Efficient Decision Tree Construction on Streaming Data

Ruoming Jin
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
jinr@cis.ohio-state.edu

Gagan Agrawal
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
agrawal@cis.ohio-state.edu

ABSTRACT

Decision tree construction is a well studied problem in data mining. Recently, there has been much interest in mining streaming data. Domingos and Hulten have presented a one-pass algorithm for decision tree construction. Their work uses Hoeffding inequality to achieve a probabilistic bound on the accuracy of the tree constructed.

In this paper, we revisit this problem. We make the following two contributions: 1) We present a numerical interval pruning (NIP) approach for efficiently processing numerical attributes. Our results show an average of 39% reduction in execution times. 2) We exploit the properties of the gain function entropy (and gini) to reduce the sample size required for obtaining a given bound on the accuracy. Our experimental results show a 37% reduction in the number of data instances required.

Overall, the two new techniques introduced here significantly improve the efficiency of decision tree construction on streaming data.

1. INTRODUCTION

Decision tree construction is an important data mining problem. Over the last decade, decision tree construction over disk-resident datasets has received considerable attention [11, 13, 25, 27]. More recently, the database community has focused on a new model of data processing, in which data arrives in the form of continuous streams [3, 4, 9, 12, 14, 16, 23, 30]. The key issue in mining on streaming data is that only one pass is allowed over the entire data. Moreover, there is a *real-time* constraint, i.e. the processing time is limited by the rate of arrival of instances in the data stream, and the memory available to store any summary information may be bounded. For most data mining problems, a one pass algorithm cannot be very accurate. The existing algorithms typically achieve either a deterministic bound on the accuracy [17], or a probabilistic bound [10]. Data mining algorithms developed for streaming data also serve as a useful basis for creating approximate, but scalable, implementations for very large and disk-resident datasets.

Domingos and Hulten have addressed the problem of decision tree construction on streaming data [10, 21]. Their algorithm guarantees a probabilistic bound on the accuracy of the decision tree that is constructed. In this paper, we revisit the problem of decision tree construction on streaming data. We make the following two contributions:

Efficient Processing of Numerical Attributes: One of the challenges in processing of numerical attributes is that the total number of candidate split points is very large, which can cause high computational and memory overhead for determining the best split point. The work presented by Domingos and Hulten is evaluated for categorical attributes only. We present a *numerical interval pruning* (NIP) approach which significantly reduces the processing time for numerical attributes, without any loss of accuracy. Our experimental results show an average of 39% reduction in execution times.

Using Smaller Samples Size for the Same Probabilistic Bound: Domingos and Hulten use Hoeffding's bound [19] to achieve a probabilistic bound. Hoeffding's result relates the sample size, the desired level of accuracy, and the probability of meeting this level of accuracy, and is applicable independent of the distribution of input data. In this paper, we show how we can use the properties of the gain function entropy (and gini) to reduce the sample size required to obtain the same probabilistic bound. Again, this result is independent of the distribution of input data. Our experimental results show that the number of samples required is reduced by an average of 37%.

Overall, these two contributions increase the efficiency of processing streaming data, where a real-time constraint may exist on the processing times, and only limited memory may be available. Our work also has important implications for analysis of streaming data beyond decision tree construction. We will be exploring these further in our future work.

The rest of the paper is organized as follows. Section 2 gives background information on the decision tree construction problem, and reviews the gain function entropy. The problems and issues in processing of streaming data are discussed in Section 3. Our new technique for efficient handling of numerical attributes is presented in Section 4. The new sampling method is described in Section 5. We evaluate our techniques in Section 6. We compare our work with related research efforts in Section 7 and conclude in Section 8.

2. DECISION TREE CONSTRUCTION

This section provides background information on the decision tree construction problem.

2.1 Decision Tree Classifier

Assume there is a data set $D = \{t_1, t_2, ..., t_N\}$, where $t_i =< \vec{x}, c >\in \vec{X} \times C$. $\vec{x} \overset{\text{def}}{=}< x_1, ..., x_m >$ is the data associated with the instance and c is the class label. Each x_j is called a field or an attribute of the data instance. $\vec{X} \overset{\text{def}}{=} X_1 \times ...X_m$ is the domain of data instances and X_j is the domain of the attribute x_j. The domain of an attribute can either be a *categorical* set, such as $\{red, blue, yellow\}$, or a *numerical* set, such as [1...100]. C is the domain of class labels. In this paper, our discussion will assume that there are only two distinct class labels, though our work can be easily extended to the general case.

The classification problem is to find a computable function $f : \vec{X} \mapsto C$, such that for any instance t extracted from the same distribution as D, $f(t.\vec{x})$ will give an as accurate as possible prediction of $t.c$. Decision tree classifiers are frequently used for achieving the above functionality. A decision tree classifier is typically a binary tree, where every non-leaf node t is associated with a predicate p. A predicate partitions the set of data instances associated with node based upon the value of a particular attribute x_i. If x_i belongs to a categorical domain, p is a subset predicate, for example, $p = true$ if $x_i \in \{red, blue\}$. If x_i belongs to a numerical domain, p is a range predicate, for example, $p = true$ if $x_i \leq 50$. Here, 50 is called the *cutting* or the *split* point.

Building a decision tree classifier generally includes two stages, a growing stage and a pruning stage. The tree growing stage involves recursively partitioning the dataset, till the records associated with a leaf node either all have the same class label, or their cardinality is below a threshold. In partitioning any node, a number of different candidate splitting conditions are evaluated. The best splitting condition is typically chosen to maximize a *gain function*, which are based upon impurity measurements such as gini or entropy. The pruning stage eliminates some nodes to reduce the tree size. This paper will focus only on the tree growing stage.

There are two commonly used metrics to evaluate a decision tree classifier, *inaccuracy* and *tree size*. Inaccuracy is defined as $r = Prob(f(t) \neq t.c)$, where t is a random instance from the underlying distribution. Tree size is defined as the total number of nodes in the tree, and measures the conciseness of the classifier.

2.2 Entropy Function

An impurity function gives a measurement of the impurity in the dataset. Originally proposed in the information theory literature, *entropy* has become one of the most popular impurity functions. Suppose, we are looking at a training dataset D. Let p_1 and p_2 be the proportion of instances with class labels 1 and 2, respectively. Clearly, $p_1 + p_2 = 1$.

Entropy function is defined as

$$Entropy(D) = -p_1 \times \log p_1 \; - \; p_2 \times \log p_2$$

$$= -p_1 \times \log p_1 \; - \; (1 - p_1) \times \log(1 - p_1)$$

Now, suppose we split the node using a split predicate c and create two subsets D_L and D_R, which are the *left* and *right* subsets, respectively. Let p_L denote the fraction of the data instances in D that are associated with D_L. Then, the gain associated with splitting using the predicate c is defined as

$$g_c = g(D_L, D_R)$$

$$= \; Entropy(D) - ((p_L \times Entropy(D_L)) + (p_R \times Entropy(D_R)))$$

Further, let p_{1L} be the proportion of instances with the class label 1 within D_L, and let p_{1R} be the proportion of instances with the class label 1 within D_R. Because $Entropy(D)$ is a constant, we can treat g_c as a function of three variables, p_L, p_{1L}, and p_{1R}.

$$g_c \; = \; g(p_L, p_{1L}, p_{1R}) \; = \; Entropy(D)$$
$$-p_L \times (-p_{1L} \times \log p_{1L} - (1 - p_{1L}) \times \log(1 - p_{1L}))$$
$$-(1 - p_L) \times (-p_{1R} \times \log p_{1R} - (1 - p_{1R}) \times \log(1 - p_{1R}))$$

For a given attribute x_i, let $g(x_i)$ denote the best gain possible using this attribute for splitting the node. If we have m attributes, we are interested in determining i, such that

$$g(x_i) \; \geq \max_{j \in \{1,\ldots,m\} - \{i\}} g(x_j)$$

If more than one attribute satisfies this condition, a pre-defined rule (such as randomization) can be used to select one of these.

2.3 Evaluating Split Conditions

Selecting the attribute and the split condition that maximizes information gain is the key step in decision tree construction. The scalable decision tree construction algorithms proposed in the literature [11, 13, 25, 27] take a number of approaches towards performing this step efficiently.

The major issues that need to be addressed are, what information is required for evaluating different candidate split conditions, and how can this information be stored and processed efficiently. In the initial work on decision tree construction for disk-resident datasets, the training dataset is separated into *attribute* lists [25, 27]. For a particular attribute, the attribute list maintains the record-identifier and the value of that attribute for the training record. Moreover, for efficiently choosing the best split point for a numerical attribute, the attribute lists for such attributes is kept sorted.

A significantly different approach is taken as part of the RainForest approach by Gehrke *et al.* [13]. Here, a new data structure called an AVC (Attribute-Value, Classlabel) group is used. An AVC group for a decision tree node comprises AVC sets for all attributes. For a given attribute and a node being processed, the AVC set simply records the count of occurrence of each class label for each distinct value the attribute can take. Thus, it is essentially a *class histogram*. The size of the AVC set for a given node and attribute is proportional to the product of the number of distinct values of the attribute and the number of distinct class labels.

3. STREAMING DATA PROBLEM

In this section, we focus on the problem of decision tree construction on streaming data. We give a template of the algorithm, which will be used as the basis for our presentation in the next three sections. Moreover, we describe how sampling is used to achieve a probabilistic bound on the quality of the tree that is constructed.

3.1 Algorithm Template

```
StreamTree(Stream $\mathcal{D}$)
 global $Tree\ root,\ Queue\ \mathcal{Q}, \mathcal{AQ}$;
 local $Node\ node$;
 local $Tuple\ t$;
 $\mathcal{Q} \leftarrow NULL$; $\mathcal{AQ} \leftarrow NULL$;
 $add(root, \mathcal{AQ})$;
 while not $(empty(\mathcal{Q})$ and $empty(\mathcal{AQ}))$
      $t \leftarrow \mathcal{D}.get()$;
      $node \leftarrow classify(root, t)$;
      if $node \in \mathcal{AQ}$
           $add(node.sample, t)$;
           if $node.satisfy\_stop\_condition$
                $remove(node, \mathcal{AQ})$;
           if $node.enough\_samples()$
                $use\ split\ function\ to\ get\ the\ best\ split$;
                $(node_1, node_2) \leftarrow node.create()$;
                $remove(node, \mathcal{AQ})$;
                $add((node_1, node_2), \mathcal{Q})$;
           while $enough\_memory(\mathcal{AQ}, \mathcal{Q})$
                $get(node, \mathcal{Q})$;
                $add(node, \mathcal{AQ})$;
```

Figure 1: StreamTree Algorithm

We first list the issues in analyzing streaming data. The total size of the data is typically much larger than the available memory. It is not possible to store and re-read all data from memory. Thus, a single pass algorithm is required, which also needs to meet the real-time constraint, i.e. the computing time for each item should be less than the interval between arrival times for two consecutive items.

A key property that is required for analysis of streaming data is that the data instances arriving follow an underlying distribution. It implies that if we collect a specific interval of streaming data, we can view them as a random sample taken from the underlying distribution. It may be possible for a decision tree construction algorithm to adjust the tree to changes in the distribution of data instances in the stream [21], but we do not consider this possibility here.

Figure 1 presents a high-level algorithm for decision tree construction on streaming data. This algorithm forms the basis for our presentation in the rest of the paper. The algorithm is based upon two queues, $\mathcal{Q}$ and $\mathcal{AQ}$. $\mathcal{AQ}$ stands for *active queue* and denotes the set of decision tree nodes that we are currently working on expanding. $\mathcal{Q}$ is the set of decision tree nodes that have not yet been split, but are not currently being processed. This distinction is made because actively processing each node requires additional memory. For example, we may need to store the counts associated with each distinct value of each attribute. Therefore, the set AQ is constructed from the set $\mathcal{Q}$ by including as many nodes as possible, till sufficient memory is available. The algorithm is initiated by putting the root of the decision tree in the set $\mathcal{Q}$.

The input to the algorithm is a stream of data instances, denoted by $\mathcal{D}$. We successively obtain a data instance t from this stream. We determine the current decision tree node (denoted by $node$) that this data instance belongs to. If $node$ belongs to the set $\mathcal{AQ}$, then the data instance is added to the set of samples available to process *node*. We then check if $node$ satisfies the stop condition (which can only be applied statistically). If so, $node$ is removed from $\mathcal{AQ}$. Otherwise, we check if we now have sufficient information to split $node$. If so, the node is split, removed from $\mathcal{AQ}$, and its two child nodes are added to the set $\mathcal{Q}$. The algorithm terminates when both $\mathcal{Q}$ and $\mathcal{AQ}$ are empty.

The algorithm, as presented here, is only different from the work by Domingos and Hulten [10] in not assuming that all nodes at one level of the tree can be processed simultaneously. The memory requirements for processing a set of nodes is one of the issues we are optimizing in our work. If the memory requirements for processing a given node in the tree are reduced, more nodes can be fit into the set $\mathcal{AQ}$, and therefore, it is more likely that a given data instance can be used towards partitioning a node.

Besides the memory requirements, this algorithm also exposes a number of other issues in decision tree construction on streaming data. As we can see, one crucial problem here is to decide when we have sufficient samples available to make the splitting decision. Another question is, what information needs to be stored from the sample collected in order to make the splitting conditions. Simply storing all samples is one, but not the only possibility. Computationally, yet another issue is how we efficiently examine all possible splitting conditions associated with a node. Particularly, the number of distinct values associated with a numerical attribute can be very large, and can make it computationally demanding to choose the best split point.

3.2 Using Sampling

Here, we review the problem of selecting splitting predicate based upon a sample. Our discussion assumes the use of entropy as the gain function, though the approach can be applied to other functions such as gini.

Let S be a sample taken from the dataset D. We focus on the gain g_c associated with a potential split point c for a numerical attribute x_i. If p_1 and p_2 are the fractions of data instances with class labels 1 and 2, respectively, $\overline{p_1}$ and $\overline{p_2}$ are the estimates computed using the sample. Similarly, we have the definitions for $\overline{p_L}$, $\overline{p_{1L}}$, $\overline{p_{1R}}$, $\overline{g_c}$, and $\overline{Entropy(D)}$.

We have,

$$\overline{g_c} \; = \; g(\overline{p_L}, \overline{p_{1L}}, \overline{p_{1R}}) \; = \; \overline{Entropy(D)}$$

$$-\overline{p_L}(1 - \overline{p_{1L}} \log \overline{p_{1L}} - (1 - \overline{p_{1L}}) \log(1 - \overline{p_{1L}}))$$

$$-(1 - \overline{p_L})(\overline{p_{1R}} \log \overline{p_{1R}} - (1 - \overline{p_{1R}}) \log(1 - \overline{p_{1R}}))$$

The value of $\overline{g_c}$ serves as the estimate of of g_c. Note that we do not need to compute $\overline{Entropy(D)}$, since we are only interested in the relative values of the gain values associated with different split points.

Now, we consider the procedure to find the best split point using the above estimate of gains. Let $\overline{g(x_i)}$ be the estimate of the best gain that we could get from the attribute x_i. Assuming there are m attributes, we will use the attribute x_i, such that

$$\overline{g(x_i)} \; - \max_{j \in \{1,\ldots,m\} - \{i\}} \overline{g(x_j)} \geq \epsilon$$

where ϵ is a small positive number. The above condition (called the *statistical test*) is used to infer that x_i is likely to satisfy the *original test* for choosing the best attribute, which is

$$g(x_i) \; \geq \max_{j \in \{1,\ldots,m\} - \{i\}} g(x_j)$$

To describe our confidence of above statistical inference, a parameter α is used. α is the probability that the original test holds if the statistical test holds, and should be as close to 1 as possible. ϵ can be viewed as a function of α and sample size $|S|$, i.e.

$$\epsilon \; = \; f(\alpha, |S|)$$

Domingos and Hulten use the Hoeffding bound [19] to construct this function. The specific formula they use is

$$\epsilon_h \; = \; \sqrt{\frac{R^2 \ln(1/(1-\alpha))}{2 \times |S|}}$$

where R is the spread of the gain function. In this context, where there are two classes and entropy is used as the impurity function, $R \; = \; 2$. In Section 5, we will describe an alternative approach, which reduces the required sample size.

Based upon the probabilistic bound on the splitting condition for each node, Domingos and Hulten derive the following result on the quality of the resulting decision tree. This result is based on the measurement of *intensional disagreement*. The intensional disagreement $\triangle_i$ between two decision trees DT_1 and DT_2 is the probability that the path of an example through DT_1 will differ from its path through DT_2.

THEOREM 1. *If HT_α is the tree produced by the algorithm for streaming data with desired accuracy level α, DT_* is the tree produced by batch processing an infinite training sequence, and p is the leaf probability, i.e., the probability that a given example reaches a leaf node at any given level of the decision tree, then*

$$E[\triangle_i(HT_\alpha, DT_*] \leq (1-\alpha)/p$$

where $E[\triangle_i(HT_\alpha, DT_*)]$ is the expected value of $\triangle_i(HT_\alpha, DT_*)$ taken over an infinite training sequence.

4. A NEW ALGORITHM FOR HANDLING NUMERICAL ATTRIBUTES

In this section, we present our *numerical interval pruning* approach for making decision tree construction on streaming data more memory and computation efficient.

4.1 Problems and Our Approach

One of the key problems in decision tree construction on streaming data is that the memory and computational cost of storing and processing the information required to obtain the best split gain can be very high. For categorical attributes, the number of distinct values is typically small, and therefore, the class histogram does not require much memory. Similarly, searching for the best split predicate is not expensive if number of candidate split conditions is relatively small.

However, for numerical attributes with a large number of distinct values, both memory and computational costs can be very high. Many of the existing approaches for scalable, but multi-pass, decision tree construction require a preprocessing phase in which attribute lists for numerical attributes are sorted [25, 27]. Preprocessing of data, in comparison, is not an option with streaming datasets, and sorting during execution can be very expensive. Domingos and Hulten have described and evaluated their one-pass algorithm focusing only on categorical attributes [10]. It is claimed that numerical attributes can be processed by allowing predicates of the form "$x_i < x_{ij}$", for each distinct value x_{ij}. This implies a very high memory and computational overhead for determining the best split point for a numerical attribute.

We have developed a Numerical Interval Pruning (NIP) approach for addressing these problems. The basis of our approach is to partition the range of a numerical attribute into *intervals*, and then use statistical tests to *prune* these intervals. At any given time, an interval is either *pruned* or *intact*. An interval is pruned if it does not appear likely to include the split point. An intact interval is an interval that has not been pruned. In our current work, we have used *equal-width* intervals, i.e. the range of a numerical attribute is divided into intervals of equal width.

In Section 2.3, we had discussed how we can either store samples, or create class histograms to have sufficient information to determine the best split condition. In the numerical interval pruning approach, we instead maintain the following sets for each node that is being processed.

Small Class Histograms: This is primarily comprised of class histograms for all categorical attributes. The number of distinct elements for a categorical attribute is not very large, and therefore, the size of the class histogram for each attribute is quite small. In addition, we also add the class histogram for numerical attributes for which the number of distinct values is below a threshold.

Concise Class Histograms: The range of numerical attributes which have a large number of distinct elements in the dataset is divided into intervals. For each interval of a numerical attribute, the concise class histogram records the number of occurrences of instances with each class label whose value of the numerical attribute is within that interval.

Detailed Information: The detailed information for an interval can be in one of the two formats, depending upon what is efficient. The first format is class histogram for the samples which are within the interval. When the number of samples is large and the number of distinct values of a numerical attribute is relatively small, this format is more efficient. The second format is to simply maintain the set of samples with each class label. It is not necessary to process the detailed information in the pruned interval to get best split point.

The advantage of this approach is that we do not need to process detailed information associated with a pruned interval. This results in a significant reduction in the execution time, but no loss of accuracy. Further, as we will argue towards the end of this section, we may not even store the detailed information associated with a pruned interval. This further reduces the memory requirements, but can result in a small loss of accuracy.

```
NIP-Classifier(Node $\mathcal{N}$, Stream $\mathcal{D}$)
  while not $satisfy\_stop\_condition(\mathcal{N})$
      { * Get Some Samples from Stream $\mathcal{D}$ *}
      $Sample\ \mathcal{S} \leftarrow (\mathcal{D}.get())$;
      $Update\_Small\_ClassHist(\mathcal{S})$;
      $Update\_Concise\_ClassHist(\mathcal{S})$;
      $Update\_Detailed\_Information(\mathcal{S})$;
      { * Find the best gain *}
      $g' \leftarrow$ Find_Best_Gain(ClassHist);
      $\bar{g} \leftarrow UnPruning(g'$, Concise_ClassHist$)$;
      { * Split *}
      if $Statistically\_Best\_Gain(\bar{g})$
          $Split\_Node(\mathcal{N})$;
          $break$;
      { * Pruning *}
      $Pruning(\bar{g}$, Concise_ClassHist$)$;
```

Figure 2: NIP Algorithm for Numerical Attributes Handling

The main challenge in the algorithm is to effectively but correctly prune the intervals. *Over-pruning* is a situation occurring when an interval does not appear likely to include the split point after we have analyzed a small sample, but could include the split point after more information is made available. *Under-pruning* means that an interval does not appear likely to include the split point but has not yet been pruned. We refer to over-pruning and under-pruning together as *false pruning*.

The pseudo-code for our Numerical Interval Pruning (NIP) algorithm is presented in Figure 2. Here, after collecting some samples, we use small class histograms, concise class histograms, and the detailed information from intact intervals and get an estimate of the best (highest) gain. This is denoted as g'. Then, by using g', we unprune intervals that look promising to contain the best gain, based upon the current sample set. The best gain $\bar{g}$ can come from g' or a newly unpruned intervals. Then, by performing a statistical test, we check if we can now split this node. If not, we need to collect more samples. Before that, however, we check if some additional intervals can be pruned.

The rest of this section presents more technical details, the correctness, and the computational and memory cost optimizations.

4.2 Technical Details

In this subsection, we discuss the details of how pruning and testing for false pruning are performed.

For our discussion here, we initially assume that we have processed the entire dataset, and are trying to prune the space of potential split points from the range of a numerical attribute. Assume that g is the best (highest) gain possible from any splitting condition associated with a node of the decision tree. Suppose $[x_i, x_{i+1})$ is the i^{th} interval on the numerical attribute X. Let the class distribution of the interval i be

$$\vec{H}_i = (h_i^1, h_i^2, \ldots h_i^c)$$

where $h_i^j, 1 \leq j \leq c$ is the number of training records with the class label j that fall into the interval i, and c is the total number of class labels in the training data.

We want to determine if any point within this interval can provide a higher gain than g. For this, we need to determine the highest possible value of gain from any point within this interval. For the boundary point x_i, we define the cumulative distribution function

$$\vec{N}_{x_i} = (n_{x_i}^1, \ldots, n_{x_i}^c)$$

where,

$$n_{x_i}^k = \sum_{j=1}^{i-1} h_j^k, 1 \leq k \leq c$$

Here, $n_{x_i}^k$ is the number of training records with the class label k such that their value of the numerical attribute under consideration is less than x_i. Similarly, for the boundary point x_{i+1}, we have

$$\vec{N}_{x_{i+1}} = (n_{x_i}^1 + h_i^1, \ldots, n_{x_i}^c + h_i^c)$$

Now, consider any point y between x_i and x_{i+1}. Let the cumulative distribution function be

$$\vec{N}_y = (n_y^1, \ldots, n_y^c)$$

Clearly, the following property holds.

$$\forall j,\ 1 \leq j \leq c,\ n_{x_i} \leq n_y \leq n_{x_{i+1}} \quad (a)$$

If we do not have any further information about the class distribution of the training records in the interval, all points y satisfying the above constraint need to be considered in determining the highest gain possible from within the interval $[x_i, x_{i+1})$. Formally, we define the set P of possible internal points as all values within the interval $[x_i, x_{i+1})$ that satisfy the constraint a.

The number of points in the set P can be very large, making it computationally demanding to determine the highest possible gain from within the interval. However, we are helped by a well-known mathematical result. To state this result, we define a set S, comprising of *corner points* within the interval. Formally,

$$S = \{s | s \in [x_i, x_{i+1}] \wedge \vec{N}_s\ satisfies\ that$$

$$\forall j, 1 \leq j \leq c,\ (n_s^j = n_{x_i}^j) \vee (n_s^j = n_{x_{i+1}}^j)\}$$

It is easy to see that the set S has 2^c points. Now, the following result allows us to compute the highest possible gain from the interval very efficiently.

LEMMA 1. *Let f be a concave gain function. Let P be the set of possible internal points of interval i, and S be the set of corner points. Then*

$$\max_{p \in P} (f(\vec{N}_p)) \leq \max_{s \in S} (f(\vec{N}_s))$$

The above lemma is derived from a general mathematical theorem (see Magasarian [24]) and was also previously used by Gehrke *et al.* in the BOAT approach [11]. Note that the gain functions entropy and gini are concave functions.

By recording the frequency of intervals, computing gains at the interval boundaries, and applying this lemma, we can compute the upper bound u_i of the gain possible from the interval. If we already know the best gain g, we can compare u_i and g. The interval can contain the split point only if $u_i \geq g$, and can be pruned if this is not the case.

As described above, this method is useful only if we have already scanned the entire dataset, know the value of the best gain g as well as the gains at the interval boundaries. However, we perform this pruning after the sampling step, i.e., by examining only a fraction of the training records. As we had described earlier in this section, we use a sample and compute small and concise class histograms. Thus, for numerical attributes with a large number of distinct values, we are processing the class frequencies for only the intervals, and only using the sample. As defined above, g is the best gain possible using the entire dataset and all possible split conditions. Now, let $\bar{g}$ be the best gain computed using the current sample set. Further, let $g\prime$ be the best gain noted after using small class histograms, concise class histograms and the detailed information from the intact intervals. This is the value first computed by our algorithm.

We have,

$$g' \leq \bar{g}$$

This follows simply from the fact that g' is computed from among a subset of the split points that $\bar{g}$ would be computed from.

Using the values of gain at interval boundaries, and using the Lemma 1, we can estimate the upper bound on the gain possible from a split point within a given interval i. We denote this value as $\bar{u_i}$, and is an estimate (using the sample) of the value u_i that could be computed using the entire dataset.

We focus on the function

$$\triangle = g - u_i$$

If we have computed g and u_i using the entire dataset, we can prune an interval i if $\triangle > 0$. However, using sampling, we can only have estimate of this function. Suppose, we consider

$$\bar{\triangle} = \bar{g} - \bar{u_i}$$

Based upon our discussion in Section 3.2, if $\bar{\triangle} \geq \epsilon$, then with probability α, we have $\triangle > 0$, where, ϵ, α, and the sample size are related through a function. Thus, pruning using the condition $\bar{\triangle} > \epsilon$ gives us a probabilistic bound on the accuracy of pruning, i.e., we could still prune incorrectly, but the probability is low and bounded.

Further, since we do not even have the value of $\bar{g}$, we use g'. Since $g' \leq \bar{g}$, we have

$$(g' - \bar{u_i}) > \epsilon \implies (\bar{g} - \bar{u_i}) > \epsilon$$

Thus, after the first step, we perform statistical test using the condition $g' - \bar{u_i} > \epsilon$ for all of the pruned intervals i. We call an interval *over-pruned* if it is pruned using the statistical estimates described above from a small sample, but after the more data instances are available, it turns out not to be the case. Note that to be able to unprune intervals, we must not discard the detailed information associated with an interval that may be marked as pruned. Thus, the algorithm presented so far only reduces the computational costs, but not the memory costs.

Next, we briefly discuss how we test for under-pruning. The algorithm simply computes $\bar{u_i}$ for intact intervals to see if the following condition holds:

$$(\bar{g} - \bar{u_i}) > \epsilon$$

If it is true, it means that this interval is not likely to have the best split point and can be pruned.

By combining g' with the detailed information from the over-pruned intervals, we get a new best gain $\bar{g}$. We know that this is the best gain possible from looking at the current sample size. This is because if the pruned intervals could not achieve the gain $g' - \epsilon$, they cannot achieve the gain $\bar{g} - \epsilon$ either.

Suppose, we decide to partition a node using a sample S and the current best gain $\bar{g}$. This gain is identical to the best gain that an algorithm not performing any pruning would achieve by using the same sample set, as we formulate through the following theorem.

THEOREM 2. *The best gain $\bar{g}$ computed using our numerical interval pruning approach is the same as the one computed by an algorithm that uses full class histograms, provided the two algorithms use the same sample set.*

Proof: This follows from our discussion above. □

Thus, the numerical interval pruning approach we have presented does not limit accuracy in any way, as compared to any other algorithm that uses samples.

4.3 Computational and Memory Costs and Optimization

Initially, let us focus on the computational costs associated with the algorithm here. As a new data instance is received we check if it is associated with a node that is being processed. If so, a number of update operations are performed. The processing time, however, is a constant. The dominant computational part is when we want to determine the best split condition. Unlike in a batch algorithm, this step may have to repeated several times, till we have a sufficient statistical confidence to perform the split. This step requires processing the small and concise histograms, and the detailed information associated with intact intervals. Our experiments have determined that the main cost is associated with the processing of detailed information. Thus, pruning of intervals is crucial for reducing this cost.

In the algorithm presented here, unpruning intervals is a requirement for provably achieving the same accuracy as in an algorithm that does not do any pruning. Therefore, we need to maintain and continue to update the detailed information associated with pruned intervals. However, the probability of over-pruning can be shown to be very small. Therefore, we can modify our original algorithm to not store the detailed information associated with pruned intervals. This optimization has two benefits. First, the memory requirements are reduced significantly. Second, we can further save on the computational costs by not having to update detailed information associated with a pruned interval.

5. A NEW SAMPLING APPROACH

This section introduces a new approach for choosing the sample size. As compared to the Hoeffding inequality [19] based approach used by Domingos and Hulten [10], our method allows the same probabilistic accuracy bound to be achieved using significantly smaller sample sizes.

5.1 Exploiting Gain Functions

As we have mentioned previously, the one-pass decision tree construction algorithm by Domingos and Hulten uses Hoeffding inequality to relate the bound on the accuracy ϵ, the probability α, and the sample size $|S|$. Hoeffding bound based result is independent of the distribution of the data instances in the dataset. Here, we derive another approach, which is still independent of the distribution of the data instances, but uses properties of gain functions like entropy and gini.

We use the following theorem, also known as the *multivariant delta* result [5]. Here, the symbol $E(x)$ denotes the expected value of a variable x, $Cov(x, y)$ denotes the *covariance* of the two variables x and y, and $N(0, \tau^2)$ is the normal distribution with the mean 0 and the variance (or the square of the standard deviation) τ^2.

THEOREM 3. *(Multivariate Delta Method) Let $X_1, \dots, X_n$ be a random sample. Let $X_i = X_{1i}, \dots, X_{pi}$. Further, let $E(X_{ij}) = \mu_i$ and $Cov(X_{ij}, X_{jk}) = \sigma_{ij}$. Let $\overline{X}_i$ be the mean of $X_{i1}, X_{i2}, \dots, X_{in}$ and let $\vec{\mu} = (\mu_1, \dots, \mu_p)$. For a given function g with continuous first partial derivatives, we have*

$$g(\overline{X}_1, \dots, \overline{X}_p) - g(\vec{\mu}) \rightarrow N(0, \tau^2/n)$$

where,

$$\tau^2 = \sum\sum \sigma_{ij} \frac{\partial g(\vec{\mu})}{\partial \mu_i} \cdot \frac{\partial g(\vec{\mu})}{\partial \mu_j}$$

Proof: See the reference [5], for example. □

Below, we show the application of the above result on the gain function entropy. This could similarly be applied on the gain function gini, but we do not present the details here.

In applying the above result on the entropy function, we consider the following. The function g is a function of three measurements, p_L, p_{1L}, and p_{1R}. The three values or measurements are independent of each other, i.e. the covariance $Cov(x, y)$ is 0 if $x \neq y$.

LEMMA 2. *Let n be the sample size of S, N be the normal distribution. Then, for the entropy function g, we have*

$$\overline{g_x} = g(\overline{p_L}, \overline{p_{1L}}, \overline{p_{1R}}) \rightarrow N(g(x), \tau_x^2/n)$$

where,

$$\tau_x^2 = (\frac{\partial g}{\partial p_L})^2 \cdot p_L(1 - p_L)$$

$$+(\frac{\partial g}{\partial p_{1L}})^2 \cdot p_{1L}(1 - p_{1L}) + (\frac{\partial g}{\partial p_{1R}})^2 \cdot p_{1R}(1 - p_{1R})$$

Proof: The proof follows from the application of the multivariate delta result (presented above), and the observation that the first derivatives for entropy are continuous functions (details are omitted here). □

Next, we focus on the following problem. Assume there is a point y belonging to the attribute $X_j, i \neq j$. We need to determine if $g_x > g_y$ or $g_x < g_y$, using just the sample S. Because y also satisfies the Lemma 2, and x and y are independent, we have

$$\overline{g_y} \rightarrow N(g_y, \tau_y^2/n)$$

Therefore,

$$\overline{g_x} - \overline{g_y} \rightarrow N(g_x - g_y, (\tau_x^2 + \tau_y^2)/n)$$

This leads to the following lemma.

LEMMA 3. *Let*

$$\epsilon_n = z_\alpha \cdot \frac{\sqrt{\tau_x^2 + \tau_y^2}}{\sqrt{n}}$$

where z_α is the $(1 - \alpha)$th percentile of the standard normal distribution. If $\overline{g_x} - \overline{g_y} \geq \epsilon_n$, then with probability α, we have $g_x \geq g_y$. If $\overline{g_x} - \overline{g_y} \leq -\epsilon_n$, then with probability α, we have $g_y \geq g_x$.

Proof: The above lemma follows from the application of well known results on simultaneous statistical inference [20]. □

We call the above test the *Normal test*.

5.2 Sample Size Problem

Once a desired level of accuracy α is chosen, the key issue with the performance of a one-pass algorithm is the sample size selection problem, i.e. how large a sample is needed to find the best split point with the probability α. Specifically, we are interested in the sample size that could separate g_{x_a} and g_{y_b}, where x_a and x_b are the points that maximize the gain of split function for the top two attributes X_a and X_b.

Let $\overline{g_{x_a}} - \overline{g_{y_a}} = \epsilon$. Thus, by normal distribution, the required sample size is

$$N_n = \frac{z_\alpha^2 \sqrt{\tau_x^2 + \tau_y^2}}{\epsilon^2}$$

The required sample size from Hoeffding bound is

$$N_h = \frac{R^2 \ln(1/(1-\alpha))}{2\epsilon^2}$$

Comparing the above two equations, we have the following result.

THEOREM 4. *The sample size required using the normal test will always be less or equal to the sample size required for the Hoeffding test, i.e.,*

$$N_n \leq N_h$$

Proof: This follows from comparing the two equations above. □

6. EXPERIMENTAL RESULTS

In this section, we report on a series of experiments designed to evaluate the efficacy and performance of our new techniques. Particularly, we are interested in evaluating 1) the advantages of using Numerical Interval Pruning (NIP), and 2) the advantages of using normal distribution of the estimate of entropy function, as compared to Hoeffding's bound.

The datasets we used for our experiments were generated using a tool described by Agrawal *et al.* [1]. There were two reasons for using these datasets. First, these datasets have been widely used for evaluating a number of existing efforts on scalable decision construction [13, 11, 25, 27]. Second, the only real datasets that we are aware of are quite small in size, and therefore, were not suitable for our experiments. The datasets we generated had 10 million training records, each with 6 numerical attributes and 3 categorical attributes. We used the functions 1, 6, and 7 for our experiments. For each of these functions, we generated separate datasets with 0%, 2%, 4%, 6%, 8%, and 10% noise.

In growing the decision tree, our implementation did not expand a node any further if one of the following conditions were true: 95% or greater fraction of the training records had the same class label, the depth of the node was 12, or less than 1% of all training records were associated with the node. For each node, we start evaluation for finding split condition after at least 10,000 records associated with the node had been read. Further, we reevaluated each node every time after another 5,000 records had been read. The range of each numerical attribute was divided into 500 equal sized intervals. The value of α used in our experiments was $1 - 10^{-6}$. All our experiments were conducted on a 700 MHz Intel Pentium III machine, with 1 GB of SDRAM and a 18 GB disk with 15000 rpm Ultra 160 SCSI drive.

The results from experiments designed to evaluate the NIP approach and the benefits of using normal distribution of the estimate of entropy function are reported together. We created 4 different versions, all based upon the basic StreamTree algorithm presented in Figure 1. `Sample-H` is the version that uses Hoeffding bound, and stores samples to evaluate candidate split conditions. `ClassHist-H` uses Hoeffding bound and creates full class histograms. `NIP-H` and `NIP-N` use numerical interval pruning, with Hoeffding bound and the normal distribution of entropy function, respectively. The version of NIP that we implemented and evaluated creates intervals after 10,000 samples have been read for a node, performs interval pruning, and then deletes the samples. Thus, unpruning is not an option here, and therefore, the accuracy can be lower than an approach that uses full class histograms. Our implementation used a memory bound of 60 MB for all four versions. Consistent with what was reported for the implementation of Domingos and Hulten, we performed *attribute pruning*, i.e., did not further consider an attribute that appeared to show poor gains after some samples were analyzed.

Figure 3 shows the average number of nodes in the decision tree generated using functions 1, 6, and 7, and using noise levels of 0%, 2%, 4%, 6%, 8%, and 10%, respectively. This number does not change in any significant way over the four different versions we experimented with. As expected, the size of the decision tree increases with the level of noise in data.

One interesting question is, what inaccuracy may be introduced by our version of `NIP-H`, since it does not have the option of unpruning. Figure 4 shows the increase in inaccuracy for `NIP-H`, as compared to the average of inaccuracy from `Sample-H` and `ClassHist-H`. As can be seen from the figure, there is no significant chance in inaccuracy. Note that whenever a different set of data instances are used to split a node, the computed inaccuracy value can be different. Similarly, Figure 5 shows the increase in inaccuracy for `NIP-N`, as compared to the average of inaccuracy from `Sample-H` and `ClassHist-H`. Again, there is no significant change, and the average value of the difference is very close to zero.

Figures 6, 7, and 8 show the execution times for decision tree construction with the four versions and different levels of noise, for functions 1, 6, and 7, respectively. Through-out, we will focus on comparing the performance of `NIP-N` and `NIP-H` with the better one between `Sample-H` and `ClassHist-H`, which we denote by `existing`.

Initially, we focus on functions 6 and 7. For function 6, the execution times of `NIP-H` are between 40% and 70% of the execution time of `existing`. Moreover, `NIP-N` further reduces the execution time by between 7% and 80%. For function 7, the execution times with `NIP-H` are between 35% and 75% of `existing`. `NIP-N` further reduces the execution times by between 3% and 65%. Results are relatively mixed from using the function 1. Our best version `NIP-N` is significantly better in 3 of the 6 cases, but quite comparable (i.e. either marginally better or worse) in other 3 cases. This is because with this function, the decision to choose the best split condition can usually be made by examining only a small number of samples. Therefore, the use of numerical interval pruning does not give better results in many cases.

We next compare these four version using two metrics we consider important. These metrics are, *total instances read* (TIR), and *instances actively processed* (IAP). TIR is the number of samples or data instances that are read before the decision tree converges. When a sample is read, it cannot always be used as part of the algorithm. This is because it may be assigned to a node that does not need to be expanded any further, or is not being processed currently because of memory considerations. Therefore, we measure IAP as the number of data instances that were used for evaluating candidate split conditions. Figures 9, 10, and 11 show TIR for the four versions and for functions 1, 6, and 7, respectively. The use of class histograms results in high memory requirements, which

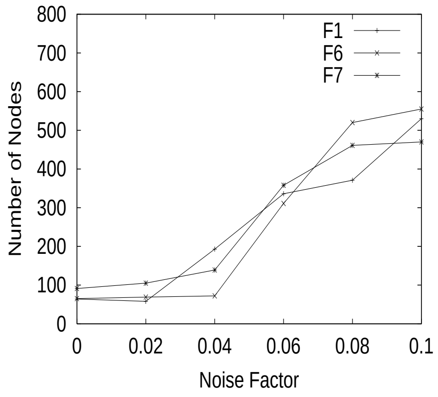

Figure 3: Concept Size

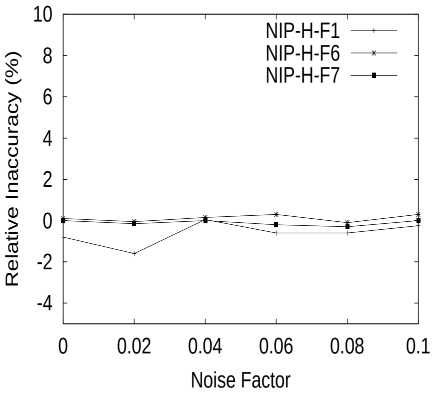

Figure 4: Inaccuracy with NIP

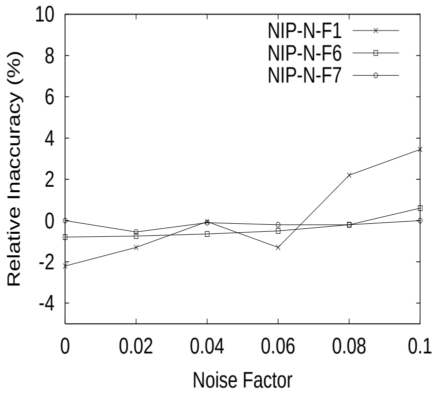

Figure 5: Inaccuracy with Normal

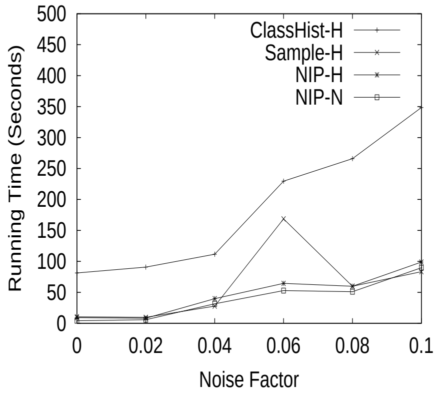

Figure 6: Running Time: F1

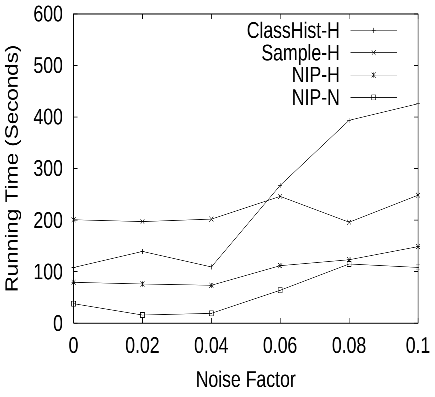

Figure 7: Running Time: F6

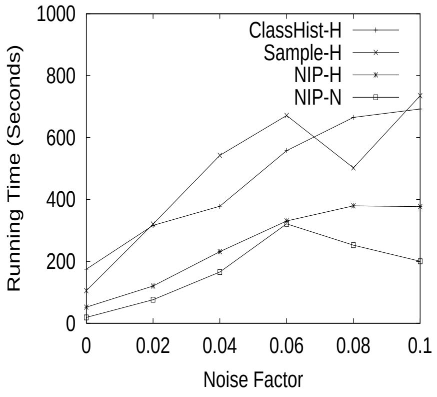

Figure 8: Running Time: F7

results in very high values of TIR. In all cases, the values of TIR for Sample-H and NIP-H are almost identical. This shows the main performance advantage of the NIP approach comes because of the reduction in computational costs, and not because of memory. Moreover, the reduction in execution time with the use of NIP approach shown earlier is actually a reduction in processing time per data instance, which is an important issue in processing of data streams. Comparison between NIP-H and NIP-N versions shows the benefits of exploiting the normal distribution of the estimated entropy function. The reduction in TIR is between 8% and 40% for function 1, between 18% and 60% for function 6, and between 16% and 55% for function 7. Figures 12, 13, and 14 show the values of IAP for functions 1, 6, and 7, respectively. The three versions, Sample-H, ClassHist-H, and NIP-H have almost identical values of IAP. This is because they are using the same statistical test to make decisions. The reduction in IAP for the NIP-N version is quite similar to the reduction seen in the values of TIR for this version.

7. RELATED WORK

Mining and managing streaming data has received significant attention in recent years. Our work directly builds on top of Domingos and Hulten's work on decision tree construction on streaming data [10]. Srikant and Agrawal [28] and Toivonen [29] have focused on the use of sampling for mining frequent itemsets. More recently, Yang *et al.* have used sampling to reduce the number scans of the dataset [31]. Sampling based approaches have also been studied for efficiently constructing histograms [16, 8] and for estimating the number of distinct values of an attribute [18].

Before the current focus on streaming data, sampling has been applied for decision tree construction on large datasets. Carlett [6] used sequential samples called *peeholes* to split nodes. Musick *et al.* [26] empirically observed that the attribute gain distribution was close to normal distribution and used the digamma function to approximate it. Gratch's sequential ID3 [15] uses Multiple comparison Sequential Probability Ratio Test (McSPRT) for attribute selection. Gratch is also the first one to use delta method to describe the distribution of attribute gain. Our method is different in that we apply the normal distribution to compare the gain difference between two attributes. More recently, Chauchat and Rakotomalala [7] proposed using a statistical significance test to find the best split.

Our work is also related to the existing work on scalable decision tree construction, particularly, BOAT [11] and CLOUDS [2]. BOAT uses bootstrapping to construct an approximate tree on a fixed sized sample and then scans the entire dataset to build an exact decision tree. Some of our ideas on interval pruning are derived from this approach. CLOUDS [2] also partitions the range of a numerical attribute into intervals. However, it requires two passes on the entire data to partition nodes at one level of the tree, and further, does not guarantee the same best gain. In our recent work, we have applied interval pruning similar to the NIP approach presented in this paper for making a multi-pass parallel decision tree construction algorithm more communication efficient [22].

8. CONCLUSIONS AND FUTURE WORK

This paper has focused on a critical issue arising in decision tree construction on streaming data, i.e., the space and time efficiency.

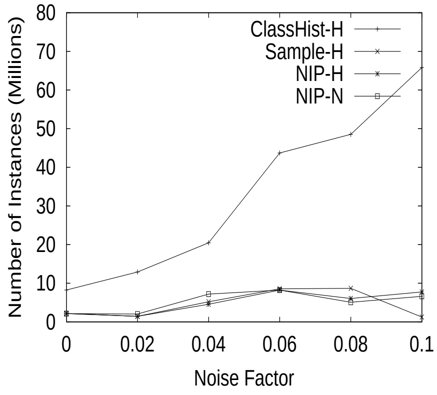

Figure 9: Total Instances Read (TIR)-F1

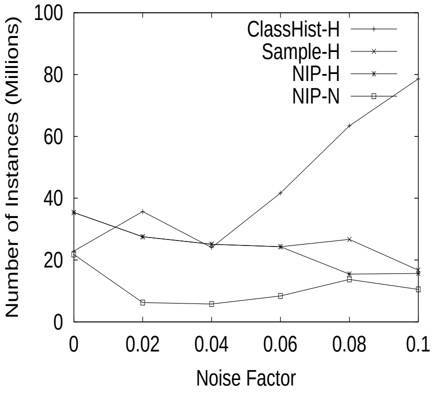

Figure 10: TIR: F6

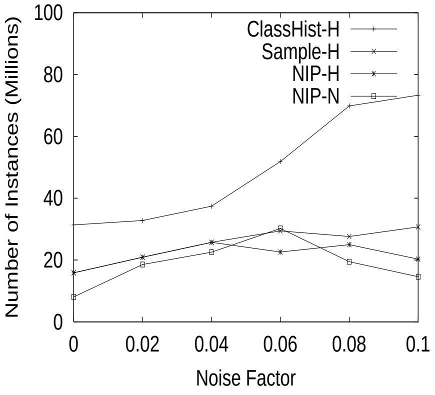

Figure 11: TIR: F7

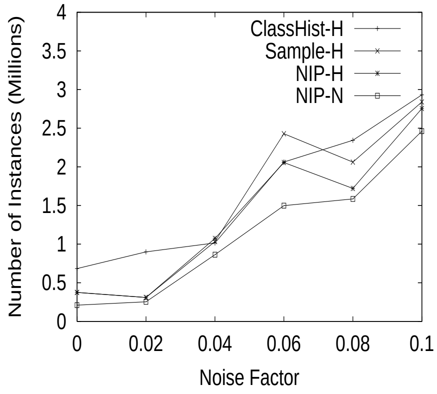

Figure 12: IAP (Instances Actively Processed)-F1

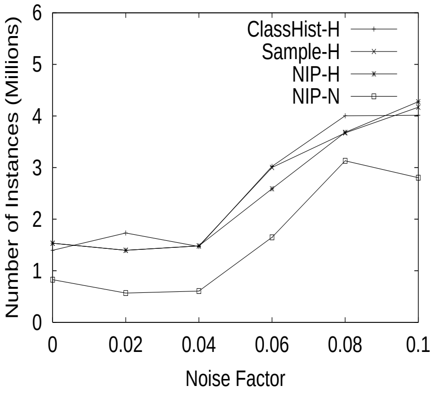

Figure 13: IAP: F6

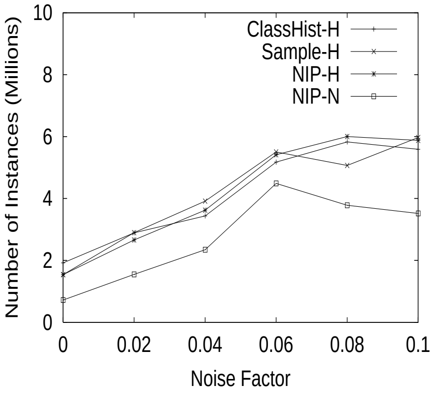

Figure 14: IAP: F7

This includes processing time per data instance, memory requirements (or the number of data instances required), and the total time required for constructing the decision tree. We have developed and evaluated two techniques, numerical interval pruning and exploiting the normal distribution property of the estimated value of the gain function.

In the future, we will like to expand our work in many directions. First, we want to consider other ways of creating intervals, besides the equal-width intervals we are currently using. Second, we want to extend our work to drifting data streams [21]. Another area will be to apply the ideas behind our normal test to other mining problems, such as k-means and EM clustering.

9. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Eng., 5(6):914-925,*, December 1993.

[2] K. Alsabti, S. Ranka, and V. Singh. Clouds: Classification for large or out-of-core datasets. http://www.cise.ufl.edu/ ranka/dm.html, 1998.

[3] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. In *Proc. of the 2002 ACM Symp. on Principles of Database Systems*. ACM Press, June 2002.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 2002 ACM Symposium on Principles of Database Systems (PODS 2002) (Invited Paper)*. ACM Press, June 2002.

[5] George Casella and Roger L. Berger. *Statistical Inference, 2nd. Edition*. DUXBURY Publishers, 2001.

[6] J. Catlett. *Megainduction: Machine Learning on Very Large Databases*. PhD thesis, Departement of Computer Science, University of Sydney, Sydney, Australia, 1991.

[7] Jean-Hugues Chauchat and Ricco Rakotomalala. Sampling strategy for building decision trees from very large databases comprising many continuous attributes. In *Instance Selection and Construction for Data Mining, edited by Huan Liu and Hiroshi Motoda*, pages 169–188. Kluwer Academic Publishers, 2002.

[8] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, 1998.

[9] A. Dobra, J. Gehrke, M. Garofalakis, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, June 2002.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the ACM Conference on Knowledge and Data Discovery (SIGKDD)*, 2000.

[11] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat– optimistic decision tree construction. In *Proc. of the ACM*

SIGMOD Conference on Management of Data, June 1999.

[12] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, pages 13–24. acmpress, June 2001.

[13] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. In *VLDB*, 1998.

[14] Phillip B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc. of the 2001 ACM Symp. on Parallel Algorithms and Architectures*, pages 281–291. ACM Press, August 2001.

[15] J. Gratch. Sequential inductive learning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 779–786, Portland, OR, 1996. AAAI Press.

[16] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of ACM Symp. on Theory of Computing (STOC)*, pages 471–475. ACM Press, 2001.

[17] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *Proceedings of 2000 Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 359–366. ACM Press, 2000.

[18] P. Haas, J. Naughton, P. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of the 1995 Intl. Conf. on Very Large Data Bases*, pages 311–322, August 1995.

[19] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, pages 58:18–30, 1963.

[20] Jason C. Hsu. *Multiple Comparisons, Theory and methods*. Chapman and Hall, 1996.

[21] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the ACM Conference on Knowledge and Data Discovery (SIGKDD)*, 2001.

[22] Ruoming Jin and Gagan Agrawal. Communication and Memory Efficient Parallel Decision Tree Construction. In *Proceedings of Third SIAM Conference on Data Mining*, May 2003.

[23] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):583–590, 1999.

[24] O. L. Mangasarian. *Nonlinear Programming. Classics in Applied Mathematics.* Society for Industrial and Applied Mathematics., 1994.

[25] M. Mehta, R. Agrawal, and J.Rissanen. Sliq: A fast scalable classifier for data mining. In *In Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, 1996.

[26] R. Musick, J. Catlett, and S. Russell. Decision theoretic subsampling for induction on large database. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 212–219, Amherst, MA, 1993. Morgan Kaufmann.

[27] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB)*, pages 544–555, September 1996.

[28] R. Srikant and R. Agrawal. Proc. of 22nd conference on very large databases (vldb). In *Mining generalized association rules*, pages 407–419, 1996.

[29] H. Toivonen. Sampling large databases for association rules. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pages 134–145, Mumbai, India, 1996. Morgan Kaufmann.

[30] S. Viglas and J. Naughton. Rate-based query optimization for streaming information sources. In *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, June 2002.

[31] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining long sequential patterns in a noisy environment. In *Proceedings of ACM SIGMOD*, Madison, Wisconsin, June 4-6, 2002.