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.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Streaming Data, Decision Tree, Sampling

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 [7, 9, 15, 16]. More recently, the database community has focused on a new model of data processing, in which data arrives in the form of continuous streams [2, 3, 5, 8]. 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 [10], or a probabilistic bound [6]. 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 [6, 13]. 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 [11] 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.

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 $t_i =< \vec{x}, c > \in \vec{X} \times C$. $\vec{x} \stackrel{\text{def}}{=} < x_1, ..., x_m >$ is the data associated

*This work was supported by NSF grant ACR-9982087, NSF CAREER award ACR-9733520, and NSF grant ACR-0130437.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA

with the instance and c is the class label. Each x_j is called a field or an attribute of the data instance. $\vec{X} \stackrel{\text{def}}{=} X_1 \times ...X_m$ 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.

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

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

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 two 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 D)
 global Tree root, Queue Q , AQ;
 local Node node;
 local Tuple t;
 Q ← NULL; AQ ← NULL;
 add(root, AQ);
 while not (empty(Q) and empty(AQ))
      t ← D.get();
      node ← classify(root, t);
      if node ∈ AQ
           add(node.sample, t);
           if node.satisfy_stop_condition
                remove(node, AQ);
           if node.enough_samples()
                use split function to get the best split;
                (node1, node2) ← node.create();
                remove(node, AQ);
                add((node1, node2), Q);
           while enough_memory(AQ, Q)
                get(node, Q);
                add(node, 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 [13], 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, as presented here, is only different from the work by Domingos and Hulten [6] 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.

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 [11] 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 [15, 16]. 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 [6]. 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 the numerical interval pruning approach, we 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.

```
NIP-Classifier(Node N, Stream D)
 while not satisfy_stop_condition(N)
      { * Get Some Samples from Stream D *}
      Sample S ← (D.get());
      Update_Small_ClassHist(S);
      Update_Concise_ClassHist(S);
      Update_Detailed_Information(S);
      { * Find the best gain *}
      g' ← Find_Best_Gain(ClassHist);
      ḡ ← UnPruning(g', Concise_ClassHist);
      { * Split *}
      if Statistically_Best_Gain(ḡ)
           Split_Node(N);
           break;
      { * Pruning *}
      Pruning(ḡ, 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.

Further details of the above approach are available in a technical report from the authors [14].

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.*

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 [11] based approach used by Domingos and Hulten [6], 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 [4]. 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 $X_1, \ldots, X_n$ be a random sample. Let $X_i = X_{1i}, \ldots, X_{pi}$. Further, let $E(X_{ij}) = \mu_i$ and $Cov(X_{ij}, X_{jk}) = \sigma_{ij}$. Let $\overline{X}_i$ be the mean of $X_{i1}, X_{i2}, \ldots, X_{in}$ and let $\vec{\mu} = (\mu_1, \ldots, \mu_p)$. For a given function g with continuous first partial derivatives, we have*

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

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 [4], 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 1. *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 1, 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 2. *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 [12]. □

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 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 [9, 7, 15, 16]. 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.

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.

For the remaining part of this section, we only report results from the use of function 6. Results from functions 1 and 7 are available in a technical report [14].

Figure 6 shows the execution times for decision tree construction with the four versions and different levels of noise, for function 6. 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`. 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%.

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 cur-

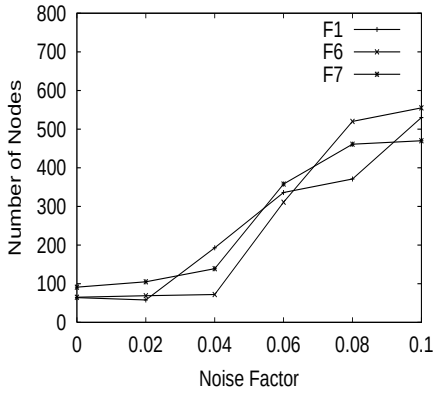

Figure 3: Concept Size

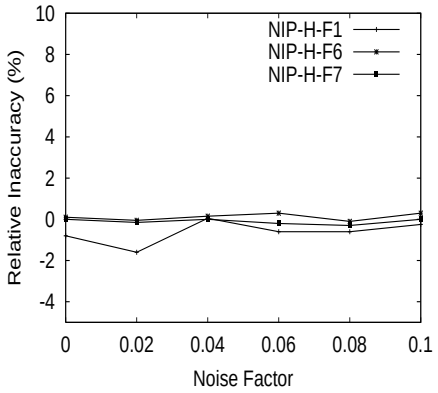

Figure 4: Inaccuracy with NIP

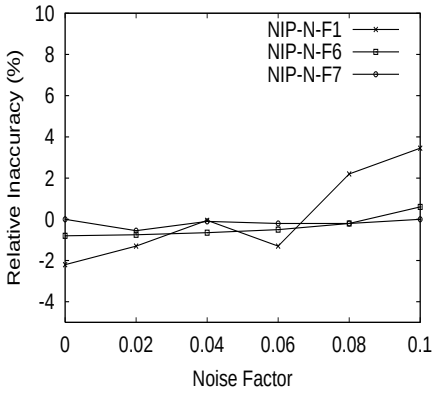

Figure 5: Inaccuracy with Normal

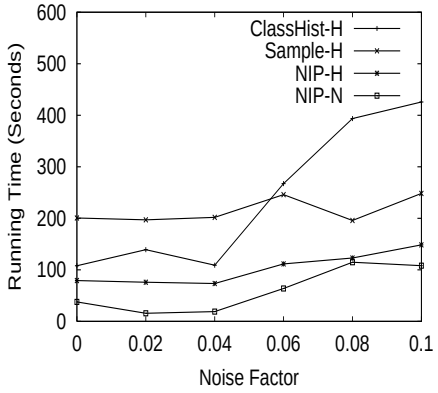

Figure 6: Running Time: F6

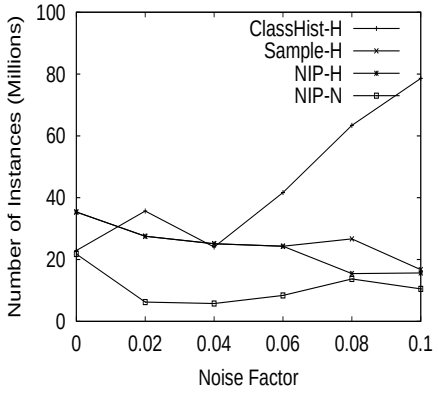

Figure 7: TIR: F6

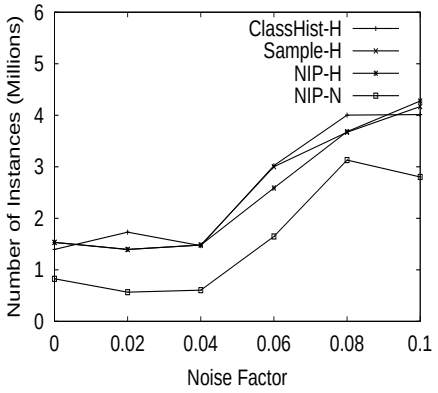

Figure 8: IAP: F6

rently because of memory considerations. Therefore, we measure IAP as the number of data instances that were used for evaluating candidate split conditions. Figure 7 shows TIR for the four versions and for the function 6. The use of class histograms results in high memory requirements, which 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 18% and 60% for function 6. Figure 8 shows the values of IAP. 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. 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. 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 [13]. Another area will be to apply the ideas behind our normal test to other mining problems, such as k-means and EM clustering.

8. 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] 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.

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

[13] 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.

[14] Ruoming Jin and Gagan Agrawal. Efficient Decision Tree Construction on Streaming Data. Technical Report OSU-CISRC-6/03-TR34, Department of Computer and Information Sciences, The Ohio State University, June 2003.

[15] 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.

[16] 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.