MULTIVARIATE DECISION TREES FOR MACHINE LEARNING

by

Olcay Taner Yıldız

B.S. in CmpE., Boğaziçi University, 1997

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
The requirements for the degree of
Master of Science
In
Computer Engineering

Boğaziçi University

2000

MULTIVARIATE DECISION TREES FOR
MACHINE LEARNING

APPROVED BY:

Assoc Prof. Dr. Ethem Alpaydın .....
(Thesis Supervisor)

Assoc Prof. Dr. H. Levent Akın .....

Prof. Dr. Aytül Erçil .....

DATE OF APPROVAL: 27-12-1993

ACKNOWLEDGMENTS

I would like to thank Assoc. Prof. Ethem Alpaydın for his supervision in this MS study and for his support and advises. I am also grateful to all my friends, my family, my teachers and my love. Without their support, this thesis would have not been accomplished.

ABSTRACT

In this thesis, we detail and compare univariate, linear and nonlinear decision tree methods using a set of simulations on twenty standard data sets. For univariate decision tree methods, we have used the ID3 algorithm and for multivariate decision tree methods, we have used the CART algorithm. For linear and nonlinear methods, we have used neural networks at each decision node. We also propose to use the LDA algorithm in constructing linear multivariate decision trees.

Univariate decision trees at each decision node consider the value of only one feature leading to axis-aligned splits. In a linear multivariate decision tree, each decision node divides the input space into two with an arbitrary hyperplane leading to oblique splits. In a nonlinear one, a multilayer perceptron at each node divides the input space arbitrarily, at the expense of increased complexity. We propose hybrid trees where the decision node may be linear or nonlinear depending on the outcome of a statistical test on accuracy. We also propose to use linear discriminant analysis at each decision node.

Our results indicate that if the data set is small and has few classes, then a univariate technique does not overfit and can be sufficient and the univariate ID3 has better performance than multivariate linear methods. ID3 learns fast, learns simple and interpretable rules.

If the variables are highly correlated, then the univariate method is not sufficient and we may resort to multivariate methods. We have shown that ID-LDA has better performance than CART in terms of accuracy, node size and very significantly in learning time. It has also smaller learning time than ID-LP and the same accuracy. ID-LDA generates smaller trees than ID3 and CART. This shows that to generate a linear multivariate tree, using ID-LDA is preferable over CART, and may be preferable over ID-LP if learning time is critical.

ÖZET

Bu tezde, tek değişkenli, doğrusal, ve doğrusal olmayan çok değişkenli karar ağaç kurma metodları karşılaştırıldı. Tek değişkenli karar ağaçları için örnek olarak ID3, çok değişkenli karar ağaçları için CART yöntemi kullanıldı. Doğrusal ve doğrusal olmayan metodlar içinse karar düğümünde değişik sinir ağları yapılarından faydalanıldı. Ayrıca Fisher'in doğrusal ayırma analizinin çok değişkenli karar ağacı oluşturulmasında kullanılması önerildi.

Tek değişkenli karar ağaçları her karar düğümünde tek değişkenin değerine bakarak eksenlere dik bölmeler yaparlar. Doğrusal karar ağaçlarında ise her dalda giriş uzayı rasgele bir düzlemle bölünür. Doğrusal olmayan karar ağaçlarında ise çok katmanlı sinir ağları giriş uzayını rasgele böler. Bu tezde melez ağaçlar önerildi. Bu ağaçlarda karar düğümü doğrusal veya değildir. Kararın doğrusal olup olmamasına ise bir istatistik testinin sonucuna bakılarak karar verilmektedir. Doğrusal ayırma analizi ile doğrusal çok değişkenli karar ağaçları yapay sinir ağı temelli karar ağaçlarından çok daha hızlı öğrenilir.

Sonuçlarımız gösteriyor ki, eğer veri kümesi küçük ve bu kümede az sınıf varsa, tek değişkenli metod yeterli olabilir ve ID3 çok değişkenli metodlardan daha iyi performans gösterir. ID3 hızlı ve kolay öğrenir, kuralları da kolayca yorumlanabilir.

Eğer değişkenler birbiriyle çok ilişkiliyse, tek değişkenli metod yeterli olmayabilir ve çok değişkenli metodları kullanabiliriz. Bu tezde gösterildi ki ID-LDA metodu CART'tan daha başarılı, daha küçük ağaçlar üretmekte ve daha az zaman harcamaktadır. Aynı zamanda ID-LP den daha hızlı ve eşit başarılıdır. ID-LDA ID3 ve CART'tan daha küçük ağaçlar üretir. Bu da gösteriyor ki çok değişkenli ağaç üretmek için ID-LDA CART'a göre tercih edilebilir ve eğer zaman önemli ise ID-LP'ye gore de tercih edilebilir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

n	the number of instances in a node
x_i	attribute i of instance
x^t	instance t
d^t	desired output for instance t
y^t	real output for instance t
t	a decision node
m	number of splits in a node
x_{ij}	possible value j of an unordered feature i
f	the number of features for a data set
C_i	class i in a node
C_L	classes in the left branch
C_R	classes in the right branch
p_i	merit value of partition i
c	the number of classes in a node
l	the number of nodes in the tree
T	an unpruned tree T
T'	a pruned tree T
w_i	weight of the feature i

1. INTRODUCTION

Machine learning aims at determining a description of a given concept from a set of concept examples provided by teacher and from the background knowledge. Concept examples can be positive or negative. Background knowledge contains the information about the language used to describe the examples and concepts. For instance, it can include possible values of variables and their hierarchies or predicates. The learning algorithm then builds on the type of examples, on the size and relevance of the background knowledge, on the representational issues (Mitchell, 1996).

Having explained the principles of machine learning, we can proceed with the decision tree construction method, which will be discussed in this thesis. The essence of this method is very simple. The entire set of examples is split into subsets that are easier to handle (Mitchell, 1996). The type of the split will determine type of the decision tree. The split can be based on one feature (Quinlan, 1986) or a linear combination of features (Breiman et. al., 1984).

The goodness of the split is based on a criterion called the partition-merit criterion. In the literature several experiments are done to compare these criteria (Brodley and Utgoff, 1995) (Mingers, 1989). We will also discuss three of them (Dietterich et. al., 1996) (Breiman et. al., 1984) (Quinlan, 1986).

In this thesis, we will discuss also other aspects of decision tree construction such as multiple splits, filling in missing values (Quinlan, 1989) and avoiding overfitting, using different pruning techniques. Although several experiments (Breiman et. al., 1984) (Quinlan, 1993) show that earlier pruning can decrease performance, we will compare pre-pruning and post-pruning.

After univariate methods, we will continue with multivariate methods. The methods can be classified by the method they use to find the contribution of the attributes. They can be heuristic (Breiman et. al., 1984) or can be based on other machine learning methods (Guo and Gelfond, 1992). We will also propose a statistical method to construct the tree (Duda and Hart, 1973) (Rencher, 1995).

In Chapter 2, we will discuss decision tree learning and issues in decision tree construction. We will also see a univariate algorithm ID3.

In Chapter 3, we give the drawbacks of univariate algorithms and show how the multivariate methods work introducing the CART algorithm.

In Chapter 4, we discuss neural trees, which are decision trees using neural networks at each node. We will also propose a hybrid algorithm.

In Chapter 5, we propose a statistical method to construct multivariate decision tree and discuss problems in this statistical method and how to get rid of them.

Chapter 6 gives simulation results. In this chapter, we see the results of several aspects of decision tree learning, comparing different types of decision trees on twenty standard data sets based on accuracy of classification, the size of the constructed decision tree and learning time.

We conclude and propose future work in Chapter 7.

2. DECISION TREE LEARNING

In machine learning the knowledge is extracted from a training sample for future prediction. Most machine learning methods make accurate predictions but are not interpretable. In this study, we concentrate on decision trees, which are simple and easily comprehensible. They are robust to noisy data and can learn disjunctive expressions (Mitchell, 1996).

A decision procedure inferring from an example space can be formalized as follows: The given data represents a set of objects or "instances". Each object is described in terms of a collection of discrete or continuous valued independent variables or "attributes" x_i. Each instance has a dependent variable or a "class" associated with it. The data consists of vectors that give values for the attributes and the class of each object ($x_1, x_2, \ldots x_f \in x$).

The object of supervised learning is to find the function of the attributes that best predicts the class of an object (a function F: $X_1 \times X_2 \times \ldots \times X_f \rightarrow C$). The given data is considered as the *training set* and the data that will be predicted is called the *test set*.

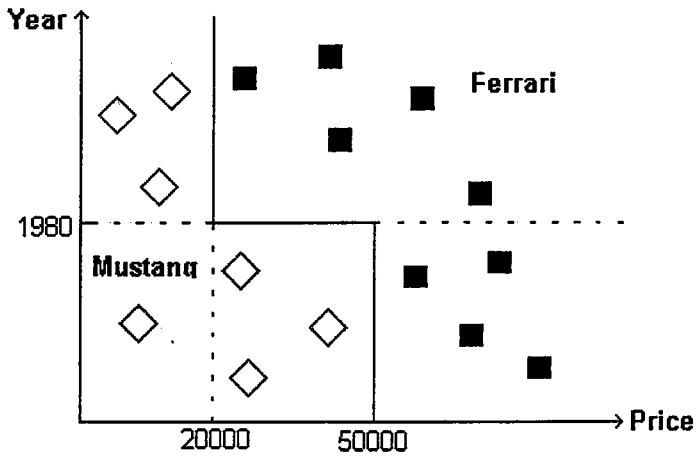

FIGURE 2.1. Instances of the problem *Choosing Car* (This is an imaginary data set).

An example data (training set) for the problem *Choosing Car* is shown in Figure 1.1. *Year* of the car and *price* are the two continuous attributes of this example. Both of the attributes are continuous. There are 16 instances in this data set. Seven of them are for the class *Mustang* (shown as diamonds) and nine of them are for the class *Ferrari* (shown as squares). Data is splitted by axis-aligned lines, which define the decision tree.

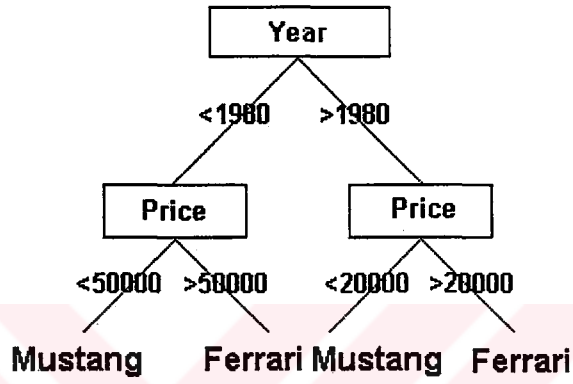

FIGURE 2.2. Decision tree for the problem *Choosing Car*

The constructed decision tree is shown in Figure 2.2. Decision trees consist of internal nodes (drawn as rectangles in Figure 2.2) having one or more attributes to test and leaves (drawn as *Mustang* or *Ferrari* in Figure 2.1) to show the decision made. The test for the internal nodes is shown as lines below the rectangles. For example the test *Year = 1980* line divides the data space in Figure 2.1 into two parts as:

- *Year* $\leq$ *1980* : Diamonds (*Mustang*), squares (*Ferrari*) and
- *Year* > *1980*: Diamonds (*Mustang*), squares (*Ferrari*).

The test for the data below the *Price = 50000* line (shown as left subtree in Figure 2.2) further splits that subspace into two parts as:

- *Price* ≤ 50000: Diamonds (*Mustang*)
- *Price* > 50000: Squares (*Ferrari*)

After the second test we have samples of only one class in each node. Hence we stop further splitting. The leaf *Mustang* below *Price* ≤ 50000 is a leaf node, where the decision is made that if the year of the car is less than 1980 and the price of the car is less than 50000, we can say that this car is a *Mustang*. This is a rule, which we have derived from the decision tree in Figure 2.2.

The rules of decision tree consist of *disjunctions of conjunctions*, which are paths from the root node to the leaf nodes. For this example, the rules are:

If $(Year \leq 1980 \wedge Price \leq 50000) \vee (Year > 1980 \wedge Price \leq 20000)$, then this car is a *Mustang*.
If $(Year \leq 1980 \wedge Price > 50000) \vee (Year > 1980 \wedge Price > 20000)$, then this car is a *Ferrari*.

After the tree is constructed, any instance can be classified given the values of *Year* and *Price* attributes. For example the instance

$< Year = 1970, Price = 60000 >$

is classified as *Ferrari* whereas

$< Year = 1990, Price = 10000 >$

would be classified as *Mustang*. Classification of one instance is done by tracing the corresponding path for that instance, from the root node until a leaf node. For example, for the first instance, we compare its year with 1980, while its year is smaller than 1980, we go to left subtree and compare its price with 50000, while its price is bigger than 50000 we say that, this car is a *Ferrari*.

Readability can also be improved by changing the rules of decision tree into if-then rules. For example the rules above could be changed into if-then rules as follows:

If the year of the car is smaller than 1980 and its price is smaller than 50000, then this car is a Mustang.

If the year of the car is bigger than 1980 and its price smaller than 20000, then this car is a Mustang.

If the year of the car is smaller than 1980 and its price bigger than 50000, then this car is a Ferrari.

If the year of the car is bigger than 1980 and its price bigger than 20000, then this car is a Ferrari.

Decision tree learning can also be generalized to the case where the target function is continuous, i.e., regression. This is beyond the scope of this thesis.

2.1. Algorithm for Tree Construction

The basic tree construction algorithm is a greedy search on the space of possible decision trees. The search starts by creating a root node and continues by processing the training set according to the following steps given below.

1. If all of the instances belong to the same class C_i stop and return the leaf node with class C_i.
2. Find the split that best classifies the instances (as will be explained later).
3. Each split divides the data into subsets. (For example the split *Year = 1980* divides the data into two subsets with seven and nine instances in each)
4. For each subset of the data, repeat steps 1, 2, 3 and 4 to construct the decision tree.

So this algorithm is recursive. For each node of the tree it is called once. Finding the best split takes most of the time. Time complexity of the algorithm is O (l * Timecomplexity (Best Split Function)), where l is the number of nodes in the decision tree.

2.2. Identification Trees (ID3)

Identification trees classify instances by sorting them down the tree from the root to some leaf node. Each internal node in the tree specifies a test of *one* attribute of the instance, and each branch descending from that node corresponds to one of the possible values or intervals for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the corresponding tree branch. This process is recursively repeated for the subtree of the new node until a leaf node is reached. The leaf node stores the class code.

In the ID3 algorithm (Quinlan, 1989) the best split is found as follows:

1. For each attribute x_i do the following:
    - If the feature is symbolic with m possible values, the instances are divided into m groups, where in each group the instances have the same value for the attribute x_i. Calculate the partition-merit criteria (will be explained in Section 2.3) as p_i.
    - If the feature is numeric, the instances could be divided into two in k different ways where k is the number of different values of the attribute x_i. For each of

these k ways, the partition-merit criterion is computed and the best is selected as p_i.

2. Find the attribute j such that $p_j = min_i p_i$ as the split node attribute.
   - If x_i is symbolic with m possible values, partition the set of instances into m subsets where at each partition $x_j = a_k$, $k = 1, \ldots, m$.
   - If x_i is numeric, partition the set of instances into two; $x_j \leq a$ and $x_j > a$, where a is the split threshold that optimizes the partition-merit criterion.

This algorithm has a complexity of O ($f * n$) where f is the number of attributes and n is the number of instances.

To classify the instances in the test set, we start from the root node and trace the tree node by node. At each internal node, we take the subtree for which the instance has the correct attribute value or on the correct side of the split threshold. At each leaf node, if the instance has the same class as the class of the leaf node the instance is correctly classified, else it is misclassified.

2.3. Partition-Merit Criteria

The central choice in tree algorithms is finding the best split. Each split partitions the sample into two or more parts and each subset of the partition has one or more classes in it. If there is only one class in a subset, then it is *pure*, else it is *impure*. The purer the partition is, the better it is. This measure of impurity is specified as the partition-merit criteria. An impurity measure has the characteristic of being minimum when there are only instances from one class and maximum when all classes have the same number of instances.

There are different types of partition-merit criteria. A comparative study (Mingers, 1989), found no significant differences in accuracy whereas there could be an interaction

between partition-merit criteria and data sets (Brodley and Utgoff, 1995). We have used three types of impurity measures. They are:

1. Weak Theory Learning Measure
2. Information Gain
3. The Gini Index

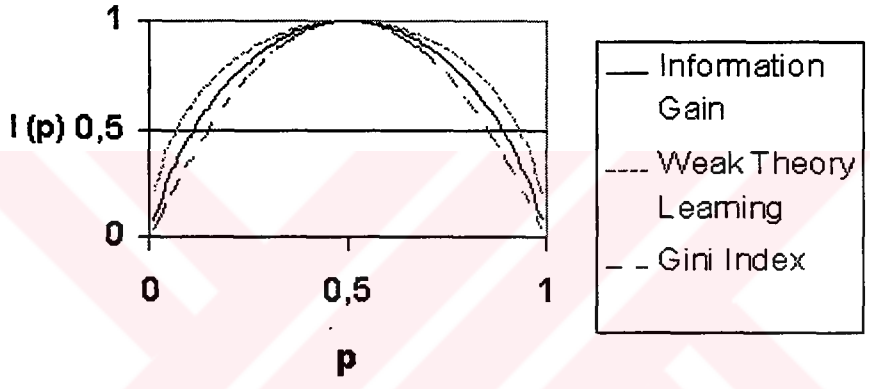

FIGURE 2.3.1 Graphs of Impurity Measures

Impurity measures are shown graphically in Figure 2.3.1. For simplicity, this graph is plotted for two classes. p denotes the probability of occurrence of the first class and $I(p)$ denotes the calculated impurity measure. We see that the Weak Theory Learning Measure has the highest concavity whereas the Gini Index has the lowest concavity. So the Weak Theory Learning Measure is the most discriminating function.

2.3.1. Weak Theory Learning Measure

Theoretical motivation for an impurity measure is defined as

$$G(p) = 2\sqrt{p(1-p)} \tag{2.1}$$

where p is the probability of positive instances (Dietterich et. al., 1996). This model gives a basis for our implementation. So given a node t with c different classes where each class C_i has the probability of occurrence p_i, the impurity of node t is

$$\hat{G}(N) = \sum_{i=1}^{c} \sqrt{p_i(1-p_i)} \tag{2.2}$$

Let S be a split at a node t with m-way split. Assume there are c classes and there are n_k instances having the attribute value a_k, where n_{kl} of them belong to class l. n is the total number of instances at that node satisfying $\sum_l n_{kl} = n_k$, $\sum_k n_k = n$. Then the impurity of split S is

$$\text{Impurity}(S) = \sum_{k=1}^{m} \frac{n_k}{n} \sum_{l=1}^{c} \sqrt{\frac{n_{kl}}{n_k}\left(1 - \frac{n_{kl}}{n_k}\right)} \tag{2.3}$$

2.3.2. Information Gain

This measure of information gained from a particular split is popularized in (Quinlan, 1986). Quinlan takes the famous entropy formula of Information Theory, which is the minimum number of bits to encode the classification of an arbitrary member of a collection S. So if we transform this idea into our problem, the entropy of node t is

$$\text{Entropy}(t) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.4}$$

whereas the entropy of a split S is

$$\text{Entropy}(S) = \sum_{k=1}^{m} \frac{n_k}{n} \sum_{l=1}^{c} -\frac{n_{kl}}{n_k} \log_2 \frac{n_{kl}}{n_k} \tag{2.5}$$

2.3.3. Gini Index

The Gini Criterion (or Index) was first proposed in (Breiman et. al., 1984). The Gini Index is originally defined as the probability of misclassification of a set of instances, rather than the impurity of the split. Gini index of a node t is

$$\text{Gini}(t) = 1 - \sum_{i=1}^{c} p_i^{\,2} \tag{2.6}$$

From that index we can refer the Gini index of a split as

$$Gini(S) = \sum_{k=1}^{m} \frac{n_k}{n} \left[1 - \sum_{l=1}^{c} \left(\frac{n_{kl}}{n_k} \right)^2 \right] \tag{2.7}$$

2.4. Multiple Splits

In the univariate decision tree algorithm ID3, there are m splits for symbolic attributes, one for each possible value of that attribute and there are two splits for numeric attributes as $x_i \leq a$ and $x_i > a$. Most of the partition-merit criteria give good results for more splits. So in the selection of the attribute, to make the split, the symbolic attributes have greater advantage over numeric features. To see and test if this is an advantage we

have also put a *multiple split* option to our decision tree induction algorithms (basically for ID3).

If there are s splits then the instances are divided into $s+1$ parts as $x_i<a_1$, $a_1 \leq x_i<a_2$, $\ldots a_{s-1} \leq x_i < a_s$ and $x_i \geq a_s$, where $a_1, a_2, \ldots, a_s$ are the split points. If the feature x_i has k values, then s-way split can be done in k^s different ways. During splitting, we split instances into possible different $s+1$ parts and for each partition we calculate the partition-merit criterion p_j and choose the best one.

This multiple splits only apply to numeric attributes; the data sets with only symbolic attributes will have no change in their results. Also this replacement prefers smaller split numbers in case of ties, so that the tree will have less nodes.

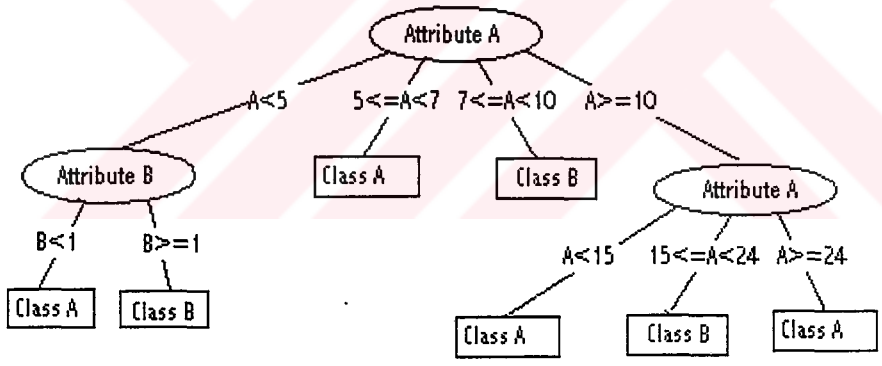

FIGURE 2.4.1 A decision tree with multiple splits

Figure 2.4.1 shows a decision tree with multiple splits. For the first decision node three split points are selected as $a<5$, $5 \leq a < 7$, $7 \leq a < 10$ and $a \geq 10$ whereas there are one split point for the second decision node and two split points for the third decision node. As can be seen, the attribute A is used twice while constructing the tree.

But we also change the iteration time of the algorithm. The algorithm has a time complexity of O (n^s) instead of O (n) because of the search in a space of k^s. So the total

complexity is O ($f * n^S$) instead of O ($f * n$), which can be very huge with large number of samples. So we have decided S to be at most three.

2.5. Filling in Missing Values

In some cases, some of the values of features may not be available. In such cases we must fill in the values of missing cases. There are a number of ways (Quinlan, 1989):

1. Ignoring any instance with a missing value of an attribute. This will reduce the number of available instances.
2. Filling in with the most likely value of the attribute with missing value of the instance.
3. Combining the results of classification using each possible value according to the probability of that value.

In our implementation, if the missing value is for an ordered feature, we fill it with the sample mean value of that attribute as it appears in the training set. When the missing feature is unordered, we fill it with the most probable value of that attribute in the data set. These statistics are stored so that the same filling can also be done for the test set. This method is called *mean imputation.*

2.6. Avoiding Overfitting

The algorithms growing the tree deeply enough to perfectly classify all of the training examples will not give always good results. This mainly occurs due to two different causes. Firstly, the data set may contain noise and if we learn all examples we will also learn noise, which will reduce our performance over the test set. Secondly, our training set data may not be a good representative (big enough) of the data set. In either of

these cases, univariate and multivariate algorithms can produce trees that *overfit* the training examples.

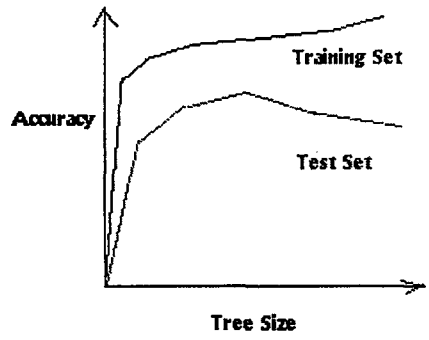

FIGURE 2.6.1. Overfitting in Learning

As the ID3 algorithm adds new nodes to the tree, the training set accuracy increases. However the test set accuracy first increases then decreases as can be seen in Figure 2.6.1.

One possibility is to prune unnecessary nodes or subtrees after the construction of the tree to avoid overfitting. In our implementation, we have used two types of pruning methods namely pre-pruning and post-pruning.

2.6.1. Pre-pruning

Pre-pruning methods simplify decision trees by preventing the tree to be complete. A simple form of pre-pruning that stops tree expansion in depth two performs surprisingly well (Holte, 1993). Usually, however the tree is no more expanded when no or not enough gain is expected.

Pre-pruning methods are more efficient than post-pruning methods because they terminate tree generation earlier whereas post-pruning methods require a post-processing step, where the tree is pruned back to give a smaller tree.

In our implementation of pre-pruning, we stop splitting further when the instance ratio (the number of instances of that node divided by the number of instances of the whole data set) is below a threshold (e.g., five percent). Then we create a leaf node and label it with class C_i where C_i is the class having the most instances.

Pre-pruning methods give inconsistent performance because of the *horizon effect* (Breiman et. al., 1984). This occurs mainly by stopping tree expansion prematurely. After this inconsistent behavior is noticed, the research in this area is abandoned in favor of post-processing methods. But in the case of large-scale data sets, where efficiency may be more important than accuracy, pre-pruning methods will be considered again.

2.6.2. Post-pruning

Post-pruning is the most frequently used tree simplification algorithm, producing from an unpruned tree T, a pruned tree T'. Pruning the tree replaces a subtree with a leaf node, if doing so, the accuracy on a pruning set, distinct from the training set, improves.

If a decision tree is expanded using only the homogeneity stopping criteria, it will contain no resubstitution errors on the training set. Thus post-pruning can only increase resubstitution errors. However when the tree is expanded it may overfit the sample space by learning noise. So post-pruning can decrease error rate on unseen test cases.

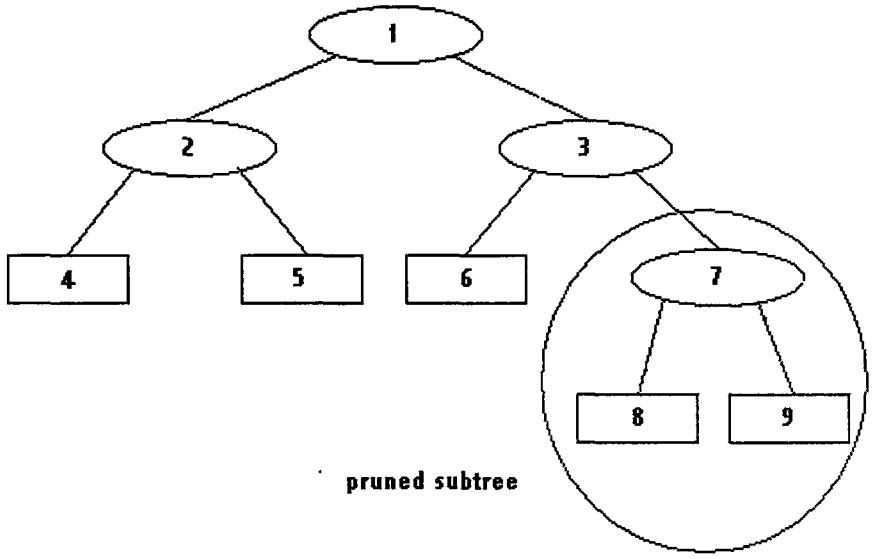

FIGURE 2.6.2.1 A pruned subtree

In the post-pruning algorithm, at each internal node, we check the classification accuracy change on the pruning set by pruning the subtree having that node as its root. If the classification accuracy does not decrease, we decide to prune that subtree into a leaf node.

Figure 2.6.2.1 shows a decision tree where nodes numbered 8 and 9 are pruned. After pruning, the subtree below the decision node 7 is converted into a leaf node.

In post-pruning, we use a set other than the training and test sets, called the pruning set. So in our implementation, we divide the whole data set equally into two parts; one training set and one test set and then we take 80 percent of the train set to form the growing set and 20 percent to obtain the pruning set. One observed disadvantage of this division is that it reduces the number of training cases involved in tree induction, which is not desirable for small data sets.

3. MULTIVARIATE DECISION TREES

3.1. Univariate vs. Multivariate Splits

Multivariate decision trees mainly differ from univariate decision trees in the way they test the attributes. Univariate decision trees test only one attribute at a node whereas multivariate decision trees test more than one attribute (generally a linear combination of attributes) at a node. This limitation to one attribute reduces the ability of expressing concepts. It shows its disability in three forms. Splits could only be orthogonal to axes, subtrees may be replicated in the decision tree and there may be fragmentation.

For the first disability, consider the two-dimensional instance space shown in Figure 3.1.1. To approximate the class boundary, the corresponding univariate decision tree uses a series of orthogonal splits, whereas the multivariate test uses only one linear split.

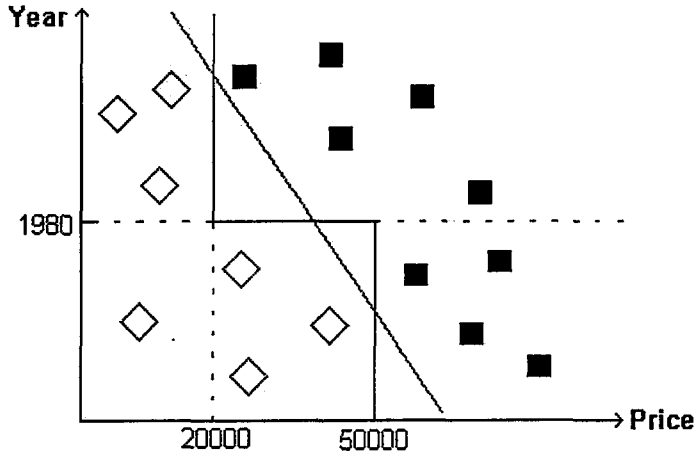

FIGURE 3.1.1 Comparison of univariate and multivariate splits on the plane

This example shows the well-known problem that a univariate test using feature x_i can only split a space with a boundary that is orthogonal to the x_i axis. This results in larger trees and poor generalization.

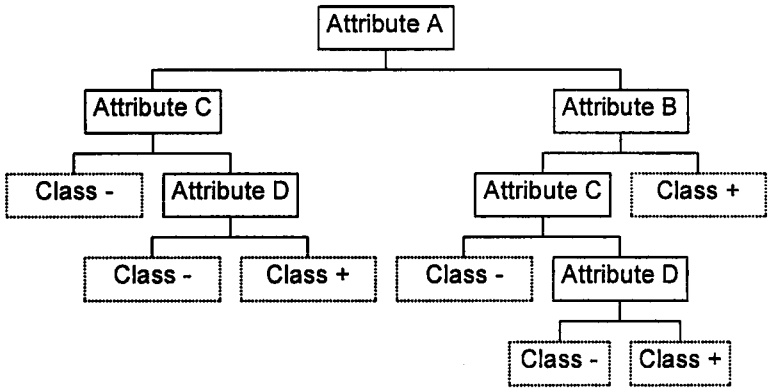

FIGURE 3.1.2. An example decision tree for replication problem

The second problem is the replication of trees. The decision tree shown in Figure 3.1.2 gives an inefficient representation of the proposition $(A \wedge B) \vee (C \wedge D)$. While the term $(A \wedge B)$ is described efficiently, the term $(C \wedge D)$ requires two identical subtrees to be represented. In general, conjunctions can be described efficiently by decision trees while disjunctions require a large tree to describe (Pagallo and Haussler, 1990) (Mathues and Rendell, 1989). One solution to the replication problem is to allow decision nodes consisting of more than one feature by combining them with an appropriate function.

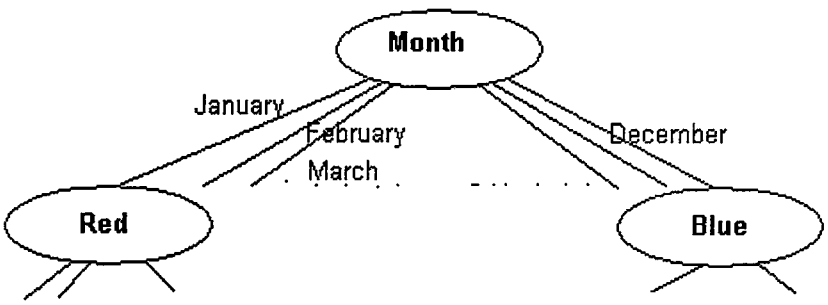

FIGURE 3.1.3. An example decision tree for fragmentation problem

Figure 3.1.3 shows another kind of replication problem that can occur when the data contains attributes with high arity values i.e., attributes with large number of possible values. If a tree has high arity attributes (say arity ≥ 5) then it will quickly fragment the data in that node into small partitions. To avoid this problem we can construct subsets of the attribute values (Fayyad and Irani, 1992), e.g., seasons instead of months.

But in a multivariate decision tree, each test can be based on more than one feature. In Figure 3.1.1, we can separate the examples of two classes with one line. The test node is the multivariate test $w_1 x_1 + w_2 x_2 \leq c$. Instances for which $w_1 x_1 + w_2 x_2$ is less than or equal to c are classified as one class; otherwise they are classified as the other class.

The multivariate decision tree-constructing algorithm selects not the best attribute but the best linear combination of the attributes: $\sum_{i=1}^{f} w_i x_i > w_0$. w_i are the weights associated with each feature x_i and w_0 is the threshold to be determined from the data. So there are basically two main operations in multivariate algorithms: Feature Selection determining which features to use and finding the weights w_i of those features and the threshold w_0.

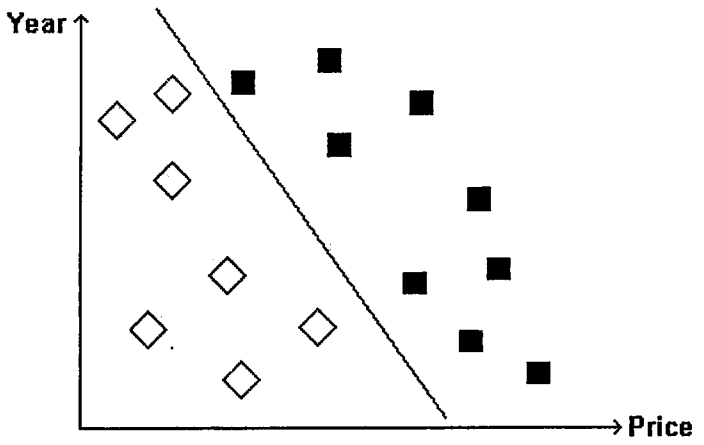

FIGURE 3.1.4 Instances of the problem *Choosing Car* with multivariate split

Figure 3.1.4 illustrates a multivariate split for the same problem in Chapter 2. As it can be seen, the classes (*Ferrari*'s and *Mustang*'s) can now be splitted using only one multivariate split. The corresponding multivariate decision tree is shown in Figure 3.1.5. The split for the first node is

$w_1\ Year + w_2\ Price \leq w_0$,

which is a linear combination of features. The size of the tree is now reduced from seven nodes into three nodes.

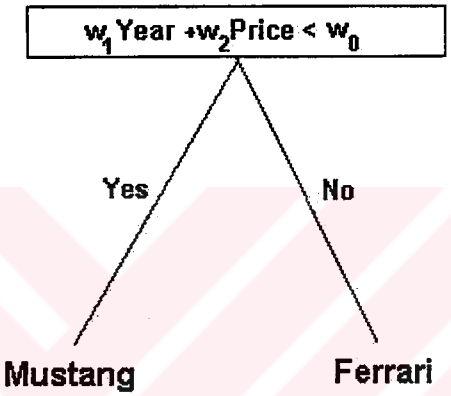

FIGURE 3.1.5 Multivariate decision tree for the problem *Choosing Car*

Multivariate decision trees differ from univariate trees in two other respects: First, because the features in the linear combination split are multiplied with the coefficients, we must convert the symbolic features into numeric features. And second, because the final weighted sum is numeric, all splits are binary.

3.2. Symbolic and Numeric Features

A decision tree algorithm must handle both ordered and unordered features. In a univariate decision tree algorithm, we can use ordered features as $x_i \leq a$, where a is in the

observed range of the feature i and unordered features as $x_i = a_j$ where a_j are possible values of the unordered feature x_i in the train set.

In a multivariate decision tree algorithm, we must convert each unordered feature i into a numeric representation to be able to take a linear combination of the features. So in our implementation, where i is the unordered feature with possible values $a_1, \ldots, a_m$ we convert it into multiple feature set as $i_1, \ldots, i_m$ where $i_j = 1$ if x_i takes value a_j and 0 otherwise. At the end of this preprocessing we convert previous features (some ordered, some unordered) $x_1, \ldots, x_f$ into all ordered features as $x_1, x_{21}, x_{22}, x_{23} \ldots x_{2m}, x_{31}, x_{32}, x_{33} \ldots x_{3p}, x_4, \ldots, x_f$ where $x_1, x_4 \ldots$ are ordered, $x_{2j}, x_{3j} \ldots$ are unordered features.

This encoding avoids imposing any order on the unordered values of the feature (Hampson and Volper, 1986), (Utgoff and Brodley, 1991). Note that the dimensionality and complexity increases when the number of attributes is increased.

3.3. Feature Selection

In multivariate decision tree construction, we take a linear combination of features. Our main purpose is to find the coefficients w_i for these features. But using all of the features may be costly and may lead to overfitting on a small training set. So in order to get rid of these unnecessary features, we use feature selection, thereby effectively setting w_i of unnecessary features to 0.

Feature selection algorithm proceeds as follows: Firstly the coefficients of all attributes are determined. Afterwards, each attribute x_i is dropped by setting $w_i = 0$ and the increase in the impurity for that attribute is found. If dropping an attribute increases impurity significantly, it is an important attribute. Applying this rule we can find the most and least important attributes. Then we check if the impurity increase for the least important attribute is less than the impurity increase for the most important attribute by a

constant ϕ (normally 0.1 or 0.2). If the difference is smaller we conclude that it is an unnecessary attribute and we drop it. We repeat this process until this condition is not satisfied for any attribute.

3.4. Classification and Regression Trees (CART)

CART algorithm (Breiman et. al., 1984) for finding the coefficients of the available features is a step-wise procedure, where in each step, one cycles through the features x_1, x_2, ..., x_f doing a search for an improved linear combination split. If there are symbolic features, they are converted to numeric features. Each instance is normalized by centering each value of each feature at its median and then dividing by its interquartile range. CART algorithm is very similar to ID3.

Finding the best split with CART algorithm is done as follows:

1. While disorder decreases
2. For each of the feature x_i
3. For $\gamma \in \{-0.25,0,0.25\}$
   - Let the current split be $v \le c$, where $v = \sum_{m=1}^{f} \beta_m x_m$, the aim is to find the best split of the form $v - \delta (x_i + \gamma) \le c$.
   - Divide the instances into two groups as

   $\delta \ge \frac{v-c}{x_i+\gamma}, x_i+\gamma \ge 0$ and $\delta \le \frac{v-c}{x_i+\gamma}, x_i+\gamma < 0$.
4. For each instance:
   - Calculate $u_n = \frac{v_n - c}{x_{i,n}+\gamma}$. Divide the instances into two groups as for all u_n such that $x_{i,n} + \gamma \ge 0$, the algorithm finds the best split of the form $u \le \delta_1$. For all u_n such that $x_{i,n} + \gamma < 0$, the algorithm also finds the best split of the form $u \ge \delta_2$. Take the better of these two splits and let δ be the corresponding threshold.
5. These three best splits are compared and δ, γ values corresponding to the best of the three are used to update v as follows:

$v' = \sum_{m=1}^{f} \beta'_m x_m$, $\beta'_i = \beta_i - \delta$, $\beta'_m = \beta_m$, $m > 1$ and $c' = c + \delta\gamma$.

6. At this point the updated split is of the form $v_l \leq c_l$. The cycle is completed by finding the best split of the form $v_l \leq c_l'$ as c_l' ranges over all values.

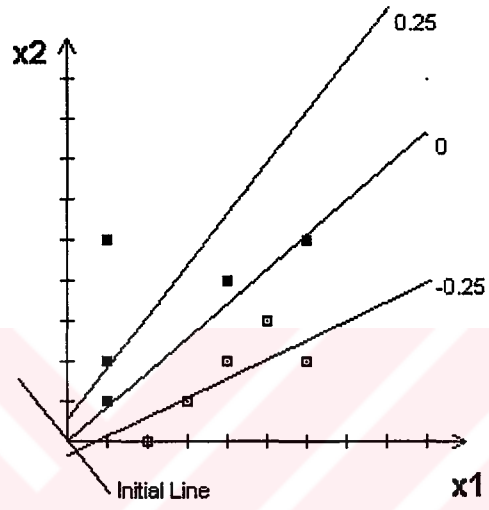

FIGURE 3.4.1 A step in CART algorithm

Figure 3.4.1 shows the first step of the CART algorithm for an example data set. The initial line is given as $x_1+x_2<0$. The lines shown as –0.25,0 and 0.25 are the best splits found for γ = -0.25,0 and 0.25. Here only the coefficient of attribute x_1 is changed. The line with $\gamma = 0$ will be selected for further iteration.

Once the best split is found, we create two children as $v = \sum_{m=1}^{f} \beta_m x_m < c$ and $v > c$, and divide the set of instances into two. We then continue recursively until either a subset is pure enough or the subset is small enough.

This algorithm has a complexity of O (6 * k * f * n) where k is the number iterations done while disorder decreases, f is the number of features, N is the number of instances.

Loop (1) is iterated k times, loop (2) is iterated three times, loop (3) is iterated f times and loop (4) is iterated $2*n$ times. Also the algorithm is called for each node one time so the total complexity in training phase is O ($6 * k * f * n$).

The testing phase of the algorithm is as follows: Each instance is first normalized according to the median and quartile computed from the training set. Then the tree is traversed until a leaf node is encountered, by taking the linear combination of input attributes with the coefficients found by the CART algorithm for that node. If the instance has the class code same as the code of the leaf node, it is correctly classified else it is misclassified.

Although CART is a good multivariate algorithm it has some basic limitations. First of all, it is fully deterministic as ID3. There is no built in mechanism for escaping local minima, although such minima may be very common for some domains. Secondly, it sometimes makes adjustments that increase the impurity of the split. This feature was probably included to overcome local minima problem but it also has a drawback.

There is no upper bound on the time spent at any node in the decision tree. Loop (1) halts when no perturbation changes the impurity more than ε, but because impurity may increase and decrease, the algorithm can spend arbitrarily long time, or in some times infinite time, at a node. To overcome this problem, we have included also an iteration constant. If the algorithm cycles more than this iteration constant, it stops and returns the current split.

4. NEURAL NETWORK MODELS FOR TREE CONSTRUCTION

As we have seen in Chapter 3, the aim in multivariate decision tree construction is to find a way to determine the coefficients of the attributes. In this chapter, we will discuss how to determine these set of coefficients by using methods based on neural networks.

When neural networks are used in decision trees, at each node of the decision tree, there is a neural network trained with its corresponding data. Once the weights of the neural networks are found, they can be used to classify a test example.

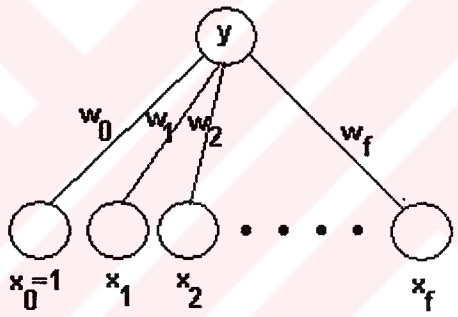

FIGURE 4.1 Linear perceptron model for multivariate decision trees

A model for a linear perceptron is shown in Figure 4.1 (Bishop, 1996). $x_0, x_1, x_2 \ldots x_f$, form the input layer of the neural network whereas the node on top is the output node of the neural network denoted by y. The output of the neural network is binary. w_i's are the weights of the input neurons. The weight of $x_0 = 1.0$ is w_0, which will be used as a threshold unit (Corresponding to c of CART). y is computed as follows:

$$y = sigmoid(\sum_{i=0}^{f} w_i x_i) = \frac{1}{1+e^{-\sum_{i=0}^{f} w_i x_i}} \qquad (4.1)$$

The sigmoid function gives a value between 0 and 1.

In order to train the linear perceptron, gradient-descent is used. This algorithm is used to train a single-output linear perceptron to differentiate between two disjoint groups of classes, which are the classes in the left branch C_L and the classes in the right branch C_R. The desired output for an instance is 1 when its class belongs to the group C_L and 0 when its class belongs to the group C_R. If there are only two classes present in that node, one class belongs to C_L and other to C_R. Otherwise we must select an appropriate partition for the classes available.

Finding the best split with neural network algorithms is done as a nested optimization problem. In the inner optimization problem, the gradient-descent algorithm is used to minimize the mean-square error and so find a good split as defined by w_i for the given two distinct groups of classes C_L and C_R. In the outer optimization problem, we find the best split of classes into the two groups, C_L and C_R.

4.1. Training Neural Networks

The inner optimization problem will be solved by using neural networks at each node of the decision tree. For this purpose we have used three different kinds of neural network models. These are linear perceptrons, multilayer perceptrons and a hybrid combination.

4.1.1. Linear Perceptron Model

A linear perceptron neural network model is shown in Figure 4.1. The training of this network is done by gradient-descent algorithm. Let (x^t, d^t) denote the training data, where x^t is the instance t and d^t is the desired output for that instance, which is 1 for left child and 0 for right child. y^t is the real output found by the formula:

$$y^t = sigmoid(\sum_{i=1}^{f} w_i x_i^t + w_0) \tag{4.2}$$

A stochastic gradient algorithm for minimizing the mean-square error criterion is used.

$$E = \frac{1}{n}\sum_{t=1}^{n}(d^t - y^t)^2 \tag{4.3}$$

At each epoch of training, all instances are passed one at a time in random order. While passing the coefficients of the input layer, w_i and threshold unit w_0 are updated by the given formulas. In these formulae η stands for learning rate. Learning rate is started as $0.3 / f$ and decreased to 10^{-5} by multiplying at each epoch with a constant. α stands for momentum rate (0.7 in our application). It is used to update w_i by multiplying with the previous update value as:

$$\Delta w_i^t = \eta(d^t - y^t)\, y^t\, (1 - y^t)\, x_i^t + \alpha\, \Delta w_i^{t-1}\ ,\ i=0,\ldots,n,\ x_0=+1 \tag{4.4}$$

$$w_i^{t+1} = w_i^t + \Delta w_i^t \tag{4.5}$$

The training of linear perceptron takes $O(e * n * (f+1))$ where e is the number of epochs to train the network, n is the number of instances and $(f+1)$ is the number of inputs (+threshold), which is the number of weights to update.

4.1.2. Multilayer Perceptron Model

A multilayer perceptron model for decision making at a node is shown in Figure 4.1.2.1.

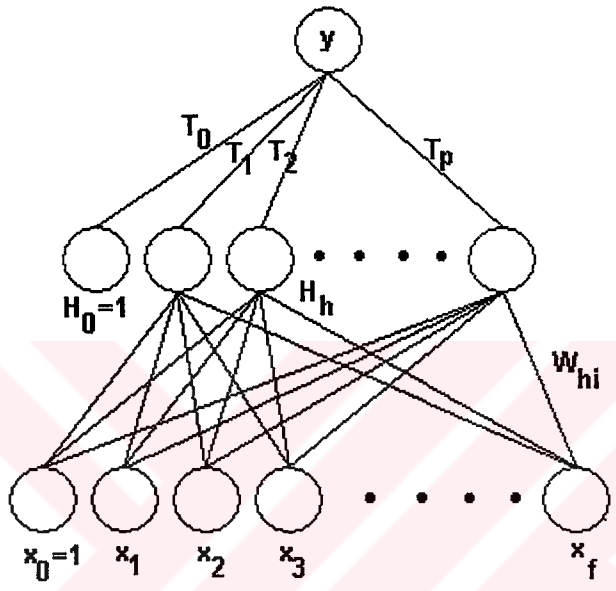

FIGURE 4.1.2.1 Multilayer perceptron model with one hidden layer

There is a hidden layer between input and output layers which makes y a nonlinear function of x. $x_0, x_1, x_2 \ldots x_f$ form the input layer, $H_0, H_1, \ldots H_m$ form the hidden layer and y denotes the output of the neural network. For the weights connecting the layers, w_{hi} connects input neuron i to hidden neuron h and T_h connects hidden neuron h to the output layer. x_0 and h_0 denote the bias units for the input and hidden layers respectively. The number of units in the hidden layer is taken as half of the number of features, for suitable dimensionality reduction from f to 1.

At instance t the real output y^t and the hidden layer output H_h^t are found as:

$$H_h^t = sigmoid(\sum_{i=0}^{f} w_{hi} x_i^t + w_{h0}) \tag{4.6}$$

$$y^t = sigmoid(\sum_{h=0}^{m} T_h H_h^t + H_0) \tag{4.7}$$

The update rules are different from single layer perceptron. There are two layers so there are two update rules, one for updating the w_{hi} and the other for updating T_h. Learning rate is started as 0.3 / f and decreased as explained in linear perceptron model. The update rules are:

$$\Delta T_h^t = \eta (d^t - y^t) y^t (1 - y^t) H_h^t + \alpha \Delta T_h^{t-1}, \; h=0,...,m, \; H_0=+1 \tag{4.8}$$

$$T_h^{t+1} = T_h^t + \Delta T_h^t \tag{4.9}$$

$$\Delta w_{hi}^t = \eta (d^t - y^t) y^t (1 - y^t) T_h^t H_h^t (1 - H_h^t) x_i^t + \alpha \Delta T_h^{t-1} \tag{4.10}$$

where $h=0,...,m$ and $i=0,...,n$ $H_0=+1$ $x_0=+1$

$$w_{hi}^{t+1} = w_{hi}^t + \Delta w_{hi}^t \tag{4.11}$$

The training of the multilayer perceptron takes $O(e * N * f^2)$ where e is the number of epochs to train the network, n is the number of instances and f is the number of features. It is f^2 because there are $m * f$ updates for the weights in the first layer and m updates in the second layer. m equals to $f / 2$ as explained before. So the total number of updates is $f^2 / 2 + f / 2$ which is $O(f^2)$.

Multilayer perceptron models differ from other multivariate models in its nonlinear nature. With this model one can have nonlinear split at a decision node. In Figure 4.1.2.2, an example nonlinear split is shown to solve the problem *Choosing Car*.

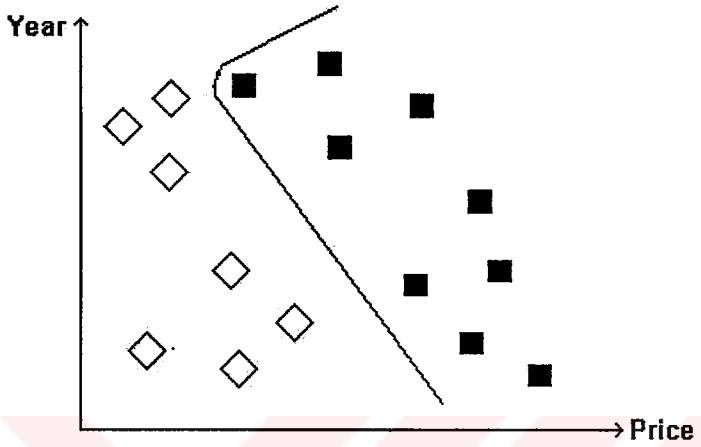

FIGURE 4.1.2.2 A nonlinear split to *Choosing Car* problem

4.1.3. The Hybrid Model

If we compare the complexity of the decision at a node, we see that the univariate methods are the simplest methods as they test only one feature. Linear methods, which take a linear combination of features, are more complex and nonlinear methods such as multilayer perceptrons are the most complex. But the aim of the decision trees is to find a way of representation of the decision that is as simple as possible and as accurate as possible. So we should not use always nonlinear methods, which have the additional disadvantage that they are not easily interpretable.

Therefore, it seems better to find a way of combining linear and nonlinear methods in a hybrid model. In this model, at each node we train both a linear and nonlinear perceptron and use a statistical test to check if there is a significant performance difference between the two. If the performance of the multilayer perceptron is better then the performance of the linear perceptron with a confidence level of %95, the multilayer perceptron is chosen

to have a nonlinear decision node, else linear perceptron is chosen to have a linear decision node.

The tests we have used to compare linear and nonlinear models are combined 5x2 cv F test and 30 fold cross-validation paired t test. The details of the two tests are given in Appendix B.

4.2. Class Separation by Selection Method

Class separation is a process that must be done when there are three or more classes available at the decision node. If there are c classes, then there are 2^{c-1}-1 distinct partitions possible. Because we can not test for all, we use heuristics to get a reasonable partition in a reasonable amount of time. The first method we have used in class separation is the selection method, which is a depth-first search method with no backtracking. Let t be a decision node and $C=\{C_1, \ldots, C_m\}$ be the set of c classes at node t.

1. Select an initial partition, C_L and C_R where each part only consists of examples of two arbitrarily chosen classes C_i and C_j respectively.
2. Train the network at node t with the given partition. Do not consider the elements of other classes yet.
3. For other classes in the class list, search for the class C_k that is best placed into one of the partitions. Best placing is the placing when C_k is assigned into the child where the impurity is minimum.
4. Put C_k into its best partition and continue adding classes one by one using steps 2 to 4 until no more classes are left.

This algorithm is sensitive to the initial class partition due to its depth-first nature. So a heuristic technique to select the initial partition is used instead of selecting randomly.

The Euclidean distance of the means of two classes (for all classes) is calculated, and the two furthest classes are taken as the initial two classes.

Algorithm traces steps 2 to 4 c-2 times. So the complexity of this algorithm is O(c) where c is the number of the class available at node t.

4.3. Class Separation by Exchange Method

The second method in class separation is the exchange method (Guo and Gelfond, 1992). This is a local search with backtracking. Let t be a decision node and $C=\{C_1, \ldots, C_c\}$ be the set of c classes at node t.

1. Select an initial partition of all in classes into two subsets, C_L and C_R.
2. Train the network to separate C_L and C_R. Compute the entropy E_0 with the selected entropy formulae.
3. For each of the classes $k \in \{C_1, \ldots, C_c\}$ form the partitions $C_L(k)$ and $C_R(k)$ by changing the assignment of the class C_k in the partitions C_L and C_R. Train the neural network with the partitions $C_L(k)$ and $C_R(k)$. Compute the entropy E_k and the decrease in the entropy $\Delta E_k=E_k-E_0$.
4. Let ΔE_* be the maximum of the impurity decreases for all classes. If this impurity decrease is less than 0 then exit else set $C_L = C_L(k)$ and $C_R = C_R(k)$, and do the steps 2 to 4 again.

We use a heuristic technique to start, instead of starting randomly.

1. The two classes C_i and C_j with the maximum distance are found and placed into C_L and C_R.
2. For each of the classes $k \in \{C_1, \ldots, C_c\}$ find the one with the minimum distance to C_L or C_R and then put it into that group. Repeat this second step until no more classes are left.

5. THE STATISTICAL MODEL FOR TREE CONSTRUCTION

In this section we will discuss how to use a statistical approach to determine the set of coefficients in a linear decision node. So when we want to find a new split in a node of the tree, we use a statistical criterion to determine the coefficients of the features.

The statistical approach is named as Fisher's *Linear Discriminant Analysis* (LDA) (Duda and Hart, 1973). This approach aims to find a linear combination of the variables that separates the two classes as much as possible, which is what we want to do. The criterion proposed by Fisher is the ratio of between-class to within-class variances, which we want to maximize. Formally we must find a direction **w** to maximize

$$J_F = \frac{\left|\mathbf{w}^T(\mathbf{m}_L - \mathbf{m}_R)\right|^2}{\left|\mathbf{w}^T \mathbf{S}_w \mathbf{w}\right|} \tag{5.1}$$

where $\mathbf{m}_L$ and $\mathbf{m}_R$ are the two left and right groups means

$$\mathbf{m}_L = \frac{1}{\mathrm{n}_L} \sum_{\mathbf{x} \in C_L} \mathbf{x}, \; \mathbf{m}_R = \frac{1}{\mathrm{n}_R} \sum_{\mathbf{x} \in C_R} \mathbf{x} \tag{5.2}$$

and $\mathbf{S}_w$ is within-class covariance matrix, which is bias corrected as

$$\frac{1}{n_L + n_R}(n_L \Sigma_L + n_R \Sigma_R) \tag{5.3}$$

where Σ_L and Σ_R are the covariance matrices of class groups C_L and C_R respectively.

$$\mathbf{\Sigma}_L = \sum_{\mathbf{x} \in C_L} (\mathbf{x} - \mathbf{m}_L)(\mathbf{x} - \mathbf{m}_L)^T \, , \mathbf{\Sigma}_L = \sum_{\mathbf{x} \in C_R} (\mathbf{x} - \mathbf{m}_R)(\mathbf{x} - \mathbf{m}_R)^T \tag{5.4}$$

There are n_L samples in the left class group and n_R samples in the right class group. The solution for **w** that maximizes J_F can be obtained by differentiating J_F with respect to **w** and equating it to zero. So we define **w**, namely the set of coefficients of the linear combination as:

$$\mathbf{w} = \mathbf{S}_w^{-1}(\mathbf{m}_L - \mathbf{m}_R) \tag{5.5}$$

But the coefficients of the features is only for the direction, we must also specify a threshold w_0, which is defined as:

$$\mathbf{w}_0 = -\frac{1}{2}(\mathbf{m}_L + \mathbf{m}_R)^T \mathbf{S}_w^{-1}(\mathbf{m}_L - \mathbf{m}_R) - \log(\frac{n_R}{n_L}) \tag{5.6}$$

Note however that though the direction given in Equation 5.3 has been derived without any assumptions of normality, normal assumptions are used in Equation 5.4 to set the threshold for discrimination.

So in our case, we first divide classes into two groups as C_L and C_R by an appropriate class selection procedure (as defined in Sections 4.2 or 4.3). But this time the inner optimization procedure will be carried out by LDA as we will find the linear split with the Equations 5.3 and 5.4. There will be no training but only simple calculations with matrices. So we expect less training time in LDA, than a neural network.

In implementing the idea, we see that if there is a linear dependency between two or more features then some of the rows or columns of the covariance matrix S_w are the same or a multiple of the other and it becomes singular. But in this case the determinant will be zero and we can not find S_w^{-1}.

A possible answer to the problem is PCA, which is *principal component analysis* (Rencher, 1995). In this analysis we first find the eigenvectors and eigenvalues of the matrix S_w. As the multiplication of the eigenvalues of a matrix is equal to the determinant, we sort the eigenvalues of the matrix and get rid of the eigenvectors with small eigenvalues. Let us say that the eigenvalues of the matrix S_w are $\lambda_1, \lambda_2 \ldots \lambda_f$, which are sorted in decreasing order. We will find the new number of features k such that;

$$\frac{\lambda_1 + \ldots + \lambda_k}{\lambda_1 + \ldots + \lambda_k + \ldots + \lambda f} > \varepsilon \tag{5.7}$$

where ε is the proportion of variance explained. k is the number of eigenvectors that are linearly independent. After finding k, we will find the corresponding k eigenvectors and store them. For each instance with f features, we will multiply it with the k eigenvectors to have a new instance with the number of features k. So our instance space is mapped from f dimensions into k dimensions. After this we will do LDA now with the mapped instances and find a new S_w and means and calculate the split. Now S_w can have an inverse so there is no problem.

In order to test the tree we will convert first the test instances into a space of k dimensions and then we will multiply the new test instance with the linear split found with LDA. If the result is greater then $-w_0$ this instance is assigned to the left subtree of the node else to the right subtree of the node.

6. RESULTS

For testing the algorithms discussed in this thesis, 20 data sets from the UCI Repository (Merz and Murphy, 1998) are used. The properties of these data sets are shown in Table 3.1 (See Appendix A for more details). The number of instances of these sets varies from 100 to 8000, the number of attributes varies from five to 65 and the number of classes varies from two to ten. There are also three different types of attributes: Continuous, discrete and mixed. Seven of these data sets have also missing values.

TABLE 6.1 Data sets properties

Data set name	Instances	Attributes	Classes	Missing	Type of Attributes
Breast	699	9	2	Y	Continuous
Bupa	345	6	2	N	Continuous
Car	1728	21	4	N	Discrete
Cylinder	541	69	2	Y	Mixed
Dermatology	366	34	6	Y	Continuous
Ecoli	336	7	8	N	Continuous
Flare	323	23	3	N	Mixed
Glass	214	9	7	N	Continuous
Hepatitis	155	19	2	Y	Continuous
Horse	368	97	2	Y	Mixed
Iris	150	4	3	N	Continuous
Ironosphere	351	34	2	N	Continuous
Monks	432	6	2	N	Continuous
Mushroom	8124	66	2	Y	Discrete
Ocrdigits	3823	64	10	N	Continuous
Pendigits	7494	16	10	N	Continuous
Segment	2310	18	7	N	Continuous
Vote	435	32	2	Y	Discrete
Wine	178	13	3	N	Continuous
Zoo	101	16	7	N	Continuous

For each method, we performed ten runs on each data set. The results of ten runs are then averaged and we report the mean and standard deviation of each method classification rate for each data set. For comparing performance of the methods we have used the combined 5x2 cv F Test (Alpaydın, 1999).

In our results, > denotes a confidence level between %90 and %95, >> denotes a confidence level between %95 and %99, >>> denotes a confidence level of over %99.

6.1. Results for Identification Trees

For the rest of these results, the definitions in Table 6.1.1 apply:

TABLE 6.1.1 Definition of methods

Name of the Method	Uni/Multi	Impurity Measure	Pruning	Multiple Splits
ID3	Uni	Information Gain	Pre-pruning	No
ID3Gini	Uni	Gini Index	Pre-pruning	No
ID3Root	Uni	Weak Theory L.	Pre-pruning	No
ID3P	Uni	Information Gain	Post-pruning	No
ID3-2	Uni	Information Gain	Pre-pruning	Yes Degree 2
ID3-3	Uni	Information Gain	Post-pruning	Yes Degree 3

6.1.1. Comparison of Different Kinds of Learning Measures

In this part, the three impurity learning measures are compared: Information Gain, Gini Index and Weak Theory Learning Measure. For pruning purposes, pre-pruning is applied. This section compares these three measures in terms of accuracy, node size and learning computation time. Accuracy results for impurity measures are shown in Table

6.1.1.1 and Figure 6.1.1.1. Node results are shown in Table 6.1.1.2 and Figure 6.1.1.2. Learning time results are shown in Table 6.1.1.3 and Figure 6.1.1.3.

For three impurity measures there is no significant difference in accuracy (except in one data set).

For larger data sets, which have more than 1000 samples, in three of five cases, ID3Root is better then ID3 and ID3 is better then ID3Gini in node size significantly. In other cases no significant increase or decrease is found.

For mixed data sets, where continuous and discrete attributes are together, it is seen that the discrete attribute with larger arity is firstly selected as a split attribute. This is due to the fragmentation problem.

As the node size increases, learning time increases accordingly and this result becomes significant while the data set size growing.

In terms of learning time, in seven of 20 data sets, ID3 is better then ID3Gini significantly.

TABLE 6.1.1.1 Accuracy results for three different types of impurity measures

Data set name	ID3	ID3Gini	ID3Root	Significance
Breast	94.11±1.24	94.13±1.57	94.34±1.53	
Bupa	62.26±5.33	60.46±3.85	61.39±4.38	
Car	80.97±1.26	80.49±1.32	80.90±1.24	
Cylinder	68.50±2.22	67.39±2.91	70.06±3.66	
Dermatology	92.84±2.37	93.33±2.36	92.51±2.32	
Ecoli	78.10±3.57	78.21±2.50	77.92±4.18	
Flare	85.26±2.03	84.89±2.00	85.07±2.14	
Glass	60.65±5.97	59.35±6.14	63.36±6.27	
Hepatitis	78.44±3.71	74.33±10.36	75.47±5.53	
Horse	87.55±1.98	87.50±1.93	87.83±1.94	
Iris	93.87±2.75	93.87±2.75	93.47±2.47	
Ironosphere	87.63±3.15	84.96±2.83	87.00±2.37	
Monks	92.27±10.15	92.22±10.20	92.22±10.20	
Mushroom	99.70±0.06	99.68±0.08	99.62±0.15	
Ocrdigits	78.40±1.47	77.33±1.74	76.74±1.32	
Pendigits	85.73±1.01	86.59±0.85	85.37±1.16	
Segment	**91.08±1.16**	90.49±2.02	**89.08±1.06**	1>>3
Vote	94.94±1.06	95.63±1.83	94.94±0.94	
Wine	88.65±3.72	89.55±3.97	90.11±3.78	
Zoo	92.06±4.80	92.26±4.75	92.45±4.79	

TABLE 6.1.1.2 Node results for three different types of impurity measures

Data set name	ID3	ID3Gini	ID3Root	Significance
Breast	17.00±2.11	18.80±2.90	18.60±4.30	
Bupa	53.40±5.48	54.20±6.20	58.00±6.75	
Car	25.40±0.70	25.10±0.74	25.60±0.52	
Cylinder	**54.10±5.90**	**59.40±7.75**	56.60±6.70	2>>1
Dermatology	20.40±2.67	19.80±2.15	19.20±2.39	
Ecoli	33.80±2.70	35.00±2.67	34.60±6.52	
Flare	37.90±4.51	39.30±4.30	37.50±3.87	
Glass	38.20±5.90	38.80±4.16	40.20±5.27	
Hepatitis	19.60±3.78	20.60±2.95	21.20±2.90	
Horse	55.80±5.92	56.60±6.64	55.80±5.92	
Iris	8.40±1.35	8.40±1.35	8.40±1.35	
Ironosphere	19.20±3.05	21.60±4.53	19.60±3.41	
Monks	25.40±13.53	25.20±13.21	25.20±13.21	
Mushroom	23.00±0.00	24.40±1.71	22.40±1.26	
Ocrdigits	**74.40±4.01**	**97.80±7.50**	**61.40±3.63**	2>>1>>>3
Pendigits	**81.80±5.51**	**99.20±7.27**	**67.80±3.79**	2>>>1>>>3
Segment	41.80±3.79	**47.80±5.43**	**34.40±3.66**	2>>3
Vote	18.20±3.16	18.00±3.02	19.40±3.86	
Wine	10.40±1.35	10.20±2.15	9.20±1.75	
Zoo	15.00±1.89	14.60±1.58	14.60±1.58	

TABLE 6.1.1.3 Learning time results for different types of impurity measures (in sec.)

Data set name	ID3	ID3Gini	ID3Root	Significance
Breast	**2±0**	**3±1**	2±0	2>>1
Bupa	**3±1**	**4±0**	5±1	2>>1
Car	5±0	5±0	5±0	
Cylinder	**10±2**	**11±1**	10±1	2>1
Dermatology	3±0	3±0	3±0	
Ecoli	**3±0**	4±1	**4±1**	3>1
Flare	2±0	2±0	2±1	
Glass	**3±0**	**4±0**	4±1	2>>1
Hepatitis	1±0	2±0	1±0	
Horse	4±0	4±1	4±0	
Iris	0±0	0±0	0±0	
Ironosphere	39±7	48±7	39±7	
Monks	2±1	2±1	2±1	
Mushroom	**113±33**	**84±33**	**92±66**	3>2,1>>3
Ocrdigits	**207±9**	**254±40**	**170±24**	2>>1>>>3
Pendigits	**476±22**	**516±107**	**415±90**	3>>1>>>2
Segment	**345±10**	**493±33**	**342±56**	2>>>1>>3
Vote	1±0	1±0	1±1	
Wine	1±0	2±0	2±0	
Zoo	1±0	1±0	1±0	

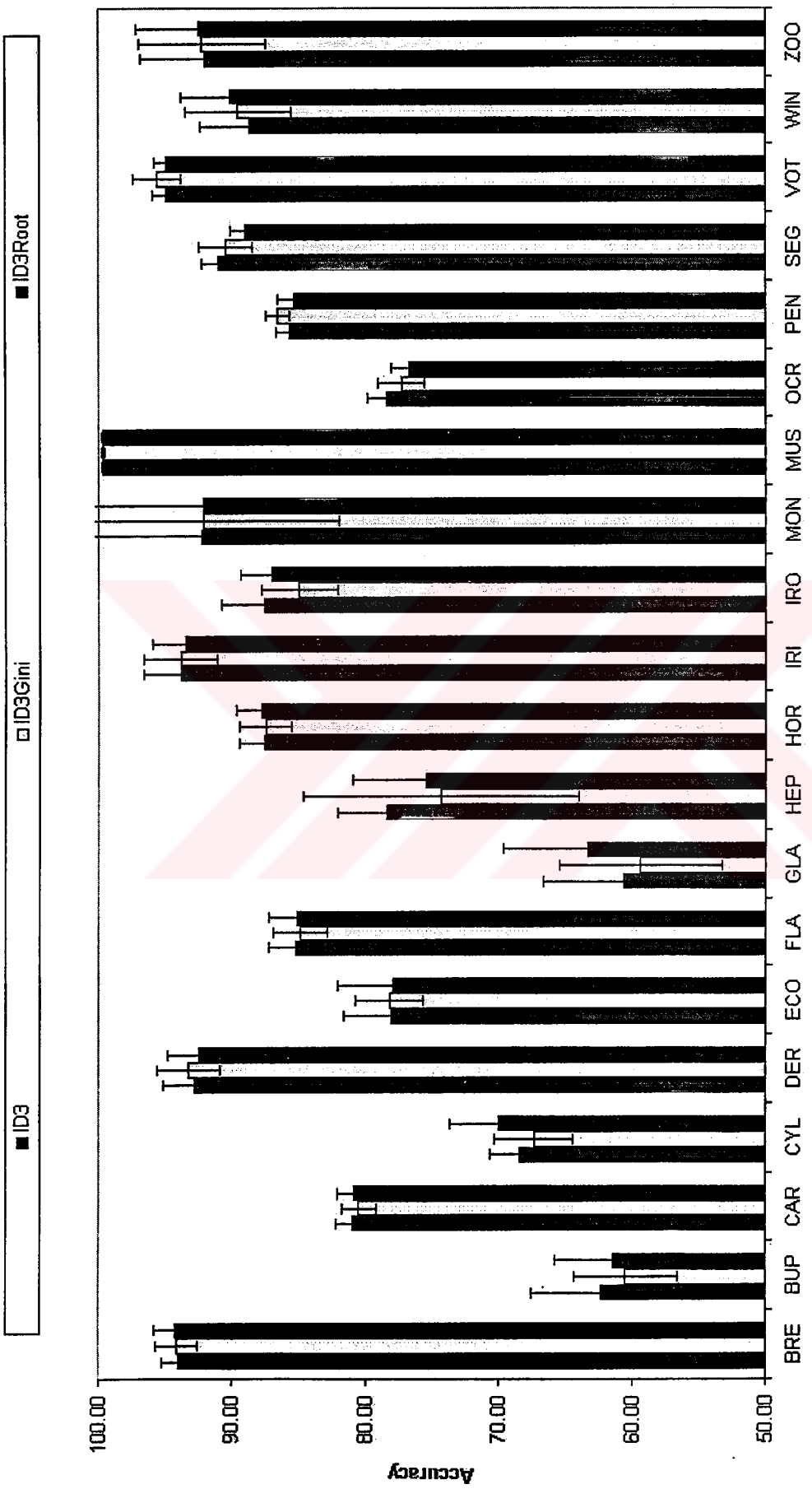

FIGURE 6.1.1.1 Accuracy results for three types of impurity measures

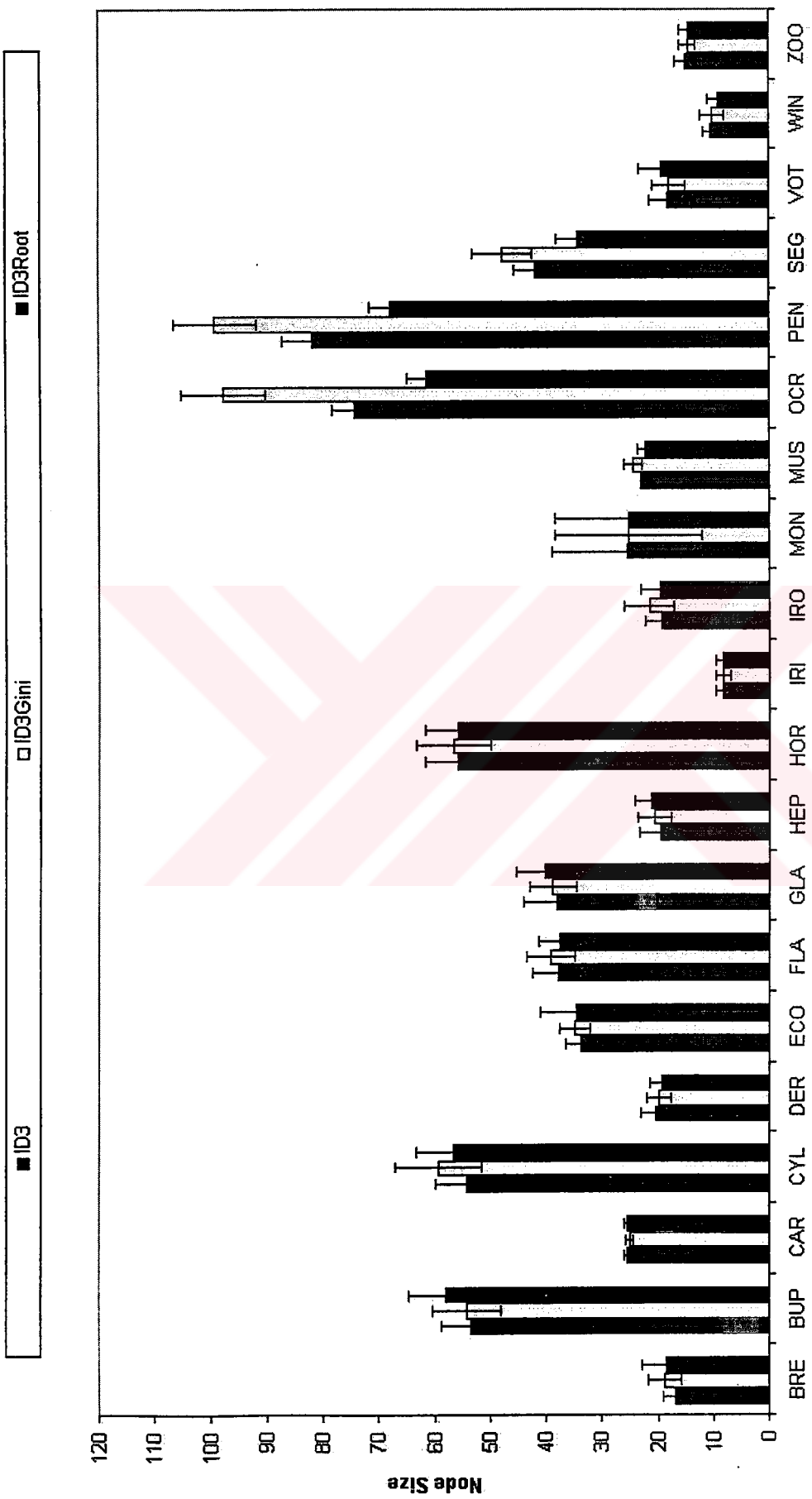

FIGURE 6.1.1.2 Node results for three types of impurity measures

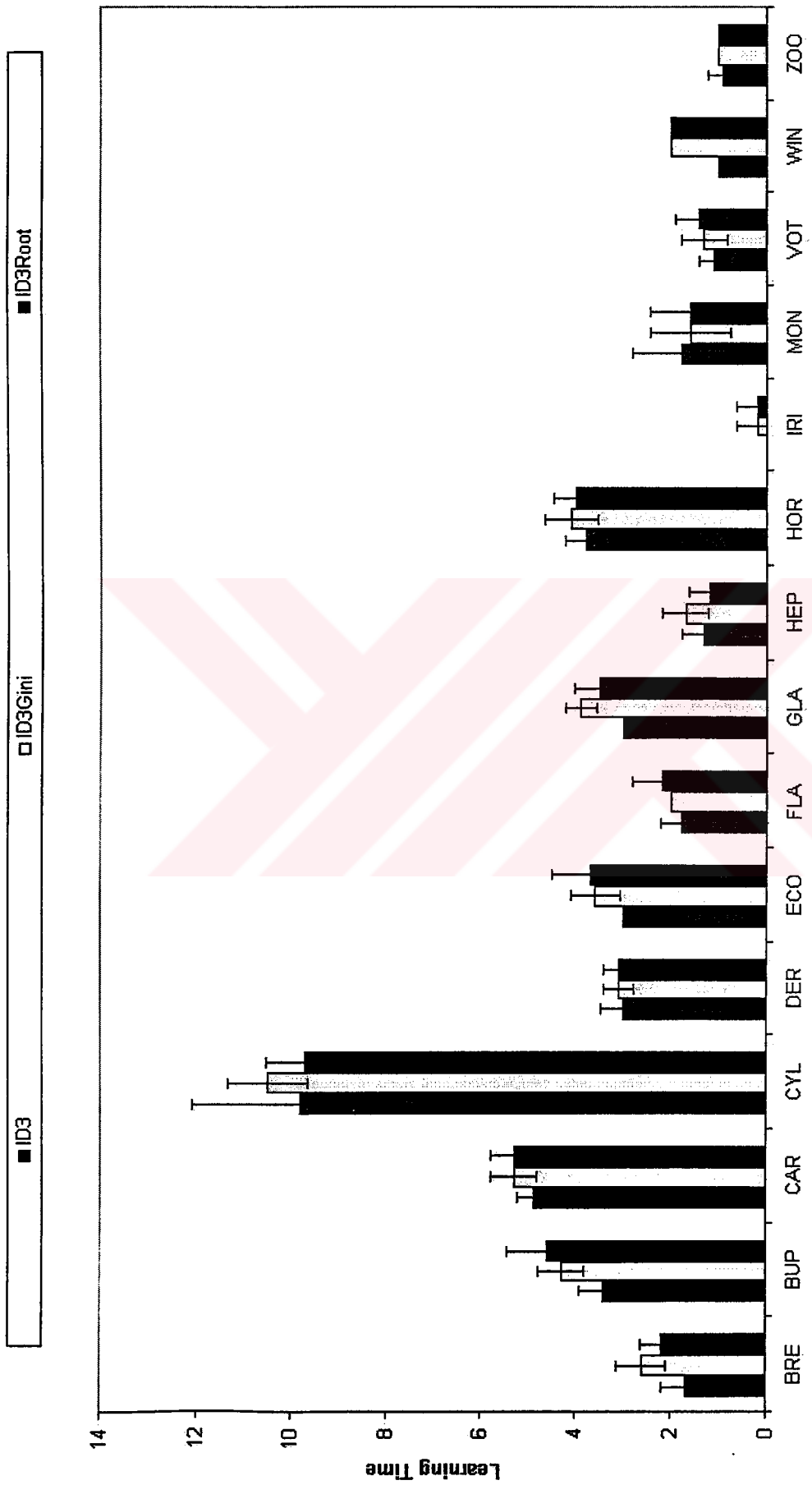

FIGURE 6.1.1.3 Learning time results for three impurity measures (small data sets)

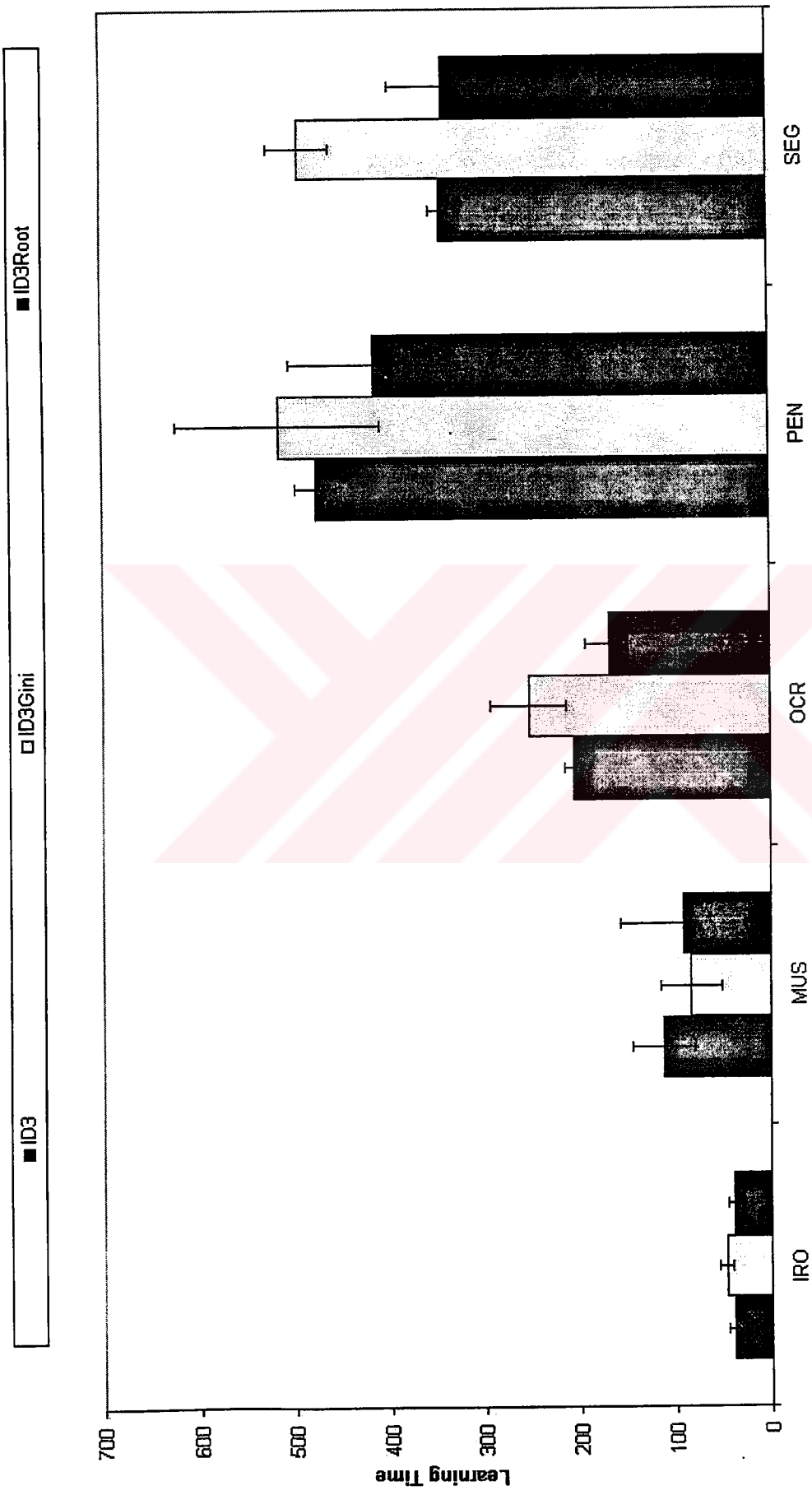

FIGURE 6.1.1.4 Learning time results for three impurity measures (large data sets)

6.1.2. Comparison of Pruning Techniques

As mentioned, two different types of pruning techniques have been used: pre-pruning and post-pruning. For simplicity, Information Gain is used as the impurity measure. In this section, we would like to find which pruning technique is better than the other. Accuracy results for these two pruning techniques are given in Table 6.1.2.1 and Figure 6.1.2.1. Node results are shown in Table 6.1.2.2 and Figure 6.1.2.2. Learning time results are shown in Table 6.1.2.3 and Figure 6.1.2.3.

TABLE 6.1.2.1 Accuracy results for pre-pruning and post-pruning techniques

Data set name	ID3	ID3P	Significance
Breast	94.11±1.24	94.68±1.84	
Bupa	62.26±5.33	62.84±.3.39	
Car	80.97±1.26	79.93±7.90	
Cylinder	68.50±2.22	67.62±5.11	
Dermatology	92.84±2.37	92.51±2.42	
Ecoli	78.10±3.57	78.27±4.00	
Flare	85.26±2.03	88.35±2.55	
Glass	60.65±5.97	60.19±5.35	
Hepatitis	78.44±3.71	78.95±4.48	
Horse	87.55±1.98	88.80±3.02	
Iris	93.87±2.75	92.93±3.33	
Ironosphere	87.63±3.15	86.15±3.72	
Monks	92.27±10.15	89.81±7.82	
Mushroom	99.70±0.06	99.87±0.11	
Ocrdigits	**78.40±1.47**	**84.34±1.48**	2>>1
Pendigits	**85.73±1.01**	**92.54±0.61**	2>>>1
Segment	91.08±1.16	91.99±0.95	
Vote	94.94±1.06	95.63±0.66	
Wine	88.65±3.72	86.63±1.94	
Zoo	92.06±4.80	82.97±7.36	

TABLE 6.1.2.2 Node results for pre-pruning and post-pruning techniques

Data set name	ID3	ID3P	Significance
Breast	17.00±2.11	13.00±4.99	
Bupa	**53.40±5.48**	**17.40±12.54**	1>>>2
Car	25.40±0.70	60.78±45.00	
Cylinder	**54.10±5.90**	**20.40±8.47**	1>>>2
Dermatology	**20.40±2.67**	**12.40±1.35**	1>>2
Ecoli	**33.80±2.70**	**14.20±4.64**	1>>>2
Flare	**37.90±4.51**	**6.10±6.62**	1>>2
Glass	**38.20±5.90**	**14.40±4.01**	1>>>2
Hepatitis	**19.60±3.78**	**2.80±2.39**	1>>2
Horse	**55.80±5.92**	**45.60±3.92**	1>>2
Iris	8.40±1.35	5.40±0.84	
Ironosphere	**19.20±3.05**	**7.60±2.67**	1>>2
Monks	25.40±13.53	25.40±9.28	
Mushroom	**23.00±0.00**	**26.80±1.99**	2>1
Ocrdigits	**74.40±4.01**	**104.40±12.44**	2>1
Pendigits	**81.80±5.51**	**134.80±13.48**	2>>1
Segment	41.80±3.79	43.00±6.93	
Vote	**18.20±3.16**	**4.00±2.16**	2>>>1
Wine	10.40±1.35	6.80±2.57	
Zoo	**15.00±1.89**	**9.20±2.39**	1>>>2

There is no significant difference in accuracy between pre-pruning and post-pruning techniques. But due to the *horizon effect,* two data sets have significant accuracy improvement by using post-pruning.

Post-pruning technique lends to less nodes then pre-pruning technique. (In 11 out of 20 data sets)

When *horizon effect* applies, the node size also increases. So in those two data sets, the node size is significantly larger then in pre-pruning technique.

In discrete data sets, where the arity is greater then five, like in *Car* and *Mushroom* data sets, post-pruning technique can not prune the tree well. So it has large number of nodes.

In some data sets, where the number of instances for one class is very high compared to the other classes, if post-pruning is applied, then the number of nodes goes to one. So the whole tree is pruned back into only one node.

Post-pruning technique takes significantly large amount of time to learn. It is because of the fact that post-pruning technique prunes the tree after its construction. In some cases, pre-pruning takes less amount of time. This is because pruning set is taken from the training set, so in post-pruning the instances in the training set is less.

TABLE 6.1.2.3 Learning time results for pre-pruning and post-pruning techniques (in sec.)

Data set name	ID3	ID3P	Significance
Breast	2±0	3±1	
Bupa	3±1	4±0	
Car	**5±0**	**40±3**	2>>1
Cylinder	**10±2**	**10±1**	1>>2
Dermatology	3±0	3±0	
Ecoli	3±0	3±0	
Flare	2±0	2±1	
Glass	3±0	2±0	
Hepatitis	1±0	1±0	
Horse	4±0	4±0	
Iris	0±0	0±0	
Ironosphere	**39±7**	**21±4**	1>>>2
Monks	2±1	3±1	
Mushroom	113±33	84±29	
Ocrdigits	**207±9**	**497±109**	2>>>1
Pendigits	**476±22**	**931±255**	2>>>1
Segment	**345±10**	**262±8**	1>>>2
Vote	**1±0**	**2±0**	2>>1
Wine	1±0	1±0	
Zoo	1±0	0±0	

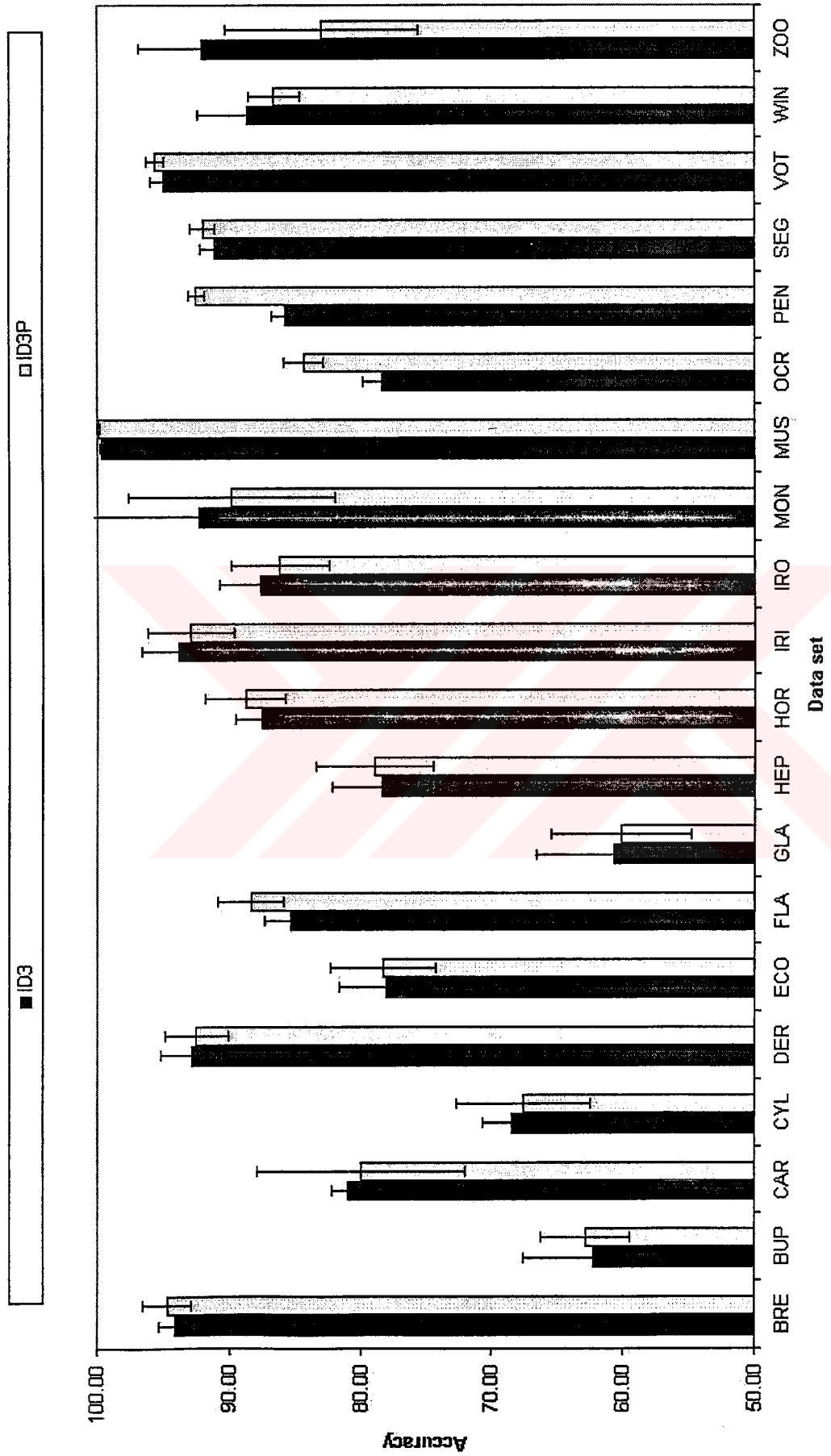

FIGURE 6.1.2.1 Accuracy results for pre-pruning and post-pruning techniques

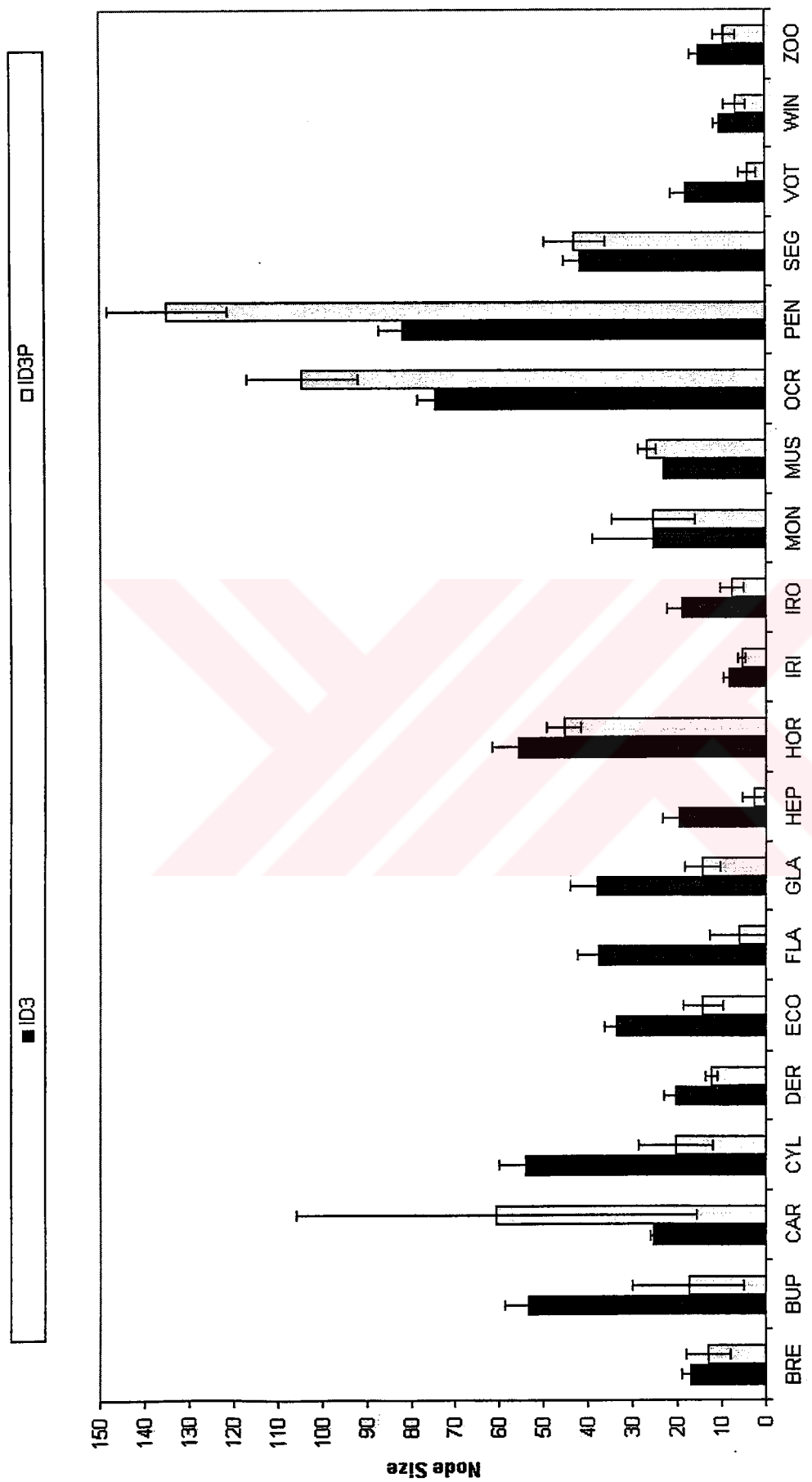

FIGURE 6.1.2.2 Node results for two pruning techniques

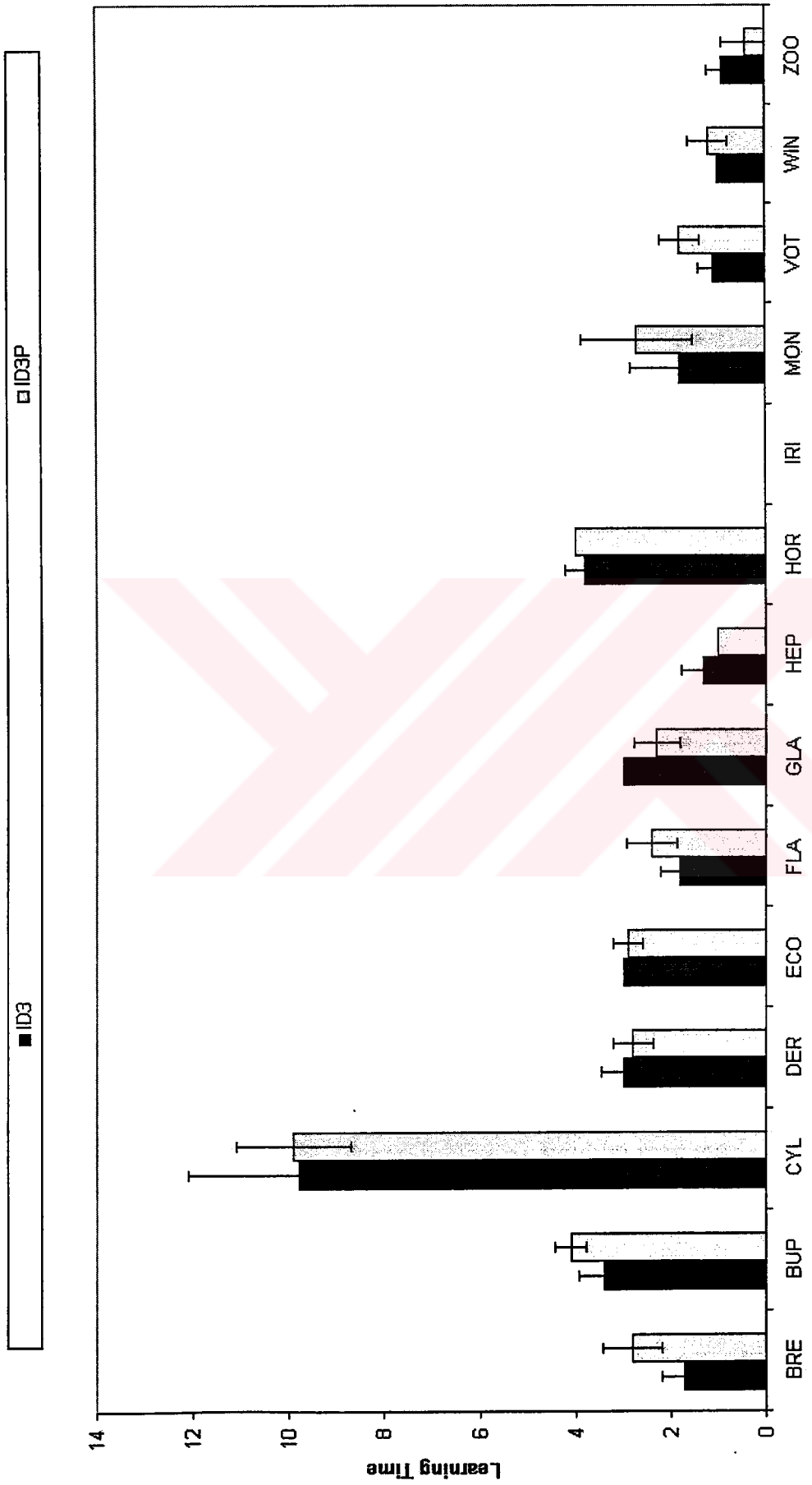

FIGURE 6.1.2.3 Learning time results for two pruning techniques (small data sets)

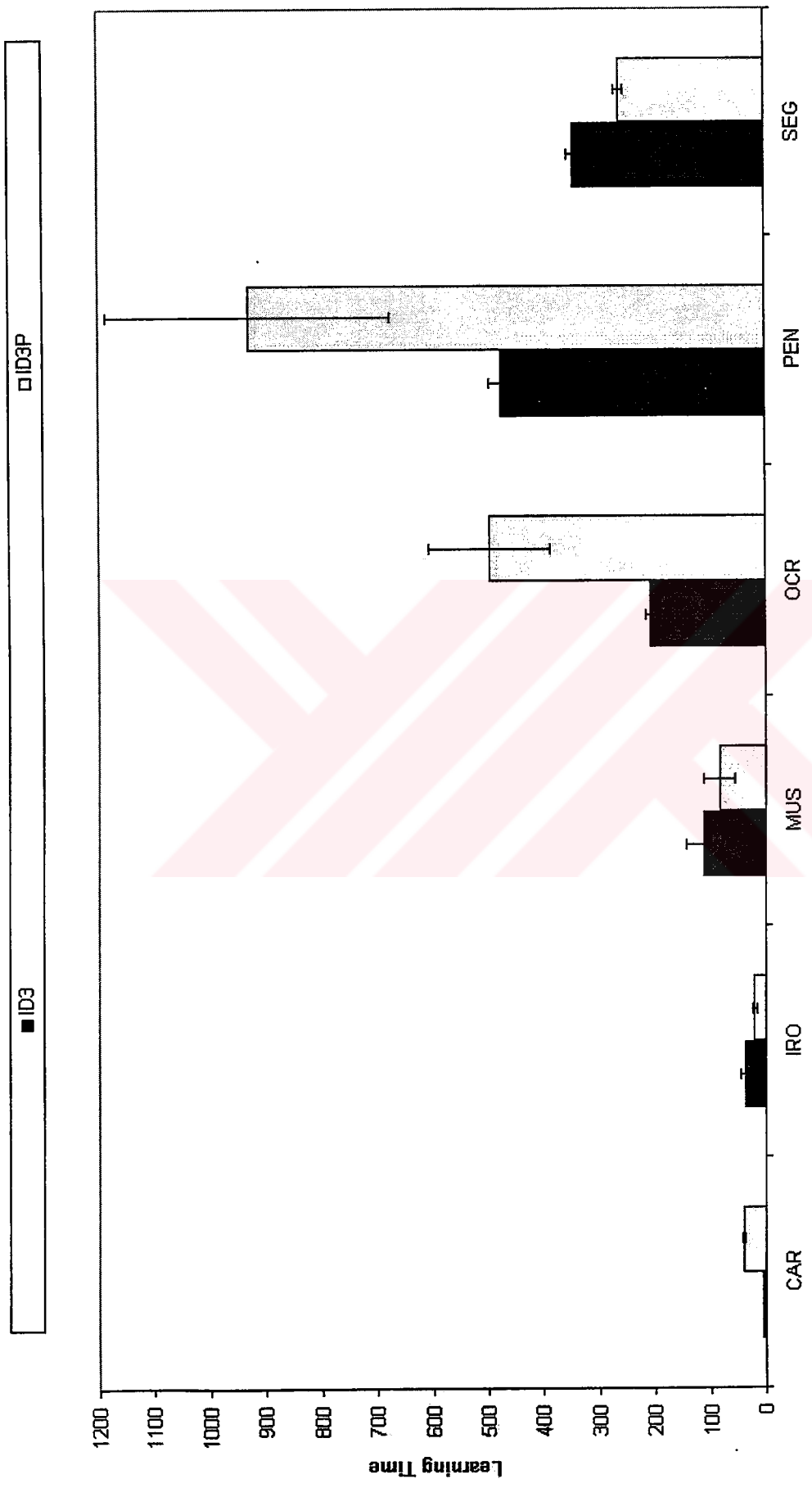

FIGURE 6.1.2.4 Learning time results for two pruning techniques (large data sets)

6.1.3. Comparison of Multiple Splits

In this section we want to find out if it is better to use multiple splits instead of binary splits. To check this, we have made experiments on the data set with three-way and four-way splits and compared it with two-way splits. The results are shown in Table 6.1.3.1 and Figure 6.1.3.1. Node results are shown in Table 6.1.3.2 and Figure 6.1.3.2. Learning time results are shown in Table 6.1.3.3, Figure 6.1.3.3 and Figure 6.1.3.4.

TABLE 6.1.3.1 Accuracy results for splits with degrees two,three and four

Data set name	ID3	ID3-2	ID3-3	Significance
Breast	**94.11±1.24**	94.08±1.38	**93.65±0.87**	1>3
Bupa	**62.26±5.33**	**59.41±4.61**	59.70±2.78	1>>>2
Cylinder	68.50±2.22	63.62±4.08	65.44±5.66	
Dermatology	92.84±2.37	92.46±1.80	91.37±2.51	
Ecoli	78.10±3.57	76.61±3.97	75.24±4.61	
Flare	85.26±2.03	85.26±2.03	85.26±2.03	
Glass	60.65±5.97	56.92±5.82	54.21±4.89	
Hepatitis	78.44±3.71	73.38±8.65	71.07±8.41	
Horse	87.55±1.98	87.12±2.22	86.90±2.37	
Iris	93.87±2.75	92.67±3.28	92.93±2.27	
Ironosphere	87.63±3.15	87.63±1.39	N/A	
Monks	**92.27±10.15**	**91.53±7.29**	**80.28±8.26**	1>>2,1>>3
Ocrdigits	**78.40±1.47**	**67.25±2.24**	**63.41±1.72**	1>>>2>>>3
Pendigits	**85.73±1.01**	**82.19±1.47**	N/A	1>>2
Segment	91.08±1.16	N/A	N/A	
Wine	**88.65±3.72**	**86.63±5.28**	83.03±4.90	
Zoo	**92.06±4.80**	**87.10±4.96**	88.69±5.35	1>>2

TABLE 6.1.3.2 Node results for splits with degrees two,three and four

Data set name	ID3	ID3-2	ID3-3	Significance
Breast	17.00±2.11	17.90±3.90	18.80±3.36	
Bupa	53.40±5.48	**50.70±3.53**	**54.40±5.58**	3>>2
Cylinder	54.10±5.90	52.40±5.21	54.80±4.85	
Dermatology	**20.40±2.67**	20.30±3.56	**27.00±3.89**	3>>1
Ecoli	33.80±2.70	34.40±3.17	36.90±3.87	
Flare	37.90±4.51	37.20±3.99	37.80±3.55	
Glass	38.20±5.90	38.70±4.60	37.30±5.93	
Hepatitis	19.60±3.78	20.60±3.81	21.40±3.95	
Horse	55.80±5.92	57.50±6.59	58.20±6.92	
Iris	8.40±1.35	8.00±2.26	8.10±1.97	
Ironosphere	19.20±3.05	20.60±3.24	N/A	
Monks	**25.40±13.53**	**33.90±6.76**	**38.50±5.04**	2>>1,3>>>1
Ocrdigits	74.40±4.01	63.50±4.03	67.90±5.61	
Pendigits	81.80±5.51	73.60±5.40	N/A	
Segment	41.80±3.79	N/A	N/A	
Wine	**10.40±1.35**	12.90±1.91	**15.00±2.36**	3>1
Zoo	**15.00±1.89**	**16.20±1.99**	16.50±2.17	2>1

For multiple splits the accuracy decreases while the degree of the split is increased from two to four; this difference is significant in six out of 20 data sets. This may be due to the fragmentation problem.

The number of nodes also increases when the degree of the split increases. Only in some small data sets there is a drop in node size from going degree two to three.

Learning time of higher degree splits is significantly greater then lower degree splits.

The accuracy, number of nodes and learning time does not change in data sets where all attributes are discrete (as can be expected).

TABLE 6.1.3.3 Learning time results for splits with degrees two,three and four

Data set name	ID3	ID3-2	ID3-3	Significance
Breast	**2±0**	**4±0**	**11±1**	3>>>2>>>1
Bupa	**3±1**	**15±2**	**168±63**	3>>>2>>>1
Cylinder	**10±2**	**70±10**	**1013±246**	3>>2>>>1
Dermatology	**3±0**	**7±1**	36±21	2>>1
Ecoli	**3±0**	**23±1**	**419±53**	3>>>2>>>1
Flare	2±0	2±0	2±0	
Glass	**3±0**	**29±4**	**556±105**	3>>>2>>>1
Hepatitis	**1±0**	**5±1**	**41±8**	3>>>2>>>1
Horse	**4±0**	**14±2**	**158±80**	3>2>>>1
Iris	**0±0**	**1±0**	**13±2**	3>>>2>>1
Ironosphere	**39±7**	**604±80**	N/A	2>>>1
Monks	**2±1**	**2±0**	**3±1**	2>>1,3>>>1
Ocrdigits	**207±9**	**596±98**	**2384±352**	3>>>2>>>1
Pendigits	**476±22**	**9272±2356**	N/A	2>>>1
Segment	345±10	N/A	N/A	
Wine	**1±0**	**18±3**	**266±60**	3>>>2>>>1
Zoo	1±0	1±0	1±0	

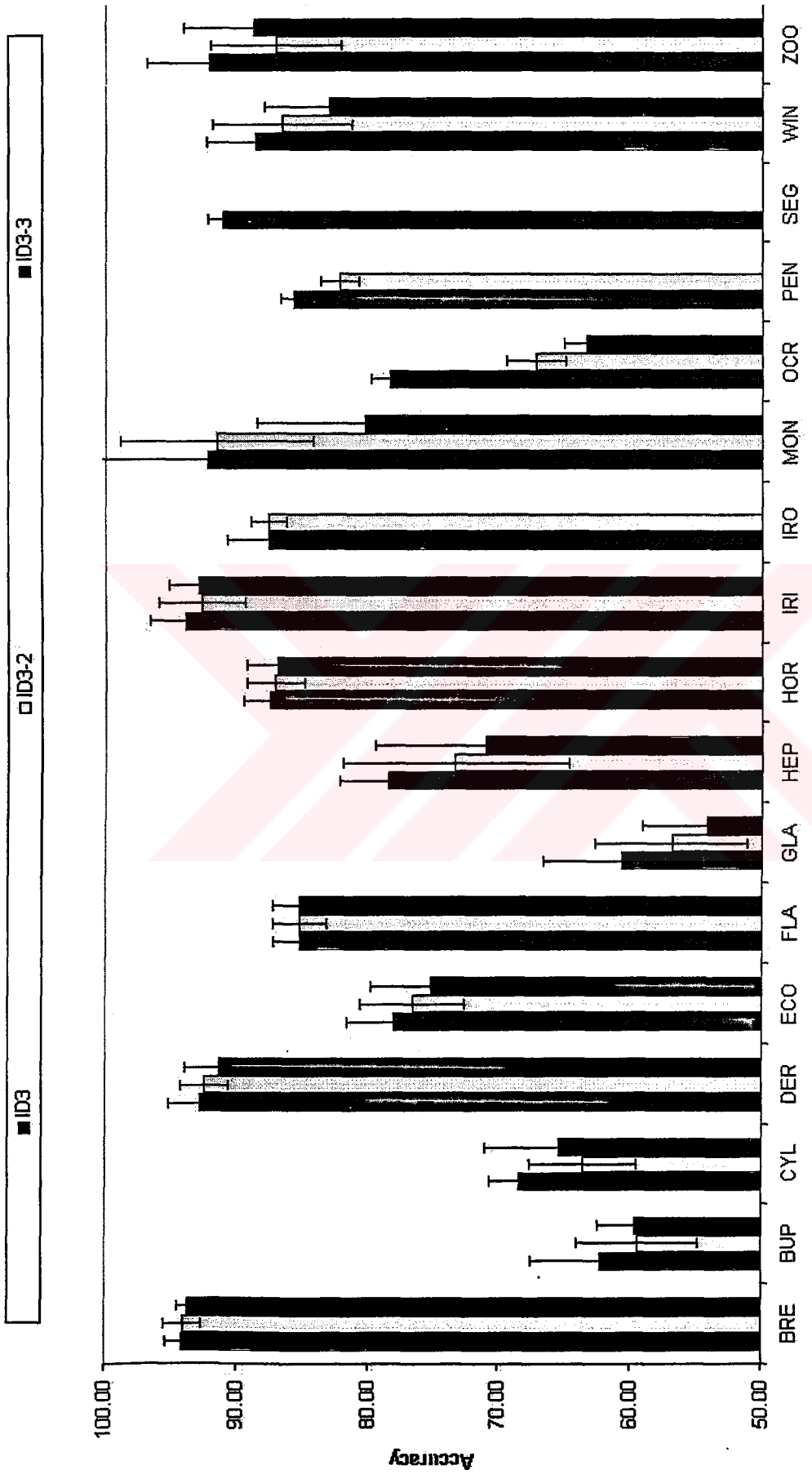

FIGURE 6.1.3.1 Accuracy results for splits with degrees two, three and four.

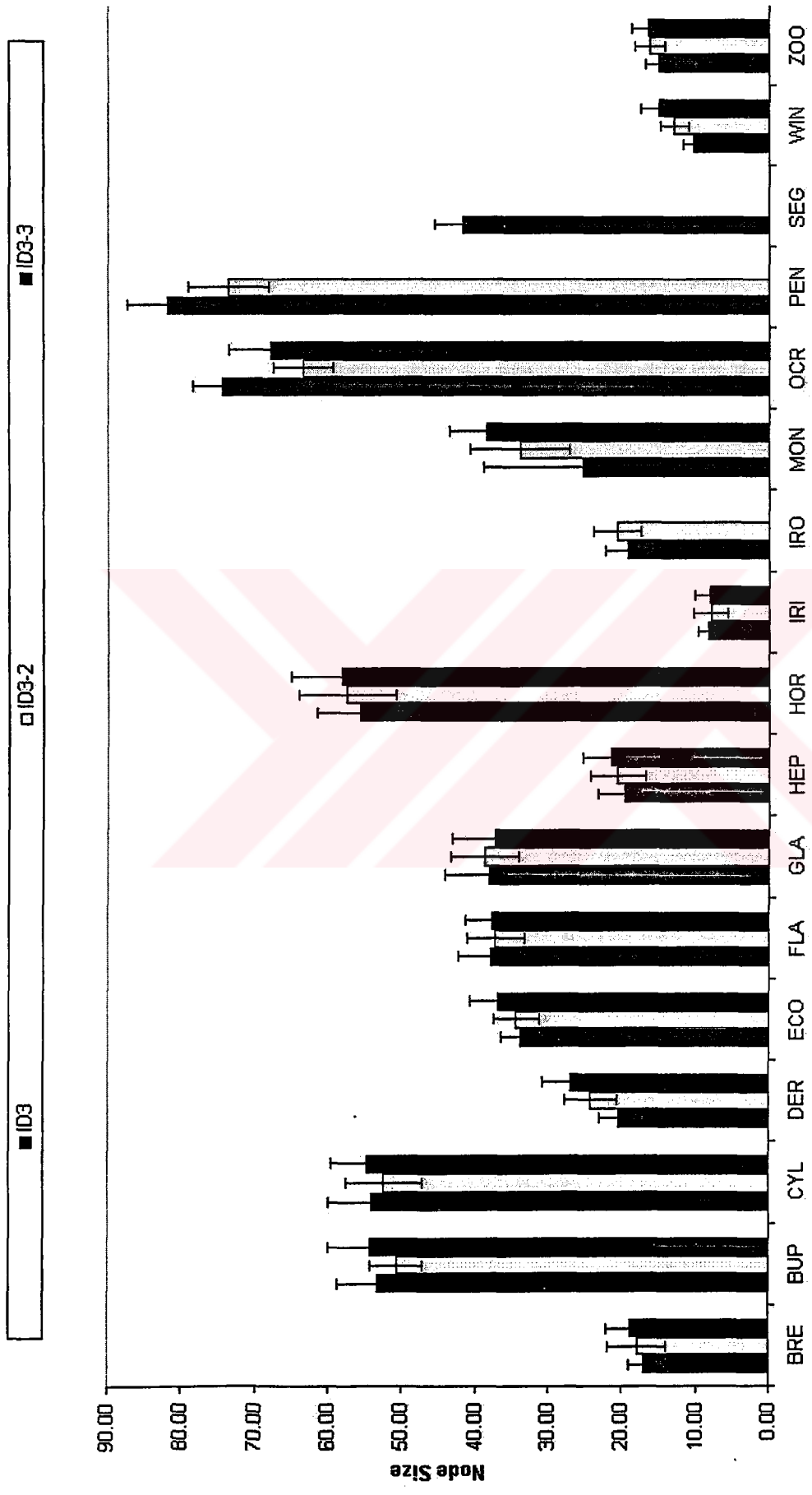

FIGURE 6.1.3.2 Node results for splits with degrees two, three and four.

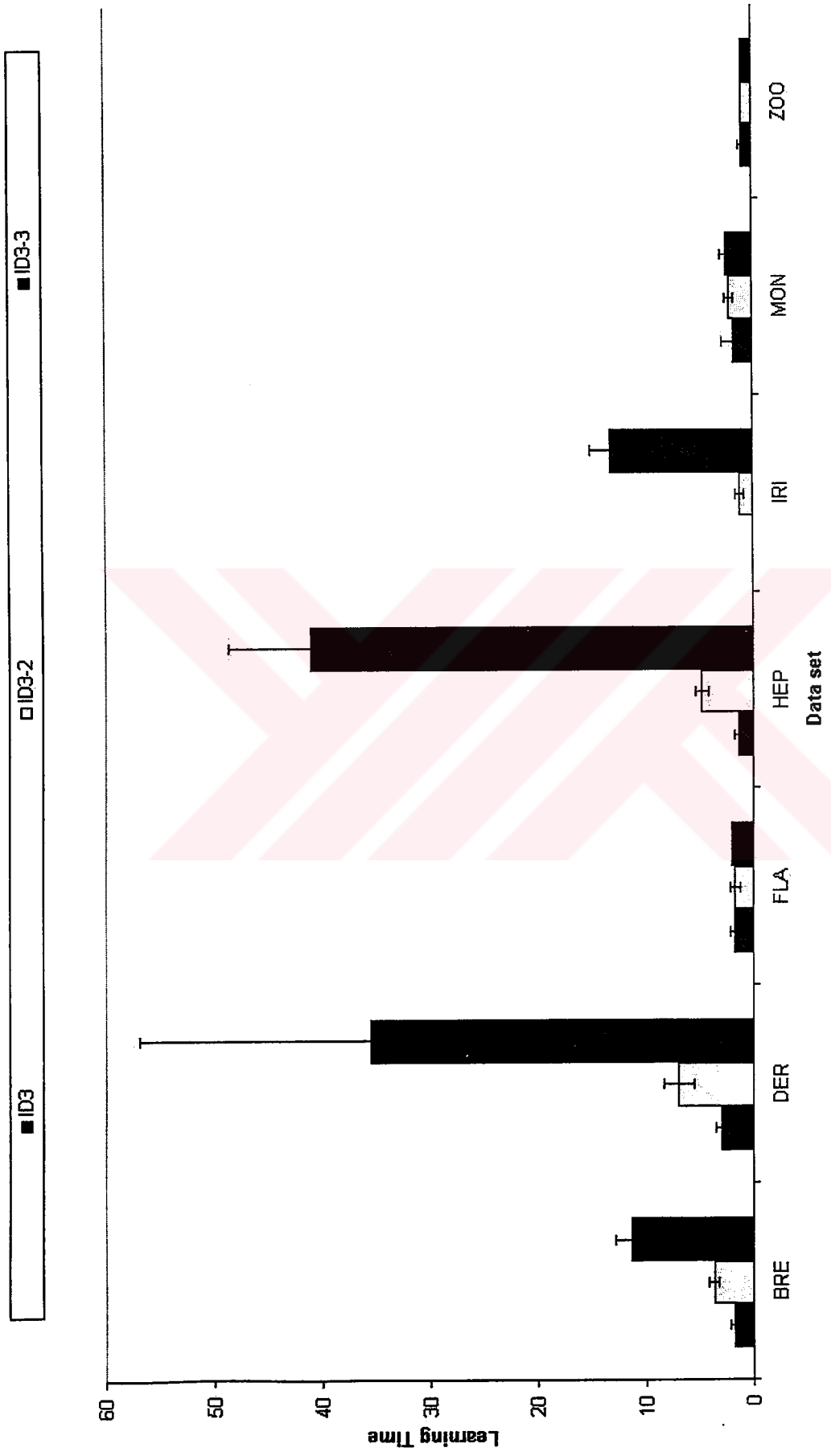

FIGURE 6.1.3.3 Learning time for splits degrees two, three and four (small data sets).

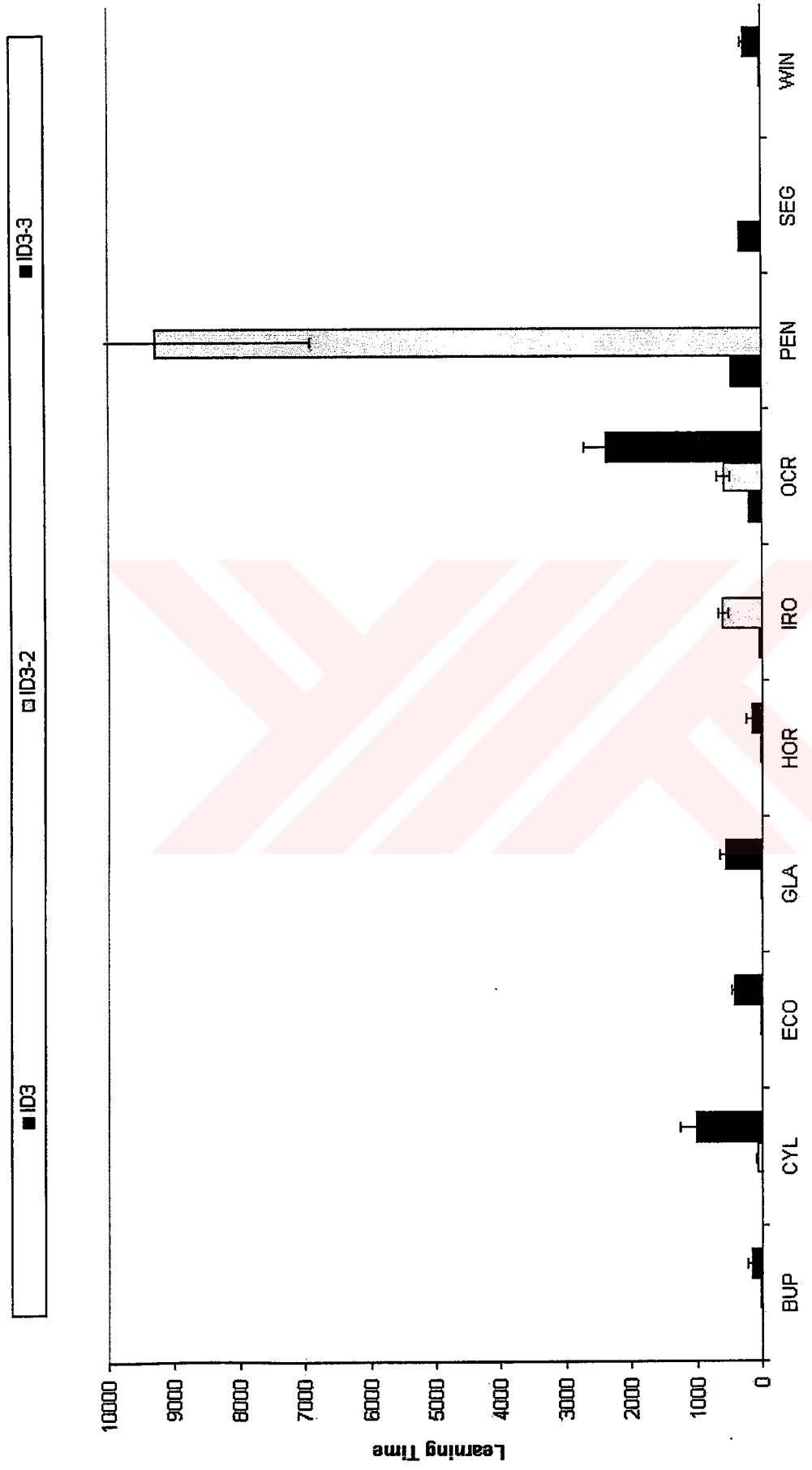

FIGURE 6.1.3.4 Learning time for splits with degrees two, three and four (large data sets).

6.2. Results for Classification and Regression Trees

For the rest of these results, the definition given in Table 6.2.1 applies.

TABLE 6.2.1 Definition of tree-based methods

Name of the Method	Uni/Multi	Impurity Measure	Pruning	Feature Selection
ID3	Uni	Information Gain	Pre-pruning	No
CART	Multi	Information Gain	Pre-pruning	No
FSCART	Multi	Information Gain	Pre-pruning	Yes

In this section, the multivariate method CART and the univariate method ID3 are compared. Also feature selection is applied to CART to see if there will be a difference in accuracy or node size. The results of this comparison are shown in Table 6.2.2 and in Figure 6.2.1. Node results are shown in Table 6.2.3 and Figure 6.2.2. Learning time results are shown in Table 6.2.3, Figure 6.2.3 and Figure 6.2.4.

ID3 is statistically significantly better than CART in six out 20 data sets. ID3 is better then FSCART in two cases with 95% and FSCART is better then ID3 in one case with 99% confidence. So we can say that no one of the three methods is clearly the best.

Concerning the tree size, generally multivariate techniques FSCART and CART perform better than ID3 and this is significant in six data sets. Note that a CART node internally is more complex then an ID3 node.

The learning times of univariate technique ID3 is significantly better than multivariate techniques CART and FSCART.

Feature selection improves accuracy and node size generally but increases learning time significantly.

TABLE 6.2.2 Accuracy results for ID3 and CART

Data set name	ID3	CART	FSCART	Significance
Breast	94.11±1.24	94.85±1.44	94.65±1.23	
Bupa	62.26±5.33	61.74±3.38	61.50±2.48	
Car	**80.97±1.26**	**83.84±2.03**	**78.34±7.40**	1>3,2>>3
Cylinder	**68.50±2.22**	**59.52±4.05**	N/A	1>2
Dermatology	**92.84±2.37**	**80.87±4.56**	83.72±7.66	1>>2
Ecoli	**78.10±3.57**	74.74±3.80	**76.90±3.89**	1>>3
Flare	85.26±2.03	81.55±3.60	85.75±3.89	
Glass	60.65±5.97	53.93±4.20	58.13±6.58	
Hepatitis	78.44±3.71	78.96±4.04	78.58±3.72	
Horse	**87.55±1.98**	**76.96±3.02**	N/A	1>>2
Iris	93.87±2.75	89.33±4.44	90.40±4.48	
Ironosphere	**87.63±3.15**	86.84±4.03	**84.78±2.78**	1>3
Monks	92.27±10.15	91.20±6.89	82.87±8.09	
Mushroom	**99.70±0.06**	**93.45±1.75**	N/A	1>>2
Ocrdigits	78.40±1.47	81.35±2.08	N/A	
Pendigits	**85.73±1.01**	87.10±2.91	**91.47±0.86**	3>>>1
Segment	91.08±1.16	**88.07±1.69**	**92.46±1.71**	3>2
Vote	**94.94±1.06**	**90.30±3.17**	90.44±3.88	1>>2
Wine	88.65±3.72	87.30±4.40	93.03±3.62	
Zoo	**92.06±4.80**	**69.92±9.69**	**69.33±8.93**	1>>2,1>>3

TABLE 6.2.3 Node results for ID3 and CART

Data set name	ID3	CART	FSCART	Significance
Breast	**17.00±2.11**	**11.60±2.67**	**10.80±2.39**	1>2,1>3
Bupa	**53.40±5.48**	**43.20±3.82**	**40.60±4.20**	1>2,1>>3
Car	25.40±0.70	29.00±3.40	30.00±4.14	
Cylinder	54.10±5.90	45.00±4.90	N/A	
Dermatology	**20.40±2.67**	28.00±4.74	**20.80±3.46**	3>>1
Ecoli	33.80±2.70	34.00±5.01	31.40±3.24	
Flare	**37.90±4.51**	**33.80±6.20**	25.80±9.48	1>>2
Glass	38.20±5.90	42.40±4.12	38.20±4.34	
Hepatitis	**19.60±3.78**	14.00±3.43	**11.60±1.90**	1>>3
Horse	**55.80±5.92**	**28.00±5.19**	N/A	1>>2
Iris	8.40±1.35	10.20±2.35	8.20±1.40	
Ironosphere	19.20±3.05	16.40±3.78	16.00±3.68	
Monks	**25.40±13.53**	**17.80±10.16**	**11.40±2.27**	1>2,1>>>3
Mushroom	**23.00±0.00**	**43.00±6.53**	N/A	2>>>1
Ocrdigits	**74.40±4.01**	**70.80±3.98**	N/A	1>>2
Pendigits	81.80±5.51	77.80±10.08	71.00±5.16	
Segment	41.80±3.79	45.20±8.97	36.80±4.57	
Vote	18.20±3.16	17.20±5.29	18.20±5.75	
Wine	10.40±1.35	9.40±2.27	9.00±1.33	
Zoo	**15.00±1.89**	**25.20±4.94**	**16.40±2.32**	2>>1,2>>>3

TABLE 6.2.4 Learning time results for ID3 and CART

Data set name	ID3	CART	FSCART	Significance
Breast	**2±0**	**107±17**	**482±122**	3>>>2>>>1
Bupa	**3±1**	**252±23**	**829±119**	3>>>2>>>1
Car	**5±0**	**1178±148**	**13056±1661**	3>>>2>>>1
Cylinder	**10±2**	**4589±343**	N/A	2>>>1
Dermatology	**3±0**	**858±170**	**10553±1748**	3>>>2>>>1
Ecoli	**3±0**	**221±25**	**859±76**	3>>>2>>>1
Flare	**2±0**	**1032±203**	**8892±3203**	3>>2>>>1
Glass	**3±0**	**320±25**	**1481±154**	3>>>2>>>1
Hepatitis	**1±0**	**209±47**	**1709±265**	3>>>2>>>1
Horse	**4±0**	**3481±1101**	N/A	2>>1
Iris	**0±0**	**31±11**	**69±17**	3>2>>1
Ironosphere	**39±7**	**544±94**	**8664±1662**	3>>>2>>>1
Monks	**2±1**	**126±61**	**273±55**	3>>>2>>1
Mushroom	**113±33**	**33613±2942**	N/A	2>>>1
Ocrdigits	**207±9**	**9148±713**	N/A	2>>>1
Pendigits	**476±22**	**3311±350**	**24544±1374**	3>>>2>>>1
Segment	**345±10**	**1212±170**	**9173±905**	3>>>2>>>1
Vote	**1±0**	**805±167**	**13058±3920**	3>>>2>>>1
Wine	**1±0**	**84±26**	**566±147**	3>>>2>>>1
Zoo	**1±0**	**453±61**	**2088±375**	3>>>2>>>1

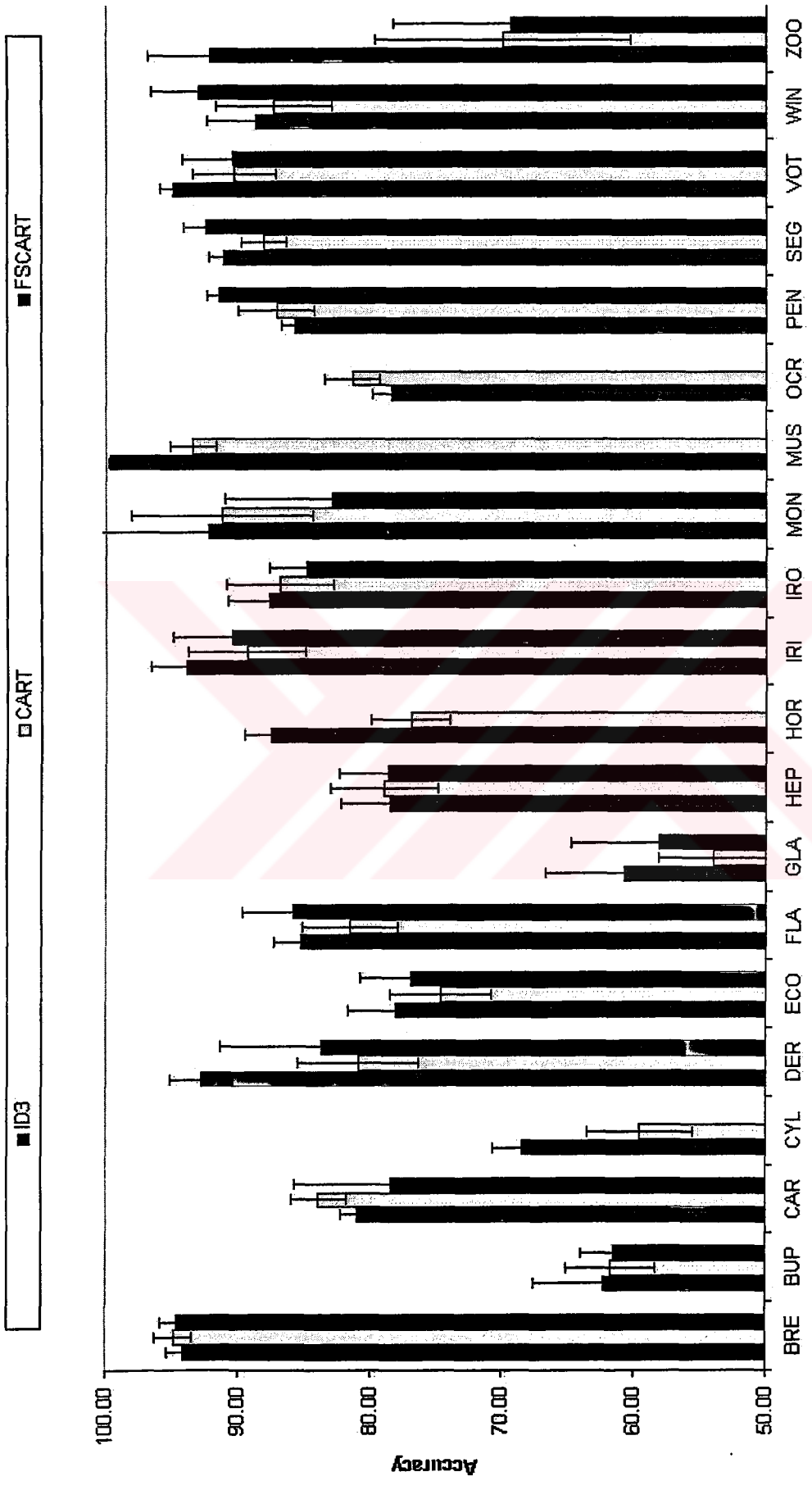

FIGURE 6.2.1 Accuracy results for ID3 and CART

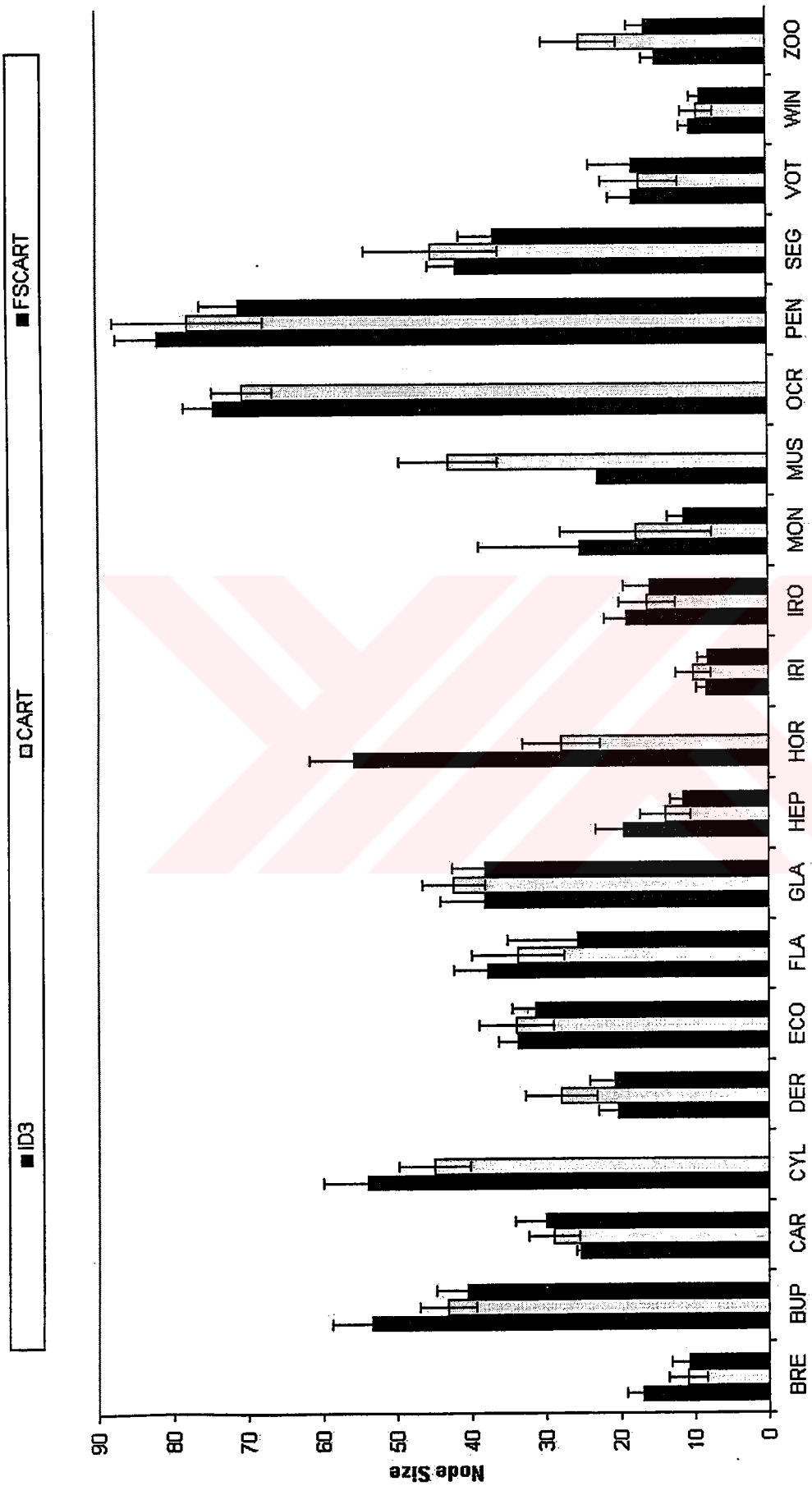

FIGURE 6.2.2 Node results for ID3 and CART

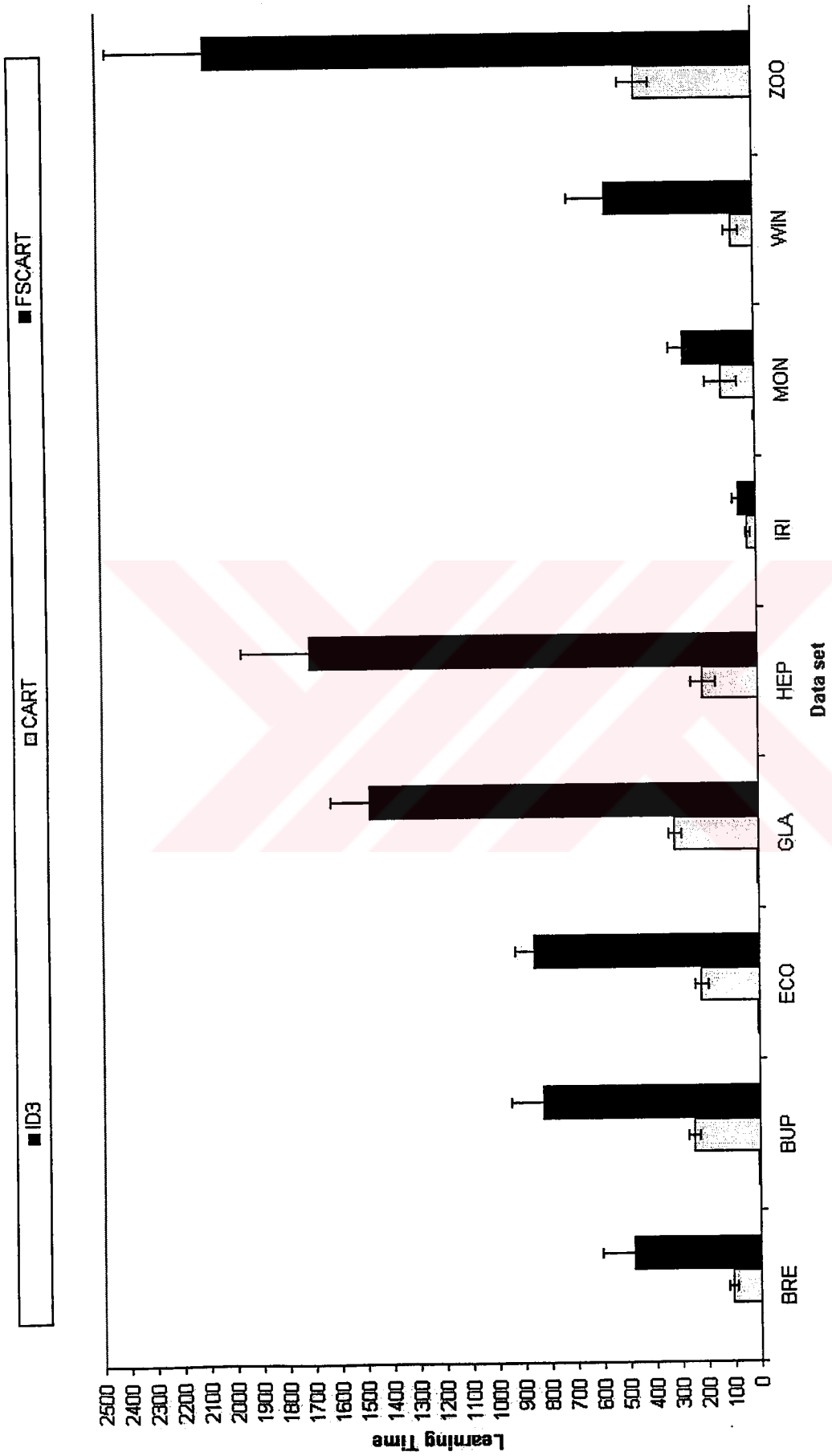

FIGURE 6.2.3 Learning time results for ID3 and CART (small data sets)

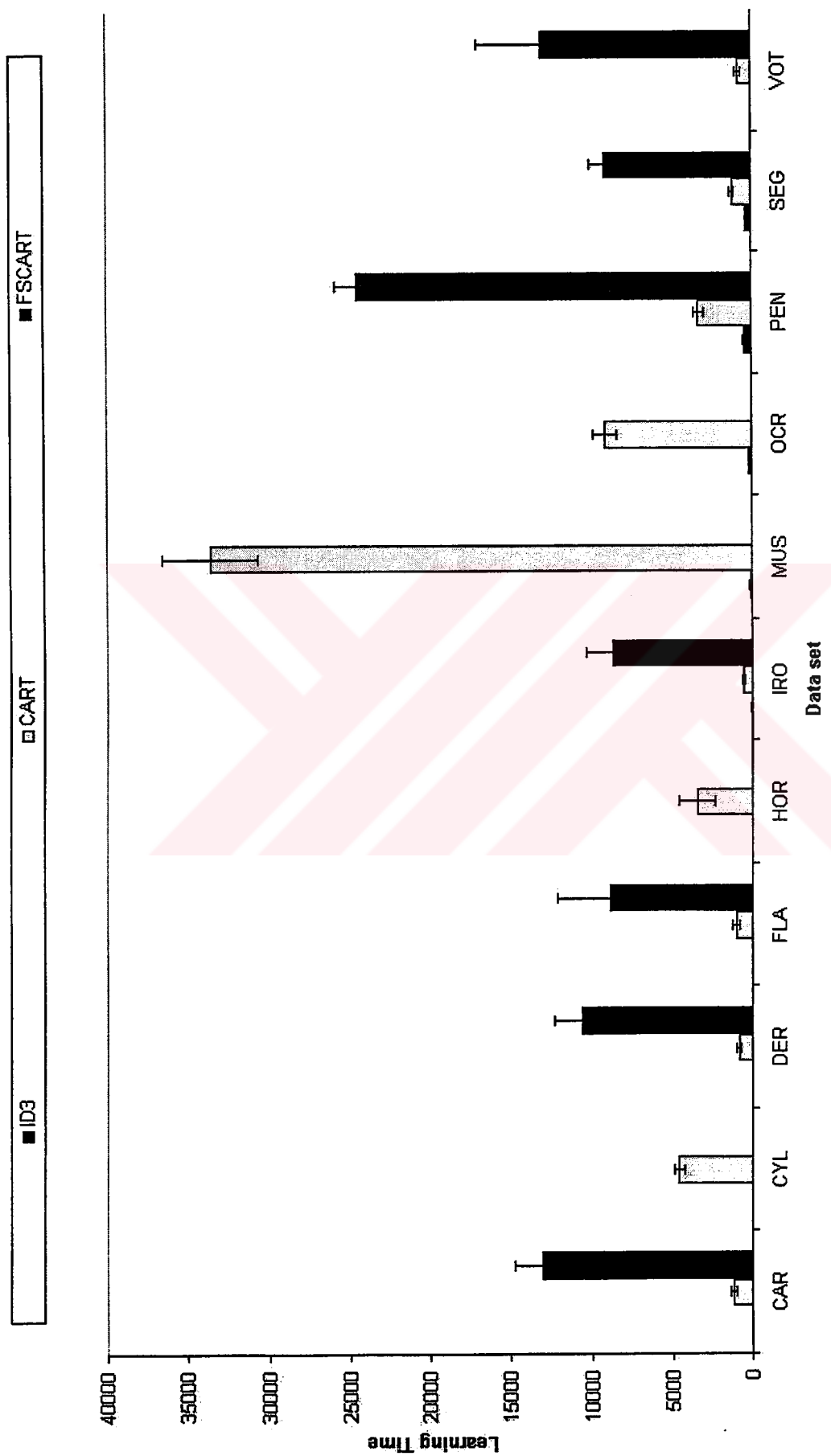

FIGURE 6.2.4 Learning time results for ID3 and CART (large data sets)

6.3. Results for Neural Network Methods

For the rest of these results, the definition given in Table 6.3.1 applies.

TABLE 6.3.1 Definition of neural-network based methods

Name	Class Separation	Impurity Measure	Pruning	Linearity
ID-LPS	Selection	Information Gain	Pre-pruning	Linear
ID-LPE	Exchange	Information Gain	Pre-pruning	Linear
ID-MLPE	Exchange	Information Gain	Pre-pruning	Nonlinear
ID-Hybrid-F	Exchange	Information Gain	Pre-pruning	Both with F-test
ID-Hybrid-t	Exchange	Information Gain	Pre-pruning	Both with t-test

6.3.1. Comparison of Class Separation Techniques

The aim of this section is to find which class separation technique (selection or exchange) is better than the other. For simplicity, other variables such as impurity measure or pruning technique are fixed. If there are only two classes available in a data set, it is not included in the results because there will be no class separation. The results are shown in Table 6.3.1.1 and Figure 6.3.1.1. Node results are shown in Table 6.3.1.2 and Figure 6.3.1.2. Learning time results are shown in Table 6.3.1.3, Figure 6.3.1.4 and Figure 6.3.1.5.

In none of the data sets, the selection method is more accurate than the exchange method in accuracy. But the exchange method is more accurate than selection method in three data sets. Two of these data sets *Ocrdigits* and *Pendigits* have 10 classes. The other data set *Ecoli* has eight classes. So we can conclude that, the more classes you have, the better is the exchange method compared to the selection method, due to the large number of division candidates.

If the node size results are compared, it is also seen that in two data sets, *Pendigits* and *Glass* (which has eight classes), out of 11, the exchange method is better than the selection method while the selection is never better.

As we have explained, the exchange method has larger time complexity. So in all data sets except one, the selection method is better than the exchange method in terms of learning time. This significance also increases with the size of the data set and the number of classes.

TABLE 6.3.1.1 Accuracy results for ID-LPS and ID-LPE

Data set name	ID-LPS	ID-LPE	Significance
Car	87.50±3.07	89.48±4.01	
Dermatology	69.51±22.01	85.74±7.06	
Ecoli	**68.51±5.39**	**82.62±4.06**	2>1
Flare	88.17±2.21	88.36±2.37	
Glass	55.53±6.16	54.95±7.83	
Iris	81.73±14.40	77.60±15.70	
Ocrdigits	**54.14±6.25**	**93.87±0.92**	2>>>1
Pendigits	**67.46±5.44**	**91.94±4.16**	2>>>1
Segment	70.55±6.68	79.76±11.58	
Wine	85.06±14.00	87.75±12.62	
Zoo	78.01±7.67	79.38±8.10	

TABLE 6.3.1.2 Node results for ID-LPS and ID-LPE

Data set name	ID-LPS	ID-LPE	Significance
Car	11.40±6.10	7.40±0.84	
Dermatology	7.40±3.10	8.80±1.48	
Ecoli	15.00±5.33	10.80±2.90	
Flare	2.80±1.99	3.20±2.20	
Glass	**20.80±3.46**	**10.20±4.64**	1>>2
Iris	5.60±3.13	4.00±1.05	
Ocrdigits	45.20±4.76	34.80±4.94	
Pendigits	**58.40±9.52**	**30.40±6.40**	1>>>2
Segment	28.60±6.31	16.60±6.65	
Wine	4.20±1.03	4.40±0.97	
Zoo	11.40±2.07	8.80±1.75	

TABLE 6.3.1.3 Learning time results for ID-LPS and ID-LPE

Data set name	ID-LPS	ID-LPE	Significance
Car	**79±17**	**152±16**	2>>>1
Dermatology	**22±4**	**42±9**	2>>1
Ecoli	**22±4**	**57±15**	2>>1
Flare	**5±2**	**9±4**	2>>1
Glass	**13±1**	**33±9**	2>>>1
Iris	**2±0**	**3±0**	2>1
Ocrdigits	**2764±384**	**8035±757**	2>>>1
Pendigits	**4164±246**	**18340±3319**	2>>>1
Segment	**407±63**	**937±103**	2>>>1
Wine	2±0	4±1	
Zoo	**5±1**	**10±2**	2>>1

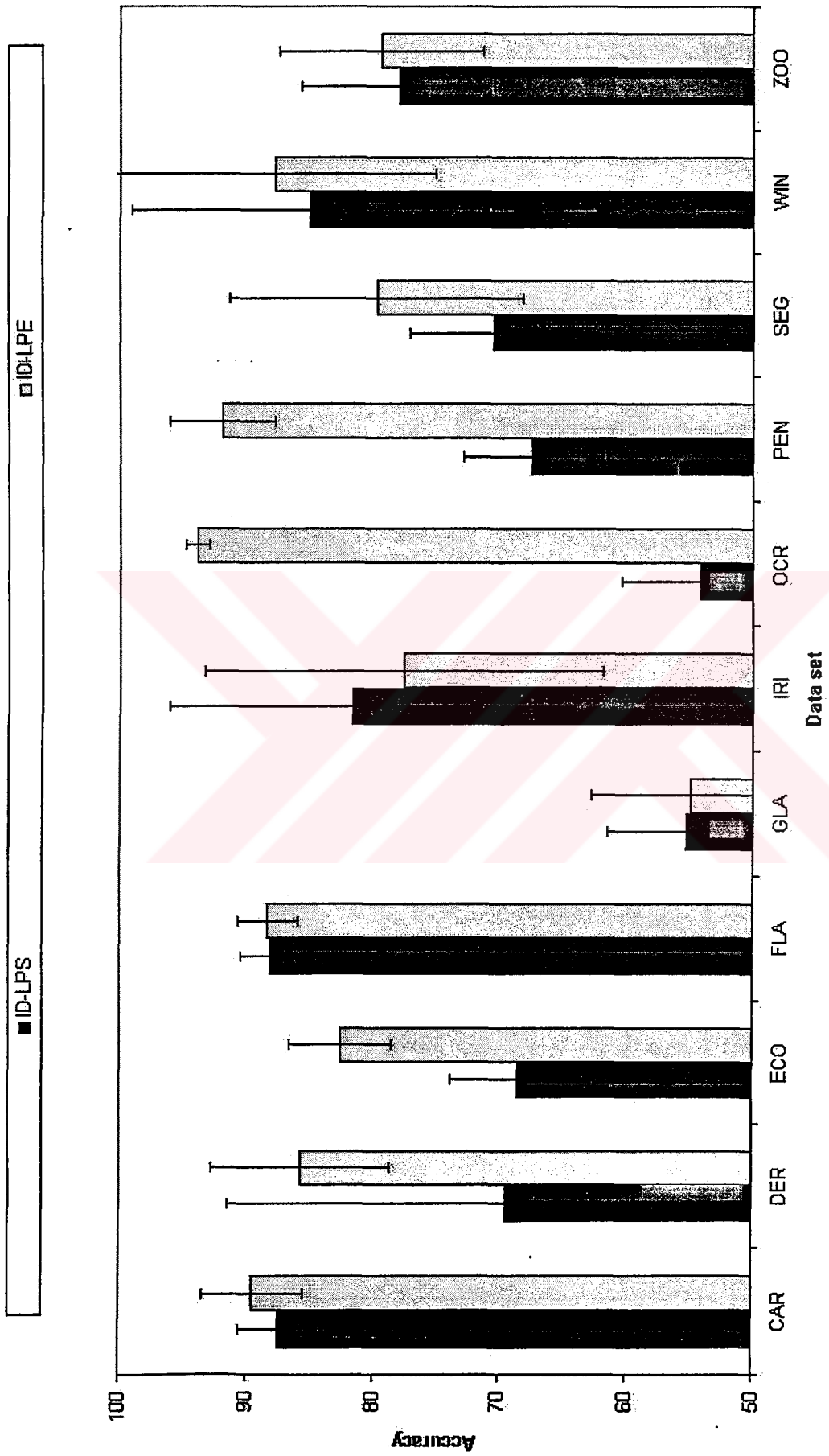

FIGURE 6.3.1.1 Accuracy results for ID-LPS and ID-LPE

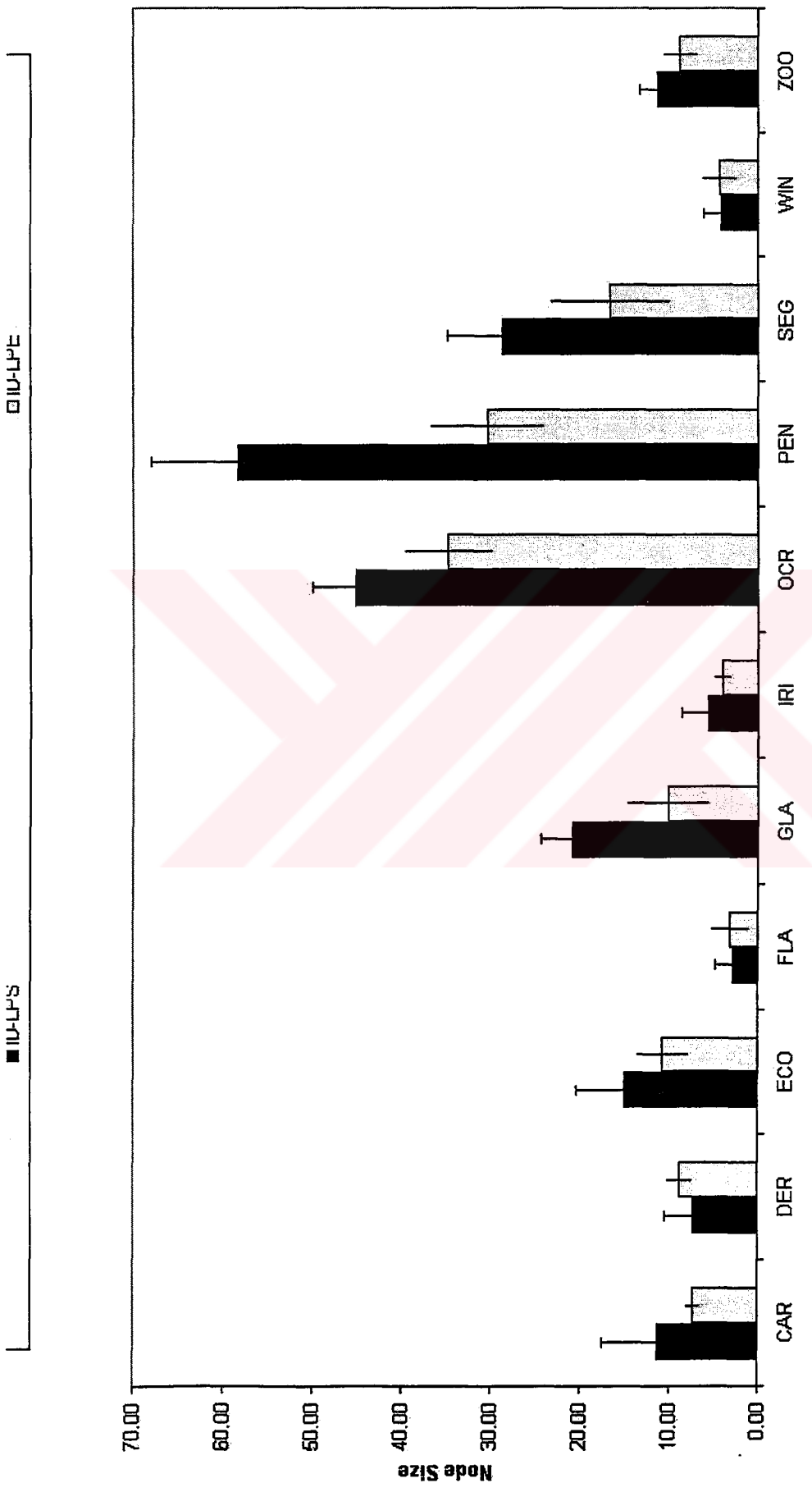

FIGURE 6.3.1.2 Node results for ID-LPS and ID-LPE

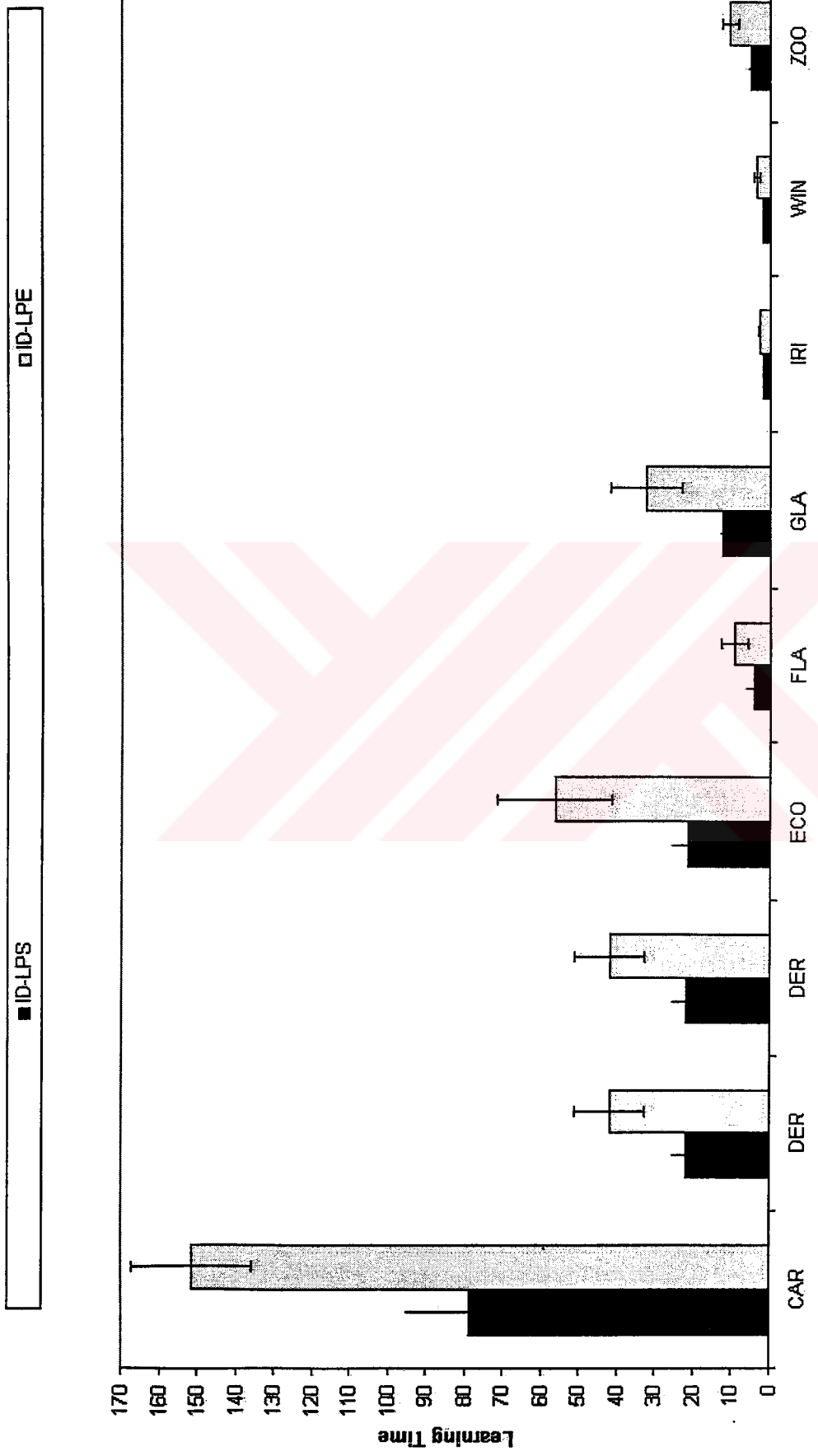

FIGURE 6.3.1.3 Learning time results for ID-LPS and ID-LPE (small data sets)

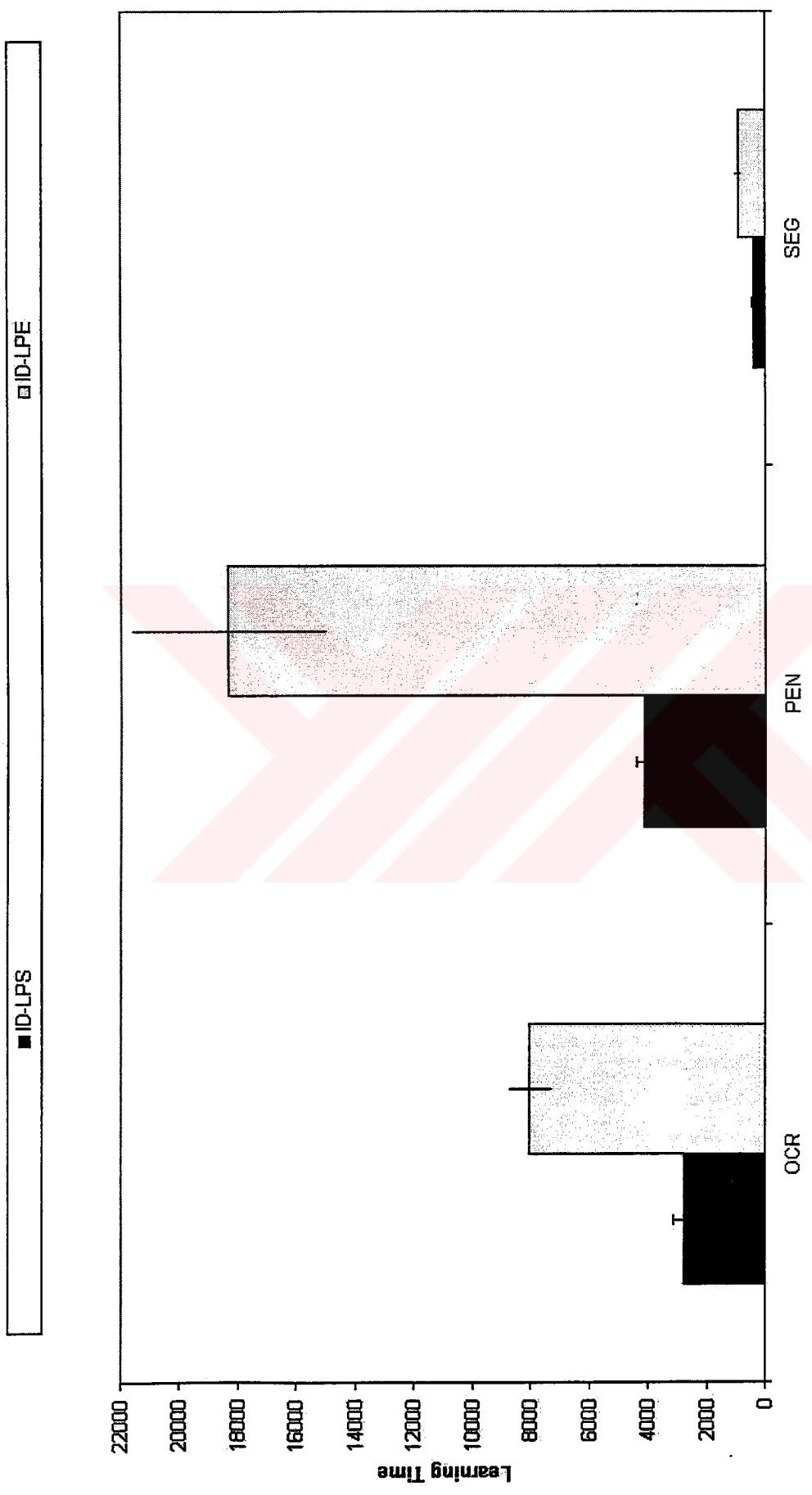

FIGURE 6.3.1.4 Learning time results for ID-LPS and ID-LPE (large data sets)

6.3.2. Comparison of Hybrid Tests in Decision Nodes for Neural Networks

The aim of this section is to find which test measure (F-test or t-test) is best in comparing the performance of hybrid trees. In big data sets as *Mushroom*, *Ocrdigits*, *Pendigits* and *Segment*, training is done with 10 epochs instead of 50 epochs. This is due to the large amount of computation to train the networks with t-test. For example training with t-test of *Ocrdigits* data set takes approximately 4 days, where we have 160 runs like that.

TABLE 6.3.2.1 Accuracy results for hybrid network models

Data set name	ID-Hybrid-F	ID-Hybrid-t	Significance
Breast	96.62±0.55	96.62±0.63	
Bupa	63.42±2.57	63.71±3.24	
Car	**94.51±1.15**	**92.19±1.37**	1>2
Cylinder	71.31±1.74	71.24±1.89	
Dermatology	94.54±4.67	85.74±11.97	
Ecoli	83.10±4.19	81.43±3.75	
Flare	88.11±2.43	87.98±2.28	
Glass	55.05±9.72	60.37±6.60	
Hepatitis	83.74±3.41	83.48±3.38	
Horse	82.66±2.58	82.01±3.28	
Iris	92.67±3.28	92.80±3.34	
Ironosphere	87.80±2.15	87.35±1.79	
Monks	66.39±1.85	66.30±1.77	
Mushroom	99.96±0.03	99.95±0.03	
Ocrdigits	92.79±2.20	N/A	
Pendigits	90.82±9.62	N/A	
Segment	81.77±12.97	85.13±6.33	
Vote	94.71±1.13	94.80±1.06	
Wine	96.07±2.07	95.96±2.32	
Zoo	86.93±5.39	86.74±4.15	

TABLE 6.3.2.2 Node results for hybrid network models

Data set name	ID-Hybrid-F	ID-Hybrid-t	Significance
Breast	3.00±0.00	3.00±0.00	
Bupa	4.40±1.90	4.40±1.65	
Car	7.60±0.97	7.20±1.48	
Cylinder	8.80±1.75	9.00±2.11	
Dermatology	11.20±1.14	11.00±0.00	
Ecoli	10.60±2.27	10.80±2.57	
Flare	3.00±1.33	2.40±0.97	
Glass	11.00±5.50	11.80±2.70	
Hepatitis	3.00±0.00	3.00±0.00	
Horse	5.60±2.84	5.20±1.75	
Iris	5.00±0.00	5.00±0.00	
Ironosphere	4.00±1.05	3.80±1.03	
Monks	3.00±0.00	3.00±0.00	
Mushroom	3.00±0.00	3.00±0.00	
Ocrdigits	25.40±3.75	N/A	
Pendigits	23.40±5.80	N/A	
Segment	14.40±2.84	14.60±3.10	
Vote	4.20±1.93	4.40±1.90	
Wine	5.00±0.00	5.20±0.63	
Zoo	12.40±1.90	12.60±1.26	

The results are shown in Table 6.3.2.1 and Figure 6.3.2.1. Node results are shown in Table 6.3.2.2 and Figure 6.3.2.2. Learning time results are shown in Table 6.3.2.3, Figure 6.3.2.4 and Figure 6.3.2.5.

There is no significant difference in terms of accuracy and node size between the two test selection measures (Only in *Car* in terms of accuracy).

But the difference is in learning time. In all data sets, t-test is slower than F-test with over than %99 level. Because t-test runs with 30 fold cross validation with the whole training set while F-test runs only 10 fold cross validation with half of the training set.

TABLE 6.3.2.3 Learning time results for hybrid network models

Data set name	ID-Hybrid-F	ID-Hybrid-t	Significance
Breast	**30±1**	**175±4**	2>>>1
Bupa	**15±3**	**91±17**	2>>>1
Car	**911±151**	**6422±656**	2>>>1
Cylinder	**366±45**	**2455±346**	2>>>1
Dermatology	**399±25**	**3428±899**	2>>>1
Ecoli	**219±38**	**1697±338**	2>>>1
Flare	**67±26**	**420±134**	2>>1
Glass	**137±28**	**1275±281**	2>>>1
Hepatitis	**10±0**	**66±1**	2>>>1
Horse	**327±89**	**2089±392**	2>>>1
Iris	**14±1**	**95±11**	2>>>1
Ironosphere	**53±9**	**300±59**	2>>>1
Monks	**16±0**	**97±1**	2>>>1
Mushroom*	**5529±414**	**8546±934**	2>>>1
Ocrdigits*	**14791±3204**	**N/A**	2>>>1
Pendigits*	**9942±1566**	**N/A**	2>>>1
Segment*	**4756±453**	**6217±529**	2>>1
Vote	**53±11**	**376±84**	2>>>1
Wine	**19±2**	**125±26**	2>>>1
Zoo	**66±10**	**422±80**	2>>>1

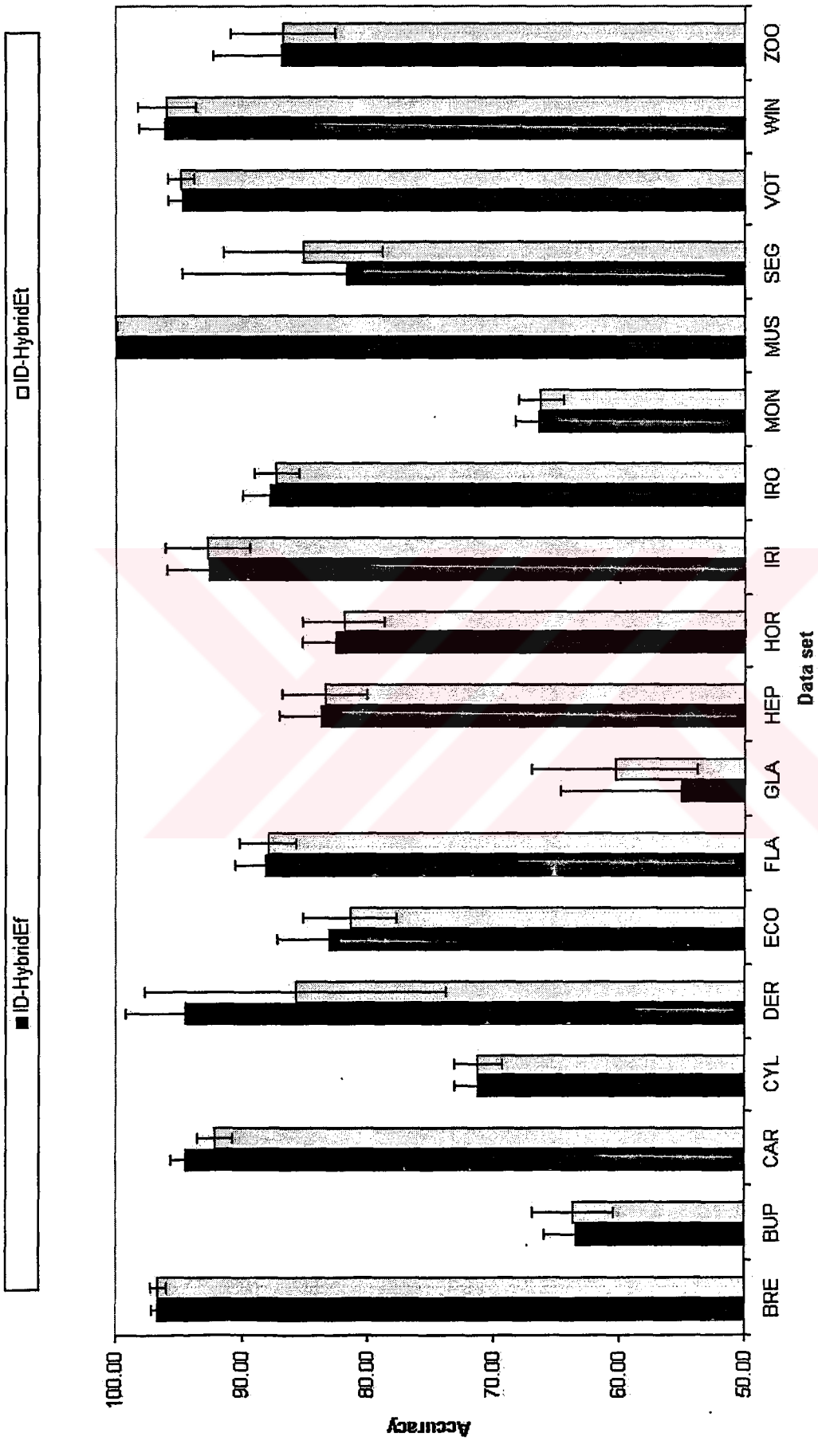

FIGURE 6.3.2.1 Accuracy results for hybrid network models

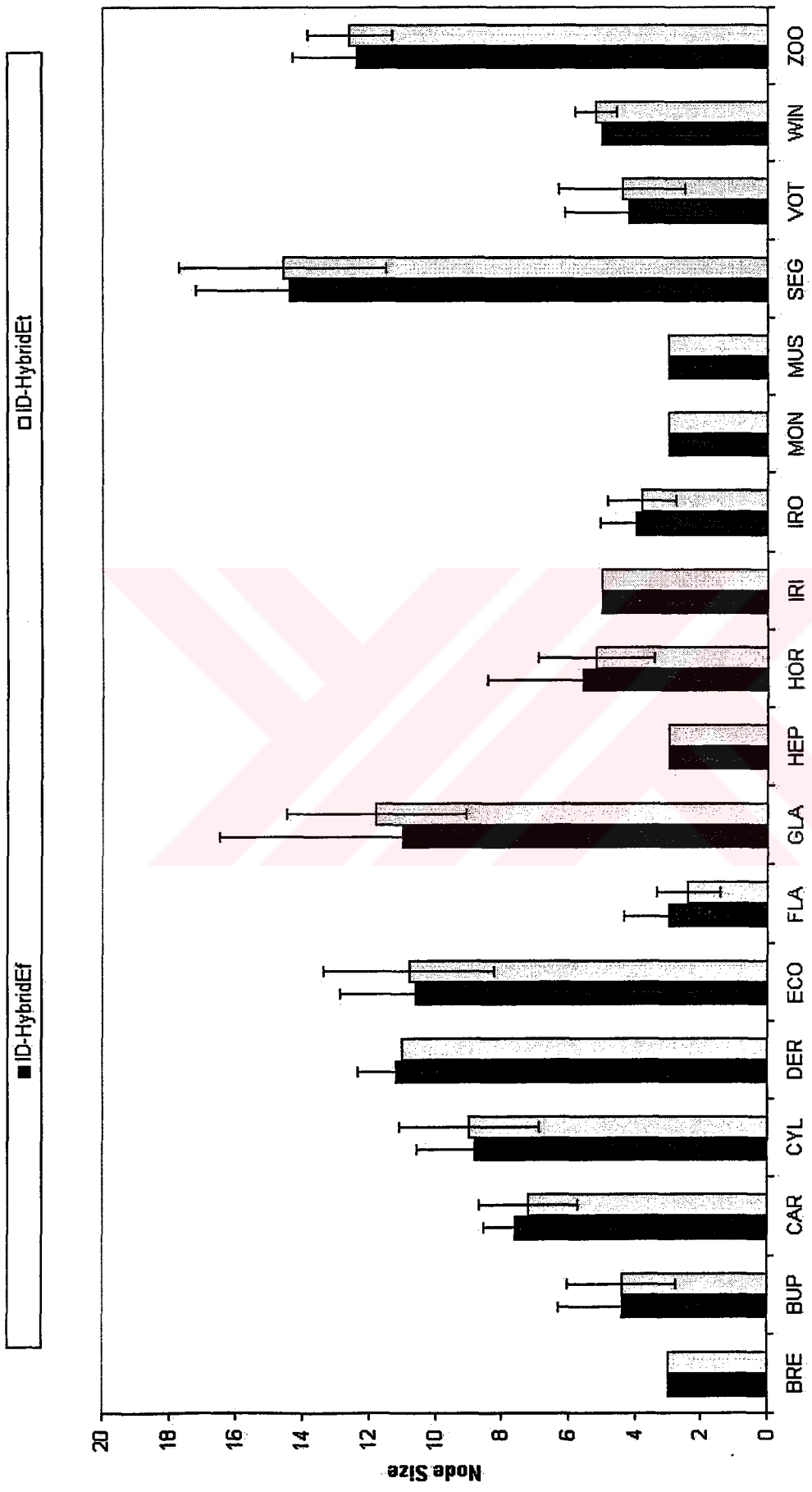

FIGURE 6.3.2.2 Node results for hybrid network models

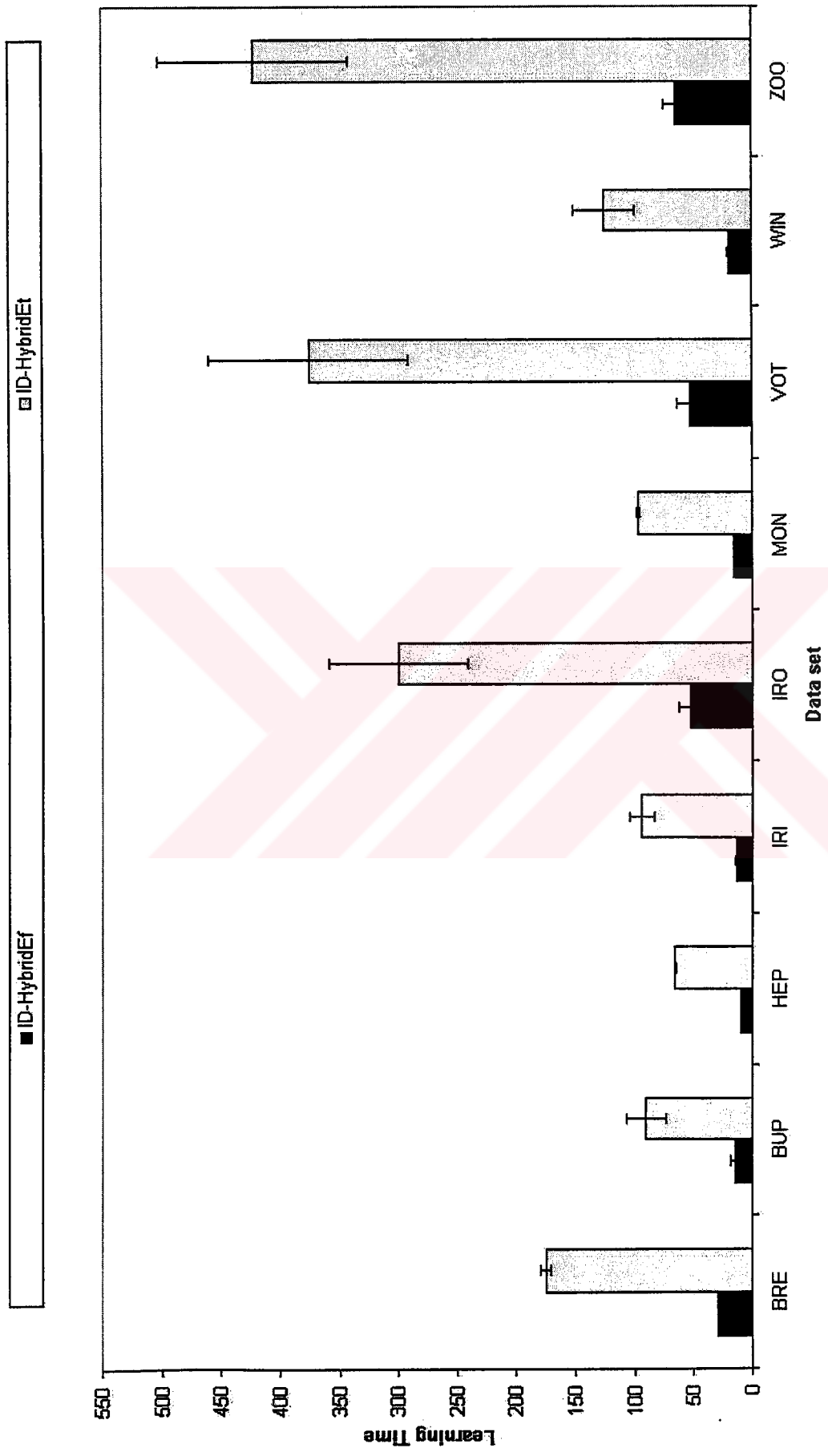

FIGURE 6.3.2.3 Learning time results for hybrid network models (small data sets)

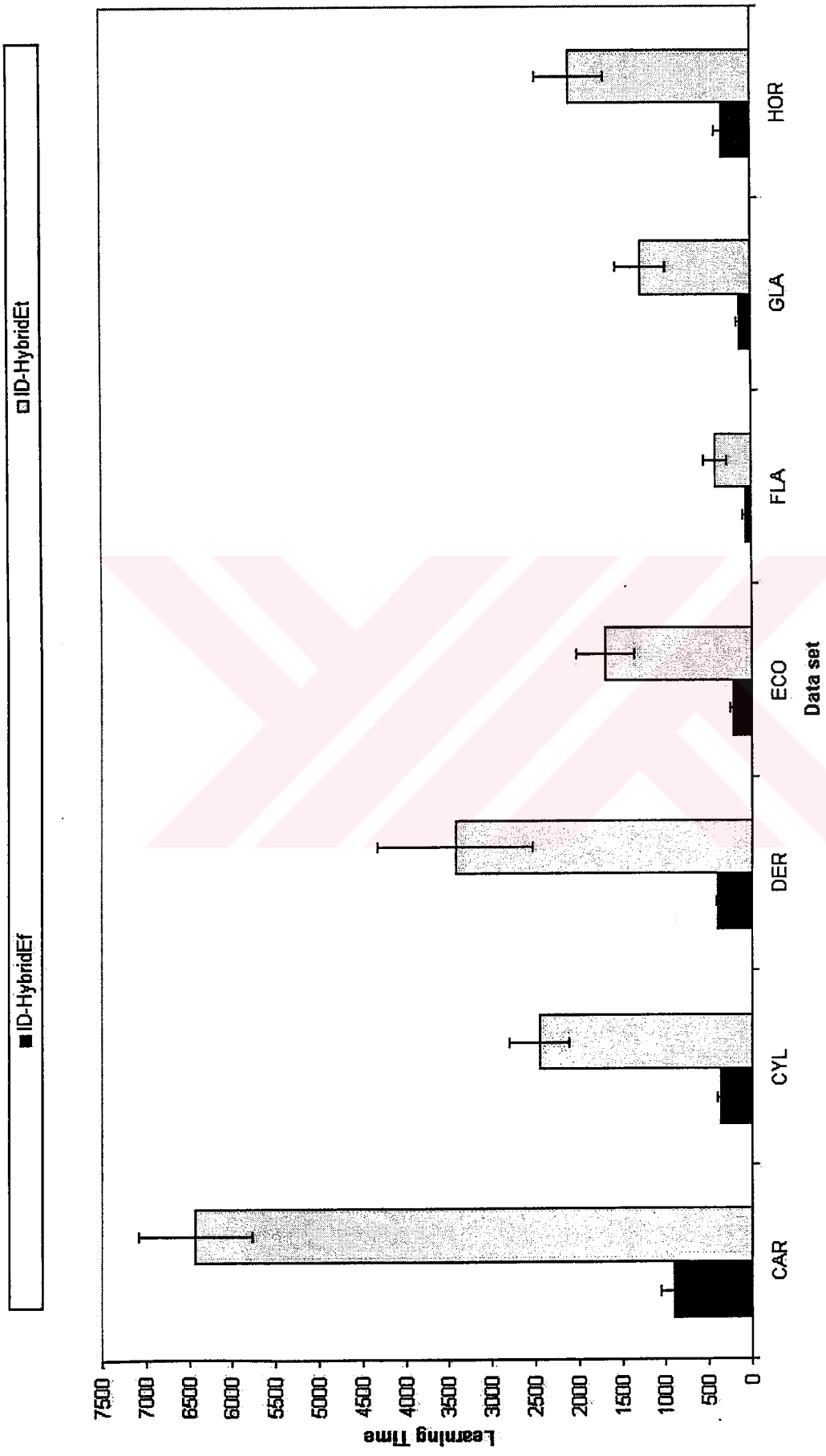

FIGURE 6.3.2.4 Learning time results for hybrid network models (medium size data sets)

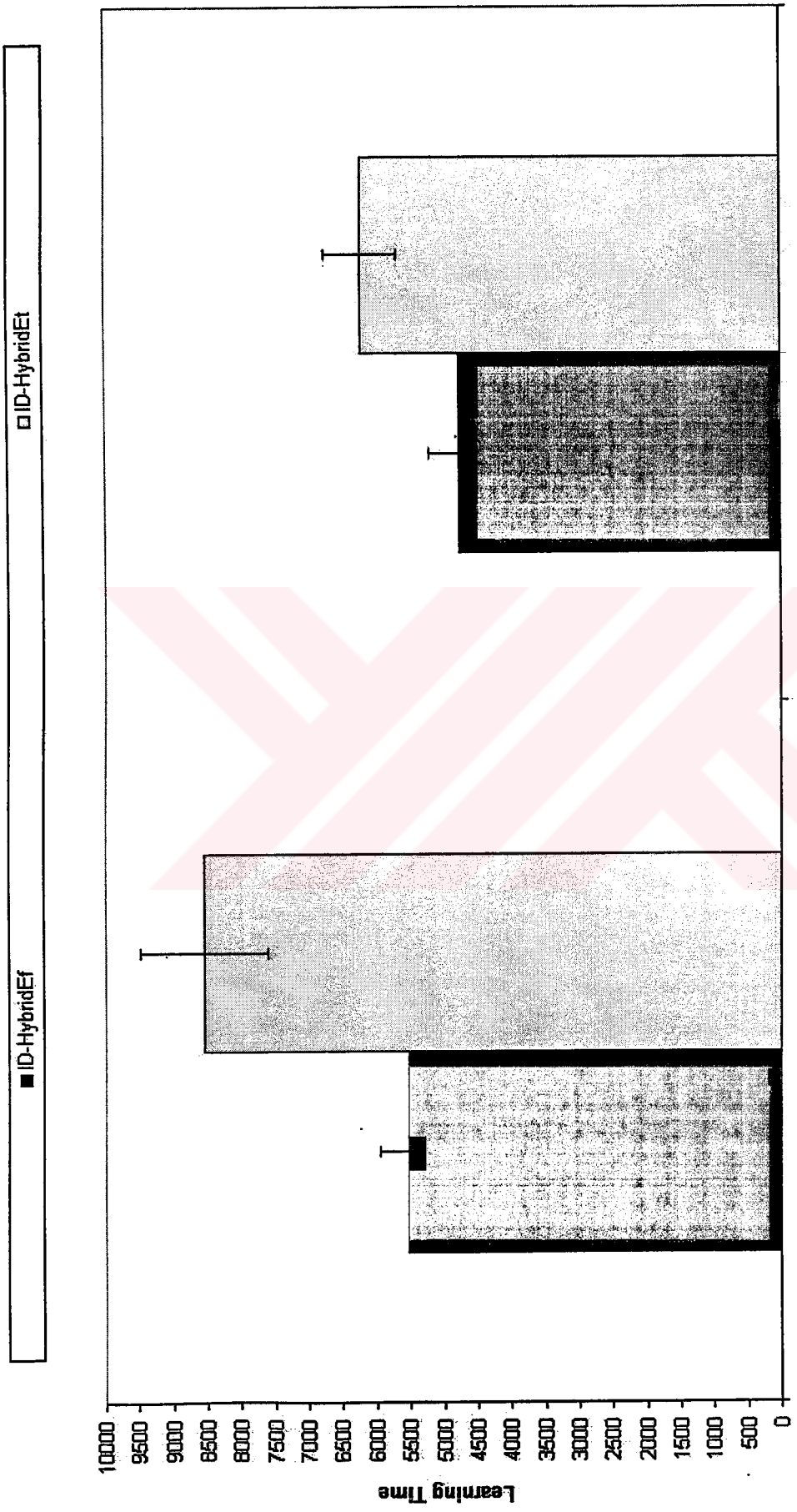

FIGURE 6.3.2.5 Learning time results for hybrid network models (large data sets)

6.3.3. Comparison of the Network Structures in Decision Nodes

We must also determine the type of the neural network to train in a decision tree. So we must find out which type of neural network performs the best. In order to accomplish this task, we have three different types of networks: Linear perceptron, multilayer perceptron and a hybrid of them (with F-test). Multilayer perceptron is a nonlinear method. These three networks are compared according to accuracy, node size and learning time. Accuracy results are shown in Table 6.3.3.1 and Figure 6.3.3.1. Node results are shown in Table 6.3.3.2 and Figure 6.3.3.2. Learning time results are shown in Table 6.3.3.3, Figures 6.3.3.3, 6.3.3.4 and 6.3.3.5.

Linear neural network methods results can be divided into two groups. Data sets having two classes and data sets having more than two classes. If the data set has two classes and if the classes are not linearly separable, the accuracy results can be very low. But if they are linearly separable, the results can be very good as in *Breast* data set. For nonlinear network models, results are higher in such data sets. More generally in three data sets out of 20, the nonlinear model outperforms the linear model and in two data sets, the hybrid model outperforms the linear model. In four data sets; *Ocrdigits*, *Dermatology*, *Zoo* and *Segment*, the nonlinear model has good results but does not converge all the time. So these data sets have larger variance.

If we look at the node results, the nonlinear model is better than linear model in two data sets and it is better than the hybrid model in two data sets. Any data set having c classes must have at least $2c-1$ nodes so that each class can be in one leaf. The nonlinear model converges to the optimum solution in the number of nodes. There is an order between the node size of the models as Linear > Hybrid > Nonlinear model. In some data sets we see that the hybrid model performs worse than the other two in terms of node size. This mainly depends on the deviation of the results. The nonlinear model outperforms the hybrid model in four data sets and the linear model in two data sets. The hybrid and linear models outperform each other in only one data set.

In terms of the time consumed for learning, linear model performs the best as we have expected. If we compare times we see an ordering as Hybrid > Nonlinear > Linear. But sometimes, the linear model has larger training time than the nonlinear model, which is due to the large number of nodes in the tree with the linear model and the large number of instances of that data set.

TABLE 6.3.3.1 Accuracy results for different network models

Data set name	ID-LPE	ID-MLPE	ID-Hybrid-F	Significance
Breast	96.60±0.61	96.77±0.91	96.62±0.55	
Bupa	63.53±2.76	63.24±4.31	63.42±2.57	
Car	89.48±4.01	96.86±2.30	94.51±1.15	
Cylinder	70.21±4.48	70.35±9.56	71.31±1.74	
Dermatology	85.74±7.06	87.81±13.59	94.54±4.67	
Ecoli	82.62±4.06	80.12±5.12	83.10±4.19	
Flare	88.36±2.37	87.67±2.56	88.11±2.43	
Glass	54.95±7.83	58.04±13.30	55.05±9.72	
Hepatitis	84.13±2.86	83.74±2.43	83.74±3.41	
Horse	82.07±3.48	84.67±2.64	82.66±2.58	
Iris	**77.60±15.70**	**92.67±3.57**	**92.67±3.28**	2>1,3>1
Ironosphere	87.80±2.18	87.52±2.11	87.80±2.15	
Monks	66.34±1.87	66.99±2.17	66.39±1.85	
Mushroom	**99.95±0.03**	**99.99±0.02**	99.96±0.03	2>1
Ocrdigits	93.87±0.92	83.90±10.22	92.79±2.20	
Pendigits	91.94±4.16	91.35±6.55	90.82±9.62	
Segment	79.76±11.58	80.35±12.36	81.77±12.97	
Vote	94.71±1.05	95.58±1.72	94.71±1.13	
Wine	**87.75±12.62**	**95.96±2.13**	**96.07±2.07**	2>1,3>1
Zoo	79.38±8.10	85.33±11.86	86.93±5.39	

TABLE 6.3.3.2 Node results for different network models

Data set name	ID-LPE	ID-MLPE	ID-HybridEf	Significance
Breast	3.00±0.00	3.00±0.00	3.00±0.00	
Bupa	4.60±1.84	3.80±1.03	4.40±1.90	
Car	7.40±0.84	6.60±1.26	7.60±0.97	
Cylinder	**8.40±1.90**	**6.40±1.65**	**8.80±1.75**	3>>2
Dermatology	**8.80±1.48**	9.60±2.32	**11.20±1.14**	3>1
Ecoli	10.80±2.90	8.80±2.20	10.60±2.27	
Flare	3.20±2.20	2.20±1.03	3.00±1.33	
Glass	10.20±4.64	7.20±3.71	11.00±5.50	
Hepatitis	3.00±0.00	3.00±0.00	3.00±0.00	
Horse	5.00±1.63	4.00±1.41	5.60±2.84	
Iris	**4.00±1.05**	**5.00±0.00**	**5.00±0.00**	2>>1,3>>1
Ironosphere	3.80±1.03	3.80±1.03	4.00±1.05	
Monks	3.00±0.00	3.00±0.00	3.00±0.00	
Mushroom	3.00±0.00	3.00±0.00	3.00±0.00	
Ocrdigits	**34.80±4.94**	**18.40±3.53**	**25.40±3.75**	1>3>>2
Pendigits	**30.40±6.40**	**17.60±1.35**	**23.40±5.80**	1>>>2,3>>2
Segment	16.60±6.65	11.60±2.12	14.40±2.84	
Vote	4.20±1.93	3.00±0.00	4.20±1.93	
Wine	4.40±0.97	5.00±0.00	5.00±0.00	
Zoo	**8.80±1.75**	10.60±2.80	**12.40±1.90**	3>1

TABLE 6.3.3.3 Learning time results for different network models

Data set name	ID-LPE	ID-MLPE	ID-HybridEf	Significance
Breast	**5±0**	**7±0**	**30±1**	3>>>1,3>>>2
Bupa	**3±1**	**3±1**	**15±3**	3>>>2>>>1
Car	**152±16**	**216±18**	**911±151**	3>>>2>>>1
Cylinder	**19±2**	**102±15**	**366±45**	3>>>2>>>1
Dermatology	**42±9**	**122±19**	**399±25**	3>>>2>>>1
Ecoli	**57±15**	**50±13**	**219±38**	3>>>1,3>>>2
Flare	**9±4**	**18±7**	**67±26**	3>>2>>1
Glass	**33±9**	**27±8**	**137±28**	3>>>1,3>>>2
Hepatitis	**1±0**	**3±0**	**10±0**	3>>>1,3>>>2
Horse	**14±2**	**97±20**	**327±89**	3>2>>>1
Iris	**3±0**	**3±0**	**14±1**	3>>>1,3>>>2
Ironosphere	**4±1**	**14±2**	**53±9**	3>>>2>>>1
Monks	**3±0**	**3±0**	**16±0**	3>>>1,3>>>2
Mushroom	**628±204**	**1858±270**	**5529±414**	3>>>2>>>1
Ocrdigits*	**8035±757**	**10993±1402**	**14791±3204**	3>>>2>1
Pendigits*	**18340±3319**	**8473±1742**	**9942±1566**	3>>>1>>2
Segment	**937±103**	**927±126**	**4756±453**	3>>>1,3>>>2
Vote	**6±1**	**13±2**	**53±11**	3>>2>>1
Wine	**4±1**	**4±1**	**19±2**	3>>>1,3>>>2
Zoo	**10±2**	**18±4**	**66±10**	3>>>2>>>1

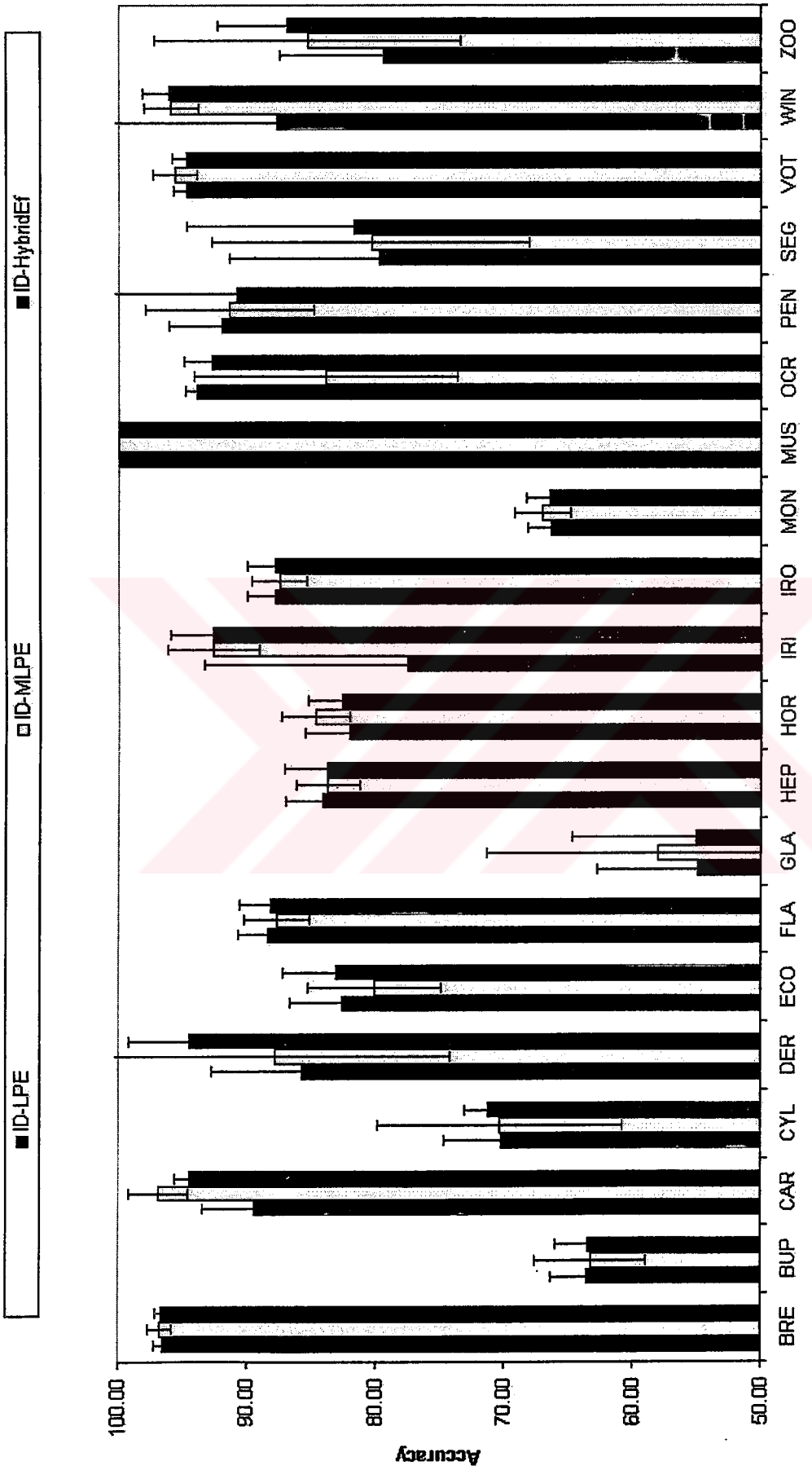

FIGURE 6.3.3.1 Accuracy results for network models

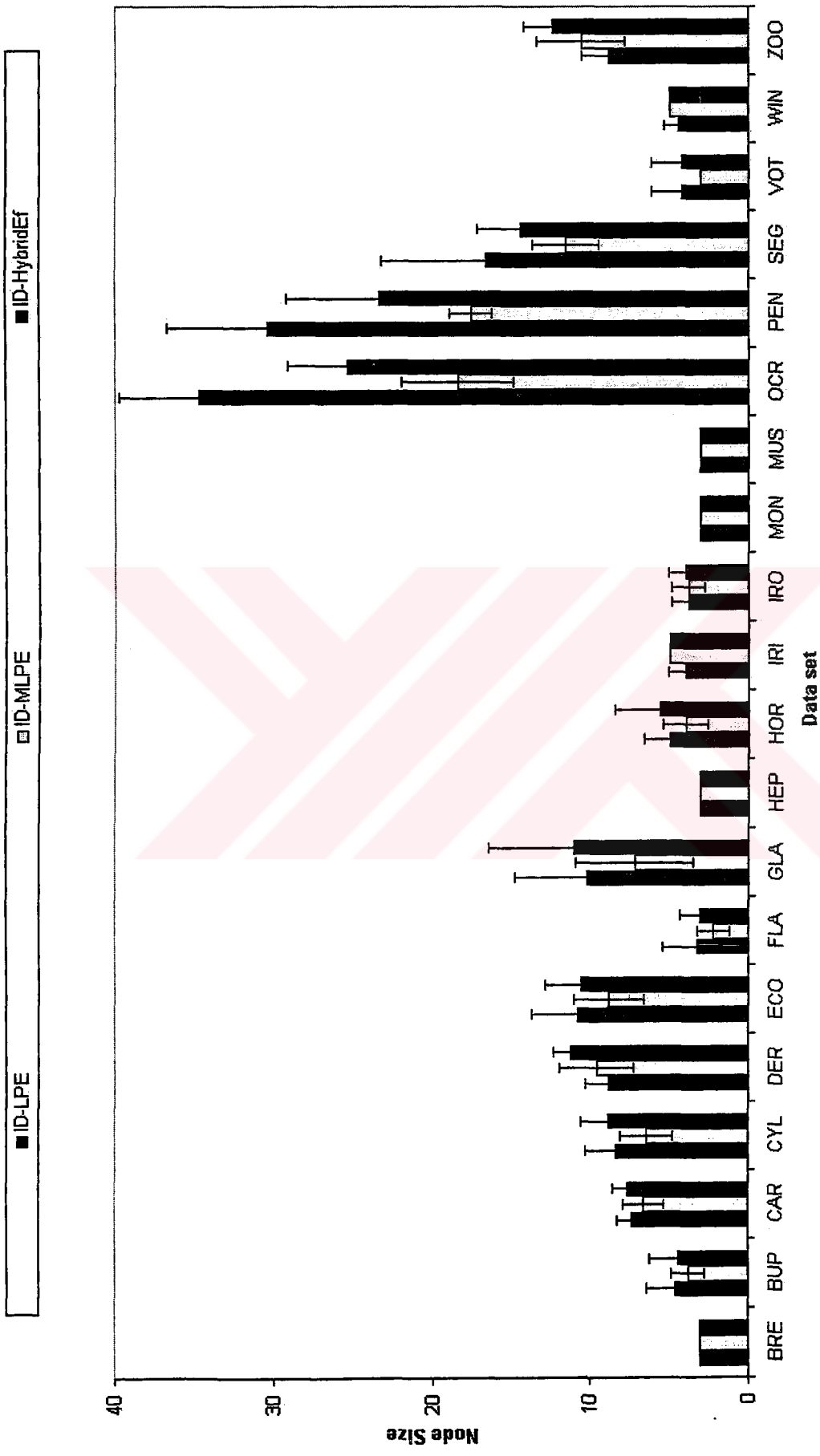

FIGURE 6.3.3.2 Node results for network models

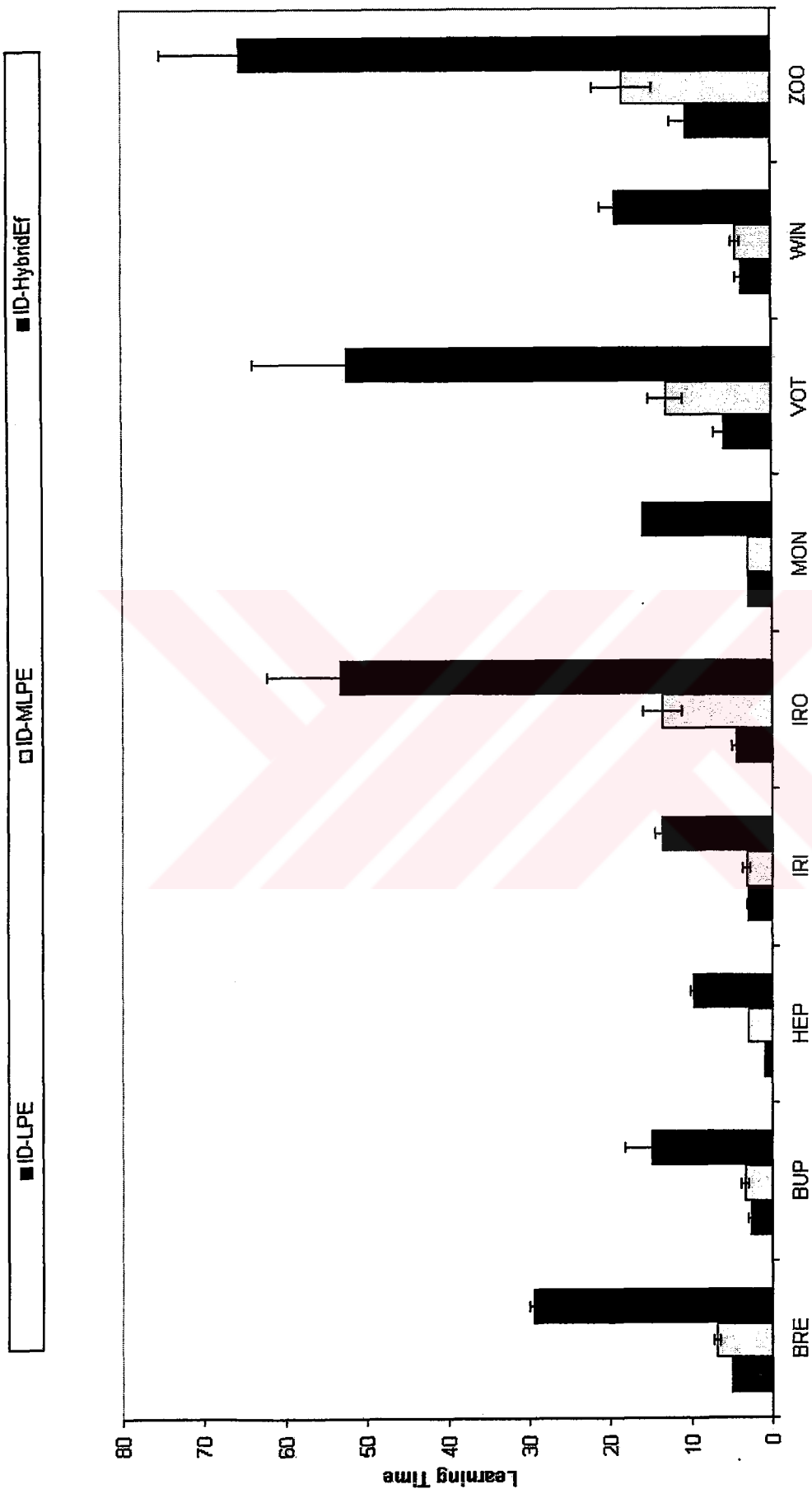

FIGURE 6.3.3.3 Learning time results for network models (small data sets)

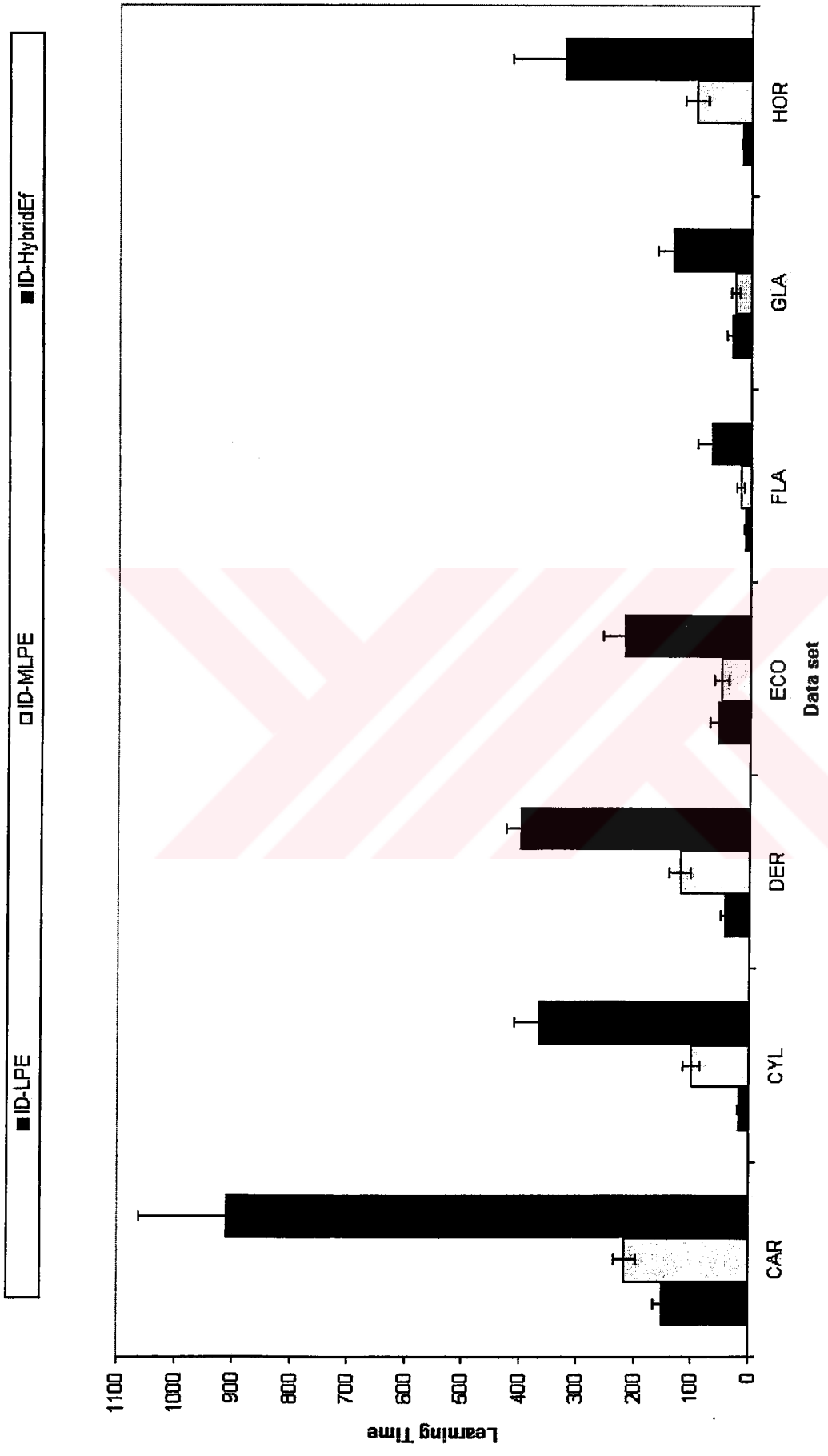

FIGURE 6.3.3.4 Learning time results for network models (medium size data sets)

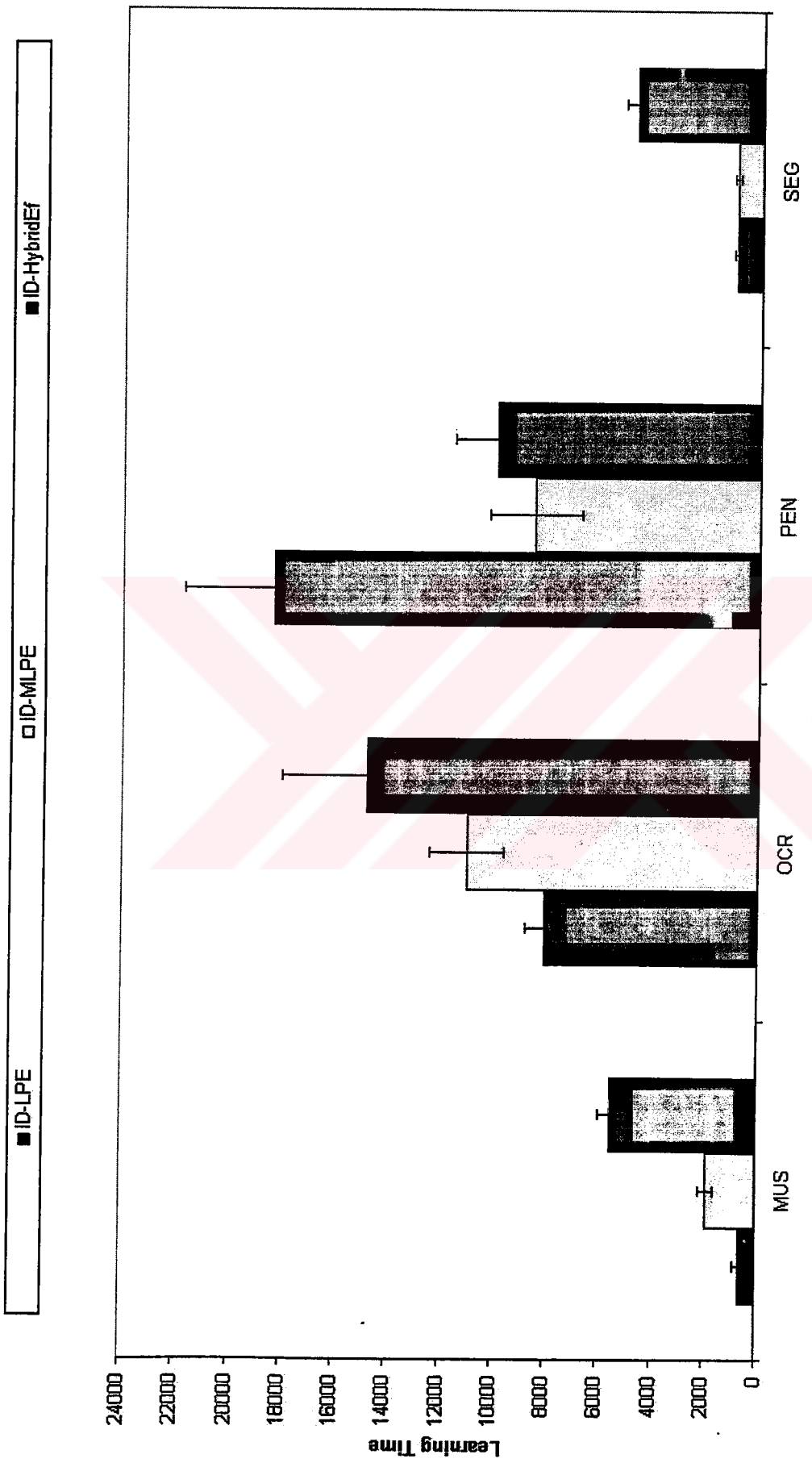

FIGURE 6.3.3.5 Learning time results for network models (large data sets)

6.4. Results for LDA

For the rest of these results, the definition given in Table 6.4.1 applies.

TABLE 6.4.1 Definition of neural-network based methods

Name	Class Separation	Pruning	PCA	PCA percentage
ID-LDA	Exchange	Pre-pruning	Always	%90
ID-LDA-R	Exchange	Pre-pruning	If Required	%90
ID-LDA-R99	Exchange	Pre-pruning	If Required	%99

6.4.1. Effects of PCA on the Results

Previously we saw that PCA must be used to solve the singular covariance matrix problem in Chapter 5. But there are also data sets where we do not need PCA in some nodes because the covariance matrix is invertible in those nodes. Hence, we took those data sets performances two times, one time we used always PCA and the other time we used PCA when it is required. In this section we will compare these two results and want to find out if PCA decrements the performance because of the %10 loss in variance. The results are shown in Table 6.4.1.1 and Figure 6.4.1.1 for accuracy, in Table 6.4.1.2 and Figure 6.4.1.2 for tree sizes and in Table 6.4.1.3 and Figure 6.4.1.3 for learning time. Some of the data sets are shown with an asterisk near them. In those data sets, PCA is never required.

If we look at the accuracy results we see that PCA causes a decrease in performance. In three data sets out of five, accuracy is significantly dropped when PCA is applied. In these data sets, PCA is never required. In other data sets where PCA is applied, accuracy does not change significantly.

TABLE 6.4.1.1 Accuracy results for ID-LDA and ID-LDA-R

Data set name	ID-LDA	ID-LDA-R	Significance
Breast*	96.65±0.66	95.85±0.72	
Bupa*	**57.28±3.23**	**67.42±2.97**	2>>>1
Ecoli	83.10±2.50	83.69±3.58	
Glass	57.85±3.67	55.51±4.43	
Iris*	**82.67±5.52**	**97.20±1.47**	2>>>1
Monks*	**66.34±1.93**	**74.31±2.26**	2>>1
Wine*	94.04±3.18	96.07±2.66	
Zoo	80.79±6.97	82.56±5.62	

TABLE 6.4.1.2 Node results for ID-LDA and ID-LDA-R

Data set name	ID-LDA	ID-LDA-R	Significance
Breast*	8.00±1.05	7.20±0.63	
Bupa*	6.40±4.90	8.20±1.93	
Ecoli	20.20±4.34	17.60±4.81	
Glass	20.60±5.64	25.60±3.78	
Iris*	**8.00±2.16**	**5.40±0.84**	1>>>2
Monks*	**3.00±0.00**	**7.20±2.39**	2>1
Wine*	7.40±1.84	5.40±0.84	
Zoo	12.60±2.07	12.60±2.07	

For the node results, ID-LDA-R is better than ID-LDA in one data set, whereas ID-LDA is better than ID-LDA-R in one data set. ID-LDA is better than ID-LDA-R in *Monks* data set, where it can not find a split after one split. So it has lower node size. These results also effect learning time. On the *Iris* data set where the tree size is significantly smaller with ID-LDA-R, learning time is also significantly less.

When PCA is applied, the number of reduced dimensions is usually decreased from the root node to a leaf node. For example, while in the root node we need 14 eigenvectors to define the data on *Ecoli* data set, we only need 5 eigenvectors to define data in a leaf node.

TABLE 6.4.1.3 Learning time results for ID-LDA and ID-LDA-R

Data set name	ID-LDA	ID-LDA-R	Significance
Breast*	3±1	2±0	
Bupa*	1±1	1±0	
Ecoli	6±2	6±2	
Glass	5±1	7±1	
Iris*	**1±1**	**0±0**	1>>2
Monks*	1±1	1±0	
Wine*	1±0	1±0	
Zoo	2±0	2±0	

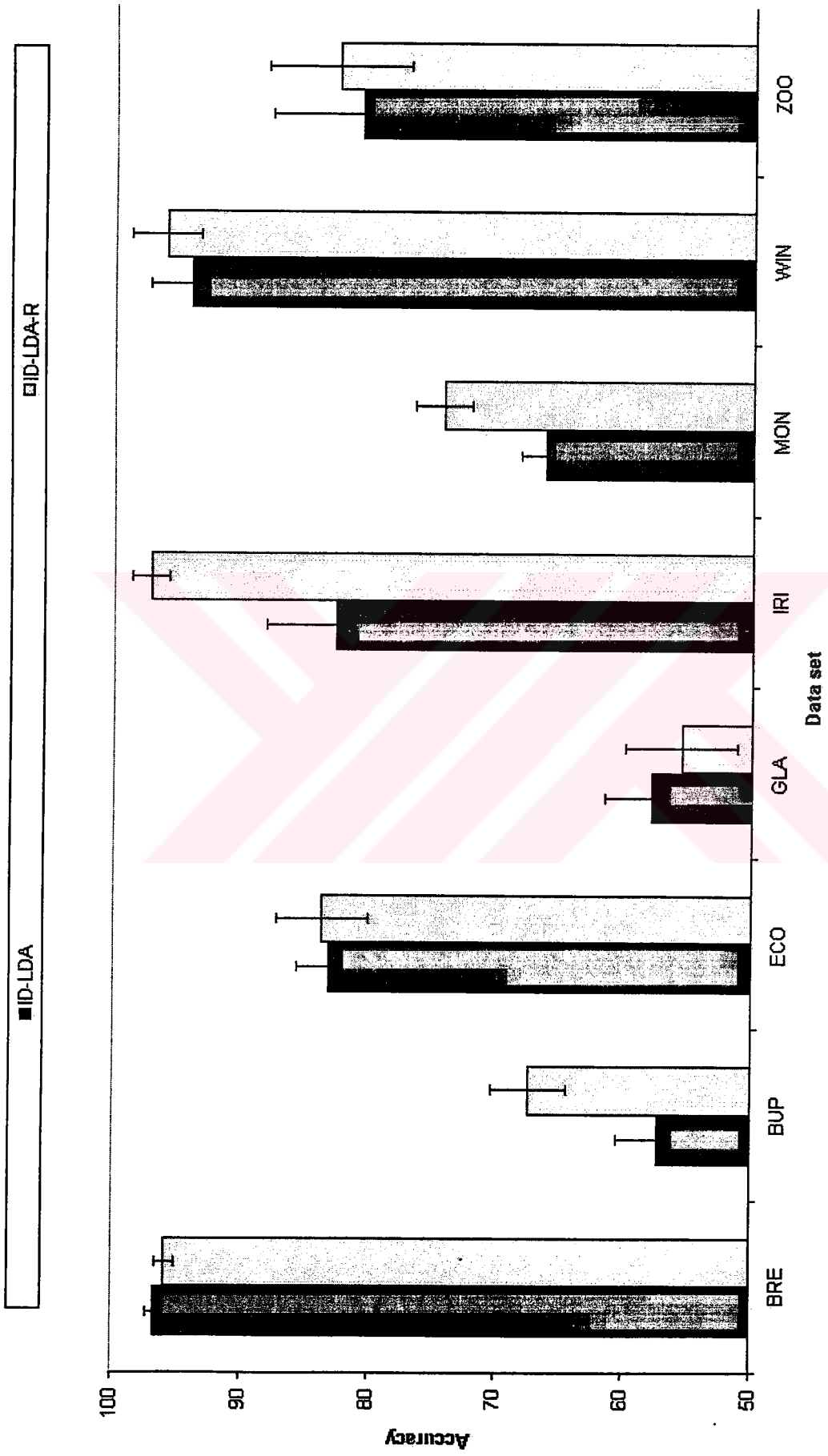

FIGURE 6.4.1.1 Accuracy results for ID-LDA and ID-LDA-R

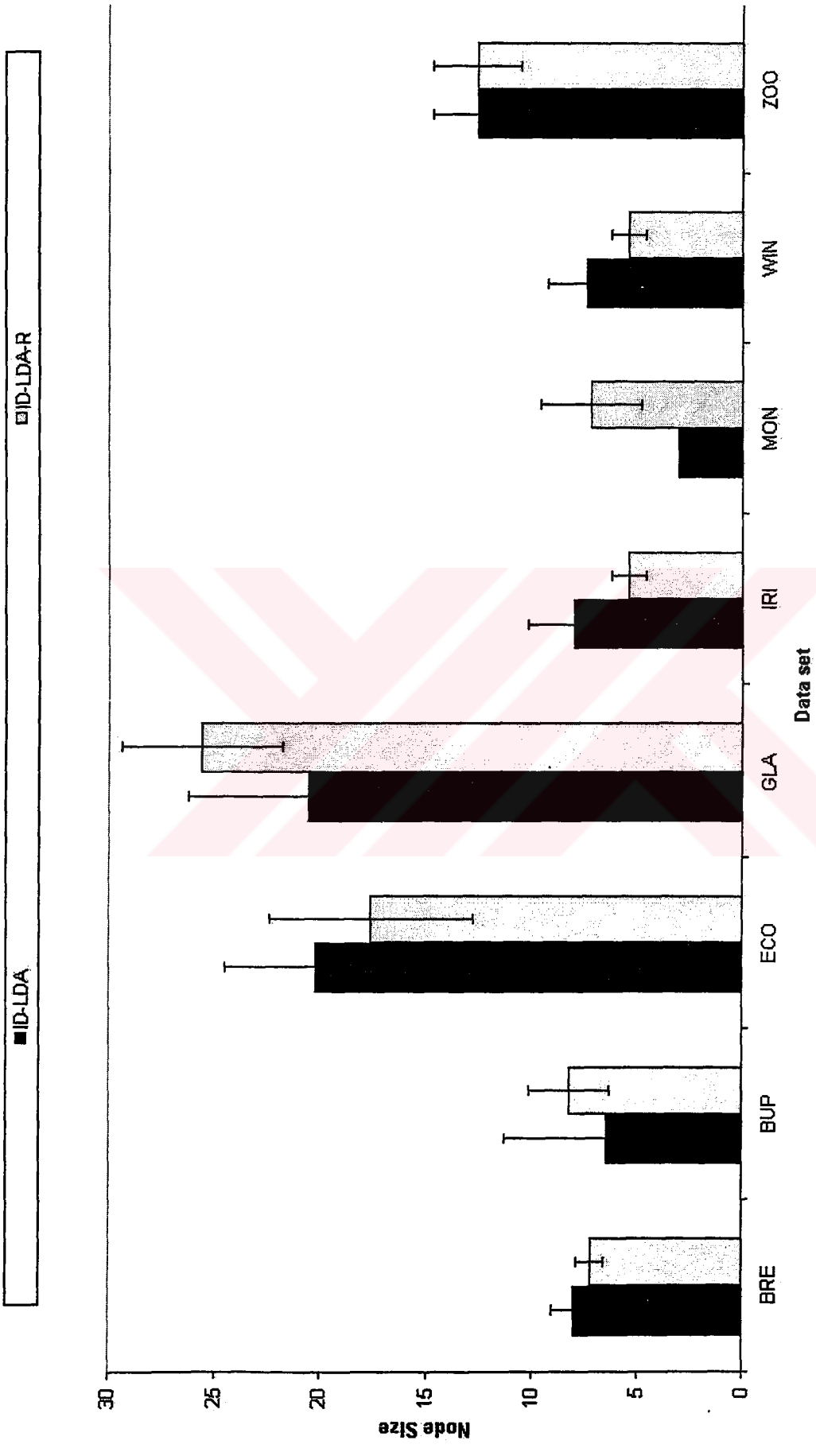

FIGURE 6.4.1.2 Node results for ID-LDA and ID-LDA-R

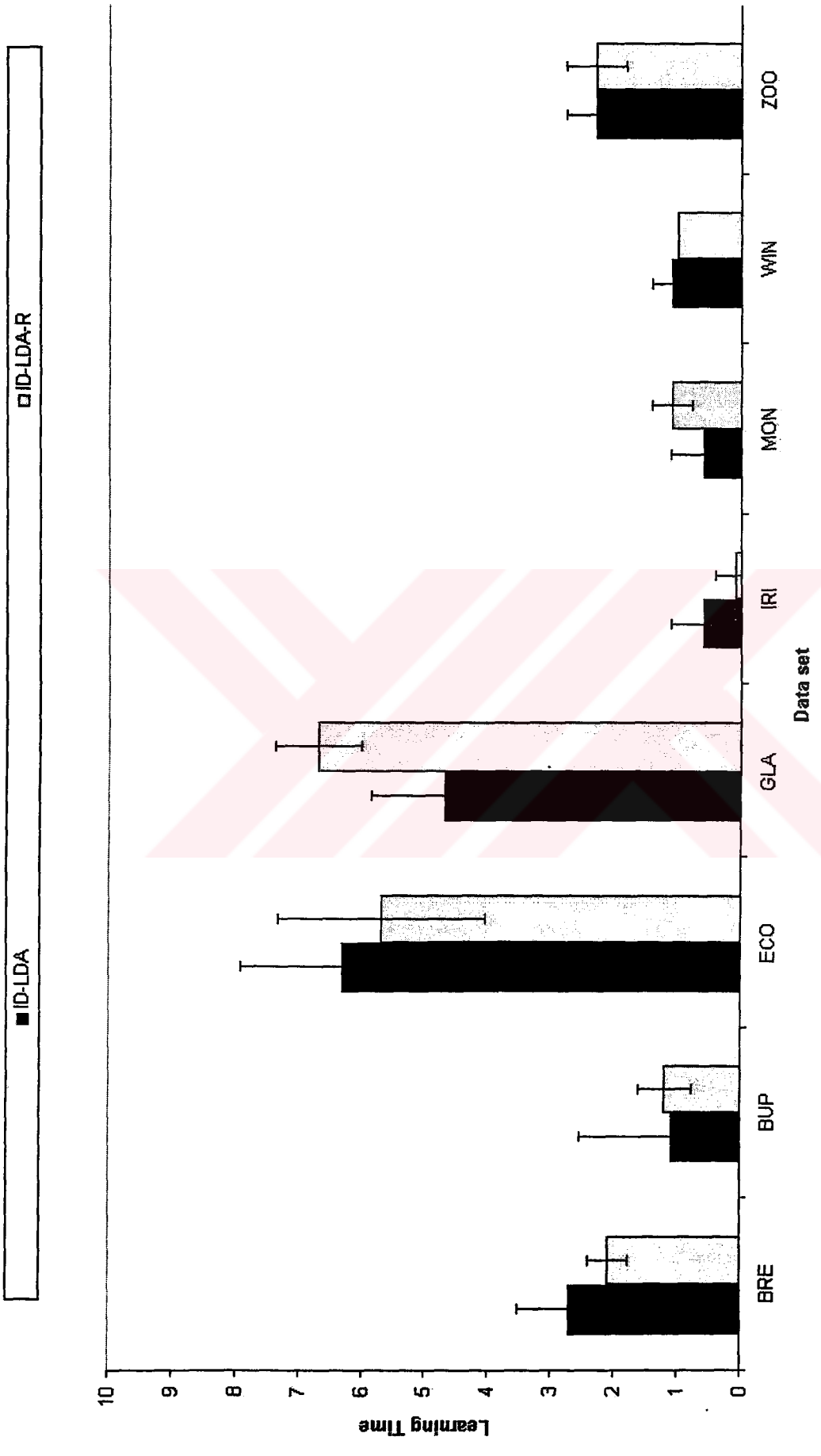

FIGURE 6.4.1.3 Learning time results for ID-LDA and ID-LDA-R

6.4.2. Effects of PCA Percentage on the Results

As Section 6.4.1 shows, LDA performance is decreased when PCA is applied because of the 10% loss ($\varepsilon = 0.90$). We have also made experiments with another percentage levels; with %99 ($\varepsilon = 0.99$), and compared the results of two. The results are shown in Table 6.4.2.1 and Figure 6.4.2.1 for accuracy, in Table 6.4.2.2 and Figure 6.4.2.2 for tree size and in Table 6.4.2.3, Figure 6.4.2.3 and Figure 6.4.2.4 for learning time.

TABLE 6.4.2.1 Accuracy results for ID-LDA-R and ID-LDA-R99

Data set name	IDA-LDA-R	ID-LDA-R99	Significance
Car	**70.02±1.75**	**92.09±1.07**	2>>>1
Cylinder	67.39±2.39	69.80±3.01	
Dermatology	94.75±1.91	96.17±1.59	
Ecoli	83.69±3.58	83.75±2.53	
Flare	88.05±2.39	88.17±2.83	
Glass	55.51±4.43	57.29±4.16	
Hepatitis	83.61±2.12	82.06±5.60	
Horse	**72.39±2.62**	**81.09±1.64**	2>>1
Ironosphere	86.38±2.68	91.11±2.22	
Mushroom	**94.15±0.83**	**98.25±0.57**	2>>>1
Ocrdigits	**89.19±0.92**	**94.59±0.49**	2>>>1
Pendigits	**91.99±0.94**	**95.52±0.44**	2>>>1
Segment	**82.19±2.35**	**90.31±1.20**	2>>>1
Vote	**90.85±2.35**	**94.85±2.17**	2>1
Zoo	82.56±5.62	81.41±7.25	

TABLE 6.4.2.2 Node results for ID-LDA-R and ID-LDA-R99

Data set name	IDA-LDA-R	ID-LDA-R99	Significance
Car	**1.00±0.00**	**12.00±2.54**	2>>>1
Cylinder	10.80±3.33	16.40±8.95	
Dermatology	17.00±2.11	12.80±1.48	
Ecoli	17.60±4.81	20.00±2.71	
Flare	5.20±3.71	5.60±3.13	
Glass	25.60±3.78	26.20±4.44	
Hepatitis	4.60±3.10	8.60±3.10	
Horse	10.20±3.29	16.80±4.05	
Ironosphere	5.40±1.84	11.60±2.67	
Mushroom	17.40±3.63	19.20±4.85	
Ocrdigits	**87.40±8.37**	**59.40±2.07**	1>>2
Pendigits	80.80±3.71	89.00±6.25	
Segment	40.40±5.66	39.80±11.08	
Vote	9.20±3.05	9.80±2.53	
Zoo	12.60±2.07	11.80±1.93	

When we look at the accuracy results, we see that there is a dramatic increase in accuracy while going from ID-LDA-R to ID-LDA-R99. In seven data sets out of 20, there is a significant increase in accuracy that is especially noticeable on large data sets.

On the *Car* data set, for which ID-LDA-R can not find any split, ID-LDA-R99 gives a performance of 92 percent. Therefore its node size in ID-LDA-R99 is significantly more than in ID-LDA-R. On *Ocrdigits* the effect is the opposite, that is, while going from ID-LDA-R to ID-LDA-R99, the accuracy increases and the node size decreases significantly. These results have also an effect on learning time. ID-LDA-R has significantly lower learning time on *Car* because of no split.

TABLE 6.4.2.3 Learning time results for ID-LDA-R and ID-LDA-R99

Data set name	IDA-LDA-R	ID-LDA-R99	Significance
Car	**6±2**	**28±6**	2>>1
Cylinder	34±14	74±57	
Dermatology	18±2	16±1	
Ecoli	6±2	6±1	
Flare	3±2	4±3	
Glass	7±1	7±1	
Hepatitis	1±1	2±1	
Horse	38±16	94±40	
Ironosphere	3±1	11±4	
Mushroom	1846±776	2533±1533	
Ocrdigits	**3189±270**	**2288±108**	1>>2
Pendigits	**996±86**	**1211±91**	2>1
Segment	154±15	148±40	
Vote	8±4	9±3	
Zoo	2±0	2±0	

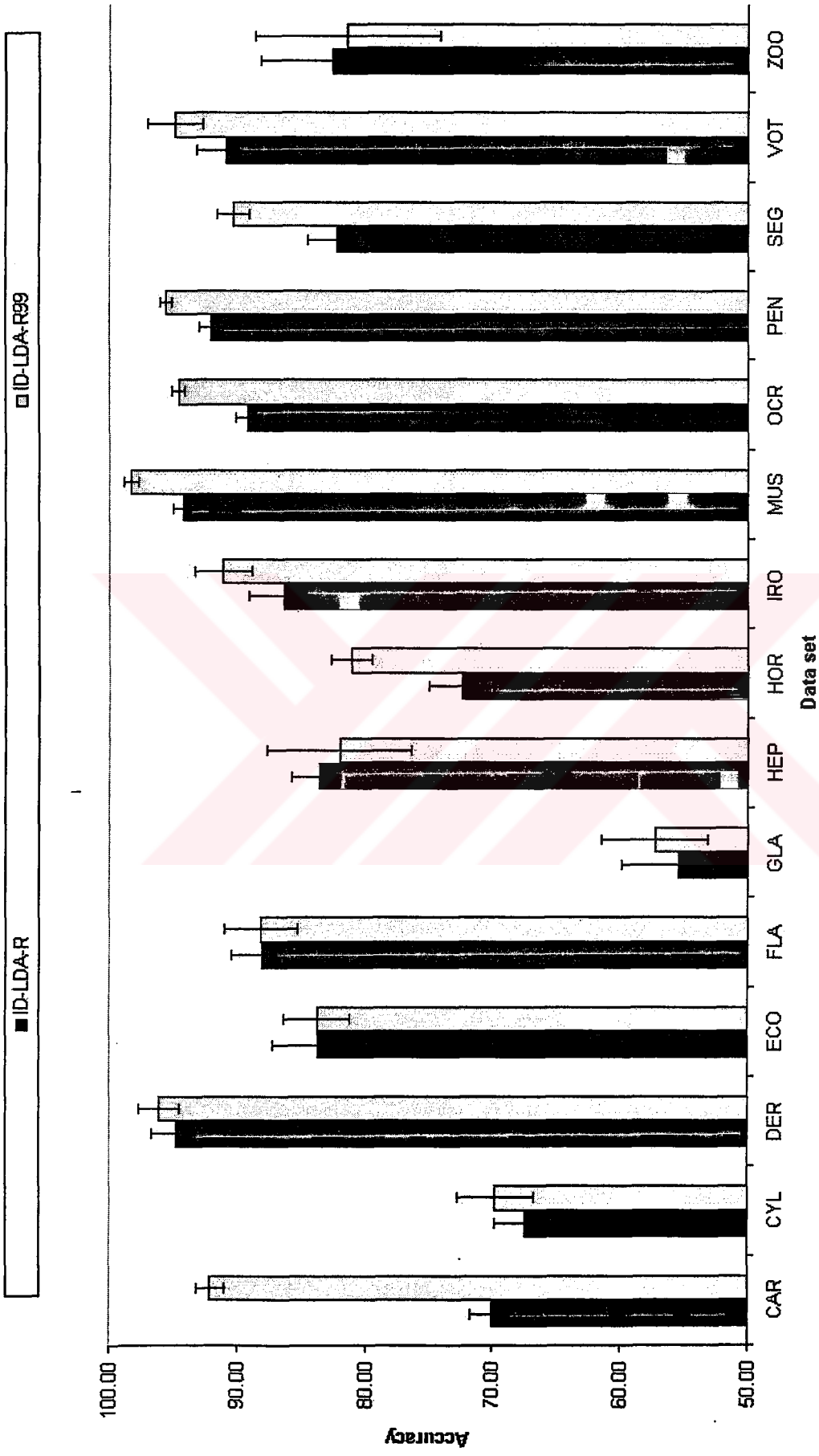

FIGURE 6.4.2.1 Accuracy results for ID-LDA-R and ID-LDA-R99

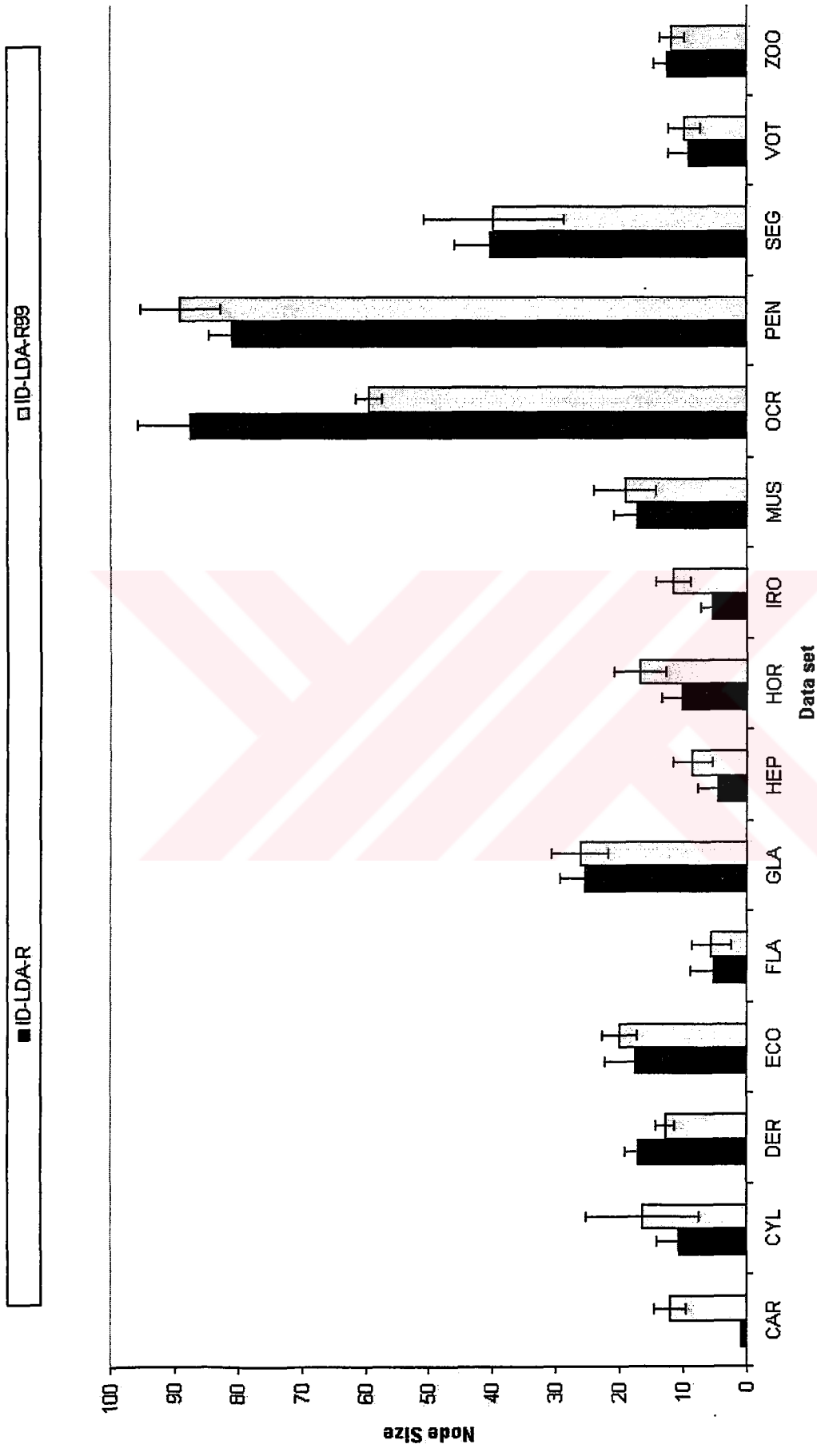

FIGURE 6.4.2.2 Node results for ID-LDA-R and ID-LDA-R99

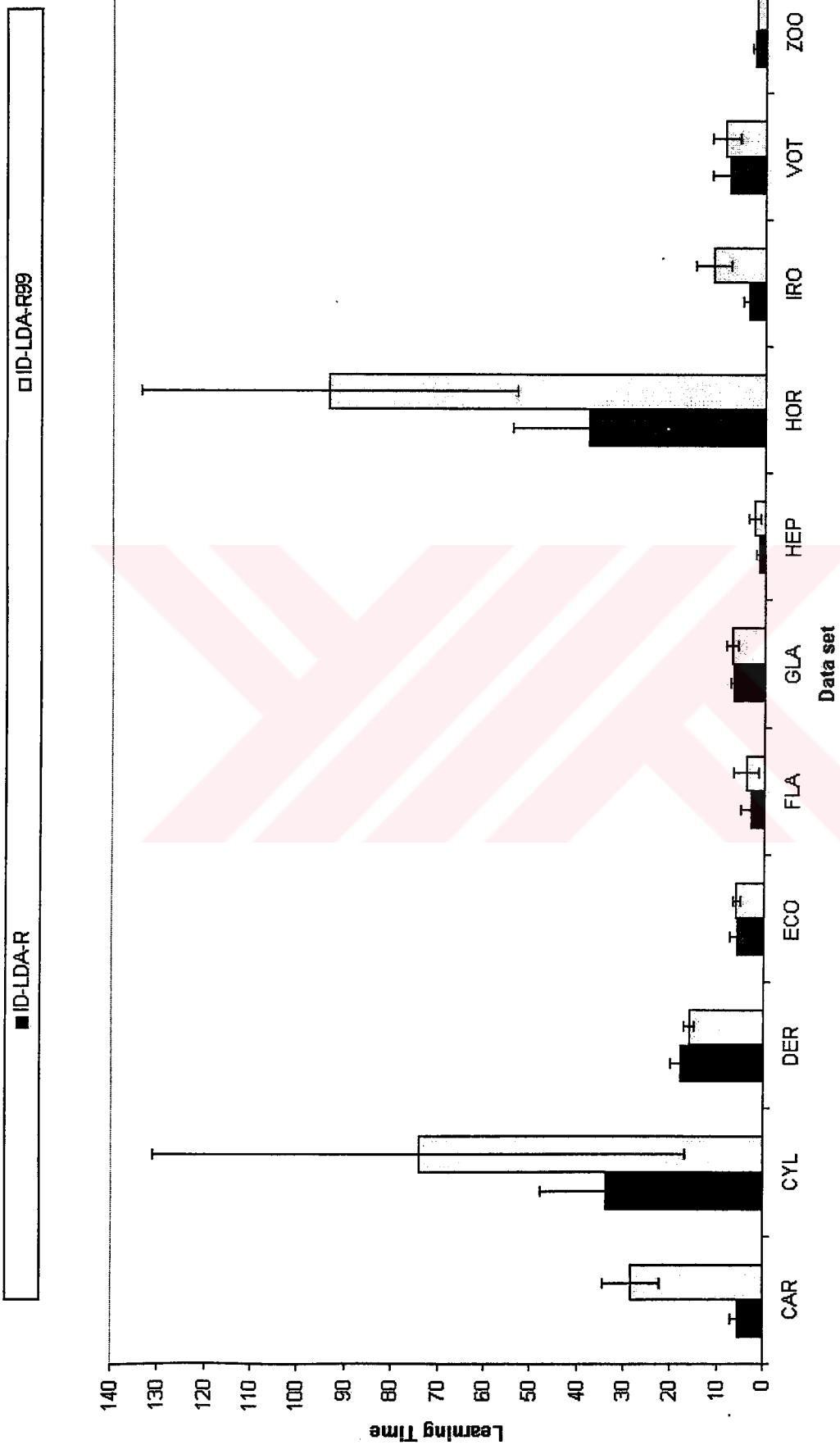

FIGURE 6.4.2.3 Learning time results for ID-LDA-R and ID-LDA-R99 (small data sets)

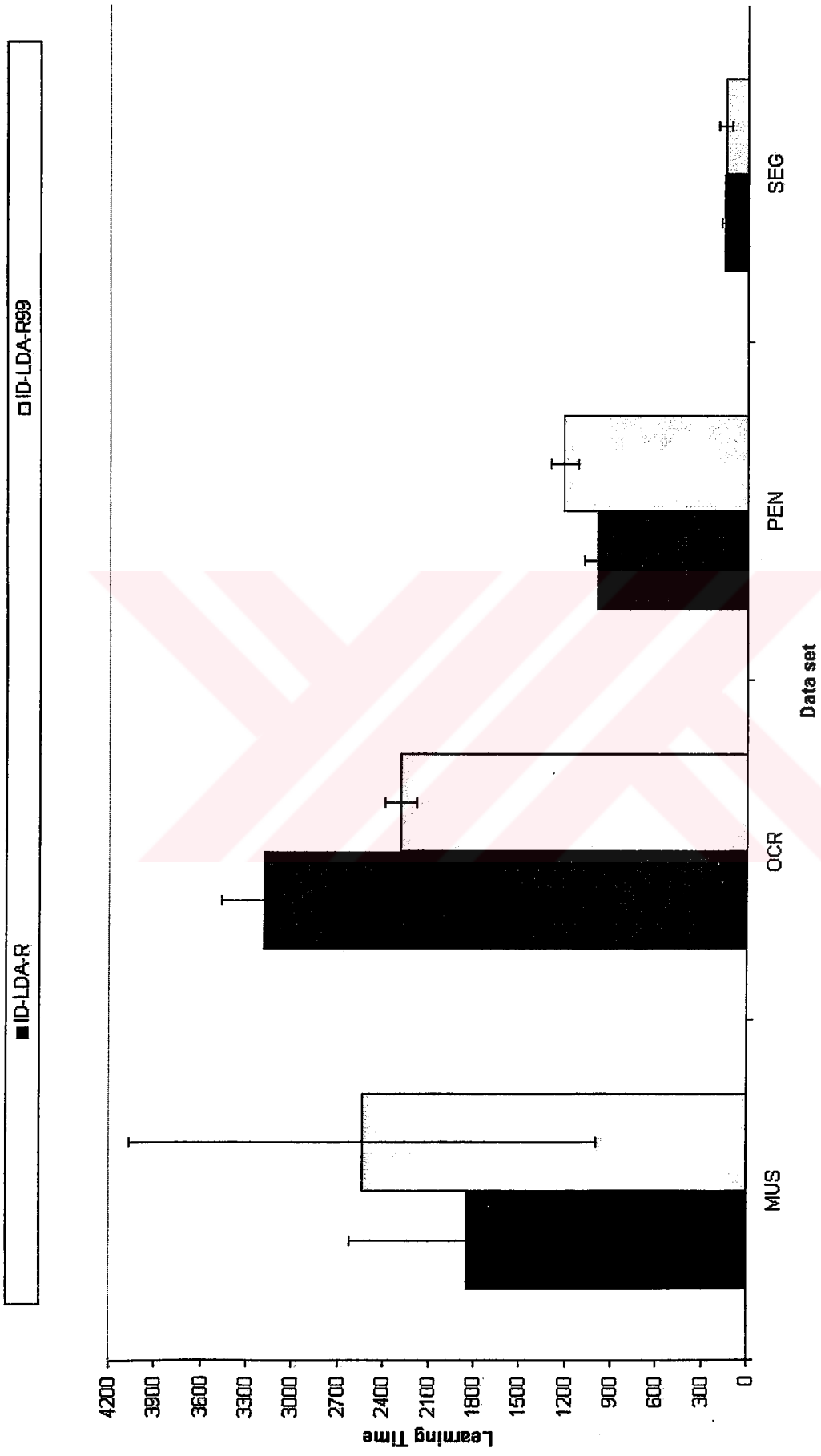

FIGURE 6.4.2.4 Learning time results for ID-LDA-R and ID-LDA-R99 (large data sets)

6.4.3. Comparison of Different Linear Multivariate Techniques

In this section we compare three types of linear decision tree construction methods. These are CART (Classification and regression trees), ID-LP (Multivariate decision tree with neural perceptron) and ID-LDA (Multivariate decision tree with linear discriminant ID-LDA-R99). The results are shown in Table 6.4.3.1, Table 6.4.3.2 and Figure 6.4.3.1 for accuracy results, in Table 6.4.3.3, Table 6.4.3.4 and Figure 6.4.3.2 for node results and in Table 6.4.3.5, Table 6.4.3.6, Figure 6.4.3.3, Figure 6.4.3.4 and Figure 6.4.3.5 for learning time results. The exchange method is used for simplicity for ID-LP and ID-LDA.

TABLE 6.4.3.1 Accuracy results for linear decision tree methods

Data set name	CART	ID-LP	ID-LDA	Significance
Breast	94.85±1.44	96.60±0.61	95.85±0.72	
Bupa	61.74±3.38	63.53±2.76	67.42±2.97	
Car	**83.84±2.03**	89.48±4.01	**92.09±1.07**	3>>>1
Cylinder	**59.52±4.05**	**70.21±4.48**	**69.80±3.01**	2>>1,3>>>1
Dermatology	**80.87±4.56**	85.74±7.06	**96.17±1.59**	3>>1
Ecoli	**74.74±3.80**	**82.62±4.06**	**83.75±2.53**	2>>1,3>1
Flare	81.55±3.60	88.36±2.37	88.17±2.83	
Glass	53.93±4.20	54.95±7.83	57.29±4.16	
Hepatitis	**78.96±4.04**	**84.13±2.86**	82.06±5.60	2>>1
Horse	**76.96±3.02**	**82.07±3.48**	81.09±1.64	2>>1
Iris	**89.33±4.44**	**77.60±15.70**	**97.20±1.47**	3>>1>>2
Ironosphere	86.84±4.03	87.80±2.18	91.11±2.22	
Monks	**91.20±6.89**	**66.34±1.87**	**74.31±2.26**	1>>3>>>2
Mushroom	**93.45±1.75**	**99.95±0.03**	**98.25±0.57**	2>>3>>1
Ocrdigits	**81.35±2.08**	**93.87±0.92**	**94.59±0.49**	3>>>1,2>>>1
Pendigits	**87.10±2.91**	91.94±4.16	**95.52±0.44**	3>>1
Segment	88.07±1.69	79.76±11.58	90.31±1.20	
Vote	**90.30±3.17**	**94.71±1.05**	**94.85±2.17**	2>1,3>>1
Wine	**87.30±4.40**	87.75±12.62	**96.07±2.66**	3>1
Zoo	69.92±9.69	79.38±8.10	81.41±7.25	

TABLE 6.4.3.2 Accuracy comparisons

Method	CART	ID-LP	ID-LDA
CART		2	1
ID-LP	7		1
ID-LDA	10	2	

TABLE 6.4.3.3 Node results for linear decision tree methods

Data set name	CART	ID-LP	ID-LDA	Significance
Breast	**11.60±2.67**	**3.00±0.00**	**7.20±0.63**	3>>>2,1>>2
Bupa	**43.20±3.82**	**4.60±1.84**	**8.20±1.93**	1>>>3>>2
Car	**29.00±3.40**	**7.40±0.84**	**12.00±2.54**	1>>>2,1>>>3
Cylinder	**45.00±4.90**	**8.40±1.90**	**16.40±8.95**	1>>3,1>>>2
Dermatology	**28.00±4.74**	**8.80±1.48**	**12.80±1.48**	1>>3>2
Ecoli	**34.00±5.01**	**10.80±2.90**	**20.00±2.71**	1>3>>>2
Flare	**33.80±6.20**	**3.20±2.20**	**5.60±3.13**	1>>>3,1>>>2
Glass	**42.40±4.12**	**10.20±4.64**	**26.20±4.44**	1>>>3>>>2
Hepatitis	**14.00±3.43**	**3.00±0.00**	8.60±3.10	1>>>2
Horse	**28.00±5.19**	**5.00±1.63**	**16.80±4.05**	1>>>3>>>2
Iris	**10.20±2.35**	**4.00±1.05**	**5.40±0.84**	1>>3,1>>2
Ironosphere	**16.40±3.78**	**3.80±1.03**	**11.60±2.67**	1>>2,3>>>2
Monks	**17.80±10.16**	**3.00±0.00**	7.20±2.39	1>>>2
Mushroom	**43.00±6.53**	**3.00±0.00**	**19.20±4.85**	1>>3>>2
Ocrdigits	**70.80±3.98**	**34.80±4.94**	**59.40±2.07**	1>3>>>2
Pendigits	**77.80±10.08**	**30.40±6.40**	**89.00±6.25**	3>>>2,1>>>2
Segment	**45.20±8.97**	**16.60±6.65**	39.80±11.08	1>>2
Vote	**17.20±5.29**	**4.20±1.93**	9.80±2.53	1>>>2
Wine	**9.40±2.27**	**4.40±0.97**	5.40±0.84	1>2
Zoo	**25.20±4.94**	**8.80±1.75**	**11.80±1.93**	1>>>3,1>>>2

TABLE 6.4.3.4 Node comparisons

Method	CART	ID-LP	ID-LDA
CART		0	0
ID-LP	20		10
ID-LDA	12	0	

TABLE 6.4.3.5 Learning time results for linear decision tree methods

Data set name	CART	ID-LP	ID-LDA	Significance
Breast	**107±17**	**5±0**	**2±0**	1>>>2>>>3
Bupa	**252±23**	**3±1**	**1±0**	1>>>2>>3
Car	**1178±148**	**152±16**	**28±6**	1>>>2>>>3
Cylinder	**4589±343**	**19±2**	**74±57**	1>>>2,1>>>3
Dermatology	**858±170**	**42±9**	**16±1**	1>>>2>>3
Ecoli	**221±25**	**57±15**	**6±1**	1>>>2>>>3
Flare	**1032±203**	**9±4**	**4±3**	1>>>2,1>>>3
Glass	**320±25**	**33±9**	**7±1**	1>>>2>>>3
Hepatitis	**209±47**	**1±0**	**2±1**	1>>>2,1>>>3
Horse	**3481±1101**	**14±2**	**94±40**	1>>3>>2
Iris	**31±11**	**3±0**	**0±0**	1>>2>>>3
Ironosphere	**544±94**	**4±1**	**11±4**	1>>>3>2
Monks	**126±61**	**3±0**	**1±0**	1>>2>>>3
Mushroom	**33613±2942**	**628±204**	**2533±1533**	1>>>2,1>>>3
Ocrdigits	**9148±713**	**8035±757**	**2288±108**	2>>>3,1>>>3
Pendigits	**3311±350**	**18340±3319**	**1211±91**	2>>>1>>>3
Segment	**1212±170**	**937±103**	**148±40**	1>2>>>3,
Vote	**805±167**	**6±1**	**9±3**	1>>>2,1>>>3
Wine	**84±26**	**4±1**	**1±0**	1>>>2>>3
Zoo	**453±61**	**10±2**	**2±0**	1>>>2>>>3

TABLE 6.4.3.6 Learning time comparisons

Method	CART	ID-LP	ID-LDA
CART		1	0
ID-LP	18		2
ID-LDA	20	13	

If we compare the three linear methods in terms of accuracy, node size and learning time, we see that:

- Accuracy: ID-LP=ID-LDA>CART.
- Node Size: CART>ID-LDA>ID-LP.
- Learning Time: CART >ID-LP>ID-LDA.

In terms of accuracy, CART outperforms ID-LP in those data sets where ID-LP does not always converge. On the *Monks* data set, CART outperforms ID-LP and ID-LDA quite significantly.

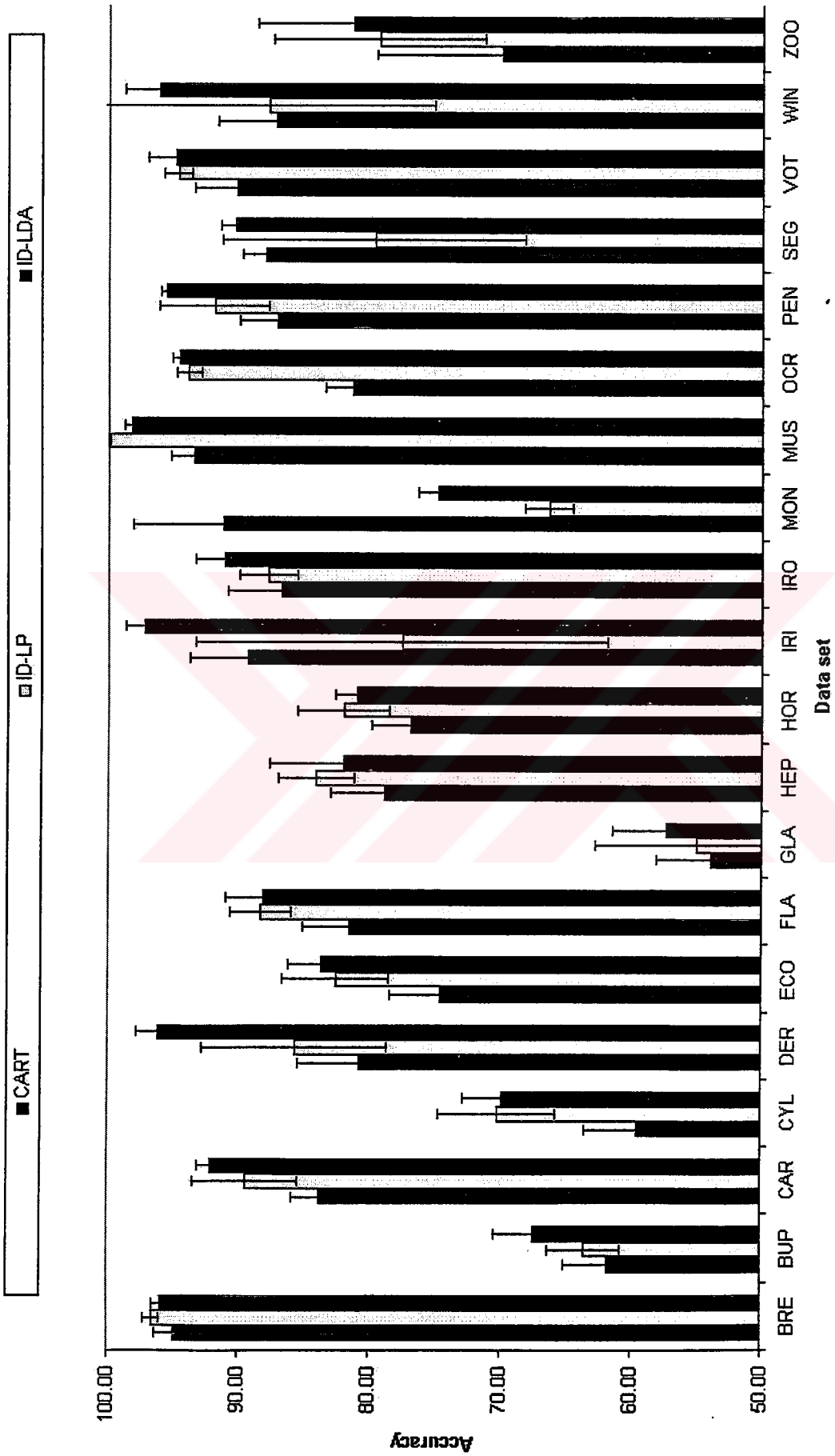

FIGURE 6.4.3.1 Accuracy results for linear decision tree methods

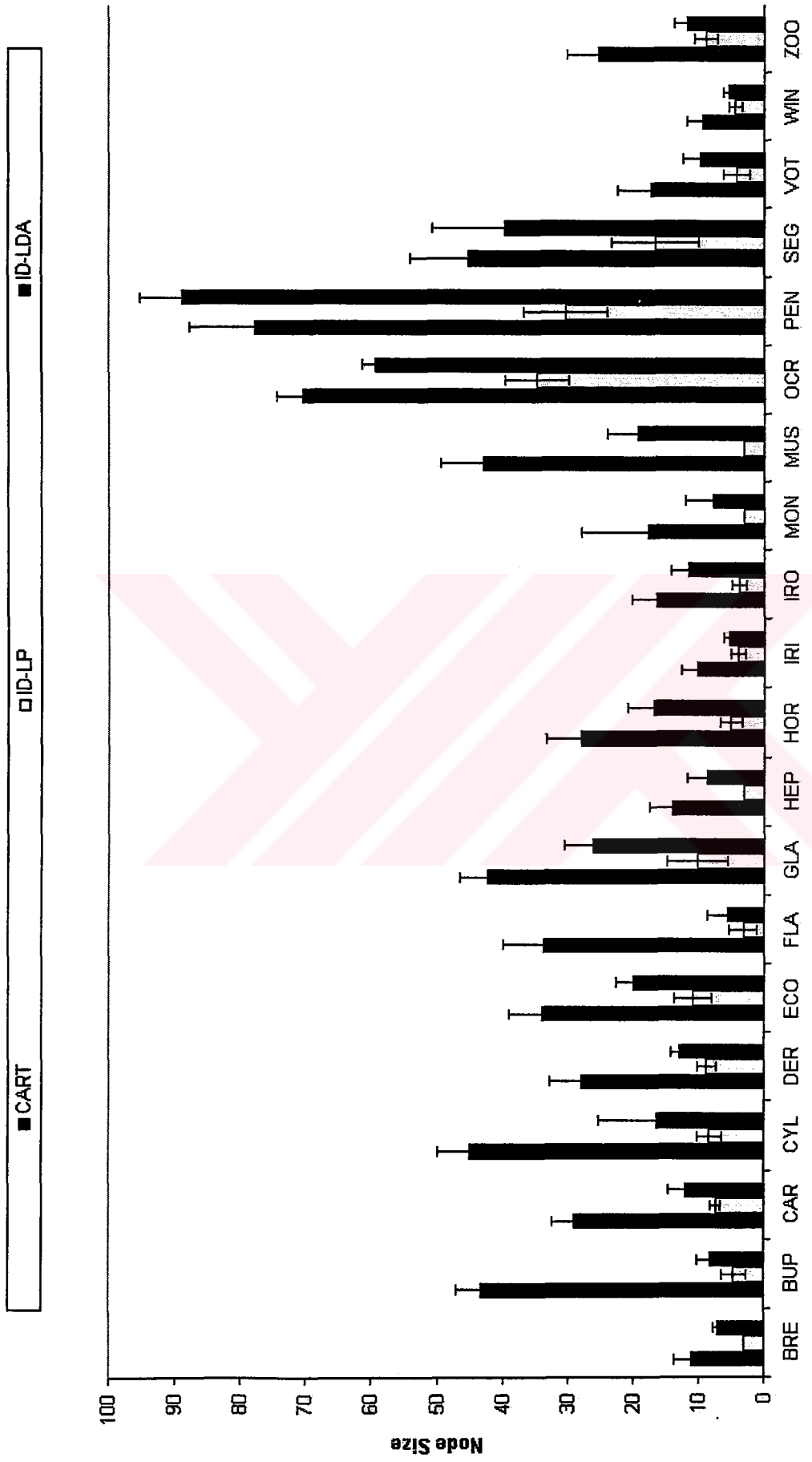

FIGURE 6.4.3.2 Node results for linear decision tree methods

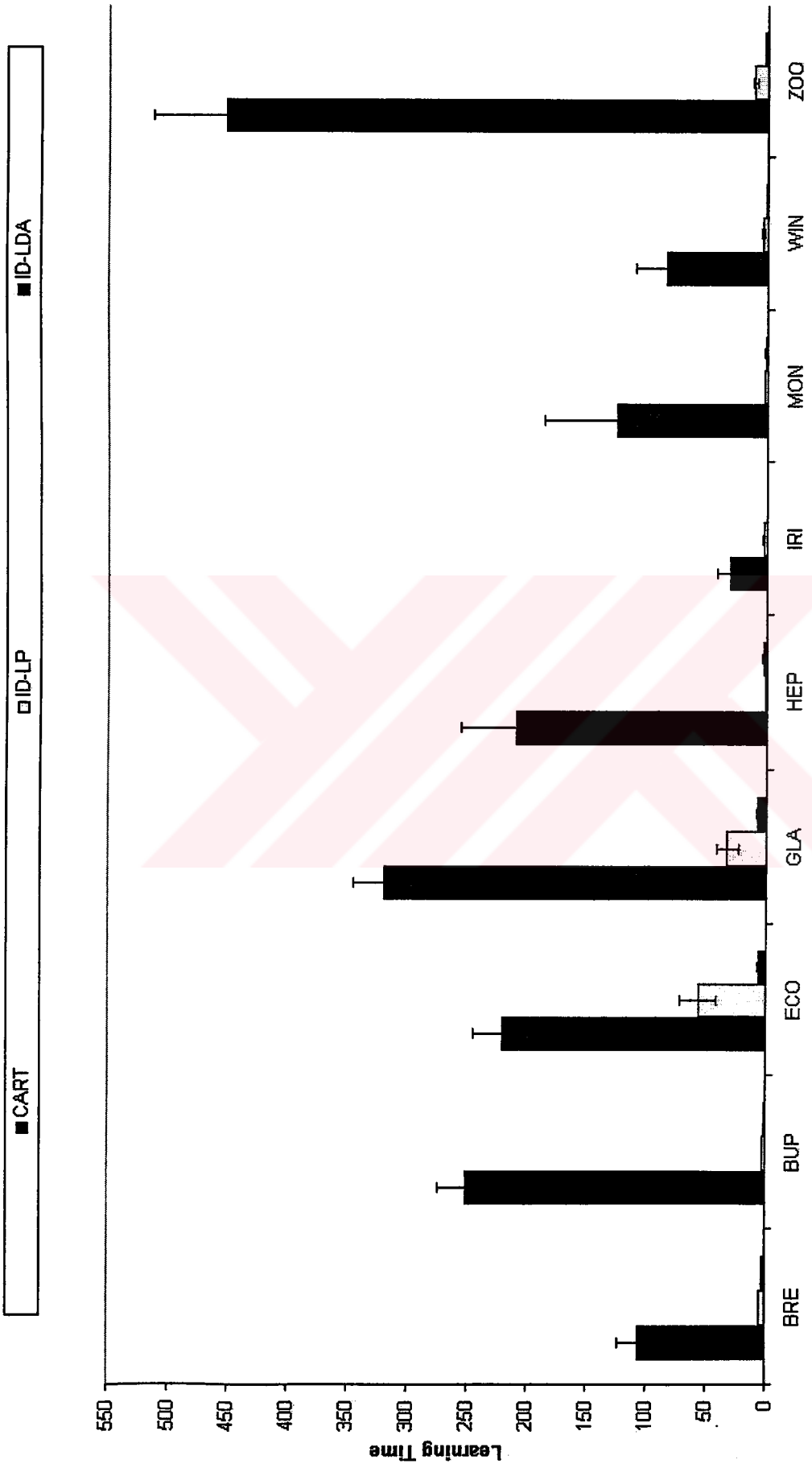

FIGURE 6.4.3.3 Learning time results for linear decision methods (small data sets)

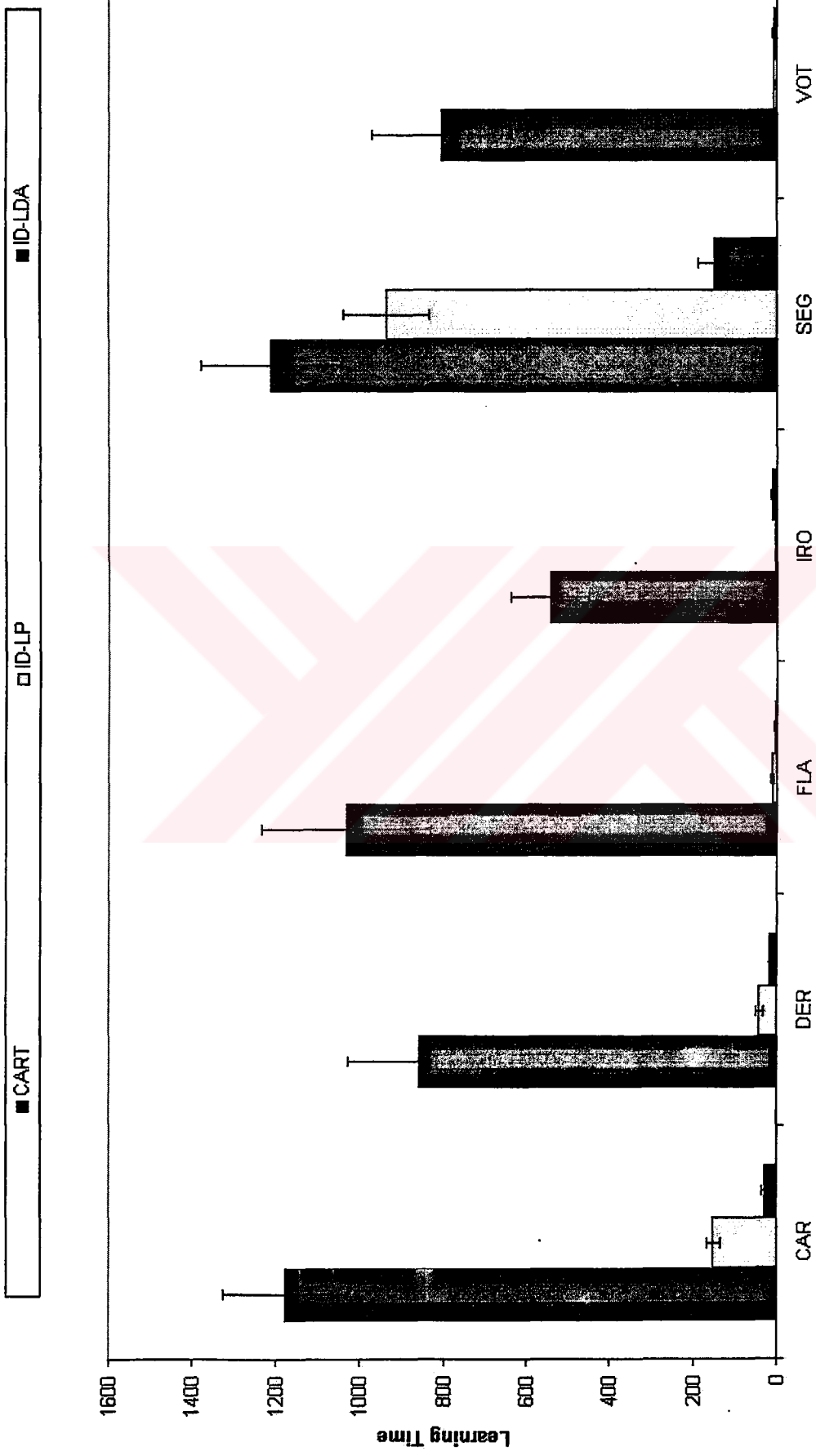

FIGURE 6.4.3.4 Learning time results for linear decision methods (medium size data sets)

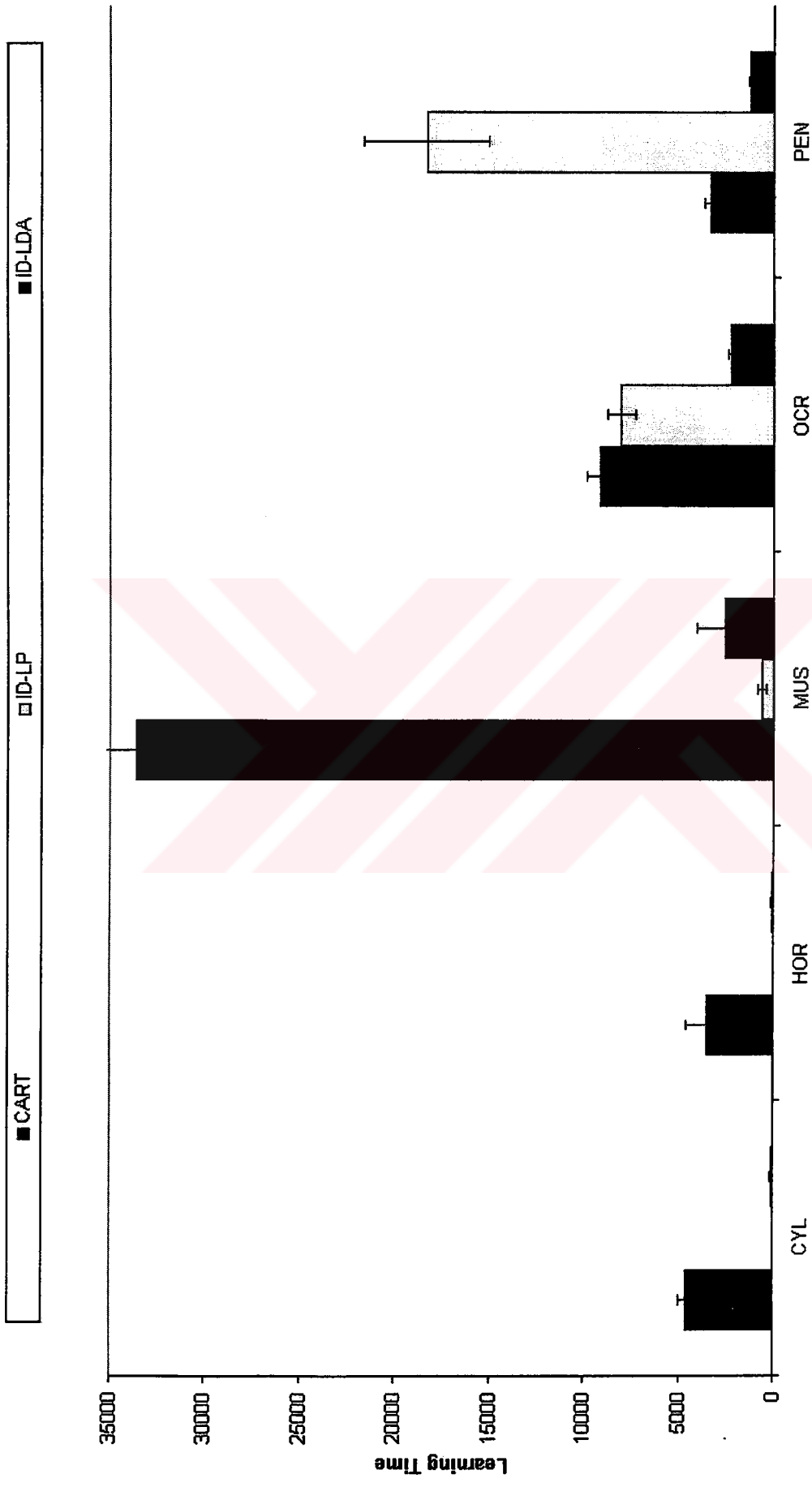

FIGURE 6.4.3.5 Learning time results for linear decision methods (large data sets)

7. CONCLUSIONS

In this study, we have detailed and compared univariate, linear and nonlinear decision tree methods using a set of simulations on twenty standard data sets. For univariate decision tree methods, we have used the ID3 algorithm and for multivariate decision tree methods, we have used the CART algorithm. For linear and nonlinear methods, we have used neural networks at each decision node. We also propose to use the Linear Discriminant Analysis (LDA) algorithm in constructing linear multivariate decision trees.

The comparison results of these four decision tree methods can be seen in Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6 and 7.7. The first four figures compare the four methods in terms of accuracy, tree size and learning time. Last three figures compare the four methods in terms of two of these criteria.

Results for ID3 and CART can be grouped as follows:

- There is not a significant difference between different impurity measures in terms of accuracy, tree size and learning time.
- Post-pruning is better than pre-pruning in terms of tree size; they are same in terms of accuracy and learning time. Also post-pruning sometimes finds significantly better trees than pre-pruning, where pre-pruning stops tree creation process earlier.
- Feature selection does not always improve performance of CART, but increases learning time significantly.

Results for neural trees can be grouped as follows:

- There is not a significant difference between the F-test and the t-tests in terms of accuracy and tree size. But t-test has high learning time because of 30-fold crossvalidation.
- Among the class separation heuristics, the exchange method is better than the selection method in terms of accuracy and tree size but is worse in terms of learning time.

- There is low difference in terms of tree size and accuracy between neural trees (Linear, Nonlinear and Hybrid).
- Learning time results of neural trees is in the following order: Hybrid > Nonlinear > Linear.

Results for LDA trees can be grouped as follows:

- Using PCA with low percentage of explained variance decreases performance in accuracy and tree size.
- Using PCA when it is not required decreases performance in accuracy, tree size and learning time.

Results for univariate and multivariate methods can be ordered as follows:

- Accuracy: ID-LP = ID-LDA > CART ?= ID3.
- Tree Size: ID3 > CART > ID-LDA > ID-LP.
- Learning Time: CART > ID-LP > ID-LDA > ID3.

We can conclude by the following statements:

- If the features are not correlated, we should use univariate decision trees (ID3)
- If the features are correlated, we should use multivariate decision trees.
- If pre-pruning is to be applied, we should not use nonlinear multivariate decision trees.
- If a multivariate method is to be used, do not use CART, instead:
    - If time is important, use ID-LDA.
    - If space is important, use ID-LP.
    - ID-LDA has the same accuracy as ID-LP.

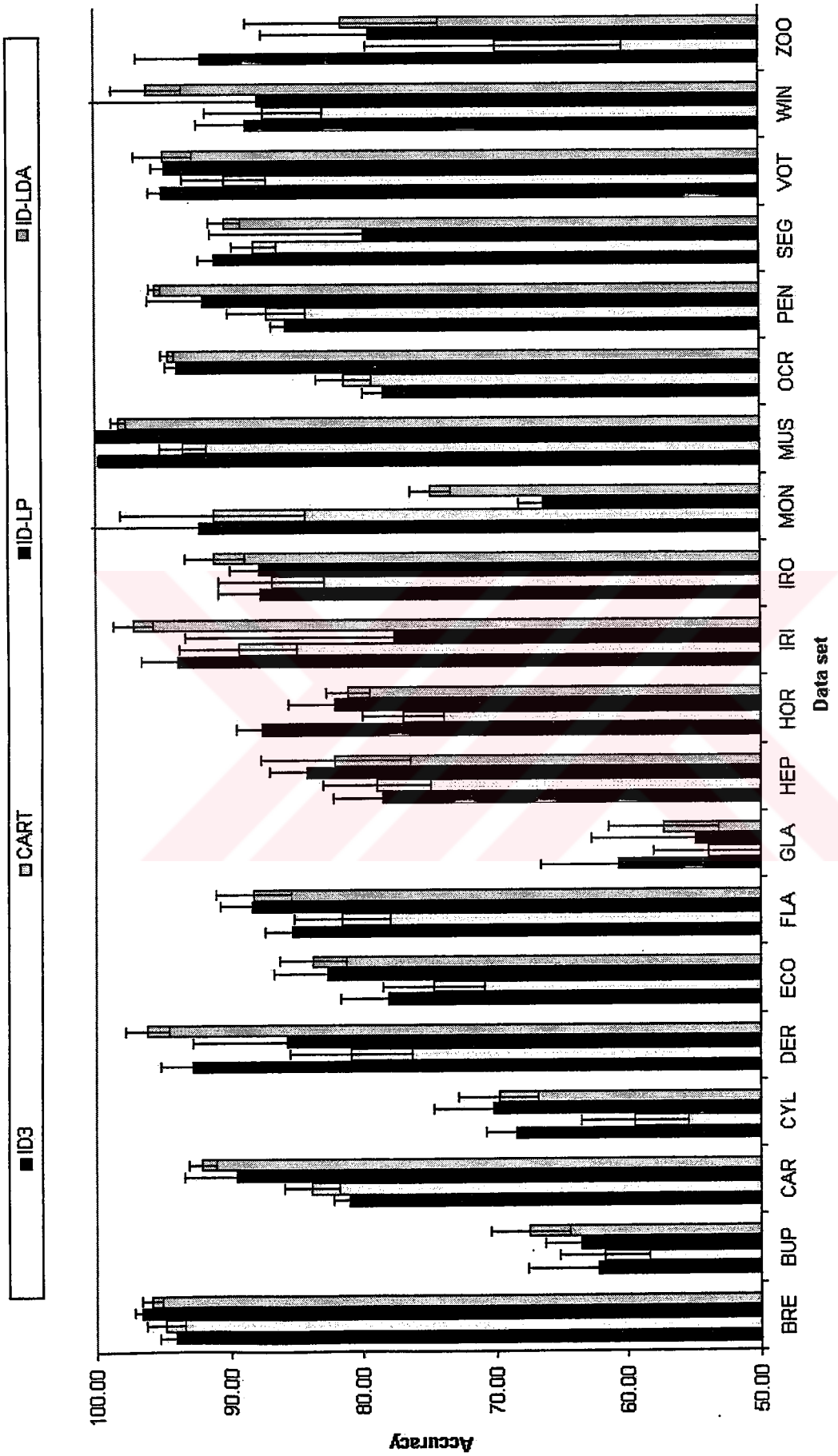

FIGURE 7.1 Comparison of accuracy results of decision tree methods

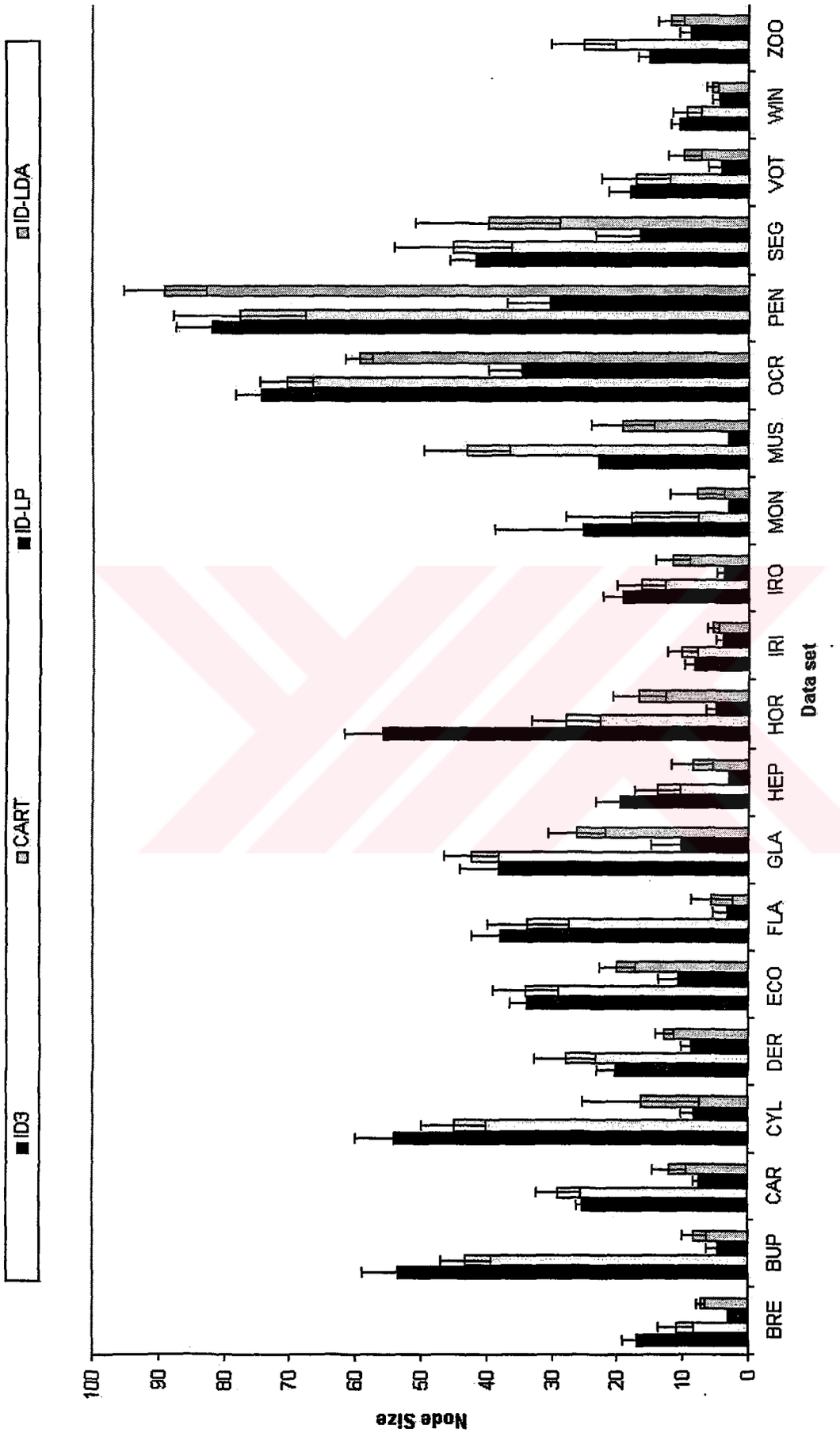

FIGURE 7.2 Comparison of node results of decision tree methods

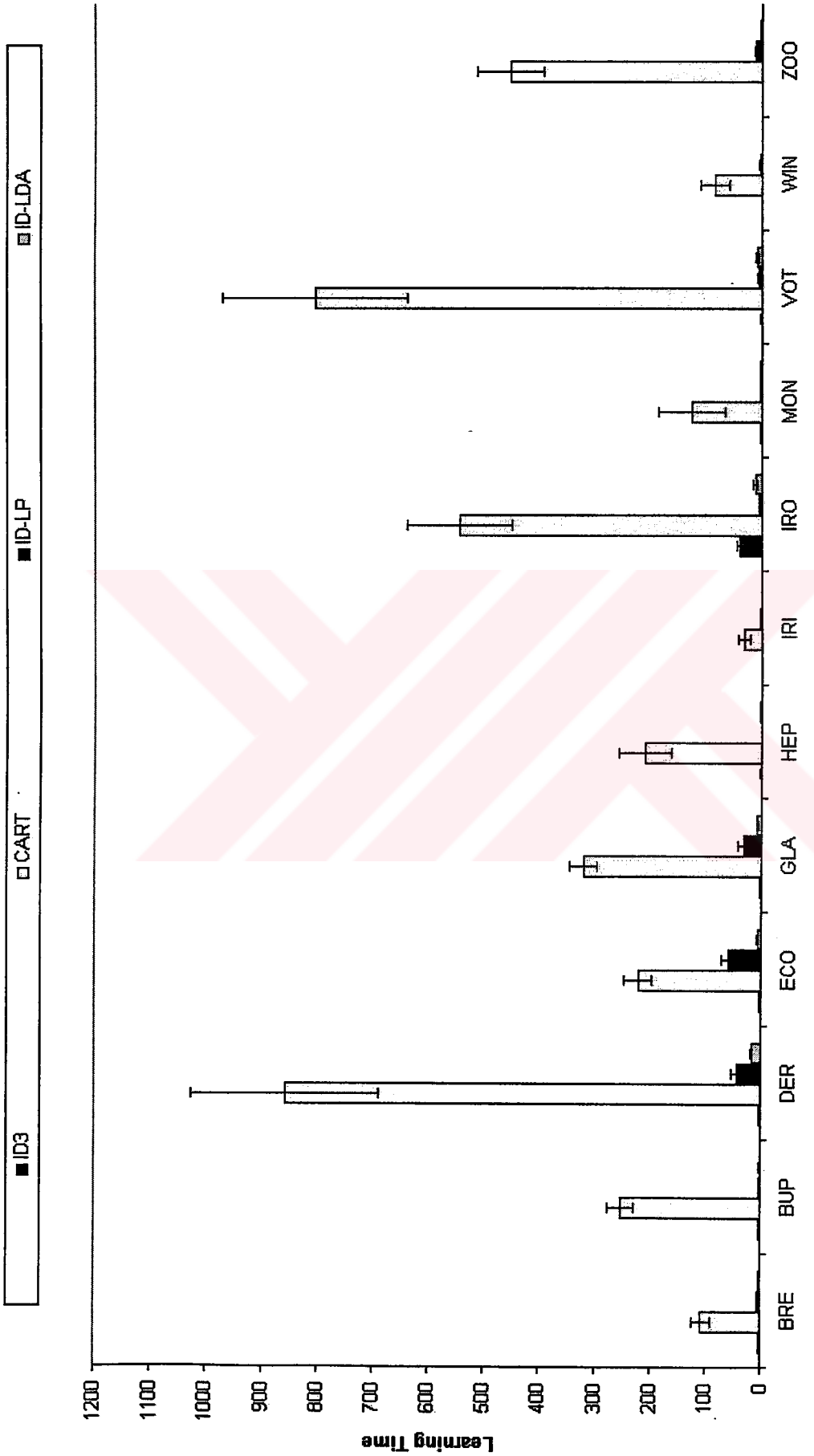

FIGURE 7.3 Comparison of learning time results of decision tree methods (small data sets)

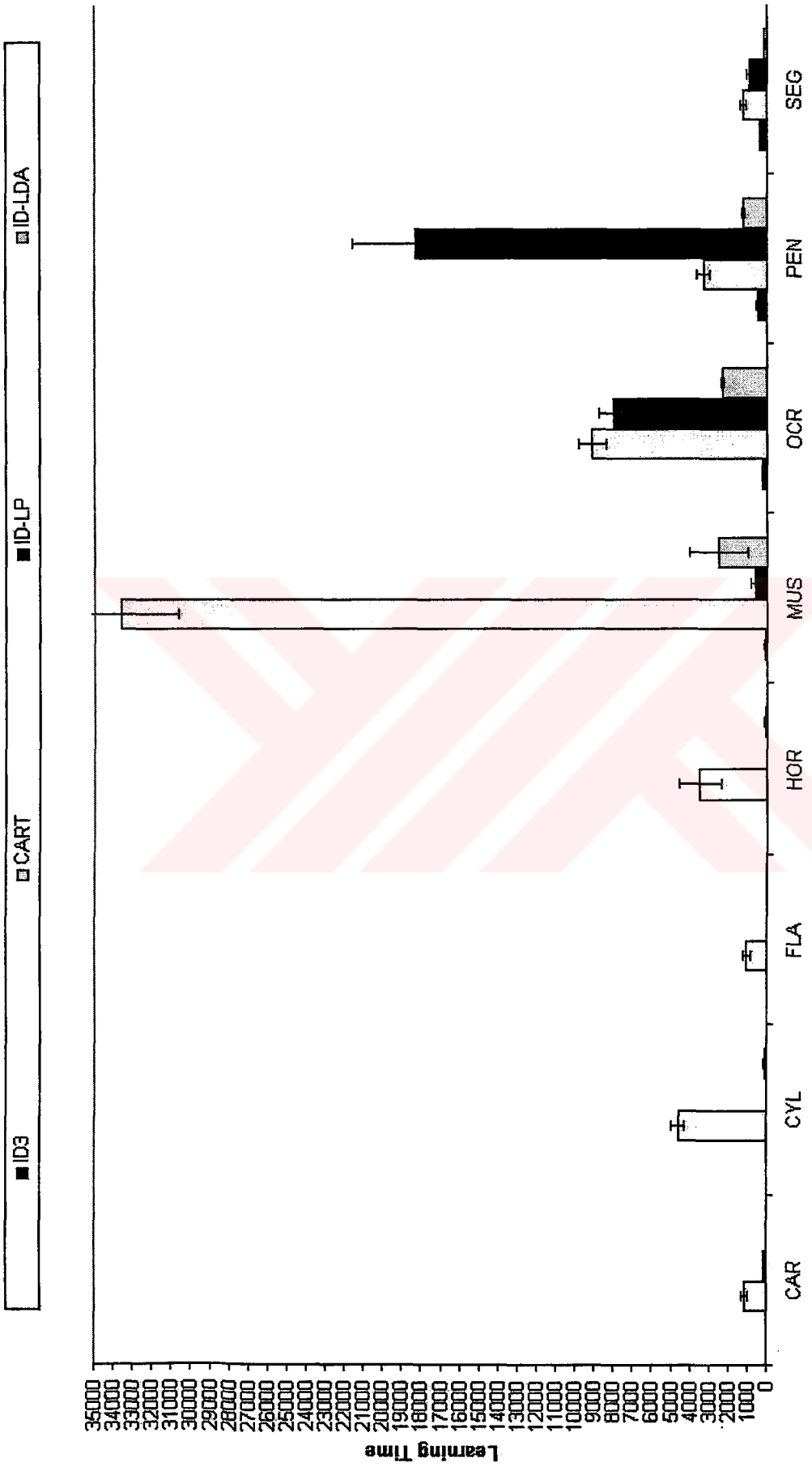

FIGURE 7.4 Comparison of learning time results of decision tree methods (large data sets)

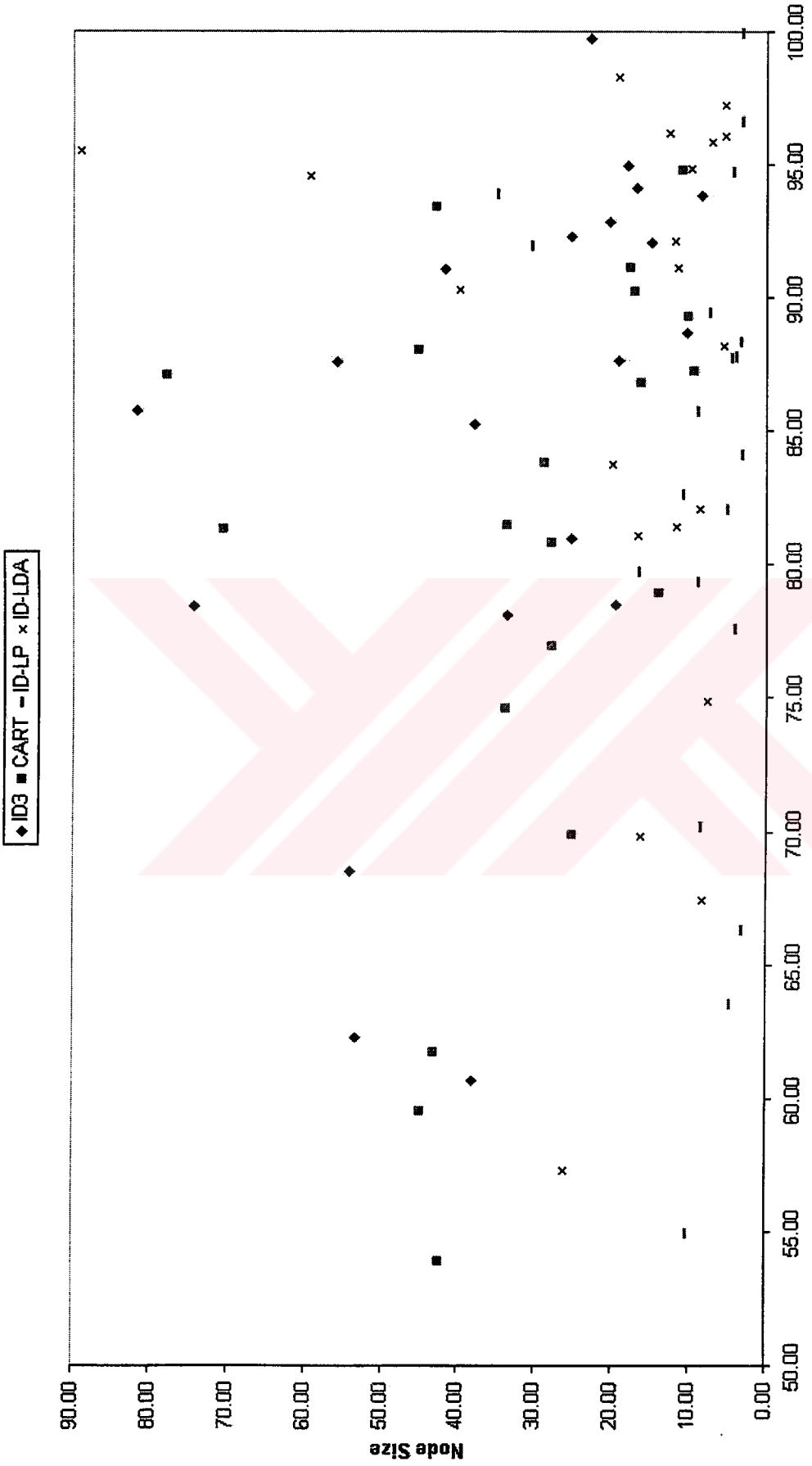

FIGURE 7.5 Comparison of decision tree methods in terms of accuracy and tree size

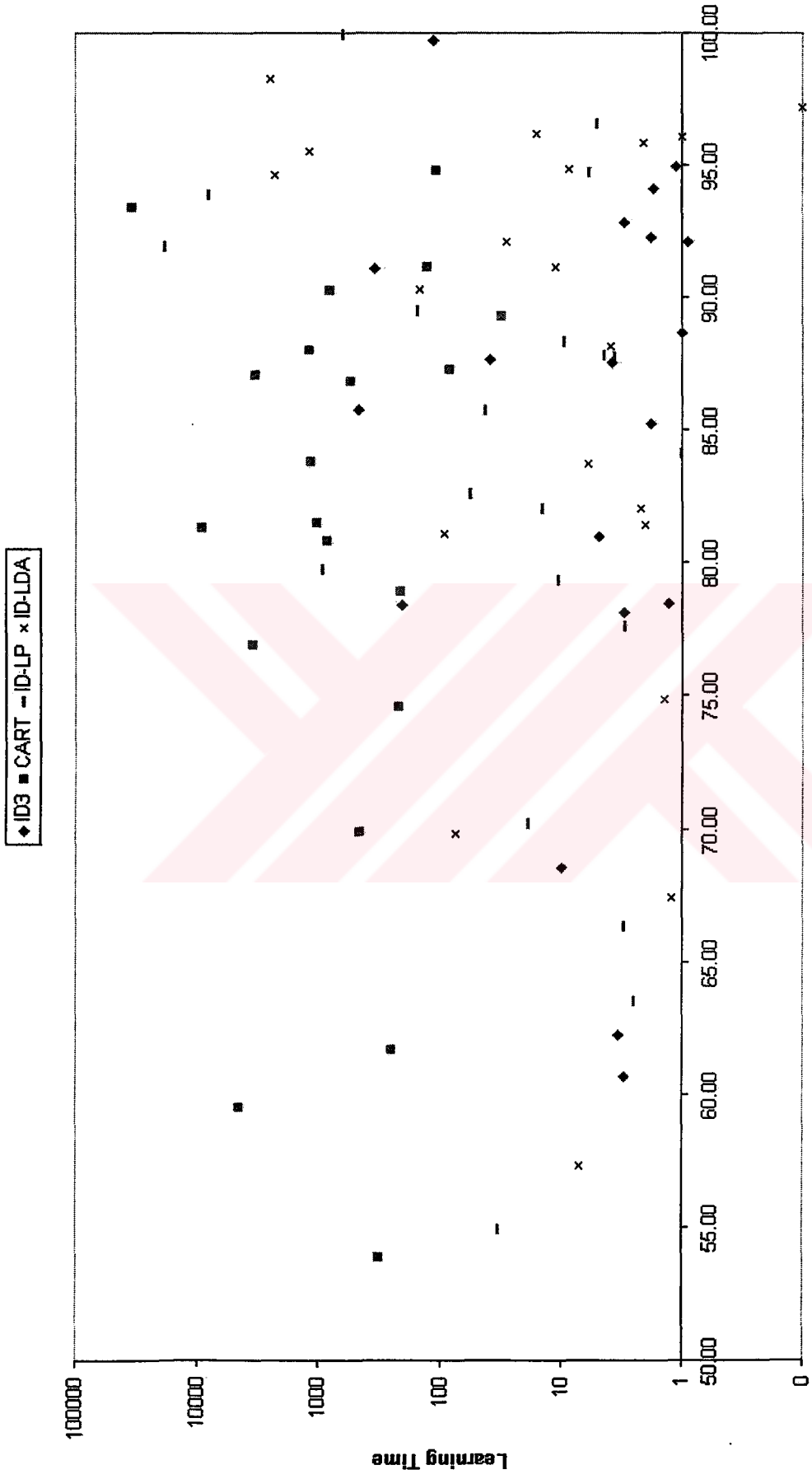

FIGURE 7.6 Comparison of decision tree methods in terms of accuracy and learning time

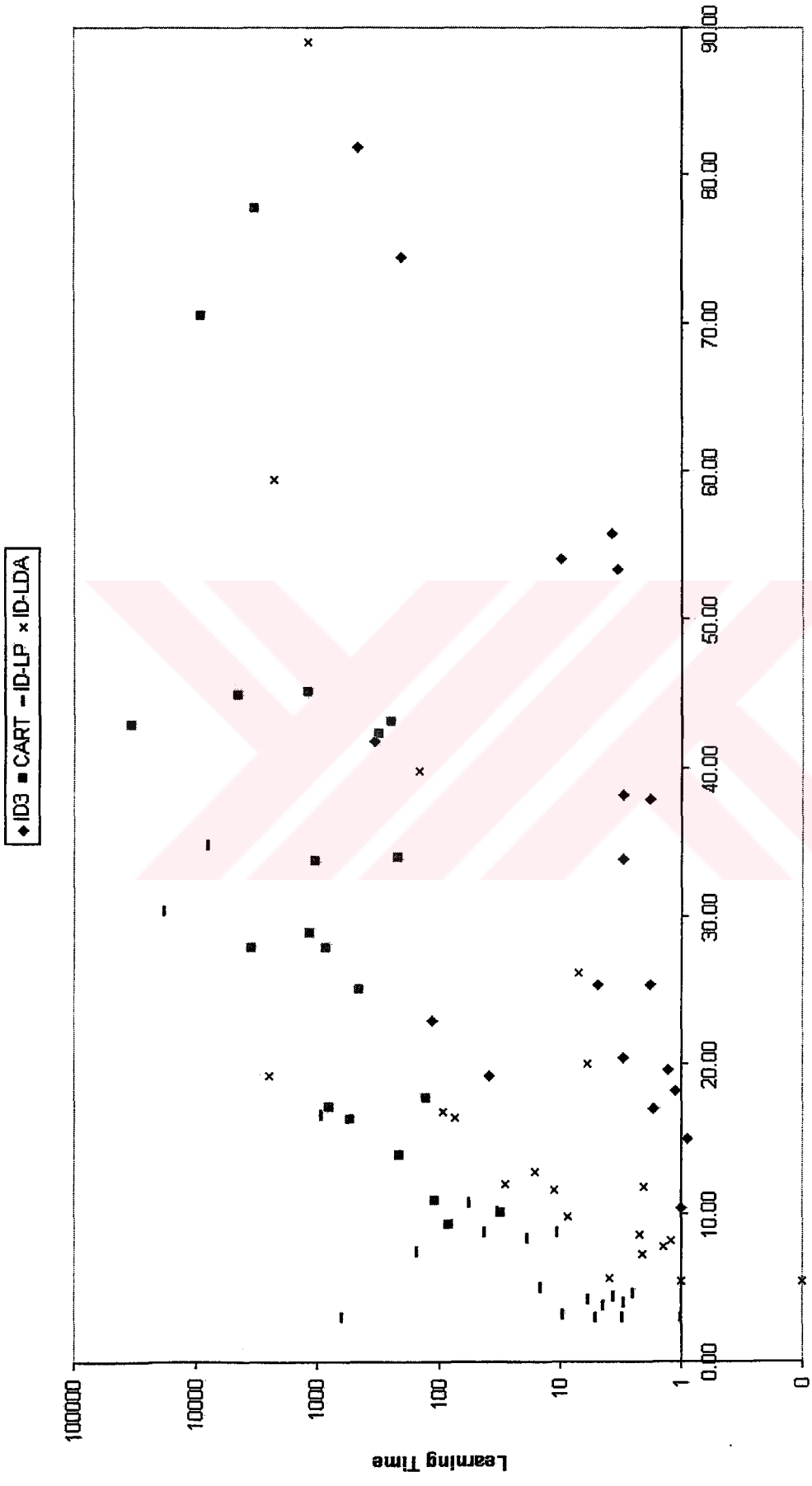

FIGURE 7.7 Comparison of decision tree methods in terms of tree size and learning time

APPENDIX A

Brief description of data sets used in thesis is given below: (See http://www.ics.uci.edu/~mlearn/MLRepository.html)

Breast: This data set was created by Dr. William H. Wolberg (physician) University of Wisconsin Hospitals USA. The aim is to detect the type of breast cancer (malignant or benign) from 9 different attributes.

Bupa: This data set was created by BUPA Medical Research Ltd. Five Attributes of the data set are blood tests results, by which users want to find out liver disorders induced by alcohol consumption.

Car: This data set was created by Marko Bohanec. Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX (M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.)

Cylinder: This data set was created by Bob Evans RR Donnelley & Sons Co. 40 attributes are used to determine the existence of cylinder bands.

Dermatology: This data set was created by Nilsen İlter from Gazi University and H Altan Güvenir from Bilkent University. The aim is to determine the type of Eryhemato-Squamous Disease from clinical and histopathological attributes.

Ecoli: This data set was created by Kenta Nakio from Institue of Molecular and Cellular Biology in Osaka University. The aim is to localize the site of the protein.

Flare: This data set was created by Gary Bradshaw. The database contains 3 potential classes, one for the number of times a certain type of solar flare occured in a 24 hour period. Each instance represents captured features for 1 active region on the sun.

Glass: This data set was created by B. German from Central Research Establishment. The aim is to find out type of the glass.

Hepatitis: The task for this domain is to predict from test results whether a patient will live or die from hepatitis.

Horse: This data set was created by Mary McLeish and Matt Cecile. The aim of this data set is to determine whether the lesion is surgical.

Iris: This data set was created by R. A. Fisher. This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The aim is to decide the class of the iris plant.

Ironosphere: This data set was created by Vince Sigillito. The aim is to find out the type of the radar returns. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Monks: This data set was created by Sebastian Thrun from Carnegie Mellon University. The MONK's problem were the basis of a first international comparison of learning algorithms. The result of this comparison is summarised in "The MONK's Problems - A Performance Comparison of Different Learning algorithms".

Mushroom: Mushroom records were drawn from The Audubon Society Field Guide to North American Mushrooms. This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as edible or poisonous.

Ocrdigits: This data set was created by E. Alpaydın and C. Kaynak from Boğaziçi University. The aim is to recognize optically of ten different digits.

Pendigits: This data set was created by E. Alpaydın and F. Alimoğlu from Boğaziçi University. The aim is to recognize pen written digits.

Segment: This data set was created by Vision Groups from University of Massachusetts. For this data set the task is to learn to segment an image into seven classes: sky, cement, window, brick, grass, foliage and path. The data set was formed from seven images of buildings from the University of Massachusetts campus that were segmented by hand to create the class labels.

Vote: This data set was drawn from Congressional Quarterly Almanac. This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The task is to determine the party of the senators voted.

Wine: This data set was formed by Forina, M. et al, PARVUS of Institute of Pharmaceutical and Food Analysis and Technologies. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Zoo: This data set was created by Richard Forsyth. The animal in zoo are divided into seven groups and the task is to find from 17 different attributes of the animals the type of the animal.

APPENDIX B

Statistical tests we have used in this thesis are the F test and the t test.

Steps for the 5x2 F tests are as follows 0:

1. Split the original data randomly into two equal-sized parts. Call the first one training set and the other one, the test set.
2. Run the two algorithms on the training set and test on the test set.
3. For each algorithm divide the number of correct classifications to the size of the test set.
4. Record also other measures such as the number of nodes in the tree and the average time spent in learning.
5. Exchange train and test sets, do steps 2, 3 and 4 again.

In this test, $p_i^{(j)}$ is the difference between the error rates of the two methods on fold $j = 1, 2$ of replication $i = 1, \ldots, 5$. The average on replication i is $p_i = (p_i^{(1)} + p_i^{(2)}) / 2$ and the variance is $s_i^2 = (p_i^{(1)} - p_i)^2 + (p_i^{(2)} - p_i)^2$. The following statistic is approximately F distributed with ten and five degrees of freedom:

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}(p_i^{(j)})^2}{2\sum_{i=1}^{5}s_i^2} \qquad \text{(B.1)}$$

According to the value of f, the hypothesis that they have the same error rate is rejected or accepted according to a specified confidence level.

Steps for the 30 fold cross-validation t test are 0:

- Partition the available data into 30 disjoint subsets T_1, T_2, ... , T_{30} of equal size.
- For each subset T_i use it for test set and the remaining data for training set.
- Train both algorithms with the training set and test them on the test set. Call the difference between error rates of the two methods on the test set at iteration i, δ_i. Let $\overline{\delta}$ denote the average of δ_i.
- Find the estimate of the standard deviation. The following statistic is approximately t-distributed with 29 degrees of freedom.

$$t = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \overline{\delta})^2} \qquad \text{(B.2)}$$

- According to the value of t, the hypothesis that they have the same error rate is accepted or rejected according to a specified confidence level.

REFERENCES

Alpaydın, E., "Combined 5x2 cv F Test for Comparing Supervised Classification Learning Algorithms", *Neural Computation*, Vol. 11, pp.1975-1982, 1999.

Bishop, C., *Neural Networks for Pattern Recognition,* Oxford University Press, 1996.

Breiman, L., J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth: Belmont, CA, 1984.

Breslow, L. A. and D. W. Aha, Simplifying Decision Trees: A Survey, NCARAI Technical Report No. AIC-96-014, 1997.

Brodley, C. E. and P. E. Utgoff, "Multivariate Decision Trees", *Machine Learning* Vol. 19, pp. 45-77, 1995.

Dietterich, T., M. Kearns and Y. Mansour, "Applying the weak learning framework to understand and improve C4.5", *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy: Morgan Kaufmann, 1996.

Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience Publication, 1973.

Esposito, F., D. Malerba and G. Semeraro, "Decision tree pruning as a search in the state space", *Proceedings of the European Conference on Machine Learning,* Vienna, Austria: Springer-Verlag, pp.165-184, 1993.

Fayyad U. M. and K. B. Irani, "The attribute selection problem in decision tree generation", In *Proceedings of AAAI-92,* pp. 104-110, 1992.

Guo, H. and S. B. Gelfond, "Classification Trees with Neural Network Feature Extraction," *IEEE Transactions on Neural Networks*, Vol. 3, pp. 923-933, 1992.

Hampson, S. E. and D. J. Volper, "Linear Function neurons: Structure and Training", *Biological Cybernetics*, Vol. 53, pp. 203-210, 1986.

Holte, R. C., "Very simple classification rules perform well on most commonly used data sets", *Machine Learning,* Vol. 11, pp. 63-91, 1993.

Mathues, C. J. and L. A. Rendell, "Constructive induction on decision trees", *In IJCAI-89,* pp. 645-650, 1989.

Merz, C. J. and P. M. Murphy, UCI Repository of Machine Learning Databases, 1998, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Mingers, J., "An empirical comparison of selection measures for decision tree induction", *Machine Learning,* Vol. 3, pp. 319-342, 1989.

Mitchell, T., *Machine Learning*, McGraw-Hill, 1996.

Pagallo G. and D. Haussler, "Boolean feature discovery in empirical learning", *Machine Learning*, pp. 71-99, 1990.

Rencher, A. C., *Methods of multivariate analysis,* Wiley Series, 1995.

Quinlan, J. R., "Induction of decision trees", *Machine Learning*, Vol. 1, pp. 81-106, 1986.

Quinlan, J. R., "Unknown attribute values in induction", *Proceedings of the Sixth International Workshop on Machine Learning* pp. 164-168, 1989.

Quinlan, J. R., "C4.5: Programs for machine learning", San Mateo, CA: Morgan Kaufmann, 1993.

Utgoff, P. E. and C. E. Brodley. "Linear Machine decision trees", (COINS Technical Report 91-10), Amberst, MA: University of Massachusetts, Department of Computer and Information Science, 1991.

REFERENCES NOT CITED

Bennett, K. P. and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets", *Optimization Methods and Software*, Vol. 1, pp. 23-24, 1992.

Murthy, K. S., S. Kasif and S. Salzberg, "A system for induction of oblique decision trees", *Journal of Artificial Intelligence Research,* Vol. 2, pp. 1-32, 1994.

Oliver, J. J., "Decision Graphs – An Extension of Decision Trees", *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics,* pp. 343-350, 1993.