

Parallel univariate decision trees

Olcay Taner Yıldız^{a,*}, Onur Dikmen^b

^a Department of Computer Engineering, Işık University, Şile, İstanbul, Turkey

^b Department of Computer Engineering, Boğaziçi University, Bebek, İstanbul, Turkey

Received 8 May 2006; received in revised form 16 November 2006

Available online 19 January 2007

Communicated by J.A. Robinson

Abstract

Univariate decision tree algorithms are widely used in data mining because (i) they are easy to learn (ii) when trained they can be expressed in rule based manner. In several applications mainly including data mining, the dataset to be learned is very large. In those cases it is highly desirable to construct univariate decision trees in reasonable time. This may be accomplished by parallelizing univariate decision tree algorithms. In this paper, we first present two different univariate decision tree algorithms C4.5 and univariate linear discriminant tree. We show how to parallelize these algorithms in three ways: (i) feature based; (ii) node based; (iii) data based manners. Experimental results show that performance of the parallelizations highly depend on the dataset and the node based parallelization demonstrate good speedups.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Decision trees; Parallel processing; Univariate decision trees; Linear discriminant trees

1. Introduction

Univariate decision trees are one of the most widely used classification model in data mining. First ID3 algorithm based on discrete features appeared then in C4.5 (Quinlan, 1993) it is expanded to include continuous features. Constructing a univariate decision tree has time complexity of roughly $O(dfN \log N)$, where N is the total sample size, f is the number of features and d is number of nodes in the tree. In data mining applications, the sample size tends to be very large. So constructing decision trees in parallel manner becomes an important fact.

Parallel construction of univariate decision trees can be divided into two groups. Proposing parallel decision tree algorithms and parallel formulations of existing algorithms.

SLIQ (Mehta et al., 1996) is a univariate parallel decision tree classifier that can handle both numeric and con-

tinuous attributes. By using a pre-sorting technique in a breadth-first tree-growing phase, it is able to classify disk-resident datasets. SLIQ also introduced a new tree-pruning algorithm. Since pruning phase of the decision trees takes much smaller time than tree growing phase, we will not consider the time used in tree pruning in our discussions.

SPRINT (Shafer et al., 1996) extends the idea of SLIQ by addressing the problems with it. Although SLIQ uses the entire dataset to build the tree, it requires data, which is directly proportional to the number of features in the dataset to stay in memory all the time. This limits the amount of data that can be classified by SLIQ. SPRINT is intended to remove these memory restrictions.

Kufirin (1997) proposed a data distributed parallel formulation of C4.5 algorithm. He mentioned that, since we only use sorting to gather the frequency statistics from data, we can also perform concurrent sorts on each processor. Frequency statistics for each local candidate split point is evaluated and shared with other processors to get the best split at each node.

* Corresponding author. Tel.: +90 216 3341508; fax: +90 216 7102872.
E-mail address: olcaytaner@isikun.edu.tr (O.T. Yıldız).

Jin et al. (2003) proposed a new decision tree construction algorithm called SPIES, in which the number of possible split points is limited by taking a sample from the data set, partitioning the values into intervals and computing the class histograms for candidate split points. This reduces the space complexity of the algorithm and the communication cost between processors. They parallelized this algorithm using the FREERIDE framework and obtained nearly linear speedups.

Srivastava et al. (1999) described two parallel formulations in their paper. Synchronous tree construction approach and partitioned tree construction approach. They discuss the advantages and disadvantages of these two approaches and propose a hybrid methodology. In synchronous tree construction approach, all processors construct a decision tree synchronously by sending and receiving class distribution information of local data. In partitioned tree construction approach, different processors work on different parts of the classification tree, whenever feasible. The hybrid approach they proposed, starts and continues with the first approach as long as the communication cost of the approach is not high. Once this cost is high, they switch to the second approach.

In this paper we give and compare the performances of three different formulations of two different univariate decision tree construction algorithms. C4.5 and the univariate version of linear discriminant trees (Yildiz and Alpaydin, 2005). In Section 2 we give serial formulations of these two algorithms. Three different parallel formulations are explained in Section 3. Hardware and software platforms used in the experiments and the descriptions of the datasets are detailed in Section 4. Section 5 presents experiments and discussion on the results of these experiments. We conclude in Section 6.

2. Serial formulations

In this section, we explain two univariate decision tree algorithms C4.5 and LDT. Since each algorithm does the same job at each decision node, we give the algorithms for a single node n . Each node works on an instance space

of x . Each instance of the space x has f features, which can be discrete or continuous.

If the best split is found at node n , both algorithms create two or more child nodes. The instances of node n are also distributed to child nodes according to the best split. Fig. 1 gives the pseudocode of this child node creation.

2.1. C4.5

Original serial C4.5 algorithm is given in Fig. 2. Finding best split differs in discrete and continuous features. There is one possible split for discrete feature, whereas in continuous features there are as many split points as the number of samples in that node. The split points are compared according to the information gain they provide (Quinlan, 1986). Quinlan takes the famous entropy formula of infor-

```

C45(Node  $n$ , Instances  $x$ , Features  $f$ )
  for each feature  $i$  in  $f$ 
    if  $f_i$  discrete
      Calculate information gain  $g_i$ 
      if  $g_i < \text{bestgain}$ 
        bestgain =  $g_i$ 
        bestfeature =  $i$ 
      endif
    endif
  endif
  if  $f_i$  continuous
    Sort  $x$ 
    for each different value  $j$  of  $f_i$ 
      Calculate information gain  $g_{ij}$ 
      if  $g_{ij} < \text{bestgain}$ 
        bestgain =  $g_{ij}$ 
        bestfeature =  $i$ 
      endif
    endfor
  endif
endif
CREATE_CHILDREN( $n$ ,  $x$ , bestfeature)
Call C45 for each child node

```

Fig. 2. C4.5 algorithm.

```

CREATE_CHILDREN(Node  $n$ , Instances  $x$ , bestfeature)
  if bestfeature discrete
    Partition instances  $x$  into  $k$  groups,  $k$  is the
    number of all possible values of the bestfeature
    Split node  $n$  into  $k$  child nodes
  endif
  if bestfeature continuous
    Partition instances  $x$  into 2 groups
    Split node  $n$  into 2 child nodes
  endif
endif

```

Fig. 1. Creating child nodes.

mation theory, which is the minimum number of bits to encode the classification of an arbitrary member of a collection S . The information gain of node n is

$$\sum_{i=1}^c -p_i \log_2 p_i, \quad (1)$$

where p_i is the occurring probability of class i at node n . The information gain of a split is then calculated by taking the weighted average of the information gains of the child nodes.

By sorting with respect to the values of the instances for a feature, we can evaluate all possible split points using the information gain formula in one pass. Since sorting values takes $O(N \log N)$ by using Quicksort algorithm and making one pass to evaluate all possible split points takes $O(N)$ time, the bottleneck of the C4.5 algorithm is the sorting phase.

2.2. LDT

Pseudocode of the LDT algorithm is given in Fig. 3. This algorithm works the same as C4.5 for discrete features. For continuous features, finding the best split via Fisher's linear discriminant analysis (LDA) (Duda et al., 2001) is done as a nested optimization problem. In the inner optimization problem, Fisher's linear discriminant finds a good split for the given two distinct groups of classes. In the outer optimization problem, *exchange method*

(Yıldız and Alpaydın, 2005) is used to divide K classes into two groups.

Assuming the data is normally distributed, one dimensional LDA reduces to a second order equation

$$ax^2 + bx + c = 0 \quad (2)$$

and the two candidate best split points are the roots of that equation, where

$$\begin{aligned} a &= s_L^2 - s_R^2, \\ b &= 2(m_L s_R^2 - m_R s_L^2), \\ c &= (m_R s_L)^2 - (m_L s_R)^2 + 2s_L^2 s_R^2 \log \frac{n_L s_R}{n_R s_L}, \end{aligned} \quad (3)$$

m_L , m_R are the means and s_L , s_R are the standard deviations of the feature i of the left and right class groups. n_L and n_R are the number of data that are assigned to the left and right class groups, respectively.

If the two groups have the same variance, there is only one root. If the variances are different, there are two roots and the one which is between the two means is used. If neither of the two roots is between the means nor there are no roots of the quadratic equation, the middle point of two means is chosen as the split point.

Since we need one pass over the data to find the mean and variance of two groups, complexity of the LDT algorithm is $O(N)$ for inner optimization. If we make l passes in the outer optimization problem, the complexity of finding best split for a single feature will be $O(lN)$.

3. Parallel formulations

3.1. Feature based parallelization

For each feature, we do the same operations to find the best split, therefore we can easily parallelize the job at each node by distributing the features to the slave processors. Fig. 4 gives the pseudocode of this idea. We first send the features and the data they will process to the slave processors. In the slave processors part, they find the best splits (s_i) and the best gains (g_i) for each feature f_i . After those

```
LDT(Node  $n$ , Instances  $x$ , Features  $f$ , Classes  $c$ )
  for each feature  $i$  in  $f$ 
    if  $f_i$  discrete
      Calculate information gain  $g_i$ 
      if  $g_i < \text{bestgain}$ 
        bestgain =  $g_i$ 
        bestfeature =  $i$ 
      endif
    endif
    if  $f_i$  continuous
      Find best split point using LDA
      and exchange method
      Calculate information gain  $g_i$ 
      if  $g_i < \text{bestgain}$ 
        bestgain =  $g_i$ 
        bestfeature =  $i$ 
      endif
    endif
  endfor
  CREATE_CHILDREN( $n$ ,  $x$ , bestfeature)
  Call LDT for each child node
```

Fig. 3. LDT algorithm.

```
FParallel(Node  $n$ , Instances  $x$ , Features  $f$ )
  for each feature  $i$  in  $f$ 
    Submit  $f_i$  and  $n$  to slave processor
  endfor
  for each feature  $i$  in  $f$ 
    Receive best split  $s_i$  and gain  $g_i$ 
    of feature  $f_i$  from slave processor
  endfor
  bestfeature = arg min $_i g_i$ 
  bestsplit =  $s_{\text{bestfeature}}$ 
  CREATE_CHILDREN( $n$ ,  $x$ , bestfeature)
  Call FParallel for each child node
```

Fig. 4. Feature based parallelization.

calculations, the best splits and information gains are sent to the host processor, where by taking the smallest of these best information gains, we can find the overall best split.

The advantage of feature based parallelization is its simple implementation. Since we have the serial codes of LDT and C4.5 for each feature, we can easily plug that code into the slave processors to find best split for one feature.

According to the dataset we have, the performance of feature based parallelization may change. For example, if we have a dataset that contains only discrete features, both LDT and C4.5 reduce to the same algorithm. Since finding the information gain of a discrete feature has a time complexity of $O(N)$, the load balance will be good.

If there are two classes in the dataset, we do not need a method to divide classes into two groups. So $l = 1$ (number of iterations of exchange method) and LDT will have a time complexity of $O(N)$ for that feature. In that case, processing discrete and continuous features will have the same complexity and slave processors will have same load. In the case of continuous features, C4.5 will have the same load at each continuous feature but has a significantly large load over discrete features ($O(N \log N)$ instead of $O(N)$). If there are more than two classes in the dataset, the time complexity of finding best split for a continuous feature will be $O(IN)$. Since dividing classes into two class groups may differ from one feature to another feature, the load of slave processors will be imbalanced.

3.2. Node based parallelization

As explained above, decision tree construction is done recursively at each decision node. So why not distribute decision node's to processors? Fig. 5 shows the pseudocode for node based parallelization. We use a queue for handling current unexpanded nodes. If there are nodes in the queue, we dequeue them from the queue and send to the slave processor(s) to find the best split for that node. Since LDT and C4.5 algorithms are defined for a single node, each slave processor can call those serial codes to find the best split for that node. After expanding the node(s), produced child nodes are put into the queue to be processed later. Algo-

rithm terminates, when there are no nodes in the queue to be expanded.

Node based parallelization requires the smallest time for communication between host and slave processors. For a single node there is only one message passing, in which host processor sends the instances for that node. Like the feature based parallelization, its implementation is simple, only queue processing should be handled.

If the decision tree is small, processing time of the root node takes a significant amount of time compared to the whole tree generation. In that case, since we only have one processor job in the starting phase all but one processors will wait the single processor to complete its process. Second disadvantage of the node based parallelization is the complexity of finding the best split significantly drops in the deeper nodes, because small instances sets come to those nodes. In such cases, the load imbalance may occur between slave processors.

3.3. Data based parallelization

We can also divide data into K parts, where we have K processors. At each node of the decision tree, for each feature f_i , we send divided data to the corresponding processors. The slave processors handle the data and return frequency statistics to the host processor. Since the continuous feature phase of LDT and C4.5 algorithm differs, we need different parallelizations for two algorithms. The pseudocodes of these two parallelizations are given in Figs. 6 and 7.

In the data parallel version of C4.5, first the instances in the slave processors are sorted in $O(\frac{N}{k} \log(\frac{N}{k}))$ time using Quicksort. After sorting, each slave processor sets its iterator to show the first instance. At each step, slave processors send the values their iterators point to, to the host processor, which selects the minimum of them as the split point. Host processor sends the split point to slave processors. After getting the split point, slave processors send frequency statistics of this split point to the host processor. By frequency statistics, we mean total number of elements, those have a feature value smaller or equal to the split

```

NParallel(Instances  $x$ , Features  $f$ )
  Queue  $q = \text{Emptyqueue}$ 
  Enqueue( $q$ , RootNode)
  while (Not Empty( $q$ ))
    Node = Dequeue( $q$ )
    Submit Node with its instances to slave processor
    Receive best split and bestfeature
    of Node from slave processor
    CREATE_CHILDREN(Node, Node.instances, bestfeature)
    Enqueue( $q$ , Node.child nodes)
  endwhile

```

Fig. 5. Node based parallelization.

```

DParallelC45(Instances  $x$ , Feature  $f_i$ )
  Send  $x$  and  $f_i$  to each slave processor
  Sort  $x$  in each slave processor
  Receive possible split points from slave processor(s)
  Determine the minimum of them
  Send the minimum to the slave processor(s)
  Receive the frequency statistics from slave processor(s)
  Slave processor(s) update their iterator(s) if necessary
  Calculate information gain  $g_i$  for that split point
  from the gathered frequency statistics
  Compare  $g_i$  with bestgain and update if former is better
  Goto above to receive again possible split points

```

Fig. 6. Data based parallelization of C45.

```

DParallelLDT(Instances  $x$ , Feature  $f_i$ )
  Send  $x$  and  $f_i$  to each slave processor
  for each possible class partition  $C_L, C_R$ 
    Send class partition to the slave processor(s)
    Receive frequency statistics of that partition
    Determine the performance of the partition
  endfor
  Find the best split point using best class partition
  Send the split point to slave processor(s)
  Receive frequency statistics from slave processor(s)
  Find information gain of the feature  $f_i$  using
  gathered frequency statistics

```

Fig. 7. Data based parallelization of LDT.

point, for each class. By using gathered frequency statistics, host processor can now determine the goodness of the split. Before continuing with the next step, each slave processor updates its iterator such that it points to the next feature value.

Fig. 8 shows a sample execution of parallel version of LDT using 3 slave processors. One important point is that we do not have to sort instances in slave processors in LDT. In step I, each slave processor sends the sum of its feature values of each class to the host processor. In step II, using these sums, host processor easily finds the means of the left and right class groups. The means are sent to the slave processors. In step III, using group means, slave processors find the sum of squares of the differences between the feature values and means. The sums are sent to host processor. In step IV, using the sums, host processor first calculate the standard deviations, coefficients of the second order equation, and roots of that equation. With the roots of the equation, LDT finds the best split point and sends it to the slave processors. In step V, slave processors calculate the frequency statistics for the best split point and send them to host processor. In step VI, host processor finds the goodness of the split using gathered frequency statistics.

The advantage of data based parallelization is its potential for scalability. The datasets in Data Mining usually have large number of instances. Distributing these instances to slave processor(s) equally, makes data base parallelization scalable. Feature based parallelization needs large number of features in a dataset to scale well, which does not occur frequently in data mining applications. Node based parallelization needs a large tree to scale well.

The communication cost of data based parallelization is much higher compared to both feature based parallelization and node based parallelization.

Second disadvantage of data based parallelization is load imbalance between slave processor(s). Even though each processor started with the same number of training instances at the root node, in deeper nodes the number of training instances belonging to the nodes can vary substantially among processors. For example, processor 1 might have all training instances of node A, whereas none of node B; processor 2 might have all training instances of the node B, whereas none of node A. When A is selected to expand, processor 1 will do all the job and processor 2 will do nothing and similarly when B is selected to expand, processor 2 will do all the job and processor 1 will do nothing.

4. Experimental details

4.1. Specifications

In our experiments we use a Beowulf cluster with 24 processors (Agayev et al., 1999). Each node of the cluster has a Pentium II 400 Mhz processor with 512 KB Cache, 128 MB RAM and an Intel EtherExpress Pro/100+ Network Adapter. All the nodes are connected on HP Procurve 4000M Fast Ethernet Switch. We have Linux 2.2.12 Kernel and Redhat 6.1 distribution on each node. The parallel programs are coded using MPI parallel programming library (Message Passing Interface Forum, 1994) and compiled using GNU C compiler.

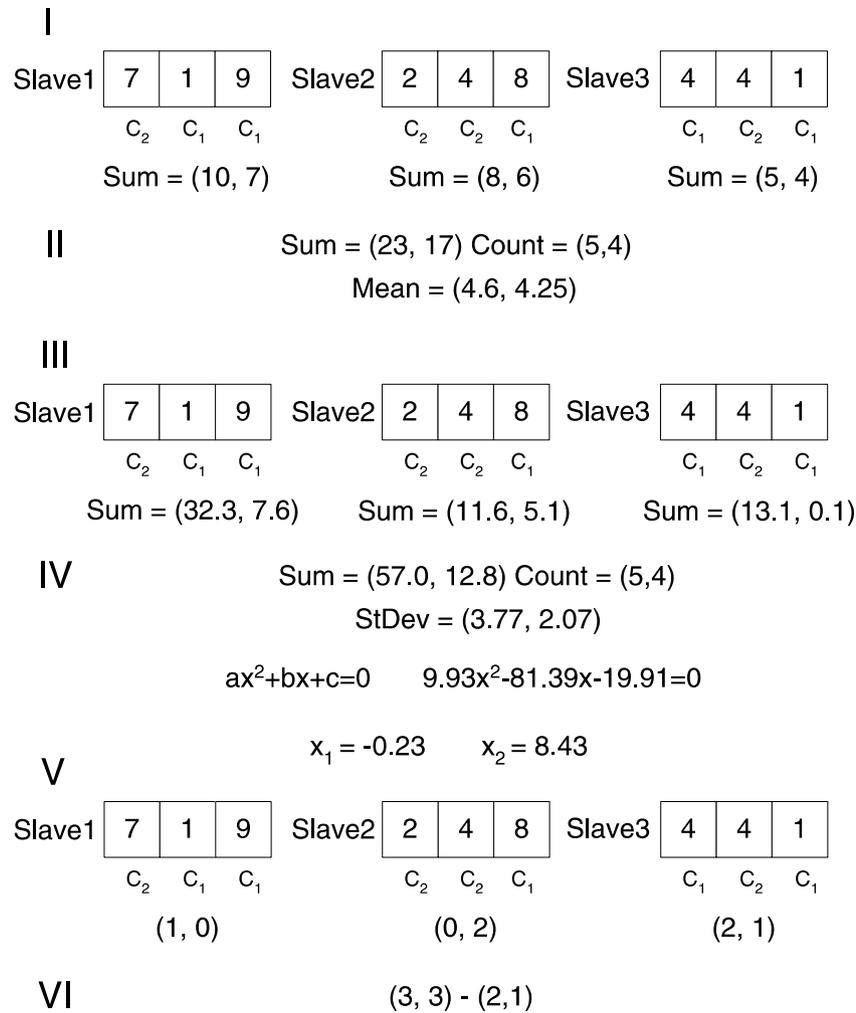


Fig. 8. Sample execution of data parallelization of LDT.

4.2. Datasets

Since the datasets in UCI Machine Learning Repository are small either in sample size or feature size, we get three different datasets from three different resources. Attributes of the datasets are given in Table 1. To test all three types of parallelizations, for feature based, we get a dataset with a large number of attributes (*Face*), for node based, we get a dataset with a large tree size (*Aibo*), for data based we get datasets with large number of data (*Aibo*, *Census*). Since LDT behaves differently according to the number of classes, we take one dataset with two classes (*Census*) and

two others with large number of classes (*Aibo*, *Face*). The performance difference between continuous and discrete features will be tested on the (*Census*) dataset, which has continuous and discrete features.

5. Results

In our experiments, we want to check the speedup of the three types of parallelizations of LDT and C4.5. A linear speedup curve is the best possible output, meaning the computational load is perfectly distributed among the processors and adding new processors to the system results in the expected gain. LDT and C4.5 algorithms are parallelized as node based, feature based and data based manner. We expect that on a data set with high dimensionality, feature based parallelization would have the highest speedup because the load would be balanced among slave processors. Similarly, data based parallelization would give the best results on a data set with too many instances, and node based parallelization, on a data set which has the largest decision tree. The ideal case of linear speedup curve mentioned above is obtained when the load balancing is

Table 1
Descriptions of the datasets

Set	Class	Size	Discrete	Continuous	Missing
Aibocolor (Akin et al., 2002)	8	250000	0	3	No
Census (Newman et al., 1998)	2	100000	33	6	Yes
Face (ORL Face Database)	40	400	0	4096	No

perfect and the communication burden between the processors is minimum.

For each of these six types of parallelizations, we performed simulations using 1 master processor and a range of slave processors from 1 to 7. The master processor distributes the jobs and collects the statistics, i.e. does no actual data processing. Every processor has its own copy of the data set and the master processor only sends the indices of the instances when it needs to distribute data to the slaves. We record the time to create a decision tree. Since one run will not be appropriate, we made 10-fold cross-validation for each experiment and took the average time. For feature based parallelization, since *Aibo* dataset has only three features, we used maximum three slave processors for that dataset.

Fig. 9 shows the speedup curves of different parallelizations on *Aibocolor* dataset. Since *Aibocolor* dataset has three features, Feature Based parallelizations can be done at max for three slave processors. Here C4.5 has larger speedup than LDT. Since *Aibocolor* dataset has more than two classes, at different features LDT may work differently (*l* values may differ because of exchange method). On the other hand, C4.5 will behave similarly at each feature. NodeBased Parallelization of LDT has the largest speedup on this dataset.

Fig. 10 shows the speedup curves of different parallelizations on *Face* dataset. *Face* dataset has many features and the tree generated is not so large, because of this, Feature Based parallelizations have better speedups than Node Based parallelizations. Like above, the number of classes are more than 2, so LDT needs exchange method in separating those classes into two optimal subgroups. For each feature, the exchange method is done separately. Therefore the number of exchanges so the load will be different than the other features. Because of this load imbalance, Feature

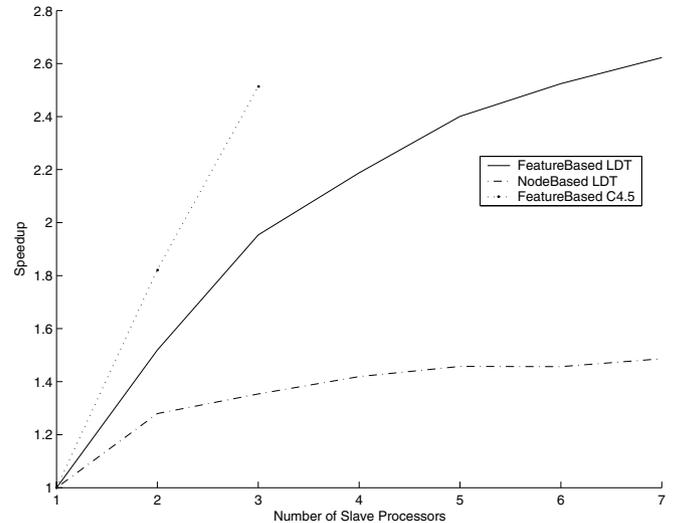


Fig. 10. The speedup curves on *Face* dataset for feature based and node based parallelizations of LDT and feature based parallelization of C4.5.

Based parallelization of LDT has worse speedup than Feature Based parallelization of C4.5.

Fig. 11 shows the speedup curves of different parallelizations on *Census* dataset. *Census* dataset is a mixed type of dataset, 33 of the features are discrete and six of the features are numeric. In LDT, finding the best split for a discrete feature is very different from finding the best split for a numeric feature in terms of cost and adding to this, in Feature Based parallelization, the features are distributed among the processors (some of them may get a discrete feature, some of them may get a numeric feature). Therefore, the load is imbalanced among processors and Feature Based parallelization shows bad speedups in *Census* dataset. On the other hand, the number of nodes generated in *Census* dataset is very large, so NodeBased Parallelization of LDT has larger speedup on this dataset.

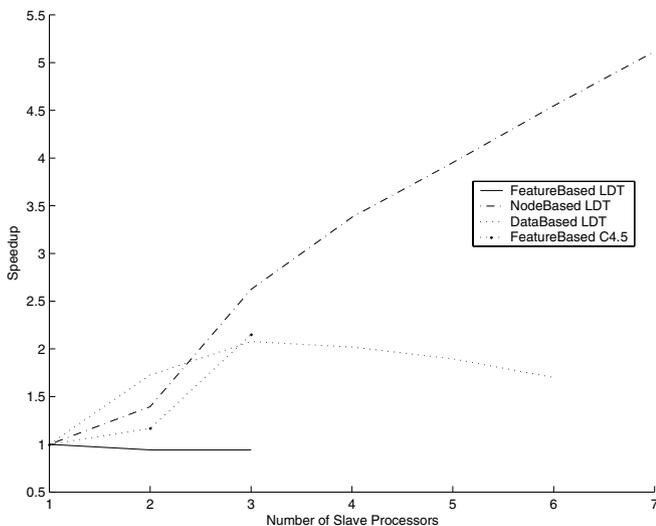


Fig. 9. The speedup curves on *Aibocolor* dataset for feature based, node based and data based parallelizations of LDT and feature based parallelization of C4.5.

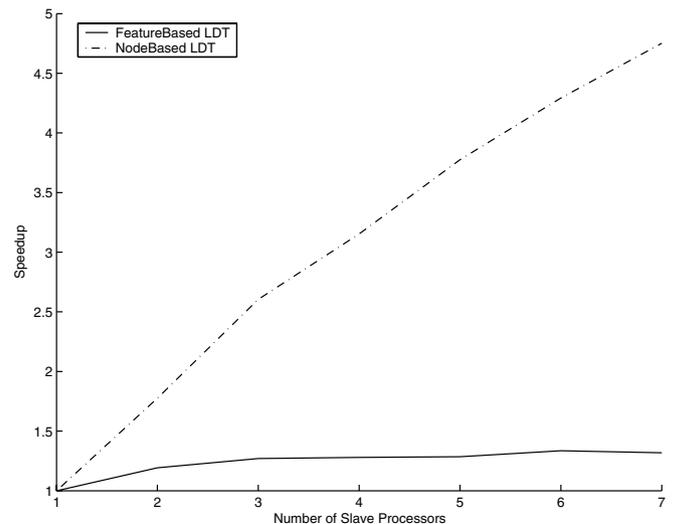


Fig. 11. The speedup curves on *Census* dataset for feature based and node based parallelizations of LDT.

6. Conclusion

Recently, as the data mining applications became more widespread and the processing power of computers increased significantly, the expectations from data mining algorithms grew high accordingly. The algorithms require to process large data sets very quickly, mostly in real-time. For example, biometric systems, with the high attention they draw in recent years, have to be fast and robust. They need to be trained on large data sets containing diverse and noisy conditions. These data sets also have very high dimensionality.

In this paper, we proposed parallel implementations of two univariate decision tree algorithms (C4.5 and linear discriminant tree). The algorithms are parallelized by distributing the features, the data or the nodes among the slave processors. We presented detailed complexity analyses for these implementations. Theoretically, feature based parallelization would have a high speedup on a data set with high dimensionality, data based parallelization on a data set with too many instances and node based parallelization on a data set which has a tree with a high number of nodes when discriminated by the serial decision tree. But, this requires perfect load balancing. In our simulations, we roughly observed this tendency, but which parallelization performs the best on a data set and how much speedup it demonstrates depend on how well the load is distributed among slave processors. That depends on the data set.

Acknowledgement

We thank to Burak Gürdağ for fruitful discussions on parallel computing and his help in using Asma.

References

- Agayev, E., Gürdağ, B., Özturan, C., Çağlayan, M.U., 1999. Development of a low cost network based multicomputer. In: Proceedings of 4th Symposium on Computer Networks, Istanbul, pp. 53–60.
- Akın, H.L., Pavlova, P., Yıldız, O.T., 2002. Cerberus 2002, Robocup 2002: Robot soccer world cup VI. In: The 2002 International Robocup Symposium Pre-Proceedings, Fukuoka, June 24–25, 2002, pp. 448.
- Duda, R.O., Hart, P.E., Stork, D.G., 2001. Pattern Classification. Wiley-Interscience Publication, New York.
- Jin, R., Agrawal, G., 2003. Communication and memory efficient parallel decision tree construction. In: Proceedings of Third SIAM Conference on Data Mining.
- Kufirin, R., 1997. Decision trees on parallel processors. In: Geller, J., Kitano, H., Suttner, C.B. (Eds.). In: Parallel Processing for Artificial Intelligence, vol. 3. Elsevier Science.
- Mehta, M., Agrawal, R., Rissaneh, J., 1996. SLIQ: A fast scalable classifier for data mining. In: Proceedings of the Fifth International Conference on Extending Database Technology, Avignon. France.
- Message Passing Interface Forum, 1994. MPI: A Message-Passing Interface Standard.
- Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J., 1998. UCI Repository of machine learning databases, [<http://www.ics.uci.edu/mllearn/MLRepository.html>], Irvine, CA: University of California, Department of Information and Computer Science.
- ORL Face Database, <<http://www.uk.research.att.com/facedatabase.html>>.
- Quinlan, J.R., 1986. Induction of decision trees. Machine Learning 1, 81–106.
- Quinlan, J.R., 1993. C4.5: Programs for Machine Learning San Mateo. Morgan Kaufmann, CA.
- Shafer, J., Agrawal, R., Mehta, M., 1996. SPRINT: A scalable parallel classifier for data mining. In: Proceedings of the 22nd VLDB Conference.
- Srivastava, A., Han, E., Kumar, V., Singh, V., 1999. Parallel Formulations of Decision-Tree Classification Algorithms Data Mining and Knowledge Discovery. 3, 237–261.
- Yıldız, O.T., Alpaydın, E., 2005. Linear discriminant trees. International Journal of Pattern Recognition and Artificial Intelligence 19 (3), 323–353.