

# A Parallel Approach for Decision Trees Learning from Big Data Streams

Ionel Tudor Calistru<sup>(✉)</sup>, Paul Cotofrei, and Kilian Stoffel

Information Management Institute, University of Neuchatel, Neuchatel, Switzerland  
{ionel.calistru,paul.cotofrei,kilian.stoffel}@unine.ch

**Abstract.** In this paper we introduce PdsCART, a parallel decision tree learning algorithm. There are three characteristics that are important to emphasize and make this algorithm particularly interesting. Firstly, the algorithm we present here can work with streaming data, i.e. one pass over data is sufficient to construct the tree. Secondly, the algorithm is able to process in parallel a larger amount of data stream records and can therefor handle efficiently very large data sets. And thirdly, the algorithm can be implemented in the MapReduce framework. Details about the algorithm and some basic performance results are presented.

**Keywords:** Bigdata · Business datamining · Streams · MapReduce · Decision trees

## 1 Introduction

Big Data has become the standard term referring the analysis of very large collections of data. Traditionally, these large collections of data were produced in the context of scientific applications e.g. physics, genomics or meteorology. But once the applications in the domain of business and finance gained in interest, an increasing amount of data was generated and collected in this context, confronting the researchers with the difficulties of dealing with very large data set. Finally, a third domain of great importance in the context of Big Data is the domain of data streams, usually generated by all kinds of sensors, such as mobile devices, remote sensors, radio-frequent identification (RFID), etc.

In this paper we are essentially interested in the intersection of the second and third domain mentioned in the previous paragraph, i.e. data produced in the context of business applications in form of streams. Today, the use of apps on mobile phones has become one of the cornerstones at the interaction between enterprises and their clients. These apps, together with the more traditional web sites, produce data logs and other streams which, in volume, count for the most important raw data sets gathered by the companies. Furthermore, the combination of mobile apps and mobile phones containing sensor, such as GPS, generates data in a strict business setting, having a lot of the characteristics typical for a sensor network.

The MapReduce framework [1] has become the de facto standard for the implementation of processes for analysing very large data sets in parallel, using

distributed clusters. Its basic structure consists of two main steps: first a map-step which essentially filters and sorts the data to be analysed, followed by a reduce-step which essentially aggregates the data to be analysed. This simple framework allows for great performance, but it is also quite limited in respect to the algorithms that can be implemented in a straight forward manner. Motivated by some business applications, we are particularly interested in decision tree algorithms. But this category of algorithms is exactly one of those which cannot easily be ported to the MapReduce framework, particularly if some of the specific requirements for business applications have to be respected. In this paper we will propose and analyse an approach allowing to implement decision tree algorithms for big data streams in a MapReduce framework.

The remainder of the papers is organized in the following way. In the next chapter we will present the arguments supporting the choice of decision trees as the data mining algorithm of preference. Then we will describe how decision trees can be used to analyse data streams and how this analysis can be integrated into a MapReduce framework. Then we will present the details regarding the implementation of our parallel decision tree algorithm followed by an analysis of its performance, before to conclude and give some outlooks and future work.

## 2 Decision Trees for Mining Big Data

One of the most effective and widely used techniques in machine learning today is decision tree learning. These models are popular not only because of their adaptability and accurate prediction capabilities, but also because they can provide classification rules that may be easily interpreted by humans. This is a particularly interesting property in the context of mining business data.

However, decision trees come also with some disadvantages. Traditional decision tree algorithms [2,3] have difficulties when the data does not fit into memory, because they have to recursively read the training data sets to construct split decisions. Furthermore, numerical values need to be sorted in order to find the splitting points of a specific node of the tree. To overcome these time and memory consuming constraints several solutions have been proposed.

### 2.1 Related Work

One of the techniques used by learning decision trees algorithms is the pre-sorting of attributes values, as in SPRINT [4] or ScalParC [5]. Alternatively, another approach is to approximate the data, instead of sorting it, by using histogram data structures, as in pCLOUDS [6], SPIES [7] and SPDT [8]. To build the histograms, some authors use simply the frequency of data, where others (SPIES and pCLOUDS) use sampling techniques. Another important difference lays in the number of passes through the data needed, e.g. SPIES and SSE (a version of CLOUDS) may need to pass several times over the data during the construction process. Although the pre-sorting approaches are in general more accurate, they may not be suitable for streams of data.

**Parallel Decision Trees Algorithms.** The challenge of dealing with important amounts of data was largely addressed by several parallel decision tree algorithms such as described in [4–7,9].

In Amado et al. [10] and Srivastava et al. [9] four different types of parallel decision tree algorithms have been described: horizontal, vertical, task and hybrid approaches. In horizontal parallel decision trees, the entire data set is split into subsets which are processed individually. In the vertical case, the set of attributes is partitioned. Task parallelism enables the distribution of decision trees nodes to be processed independently. The fourth type, hybrid parallelism, is a combination of all three already mentioned approaches, i.e. in the first phases of the decision tree growing process, vertical and horizontal parallelism are combined, letting the task parallelism taking over at the end.

An example of a hybrid approach is Google’s PLANET [11], a technique that applies horizontal parallelism (by implementing a MapReduce algorithm) at the first few levels of the tree and applies task parallelism to the leaves, as soon as the data fits into memory.

In [8] the authors use horizontal parallelism to build data histograms, which are then merged to take decisions and to build the tree in a breadth-first manner. Other examples of horizontal parallelism for building trees are gradient boosted decision trees (GBDT [12]) or regression trees (GBRT [13]).

Based on SPDT, Li [14] proposes a random forest algorithm (SRF) with a MapReduce implementation (similar to PLANET one): in the map phase the local histograms are computed, while in the reduce phase the global histogram enables the best split decision to be taken. However, in each iteration, when a new level of the tree needs to be build, the complete data set (or a predefined number of samples, if the data is too big) has to be read. This overrules the single-pass constraint and makes it inadequate for a potentially infinite number of records like in a data stream.

**Decision Trees for Online Stream Mining.** While most of the decision tree algorithms available today are designed to enable the mining of data sets that do not fit into memory, the extremely fast growth - in recent years - of the volume of information that needs to be analysed rises the necessity of new techniques. Ideally, these techniques would be able to continuously process streams of data, without losing any valuable information [15].

The (theoretically) infinite number of records of data streams made the exact determination of the best attribute to split impossible. This lead to the idea of estimation of the best attribute. Due to the fact that this estimation needs to be done in respect to the entire data stream, several techniques supporting the single-pass constraint as well as other data streams particularities [16] have been studied [17–19]. These techniques include Hoeffding’s tree algorithm, Very Fast Decision Tree (VFDT) and Concept-adapting Very Fast Decision Tree (CVFDT). The Vertical Hoeffding Tree (VHT) classifier, introduced in [20], utilizes vertical parallelism to extend the VFDT classifier. All these methods were supported mathematically by the Hoeffding’s inequality [21].

However, in [22,23] the authors showed that the Hoeffding’s inequality is not an adequate probabilistic model for the descriptions of ID3, C4.5 or CART

algorithms. They propose a new approach (inspired by [17]), called CART for data streams (dsCART), which applies a Gaussian approximation to establish the best attribute for splitting a tree node. One of the major results of dsCART is that the selected attribute to split on in a considered tree node relative to its data set, is the same, with some high probability, as the attribute chosen by analysing the entire data stream [24].

## 2.2 The Parallel Approach for Data Streams

Following our interest in online mining of business data, we propose PdsCART, a parallel approach to build the decision trees for inferring and predicting from big data streams. We select the dsCART [24] decision tree algorithm for data stream classification as the basis for our approach.

Our proposed solution is a method to adapt the dsCART algorithm to horizontal parallelism by implementing the MapReduce programming model. While several other horizontal parallel solutions have been mentioned already, like SPDT, PLANET, SRF, GBDT, GBRT etc., to the best of our knowledge, none of them have been applied to a single-pass decision tree for data streams algorithm. More details about our approach are presented in the Implementation section.

## 3 PdsCART Implementation

In this section we describe PdsCART, our approach to parallelise the dsCART decision tree algorithm. To do this, first we introduce both dsCART<sup>1</sup> and PdsCART decision tree algorithms for data streams, and then we detail our MapReduce implementation. It is important mentioning that we are only interested in showing that very similar learning models may be achieved by treating records in parallel, reducing this way the overall stream processing time.

Before presenting the pseudo-code, the following notes are important:

- for each attribute  $a^i$ , the set of attribute values  $A^i$  is partitioned into two disjoint subsets  $A_L^i$  and  $A_R^i$  such that  $A^i = A_L^i \cup A_R^i$ ;
- the choice of  $A_L^i$  automatically determines the complementary subset  $A_R^i$ .
- the set of all possible partitions of the set  $A^i$  is denoted by  $V_i$ .
- $\overline{g}_q^i$  is the Gini gain computed for the attribute  $a^i$  in the leaf  $L_q$ .
- $n_{i,\lambda,q}^k$  is the number of elements from the  $k$ -th class in the leaf  $L_q$ , for which the value of the attribute  $a^i$  is equal to  $a_\lambda^i$ , ( $a_\lambda^i \in A^i$ ).
- $n_q^k$  is the number of elements from the  $k$ -th class in the leaf  $L_q$
- the tie breaking mechanism (  $\theta$  ) forces the split after some fixed number of elements.
- the stopping condition formula is:

$$\epsilon_{G,K} = z_{1-\alpha} \frac{\sqrt{2Q(K)}}{\sqrt{n}}, \text{ where } Q(K) = 5K^2 - 8K + 4 \quad (1)$$

and  $z_{(1-\alpha)}$  is the  $(1-\alpha)$ -th quantile of the standard normal distribution  $N(0,1)$ ;  $K$  is the number of classes,  $n$  is the number of samples in the considered tree node.

<sup>1</sup> We are following very closely the description of dsCART algorithm done by Leszek Rutkowski, Maciej Jaworski, Lena Pietruczuk and Piotr Duda in [24].

**The dsCART Algorithm [24]****Inputs:**

$S$  is a sequence of examples,  
 $U$  is a set of discrete attributes,  
 $\alpha$  is one minus the desired probability of choosing the correct attribute,  
 $\theta$  is the tie breaking parameter.

**Output:** dsCART decision tree

**Procedure dsCART**( $S, U, \alpha, \theta$ );

Let dsCART be a single leaf  $L_0$  (the root);  
 Let  $U_0 = U$ ;  
 /\* Initialize counters in  $L_0$  with 0 \*/  
 $n_0 = 0$ ;

```

for each example  $s$  in  $S$  do
  Sort  $s$  into tree leaf  $L_q$ ;
  for each attr.  $a^i \in U_q$  do
     $a^i_\lambda$  is the value of  $s$  for  $a^i$ ;
     $k$  is the class of  $s$ ;
    Increment  $n^k_{i,\lambda,q}$ ;
  end
  Label  $L_q$  with the majority class;
  if  $L_q$  has more than one class then
    for each attr.  $a^i \in U_q$  do
      for each partition of the set  $A^i$ 
        into  $A^i_L, A^i_R$  do
          Get  $\overline{g_q}(A^i_L)$  using  $n^k_{i,\lambda,q}$ ;
        end
        Let  $\overline{g_q} = \max_{A^i_L \in V_i} \{\overline{g_q}(A^i_L)\}$ ;
      end
      Let  $a^x = \arg \max_{a^i \in U_q} \{\overline{g_q}^i\}$ ;
      Let  $a^y = \arg \max_{a^i \in U_q \setminus \{a^x\}} \{\overline{g_q}^i\}$ ;
      Get  $\epsilon_{G,K}$  using (1);
      if  $(\overline{g_q}^x - \overline{g_q}^y > \epsilon_{G,K})$  or  $(\epsilon_{G,K} < \theta)$  then
        Split  $L_q$  on  $a^x$ ;
        for both branches of the split do
          Add a new leaf  $L_{last+1}$ ;
          Let  $U_{last+1} = U_q \setminus \{a^x\}$ ;
           $n_{last+1} = 0$ ;
          last = last + 1;
        end
      end
    end
  end
end
return dsCART
  
```

**The PdsCART Algorithm****Inputs:**

$S$  is a sequence of examples,  
 $U$  is a set of discrete attributes,  
 $\alpha$  is one minus the desired probability of choosing the correct attribute,  
 $\theta$  is the tie breaking parameter.  
 $R$  is the number of records to process in parallel

**Output:** PdsCART decision tree

**Procedure PdsCART**( $S, U, \alpha, \theta, R$ );

Let PdsCART be a single leaf  $L_0$  (the root);  
 Let  $U_0 = U$ ;  
 Let  $Data$  be the list of examples read from the stream;  
 Let  $TLeaves$  be the list of tree leaves;  
 $TLeaves.Add(L_0)$ ;  
 /\* Initialize counters in  $L_0$  with 0 \*/  
 $n_0 = 0$ ;

```

for each example  $s$  in  $S$  do
   $Data.Add(s)$ 
  if  $Data.size == R$  then
    Controller( $Data, TLeaves, PdsCART$ )
     $Data.clear()$ 
  end
end
if  $Data.size > 0$  then
  Controller( $Data, TLeaves, PdsCART$ )
   $Data.clear()$ 
end
return PdsCART
  
```

**Procedure Controller**( $data, leaves, tree$ )

```

/* Assign Data To Mappers */
job = Controller.Assign( $data$ );
/* Call Map and Reduce */
job.Run( $tree$ );
/* Collect Histograms for each tree leaf */
Controller.CollectHistograms();
  
```

```

for each leaf  $L_q$  in  $leaves$  do
  Label  $L_q$  with the majority class;
  if  $L_q$  has more than one class then
    for each attr.  $a^i \in U_q$  do
      for each partition of the set  $A^i$  into
         $A^i_L, A^i_R$  do
          Get  $\overline{g_q}(A^i_L)$  using  $n^k_{i,\lambda,q}$ ;
        end
        Let  $\overline{g_q}^i = \max_{A^i_L \in V_i} \{\overline{g_q}(A^i_L)\}$ ;
      end
      Let  $a^x = \arg \max_{a^i \in U_q} \{\overline{g_q}^i\}$ ;
      Let  $a^y = \arg \max_{a^i \in U_q \setminus \{a^x\}} \{\overline{g_q}^i\}$ ;
      Get  $\epsilon_{G,K}$  using (1);
      if  $(\overline{g_q}^x - \overline{g_q}^y > \epsilon_{G,K})$  or  $(\epsilon_{G,K} < \theta)$  then
         $leaves.Remove(L_q)$ ;
        Split  $L_q$  on  $a^x$ ;
        for both branches of the split do
          Add a new leaf  $L_{last+1}$ ;
          Let  $U_{last+1} = U_q \setminus \{a^x\}$ ;
           $n_{last+1} = 0$ ;
           $leaves.Add(L_{last+1})$ ;
          last = last + 1;
        end
      end
    end
  end
end
  
```

### 3.1 Preliminary Considerations

Following the dsCART algorithm it is easy to see that the most time consuming phase is to find the best split decision; for each attribute in the current node, we must compute the Gini gains in respect to all possible partitions of the set of attribute values. It is important to keep in mind that all these operations take place for each and every new sample that is read from the data stream.

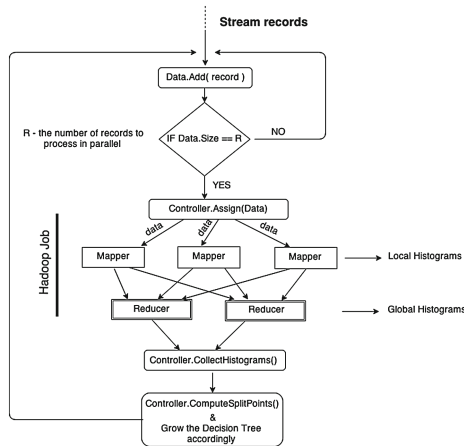
Secondly, we recall here that in [24] has been proven that the attribute chosen in a considered node, according to its current data, is the same one, with high probability, as the one selected after reading the entire data. This means that no matter when these estimations are made, they chose, with some probability, the same attribute.

All these facts motivated us to compute and check the splitting conditions after reading a variable number of samples, while processing them independently. By choosing a sufficiently high probability ( $\alpha$  parameter), our algorithm is able to produce very similar (compared with dsCART) decision trees, with the same level of accuracy but with faster processing times. The parallel approach of PdsCART algorithm is detailed in the following subsection, while a summary of the results using this algorithm are given in the Experiments section.

### 3.2 The MapReduce Implementation

In our version of the distributed PdsCART decision trees, we apply the MapReduce paradigm by using a horizontal partitioning approach.

The controller process coordinates the tree growing, while the mappers and reducers processes fulfill their standard tasks. Assuming that we have  $P$  mappers and we want to consume  $R$  records in parallel, the controller will assign to each mapper  $R/P$  records to process.



**Fig. 1.** The PdsCART algorithm logic schema.

In order to keep track of the number of distinct elements for each attribute and class, PdsCART uses some simple data frequency structures, as histograms, which are easy to merge in order compute the Gini gain functions. For each leaf node in the tree, each mapper will build its own local histograms. During the map phase, each input record it is assigned to a leaf node and it is inserted in the corresponding mapper local histogram. The reducers receive from the mappers all the local histograms and merges them into global histograms, one for each leaf node of the tree. The result is serialized into an output file. Given this output file, the controller can then perform a single pass over the leaves global histograms, estimate the best splitting attributes and decide whether to grow the tree in respect to the splitting conditions of each of the tree leaf nodes.

The Map procedure, described in the *Map algorithm*, receives the current version of the tree as well as a set of records. For each record, the tree is traversed in order to find the corresponding leaf node *id*. Based on the node *id*, the mapper updates its dedicated local histogram with the new input record. Once the last record is processed the mapper emits, for each leaf node, the local histograms and their related tree node *id*.

The Reduce procedure, described in *Reduce algorithm*, merges all the local histograms received from the mappers and builds the global ones. The result is exported to the output file.

#### Map Algorithm

---

**Inputs:**  
**tree** : current decision tree;  
**records** : data assigned to the current mapper  
**Output:** (*key, value*) pairs for each leaf;  
**key**: the id of the leaf node;  
**value**: the corresponding local histogram

---

**Procedure MAP**(*tree, records*)  
 HistoMap  $\leftarrow$  local histograms map;  
**while** *records.readNextRecord()* **do**  
   sample  $\leftarrow$  SplitRecord(record, delim);  
   nodeId  $\leftarrow$  tree.Traverse(sample);  
   HistoMap.Add(nodeId, sample);  
**end**  
**for each** *nodeId* in HistoMap **do**  
   histogram  $\leftarrow$  HistoMap(nodeId);  
   emit(nodeId, histogram);  
**end**

#### Reduce Algorithm

---

**Inputs:**  
**(key, values)** pairs emitted by Mappers  
 and grouped by the key;  
**Output:** (*key, value*) pair  
**key**: the id of the leaf node;  
**value** : the global histogram - merged

---

**Procedure REDUCE**(*key, values*)  
 leafId  $\leftarrow$  key;  
 globalHistogram  $\leftarrow$  the global histogram  
 for the current key(leafId);  
**for each** *histogram* in *values* **do**  
   globalHistogram.MergeHistogram(leafId,  
   histogram);  
**end**  
 emit(leafId, globalHistogram);

With a single iteration of the output file, the controller can now compute the first and the second best attribute, as well as their related Gini gains, for each and every leaf node in the tree. The PdsCART algorithm can now move forward to compute the splitting conditions, according to the formula (1) and/or  $\theta$  tie breaking parameter, and to split the leaf nodes if necessary.

In the Experiments section we will show that as, in our approach, the computations of attribute estimation occur quite rarely - only after a larger number of samples are read and processed in parallel from the stream - the overall time of stream processing decreases (compared to dsCART) while the learned decision trees remain similar.

## 4 Experiments

This section summarizes several results of our experiments. As previously underlined, our parallel approach for decision tree learning from data streams is designed to achieve similar results as the dsCART algorithm. In fact, in all our tests, when running with a sufficiently high  $\alpha$  value and with the same test settings (except the number of records processed), we have obtained exactly the same decision trees with the same level of accuracy as with the dsCART implementation. Due to this fact, we do not need to benchmark the differences in accuracy between learned models. Instead, we would like to emphasize some theoretical and practical performance gains obtained by implementing our solution, by processing, in parallel, more than one record at once.

To do so, we first describe our experimental scenarios as well as the details of the datasets that we have considered, then we present the results of PdsCART implementation.

### 4.1 Experimental Scenarios

While it is intuitively that handling a larger number of data records in parallel reduces the processing time, several other aspects have been considered in order to validate our parallel approach. Some of the aspects that we have taken into consideration, besides the running time, include: the number of records to consider per iteration, the number of attributes and the number of bins<sup>2</sup>. Other aspects, such as the decision tree size, the  $\alpha$  parameter and the dependency relations between all these aspects may be considered for a future work as well.

**Table 1.** The specifications of the datasets used in our experiments.

#	Data Set	# records	Attributes	Classes	# type
1	$D_1$	10 thousands	5	2	synthetic
2	$D_2$	500 thousands	70	5	synthetic
3	$D_3$	1.5 millions	20	10	synthetic
4	$D_4$	4 millions	10	5	synthetic
5	$D_5$	4 millions	15	2	synthetic
6	$D^*$	4.8 millions	34	20	web data

All these different parameters do not act independently. To evaluate how they relate to each other, we have considered 5 synthetic and one real world web data sets. The synthetic ones were generated using MOA (Massive Online Analysis software environment [25]), while the web one is the KDD CUP 99' data set<sup>3</sup>,

<sup>2</sup> The standard method of dividing the range of numerical attributes values into bins.

<sup>3</sup> <http://archive.ics.uci.edu/ml> - for simplicity we have considered only the numerical attributes.



which was also used to benchmark the dsCART algorithm. Their characteristics are listed in Table 1. In all our tests, the  $\theta$  tie breaking parameter was not taken into consideration.

## 4.2 Experiment Results

A first set of results, presented in Table 2, shows the improvement of PdsCART running times, compared with dsCART (first line in the table), while producing exactly the same trees and accuracy.

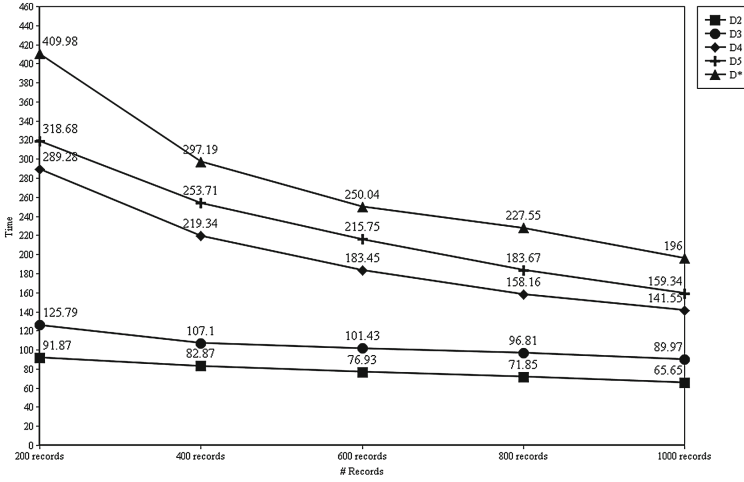
**Table 2.** Running time(ms) and accuracy for  $D_1, D_5, D^*$  datasets with different # of incoming records processed in parallel. #1 is the dsCART algorithm.

$D_1$			$D_5$			$D^*$		
Records	Accuracy	Time	Records	Accuracy	Time	Records	Accuracy	Time
1	83.11 %	11.44	1	84.94 %	5076.77	1	77 %	9106.35
20	83.11 %	2.10	200	84.94 %	318.68	200	77 %	409.98
40	83.11 %	1.56	400	84.94 %	253.71	400	77 %	297.19
60	83.11 %	1.35	600	84.94 %	215.75	600	77 %	250.04
80	83.11 %	1.23	800	84.94 %	183.67	800	77 %	227.55

Table 2 contains some of the our most important results that led to PdsCART algorithm. As we can see in the *Accuracy* column, even if the number of examples taken into consideration for computing the splitting points increases, the accuracy remains the same. This is exactly the property needed to realize the parallel algorithm. Using this characteristic, the execution time of PdsCART can be reduced as can be seen in *Time* column. This results are confirmed by other tests as well (see Fig. 2).

The faster processing times are obtained mainly because of the fewer splitting conditions computations. For example, for  $D_4$  dataset, having 4 millions records, when processing 200 records at time, there are 20 000 splitting computation steps, while when processing 800 records, there are only 5000.

While it might be tempting to choose a very large number of records to be processed in parallel, we have to underline here its side effect as well: the larger the number of records is, the later the algorithm will catch the decision tree changes / splits. Later refers here to the stream records occurrence time and not necessarily to the overall processing time. For example, for business applications that need to have a very responsive prediction model, where the changes in the stream need to be detected and processed immediately, a lower number of records would fit better; for other applications that can afford to have the same prediction model without caring about the amount of stream records consumed, larger number of records may conduct to better processing times.



**Fig. 2.** Running time(ms) per dataset with different # of incoming records processed in parallel.

Detecting the splitting points earlier may facilitate the computations, since the new leaves will have fewer records to take into account for the future splitting conditions computations. This may justify why the differences between processing times with larger number of records (1000 vs. 800) is significantly smaller than with fewer number of records (400 vs. 200).

**Table 3.** Running time(ms) per datasets having different # of attributes.

#	Data Set	TreeDepth	TreeNodes	TreeLeaves	# Attributes	Time
1	$D_{2a}$	2	3	2	2	19.59
2	$D_{5a}$	5	29	15	5	66.54
3	$D_{10a}$	10	213	107	10	259.651
4	$D_{20a}$	20	5501	2197	20	1610.75

However, in Fig. 2 we can see that while  $D_4$  and  $D_5$  synthetic datasets have the same number of records, the running times of  $D_4$  are considerable better than those of  $D_5$ , under the same tests settings. This is related this time with another aspect that it is worth mentioning: the number of attributes. Intuitively, more attributes a dataset has, more time it will be needed by the algorithm to process it. Table 3 shows the results obtained after processing 4 synthetic datasets, all of them having 4 millions of records but different number of attributes.

Another result is related to the splitting computing times. When checking or selecting the splitting attributes, the PdsCART algorithm has to consider all possible partitions of the set of attribute values. This is directly related to the

**Table 4.** Running time(ms) per dataset using histograms with different # of bins.

#	Data Set	2 bins	4 bins	6 bins	8 bins	10 bins
1	$D_2$	73.28	77.41	82.36	89.40	98.63
2	$D_3$	79.96	83.70	90.97	100.22	111.77
3	$D_4$	145.55	153.22	167.28	204.38	229.88
4	$D_5$	151.89	166.16	190.98	250.27	313.59

number of bins used in the histograms. As we can see in Table 4, the more bins a histogram has, the more time will be needed to evaluate all the partitions. This property may lead to another level of parallelization, where all partitions may be analyzed independently. We are considering this as a solid basis for our future work.

Although these are just a subset of all the experiments conducted, they should prove the potential of our algorithm.

## 5 Conclusion and Future Work

In this paper we have shown how to implement a decision tree learning algorithm in the MapReduce framework. The first achievement of this algorithm is to be able to produce the decision tree in one single pass over the data. This is crucial in the context of streaming data, where multiple passes over the same data set are very difficult or even impossible. A second important achievement is the performance of the implementation. We were able to show that the algorithm achieves very good results by treating in parallel a larger number of records.

This encouraging results provide a solid basis for the ongoing work. In particular it is necessary to analyze how the algorithm scales with an increasing number of processing units and in which way all other parameters are influencing the behavior of the algorithm. Performance-wise, the outcome is relatively easy to guess. However more work has to be done in order to asses the influence of these parameters regarding the quality of the decision trees. We know that we can achieve similar error rates as the other algorithms (e.g. C4.5). Some other parameters regarding the trees such as size, depth, order of attributes, will have to be further investigated as well.

## References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
2. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Chapman & Hall/CRC, New York (1984)
3. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)

4. Shafer, C., Agrawal, R., Mehta, M.: SPRINT: a scalable parallel classifier for data mining. In: *Proceedings of the 22th International Conference on VLDB*, pp. 544–555 (1996)
5. Joshi, M., Karypis, G., Kumar, V.: ScalParC: a new scalable and efficient parallel classification algorithm for mining large datasets. In: *Proceedings of the 12th International Parallel Processing Symposium*, pp. 573–579 (1998)
6. Sreenivas, M., Alsabti, K., Ranka, S.: Parallel out-of-core divide-and-conquer techniques with applications to classification trees. In: *The 10th Symposium on Parallel and Distributed Processing*, pp. 555–562 (1999)
7. Jin, R., Agrawal, G.: Communication and memory efficient parallel decision tree construction. In: *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM)*, pp. 119–129 SIAM, (2003)
8. Ben-Haim, Y., Tom-Tov, E.: A streaming parallel decision tree algorithm. *J. Mach. Learn. Res.* **11**, 849–872 (2010)
9. Srivastava, A., Han, E., Kumar, V., Singh, V.: Parallel formulations of decision-tree classification algorithms. *Data Min. Knowl. Discov.* **3**(3), 237–261 (1999)
10. Amado, N., Gama, J., Silva, F.: Parallel implementation of decision tree learning algorithms. In: Brazdil, P.B., Jorge, A.M. (eds.) *EPIA 2001. LNCS (LNAI)*, vol. 2258, pp. 6–13. Springer, Heidelberg (2001)
11. Panda, B., Herbach, J., Basu, S., Bayardo, R.: PLANET Massively parallel learning of tree ensembles with MapReduce. In: *Proceedings of VLDB-2009* (2009)
12. Ye, J., Chow, J.-H., Chen, J., Zheng, Z.: Stochastic gradient boosted distributed decision trees. In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pp. 2061–2064 (2009)
13. Tyree, S., Weinberger, K.Q., Agrawal, K., Paykin, J.: Parallel boosted regression trees for web search ranking. In: *Proceedings of the 20th International Conference on World Wide Web*, pp. 387–396. ACM (2011)
14. Li, B., Chen, X., Li, M.J., Huang, J.Z., Feng, S.: Scalable random forests for massive data. In: Tan, P.-N., Chawla, S., Ho, C.K., Bailey, J. (eds.) *PAKDD 2012, Part I. LNCS*, vol. 7301, pp. 135–146. Springer, Heidelberg (2012)
15. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: A new method for data stream mining based on the misclassification error. *IEEE Trans. Neural Netw. Learn. Syst.* **26**(5), 1048–1059 (2014)
16. Li, X., Barajas, J.M., Ding, Y.: Collaborative filtering on streaming data with interest-drifting. *Intell. Data Anal.* **11**(1), 75–87 (2007)
17. Domingos, P., Hulten, G.: Mining high-speed data streams. In: *Proceedings of the 6th ACM SIGKDD Conference*, pp. 71–80 (2000)
18. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 97–106 (2001)
19. Bifet, A., Holmes, G., Pfahringer, G., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: *Proceedings of the 15th ACM SIGKDD International Conference Knowledge Discovery and Data Mining* (2009)
20. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: *DATA STREAM MINING: A Practical Approach*. University of Waikato, New Zealand (2011)
21. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.* **58**, 13–30 (1963)
22. Rutkowski, L., Pietruczuk, L., Duda, P., Jaworski, M.: Decision trees for mining data streams based on the McDiarmid’s bound. *IEEE Trans. Knowl. Data Eng.* **25**, 1272–1279 (2013)

23. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: Decision trees for mining data streams based on the gaussian approximation. *IEEE Trans. Knowl. Data Eng.* **26**, 108–119 (2014)
24. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: The CART decision tree for mining data streams. *Inf. Sci.* **266**, 1–15 (2014)
25. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. *J. Mach. Learn. Res.* **11**, 1601–1604 (2010)