Efficient Determination of Dynamic Split Points in a Decision Tree

David Maxwell Chickering
Microsoft Research
Redmond WA, 98052-6399
dmax@microsoft.com

Christopher Meek
Microsoft Research
Redmond WA, 98052-6399
meek@microsoft.com

Robert Rounthwaite
Microsoft Research
Redmond WA, 98052-6399
robertro@microsoft.com

Abstract

We consider the problem of choosing split points for continuous predictor variables in a decision tree. Previous approaches to this problem typically either (1) discretize the continuous predictor values prior to learning or (2) apply a dynamic method that considers all possible split points for each potential split. In this paper, we describe a number of alternative approaches that generate a small number of candidate split points dynamically with little overhead. We argue that these approaches are preferable to pre-discretization, and provide experimental evidence that they yield probabilistic decision trees with the same prediction accuracy as the traditional dynamic approach. Furthermore, because the time to grow a decision tree is proportional to the number of split points evaluated, our approach is significantly faster than the traditional dynamic approach.

1 Introduction

Decision trees have proven to be useful tools for both solving classification tasks and for modeling conditional probability distributions. The literature is rich with studies of various decision-tree learning systems. Examples of such systems include CLS (Hunt, Marin and Stone [8]), CART (Breiman, Friedman and Olshen [1]), ID3 (Quinlan [11]), C4.5 (Quinlan [10]) and SLIQ (Mehta, Agrawal and Risanen [9]) to name just a few.

When a *predictor* variable (that is, a variable that is included as a decision in the tree) is continuous, the learning algorithm (conceptually) converts the values of that variable into two or more discrete bins. For example, a node in a decision tree may test whether or not the value of a continuous predictor is greater than some threshold; defining the threshold effectively converts the continuous variable into a binary one.

Methods for discretizing continuous predictor variables can generally be classified as either *static* or *dynamic*. Static discretization methods discretize all of the continuous predictors before the decision tree is learned, and the same "buckets" that result are used throughout the tree. Dynamic discretization methods, on the other hand, determine the discretization dynamically as the decision tree is being constructed. Thus, the threshold(s) used in one part of the tree for a particular continuous variable can be different from those used in another part of the tree.

Dougherty, Kohavi, and Sahami [5] examine the relative performance of various discretization techniques when learning both naive-Bayes models and decision trees.[1] Rather surprising, they show that the performance of C4.5 across a number of datasets did not degrade significantly when a particular static discretization algorithm was used instead of the usual dynamic discretization algorithm of C4.5.

Although we have not performed a formal study similar to Dougherty et al. [5], we have found that in the domains we work with, dynamic discretization is, in fact, noticeably superior to static discretization. This discrepancy can perhaps be explained because we usually both learn and use decision trees for the purpose of *prediction*, whereas the algorithms mentioned above learn and use decision trees for the purpose of *classification*. The distinction is that for prediction, the goal of the learning algorithm is to identify the correct conditional probability distribution of the target variable, whereas the goal of classification is to identify the correct label of the target variable. As described by (e.g) Cowell [4] in the context of Bayesian networks, models that result from optimizing these two criteria may be very different.

A second reason to prefer dynamic discretization is that the discretization itself can be interesting to a data analyst. For example, suppose we learn a decision tree for predicting whether or not a customer will buy a certain product, based on known attributes for that customer. It could be interesting that a good discretization of salary for males is different than a good discretization for females, and that by exam-

[1]Daugherty et al. [5] use the terms "global" and "local" to refer to static and dynamic, respectively.

ining that difference the analyst might gain insight into the domain.

The traditional dynamic approach to discretization is to allow splits on arbitrary values of predictor variables. In practice, because tree growing is directed by a scoring criterion, algorithms typically only consider predictor values that yield different scores. For example, algorithms often use values that actually occur in the data, or midpoints between pairs of consecutive values that actually occur in the data. To identify these potential split points efficiently, algorithms typically consider predictor values in sorted order. There are two standard methods: (1) for each leaf under consideration for a split, re-sort the data that "drops down" to that leaf by the values for each continuous predictor, or (2) maintain a sorted list of record pointers for each continuous predictor, and propagate the appropriate portions of the list to the children when a split is applied. Method (1) requires numerous expensive sorts that can significantly slow down the algorithm, particularly as the data grows large. Method (2) requires space that scales with the product of the number of records in the data and the number of continuous predictors. Furthermore, the initial sorts of the data may require a prohibitive amount of time.

In this paper, we present a number of simple methods for performing dynamic discretization. As opposed to the traditional approaches, these methods efficiently identify only a small number of potential split points for each continuous predictor variable. Unlike the traditional approach—where the number of split points depends on the number of values in the data, and where sorting contributes super-linear overhead in either time or space—our methods scale linearly in both the number of continuous predictor variables and the size of the data. Two of our methods generate split points using simple summary statistics from the data; these statistics can be gathered in time that is linear in the size of the relevant data. The third method generates split points by dividing the predictor values into k-tiles, which for a constant number of split points can be accomplished in time that grows linearly with both the size of the relevant data and the number of continuous predictors. In Section 2, we introduce notation and provide details on the standard methods for dynamic discretization.

The paper is organized as follows. In Section 3, we describe our methods and discuss their time and space complexity. In Section 4, we provide experimental evidence that some of the proposed methods work as well as the standard methods when trees are evaluated using the prediction accuracy (i.e. the log-likelihood of a holdout set). Finally, in Section 5, we conclude with a discussion of future work.

2 Background and Notation

In this section, we provide background information about decision trees and present our notation.

A probabilistic decision tree T is a structure used to encode a conditional probability distribution of a target variable Y, given a set of predictor variables $\{X_1, ..., X_n\}$. The structure is a tree, where each internal node I stores a mapping from the values of a predictor variable X_j to the children of I in the tree. Each leaf node L in the tree stores a conditional probability distribution for Y given some subset of the values of the predictor values.

For a given set of predictor values $\{x_1, ..., x_n\}$, we obtain the probability $p(Y|X_1 = x_1, ..., X_n = x_n)$ by starting at the root of T and using the internal-node mappings to traverse down the tree to a leaf node. We refer to the mappings in the internal nodes as *splits*. When an internal node I maps values of the predictor variable X_j to its children nodes, we say that X_j is the *predictor* variable of node I, and that I *is a split on* X_j.

For example, Figure 1 shows a decision tree for a probability distribution $p(Y|X_1, X_2, X_3)$. In the Figure, X_1 and X_3 are continuous predictor variables defined on the real line, and Y and X_2 are binary (with values 1 and 2). Assume that we would like to traverse the decision tree to extract the probability $p(Y|X_1 = 12.3, X_2 = 2, X_3 = 2.4)$. We start at the root node, and see that this node is a split on X_2. Because $X_2 = 2$ in the conditioning set of our probabilistic query, the traversal moves next to the right child of the root. This node is a split on X_1, which equals 12.3 in the query, so we move next to the left child. Finally, because $X_3 = 2.4$ (and consequently greater than zero), we finish the traversal by moving to the right child which is a leaf node. The conditional probability is stored in the leaf, and we conclude that $p(Y = 1|X_1 = 12.3, X_2 = 2, X_3 = 2.4) = 0.2$ and $p(Y = 2|X_1 = 12.3, X_2 = 2, X_3 = 2.4) = 0.8$.

Algorithms for learning decision trees from data typically try to maximize a scoring criterion by repeatedly replacing leaf nodes by internal splits. The majority of the classification-tree learning algorithms greedily replace each leaf node with the split that yields the highest entropy in the data. After this initial "growing phase", the tree is then pruned back by greedily eliminating leaves using (e.g.) a holdout score on a test data set.

In the experiments presented in Section 4, we instead apply the greedy growing phase using a Bayesian scoring criterion described in detail by Chickering, Heckerman and Meek [2]. This criterion avoids over-fitting by (both explicitly and implicitly) penalizing model complexity, and consequently no pruning phase is needed.

For almost all scoring criteria, including entropy and the Bayesian criterion we use, the score for replacing a leaf

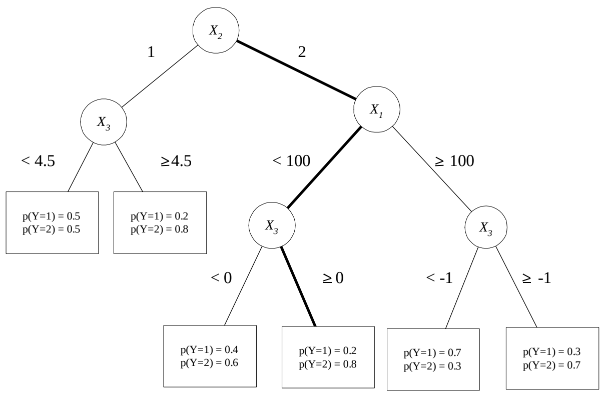

Figure 1. Example decision tree for the distribution $p(Y|X_1, X_2, X_3)$

node L by a split is a function of the subset of the training dataset that is said to be *relevant* to L: For any record in a training dataset, the case is "dropped down" the tree by traversing the tree the same way as was accomplished in the example above, using the values of the variables in the record. For every case that ends up at leaf node L, the case is said to be relevant to L.

For discrete-valued predictors, a typical candidate split considered by the learning algorithm is a binary split, where one child corresponds to exactly one of the discrete states, and the other child corresponds to all other states. Another type of discrete split is a complete split, where there is exactly one child corresponding to every value of the discrete predictor. The most general type of split maps subsets of the predictor's values to different children.

For continuous variables, the children of split nodes correspond to intervals of the predictor variable. In principle, for a given number of children (two in the example above), there are an infinite number of possible intervals that a split could define due to the fact that the interval boundaries are continuous. In the tree shown in Figure 1, for example, the left child node of the split on X_1 could correspond to $< c$ and the right child could correspond to $\geq c$, for any threshold c in the continuous range of predictor variable X_1. The number of interval-defining thresholds that can be distinguished by most scoring criteria, however, is limited by the number of examples in the training data that are relevant to the given leaf. Furthermore, Fayyad and Irani [6] prove that, for the entropy scoring criterion on a discrete target variable, only a restricted set of thresholds can ever achieve the maximum score, and consequently only this set need be considered when using a greedy search. As a result, split points are typically chosen as the midpoint between successive predictor values in the training data or as a specific predictor value in the training data.

For simplicity, we consider only *binary* splits on continuous predictors so we can concentrate our discussion on the problem of identifying a single split point (i.e. threshold). We note, however, that all of the methods for choosing a good split point can be used directly to choose multiple split points (see, e.g. Fayyad and Irani [6], for a discussion of methods for choosing multiple split points).

The standard method for choosing candidate split points for a split on leaf node L is as follows. For each continuous predictor X_i, we visit all of the records relevant to L *in sorted X_i order*, and consider for split points those values that are half-way between the X_i values in each pair of consecutive records. The optimization of Fayyad and Irani [6] allows us to avoid *scoring* all possible split points, but it still requires us to examine the records in sorted order.

There are two methods for handling the ordering requirement: (1) whenever a node N is considered for expansion, we can perform a separate sort of the data records relevant to N on each continuous predictor, or (2) we can pre-sort all of the records on each continuous predictor, retaining the sort information that can then be propagated through the tree.

Let m denote the number of records in the training data, and let m_L denote the number of these records that are relevant to node L in the decision tree. Let γ denote the number of continuous predictor variables. Method (1) requires time $O(\gamma \cdot m_L \log m_L)$ for each node L considered for splitting in the algorithm. For large γ and large datasets, this time overhead can be prohibitive. If the sort is done in-place on the data records, method (1) requires no additional space. Method (2) avoids the per-node sort, but incurs an initial $O(\gamma \cdot m \log m)$ time hit for the initial sort, and requires $O(\gamma \cdot m)$ space to retain the individual sort order for each continuous predictor. For large γ and large m, either the time overhead or space overhead (or both) may be prohibitive.

3 Some Efficient Discretization Methods

In this section, we describe three simple methods for identifying split points that have time and space complexities that grow linearly with the number of relevant records. All three methods use a common approach to discretization which we call the *quantile approach*. Using this approach, we assume a distribution over split point values at each node in the decision tree, and choose the split points such that this distribution is divided into equal-probability regions.

3.1 Gaussian Approximation

The first method, which we call the *Gaussian approximation* method, requires only that the mean and standard deviation of each continuous predictor be known for the data records relevant to each leaf node in the tree being considered for a split. These statistics can be gathered easily whenever the data is being "dropped down" to the children of a newly-formed split. For each record dropped down to leaf node L, a running sum and running sum of squares for each continuous predictor is updated within L, and when all data has reached L the mean and standard deviation are derived from these sums.

We derive the split points for each continuous predictor X_j at leaf node L as follows. First, we choose the number of split points k to consider. This choice may be made dynamically (i.e. via model selection), or as is shown to work reasonably in the following section, we can pre-define k for all predictors in all nodes before running the learning algorithm. Second, we choose the split points that yield $k+1$ equal-density regions of the domain of X_j, under the (usually bad) assumption that X_j is distributed normally in the cases relevant to L. In particular, let $\{c_1, ..., c_k\}$ denote the set of k split points. We choose c_i as

$$c_i = \mu_L + \sigma_L \cdot \Phi^{-1}(\frac{i}{k+1})$$

where Φ^{-1} is the inverse of the cumulative distribution function of the standard Gaussian, and μ_L and σ_L are the mean and standard deviation, respectively, of the values of X_j relevant to L.

Given γ continuous predictors and m_L relevant records at leaf L, this calculation requires $O(\gamma)$ space to store the Gaussian sufficient statistics and $O(\gamma \cdot m_L + k)$ time to identify the k split points.

3.2 Uniform Approximation

The second method for computing split points is similar to the first except that we choose equal-density regions of the predictor domain under the (again, usually bad) assumption that the predictor is distributed uniformly between its minimum and maximum value. As in the Gaussian-approximation method, we can easily accumulate the necessary statistics for each predictor—which in this case are simply the minimum and maximum values—as the data is dropped down to children nodes during learning. For a continuous predictor X_j with minimum and maximum values $\min(j)$ and $\max(j)$, respectively, the uniform-approximation approach chooses, for a given k, the split points $\{c_1, ..., c_k\}$ such that:

$$c_i = \min(j) + i \cdot \frac{\max(j) - \min(j)}{k+1}$$

The space and time complexities of this approach are identical to those of the Gaussian-approximation method.

3.3 K-tile method

In our third and final method for computing split points, we choose the continuous split points using k-tiles. This corresponds to the quantile approach using the empirical distribution function. That is, for continuous predictor X_j, we choose split points $\{c_1, ..., c_k\}$ such that there are (approximately) $m_L \cdot \frac{i}{k+1}$ records with $X_j \leq c_i$ and $m_L \cdot \frac{k-i}{k+1}$ records with $X_j > c_i$.[2] For example, if $k = 1$, the method simply chooses the median value of X_j in the records relevant to L. As in the previous methods, we can pre-compute k for all nodes and all predictors.

There are well-known algorithms that identify the jth-smallest value in a list of m elements. In Section 4, we use a well-known implementation that runs in time $O(m)$ on average rather than a more complicated (but also well-known) implementation that runs in time $O(m)$ in the *worst* case. See (e.g.) Cormen, Leiserson and Rivest [3] for a description of both implementations. These algorithms operate in space that is $O(m_L)$, although the average-case $O(m_L)$ implementation can work on the original data records in-place, resulting in no additional space requirement. For small k, we can call such an algorithm k separate times for each continuous predictor, yielding a total time complexity of $O(k \cdot \gamma \cdot m_L)$ to identify all split points for all predictors at leaf L. If k is of the same order as $\log m_L$, it is probably faster to simply sort the records for each X_i. In this case, we have essentially the same algorithm that is commonly used in practice, except that we consider only k split points instead of all possible ones.

4 Experiments

In this section, we present experimental results that suggest that two of our methods of selecting split points result in trees that have better prediction accuracy than trees that are learned when considering all possible split points. In addition, our methods significantly reduce the time needed to learn trees.

We ran experiments using five real-world datasets. Our selection of datasets was influenced by the fact that for small data sets, our optimizations will not provide a significant *absolute* speedup because the standard techniques will already provide adequate performance. We chose the first three datasets because they were the largest data sets from the UC Irvine repository that were included in the study by Mehta et al. [9]. These datasets were *German*, *Hypothyroid*, and *Sick-Euthyroid* and contain 1000, 3163, and 3163 records, respectively. In terms of speed, even these largest datasets are too small to warrant using our approach; as we see below, even when we use the full-blown sorting ap-

[2] There may be no such split point that satisfies these conditions exactly.

proach, trees in these domains can be learned in less than three seconds. We include the results here simply to provide more evidence that we can achieve good prediction accuracy without considering all possible split points.

The fourth dataset, *Census*, was extracted from the United States Census Bureau. The data consists of the values for a set of 37 demographic variables for approximately 300,000 citizens. Eleven of these demographic variables are continuous. The fifth dataset, *Media Metrix*, contains demographic and internet-use data for about 5,000 people during the month of January 1997. There are 24 demographic variables in this dataset, six of which are continuous. There are 13 categorical variables that indicate the type of the web page (e.g. educational, news).

The first three (small) datasets were originally collected for the purpose of predicting a single output variable. For the German dataset, the goal is to predict whether or not a person has good credit. For Hypothyroid and Sick-Euthyroid, the goal is to predict whether or not a person has a specific medical problem. As a result, we evaluate our approaches by learning a single decision tree for the appropriate target variable in these domains.

In contrast, Census and Media Metrix are datasets collected with no such obvious prediction task; for such datasets, probabilistic decision trees are important for exploratory data analysis and density estimation (e.g. Heckerman, Chickering, Meek, Rounthwaite and Kadie, 2000). In particular, we assume that every variable in the domain is a target variable, and we learn a separate decision tree for each. As a result, these datasets effectively provide an “independent” learning problem for each variable.

To evaluate the performance of our discretization methods on a particular dataset, we first divided the dataset into a training set (consisting of a random sample of roughly 70 percent) and a test set (consisting of the remainder). For the Census dataset, we further sub-sampled the training set so that our trees were learned using roughly 20,000 records. Next, we ran a series of trials to evaluate predictive accuracy, relative to the traditional sorting method, as a function of the number of split points k. In particular, for the ith trial, we (1) set $k = 2^i - 1$, (2) learned a decision tree for each target variable in the domain, and (3) evaluated the average relative increase in predictive accuracy that resulted from using k split points over sorting the data and using all possible split points.

To measure the predictive accuracy of a tree built to predict Y_j, we took the average log probability or log density that the tree assigned to the given value y_j in each test case. The average relative increase in predictive accuracy for a particular dynamic split algorithm was computed as follows. Let s_j denote the predictive accuracy of tree j using the traditional sorting method, and let $a_j(k)$ denote the predictive accuracy of the jth tree using the given method with k split points. The relative increase in predictive accuracy for tree j, $r_j(k)$ is defined as

$$r_j(k) = \frac{a_j(k) - s_j}{|s_j|}$$

For a discrete target variable Y_j, r_j is simply the relative increase in the average number of bits needed to encode an observation of Y_j.

The average increases reported in the figures for Census and Media Metrix below are the simple average of $r_j(k)$ for those trees that had a continuous split using at least one method. For the Census data, there were 31 such trees, and for the Media Metrix data, there were 16 such trees.

As mentioned in Section 1, we used a greedy learning algorithm in conjunction with a Bayesian scoring criterion. We grew trees using binary splits only. For discrete predictors, we used the method where the first child corresponds to one state and the second child corresponds to all others.

For discrete target variables, we learned multinomial distributions in the leaf nodes, and used a flat Dirichlet parameter prior. For continuous target variables, we chose either a Gaussian distribution or a log-Gaussian distribution ahead of time for all leaves of the tree; the choice was made based on which distribution had a better maximum-likelihood fit on the marginal. If we chose a log-Gaussian distribution, we implemented the learning algorithm by simply taking the log of the target variable in each case. Next, we standardized the (possibly logged) data so that the target had mean zero and standard deviation one. We learned Gaussian distributions in the leaf nodes for the standardized values, using a Normal-Wishart parameter prior that had a prior mean of zero (equivalent sample size one) and a prior precision of one (equivalent sample size two). After learning these Gaussian distributions, we transformed the parameters to correspond to the original data space. We used a structure prior of the form κ^f, where f is the number of free parameters of the tree structure. For the three small datasets and for Census, we used $\kappa = 0.1$, and for Media Metrix, we used $\kappa = 0.01$; these values have proven to work well in previous (unrelated) experiments in these domains. We also used the non-Bayesian rule that a split is never applied if one of the resulting leaves has less than 10 relevant records.

Figure 2 shows the results of our experiments for all of the datasets. For the three small datasets, each point corresponds to a single learning instance. For the Census dataset results in Figure 2(d), each point is an average across the 31 trees for which at least one method resulted in a split on a continuous variable. Similarly, for the Media Metrix dataset results in Figure 2(e), each point is an average across the 16 trees for which at least one method resulted in a split on a continuous variable. In all five plots, the zero axis corresponds to a zero increase in predictive accuracy. In other words, this line corresponds to performance that is equiva-

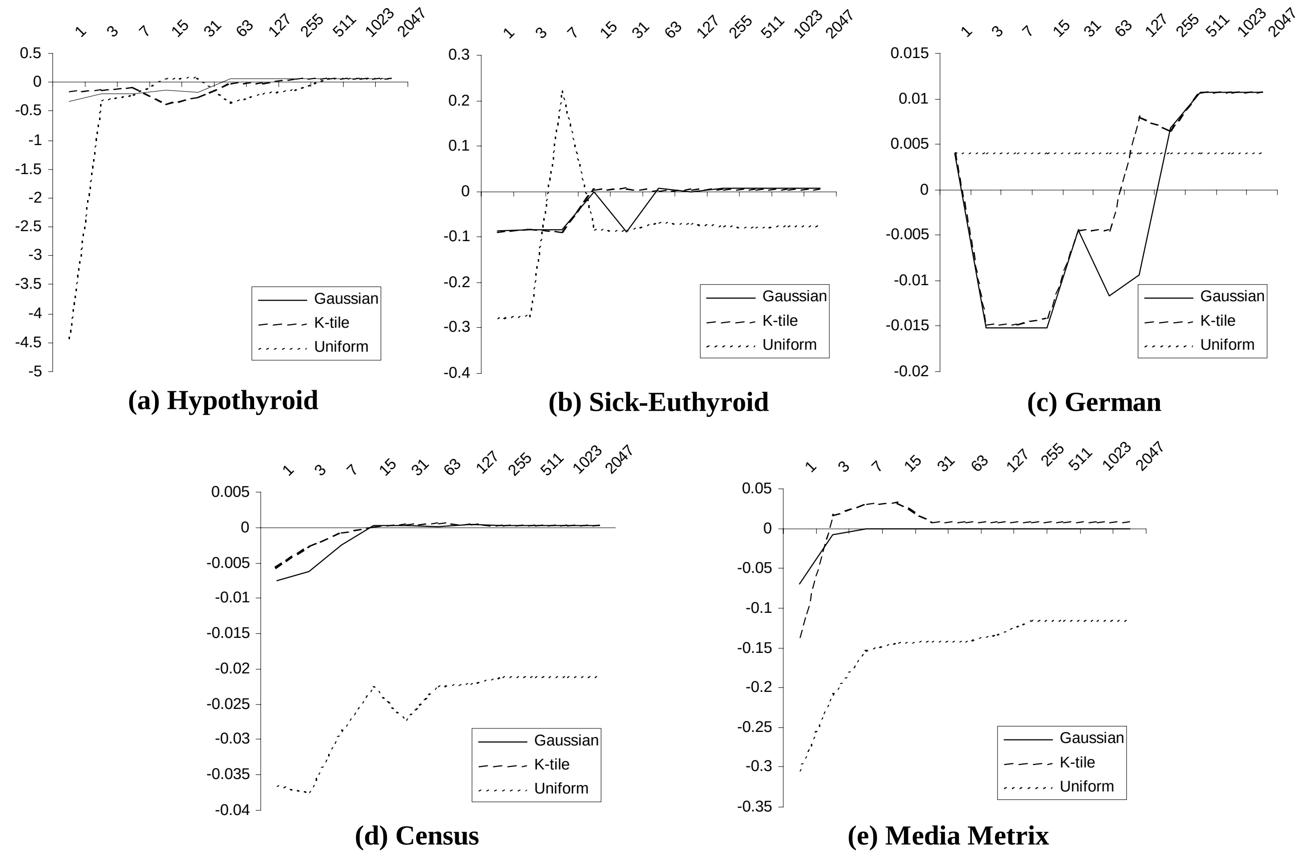

Figure 2. Relative increase in predictive accuracy (y-axis) for all of the data sets as a function of the number of split points k (x-axis) for the three proposed methods.

lent to the method of sorting and scoring all predictor values. As k grows large, all three of the methods should converge to something very close to this line because, at some point, every split point considered by the sorting method will be available for consideration by the other algorithms. It is possible, however, for the asymptotic performance of the Gaussian- and uniform-approximation methods to converge to some non-zero value; because the performance is measured on a test set, arbitrary choices of split points that have identical scoring criterion values for the training set may result in different prediction accuracy on the test set. As the size of the training data grows large, we expect such differences to be minimal. Note that the k-tile method, implemented using midpoints instead of endpoints as described, will be equivalent to the sorting method as soon as k is equal to one less than the number of cases for the given leaf.

For the small datasets Hypothyroid and Sick Euthyroid, the Gaussian-approximation and the k-tile method both worked as well as the full sort method using only a few split points. Although in the German data set the relative prediction accuracy appears not to be very monotonic in k, note that none of the approximations are ever less that a percent and a half worse than the full sort method. We believe the variance in the accuracies in the German domain are a result of the small sample size.

In none of the small datasets did the full sort method take longer than three seconds to construct the tree. Although the approximation techniques are faster, it does not make sense to adopt an approximation technique to speed up an algorithm that is already extremely fast.

We now turn our attention to the results for the Census dataset and the Media Metrix dataset. Both the Gaussian-approximation method and the k-tile method worked as well as the full-blown sorting method using only a few split points. In particular, we see that in both domains, these methods attain or surpass the sorting approach using only 15 split points.

In Figure 3 and Figure 4, we show the running times for all algorithms as a function of the number of split points, for Census and Media Metrix, respectively. Note that both axes are on a logarithmic scale in both figures. The time to learn in these domains grows roughly linearly in k (for $k > 511$), and the three approximation approaches all take about the same time. We expect that in domains with more continuous variables, the k-tile approach, although still linear in k, will prove to take longer in practice.

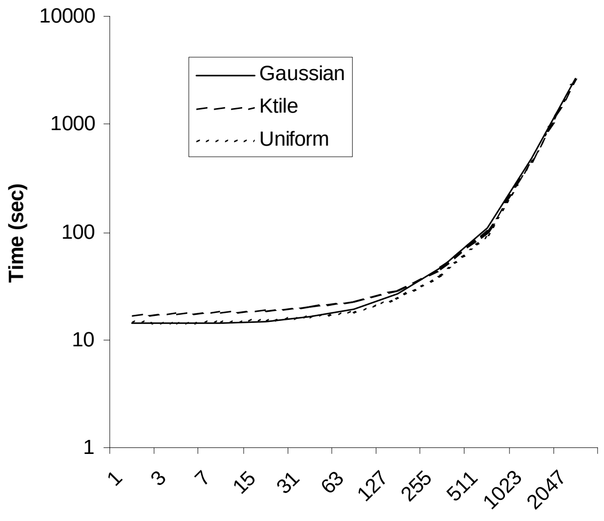

Figure 3. Time in seconds to learn all trees in the Census domain as a function of the number of split points k for the three proposed methods.

Although the k-tile method seems to be the best for Media Metrix and small k, the Gaussian approach works almost as well on the Media Metrix trees and just as well (for $k \geq 15$) on the census trees. The results also show that the uniform-approximation approach does not work well in these domains.

In Figure 5, we show the relative accuracy, using 15 split points, for each of the 31 trees in the Census domain that had at least one split using one of the three methods. Similarly, in Figure 6, we show the relative accuracy, using 15 split points, for each of the 16 trees in the Media Metrix domain that had at least one split using one of the three methods.

5 Conclusions and Future Work

We have presented three methods that dynamically determine split points for continuous predictor variables in a decision tree. We have given experimental evidence that the predictive accuracy of the trees that result from using two of them—namely, the Gaussian-approximation method and the k-tile method—are competitive with the standard approach of scoring all possible split points, while using only 15 split points.

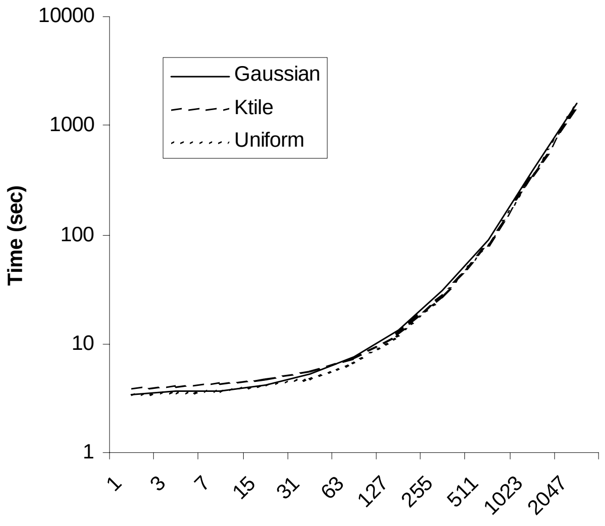

Figure 4. Time in seconds to learn all trees in the Media Metrix domain as a function of the number of split points k for the three proposed methods.

The predictive-accuracy plots that we presented in the previous section show the gain, as a function of the number of split points, that we get if we commit to the number of split points prior to running the learning algorithm. Alternatively, we could incorporate a model-selection step in the learning algorithm that decides (dynamically) how many split points to use at each split in the decision tree. We expect some modest gains in accuracy and potentially some gain in running time were we to implement such an approach.

As mentioned in Section 3, all three of our split-point selection methods can be viewed as identifying quantiles in predictor-variable distributions. It might be interesting to investigate and evaluate alternative, computationally tractable distributions. Additionally, we could try randomly sampling potential split points from predictor-variable distributions.

Finally, we point out that in many data sets, the majority of the time that the learning algorithm spends growing a decision tree is spent scoring candidate splits. If this is the case, then our methods can save time not only by spending less computation *identifying* split points to score, but by spending less time scoring splits by virtue of the fact that so few split points are needed for good accuracy.

Acknowledgments

The Media Metrix dataset for this paper was generously provided by Media Metrix. We thank the anonymous re-

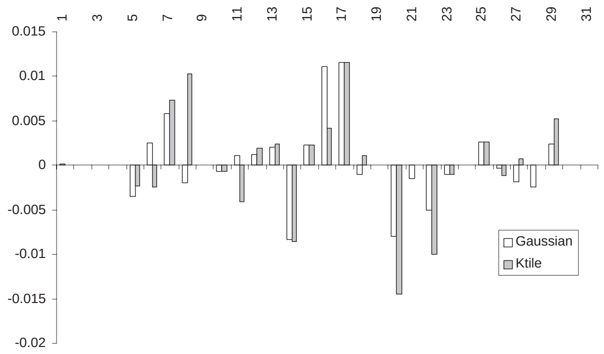

Figure 5. Relative increase in predictive accuracy, corresponding to a fixed $k = 15$, for the 31 trees in the Census data for which at least one methods resulted with a split on a continuous variable.

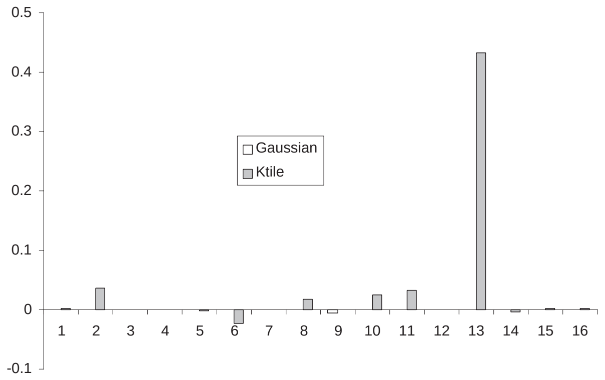

Figure 6. Relative increase in predictive accuracy, corresponding to a fixed $k = 15$, for the 16 trees in the Media Metrix data for which at least one methods resulted with a split on a continuous variable.

viewers for useful comments.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Monterey, CA, 1984.

[2] D. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence*, Providence, RI, pages 80–89. Morgan Kaufmann, August 1997.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT press, 1990.

[4] R. G. Cowell. On searching for optimal classifiers among Bayesian networks. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, pages 175–180, January 2001.

[5] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202. Morgan Kaufman, 1995.

[6] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1993.

[7] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, October 2000.

[8] E. B. Hunt, J. Marin, and P. T. Stone. *Experiments in Induction*. Academic Press, New York, NY, 1966.

[9] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology*, pages 18–32, March 1995.

[10] J. Quinlan. Programs for machine learning, 1993.

[11] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.