Generalized Skewing for Functions with Continuous and Nominal Attributes

Soumya Ray SRAY@CS.WISC.EDU
David Page PAGE@BIOSTAT.WISC.EDU
Department of Computer Sciences and Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison, WI 53706

Abstract

This paper extends previous work on *skewing*, an approach to problematic functions in decision tree induction. The previous algorithms were applicable only to functions of binary variables. In this paper, we extend skewing to directly handle functions of continuous and nominal variables. We present experiments with randomly generated functions and a number of real world datasets to evaluate the algorithm's accuracy. Our results indicate that our algorithm almost always outperforms an Information Gain-based decision tree learner.

1. Introduction

Decision tree learners, such as ID3 (Quinlan, 1983) or CART (Breiman et al., 1984a), use greedy strategies for tree induction. Given a set of examples and a partial tree, a new split variable is selected based on a criterion such as GINI gain or Information Gain, such that the partitioned data has improved class label purity. Subsequently, this choice is not revisited. This strategy is computationally efficient, and often yields good results. However, it is known to suffer from the problem of "myopia". This refers to the situation that arises when, due to the nature of the target function, the split criterion is unable to discriminate between variables that are relevant and those that are not. In these cases, the learning algorithm makes a random choice, which is often incorrect.

The traditional method of handling myopia is through depth-k Lookahead (Norton, 1989), which exhaustively searches over the next k choices that can be made by the learning algorithm, and makes the best choice over this sequence. This approach has the disadvantage that the size of the search space is exponential in k, and the search has to be repeated at each choice point. Therefore, it is only feasible for very small values of k. Moreover, it is also prone to overfitting (Murthy & Salzberg, 1995).

Our previous work introduced an approach called *skewing* (Page & Ray, 2003; Ray & Page, 2004), which attempts to solve the myopia of tree learners by modifying the split selection function. Myopia is at its worst for "hard" Boolean functions. For these, no variable has gain even given a complete data set (one copy of each possible example), or given an arbitrarily large sample drawn according to the test distribution. The skewing approach relies on the following observation. Hard functions are hard only for *some distributions*. If we can obtain data drawn according to a different distribution, or *skew* the data we have, hard functions can become easier to learn. Given a large enough dataset, if the "skewed" distribution differs significantly from the original, it is possible to isolate the relevant features from the irrelevant ones, even when the target function is hard. Unlike Lookahead, the skewing algorithms introduced in previous work incur only either a constant or $O(n)$ runtime penalty (depending on which skewing algorithm is used) over a standard tree learner, such as ID3 using Information Gain, where n is the number of variables. We applied the approach to learn decision trees, and observed significant benefits in accuracy compared to ID3 using Gain when learning hard Boolean functions. Further, on average, skewing did not hurt the accuracy of ID3; consequently, on randomly generated functions, it resulted in a small but consistent improvement in performance.

Our previous work, however, was restricted to functions described by Boolean attributes. In principle, nominal and continuous variables can be transformed into binary-valued attributes using methods such as 1-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

of-N or binning. Indeed, we evaluated skewing on data sets from the UCI machine learning repository (Blake & Merz, 1998) by first transforming the data into binary valued attributes using such techniques. However, such transformations have potential problems. Information is lost when continuous variables are discretized using these techniques. Further, such transformations can greatly increase the dimensionality of the data. We have previously noted that the accuracy of the skewing approach tends to decrease as the dimensionality of the problem grows. Finally, these transformations impose strong dependencies among the transformed attributes, which were absent in the original data. These dependencies also can lead tree learners astray. Because of these problems, we would like the skewing approach to apply directly to functions described by nominal and continuous attributes. In this paper, we describe an extension to the previously proposed skewing algorithm that achieves this goal. We present experimental results for our algorithm on several real-world data sets, as well as results on randomly generated hard functions that are described by nominal and continuous variables. Our results indicate that our algorithm is able to effectively learn hard functions with nominal and continuous variables from modest amounts of data.

2. Review of the Skewing Approach

The motivation for the skewing procedure (Page & Ray, 2003) lies in the following observation. Consider a uniformly distributed dataset over a hundred features, $x_1, \ldots, x_{100}$, where the target function is two variable exclusive-or, say $x_{99} \oplus x_{100}$. This task is clearly very difficult for a top-down greedy tree learner; no variable will have non-zero gain even given a complete data set, or truth table. Now, suppose the data are distributed differently from uniform as follows: all variables are independent as in the uniform case, but every variable has probability $\frac{1}{4}$ of taking the value 0. In this case, it can be shown that with a large enough sample the class distribution among examples with $x_{99} = 0$ will differ significantly from the class distribution among examples with $x_{99} = 1$, in such a way that x_{99} will have gain, and similarly for x_{100}. On the other hand, every variable other than x_{99} or x_{100} is likely still to have nearly zero gain. Hence unless a highly improbable sample is drawn, a greedy tree learning algorithm will choose to split on either x_{99} or x_{100}, at which point the remainder of the learning task is trivial.

Following the example above, we want to reweight the data so that it exhibits significantly different frequencies. However, known methods of reweighting, such as boosting (Freund & Schapire, 1997), or methods of resampling, such as bagging (Breiman, 1996), do not achieve this (Page & Ray, 2003). Instead, we reweight by picking a "favored setting", 0 or 1, for each variable. This setting is picked uniformly at random and independently for each variable. Then, the weight of each example where variables match their favored settings is multiplied by a constant for each matching variable. This reweighted data effectively simulates a sample from a different distribution, but it can magnify idiosyncracies in the original data. Therefore, we repeat the process some constant number of times with different favored settings for the variables, and search for a variable that consistently has gain. The selected variable is likely to be a part of the target function. Yet, in contrast to lookahead, the run-time has been increased only by a small constant.

When the total number of variables grows large, the accuracy of the preceding algorithm drops. Because weights are being multiplied across all variables, with many variables, some data points get extremely high weights due to chance. The algorithm then overfits these data points. To address this, we proposed an alternative called "sequential skewing" that skews based on one variable at a time (Ray & Page, 2004). Sequential skewing incurs a $O(n)$ run-time penalty over Gain, in contrast to the constant time penalty incurred by the original algorithm. However, for high-dimensional problems, we observed that sequential skewing was more accurate than the original algorithm.

3. Generalized Skewing

The algorithms described above are applicable to binary-valued variables only. In this section, we describe how the idea of hard function extends to functions described over continuous variables. Then we describe how we extend skewing to handle this case. Next, we describe our approach to the nominal case.

3.1. Continuous Variables

When examples have continuous variables, algorithms such as C4.5 (Quinlan, 1997) and CART (Breiman et al., 1984b) determine a set of possible splitpoints from the data for every such variable. They then consider partitioning the data into two sets based on the criteria $x < s$ and $x \geq s$, for every continuous variable x and every splitpoint s. The (x, s) combination with the highest gain is then selected. The concept of hard functions—those for which relevant variables show no gain—arises in this continuous setting in a straightforward way. Consider a function such that $\Pr(f = 1|x < s) = \Pr(f = 1)$ for every continuous

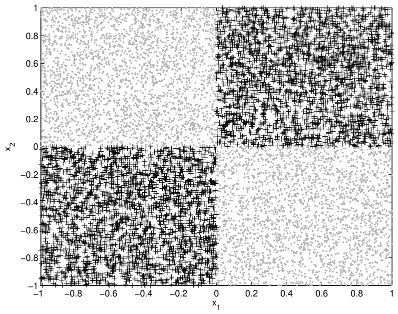

Figure 1. A hard function over two continuous variables. The function is defined by $f = (x_1 \cdot x_2) > 0$. For every possible splitpoint for x_1 or x_2, about half the examples less than the split are positive while half are negative, and the same is true for examples greater than the split. Note that in real-world problems, there are also likely to be irrelevant variables, not shown here.

variable x and every possible split s. Such a function is shown in Figure 1. "Chessboard" functions are familiar examples of such functions. In such cases, the correct (x, s) pair has gain only by chance according to measures like Information Gain or GINI Gain.

We can generate some such hard functions as follows. Let each continuous variable take on values in $(-1, 1)$ with the correct splitpoint at 0, and map values greater than 0 to 1 and values less than 0 to 0. This maps an example over continuous values to an example over Boolean values. Now if the Boolean valued examples are labeled according to a hard Boolean function, then the corresponding labels over the continuous examples creates a hard function as well. Note that this procedure implies that each Boolean hard function can be used to generate infinitely many hard functions over continuous variables. This procedure will not, however, generate many other hard continuous functions, such as those with multiple splitpoints for each axis.

A complication arises with hard continuous functions, such as chessboard functions, that does not arise with hard Boolean (or nominal) functions. Since in practice we have only a finite training sample, for any variable x, relevant or irrelevant, a splitpoint close enough to the maximum or minimum value for x almost always shows gain. To see this, let x be a continuous variable, and suppose without loss of generality that the example with the highest value for x is positive. Assuming no other example takes exactly this same value for x, splitting between this value and the second highest value will yield a pure node and hence provide non-zero gain. If the example with the second highest value for x happens to be positive also, the gain will be yet higher. These "spurious splits" are

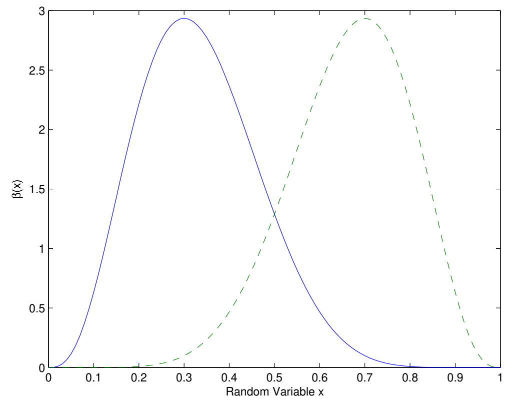

Figure 2. Beta distributions with $a = 4$ and $b = 8$ (solid line) and with $a = 8$ and $b = 4$ (dashed line). Larger values of a and b result in a higher peak and stronger skew.

unique to the case of functions described by continuous variables. Therefore, hard continuous functions are especially hard to learn for greedy tree learners, because not only do the correct splitpoints not show gain, some spurious splitpoints almost always do. In other words, these functions are hard to learn even if no irrelevant variable is present.

Applying the sequential skewing approach to functions defined over continuous variables is not as straightforward as the case for Boolean variables, because the relevant splitpoints are unknown. If they were known, this case would be identical to the case of functions described by Boolean variables. We could make the distribution around a split s different by choosing "greater than s" as the "favored setting", and reweighting accordingly. Since the correct splitpoint is unknown, we attempt to alter the input distribution so that it is asymmetric about *every* splitpoint. Further, we wish to down-weight the (possibly) spurious splits – those with extreme values for the variable. An appropriate distribution that has these properties, and is familiar in machine learning, is the β distribution with parameters a and b such that $a \neq b$. The probability density function for the beta distribution, for any $a, b > 0$, is defined on $x \in (0, 1)$ as follows:

$$\beta_{a,b}(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$$

Here $\Gamma(y) = \int_0^\infty x^{y-1} e^{-x} dx$. Figure 2 shows two example β distributions. Other asymmetric unimodal distributions may work as well, provided they have relatively low probabilities near the extreme values, but we have not tested other distributions.

To skew based on a β distribution, we rank the values of the continuous variable x in our data. The ranks are then translated uniformly into the $(0, 1)$ interval. An example is reweighted with the value of the β pdf at the translated rank of the value taken by variable x in that example. Translating the ranks instead of values ensures that outliers in the data will not affect the spread of the weights. Further, in practice, we use two distributions to reweight the data: $\beta_{a,b}$ and $\beta_{b,a}$. This is for two reasons: first, if the true splitpoint is translated to the mean of $\beta_{a,b}$, it is possible for it to not show gain. Using two distributions prevents this effect, since the mean of $\beta_{a,b}$ is not the same as that of $\beta_{b,a}$. Second if the true splitpoint is translated to a value close to 1, we may not discover it using $\beta_{a,b}, a < b$, since it down-weights this region.

3.2. Nominal Variables

When considering a nominal variable as a split variable, one possibility is to consider the entropy before and after partitioning the data on every value taken by the variable, as done by C4.5. This approach is biased towards variables with many values; therefore, in practice, an ad-hoc corrective factor is introduced based on the number of values a variable can take. The combined split criterion is known as *GainRatio* (Quinlan, 1997). A second possibility is to consider possible subsets of values of the variable, and introduce binary splits of the form $x \in S$ and $x \notin S$, where S is some subset of values for x. This is the procedure followed by the CART algorithm. In this case, the subset S of values which gives the highest gain for some nominal variable can be computed using an efficient algorithm (Breiman et al., 1984b). In either case, a variable x shows gain on the data iff for some value v, $\Pr(f = 1|x = v) \neq \Pr(f = 1)$. In our algorithm, we adopt the procedure followed by CART. We note that this procedure can also be used by the C4.5 program if invoked with a special command-line argument.

The concept of Boolean hard functions extends to functions over nominal variables in a straightforward way. Consider a function f such that $\Pr(f = 1|x_i = v_j) = \Pr(f = 1)$ for every variable x_i and every value v_j. In such a case, none of the variables will show any gain (according to either the GainRatio criterion or the CART criterion), and the function will be hard to learn for a greedy tree learner when variables irrelevant to the target are present. An example of such a function is shown in Figure 3. Some such nominal hard functions can be generated using the following procedure. Assume that each nominal variable takes on $2r$ values. Divide these values into two random sets each of size r. Map one of the sets to 0 and the other to 1. This establishes a mapping from an example over nominal values to an example over Boolean values. Now if the Boolean-valued examples are labeled according to a hard Boolean function, then the corresponding labels over the nominal examples creates a hard function as well. Note that this procedure implies that each Boolean hard function can be used to generate many hard functions over nominal variables.

color	*shape*	f
blue	circle	0
blue	ellipse	0
blue	square	1
blue	rectangle	1
green	circle	0
green	ellipse	0
green	square	1
green	rectangle	1
red	circle	1
red	ellipse	1
red	square	0
red	rectangle	0
magenta	circle	1
magenta	ellipse	1
magenta	square	0
magenta	rectangle	0

Figure 3. A hard function over two nominal attributes, *color* and *shape*. The function is defined by $f = color \in \{blue, green\} \oplus shape \in \{circle, ellipse\}$. For every subset of values for *color* or *shape*, half the examples are positive and half are negative, as is the case for f, so that neither attribute has gain. Note that in real-world problems, there are also likely to be irrelevant variables, not shown here.

When given a function over nominal variables, we can employ the sequential skewing approach as follows. As in the Boolean case, we would like to calculate Gain under a data distribution that is significantly different from the input. One way to obtain (i.e., simulate) such a distribution is by altering the input distribution $\Pr(x = v)$ for each variable x and value v. We first choose a random ordering over the values of the variable, x, we are using to skew. Each value is then mapped to a "weighting factor" inversely proportional to the rank of that value in the random ordering chosen. Weighting factors are normalized so that they sum to 1. An example is then reweighted with the weight of the value taken by x in that example. If x has only two values, this procedure is identical to the sequential skewing algorithm (Ray & Page, 2004) with the parameter $s = \frac{2}{3}$. In this special case, one reordering is sufficient (the other order would produce the same results). However, when x has more than two values, and the distribution of values for x is not uniform in our data, the ordering of values chosen the first time may not produce a significantly different distribution. Therefore, the ordering and weighting procedure is repeated a small constant number of times, with a different ordering of values chosen each time.

To further ensure a difference, one of the orderings we use is antagonistic to the input, i.e., the values are ordered so that the least prevalent gets the most weight. After reweighting, we follow the sequential skewing algorithm (Ray & Page, 2004) to combine the results over different distributions and pick a split variable.

To apply this algorithm in practice, we may need to deal with missing values. When calculating gain, we follow the approach of C4.5: we estimate a distribution over the values of the variable from the data points where the values are observed, and use this to weight the gain. We may also need to reweight data points where the variable we are skewing has its value missing. In our approach, we assign these points the "expected skew". For nominal variables, this is the distribution over the variable's values multiplied by the weight associated with each value. For continuous variables, this is the value of the β pdf at its mean.

4. Experiments

In this section, we present experiments comparing the performance of a tree learner using the Information Gain criterion and the Generalized Skewing criterion for selecting split variables. We first compare the accuracy of these methods on synthetic data, followed by results on various real-world datasets from the UCI repository. For all experiments, the base tree learner is comparable to C4.5 with the "subset-splitting" option. For the synthetic data experiments, no pruning is needed since there is no variable or label noise. For the real-world data, we greedily post-prune the trees based on their accuracy on a held-aside validation set. The β distribution parameters input to the Generalized Skewing algorithm were $a = 8$ and $b = 16$. For nominal variables, values were reordered 5 times. These parameters were chosen before the experiments were performed and were held constant across all experiments. An important direction for future work is to measure the algorithm's sensitivity to these choices.

4.1. Experiments with Synthetic Data

In experiments with synthetic data, we test the continuous and nominal variable components of the Generalized Skewing algorithm separately. To test our approach for continuous variables, we generate examples described by 30 and 100 continuous variables. Each variable takes on values uniformly in $(-1, 1)$. We randomly select 6 of these variables, and set the "true splitpoint" for these variables to 0. Each example is translated to a Boolean assignment by identifying $x_i < 0$ with 0 and $x_i >= 0$ with 1 for the 6 relevant variables, and labeled according to a Boolean function over 6 variables. If the labeling function is hard, the corresponding function over continuous variables will also be hard. We generate independent train and test sets using this procedure for each target function we test. Each test set has 10,000 examples, and we vary the size of the train set to generate a learning curve.

Figures 4 and 6 show learning curves measuring accuracy using the two split criteria as the size of the training sample is varied, when the examples are labeled according to Boolean functions drawn uniformly at random from the set of all Boolean functions. Figures 5 and 7 show the results when the labeling functions are drawn uniformly at random from the set of Boolean functions over 6 variables that are hard for greedy tree learners. For each sample size, we average over 100 runs, each with a different target and with a different sample of the specified sample size. From the figures, we observe a consistent modest improvement in accuracy over the standard Gain algorithm when the target is drawn uniformly at random, while there is a large improvement when the target is hard.

Next, we test our algorithm for nominal variables. Examples are generated according to a uniform distribution over 30 and 100 nominal variables. Each nominal variable can take on $2r$ values, r randomly chosen from 1 to 5. We partition the values of 6 randomly chosen variables into two equal sized disjoint sets, S_0 and S_1. Each example is translated to a Boolean assignment by identifying values in S_0 with 0 and values in S_1 with 1. Each example is then labeled according to a Boolean function over 6 variables, where each Boolean variable corresponds to a chosen nominal variable. If the Boolean function labeling the examples is hard to learn from examples described by Boolean variables, the corresponding function described by nominal variables will also be hard to learn for a greedy tree learner. As before, we generate independent train and test sets using this procedure for each target function.

Figures 8 and 10 show learning curves for the accuracy of the tree learner using the two split criteria as the size of the training sample is varied, when the labeling functions are drawn uniformly at random from the set of all Boolean functions. Figures 9 and 11 show the corresponding results when the labeling functions are drawn uniformly at random from the set of Boolean functions over 6 variables that are hard for greedy tree learners. In addition to the two methods being tested, we also report the accuracy of ID3 using the sequential skewing algorithm (Ray & Page, 2004), when examples are described by 30 nominal variables. Since this algorithm is only applicable to Boolean variables, we first transform each training sample so that each nominal

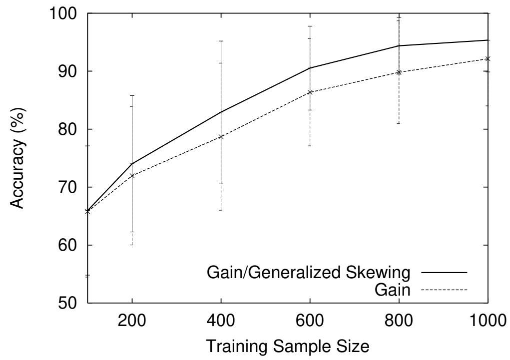

Figure 4. Random Continuous Targets, 30-v examples

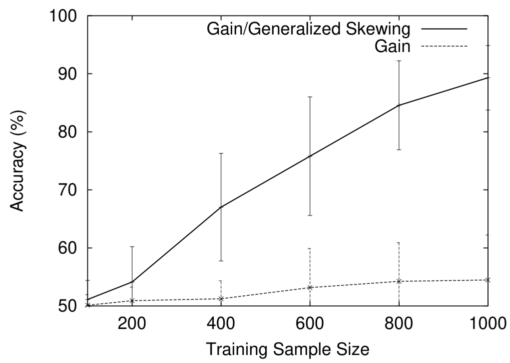

Figure 5. Hard Continuous Targets, 30-v examples

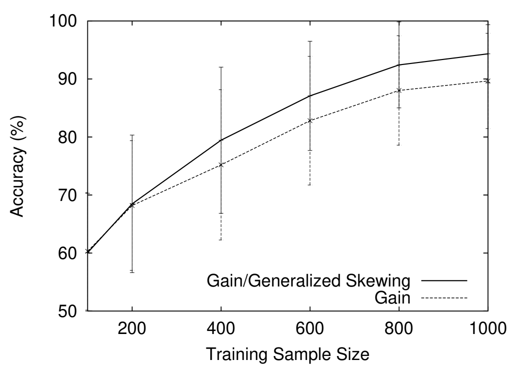

Figure 6. Random Continuous Targets, 100-v examples

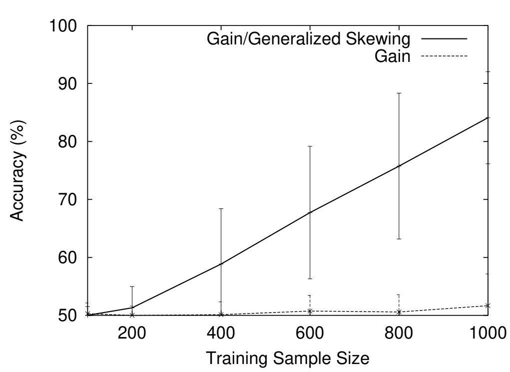

Figure 7. Hard Continuous Targets, 100-v examples

variable with N values is represented by N Boolean variables (the 1-of-N transformation).

From the figures, we observe that the skewing approach with nominal variables is able to outperform the standard Gain-based approach both when the targets are randomly drawn and when they are hard. The improvement in accuracy is large when the target is hard, and is smaller, though consistent, for a randomly drawn target function. We also observe that the sequential skewing algorithm, while outperforming Information Gain on hard targets, does not do well on random targets. Further, the accuracy improvement for this algorithm on hard targets is lower than that of the proposed algorithm. This behavior is likely caused by the fact that the sequential skewing algorithm is working on higher dimensional data sets as compared to the Generalized Skewing algorithm or the standard Gain criterion ($\sim$180 variables on average, compared to 30 for the others). When the target is hard, we observe that there is some variability in the accuracy of Generalized Skewing algorithm. This variability seems mainly to be a consequence of sample size. With a training sample size of 5000 examples, we observed that the Generalized Skewing algorithm obtained an accuracy of $96 \pm 4\%$ in the 30-variable case, and $90 \pm 10\%$ in the 100-variable case.

We have also run the Generalized Skewing algorithm successfully on small "Chessboard" functions with multiple splits per axis. This result has limited practical significance, but it indicates that the proposed algorithm can successfully handle functions that would be very hard to learn with standard tree learners. Thus, we conclude that, in the ideal (no noise) case, given modest amounts of data drawn according to a uniform distribution, our approach is almost always able to recover the target function, even when the target is hard for a standard decision tree learner. Further, it scales well to high dimensional problems – there is only a small drop in observed accuracy as the example sizes are increased from 30 to 100 dimensions.

4.2. Experiments with Real World Data

In this section, we present the results of experiments on a number of real-world data sets. Among our data sets, we chose several where tree induction algorithms and logistic regression are known to perform poorly (Perlich et al., 2003). Note that we do not know if the target function is hard for any of these data sets. However, the fact that greedy tree induction algorithms and logistic regression (which employs a linear inductive bias) do not have very high accuracy on some of them indicates that it is possible

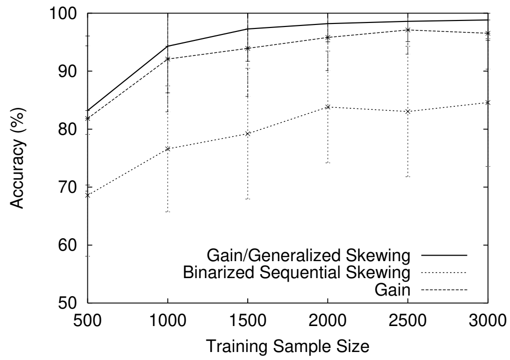

Figure 8. Random Nominal Targets, 30-v examples

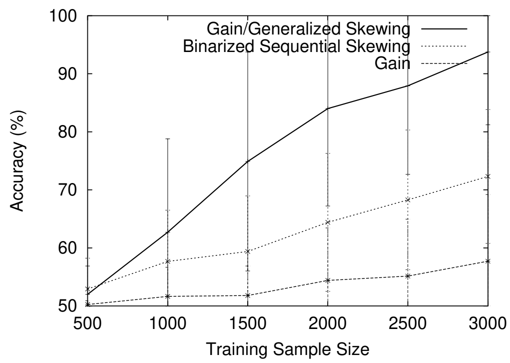

Figure 9. Hard Nominal Targets, 30-v examples

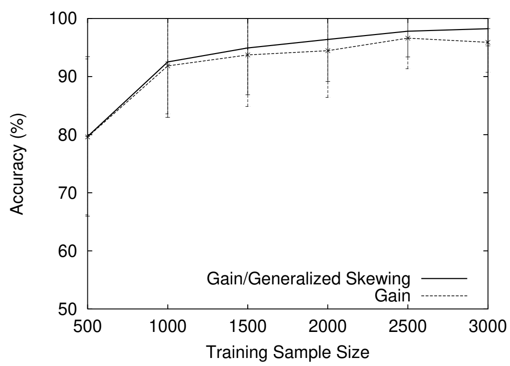

Figure 10. Random Nominal Targets, 100-v examples

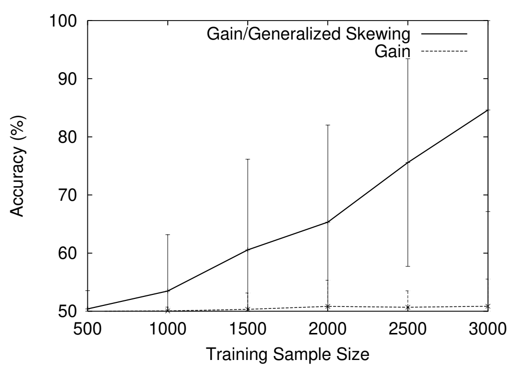

Figure 11. Hard Nominal Targets, 100-v examples

that a subfunction in the target (or the target itself) may be hard. The datasets we use are: Contraceptive Method Choice (CMC), German Credit (German), Cleveland Heart Disease (Heart), voting-records (Voting), pima-indians-diabetes (Diabetes), breast-cancer-wisconsin (BCW), Glass, Promoters, Yeast, Abalone, Credit and Spam from the UCI repository (Blake & Merz, 1998); Internet Shopping (Int. Shop.) and Internet Privacy (Int. Priv.) from previous work (Perlich et al., 2003); and ribosome binding site prediction in *E. coli* (RBS) (Opitz & Shavlik, 1996).

We compare standard Gain and Gain with Generalized Skewing. We use 10-fold stratified cross-validation (except Promoters and Glass, which are very small, where we use 3 folds) and report the average accuracy of each algorithm and the size of the tree constructed by each. We also report the ratio of the time taken by Generalized Skewing to the time taken by Gain to induce a tree on each data set. To investigate whether there is any benefit in applying the skewing approach directly to functions over continuous and nominal variables, we also report the accuracy obtained by Gain and sequential skewing, each working on a binarized version of each data set. This version was obtained by first binning each continuous variable into 16 bins, and then representing each nominal variable with v values using $\lceil \log_2 v \rceil$ binary variables. These results are shown in Table 1.

From Table 1, we observe that Gain with Generalized Skewing outperforms standard Information Gain in all but 3 cases. Further, in all but 4 cases, the Generalized Skewing algorithm constructed a tree that was smaller than the standard algorithm (sometimes much smaller). In one case (Promoters), both methods constructed identical trees in all iterations. While the individual accuracy differences are not significant, using a sign test across datasets, we find that the Generalized Skewing algorithm is significantly different from Gain at the 95% confidence level. Further, Generalized Skewing always induces a tree within a factor of $5n$ of the time taken by standard Gain, as predicted, where n is the number of attributes (the factor 5 arises from the number of times each nominal variable is skewed). Comparing Generalized Skewing to sequential skewing, we observe that Generalized Skewing has improved accuracy in all cases but one. Though sequential skewing is able to outperform Gain on binarized data in most cases, it often does not do even as well as standard Gain. Thus, we conclude that if the data contains continuous or nominal attributes, it is preferable to handle them directly, using Generalized Skewing, rather than first binarizing and then employing sequential skewing.

Table 1. Accuracies and tree sizes of Gain (G) and Gain with Generalized Skewing (GS) on real-world data sets. Also shown are accuracies of sequential skewing (SS) and Gain (BG) on the binarized versions of each data set, and the time taken by Generalized Skewing to induce a tree for each data set, relative to the same time for Gain.

Data Set	Accuracy (%)				Tree size (Nodes)		Time Ratio
	GS	G	SS	BG	**GS**	G	GS/G
Glass	82.28	81.64	78.04	74.77	6.4	6.6	2.4
BCW	94.83	94.80	93.57	92.42	14.5	14.3	1.1
Promoters	78.30	78.30	74.70	74.70	15.0	15.0	1.0
Credit	84.93	84.78	84.64	84.64	11.8	22.5	11.8
RBS	83.48	81.73	82.31	81.30	20.6	23.6	25.6
Heart	75.25	74.92	74.59	72.94	23.8	23.8	3.7
Voting	96.09	95.40	96.32	95.63	20.0	24.0	9.6
Diabetes	72.08	72.00	72.01	72.14	31.5	35.2	12.3
German	74.20	72.50	72.50	71.10	57.1	72.7	63.0
Spam	92.39	92.13	85.35	85.35	96.6	96.9	25.6
Abalone	77.28	76.00	75.46	75.10	92.1	120.2	10.1
Yeast	69.95	69.41	68.40	66.71	167.6	177.6	10.5
Int. Priv.	63.87	64.42	63.33	63.32	207.0	179.8	40.0
CMC	69.79	67.35	64.71	63.54	147.6	217.1	14.1
Int. Shop.	66.33	66.68	65.04	64.74	384.6	410.4	27.2

5. Conclusion

We have extended the skewing approach to handle hard functions over nominal and continuous variables. Experiments with synthetic and real data demonstrate that the new Generalized Skewing algorithm is usually more accurate than standard Information Gain, while also constructing smaller trees. A primary direction for future work is to test the skewing approach in biomedical domains where we expect that hard functions are likely to occur. To do this, we are incorporating this algorithm into a Bayesian Network learning framework, that we can use to learn genetic regulatory networks from data.

Acknowledgements

The first author was supported by NIH Grant 1R01 LM07050-01 and by grants from the University of Wisconsin Graduate School. The authors wish to thank the anonymous reviewers for their helpful comments.

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984a). *Classification and regression trees.* Monterey: Wadsworth and Brooks/Cole.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984b). *Classification and regression trees.* Wadsworth International Group.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and its application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Murthy, S. K., & Salzberg, S. (1995). Lookahead and pathology in decision tree induction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1025–1031). Morgan Kaufmann, San Francisco, CA.

Norton, S. (1989). Generating better decision trees. *IJCAI-89* (pp. 800–805). Los Altos, CA: Kaufmann.

Opitz, D. W., & Shavlik, J. W. (1996). Generating accurate and diverse members of a neural-network ensemble. In D. Touretzky, M. Mozer and M. Hasselmo (Eds.), *Advances in neural information processing systems*, vol. 8, 535–541. Cambridge, MA: MIT Press.

Page, D., & Ray, S. (2003). Skewing: An efficient alternative to lookahead for decision tree induction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, San Francisco, CA.

Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree Induction vs. Logistic Regression: A Learning-curve Analysis. *Journal of Machine Learning Research*, *4*, 211–255.

Quinlan, J. (1983). Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonnel and T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga.

Quinlan, J. (1997). *C4.5: Programs for machine learning.* Kaufmann.

Ray, S., & Page, D. (2004). Sequential skewing: An improved Skewing algorithm. *Proceedings of the 21st International Conference on Machine Learning.* Morgan Kaufmann, San Francisco, CA.