Journal of Machine Learning Research 11 (2010) 849-872 Submitted 2/09; Revised 12/09; Published 2/10

A Streaming Parallel Decision Tree Algorithm

Yael Ben-Haim YAELBH@IL.IBM.COM
Elad Tom-Tov YOMTOV@IL.IBM.COM
IBM Haifa Research Lab
Haifa University Campus
Mount Carmel, Haifa 31905, ISRAEL

Editor: Soeren Sonnenburg

Abstract

We propose a new algorithm for building decision tree classifiers. The algorithm is executed in a distributed environment and is especially designed for classifying large data sets and streaming data. It is empirically shown to be as accurate as a standard decision tree classifier, while being scalable for processing of streaming data on multiple processors. These findings are supported by a rigorous analysis of the algorithm's accuracy.

The essence of the algorithm is to quickly construct histograms at the processors, which compress the data to a fixed amount of memory. A master processor uses this information to find near-optimal split points to terminal tree nodes. Our analysis shows that guarantees on the local accuracy of split points imply guarantees on the overall tree accuracy.

Keywords: decision tree classifiers, distributed computing, streaming data, scalability

1. Introduction

We propose a new algorithm for building decision tree classifiers for classifying both large data sets and streaming data. As recently noted (Bottou and Bousquet, 2008), the challenge which distinguishes large-scale learning from small-scale learning is that training time is limited compared to the amount of available data. Thus, in our algorithm both training and testing are executed in a distributed environment, using only one pass on the data. We refer to the new algorithm as the Streaming Parallel Decision Tree (SPDT).

Decision trees are simple yet effective classification algorithms. One of their main advantages is that they provide human-readable rules of classification. Decision trees have several drawbacks, one of which is the need to sort all numerical attributes in order to decide where to split a node. This becomes costly in terms of running time and memory size, especially when decision trees are trained on large data. The various techniques for handling large data can be roughly grouped into two approaches: performing pre-sorting of the data, as in SLIQ (Mehta et al., 1996) and its successors SPRINT (Shafer et al., 1996) and ScalParC (Joshi et al., 1998), or replacing sorting with approximate representations of the data such as sampling and/or histogram building, for example, BOAT (Gehrke et al., 1999), CLOUDS (AlSabti et al., 1998), and SPIES (Jin and Agrawal, 2003). While pre-sorting techniques are more accurate, they cannot accommodate very large data sets or streaming data.

Faced with the challenge of handling large data, a large body of work has been dedicated to parallel decision tree algorithms (Shafer et al., 1996; Joshi et al., 1998; Narlikar, 1998; Jin and Agrawal,

2003; Srivastava et al., 1999; Sreenivas et al., 1999; Goil and Choudhary, 1999). There are several ways to parallelize decision trees, described in detail in Amado et al. (2001), Srivastava et al. (1999) and Narlikar (1998). Horizontal parallelism partitions the data so that different processors see different examples.[1] Vertical parallelism enables different processors to see different attributes. Task parallelism distributes the tree nodes among the processors. Finally, hybrid parallelism combines horizontal or vertical parallelism in the first stages of tree construction with task parallelism towards the end.

Like their serial counterparts, parallel decision trees overcome the sorting obstacle by applying pre-sorting, distributed sorting, and approximations. Following our interest in streaming data, we focus on approximate algorithms. Our proposed algorithm builds the decision tree in a breadth-first mode, using horizontal parallelism. The core of our algorithm is an on-line method for building histograms from streaming data at the processors. The histograms are essentially compressed representations of the data, so that each processor can transmit an approximate description of the data that it sees to a master processor, with low communication complexity. The master processor integrates the information received from all the processors and determines which terminal nodes to split and how.

This paper is organized as follows. In Section 2 we introduce the SPDT algorithm and the underlying histogram building algorithm. We dwell upon the advantages of SPDT over existing algorithms. In Section 3 we analyze the tree accuracy. In Section 4 we present experiments that compare the SPDT algorithm with the standard decision tree. The experiments show that the SPDT algorithm compares favorably with the traditional, single-processor algorithm. Moreover, it is scalable to streaming data and multiple processors. We conclude in Section 5.

2. Algorithm Description

Consider the following problem: given a (possibly infinite) series of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, c\}$, our goal is to construct a decision tree that will accurately classify test examples. The classifier is built using multiple processing nodes (i.e., CPUs), where each of the processing nodes observes approximately $1/W$ of the training examples (where W is the number of processing nodes). This partitioning happens for one of several reasons: for example, the data may not be stored in a single location, and may not arrive at a single location, or it may be too abundant to be handled by a single node in a timely manner.

Because of the large number of training examples, it is not feasible to store the examples (even in each separate processor). Therefore, a processor can either save a short buffer of examples and use them to improve (or construct) the classifier, or build a representative summary statistic from the examples, improving it over time, but never saving the examples themselves. In this paper we take the latter approach.

Although the setting described here is generally applicable to streams of data, it is also applicable to the classification of large data sets in batch mode, where memory and processing power constraints require the distribution of data across multiple processors and with limited memory for each processor.

We first present our histogram data structure and the methods related to it. We then describe the tree building process.

1. We refer to processing nodes as processors, to avoid confusion with tree nodes.

Algorithm 1 Update Procedure

input A histogram $h = \{(p_1, m_1), \ldots, (p_B, m_B)\}$, a point p.

output A histogram with B bins that represents the set $S \cup \{p\}$, where S is the set represented by h.

1: **if** $p = p_i$ for some i **then**
2: $m_i = m_i + 1$
3: **else**
4: Add the bin $(p, 1)$ to the histogram, resulting in a histogram of $B+1$ bins $h \cup \{(p,1)\}$. Denote $p_{B+1} = p$ and $m_{B+1} = 1$.
5: Sort the sequence $p_1, \ldots, p_{B+1}$. Denote by $q_1, \ldots, q_{B+1}$ the sorted sequence, and let π be a permutation on $1, \ldots, B+1$ such that $q_i = p_{\pi(i)}$ for all $i = 1, \ldots, B+1$. Denote $k_i = m_{\pi(i)}$, namely, the histogram $h \cup (p,1)$ is equivalent to $(q_1, k_1), \ldots, (q_{B+1}, k_{B+1})$, $q_1 < \ldots < q_{B+1}$.
6: Find a point q_i that minimizes $q_{i+1} - q_i$.
7: Replace the bins (q_i, k_i), (q_{i+1}, k_{i+1}) by the bin

$$\left(\frac{q_i k_i + q_{i+1} k_{i+1}}{k_i + k_{i+1}}, k_i + k_{i+1} \right).$$

8: **end if**

2.1 On-line Histogram Building

A histogram is a set of B pairs (called bins) of real numbers $\{(p_1, m_1), \ldots, (p_B, m_B)\}$, where B is a preset constant integer. The histogram is a compressed and approximate representation of a set S of real numbers. At any time we have $|S| = \sum_{i=1}^{B} m_i$, where $|S|$ is the number of points in S. The histogram data structure supports four procedures, named `update`, `merge`, `sum`, and `uniform`. The `update` procedure is based on an on-line clustering algorithm developed by Guedalia et al. (1999). A demonstration of the algorithms on actual input is given in the appendix.

Algorithm 1 presents the `update` procedure, which adds a new point to a set that is already represented by a given histogram. The `merge` procedure (Algorithm 2) creates a histogram that represents the union $S_1 \cup S_2$ of the sets S_1, S_2, whose representing histograms are given. The algorithm is similar to the update algorithm; in the first step, the two histograms form a single histogram with many bins. In the second step, bins which are closest are merged together (as in lines 5 and 6 in Algorithm 1) to form a single bin. The process repeats until the histogram has B bins.

The `sum` procedure estimates the number of points in a given interval $[a, b]$, that belong to a set whose histogram is given. Algorithm 3 describes how to calculate the sum for $[-\infty, b]$, and can be used to calculate the sum for $[a, b]$, since it is equal to the sum for $[-\infty, b]$ minus the sum for $[-\infty, a]$.

The algorithm assumes that for a bin (p, m), there are m points surrounding p, of which $m/2$ points are to the left of the bin and $m/2$ points are to the right. Consequently, the number of points in the interval $[p_i, p_{i+1}]$ is equal to $(m_i + m_{i+1})/2$, which is the area of the trapezoid $(p_i, 0), (p_i, m_i), (p_{i+1}, m_{i+1}), (p_{i+1}, 0)$, divided by $(p_{i+1} - p_i)$. To estimate the number of points in the interval $[p_i, b]$, for $p_i < b < p_{i+1}$, we draw a straight line from (p_i, m_i) to (p_{i+1}, m_{i+1}). We set $m_b = m_i + \frac{m_{i+1} - m_i}{p_{i+1} - p_i}(b - p_i)$, so that (b, m_b) is on this line. The estimated number of points in the interval $[p_i, b]$ is then the area of the trapezoid $(p_i, 0), (p_i, m_i), (b, m_b), (b, 0)$, divided again by $(p_{i+1} - p_i)$. The case where $b < p_1$ or $b > p_B$ requires special treatment. One possibility is to add two dummy bins $(p_0, 0)$ and $(p_{B+1}, 0)$, where p_0 and p_{B+1} are chosen using prior knowledge, according to which all

Algorithm 2 Merge Procedure

input Histograms $h_1 = \{(p_1^{(1)}, m_1^{(1)}), \ldots, (p_{B_1}^{(1)}, m_{B_1}^{(1)})\}$, $h_2 = \{(p_1^{(2)}, m_1^{(2)}), \ldots, (p_{B_2}^{(2)}, m_{B_2}^{(2)})\}$, an integer B.

output A histogram with B bins that represents the set $S_1 \cup S_2$, where S_1 and S_2 are the sets represented by h_1 and h_2, respectively.

1: For $i = 1, \ldots, B_1$, denote $p_i = p_i^{(1)}$ and $m_i = m_i^{(1)}$. For $i = 1, \ldots, B_2$, denote $p_{B_1+i} = p_i^{(2)}$ and $m_{B_1+i} = m_i^{(2)}$.

2: Sort the sequence $p_1, \ldots, p_{B_1+B_2}$. Denote by $q_1, \ldots, q_{B_1+B_2}$ the sorted sequence, and let π be a permutation on $1, \ldots, B_1+B_2$ such that $q_i = p_{\pi(i)}$ for all $i = 1, \ldots, B_1+B_2$. Denote $k_i = m_{\pi(i)}$, namely, the histogram $h_1 \cup h_2$ is equivalent to $(q_1, k_1), \ldots, (q_{B_1+B_2}, k_{B_1+B_2})$, $q_1 < \ldots < q_{B_1+B_2}$.

3: **repeat**

4: Find a point q_i that minimizes $q_{i+1} - q_i$.

5: Replace the bins (q_i, k_i), (q_{i+1}, k_{i+1}) by the bin

$$\left(\frac{q_i k_i + q_{i+1} k_{i+1}}{k_i + k_{i+1}}, k_i + k_{i+1} \right).$$

6: **until** The histogram has B bins

Algorithm 3 Sum Procedure

input A histogram $\{(p_1, m_1), \ldots, (p_B, m_B)\}$, a point b such that $p_1 < b < p_B$.

output Estimated number of points in the interval $[-\infty, b]$.

1: Find i such that $p_i \leq b < p_{i+1}$.

2: Set

$$s = \frac{m_i + m_b}{2} \cdot \frac{b - p_i}{p_{i+1} - p_i}$$

where

$$m_b = m_i + \frac{m_{i+1} - m_i}{p_{i+1} - p_i}(b - p_i).$$

3: **for all** $j < i$ **do**

4: $s = s + m_j$

5: **end for**

6: $s = s + m_i/2$

or almost all the points in S are in the interval $[p_0, p_{B+1}]$ (p_0 and p_{B+1} can be determined on the fly during the histogram's construction).

The `uniform` (Algorithm 4) procedure receives as input a histogram $\{(p_1, m_1), \ldots, (p_B, m_B)\}$ and an integer $\tilde{B}$ and outputs a set of real numbers $u_1 < \ldots < u_{\tilde{B}-1}$, with the property that the number of points between two consecutive numbers u_j, u_{j+1}, and the number of data points to the left of u_1 and to the right of $u_{\tilde{B}-1}$, is $\frac{|S|}{\tilde{B}}$. The algorithm works like the `sum` procedure in the inverse direction: After the point u_j was determined, we analytically find a point u_{j+1} such that the number of points in $[u_j, u_{j+1}]$ is estimated to be equal to $\frac{|S|}{\tilde{B}}$. This is very similar to the calculations performed in

Algorithm 4 Uniform Procedure

input A histogram $\{(p_1,m_1),\ldots,(p_B,m_B)\}$, an integer $\tilde{B}$.

output A set of real numbers $u_1 < \ldots < u_{\tilde{B}}$, with the property that the number of points between two consecutive numbers u_j, u_{j+1}, as well as the number of data points to the left of u_1 and to the right of $u_{\tilde{B}}$, is $\frac{1}{\tilde{B}}\sum_{i=1}^{B} m_i$.

1: **for all** $j = 1,\ldots,\tilde{B}-1$ **do**

2: Set $s = \frac{j}{\tilde{B}}\sum_{i=1}^{B} m_i$

3: Find i such that $\mathtt{sum}([-\infty,p_i]) < s < \mathtt{sum}([-\infty,p_{i+1}])$.

4: Set d to be the difference between s and $\mathtt{sum}([-\infty,p_i])$.

5: Search for u_j such that

$$d = \frac{m_i + m_{u_j}}{2} \cdot \frac{u_j - p_i}{p_{i+1} - p_i},$$

where

$$m_{u_j} = m_i + \frac{m_{i+1} - m_i}{p_{i+1} - p_i}(u_j - p_i).$$

Substituting

$$z = \frac{u_j - p_i}{p_{i+1} - p_i},$$

we obtain a quadratic equation $az^2 + bz + c = 0$ with $a = m_{i+1} - m_i$, $b = 2m_i$, and $c = -2d$. Hence set $u_j = p_i + (p_{i+1} - p_i)z$, where

$$z = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

6: **end for**

sum, where this time we are given the area of a trapezoid and have to compute the coordinates of its vertices (see line 5 in Algorithm 4).

2.2 Tree Growing Algorithm

We construct a decision tree based on a set of training examples $\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n)\}$, where $\mathbf{x}_1,\ldots,\mathbf{x}_n \in \mathsf{R}^d$ are the feature vectors and $y_1,\ldots,y_n \in \{1,\ldots,c\}$ are the labels. Every internal node in the tree possesses two ordered child nodes and a decision rule of the form $\mathbf{x}^{(i)} < a$, where $\mathbf{x}^{(i)}$ is the ith attribute and a is a real number. Feature vectors that satisfy the decision rule are directed to the node's left child node, and the other vectors are directed to the right child node. Thus, every example $\mathbf{x}$ has a path from the root to one of the leaves, denoted $l(\mathbf{x})$. Every leaf has a label t, so that an example $\mathbf{x}$ is assigned the label $t(l(\mathbf{x}))$.

Algorithm 5 provides an overview of the tree construction algorithm. We note that this description fits standard decision trees as well. Each time that line 3 is executed, we say that a new iteration has begun. If there are too many samples (possibly infinite in number), we read a predefined number of samples; otherwise, we use the complete data set. A new level of nodes is appended to the tree in each iteration. In line 5 we decide whether a leaf v is to be split or labeled, according to a stopping criterion. Possible stopping criteria can be some threshold on the number of samples reaching the node, or on the node's impurity. A node's impurity is a function G that measures the homogeneity

Algorithm 5 Decision Tree

input Training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

1: Initialize T to be a single unlabeled node.
2: **while** there are unlabeled leaves in T **do**
3: Navigate data samples to their corresponding leaves.
4: **for all** unlabeled leaves v in T **do**
5: **if** v satisfies the stopping criterion **or** there are no samples reaching v **then**
6: Label v with the most frequent label among the samples reaching v
7: **else**
8: Choose candidate splits for v and estimate Δ for each of them.
9: Split v with the highest estimated Δ among all possible candidate splits.
10: **end if**
11: **end for**
12: **end while**

of labels in samples reaching the node. Its parameters are $q_1, \ldots, q_c$, where q_j is the probability that a sample reaching v has label j and c is the number of labels. The most popular impurity functions are the Gini criterion,

$$1 - \sum_j q_j^2$$

and the entropy function

$$-\sum_j q_j \ln q_j \text{ where } 0 \ln 0 \equiv 0 \, .$$

In our analysis in Section 3, we require G to be continuous and satisfy $G(\{q_j\}) \geq 1 - \max_j \{q_j\}$. These properties hold for the Gini and entropy functions.

The notation Δ, appearing in lines 8 and 9, represents the gap in the impurity function before and after splitting. Suppose that an attribute i and a threshold a are chosen, so that a node v is split according to the rule $\mathbf{x}^{(i)} < a$. Denote by τ the probability that a sample reaching v is directed to v's left child node. Denote further by $q_{L,j}$ and $q_{R,j}$ the probabilities of label j in the left and right child nodes, respectively. We define the function $\Delta(\tau, \{q_j\}, \{q_{L,j}\}, \{q_{R,j}\}) = \Delta(v, i, a)$ as

$$\Delta = G(\{q_j\}) - \tau G(\{q_{L,j}\}) - (1 - \tau) G(\{q_{R,j}\}). \tag{1}$$

To complete the algorithm's description, we need to specify what are the candidate splits, mentioned in lines 8 and 9, and how the function Δ for each split is estimated in a distributed environment. We begin by providing an interpretation for these notions in the classical setting, that is, for the standard, serial algorithm. Most algorithms sort every attribute in the training set, and test splits of the form $\mathbf{x}^{(i)} < \frac{a+b}{2}$, where a and b are two consecutive numbers in the sorted sequence of the ith attribute. For every candidate split, Δ can be calculated precisely, as in (1).

In the parallel setting, we apply a distributed architecture that consists of W processors (also called workers). Each processor can observe $1/W$ of the data, but has a view of the complete classification tree built so far. We do not wish each processor to sort its share of the data set, because this operation is not scalable to extremely large data sets. Moreover, the communication complexity between the processors must be a constant that does not depend on the size of the data set. Our algorithm addresses these issues by trading time and communication complexity with

Algorithm 6 Compress Data Sets

input $1/W$ of the training set, where W is the number of processors
output histograms to be transmitted to the master processor

```
Initialize an empty histogram h(v,i,j) for every unlabeled leaf v, attribute i, and class j.
for all observed training samples (x_k, y_k), where x_k = (x_k^(1), ... x_k^(d)) do
    if the sample is directed to an unlabeled leaf v then
        for all attributes i do
            Update the histogram h(v,i,y_k) with the point x_k^(i), using the update procedure.
        end for
    end if
end for
```

classification accuracy. The processors build histograms describing the data they observed and send them to a master processor. Algorithm 6 specifies which histograms are built and how. The number of bins in the histograms is specified through a trade-off between accuracy and computational load: A large number of bins allows a more accurate description, whereas small histograms are beneficial for avoiding time, memory, and communications overloads.

For every unlabeled leaf v, attribute i, and class j, the master processor merges the W histograms $h(v,i,j)$ received from the processors. The master node now has an exact knowledge of the frequency of each label in each tree node, and hence the ability to calculate the impurity of all unlabeled leaves. Leaves that satisfy the stopping criterion are labeled. For the other leaves, the questions remain of how to choose candidate splits and how to estimate their Δ. They are answered as follows. Let v be an unlabeled leaf (that remains unlabeled after the application of the stopping criterion) and let i be an attribute. We first merge the histograms $h(v,i,1),\ldots,h(v,i,c)$ (c denotes the number of labels). The new histogram, denoted $h(v,i)$, represents the ith dimension of feature vectors that reach v, with no distinction between vectors of different labels. We now apply the `uniform` procedure on $h(v,i)$ with some chosen $\tilde{B}$. The resulting set $u_1 < \ldots < u_{\tilde{B}-1}$ constitutes the locations of the candidate splits for the ith attribute. Finally, Δ for each candidate split is estimated using the `sum` procedure and the histograms $h(v,i,j)$. We clarify the rationale behind this choice of split locations. Suppose that the best split is $\mathbf{x}^{(i)} < a$, where $u_k < a < u_{k+1}$. The number of points in the interval $[u_k, a]$ is bounded, implying a bound on the degree of change in Δ if one splits at u_k instead of a. This issue is discussed in more detail in Section 3.

Decision trees are frequently pruned during or after training to obtain smaller trees and better generalization. In the experiments presented in Section 4, we adapted the MDL-based pruning algorithm of Mehta et al. (1996), which is similar to the one used in CART (Breiman et al., 1984). This algorithm involves simple calculations during node splitting that reflect the node's purity. In a bottom-up pass on the complete tree, some subtrees are chosen to be pruned, based on estimates of the expected error rate before and after pruning. The distributed environment neither changes this pruning algorithm nor does it affect its output.

2.3 Complexity Analysis

Every iteration consists of an updating phase performed simultaneously by all the processors and a merging phase performed by the master processor. In the update phase, every processor makes one pass on the data batch assigned to it. The only memory allocation is for the histograms being

constructed. The number of bins in the histograms is constant; hence, operations on histograms take a constant amount of time. Every processor performs at most N/W histogram updates, where N is the size of the data batch and W is the number of processors. There are $W \times L \times c \times d$ histograms, where L is the number of leaves in the current iteration, c is the number of labels, and d is the number of attributes. Assuming that W, L, c, and d are all independent of N, it follows that the space complexity is $O(1)$. The histograms are communicated to the master processor, which merges them and applies the `sum` and `uniform` procedures. If the `uniform` procedure is applied with a constant parameter $\tilde{B}$, then the time complexity of the merging phase is $O(1)$.

To summarize, each iteration requires the following:

- At most N/W operations by each processor in the updating phase.
- Constant space and communication complexities.
- Constant time in the merging phase.

2.4 Related Work

In this section we discuss previous work on histogram and quantile approximations, as well as procedures for building decision trees on parallel platforms.

2.4.1 HISTOGRAMS AND QUANTILES APPROXIMATIONS

Data structures that summarize large sets are substantial components of a variety of algorithms in database management and data mining. Our histogram algorithms tackle two related problems: data compression and quantile approximations.[2] There is broad coverage of these topics in the literature, with an inclination towards one pass algorithms, see Gilbert et al. (2002), Guha et al. (2006), Ioannidis (2003) and Lin (2007) and references therein. Proposed solutions can be divided into two categories: The first category consists of algorithms with proven approximation guarantees (Cormode and Muthukrishnan, 2005; Gilbert et al., 2002; Greenwald and Khanna, 2001; Guha et al., 2006). The demand for a guaranteed accuracy level forces these algorithms to use large amounts of memory, that is, their space requirements are increasing functions of the data size. An exception is the probabilistic algorithm of Manku et al. (1998), which receives an input parameter δ and returns approximate quantiles whose guarantees hold with probability δ. The space complexity of this algorithm increases with δ but not with the data size. The second category, to which our algorithm belongs, consists of heuristics that work well empirically and demand low amounts of space, but lack any rigorous accuracy analysis (Agrawal and Swami, 1995; Jain and Chlamtac, 1985). To our knowledge, distributed environments are not addressed in either of the two categories, except for a brief mention by Manku et al. (1998).

Guaranteed accuracy at the cost of non-constant memory and increasing processing time are problematic because of the inherent nature of streaming data. For example, the algorithm proposed by Guha et al. (2006) requires roughly $O(B^2 \log n)$ memory, where n is the number of data points and B the number of bins. Thus, for example, a stream of 10^{10} data points (not a large number in today's data environments) requires more than 20 times the memory of a comparable fixed-memory algorithm.

2. For a sequence S of real numbers, the ϕ-quantile, $0 \leq \phi \leq 1$, is defined to be an element $x \in S$ such that $\lceil \phi |S| \rceil$ elements of S are smaller or equal to x.

The use of a fixed-memory algorithm, like the one proposed in this paper, naturally comes at a cost in accuracy. As we show, when the data distribution is highly skewed, the accuracy of the on-line histogram decays. Therefore, in cases where the data can be assumed to have originated in categorical distributions with a limited number of values or in distributions which are not highly skewed, the proposed algorithm is sufficiently accurate. In other cases, where distributions are known to be highly skewed, or memory sizes are not a major factor when executing the algorithm, practitioners may prefer to resort to guaranteed accuracy algorithms. This replaces the first part of the proposed algorithm, but keeps its higher levels intact.

2.4.2 PARALLEL DECISION TREES

The SPIES (Jin and Agrawal, 2003) and pCLOUDS (Sreenivas et al., 1999) algorithms build decision trees for streaming data and work in a distributed environment. They are similar to the SPDT algorithm in that they use histograms to process the data in constant time and memory. There are, however, three major differences between these algorithms and the SPDT algorithm and its analysis. The first difference is in the histogram building algorithm. Unlike SPDT, both SPIES and pCLOUDS sample the data. The second difference is in the need of a second pass. CLOUDS (AlSabti et al., 1998) has two versions, named SS and SSE.[3] SSE and SPIES may require several passes over the data, and therefore hold each data batch in memory. The purpose of the second pass is to locate exactly the best split location for every node, and hence eventually to construct the same tree as the standard algorithm. SS is more similar to SPDT, since both algorithms build histograms with an equal number of points in each bin and take the boundaries of the histograms to be the candidate splits. Since only a constant number of split locations is checked, it is possible that a suboptimal split is chosen, which may cause the entire tree to be different from the one constructed by the standard algorithm. The third difference between our work and previous works is our ability to analytically show that the error rate of the parallel tree approaches the error rate of the serial tree, even though the trees are not identical.

3. Bounding the Error of SPDT

In this section, we investigate the training error rate of SPDT. We adopt a simpler version of the framework set by Kearns and Mansour (1999), which views tree nodes as weak learners. This approach allows us to obtain an overall estimate of the tree by studying the local improvements in classification accuracy induced by the internal nodes.

3.1 Background

Let n be the number of training samples used to train a decision tree T. For a tree node v, denote by n_v the number of training samples that reach v, and by $q_{v,j}$ the probability that a sample reaching v has label j, for $j = 1, \ldots, c$. The training error rate of T is

$$e_T = \frac{1}{n} \sum_{v \text{ leaf in } T} n_v (1 - \max_j \{q_{v,j}\}).$$

3. pCLOUDS is a parallelization of the SSE version of CLOUDS. We mention the SS version as well because it can be similarly parallelized.

Henceforth, we require that the impurity function G is continuous and satisfies $G(\{q_j\}) \geq 1 - \max_j\{q_j\}$. The last inequality implies that we have $e_T \leq G_T$, where

$$G_T = \frac{1}{n} \sum_{v \text{ leaf in } T} n_v G(\{q_{v,j}\}). \tag{2}$$

For our analysis, we rewrite Algorithm 5 such that only one new leaf is added to the tree in each iteration (see Algorithm 7). The resulting full-grown tree is identical to the tree constructed by Algorithm 5. Let T_t be the tree produced by Algorithm 7 after the tth iteration. Suppose that the node v is split in the tth iteration and assigned the rule $\mathbf{x}^{(i)} < a$, and let v_L, v_R denote its left and right child nodes respectively. Then

$$\begin{aligned} &G_{T_{t-1}} - G_{T_t} \\ &= \frac{1}{n}(n_v G(\{q_{v,j}\}) - n_{v_L} G(\{q_{v_L,j}\}) - n_{v_R} G(\{q_{v_R,j}\})) \\ &= \frac{n_v}{n}\Delta(v,i,a). \end{aligned}$$

It follows that a lower bound on $\Delta(v,i,a)$ yields an upper bound on G_{T_t} and hence also on e_{T_t}.

Definition 1 *An internal node v, split by a rule $\mathbf{x}^{(i)} < a$, is said to perform locally well with respect to a function $f(\{q_j\})$ if it satisfies $\Delta(v,i,a) \geq f(\{q_{v,j}\})$. A tree T is said to perform locally well if every internal node v in it performs locally well. Finally, a decision tree building algorithm performs locally well if for every training set, the output tree performs locally well.*

Suppose that T_{t-1} has a leaf for which $\frac{n_v}{n} f(\{q_{v,j}\})$ can be lower-bounded by a quantity $h(t, G_{T_{t-1}})$ that depends only on t and $G_{T_{t-1}}$. Then a lower bound on the training error rate of an algorithm that performs locally well can be derived by solving the recurrence $G_{T_t} \leq G_{T_{t-1}} - h(t, G_{T_{t-1}})$. As a simple example, consider $f(\{q_j\}) = \alpha G(\{q_j\})$ for some positive constant α. By (2), and since the number of leaves in T_{t-1} is t, there exists a leaf v in T_{t-1} for which $\frac{n_v}{n} G(\{q_{v,j}\}) \geq G_{T_{t-1}}/t$, hence $\frac{n_v}{n} f(\{q_{v,j}\}) \geq \frac{\alpha}{t} G_{T_{t-1}}$. Let $\tilde{v}$ be the node which is split in the tth iteration. By definition (see line 10 in Algorithm 7), $\frac{n_{\tilde{v}}}{n}\Delta_{\tilde{v}} \geq \frac{n_v}{n}\Delta_v$, where Δ_v and $\Delta_{\tilde{v}}$ are the best splits for v and $\tilde{v}$. We have

$$G_{T_{t-1}} - G_{T_t} = \frac{n_{\tilde{v}}}{n}\Delta_{\tilde{v}} \geq \frac{n_v}{n}\Delta_v \geq \frac{n_v}{n} f(\{q_{v,j}\}) \geq \frac{\alpha}{t} G_{T_{t-1}}.$$

Let G_0 be an upper bound on G_{T_0}. Solving the recurrence $G_{T_t} \leq (1 - \alpha/t) G_{T_{t-1}}$ with initial value G_0, we obtain $G_{T_t} \leq G_0 (t-1)^{-\alpha/2}$, therefore $e_{T_t} \leq G_0 (t-1)^{-\alpha/2}$.

Kearns and Mansour (1999) made a stronger assumption, named the Weak Hypothesis Assumption, on the local performance of tree nodes. For binary classification and a finite feature space, it is shown that if $G(q_1, q_2)$ is the Gini index, the entropy function, or $G(q_1, q_2) = \sqrt{q_1 q_2}$, then the Weak Hypothesis Assumption implies good local performance (each splitting criterion with respect to its own $f(q_1, q_2)$). Lower bounds on the training error of trees with these splitting criteria are then derived, as described above. These bounds are subject to the validity of the Weak Hypothesis Assumption.

3.2 Main Result

To build an SPDT, we have to set the parameters B and $\tilde{B}$. Recall that B is the number of bins in the histograms constructed by the processors, and $\tilde{B}$ is the size of the output of `uniform`. Encouraged by empirical results concerning the histograms' accuracy, (see Section 4), we set $B = \tilde{B}$ and assume that all applications of the `uniform` and `sum` procedures during SPDT runtime provide us with exact information on the data set. For example, it is assumed that Δ is calculated exactly and not only "estimated" (see line 9 in Algorithm 5). We note that all our results remain intact also if we allow the calculations to be somewhat biased (the empirical evidence points to a bias of about 5%).

It follows that the only source for sub-optimality with respect to standard decision trees is in the choice of the candidate splits. We recall that for the standard decision tree, the number of candidate splits for a node v is equal to the number of training samples that reach v minus one. This luxury is out of the reach of the SPDT because of scalability requirements. The SPDT thus must test only a constant number of candidate splits before it announces the winning split. The following theorem asserts that Δ for the split chosen by the SPDT algorithm can be arbitrarily close to the optimal Δ (of the split chosen by the standard algorithm). The number of bins depends on how close to the real Δ we wish to be, and also on the shape of the training set, but not on its size.

Theorem 2 *Assume that the functions operating on histograms return exact answers. Let v be a leaf in a decision tree which is under construction, and let $\mathbf{x}^{(i)} < a$ be the best split for v according to the standard algorithm. Denote $\tau, q_j, q_{L,j}, q_{R,j}$ as in Section 2.2. Then for every $\delta > 0$ there exists B that depends on $\tau, \{q_j\}, \{q_{L,j}\}, \{q_{R,j}\}$, and δ, such that the split $\mathbf{x}^{(\tilde{i})} < \tilde{a}$ chosen by the SPDT algorithm with B bins satisfies $\Delta(v, \tilde{i}, \tilde{a}) \geq \Delta(v, i, a) - \delta$.*

Proof. Fix B and consider the split $\mathbf{x}^{(i)} < u_k$, where $u_k < a < u_{k+1}$ (take $u_k = u_1$ if $a < u_1$ or $u_k = u_r$ if $a > u_r$; in the sequel we assume without loss of generality that $a > u_1$). Denote by $\tilde{\tau}, \tilde{q}_L, \tilde{q}_R$ the quantities relevant to this split. Let ρ_j denote the probability that a training sample $\mathbf{x}$ that reaches v satisfies $u_k < \mathbf{x}^{(i)} < a$ and has label j. Then

$$\begin{aligned} \tilde{\tau} &= \tau - \rho_0 - \rho_1 \\ \tilde{q}_{L,j} &= \frac{\tau \cdot q_{L,j} - \rho_j}{\tilde{\tau}} \\ \tilde{q}_{R,j} &= \frac{(1-\tau) q_{R,j} + \rho_j}{1 - \tilde{\tau}}. \end{aligned}$$

By the continuity of $\Delta(\tau, \{q_j\}, \{q_{L,j}\}, \{q_{R,j}\})$, for every $\delta > 0$ there exists ε such that

$$\Delta(\tau, \{q_j\}, \{q_{L,j}\}, \{q_{R,j}\}) - \Delta(\tilde{\tau}, \{q_j\}, \{\tilde{q}_{L,j}\}, \{\tilde{q}_{R,j}\}) < \delta.$$

for all $\rho_j < \varepsilon$. Since $\rho_j \leq \frac{1}{B+1}$, we can guarantee that $\rho_j < \varepsilon$ for all j by setting $B = 1/\varepsilon$. We thus have $\Delta(v, \tilde{i}, \tilde{a}) \geq \Delta(v, i, u_k) \geq \Delta(v, i, a) - \delta$, as required.

Theorem 2 implies the following corollary.

Corollary 3 *Assume that the standard decision tree algorithm performs locally well with respect to a function $f(\{q_j\})$, and that the functions operating on histograms return exact answers. Then for every positive function $\delta(\{q_j\})$, the SPDT algorithm performs locally well with respect to $f(\{q_j\}) - \delta(\{q_j\})$, in the sense that for every training set there exists B such that the tree constructed by the SPDT algorithm with B bins performs locally well. Moreover, B does not depend on the size of the*

Algorithm 7 Decision Tree One Node per Iteration

input training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

Initialize T to be a single node.
while there are unlabeled leaves in T **do**
  for all unlabeled leaves v in T **do**
    if v satisfies the stopping criterion **or** there are no samples reaching v **then**
      Label v with the most frequent label among the samples reaching v
    else
      Choose candidate splits for v and estimate Δ for each of them.
    end if
  end for
  Split an unlabeled leaf v such that $n_v\Delta$ is maximal among all unlabeled leaves and all possible candidate splits, where n_v is the number of samples reaching v.
end while

training set, implying constant memory and communication complexity and constant running time at the master processor.

We conclude this section with an example in which we explicitly derive an upper bound on the error rate of SPDT. Set $f(\{q_j\}) = \alpha G(\{q_j\})$ for a positive constant α, for which we have seen in Section 3.1 that $e_{T_t} \leq G_0(t-1)^{-\alpha/2}$. We note that Kearns and Mansour (1999) show that for $G(q_1, q_2) = \sqrt{q_1 q_2}$, the Weak Hypothesis Assumption implies good local performance with $f(q_1, q_2) = \alpha G(q_1, q_2)$. Applying Corollary 3 with $\delta(\{q_j\}) = \frac{\alpha}{2} G(\{q_j\}) = f(\{q_j\})/2$, we deduce that when using histograms with enough bins, the SPDT's error rate is guaranteed to be no more than $G_0(t-1)^{-\alpha/4}$.

4. Empirical Results

In the following section we empirically test the proposed algorithms. We first show the accuracy of the histogram building and merging procedures, and later compare the accuracy of SPDT compared to a standard decision tree algorithm.

4.1 Histogram Algorithms

We evaluated the accuracy of the histogram building and information extraction algorithms. We ran experiments on seven synthetic sets, generated via different kinds of probability distributions, summarized in Table 1. Each set S, consisting of 10^5 points, was partitioned into four equal parts, denoted $S_1 - S_4$. For each part S_k we built a histogram h_k with $B = 100$ bins, using the `update` procedure. We then ran the `uniform` procedure on h_k with $\tilde{B} = 100$, resulting in a sequence of points $u_1, \ldots, u_{99}$. For each pair of subsequent numbers u_i, u_{i+1}, we checked how many points of S_k are in the interval $[u_i, u_{i+1}]$. We expect to see $\frac{|S_k|}{\tilde{B}} = 25000/100 = 250$ points in each such interval. Our findings are summarized in Table 2. We observe that the mean absolute difference between 250 and the actual number of points in an interval is equal to 11.17 (4.47% of the expected quantity).

We repeat the same experiment on the histograms $h_{1,2}, h_{3,4}$, obtained after merging h_1 with h_2 and h_3 with h_4. The mean absolute difference between $50000/100 = 500$ and the number of points

Distribution	Probability density function
Normal	$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2}$
Uniform	$f(x) = 1,\ 0 \le x \le 1$
Exponential	$f(x) = \frac{1}{\mu} e^{-(x/\mu)},\ \mu = 0.5,\ x \le 0$
Beta	$f(x) = \frac{1}{\int_0^1 t^{a-1}(1-t)^{b-1}dt} x^{a-1}(1-x)^{b-1},\ a = 0.5,\ b = 0.5,\ 0 < x < 1$
Gamma	$f(x) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{-x/b},\ a = 3,\ b = 1,\ x \ge 0$
Lognormal	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln(x)-\mu)^2/2\sigma^2},\ \mu = 1,\ \sigma = 0.5,\ x > 0$
Chi-square	$f(x) = \frac{1}{2^{v/2}\Gamma(v/2)} x^{(v-2)/2} e^{-x/2},\ v = 10,\ x \ge 0$

Table 1: Probability density functions of synthetic sets used in the experiments described in Section 4.1.

Distribution	Mean			Standard Deviation		
	Average of $h_1 - h_4$	Average of $h_{1,2}$ and $h_{3,4}$	$h_{1,2,3,4}$	Average of $h_1 - h_4$	Average of $h_{1,2}$ and $h_{3,4}$	$h_{1,2,3,4}$
Normal	11.53	26.22	68.89	15.8	36.83	107.45
Uniform	5.99	18.57	34.13	7.55	24.09	46.84
Exponential	13.78	30.5	18.36	39.28	31.52	83.93
Beta	6.95	18.51	30.91	9.56	24.7	45.26
Gamma	11.87	20.4	61.7	15.68	32.08	84.41
Lognormal	15.93	34.75	72.62	21.59	45.03	93.84
Chi-square	12.12	28.17	56	16.42	38	73.75
Average over all data sets	11.17	25.87	55.36	14.99	34.29	76.5
Percent error, averaged over all data sets	4.47	5.17	5.54			

Table 2: Mean absolute difference between the number of points in $[u_i, u_{i+1}]$ and the desired number and standard deviation of the number of points in $[u_i, u_{i+1}]$. Details are in Section 4.1.

in $(S_k \cup S_{k+1}) \cap [u_i, u_{i+1}],\ k = 1,3$, is 25.87 (5.17% of the expected quantity). Finally, we merged $h_{1,2}$ with $h_{3,4}$. Applying the `uniform` procedure, the obtained mean absolute difference between 1000 and $S \cap [u_i, u_{i+1}]$ is 55.36 (5.54% of the expected quantity).

The `sum` and `uniform` procedures assume that there are $(m_i + m_{i+1})/2$ points in every interval $[p_i, p_{i+1}]$. We tested this assumption on the histograms $h_1 - h_4, h_{1,2}, h_{3,4}$ and $h_{1,2,3,4}$. For $h_{1,2,3,4}$, the mean absolute differences between $(m_i + m_{i+1})/2$ and the actual number of points in $[p_i, p_{i+1}]$ is 28.79. Recall that on average there are 1000 points in each interval, implying an error of 2.88%. Details are in Table 3.

Figure 1 shows how accuracy is affected by the distribution's skewness.[4] The figure was obtained by calculating the histograms $h_{1,2,3,4}$ and points $u_1, \ldots, u_{99}$ for different values of the param-

4. The skewness of a distribution is defined to be κ_3/σ^3, where κ_3 is the third moment and σ is the standard deviation.

Distribution	Average of $h_1 - h_4$	Average of $h_{1,2}$ and $h_{3,4}$	$h_{1,2,3,4}$
Normal	4.22	11.07	23.01
Uniform	5.06	14.18	30.28
Exponential	3.74	12.17	24.21
Beta	6.6	15.98	33.2
Gamma	4.02	12.56	18.94
Lognormal	3.68	13.52	29.29
Chi-square	4.14	12.42	28.58
Average over all data sets	4.5	13.13	28.79
Percent error, averaged over all data sets	1.8	2.63	2.88

Table 3: Mean absolute difference between the number of points in $[p_i, p_{i+1}]$ and $(m_i + m_{i+1})/2$. Details are in Section 4.1.

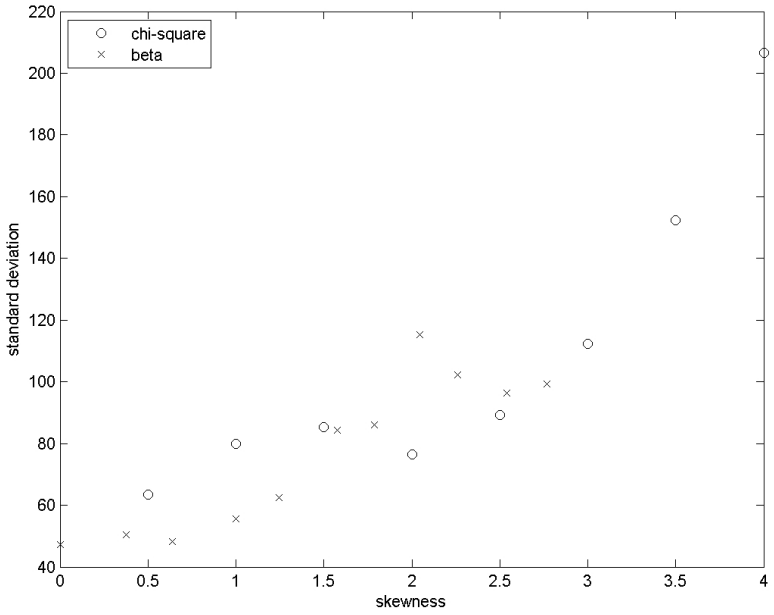

Figure 1: Standard deviation of the number if points in $[u_i, u_{i+1}]$ as a function of the distribution's skewness. The different degrees of skewness are obtained by varying the parameter v of the chi-square distribution and the parameter b of the beta distribution with $a = 0.5$ (see Table 1). More details are given in Section 4.1.

eters of the beta and chi-square distributions. We observe that highly skewed distributions exhibit less accurate results.

Data Set	Number of examples	Number of features	Number of classes
Adult	32561 (16281)	105	2
Isolet	6238 (1559)	617	2
Letter	20000	16	2
Nursery	12960	25	2
Page Blocks	5473	10	2
Pen Digits	7494 (3498)	16	2
Spam Base	4601	57	2
Magic	19020	10	2
Abalone	4177	10	28
Multiple Features	2000	649	11
Face Detection	100000 (10000)	900	2
OCR	100000 (10000)	1156	2

Table 4: Properties of the data sets used in the experiments. The number of examples in parentheses is the number of test examples (if a train/test partition exists).

4.2 Evaluation of the SPDT Algorithms

We ran our experiments on ten medium-sized data sets taken from the UCI repository (Blake et al., 1998) and two large data sets taken from the Pascal Large Scale Learning Challenge (Pascal, 2008). The characteristics of the data sets are summarized in Table 4. For the UCI data sets, we applied ten-fold cross validation when a train/test partition was not given. For the Pascal data sets, we extracted 10^5 examples to constitute a training set, and additional 10^4 examples to constitute a test set. We set the number of bins to 50, and limited the depth of the trees to no more than 100 for the UCI data sets and 10 for the Pascal data sets. We implemented our algorithm in the IBM Parallel Machine Learning toolbox (PML), which runs using MPICH2, and executed it on an 8-CPU Power5 machine with 16GB memory using a Linux operating system. We note that none of the experiments reported in previous works involved both a large number of examples and a large number of attributes.

We began by testing the assumption that splits chosen by the SPDT algorithm are close to optimal. To this end, we extracted four continuous attributes from the training sets (we chose the training set of the first fold if there was no train/test partition). For every attribute, we calculated the following three quantities: Δ of the optimal splitting point, Δ of the splitting point chosen by SPDT with 8 processors, and average Δ over all splitting points (chosen by random splitting). We then normalized by $G(\{q_j\})$, that is,

$$\tilde{\Delta} = \frac{\Delta}{G(\{q_j\})} = 1 - \frac{\tau G(\{q_{L,j}\}) + (1-\tau) G(\{q_{R,j}\})}{G(\{q_j\})}.$$

The normalized value $\tilde{\Delta}$ can be interpreted as the split's efficiency. Since $G(\{q_j\})$ is the maximum possible value of Δ, $\tilde{\Delta}$ represents the ratio between what is actually achieved and the maximum that can be achieved. Table 7 displays the gain of the various splitting algorithms.

Data Set	Constant classification	Standard tree	SPDT 1 worker	SPDT 2 workers	SPDT 4 workers	SPDT 8 workers
Adult	24	15.73	15.79	15.88	**15.69**	15.83
Isolet	50	**14.95**	22.58	26.62	23.09	26.17
Letter	50	**8.52**	8.59	8.59	8.59	8.59
Nursery	34	**2.07**	2.17	2.17	2.17	2.17
Page Blocks	10	**2.89**	3.29	3.09	3.03	3.42
Pen Digits	48	5.37	3.77	3.63	3.63	**3.63**
Spam Base	39	8.17	**6.91**	7.02	7.15	7.22
Magic	35	**17.91**	18.38	18.41	17.95	17.92
Abalone	83.5	**79.33**	79.93	80.6	79.93	80
Multiple Features	90	8.85	8.5	8.15	**8.5**	8.7
Face Detection	8.5	-	**3.31**	4.18	4.13	4.03
OCR	48	-	44.1	42.85	**39.35**	40.73

Table 5: Percent error for UCI and Pascal data sets. The lowest error rate for each data set is marked in bold. The "constant classification" column is the percent error of a classifier that always outputs the most frequent class, that is, it is 100% minus the frequency of the most frequent class.

Data Set	Standard tree	SPDT 1 worker	SPDT 2 workers	SPDT 4 workers	SPDT 8 workers
Adult	81.18	80.75	80.84	80.69	**81.38**
Isolet	**89.7**	77.72	69.45	73.93	70.71
Letter	**95.56**	94.89	94.89	94.89	94.91
Nursery	**99.72**	99.69	99.69	99.69	99.69
Page Blocks	95.48	94.69	95.84	**96.28**	95.05
Pen Digits	97.2	**97.48**	97.37	97.37	97.37
Spam Base	**95.25**	94.95	93.68	94.32	94.22
Magic	80.17	79.81	79.69	80.1	**80.27**
Face Detection	-	97.76	97.32	97.25	95.44
OCR	-	61.72	61.48	**63.85**	62.57

Table 6: Area under ROC curve (%) for UCI and Pascal data sets with binary classification problems. The highest AUC for each data set is marked in bold.

Data Set	Attribute	$\tilde{\Delta}_{OPTIMAL}$	$\tilde{\Delta}_{SPDT}$	$\tilde{\Delta}_{RANDOM}$
Isolet	1	0.0239	0.0231	0.0108
Page Blocks	9	0.1125	0.0985	0.0199
Spam Base	55	0.2044	0.1393	0.1295
Magic	1	0.128	0.1228	0.0304

Table 7: $\tilde{\Delta}$ of splits chosen by the standard tree, SPDT, and random splitting. Details are given in Section 4.2.

Data Set	Err. (%) before pruning	Err. (%) after pruning	AUC (%) before pruning	AUC (%) after pruning	Tree size before pruning	Tree size after pruning
Adult	15.83	13.83	81.38	88.08	5731	359
Isolet	26.17	25.79	70.71	69.9	403	281
Letter	8.59	9.9	94.91	95.29	1069	433
Nursery	2.17	2.28	99.69	99.66	210	194
Page Blocks	3.42	3.46	95.05	95.19	62	29
Pen Digits	3.63	4	97.37	96.75	87	77
Spam Base	7.22	9.48	94.22	94.39	384	95
Magic	17.92	14.75	80.27	88.81	3690	258
Abalone	80	73.5	-	-	4539	93
Multiple Features	8.7	8.25	-	-	173	52
Face Detection	4.03	3.91	95.44	97.75	253	169
OCR	40.73	40.63	62.57	62.63	625	447

Table 8: Percent error, areas under ROC curves, and tree sizes (number of tree nodes) before and after pruning, with eight processors.

We proceed to inspect the tree's accuracy. Tables 5 and 6 display the error rates and areas under the ROC curves of the standard decision tree and the SPDT algorithm with 1, 2, 4, and 8 processors.[5] We note that it is infeasible to apply the standard algorithm on the Pascal data sets, due to their size. For the UCI data sets, we observe that the approximations undertaken by the SPDT algorithm do not necessarily have a detrimental effect on its error rate. The FF statistics combined with Holm's procedure (Demšar, 2006) with a confidence level of 95% shows that the SPDT algorithm exhibited accuracy that could not be detected as statistically significantly different from that of the standard algorithm.

It is also interesting to study the effect of pruning on the error rate and tree size. Using the procedure described in Section 2.2, we pruned the trees obtained by SPDT. Table 8 shows that pruning usually improves the error rate (though not to a statistically significant threshold, using sign test with $p < 0.05$) while reducing the tree size by 54% on average.

Figure 2 shows the speedup for different sized subsets of the face detection and OCR data sets. Referring to data set size as the number of examples multiplied by the number of dimensions, we found that data set size and speedup are highly correlated (Spearman correlation of 0.90). We further checked the running time as a function of the data set size. In a logarithmic scale, we obtain approximate regression curves (average $R^2 = 0.99$, see Figure 3). The slopes of the curves decrease as the number of processors increases, and drops below 1 for eight processors. In other words, if we multiply the data size by a factor of 10, the running time is multiplied by less than 10.

The results presented here fit the theoretical analysis of Section 2.3. For large data sets, the communication between the processors in the merging phase is negligible relative to the gain in the update phase. Therefore, increasing the number of processors is especially beneficial for large data sets.

5. The results for the OCR data set can be somewhat improved if we increase the tree depth to 25 instead of 10. For four processors, we obtain an error of 32.56% and AUC of 67.5%.

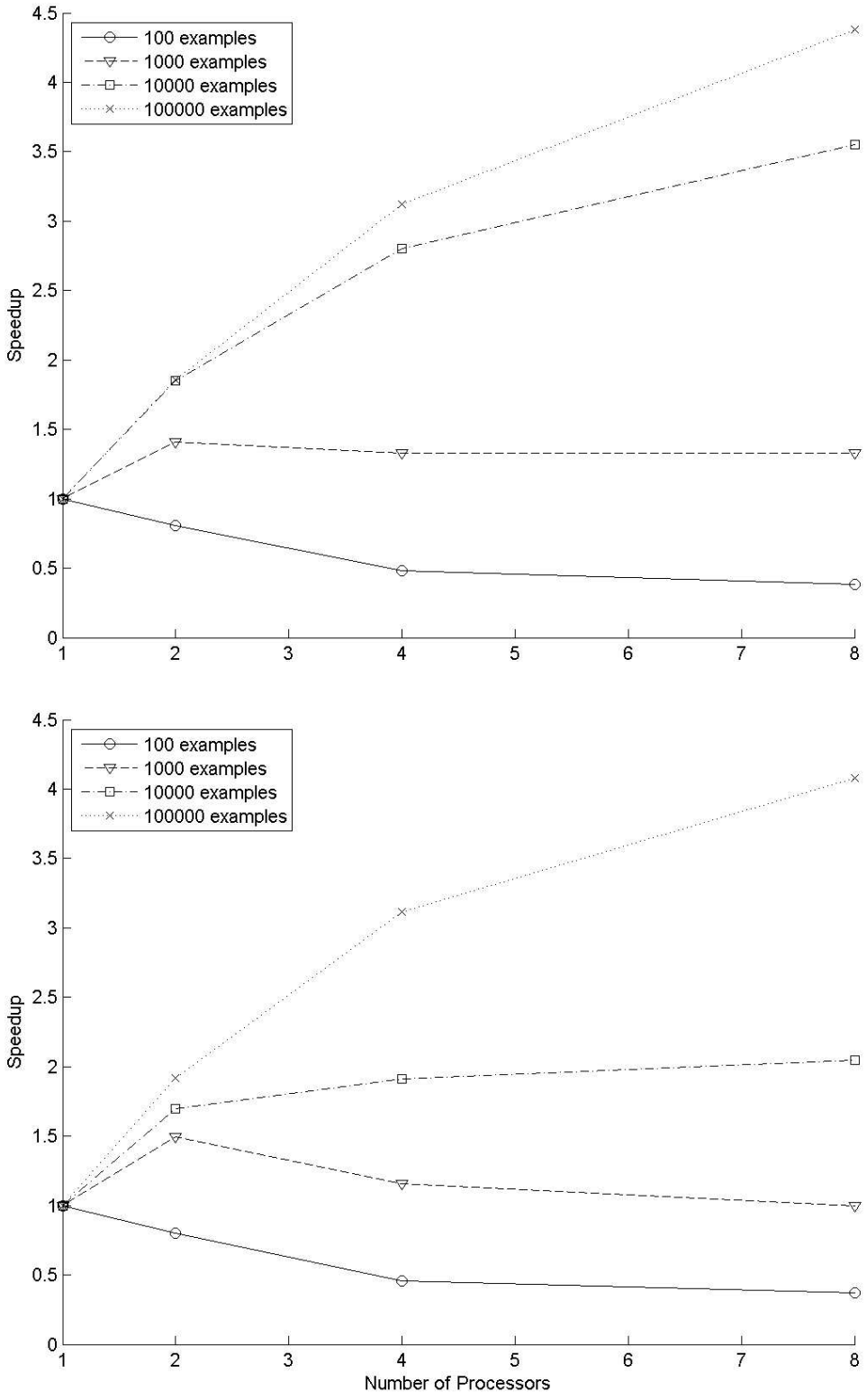

Figure 2: Speedup of the SPDT algorithm for the `face detection` (top) and `OCR` (bottom) data sets.

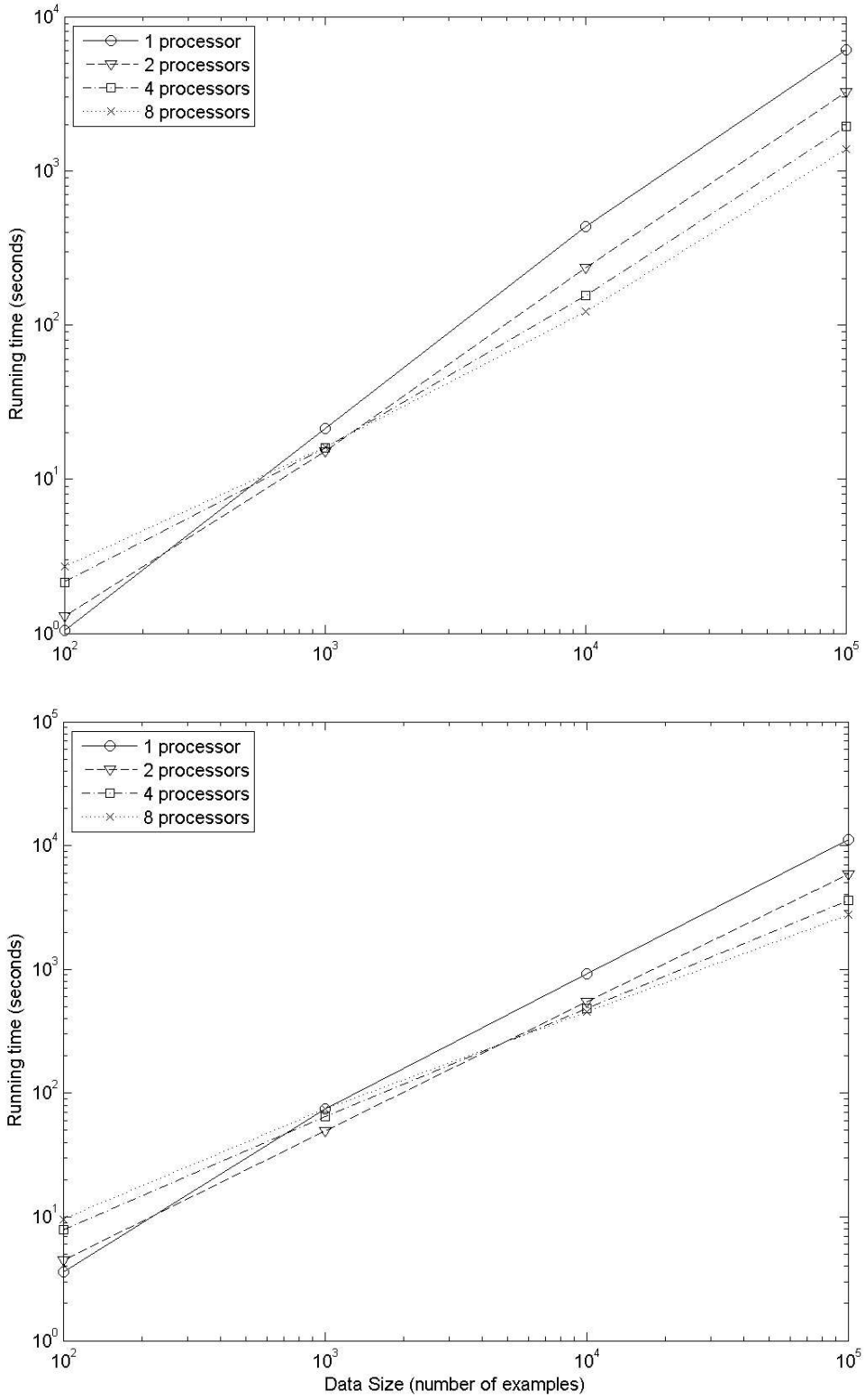

Figure 3: Running time vs. data size for the `face detection` (top) and `OCR` (bottom) data sets.

5. Conclusions

We propose a new algorithm for building decision trees, which we refer to as the Streaming Parallel Decision Tree (SPDT). The algorithm is specially designed for large data sets and streaming data, and is executed in a distributed environment. Our experiments reveal that the error rate of SPDT is approximately the same as for the serial algorithm. We also provide a way to analytically compare the error rate of trees constructed by serial and parallel algorithms without comparing similarities between the trees themselves.

Acknowledgments

We thank the referees for their valuable comments.

Appendix A.

We demonstrate how the histogram algorithms run on the following input sequence:

$$23, 19, 10, 16, 36, 2, 9, 32, 30, 45. \tag{3}$$

Suppose that we wish to build a histogram with five bins for the first seven elements. To this end, we perform seven executions of the `update` procedure. After reading the first five elements, we obtain the histogram

$$(23,1),(19,1),(10,1),(16,1),(36,1).$$

as depicted in Figure 4(a). We then add the bin $(2,1)$ and merge the two closest bins, $(16,1)$ and $(19,1)$, to a single bin $(17.5,2)$. This results in the following histogram, depicted in Figure 4(b):

$$(2,1),(10,1),(17.5,2),(23,1),(36,1).$$

We repeat this process for the seventh element: the bin $(9,1)$ is added, and the two closest bins, $(9,1)$ and $(10,1)$, form a new bin $(9.5,2)$. The resulting histogram is given in Figure 4(c):

$$(2,1),(9.5,2),(17.5,2),(23,1),(36,1).$$

Let us now merge the last histogram with the following one:

$$(32,1),(30,1),(45,1).$$

Figure 5 follows the changes in the histogram during the three iterations of the `merge` procedure. We omit details due to the similarity to the `update` examples given above. The final histogram is given in Figure 5(d):

$$(2,1),(9.5,2),(19.33,3),(32.67,3),(45,1).$$

This histogram represents the set in (3).

We now wish to estimate the number of points smaller than 15. The leftmost bin $(2,1)$ gives 1 point. The second bin, (9.5,2), has $2/2 = 1$ points to its left. The challenge is to estimate how many points to its right are smaller than 15. We first estimate that there are $(2+3)/2 = 2.5$ points inside the trapezoid whose vertices are $(9.5,0),(9.5,2),(19.33,3)$, and $(19.33,0)$ (see Figure 6). Assuming that the number of points inside a trapezoid is proportional to its area, the number of points

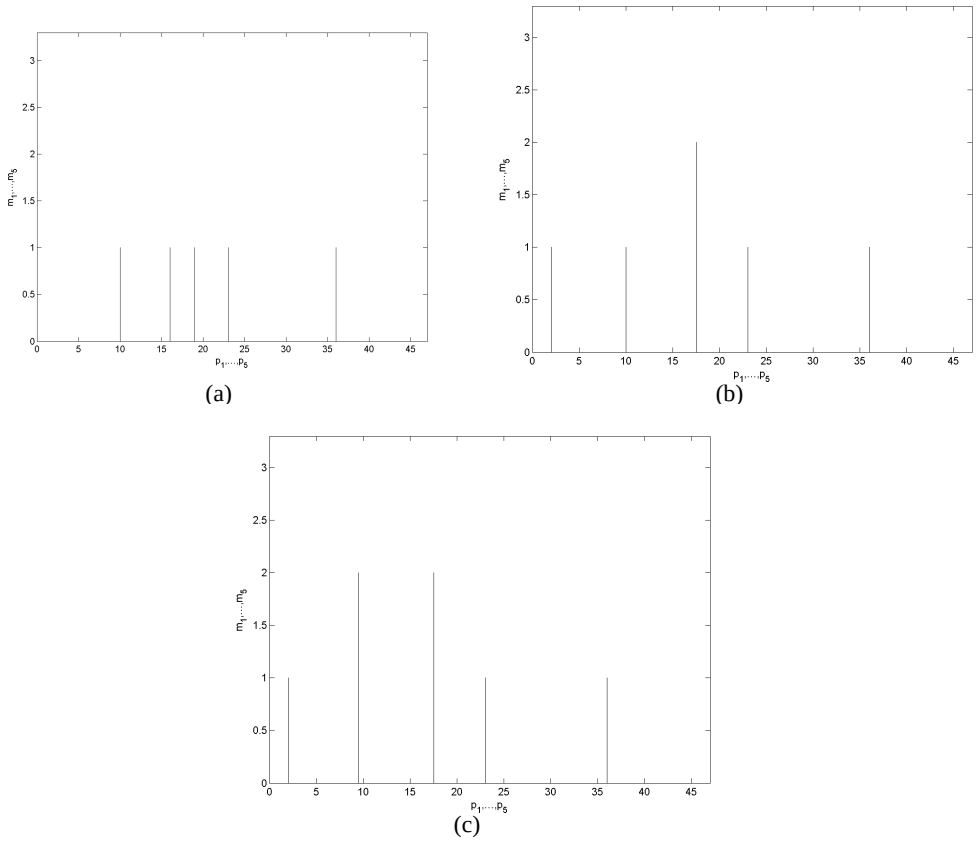

Figure 4: Examples of executions of the `update` procedure.

inside the trapezoid defined by the vertices $(9.5,0),(9.5,2),(15,2.56)$, and $(15,0)$ is estimated to be

$$\frac{2+2.56}{2} \times \frac{15-9.5}{19.33-9.5} = 1.28.$$

We thus estimate that there are in total $1+1+1.28 = 3.28$ points smaller than 15. The true answer, obtained by looking at the set represented by the histogram (see Equation (3)), is three points: 2, 9, and 10.

The reader can readily verify that the `uniform` procedure with $\tilde{B} = 3$ returns the points 15.21 and 28.98. Each one of the intervals $[-\infty, 15.21], [15.21, 28.98]$, and $[28.98, \infty]$ is expected to contain 3.33 points. The true values are 3, 2, and 4, respectively.

References

Rakesh Agrawal and Arun Swami. A one-pass space-efficient algorithm for finding quantiles. In *Proceedings of COMAD, Pune, India*, 1995.

Khaled AlSabti, Sanjay Ranka, and Vineet Singh. CLOUDS: Classification for large or out-of-core datasets. In *Conference on Knowledge Discovery and Data Mining*, August 1998.

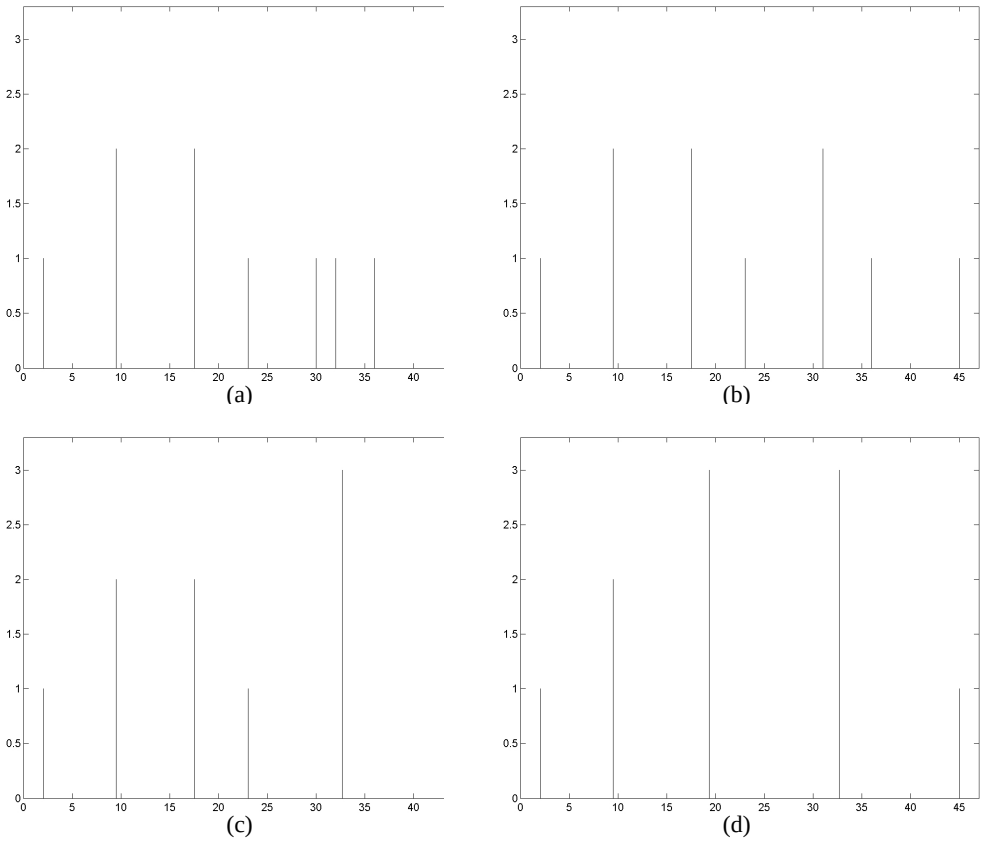

Figure 5: An example of an execution of the `merge` procedure.

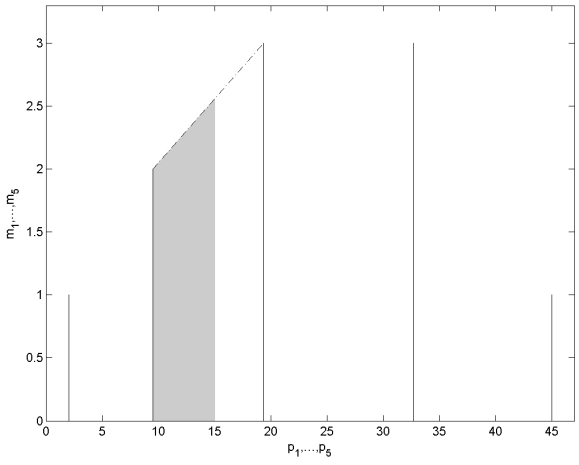

Figure 6: The `sum` procedure.

Nuno Amado, Joao Gama, and Fernando Silva. Parallel implementation of decision tree learning algorithms. In *The 10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving*, pages 6–13, December 2001.

Catherine L. Blake, Eamonn J. Keogh, and Christopher J. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Leon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20. MIT Press, Cambridge, MA, 2008. URL `http://leon.bottou.org/papers/bottou-bousquet-2008`. to appear.

Leo Breiman, Jerome H. Friedman, Richard Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Monterrey, CA, 1984.

Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. BOAT — optimistic decision tree construction. In *ACM SIGMOD International Conference on Management of Data*, pages 169–180, June 1999.

Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. How to summarize the universe: dynamic maintenance of quantiles. In *Proceedings of the 28th VLDB Conference*, pages 454–465, 2002.

Sanjay Goil and Alok Choudhary. Efficient parallel classification using dimensional aggregates. In *Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 197–210, August 1999.

Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM SIGMOD, Santa Barbara, California*, pages 58–66, 'may' 2001.

Isaac D. Guedalia, Mickey London, and Michael Werman. An on-line agglomerative clustering method for nonstationary data. *Neural Comp.*, 11(2):521–540, 1999.

Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. on Database Systems*, 31(1):396–438, 'mar' 2006.

Yannis E. Ioannidis. The history of histograms (abridged). In *Proceedings of the VLDB Conference*, pages 19–30, 2003.

Raj Jain and Imrich Chlamtac. The P^2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, 'oct' 1985.

Ruoming Jin and Gagan Agrawal. Communication and memory efficient parallel decision tree construction. In *The 3rd SIAM International Conference on Data Mining*, May 2003.

Mahesh V. Joshi, George Karypis, and Vipin Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *The 12th International Parallel Processing Symposium*, pages 573–579, March 1998.

Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 'feb' 1999.

Xuemin Lin. Continuously maintaining order statistics over data streams. In *Proceedings of the 18th Australian Database Conference, Ballarat, Victoria, Australia*, 2007.

Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of ACM SIGMOD, Seattle, WA, USA*, pages 426–435, 1998.

Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *The 5th International Conference on Extending Database Technology*, pages 18–32, 1996.

Girija J. Narlikar. A parallel, multithreaded decision tree builder. Technical Report CMU-CS-98-184, Carnegie Mellon University, 1998.

Pascal, 2008. Pascal large scale learning challenge, 2008. `http://largescale.first.fraunhofer.de`, datasets can be downloaded from `http://ftp.first.fraunhofer.de/pub/projects/largescale`.

PML. IBM Parallel Machine Learning Toolbox, 2009. `http://www.alphaworks.ibm.com/tech/pml`.

John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *The 22nd International Conference on Very Large Databases*, pages 544–555, September 1996.

Mahesh K. Sreenivas, Khaled Alsabti, and Sanjay Ranka. Parallel out-of-core divide-and-conquer techniques with applications to classification trees. In *The 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pages 555–562, 1999. Available as preprint in `http://ipdps.cc.gatech.edu/1999/papers/207.pdf`.

Anurag Srivastava, Eui-Hong Han, Vipin Kumar, , and Vineet Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(3):237–261, September 1999.