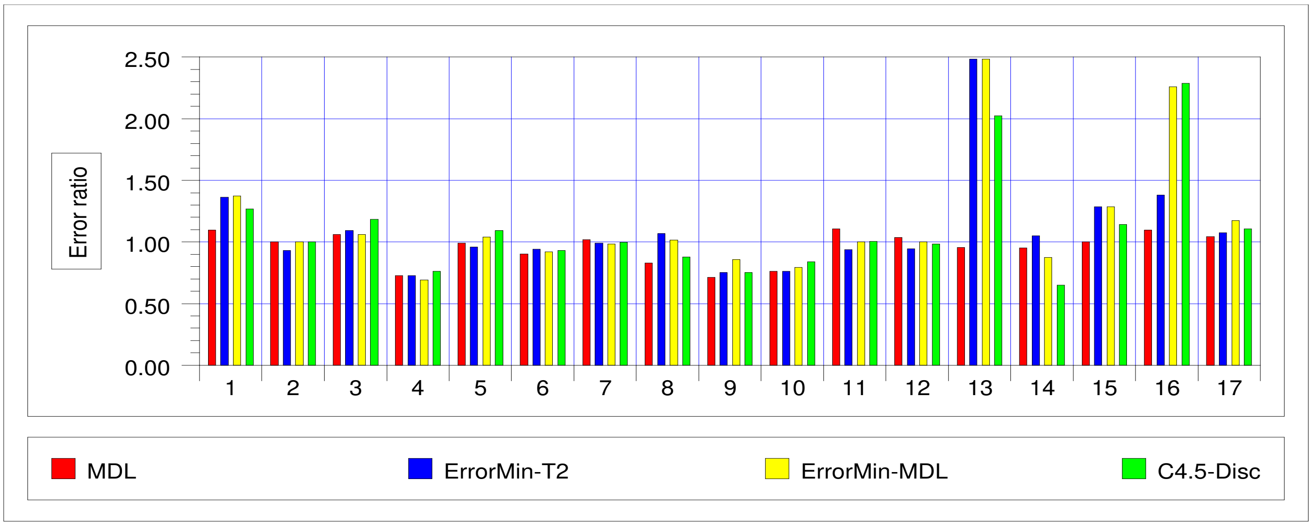

Figure 1: C4.5 with different discretization methods. Error ratios for the different discretization algorithms relative to the original C4.5. Lower values are better.

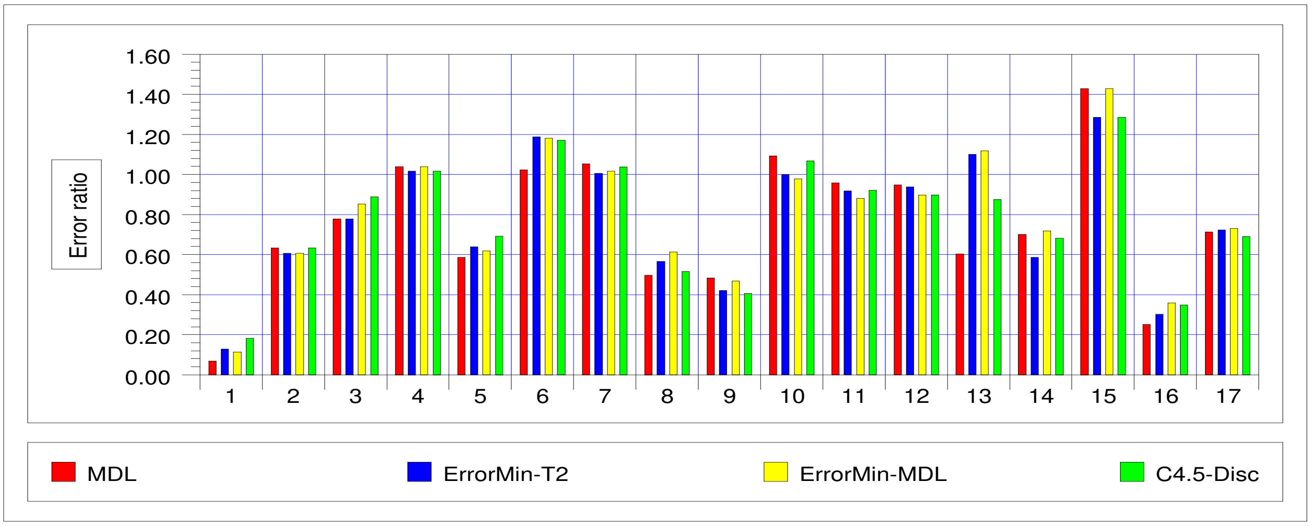

Figure 2: Naive-Bayes with different discretization methods. Error ratios for the different discretization algorithms relative to Naive-Bayes assuming normal distribution. Lower values are better.

on Breast, Diabetes, Glass, and Heart was better with p-values less than 0.05.

The running times for many of these experiments were negligible. The most time intensive datasets to discretize using *Ent-MDL* were Sick-euthyroid and Hypothyroid, which each took about 31 seconds per fold on an SGI Challenge. The longest running time for *ErrorMin* was encountered with the Glass dataset which took 153 seconds per fold to discretize, although this was much longer than any other of the datasets examined. The *ErrorMin* method could not be run on the Letter domain with 300MB of main memory.

Error vs. Entropy

To better understand why the entropy-based methods outperformed *ErrorMin* on some datasets, and why *ErrorMin* would not discretize to the suggested number of intervals, we present a simple example to show the shortcomings of error-based discretization.

Consider a Boolean target function f of two continuous variables, X_1 and X_2 in the range $[0, 1]$, defined as: $f(X_1, X_2) = ((X_1 < 0.4) \wedge (X_2 < 0.75)) \vee (X_2 < 0.25)$.

The function and its projection on X_2 are shown in Figure 3. Note that f has only two intervals of interest for X_1 (with threshold 0.4), but three intervals of interest for X_2 (with thresholds 0.25 and 0.75). *ErrorMin* is unable to form the three intervals for X_2. For this function, all instances which have $X_2 < 0.25$ will be positive whereas all instances which have $X_2 \geq 0.75$ will be negative. This leaves a large middle interval $(0.25 \leq X_2 < 0.75)$ where instances will either be labeled positive or negative depending on their value for feature X_1. Assuming a uniform distribution of instances, the middle interval will generally have more negative instances. As a result, we will have a majority of negative instances in two adjacent partitions, which is problematic for *ErrorMin* as the following observation shows.

Observation: *ErrorMin* will never generate two adjacent intervals with the same label.

The reason is that these two intervals can always be collapsed into one interval with no degradation in the error. We can thus see an inherent limitation of *ErrorMin*. The implication is that out of eight possible labelings for three intervals in a two-class problem, only two are possible with *ErrorMin*. Entropy-based discretization methods have no such limitation and can partition the space as long as the class distribution between the different partitions is different.

We generated 5,000 instances (uniformly randomly distributed) from this target concept and ran 10-fold cross-validation using *ErrorMin-T2* and *Ent-MDL*.

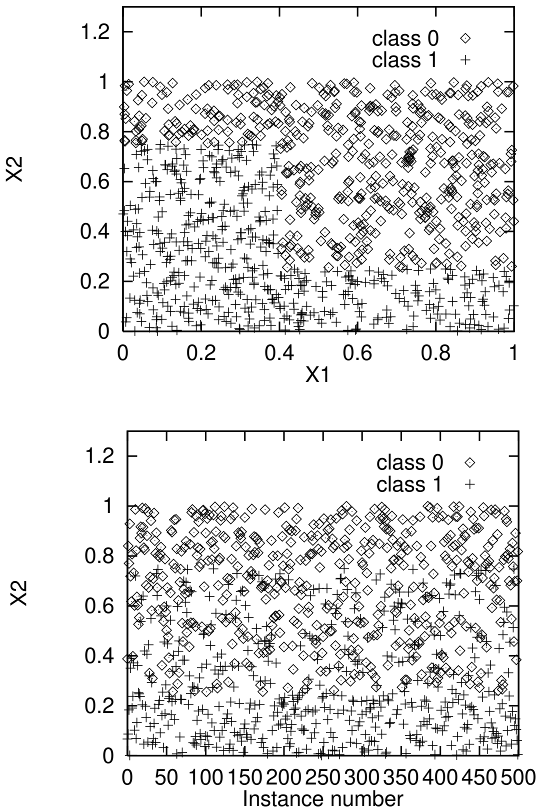

Figure 3: 500 instances from the artificial target concept f (top) and its projection on the second feature (bottom). Note that in the projection all instances below 0.25 are one class, all instances above 0.75 are of the other, and in the range 0.25-0.75, they are mixed.

The *ErrorMin-T2* heuristic recommends three intervals for each feature because this is a two-class problem. For X_1, *ErrorMin-T2* returns three intervals (although there are really only two) in nine out of the ten folds. The second threshold was always very close to the edge at 1.0. As mentioned above, *ErrorMin-T2* returns only two intervals for X_2 in all ten folds, although there are three. The Bayes-error rate for the partitions returned by *ErrorMin-T2* was 10.24%. *Ent-MDL*, on the other hand, found the correct number of partitions and with thresholds very close to the true values. The Bayes-error rate for the partitions returned by *Ent-MDL* was 0.02%.

We conclude that although error-minimization techniques always find the optimal partition to reduce the training-set error for each feature, entropy-based methods might fare better in practice because of feature interaction: as long as the distribution is different enough, a threshold will be formed, allowing other features to make the final discrimination.

Conclusions

Dougherty, Kohavi, & Sahami (1995) describe three axes along which discretization methods can be measured: *supervised* vs. *unsupervised*, *global* vs. *local*, and *static* vs. *dynamic*. The methods examined here are supervised, as they make use of the instance label information while performing discretization, whereas unsupervised methods, such as equal width binning, do not. We did not compare unsupervised methods as the previous work noted that supervised methods have a tendency to work better in practice.

The distinction between global and local methods stems from when discretization is performed. Global discretization involves discretizing all continuous features *prior* to induction. Local methods, on the other hand, carry out discretization *during* the induction process, where particular local regions of the instance space may be discretized differently (such as when C4.5 splits the same continuous feature differently down different branches of a decision tree). All the methods compared here are applied globally. In future work we aim to measure the effectiveness of these methods when applied locally.

Discretization methods often require a parameter, k, indicating the maximum number of intervals to produce in discretizing a feature. Static methods, such as those examined in this work, perform one discretization pass of the data for each feature and determine the value of k for each feature independent of the other features. Dynamic methods conduct a search through the space of possible k values for all features simultaneously, thereby capturing interdependencies in feature discretization. During the course of this study, we looked at dynamic versions of several discretization methods, using the wrapper approach (John, Kohavi, & Pfleger 1994) as a means of searching through the space of the number of discretization intervals for all variables simultaneously. We found no significant improvement in employing dynamic discretization over its static counterpart.

Our results show that *Ent-MDL* is slightly superior to the other methods for the datasets used. We have also described why *ErrorMin* methods are inappropriate in cases where features interact, and analyzed the time and space complexity of the different algorithms.

Acknowledgments We thank Lise Getoor for comments on an earlier version of this paper and Peter Auer for providing us with the code for T2. The second author is supported by an ARPA/NASA/NSF grant to the Stanford Digital Libraries Project. All the experiments reported here were done using $\mathcal{MLC}$++.

References

Apte, C., and Hong, S. 1996. Predicting equity returns from security data. In *Advances in Knowledge Discovery and Data Mining*. AAAI Press and the MIT Press. chapter 22, 541–569.

Auer, P.; Holte, R.; and Maass, W. 1995. Theory and applications of agnostic pac-learning with small decision trees. In *Machine Learning: Proceedings of the Twelfth Int. Conference*. Morgan Kaufmann.

Catlett, J. 1991. On changing continuous attributes into ordered discrete attributes. In Kodratoff, Y., ed., *Proceedings of the European Working Session on Learning*, 164–178. Berlin: Springer-Verlag.

Cost, S., and Salzberg, S. 1993. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning* 10(1):57–78.

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

Fayyad, U. M., and Irani, K. B. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th Int. Joint Conference on Artificial Intelligence*, 1022–1027. Morgan Kaufmann.

Good, I. J. 1965. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. M.I.T. Press.

Holte, R. C. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning* 11:63–90.

John, G.; Kohavi, R.; and Pfleger, K. 1994. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh Int. Conference*, 121–129. Morgan Kaufmann.

Kohavi, R.; John, G.; Long, R.; Manley, D.; and Pfleger, K. 1994. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, 740–743. IEEE Computer Society Press. `http://www.sgi.com/Technology/mlc`.

Maass, W. 1994. Efficient agnostic PAC-learning with simple hypotheses. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, 67–75.

Murphy, P. M., and Aha, D. W. 1996. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn`.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Los Altos, California: Morgan Kaufmann.