PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning

Rajeev Rastogi Kyuseok Shim

Bell Laboratories
Murray Hill, NJ 07974
rastogi@bell-labs.com shim@bell-labs.com

Abstract

Classification is an important problem in data mining. Given a database of records, each with a class label, a classifier generates a concise and meaningful description for each class that can be used to classify subsequent records. A number of popular classifiers construct decision trees to generate class models. These classifiers first build a decision tree and then prune subtrees from the decision tree in a subsequent *pruning* phase to improve accuracy and prevent "overfitting".

In this paper, we propose PUBLIC, an improved decision tree classifier that integrates the second "pruning" phase with the initial "building" phase. In PUBLIC, a node is not expanded during the building phase, if it is determined that it will be pruned during the subsequent pruning phase. In order to make this determination for a node, before it is expanded, PUBLIC computes a lower bound on the minimum cost subtree rooted at the node. This estimate is then used by PUBLIC to identify the nodes that are certain to be pruned, and for such nodes, not expend effort on splitting them. Experimental results with real-life as well as synthetic data sets demonstrate the effectiveness of PUBLIC's integrated approach which has the ability to deliver substantial performance improvements.

**Proceedings of the 24th VLDB Conference
New York, USA, 1998**

1 Introduction

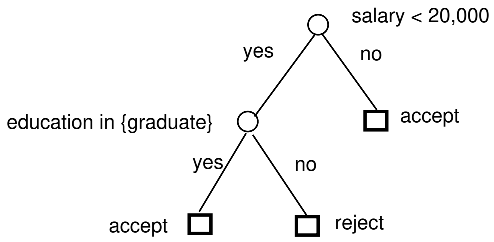
Classification is an important problem in data mining. It has been studied extensively by the machine learning community as a possible solution to the *knowledge acquisition* or *knowledge extraction* problem. The input to a classifier is a *training set* of records, each of which is tagged with a class label. A set of attribute values defines each record. Attributes with discrete domains are referred to as *categorical*, while those with ordered domains are referred to as *numeric*. The goal is to induce a model or description for each class in terms of the attributes. The model is then used to classify future records whose classes are unknown.

Figure 1(a) shows an example training set for a loan approval application. There is a single record corresponding to each loan request, each of which is tagged with one of two labels – accept if the loan request is approved or reject if the loan request is denied. Each record is characterized by two attributes, *salary* and *education*, the former numeric and the latter categorical with domain {high-school, undergraduate, graduate}. The attributes denote the income and the education level of the loan applicant. The goal of the classifier is to deduce, from the training data, concise and meaningful conditions involving *salary* and *education* under which a loan request is accepted or rejected.

Classification has been successfully applied to several areas like medical diagnosis, weather prediction, credit approval, customer segmentation and fraud detection. Among the techniques developed for classification, popular ones include bayesian classification[CKS+88], neural networks[Rip96], genetic algorithms[Gol89] and decision trees[BFOS84]. In this paper, however, we focus on decision trees. There are several reasons for this. First, compared to a neural network or a bayesian classifier, a decision tree is easily interpreted/comprehended by

salary	*education*	label
10,000	high-school	reject
40,000	under-graduate	accept
15,000	under-graduate	reject
75,000	graduate	accept
18,000	graduate	accept

(a)



(b)

Figure 1: Decision Trees

humans[BFOS84]. Second, while training neural networks can take large amounts of time and thousands of iterations, inducing decision trees is efficient and is thus suitable for large training sets. Also, decision tree generation algorithms do not require additional information besides that already contained in the training data (e.g., domain knowledge or prior knowledge of distributions on the data or classes)[Fay91]. Finally, as shown in [MST94], decision trees display good classification accuracy compared to other techniques.

Figure 1(b) is a decision tree for the training data in Figure 1(a). Each internal node of the decision tree has a test involving an attribute, and an outgoing branch for each possible outcome. Each leaf has an associated class. In order to classify new records using a decision tree, beginning with the root node, successive internal nodes are visited until a leaf is reached. At each internal node, the test for the node is applied to the record. The outcome of the test at an internal node determines the branch traversed, and the next node visited. The class for the record is simply the class of the final leaf node. Thus, the conjunction of all the conditions for the branches from the root to a leaf constitute one of the conditions for the class associated with the leaf. For instance, the decision tree in Figure 1(b) approves a loan request only if *salary* $\geq$ 20,000 or *education* $\in$ {graduate}; otherwise, it rejects the loan application.

A number of algorithms for inducing decision trees have been proposed over the years (e.g., CLS [HMS66], ID3 [Qui86], C4.5 [Qui93], CART [BFOS84], SLIQ [MAR96], SPRINT [SAM96]). Most of the algorithms have two distinct phases, a *building* or *growing* phase followed by a *pruning* phase. In the building phase, the training data set is recursively partitioned until all the records in a partition have the same class. For every partition, a new node is added to the decision tree; initially, the tree has a single root node for the entire data set. For a set of records in a partition P, a test criterion T for further partitioning the set into $P_1, \ldots, P_m$ is first determined. New nodes for $P_1, \ldots, P_m$ are created and these are added to the decision tree as children of the node for P. Also, the node for P is labeled with test T, and partitions $P_1, \ldots, P_m$ are then recursively partitioned. A partition in which all the records have identical class labels is not partitioned further, and the leaf corresponding to it is labeled with the class.

The building phase constructs a perfect tree that accurately classifies every record from the training set. However, one often achieves greater accuracy in the classification of new objects by using an imperfect, smaller decision tree rather than one which perfectly classifies all known records [QR89]. The reason is that a decision tree which is perfect for the known records may be overly sensitive to statistical irregularities and idiosyncrasies of the training set. Thus, most algorithms perform a pruning phase after the building phase in which nodes are iteratively pruned to prevent "overfitting" and to obtain a tree with higher accuracy.

An important class of pruning algorithms are those based on the Minimum Description Length (MDL) principle [QR89, FI93, MRA95]. Consider the problem of communicating the classes for a set of records. Since a decision tree partitions the records with a goal of separating those with similar class labels, it can serve as an efficient means for encoding the classes of records. Thus, the "best" decision tree can then be considered to be the one that can communicate the classes of the records with the "fewest" number of bits. The cost (in bits) of communicating classes using a decision tree comprises of (1) the bits to encode the structure of the tree itself, and (2) the number of bits needed to encode the classes of records in each leaf of the tree. We thus need to find the tree for which the above cost is minimized. This can be achieved as follows. A subtree S is pruned if the cost of directly encoding the records in S is no more than the cost of encoding the subtree plus the cost of the records in each leaf of the subtree. In [MRA95], it is shown that MDL pruning (1) leads to accurate trees for a wide range of data sets, (2) produces trees that are significantly smaller in size, and (3) is computationally efficient and does not use a separate data set for pruning. For the above reasons, the pruning algorithms developed in this paper employ MDL pruning.

Generating the decision tree in two distinct phases could result in a substantial amount of wasted effort since an entire subtree constructed in the first phase may later be pruned in the next phase. During the building phase, before splitting a node, if it can be concluded that the node will be pruned from the tree during the subsequent pruning phase, then we could avoid building the subtree rooted at the node. Consequently, since building a subtree usually requires repeated scans to be performed over the data, significant reductions in I/O and improvements in performance can be realized. In this paper, we present PUBLIC (PrUning and BuiLding Integrated in Classification), a decision tree classifier that during the growing phase, first determines if a node will be pruned during the following pruning phase, and subsequently stops expanding such nodes. Thus, PUBLIC integrates the pruning phase into the building phase instead of performing them one after the other. Furthermore, by only pruning nodes that we know will definitely be pruned in the pruning phase, we guarantee that the tree generated by PUBLIC's integrated approach is exactly the same as the tree that would be generated as a result of executing the two phases separately, one after another.

Determining, during the building phase, whether a node will be pruned during the pruning phase is problematic since the tree is only partially generated. Specifically, this requires us to estimate at each leaf of the partial tree, based on the records contained in the leaf, a lower bound on the cost of the subtree rooted at the leaf. Furthermore, the better (higher) this estimate, the more we can prune during the building phase and consequently, the more we can improve performance. We present several algorithms for estimating the subtree cost - the algorithms illustrate the trade-off between accuracy of the estimate and the computation involved. Our experimental results on real-life as well as synthetic data sets demonstrate that PUBLIC's integrated approach can result in substantial performance improvements compared to traditional classifiers.

The remainder of the paper is organized as follows. In Section 2, we survey existing work on decision tree classifiers. Details of the building and pruning phases of a traditional decision tree classifier along the lines of SPRINT [SAM96] are presented in Section 3. The PUBLIC algorithm as well as techniques for estimating lower bounds on subtree costs are described in sections 4 and 5. In Section 6, we compare PUBLIC's performance with that of a traditional decision tree classifier. Finally, in Section 7, we offer concluding remarks.

2 Related Work

In this section, we provide a brief survey of related work on decision tree classifiers. The growing phase for the various decision tree generation systems differ in the algorithm employed for selecting the test criterion T for partitioning a set of records. CLS [HMS66], one of the earliest systems, examines the solution space of all possible decision trees to some fixed depth. It then chooses a test that minimizes the cost of classifying a record. The cost is made up of the cost of determining the feature values for testing as well as the cost of misclassification. ID3 [Qui86] and C4.5 [Qui93] replace the computationally expensive look-ahead scheme of CLS with a simple information theory driven scheme that selects a test that minimizes the *information entropy* of the partitions (we discuss entropy further in Section 3), while CART [BFOS84], SLIQ [MAR96] and SPRINT [SAM96] select the test with the lowest GINI index. Classifiers like C4.5 and CART assume that the training data fits in memory. SLIQ and SPRINT, however, can handle large training sets with several million records. SLIQ and SPRINT achieve this by maintaining separate lists for each attribute and pre-sorting the lists for numeric attributes. We present a detailed description of SPRINT, a state of the art classifier for large databases, in Section 3.

In addition to MDL pruning described earlier, there are two other broad classes of pruning algorithms. The first includes algorithms like cost-complexity pruning [Qui87] that first generate a sequence of trees obtained by successively pruning non-leaf subtrees for whom the ratio of the reduction in misclassified objects due to the subtree and the number of leaves in the subtree is minimum. A second phase is then carried out in which separate pruning data (distinct from the training data used to grow the tree) is used to select the tree with the minimum error. In the absence of separate pruning data, cross-validation can be used at the expense of a substantial increase in computation. The second class of pruning algorithms, pessimistic pruning [Qui87], do not require separate pruning data, and are computationally inexpensive. Experiments have shown that this pruning leads to trees that are "too" large with high error rates.

The above-mentioned decision tree classifiers only consider "guillotine-cut" type tests for numeric attributes. Since these may result in very large decision trees when attributes are correlated, in [FMM96], the authors propose schemes that employ tests involving two (instead of one) numeric attributes and consider partitions corresponding to grid regions in the two-dimensional space. Recently, in [GRG98], the authors propose Rainforest,

```
procedure buildTree(S):
1. Initialize root node using data set S
2. Initialize queue Q to contain root node
3. while Q is not empty do {
4.    dequeue the first node N in Q
5.    if N is not pure {
6.       for each attribute A
7.          Evaluate splits on attribute A
8.       Use best split to split node N into N1 and N2
9.       Append N1 and N2 to Q
10.   }
11. }
```

Figure 2: Building Algorithm

a framework for developing fast and scalable algorithms for constructing decision trees that gracefully adapt to the amount of main memory available. In [FI93], the authors use the entropy minimization heuristic and MDL principle for discretizing the range of a continuous-valued attribute into multiple intervals.

Note that PUBLIC's integrated approach is different from that in [AGI$^+$92] where a *dynamic pruning* criterion based on pessimistic pruning is used to stop expanding nodes during the growing phase. The dynamic pruning scheme proposed in [AGI$^+$92] is ad-hoc and it does not guarantee that the resulting tree is the same as the tree that would be obtained as a result of performing the pruning phase after the completion of the building phase. This is a major drawback and could adversely impact the accuracy of the tree. For instance, in [AGI$^+$92], if successive expansions of a node and its children do not result in acceptable error reduction, then further expansions of its children are terminated.

3 Preliminaries

In this section, we present a more detailed description of the building and pruning phases of a decision tree classifier. The tree building phase is based on SPRINT [SAM96], while the MDL pruning algorithm employed for pruning the tree is along the lines described in [QR89, MRA95].

3.1 Tree building Phase

The overall algorithm for building a decision tree is as shown in Figure 2. The tree is built breadth-first by recursively partitioning the data until each partition is *pure*, that is, each partition contains records belonging to the same class. The splitting condition for partitioning the data is either of the form $A < v$ if A is a numeric attribute (v is a value in the domain of A) or $A \in V$ if A is a categorical attribute (V is a set of values from A's domain). Thus, each split is binary.

Data Structures. Each node of the decision tree maintains a separate list for every attribute. Each attribute list contains a single entry for every record in the partition for the node. The attribute list entry for a record contains three fields – the value for the attribute in the record, the class label for the record and the record identifier. Attribute lists for the root node are constructed at the start using the input data, while for other nodes, they are derived from their parent's attribute lists when the parent nodes are split. Attribute lists for numeric attributes at the root node are sorted initially and this sort order is preserved for other nodes by the splitting procedure. Also, at each node, a histogram is maintained that captures the class distribution of the records at the node. Thus, the initialization of the root node in Step 1 of the build algorithm involves (1) constructing the attribute lists, (2) sorting the attribute lists for numeric attributes, and (3) constructing the histogram for the class distribution.

Selecting Splitting Attribute. For a set of records S, the entropy $E(S) = -\sum_j p_j \log p_j$, where p_j is the relative frequency of class j in S. Thus, the more homogeneous a set is with respect to the classes of records in the set, the lower is its entropy. The entropy of a split that divides S with n records into sets S_1 with n_1 records and S_2 with n_2 records is $E(S_1, S_2) = \frac{n_1}{n}E(S_1) + \frac{n_2}{n}E(S_2)$. Consequently, the split with the least entropy best separates classes, and is thus chosen as the best split for a node[1].

To compute the best split point for a numeric attribute, the (sorted) attribute list is scanned from the beginning and for each split point, the class distribution in the two partitions is determined using the class histogram for the node. The entropy for each split point can thus be efficiently computed since the lists are stored in a sorted order. For categorical attributes, the attribute list is scanned to first construct a histogram containing the class distribution for each value of the attribute. This histogram is then utilized to compute the entropy for each split point.

Splitting Attribute Lists. Once the best split for a node has been found, it is used to split the attribute list for the splitting attribute amongst the two child nodes. Each record identifier along with information about the child node that it is assigned to (left or right) is then inserted into a hash table. The remaining attribute lists are then split using the record identifier stored with each attribute list

[1] In SPRINT, the GINI index is used instead of entropy to compute the best split.

entry and the information in the hash table. Class distribution histograms for the two child nodes are also computed during this step.

3.2 Tree Pruning Phase

To prevent overfitting, the MDL principle [Ris78, Ris89] is applied to prune the tree built in the growing phase and make it more general. The MDL principle states that the "best" tree is the one that can be encoded using the fewest number of bits. Thus, the challenge for the pruning phase is to find the subtree of the tree that can be encoded with the least number of bits.

In the following, we first present a scheme for encoding decision trees. We then present a pruning algorithm that, in the context of our encoding scheme, finds the minimum cost subtree of the tree constructed in the growing phase. In the remainder of the paper, we assume that a is the number of attributes.

Cost of Encoding Data Records. Let a set S contain n records each belonging to one of k classes, n_i being the number of records with class i. The cost of encoding the classes for the n records [QR89] is given by[2]

$$\log \binom{n+k-1}{k-1} + \log \frac{n!}{n_1! \cdots n_k!}$$

In the above equation, the first term is the number of bits to specify the class distribution, that is, the number of records with classes $1, \ldots, k$. The second term is the number of bits required to encode the class for each record once it is known that there are n_i records with class label i. In [MRA95], it is pointed out that the above equation is not very accurate when some of the n_i are either close to zero or close to n. Instead, they suggest using the following equation from [KT81], which is what we adopt in this paper for the cost $C(S)$ of encoding the classes for the records in set S.

$$C(S) = \sum_i n_i \log \frac{n}{n_i} + \frac{k-1}{2} \log \frac{n}{2} + \log \frac{\pi^{k/2}}{\Gamma(k/2)} \quad (1)$$

In Equation (1), the first term is simply $n * E(S)$, where $E(S)$ is the entropy of the set S of records. Also, since $k \leq n$, the sum of the last two terms in Equation (1) is always non-negative. We utilize this property later in the paper when computing a lower bound on the cost of encoding the records in a leaf.

In SPRINT, the cost of encoding a set of data records is assumed to be simply the number of records that do not belong to the majority class for the set. However, our experience with most real-life data sets has been that using Equation (1) instead results in more accurate trees. In PUBLIC, even though we use Equation (1) as the cost for encoding a set of records, PUBLIC's pruning techniques are also applicable if we were to use the approach adopted in SPRINT.

Cost of Encoding Tree. The cost of encoding the tree comprises of three separate costs:

1. The cost of encoding the structure of the tree.
2. The cost of encoding for each split, the attribute and the value for the split.
3. The cost of encoding the classes of data records in each leaf of the tree.

The structure of the tree can be encoded by using a single bit in order to specify whether a node of the tree is an internal node (1) or leaf (0). Thus, the bit string 11000 encodes the tree in Figure 1(b). Since we are considering only binary decision trees, the proposed encoding technique for representing trees is nearly optimal [QR89].

The cost of encoding each split involves specifying the attribute that is used to split the node and the value for the attribute. The splitting attribute can be encoded using $\log a$ bits (since there are a attributes), while specifying the value depends on whether the attribute is categorical or numeric. Let v be the number of distinct values for the splitting attribute in records at the node. If the splitting attribute is numeric, then since there are $v-1$ different points at which the node can be split, $\log(v-1)$ bits are needed to encode the split point. On the other hand, for a categorical attribute, there are 2^v different subsets of values of which the empty set and the set containing all the values are not candidates for splitting. Thus, the cost of the split is $\log(2^v - 2)$. For an internal node N, we denote the cost of describing the split by $C_{split}(N)$.

Finally, the cost of encoding the data records in each leaf is as described in Equation (1).

Pruning Algorithm. Now that we have a formulation for the cost of a tree, we next turn our attention to computing the minimum cost subtree of the tree constructed in the building phase. The simple recursive algorithm in Figure 3 computes the minimum cost subtree rooted at an arbitrary node N and returns its cost. Let S be the set of records associated with N. If N is a leaf, then the minimum cost subtree rooted at N is simply N itself. Furthermore, the cost of the cheapest subtree rooted at N is $C(S)+1$ (we require 1 bit in order to specify that the node is a leaf).

[2] All logarithms in the paper are to the base 2.

procedure computeCost&Prune(Node N):
    /* S is the set of data records for N */
1. **if** N is a leaf **return** $(C(S)+1)$
    /* N_1 and N_2 are N's children */
2. $\text{minCost}_1 := \text{computeCost\&Prune}(N_1)$;
3. $\text{minCost}_2 := \text{computeCost\&Prune}(N_2)$;
4. $\text{minCost}_N := \min\{C(S)+1,$
        $C_{split}(N) + 1 + \text{minCost}_1 + \text{minCost}_2\}$;
5. **if** $\text{minCost}_N = C(S) + 1$
6.   prune child nodes N_1 and N_2 from tree
7. **return** minCost_N

Figure 3: Pruning Algorithm

On the other hand, if N is an internal node in the tree with children N_1 and N_2, then there are the following two choices for the minimum cost subtree – (1) the node N itself with no children (this corresponds to pruning its two children from the tree, thus making node N a leaf), or (2) node N along with children N_1 and N_2 and the minimum cost subtrees rooted at N_1 and N_2. Of the two choices, the one with the lower cost results in the minimum cost subtree for N.

The cost for choice (1) is $C(S) + 1$. In order to compute the cost for choice (2), in Steps 2 and 3, the procedure recursively invokes itself in order to compute the minimum cost subtrees for its two children. The cost for choice (2) is then $C_{split}(N) + 1 + \text{minCost}_1 + \text{minCost}_2$. Thus, the cost of the cheapest subtree rooted at N is given by minCost_N as computed in Step 4. Note that if choice (1) has a smaller cost, then the children of node N must be pruned from the tree. Or stated alternately, children of a node N are pruned if the cost of directly encoding the data records at N does not exceed the cost of encoding the minimum cost subtree rooted at N.

The tree built in the "growing" phase is pruned by invoking the pruning algorithm in Figure 3 on the root node.

4 The PUBLIC Integrated Algorithm

Most algorithms for inducing decision trees perform the pruning phase only after the entire tree has been generated in the initial building phase. Our typical experience on real-life data sets has been that the pruning phase prunes large portions of the original tree – in some cases, this can be as high as 90% of the nodes in the tree (see Section 6). These smaller trees are more general and result in smaller classification error for records whose classes are unknown [QR89, Fay91].

It is clear that in most decision tree algorithms, a substantial effort is "wasted" in the building phase on growing portions of the tree that are subsequently pruned in the pruning phase. Consequently, if during the building phase, it were possible to "know" that a certain node is definitely going to be pruned, then we can stop expanding the node further, and thus avoid the computational and I/O overhead involved in processing the node. As a result, by incorporating the pruning "knowledge" into the building phase, it is possible to realize significant improvements in performance. This is the approach followed by the PUBLIC classification algorithm that combines the pruning phase with the building phase.

The PUBLIC algorithm is similar to the build procedure shown in Figure 2. The only difference is that periodically or after a certain number of nodes are split (this is a user-defined parameter), the partially built tree is pruned. The pruning algorithm in Figure 3, however, cannot be used to prune the partial tree.

The problem with applying the pruning procedure in Figure 3 before the tree has been completed is that in the procedure, the cost of the cheapest subtree rooted at a leaf N assumed to be $C(S) + 1$. While this is true for a tree that has been completely built, it is not true for a partially built tree since a leaf in a partial tree may be split later, thus becoming an internal node. Consequently, the cost of the subtree rooted at N could be a lot less than $C(S)+1$ as a result of the splitting. Thus, $C(S) + 1$ may over-estimate the cost of the cheapest subtree rooted at N and this could resulting in over-pruning, that is, nodes may be pruned during the building phase that would not have been pruned during the pruning phase. This is undesirable since we would like the decision tree induced by PUBLIC to be identical to the one constructed by a traditional classifier.

In order to remedy the above problem, we make use of the following observation – running the pruning algorithm described in Figure 3, while under-estimating the minimum cost of subtrees rooted at leaf nodes that can still be expanded, is not harmful. With an under-estimate of the minimum subtree cost at nodes, the nodes pruned are a subset of those that would have been pruned anyway during the pruning phase. PUBLIC's pruning algorithm, illustrated in Figure 4, is based on this under-estimation strategy for the cost of the cheapest subtree rooted at a "yet to be expanded" leaf node.

PUBLIC's pruning distinguishes among three kinds of leaf nodes. The first kind of leaves are those that still need to be expanded. For such leaves, as described in the following section, PUBLIC computes a lower bound on the cost of subtrees at the leaves. The two other kinds of leaf nodes consist of those that are either a result of pruning or those that cannot be expanded any further (because they

procedure computeCost&PrunePublic(Node N):
    /* S is the set of data records for N */
1. **if** N is a "yet to be expanded" leaf
        return lower bound on subtree cost at N
2. **if** N is a "pruned" or "not expandable" leaf
        return $(C(S) + 1)$
    /* N_1 and N_2 are N's children */
3. $\text{minCost}_1 := \text{computeCost\&PrunePublic}(N_1)$;
4. $\text{minCost}_2 := \text{computeCost\&PrunePublic}(N_2)$;
5. $\text{minCost}_N := \min\{C(S) + 1,$
        $C_{split}(N) + 1 + \text{minCost}_1 + \text{minCost}_2\}$;
6. **if** $\text{minCost}_N = C(S) + 1$ {
7.   Prune child nodes N_1 and N_2 from tree
8.   Delete nodes N_1 and N_2 and all their descendants from Q
9.   Mark node N as pruned
10. }
11. **return** minCost_N

Figure 4: PUBLIC's Pruning Algorithm

are pure). For such leaves, we use the usual cost of $C(S) + 1$. Thus, the pruning procedure is similar to the earlier procedure in Figure 3 except that the cost for the cheapest subtree at leaf nodes that have not yet been expanded may not be $C(S) + 1$. Also, when children of a node N are pruned, all its descendants are removed from the queue Q maintained in the build procedure – this ensures that they are not expanded by the build procedure.

In PUBLIC, the modified pruning algorithm shown in Figure 4 is invoked from the build procedure periodically on the root of the partially built tree. Note that once the building phase ends, there are no leaf nodes belonging to the "yet to be expanded" category. As a result, applying the pruning algorithm in Figure 4 at the end of the building phase has the same effect as applying the original pruning algorithm and results in the same pruned tree as would have resulted due to the previous pruning algorithm.

5 Computation of Lower Bound on Subtree Cost

Any subtree rooted at a node N must have a cost of at least 1, and thus 1 is a simple, but conservative estimate for the cost of the cheapest subtree at leaf nodes that are "yet to be expanded". In our experiments, we found that even this simple estimate enables PUBLIC to substantially reduce the number of nodes generated. Since more accurate estimates can enable PUBLIC to prune even more nodes, in this section, we propose two algorithms for computing better and higher estimates for the minimum cost subtrees at leaf nodes. The first considers split costs and the second incorporates even the cost of describing the value for each split in the computed estimates. An important point to note is that the computed estimates represent more accurate *lower bounds* on the cost of the cheapest subtree at a leaf node. As mentioned before, it is essential that we underestimate the costs at leaf nodes to prevent over-pruning.

We refer to the version of the PUBLIC algorithm based on a cost estimate of 1 as PUBLIC(1). The other two versions (presented in the following two subsections) that incorporate the cost of splits and cost of values into the estimates are referred to as PUBLIC(S) and PUBLIC(V), respectively. Note that PUBLIC(1), PUBLIC(S) and PUBLIC(V) are identical except for the value returned in Step 1 of the pruning algorithm in Figure 4. Thus, PUBLIC(1), PUBLIC(S) and PUBLIC(V) use increasingly accurate cost estimates for "yet to be expanded" leaf nodes, and result in fewer nodes being expanded during the building phase.

We must point out that our experimental results in Section 6 indicate that a significant portion of the performance benefits due to the integrated pruning can be realized by simply using PUBLIC(1). In our experiments, we found that fewer additional nodes are pruned during building due to the "better" cost estimates employed by PUBLIC(S) and PUBLIC(V). The reason for this is that the cost of encoding splits turns out be a fairly substantial part of the cost of encoding the tree. Recall from Section 3, that encoding a split involving a categorical attribute requires $\log a + \log(2^v - 2)$ bits, while for a split involving a numeric attribute, the cost is $\log a + \log(v - 1)$. During pruning, the cost for the subtree rooted at an internal node N includes the cost of encoding splits in the subtree, which is significant. In addition, for leaves in the subtree that have been pruned or cannot be expanded further, an accurate cost estimate of $C(S) + 1$ is used. As a result, even with a simple cost estimate of 1 for "yet to be expanded" leaves, due to split costs and accurate costs for other kinds of leaves, reasonable cost estimates for subtree costs at an internal node N can be computed, thus enabling a large number of nodes to be pruned by PUBLIC(1).

Furthermore, generating highly accurate cost estimates for "yet to be expanded" leaf nodes is a difficult problem. As a result, even though the estimates used by PUBLIC(S) and PUBLIC(V) are increasingly better, they are not accurate enough to deliver big gains in the number of pruned nodes compared to PUBLIC(1). Specifically, the contribution of the estimates to the subtree cost at an internal node N is still small compared to split costs and cost of pruned and unexpandable nodes in the subtree.

5.1 Estimating Split Costs

The PUBLIC(S) algorithm takes into account split costs when computing a lower bound on the cost of the cheapest subtree rooted at a "yet to be expanded" leaf node N. Specifically, for values of $s \geq 0$, it computes a lower bound on the cost of subtrees rooted at N and containing s splits (and consequently, s internal nodes). The cost estimate for the cheapest subtree at node N is then set to the minimum of the lower bounds computed for the different s values – this guarantees that PUBLIC(S) underestimates the cost of the cheapest subtree rooted at N.

Let S be the set of records at node N and k be the number of classes for the records in S. Also, let n_i be the number of records belonging to class i in S, and $n_i \geq n_{i+1}$ for $1 \leq i < k$ (that is, $n_1, \ldots, n_k$ are sorted in the decreasing order of their values). As before, a denotes the number of attributes. In case node N is not split, that is, $s = 0$, then the minimum cost for a subtree at N is $C(S) + 1$. For values of $s > 0$, a lower bound on the cost of encoding a subtree with s splits and rooted at node N is derived in the following theorem.

Theorem 5.1: *The cost of any subtree with s splits and rooted at node N is at least* $2 * s + 1 + s * \log a + \sum_{i=s+2}^{k} n_i$. ∎

Proof: The cost of encoding the structure of a subtree with s splits is $2 * s + 1$ since a subtree with s splits has s internal nodes and $s + 1$ leaves, and we require one bit to specify the type for each node. Each split also has a cost of at least $\log a$ to specify the splitting attribute. The final term is the cost of encoding the data records in the $s + 1$ leaves of the subtree.

Let n_{ij} denote the number of records belonging to class i in leaf j of the subtree. A class i is referred to as a *majority* class in leaf j if $n_{ij} \geq n_{lj}$ for every other class l in leaf j (in case for two classes i and l, $n_{ij} = n_{lj}$, then one of them is arbitrarily chosen as the majority class). Thus, each leaf has a single majority class, and every other class in the leaf that is not a majority class is referred to as a *minority* class. Since there are $s + 1$ leaves, there can be at most $s + 1$ *majority* classes, and at least $k - s - 1$ classes are a minority class in every leaf.

Consider a class i that is a minority class in leaf j. Due to Equation (1), $C(S_j)$, the cost of encoding the classes of records in the leaf is at least $\sum_i n_{ij} * E(S_j)$ where S_j is the set of records in leaf j and $\sum_i n_{ij}$ is the total number of records in leaf j. Since for class i, $E(S_j)$ contains the term $\frac{n_{ij}}{\sum_i n_{ij}} \log \frac{\sum_i n_{ij}}{n_{ij}}$, the records of class i in leaf j contribute at least $(\sum_i n_{ij}) * (\frac{n_{ij}}{\sum_i n_{ij}} \log \frac{\sum_i n_{ij}}{n_{ij}})$ to $C(S_j)$. Furthermore, since class i is a minority in leaf j, we have $\frac{\sum_i n_{ij}}{n_{ij}} \geq 2$ and so the records with class i in leaf j contribute at least n_{ij} to $C(S_j)$. Thus, if L is the set containing the $k-s-1$ classes that are a minority in every leaf, then the minority classes i in L across all the leaves contribute $\sum_{i \in L} n_i$ to the cost of encoding the data records in the leaves of the subtree.

Since we are interested in a lower bound on the cost of the subtree, we need to consider the set L containing $k - s - 1$ classes for which $\sum_{i \in L} n_i$ is minimum. Obviously, the above cost is minimum for the $k - s - 1$ classes with the smallest number of records in S, that is, classes $s + 2, \ldots, k$. Thus, the cost for encoding the records in the $s + 1$ leaves of the subtree is at least $\sum_{i=s+2}^{k} n_i$. ∎

procedure computeMinCostS(Node N):
/* $n_1, \ldots, n_k$ are sorted in decreasing order */
if $k = 1$ **return** $(C(S) + 1)$
s := 1
tmpCost := $2 * s + 1 + s * \log a + \sum_{i=s+2}^{k} n_i$
while $s + 1 < k$ **and** $n_{s+2} > 2 + \log a$ **do** {
  tmpCost := tmpCost + $2 + \log a - n_{s+2}$
  $s{+}{+}$
}
return $\min\{C(S) + 1, \text{tmpCost}\}$

Figure 5: Algorithm for Computing Lower Bound on Subtree Cost

Theorem 5.1 gives a lower bound on the cost of any subtree with s splits and can be used to estimate the cost for the minimum cost subtree rooted at a node N. Thus, we simply need to compute, using the result of Theorem 5.1, the minimum cost for subtrees rooted at N with $0, \ldots, k-1$ splits and set our cost estimate to be the minimum of all these costs. The reason for only considering upto $k - 1$ splits is that beyond $k - 1$ splits, the subtree cost does not reduce any further. This is because the last term in Theorem 5.1 (which is the sum of the number of records of the $k - s - 1$ smallest classes) becomes 0 for $k - 1$ splits and cannot decrease any further, while the other terms keep increasing with the number of splits.

In addition, if for a certain number s of splits, it is the case that $n_{s+2} \leq 2 + \log a$, then we do not need to consider subtrees with splits greater than s. The reason for this is that when s increases and becomes $s + 1$, the minimum subtree cost increases by a fixed amount which is $2 + \log a$ while it decreases by n_{s+2}. Thus, since $n_i \geq n_{i+1}$, increasing s further cannot cause the minimum subtree cost to decrease any further.

The algorithm for computing the estimate for the minimum cost subtree at a "yet to be expanded" node N in PUBLIC(S) is as shown in Figure 5. In the procedure, the variable tmpCost stores the minimum cost subtree with $s \geq 1$ splits. For $s = 0$, since the bound due to Theorem 5.1 may be loose, the procedure uses $C(S) + 1$ instead. The maximum value considered for s is $k - 1$, and if for an s, $n_{s+2} \leq 2 + \log a$, then values larger than s are not considered. The time complexity of the procedure is dominated by the cost of sorting the n_i's in the decreasing order of their values, and is thus, $O(k \log k)$.

5.2 Incorporating Costs of Split Values

In the PUBLIC(S) algorithm described in the previous section, when estimating the cost of a subtree rooted at a "yet to be expanded" node N, we estimate the cost of each split to be $\log a$ bits. However, this only captures the cost of specifying the attribute involved in the split. In order to completely describe a split, a value or a set of values is also required for the splitting attribute - this is to specify the distribution of records amongst the children of the split node.

Due to space constraints and the fact that the additional performance gains due to PUBLIC(V) over PUBLIC(1) and PUBLIC(S) are fairly modest, we defer the description of the PUBLIC(V) algorithm to [RS98]. PUBLIC(V) estimates the cost of each split more accurately than PUBLIC(S) by also including the cost of encoding the split value in the cost of each split. Except for this, the PUBLIC(V) algorithm follows a similar strategy as PUBLIC(S) for estimating the cost of the cheapest subtree rooted at a "yet to be expanded" node N. For values of $s \geq 0$, PUBLIC(V) first computes a lower bound on the cost of subtrees containing s splits and rooted at N. The minimum of these lower bounds for the various values of s is then chosen as the cost estimate for the cheapest subtree at N. The time complexity of the cost estimation procedure for PUBLIC(V) is $O(k * (\log k + a))$. A detailed description of PUBLIC(V) algorithm can be found in [RS98].

6 Experimental Results

In order to investigate the performance gains that can be realized due to PUBLIC's integrated approach to classification, we conducted experiments on real-life as well as synthetic data sets. We used an implementation of SPRINT [SAM96] as described in Section 3 as representative of traditional classifiers that carry out building and pruning in separate phases. We are thus primarily interested in the speedup due to the integrated PUBLIC algorithms as measured against SPRINT.

Since real-life data sets are generally small, we also used synthetic data sets to study PUBLIC's performance on larger data sets. The purpose of the synthetic data sets is primarily to examine the PUBLIC's sensitivity to parameters such as noise, number of classes and number of attributes. Synthetic data sets allow us to vary the above parameters in a controlled fashion. Since PUBLIC is based on SPRINT except for the integration of the building and pruning phases, and SPRINT was shown to scale well for large databases in [SAM96], our goal is not to demonstrate the scalability of PUBLIC. Instead, as we mentioned before, we are more interested in measuring the improvements in execution time due to combining the building and pruning phases compared to performing the two phases separately.

All of our experiments were performed using a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5. Our experimental results with both real-life as well as synthetic data sets demonstrate the effectiveness of PUBLIC's integrated algorithm compared to traditional classification algorithms.

6.1 Algorithms

In our experiments, we compared the execution times for four algorithms, whose characteristics we summarize below.

- **SPRINT:** This is the algorithm that we use as representative of traditional algorithms with separate building and pruning phases. We use the variant of SPRINT described in Section 3. Note that, unlike [SAM96], our variant of SPRINT uses entropy instead of the GINI index.

- **PUBLIC(1):** This is the simplest of the PUBLIC algorithms. It performs building and pruning together, and uses the very conservative estimate of 1 as the cost of the cheapest subtree rooted at a "yet to be expanded" leaf node.

- **PUBLIC(S):** Unlike PUBLIC(1), for the minimum cost subtree at a "yet to be expanded" leaf node, PUBLIC(S) considers subtrees with splits and includes the cost of specifying the splitting attribute for splits.

- **PUBLIC(V):** Among the PUBLIC algorithms, PUBLIC(V) computes the most accurate lower bound on the cost for a subtree at a "yet to be expanded" leaf node. In addition to the cost of specifying the splitting attribute for splits, it also considers the cost of specifying split values. A detailed description of PUBLIC(V) can be found in [RS98].

Data Set	breast cancer	car	letter	satimage	shuttle	vehicle	yeast
No. of Categorical Attributes	0	6	0	0	0	0	0
No. of Numeric Attributes	9	0	16	36	9	18	8
No. of Classes	2	4	26	7	5	4	10
No. of Records (Train)	469	1161	13368	4435	43500	559	1001
No. of Records (Test)	214	567	6632	2000	14500	287	483

Table 1: Real-life Data Sets

Data Set	breast cancer	car	letter	satimage	shuttle	vehicle	yeast
SPRINT	21.49	45.85	3283.20	1471.00	457.78	151.90	253.96
PUBLIC(1)	19.09	39.49	2799.17	1288.30	458.10	112.35	179.14
PUBLIC(S)	13.78	33.93	2786.57	1036.34	455.15	99.67	144.50
PUBLIC(V)	13.90	33.93	2793.32	1042.39	455.20	97.69	139.02
Max Ratio	56%	38%	18%	43%	0.6%	55%	83%

Table 2: Real-life Data Sets: Execution Time (secs)

The integrated PUBLIC algorithms are implemented using the same code base as SPRINT except that they perform pruning while the tree is being built. Furthermore, in PUBLIC, the pruning procedure is invoked repeatedly for each level, after all the nodes at the level have been split.

6.2 Real-life Data Sets

We experimented with eight real-life datasets whose characteristics are illustrated in Table 1. These datasets were obtained from the UCI Machine Learning Repository[3]. Data sets in the UCI Machine Learning Repository often do not have both training and test data sets. For these data sets, we randomly choose 2/3 of the data and used it as the training data set. The rest of the data is used as the test data set. Table 1 shows the number of records for both the training and the test data set.

6.3 Results on Real-life Data Sets

For each of the real-life data sets, we present the execution times for each algorithm (see Table 2). The final row of Table 2 (labeled "Max Ratio") indicates how much worse SPRINT is compared to the best PUBLIC algorithm. In general, we have found the max ratio number to be similar to the percentage of additional nodes generated by SPRINT compared to PUBLIC(V), the best PUBLIC algorithm. From the table, it follows that for certain data sets (e.g., yeast), SPRINT may take as much as 83% more execution time than PUBLIC(V). This confirms our conjecture that by pruning early, PUBLIC can result in a significant reduction in the number of redundant nodes generated, and consequently, in the execution times.

[3]Available at http://www.ics.uci.edu/~ mlearn/MLRepository.html.

6.4 Synthetic Data Set

In order to study the sensitivity of PUBLIC to parameters such as noise in a controlled environments, we generated synthetic data sets using the data generator used in [AIS93, MAR96, SAM96] and available from the IBM Quest home page[4]. Every record in the data sets has nine attributes and a class label which takes one of two values. A description of the attributes for the records is as shown in Table 3. Among the attributes, elevel, car and zipcode are categorical, while all others are numeric. Different data distributions are generated by using one of ten distinct classification functions to assign class labels to records. Function 1 involves a predicate with ranges on a single attribute value. Functions 2 and 3 use predicates over two attributes, while functions 4, 5, 6 have predicates with ranges on three attributes. Functions 7 through 9 are linear functions and function 10 is a non-linear function [AIS93]. Further details on these ten predicates can be found in [AIS93]. To model fuzzy boundaries between the classes, a perturbation factor for numeric attributes can be supplied to the data generator [AIS93]. In our experiments, we used a perturbation factor of 5%. We also varied the noise factor from 2 to 10% to control the percentage of noise in the data set. The number of records for each data set is set to 10000.

6.5 Results on Synthetic Data Sets

In Table 4, we present the execution times for the data sets generated by functions 1 through 10. For each data set, the noise factor was set to 10%. From the tables, we can easily see that PUBLIC outperforms SPRINT by a significant amount. For example, SPRINT is 279% slower than PUBLIC(V) for Function 8, and more than 200% slower than PUBLIC(V) for every other function. It is interesting to

[4]The URL for the page is http://www.almaden.ibm.com/cs/quest/demos.html.

Attribute	Description	Value
salary	salary	uniformly distributed from 20000 to 150000
commission	commission	if salary $\geq$ 75000 then commission is zero else uniformly distributed from 10000 to 75000
age	age	uniformly distributed from 20 to 80
elevel	education level	uniformly chosen from 0 to 4
car	make of the car	uniformly chosen from 1 to 20
zipcode	zip code of the town	uniformly chosen from 9 to available zipcodes
hvalue	value of the house	uniformly distributed from $0.5k100000$ to $1.5k100000$ where $k \in \{0, \ldots, 9\}$ depends on zipcode
hears	years house owned	uniformly distributed from 1 to 30
loan	total loan amount	uniformly distributed from 0 to 500000

Table 3: Description of Attributes in Synthetic Data Sets

Predicate No.	1	2	3	4	5	6	7	8	9	10
SPRINT	1531	1471	1399	1413	1454	1471	1277	1978	1336	1624
PUBLIC(1)	714	720	654	707	769	760	615	875	676	783
PUBLIC(S)	607	604	553	617	656	653	559	741	589	689
PUBLIC(V)	574	593	520	575	615	587	510	708	567	666
Max Ratio	267%	248%	269%	246%	236%	251%	250%	279%	236%	244%

Table 4: Synthetic Data Sets: Execution Time (secs)

observe that PUBLIC(1), the simplest of the PUBLIC algorithms, results in most of the realized gains in performance. The subsequent reductions in execution time due to PUBLIC(S) and PUBLIC(V) are not as high.

We also performed experiments to study the effects of noise on the performance of PUBLIC. We varied noise from 2% to 10% for every function, and found that the execution of the algorithms on all the data sets were very similar. As a result, in Figure 6, we only plot the execution times for functions 5 and 6. From the graphs, it follows that as execution times increase as the noise is increased. This is because as the noise is increased, the size of the tree and thus the number of nodes generated increases. Furthermore, the execution times for SPRINT increase at a faster rate than those for PUBLIC, as the noise factor is increased. Thus, PUBLIC results in better performance improvements at higher noise values. We also conducted experiments in which we varied the number of classes as well as the number of attributes in the data sets. However, we found that the performance of PUBLIC relative to SPRINT did not vary much for the different parameter settings.

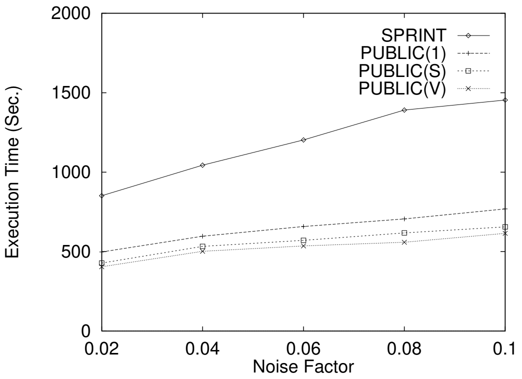
7 Concluding Remarks

In this paper, we proposed a new classifier, PUBLIC, that integrates the pruning phase into the building phase. Specifically, nodes that are certain to be pruned are not expanded during the building phase – as a result, fewer nodes are expanded during the building phase, and thus the amount of work (e.g., disk I/O) required to construct the decision tree is reduced. In order to determine, during the building phase, nodes that are certain to be pruned, we need to know the cost of encoding the subtrees at the node. To estimate the cost, we developed three techniques for computing a lower bound on the cost of a subtree at a "yet to be expanded" leaf node. By performing additional computation, each successive technique is able to generate more accurate estimates for the minimum cost subtree. Experimental results with real-life as well as synthetic data sets show that PUBLIC can result in significant performance improvements compared to traditional classifiers such as SPRINT. We also observed that PUBLIC(1), the simplest of the PUBLIC algorithms, results in most of the realized gains in performance. The subsequent reductions in execution time due to PUBLIC(S) and PUBLIC(V) are not as high.

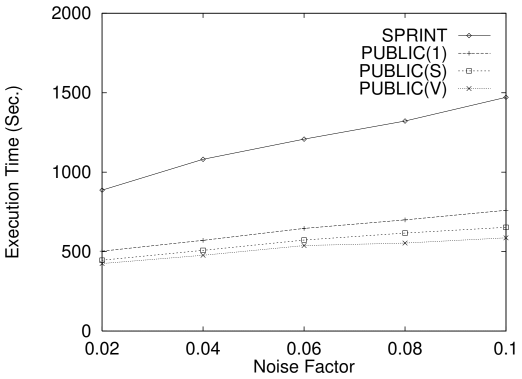
Acknowledgments: We would like to thank Narain Gehani, Hank Korth and Avi Silberschatz for their encouragement, Raghu Ramakrishnan for providing an initial implementation of SPRINT, and Johannes Gehrke and Alex Malamud for improving the SPRINT algorithm for our experiments. Without the support of Yesook Shim, it would have been impossible to complete this work.

References

[AGI+92] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, and Arun Swami. An interval classifier for database mining applications. In *Proc. of the VLDB Conference*, pages 560–573, Vancouver, British Columbia, Canada, August 1992.



(a) Predicate 5



(b) Predicate 6

Figure 6: Synthetic Data Sets: Execution Time (secs)

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[CKS+88] P. Cheeseman, James Kelly, Matthew Self, et al. AutoClass: A Bayesian classification system. In *5th Int'l Conf. on Machine Learning.* Morgan Kaufman, June 1988.

[Fay91] U. Fayyad. *On the Induction of Decision Trees for Multiple Concept Learning.* PhD thesis, The University of Michigan, Ann arbor, 1991.

[FI93] Usama Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of the 13th Int'l Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

[FMM96] Takeshi Fukuda, Yasuhiko Morimoto, and Shinichi Morishita. Constructing efficient decision trees by using optimized numeric association rules. In *Proc. of the Int'l Conf. on Very Large Data Bases*, Bombay, India, 1996.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Morgan Kaufmann, 1989.

[GRG98] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest - a framework for fast decision tree classification of large datasets. In *Proc. of the VLDB Conference*, New York City, NY, August 1998.

[HMS66] E. B. Hunt, J. Marin, and P. J. Stone, editors. *Experiments in Induction.* Academic Press, New York, 1966.

[KT81] R. Krichevsky and V. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207, 1981.

[MAR96] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *EDBT 96*, Avignon, France, March 1996.

[MRA95] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-95)*, Montreal, Canada, August 1995.

[MST94] D. Mitchie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[QR89] J. R. Quinlan and R. L. Rivest. Inferring decision trees using minimum description length principle. *Information and Computation*, 1989.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui87] J. R. Quinlan. Simplifying decision trees. *Journal of Man-Machine Studies*, 27:221–234, 1987.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993.

[Rip96] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, 1996.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publ. Co., 1989.

[RS98] R. Rastogi and K. Shim. PUBLIC: A decision tree classifier that integrates building and pruning. Technical report, Bell Laboratories, Murray Hill, 1998.

[SAM96] John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the VLDB Conference*, Bombay, India, September 1996.