Comparing Univariate and Multivariate Decision Trees*

Olcay Taner Yıldız, Ethem Alpaydın
Department of Computer Engineering
Boğaziçi University, 80815 İstanbul Turkey
yildizol@cmpe.boun.edu.tr, alpaydin@boun.edu.tr

Abstract:

Univariate decision trees at each decision node consider the value of only one feature leading to axis-aligned splits. In a multivariate decision tree, each decision node divides the input space into two with an arbitrary hyperplane leading to oblique splits. In this paper, we detail and compare using a set of simulations the univariate decision tree induction method ID3 and the multivariate decision tree induction method CART. We see that the multivariate trees when preceded by feature selection are more accurate and are smaller but require longer training time.

1 Introduction

Classification of huge amount of data has always been an extensive research area. One of the methods to accomplish this task is constructing decision trees. In this paper, we will discuss two main methods for decision tree induction: These are ID3 for univariate decision trees [8] and CART for multivariate decision trees [3].

In Sections 2 and 3, we discuss the ID3 and CART algorithms for constructing univariate and multivariate decision trees respectively. In Section 4, a number of issues related to decision trees are discussed. Section 5 contains our simulation results and we conclude in Section 6.

2 ID3 Algorithm

ID3 algorithm [8] learns decision trees by constructing them top-down manner starting with selecting the best attribute to test at the root of the tree. To find the best attribute, each instance attribute is put into a statistical test to determine how well it alone classifies the training examples. The best feature is selected and used as a test node of the tree. A child of the root node is then created for each possible value of the attribute namely two children for ordered features as $x_i<a$ and $x_i>a$, and m children for unordered feature as $x_i=a_1$, $x_i=a_2$ .. $x_i=a_m$, where m is the number of different possible values of the feature x_i. The entire process is then repeated recursively using the training examples associated with each child node to select the best attribute to test at that point in the tree. This form of ID3 algorithm never backtracks and is a greedy search algorithm.

3 CART Algorithm

Consider the two-dimensional instance space shown in Fig 1. To approximate the hyperplane boundary, the corresponding univariate decision tree uses a series of orthogonal splits.

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 x_i axis. This results in larger trees and poor generalization.

But in a multivariate decision tree each test can be based on more than one feature. In Figure 1 with one line we can separate the examples of two classes.

The multivariate decision tree-constructing algorithm selects not the best attribute but the best linear combination of the attributes. The linear combination consists of multiplication of weights with each feature x_i. So there are basically two main operations in multivariate algorithms: Feature Selection determining which features to use and secondly the weights w_i of those features.

* This work is supported by BU Research Funds Grant 98HA0101

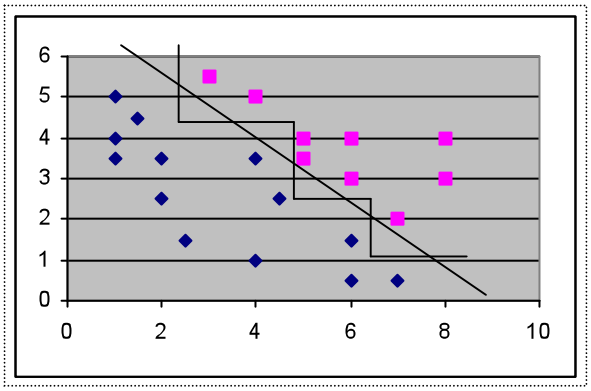

Figure 1. Comparison of univariate and multivariate splits on the plane (Diamond class 1, rectangle class 2)

Because the features in the linear combination split are multiplied with the coefficients, we must convert symbolic features into numeric features. This will be discussed in Section 4. And because all features are numeric, all splits are binary.

CART algorithm for finding the coefficients of the available features is a step-wise procedure, where in each step one cycles through the features $x_1, x_2 \ldots x_n$ doing a search for an improved linear combination split. At the beginning of the L'th cycle, let the current linear combination split be $v \leq c$. For fixed γ, CART searches for the best split of the form:

$v - \delta (x_1 + \gamma) \leq c$, such that

$\delta \geq (v-c)/ (x_1 + \gamma)$ where $x_1 + \gamma \geq 0$ and

$\delta \leq (v-c)/ (x_1 + \gamma)$ where $x_1 + \gamma \leq 0$

The search for δ is carried out for $\gamma = -0.25, 0.0, 0.25$. The resulting three splits are compared, using the chosen partition-merit criterion, and the δ and γ corresponding to the best are used to update v, producing:

$v' = w_1' x_1 + w_2' x_2 \ldots$, where $w_1' = w_1 - \delta$ and $c' = c + \delta\gamma$. This search is repeated for other features x_2, $x_3 \ldots x_n$ to obtain an updated split $v_l < c_l$. The final step of the cycle is to find the best c_l , and the system searches explicitly for the split that minimizes the impurity of the resulting partition. The cycles end when the reduction of impurity is below a constant say ε. [3].

Test operation is the same as in ID3 except before traversing tree each element on the test set must be normalized with the parameters found in the normalization of train set. Each time in selecting which node to go from the parent node, one must take the linear combination of features instead of taking one feature.

4 Issues in tree construction

4. 1 Symbolic and numeric features

A decision tree algorithm must handle both ordered and unordered features. In univariate decision tree algorithm, we can use ordered features as $x_i < a$, where a is in the observed range of the feature i; unordered features as $x_i = a_j$ where a_j are possible values of the unordered feature x_i in the train set.

In 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 we convert it into multiple feature set as $i_1, i_2, i_3 .. i_n$ where $i_j=1$ if x_i takes value a_j and 0 otherwise. At the end of this preprocessing we have converted previous features (some ordered some unordered) $x_1 , x_2 , x_3 , x_4 .. x_f$ into all ordered features as $x_1 , x_{21} , x_{22} , x_{23} .. x_{2n} , x_{31} , x_{32} , x_{33} .. x_{3m}, x_{4,} .. , x_f$ where $x_1 , x_4 ..$ are ordered, $x_{2j} , x_{3j} ..$ are unordered features.

The number of features increases by this conversion which thus increases computational complexity.

This mapping avoids imposing any order on the unordered values of the feature [9], [6].

4. 2 Filling in missing values

In some cases, some of the values of features may not be available. In such cases we must fill the values of missing cases. There are a number of ways of filling missing values [8]. In our implementation if the missing value is for an ordered feature, we fill it with the sample mean value of that feature as it appears in the training set. In the other case, when the feature is unordered, we fill it with the most probable value of that feature in the data set. These statistics are stored so that the same filling can be done for the test set.

4. 3 Representation of Uni & Multivariate Tests

For univariate decision trees at each node the test could be $x_i < a$, $x_i >= a$ for ordered features, $x_i = a_j$ for unordered features.

For a multivariate decision tree at each node the test could be of the form $\mathbf{w}^T \mathbf{x} >= c$, where *w*'s are the coefficients of the features obtained by the multivariate method. **x** is the vector of input attributes [2].

These sets divide the data set at each node into two (for ordered in the univariate and for all in the multivariate) or more (for unordered in the univariate case) subsets. Using the partition-merit criteria, whose goal is to divide the data set into meaningful subsets, these subsets are formed.

At the leaves of the tree, if all the instances of that leaf node belong to the same class C_i then we say that leaf node has the label of C_i. If at the leaf node more than one class such as C_{i1}, C_{i2}, .. C_{ik} have instances then we select that class, which has the most instances.

4. 4 Partition-Merit Criteria

The central choice in tree algorithms is finding the best split *S* in a node *N* that has *K* instances.

We have used as a partition-merit criteria impurity computed as the entropy. Our main idea in selecting best splits is to find the split, which minimizes the impurity or calculated as entropy.

Given a node *N*, which has *K* instances and *c* different classes, the entropy of node *N* is

$$\text{Entropy (N)} = \sum_{i=1}^{c} -p_i \log_2 p_i \qquad (1)$$

P_i is the probability of occurrence of the class C_i . If the node has *c* different values then the value of Entropy (N) can be as large as $\log_2 c$.

Let *S* is a split at a node *N*. It splits that node into *m* nodes. Each node *n* has a number of classes n_c and a number of instances i_k. The class numbered *l* has an instance number of i_{kl} at the node *k*. I_{total} is the number of instances at that node. Then the entropy of split *S* is

$$\text{Entropy (S)} = \sum_{k=1}^{m} \frac{i_k}{i_{total}} \sum_{l=1}^{n_c} \frac{i_{kl}}{i_k} \left(-\log_2 \frac{i_{kl}}{i_k} \right) \quad (2)$$

For an ordered feature there are many splits available and we choose the one with the minimum entropy.

4. 5 Avoiding Overfitting

The algorithms growing the tree just deeply enough to perfectly classify 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.

As ID3 algorithm adds new nodes to the tree, training set accuracy increases. However test set accuracy first increases then decreases.

In our implementation we have used pre-pruning to avoid overfitting. At a node, we stop and do not split further if the number of instances is less than, e.g., 5%, of the training set.

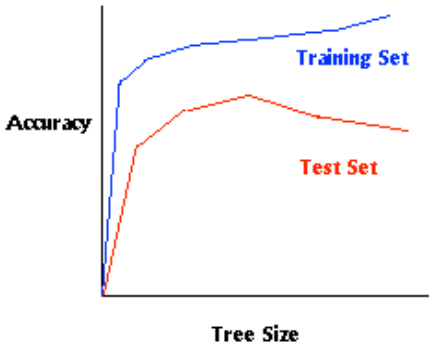

Figure 2. Overfitting in Learning

Other common technique for correcting overfitting in a decision tree model is to first build the tree then prune it back to an appropriate level. [3] [4] [5].

4. 6 Feature Selection

In multivariate decision tree construction, we take linear combination of features. Our main purpose is to find correct 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.

Feature selection algorithm used by CART proceeds as follows: First it determines the coefficients of all features. The impurity is now let say I_{all} . It drops then first feature x_1 by setting w_1=0 and computes the new impurity I_1. The impurity increase is now I_1- I_{all}. Algorithm then computes impurity increases for other features as I_2- I_{all} , I_3- I_{all} . If dropping one feature increases impurity significantly, it means that that feature is an important feature. So the increases in impurities are sorted and the most and least important features are found. Let say feature m is the most and feature l is the least important. Then we check if the impurity increase for the least important feature is less than the impurity increase for the most important feature by a constant c (normally 0.1 or 0.2). If it is smaller we conclude that it is an unnecessary feature and we drop it. Algorithm continues and finds new coefficients for the features but this time without the dropped feature. Once more features are dropped one after another and impurity increases (most and least) are found. The algorithm terminates when the condition between most and least important features is not satisfied.

5 Evaluation of decision tree construction methods

To evaluate two decision tree construction methods, we performed several experiments on datasets from the UCI repository [7].

5. 1 Experimental Method

For each decision tree method, we performed five two-fold cross-validation runs on each data set. A cross-validation run for one data set was performed as follows:

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 algorithm on the training set and test on the test set.
3. For each algorithm divide the number of correct classifications to the element size of the test set.
4. Record also other measures such as number of nodes in the tree and the average time spent in learning.
5. Exchange train and test sets and do steps 2, 3 and 4 again.

ID3 is the univariate decision tree algorithm. Nearest mean is given as a baseline, which is an Euclidean distance classifier to the nearest class mean. CART is the multivariate decision tree algorithm without feature selection and FSCART is CART with prior feature selection.

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 combined 5x2 cv F Test. [1].

Table 1 describes the chosen data sets with the objective of showing different type of characteristics. Breast, Hepatitis and Vote datasets have missing values.

Table 1. Description of the data sets.

Data Set	Classes	Instances	Features
Breast	2	699	9
Bupa	2	345	6
Car	4	1728	7
Ecoli	8	336	7
Flare	3	323	11
Glass	6	214	9
Hepatitis	2	155	19
Iris	3	150	4
Iosphere	2	351	34
Monks	2	432	6
Vote	2	435	16
Wine	3	178	13
Zoo	7	101	16

5. 2 Comparison of Accuracy

Accuracies with the four methods are given in Table 2 and Figure 3.

Table 2. Comparison of accuracies with the four methods. Values in the results are average±standard deviation.

Data Set	ID3	Cart	Nearest Mean	FSCart
Breast	94.11±1.24	95.22±1.80	96.05±0.75	95.25±1.55
Bupa	62.26±5.33	64.35±3.55	58.60±5.14	60.06±3.53
Car	80.97±1.26	92.57±0.89	52.52±2.88	93.36±0.94
Ecoli	78.10±3.57	75.89±5.70	82.32±2.79	77.98±3.50
Flare	85.26±2.03	82.61±4.46	65.03±9.96	83.65±2.68
Glass	49.53±8.45	57.76±5.21	44.11±5.55	58.13±5.82
Hepatitis	78.44±3.71	77.41±6.08	83.47±3.57	79.21±5.94
Iris	93.87±2.75	82.16±3.01	86.80±2.98	92.40±3.33
Ironosphere	86.78±5.08	89.60±4.86	80.01±6.37	83.13±4.33
Monks	92.27±10.15	79.21±10.64	66.62±1.67	98.15±3.13
Vote	94.94±1.06	92.64±3.25	87.63±1.84	93.06±2.45
Wine	88.54±3.81	89.10±4.71	96.52±1.71	89.33±4.47
Zoo	92.06±4.80	68.34±7.77	91.27±4.99	88.53±4.63

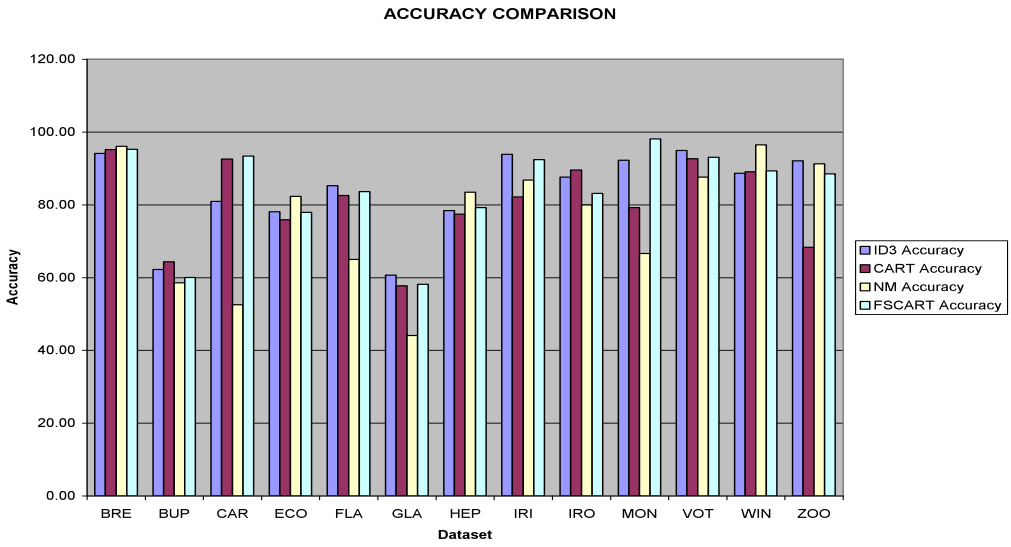

Figure 3. Comparison of Accuracies

Table 3. F distribution results over %90 confidence

Data Set	Comparison
Breast	CART>ID3; NM>ID3
Bupa	Not sure
Car	CART>ID3>NM;FSCART>ID3>NM
Ecoli	Not sure
Flare	Not sure
Glass	CART>NM;FSCART>ID3;FSCART>NM
Hepatitis	ID3>CART;NM>CART
Iris	ID3>CART;FSCART>CART
Ionosphere	CART>NM
Monks	FSCART>ID3>NM;FSCART>CART
Vote	ID3>NM
Wine	NM>ID3
Zoo	ID3>CART;NM>CART;FSCART>CART

5. 3 Comparison of Node Size

The number of nodes indicating tree complexity with the three tree construction methods is given in Table 4 and Figure 4.

Table 4. Comparison of Node Size

Data Set	ID3	Cart	FSCart
Breast	17.0±2.1	11.6±4.9	12.4±3.9
Bupa	53.4±5.5	38.6±5.2	39.6±5.3
Car	25.4±0.7	21.8±3.0	19.6±3.4
Ecoli	33.8±2.7	34.8±4.9	31.0±3.7
Flare	37.9±4.5	31.4±5.3	30.8±4.9
Glass	38.2±5.9	38.4±4.2	37.4±4.3
Hepatitis	19.6±3.8	15.8±3.4	14.4±4.4
Iris	8.4±1.4	16.8±3.5	8.4±1.4
Ionosphere	19.2±3.1	10.6±3.1	16.8±4.9
Monks	25.4±13.5	35.6±10.1	10.8±6.2
Vote	18.2±3.1	8.2±2.2	8.0±2.2
Wine	10.4±1.4	8.8±2.9	7.8±2.4
Zoo	15.0±1.9	25.4±3.9	16.8±2.2

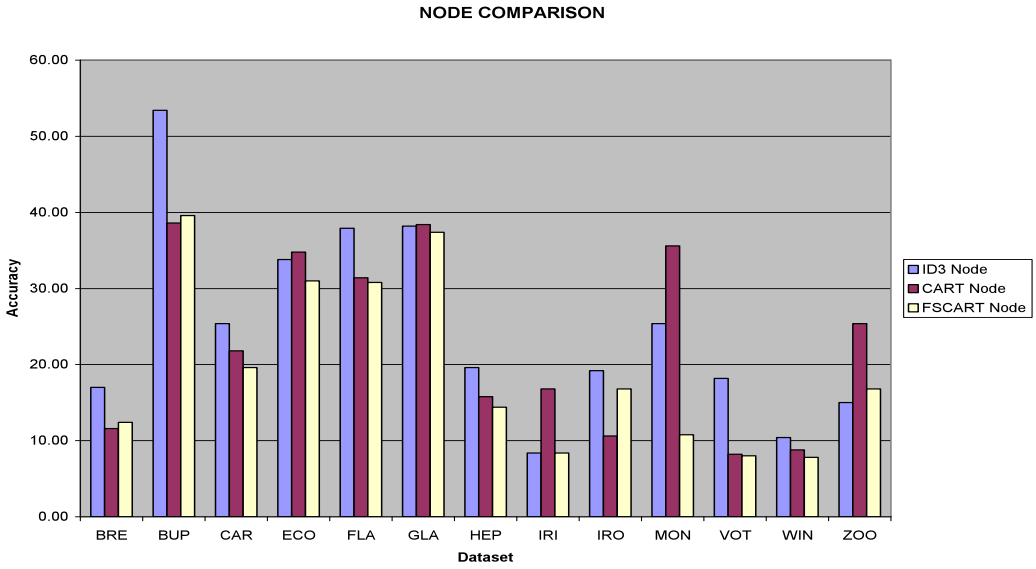

Figure 4. Comparison of Node Size

Conclusions

None of the four methods have given perfect dominance onto others for all datasets. So the accuracy of the methods mainly depends on the dataset.

Feature selection must be done in CART method in order to increase accuracy and minimize the node size.

CART methods needs time approximately ten times more than ID3.

In the datasets where CART with or without feature selection overcomes ID3 method in accuracy, it uses also fewer nodes.

In all datasets except zoo, CART with feature selection has fewer nodes than ID3.

References

1. Alpaydin, E (1998) Combined 5x2 cv F Test for Comparing Supervised Classification Learning Algorithms, IDIAP RR 98-04.

2. Bennett, K.P., & Mangasarian, O. L. (1992) Robust linear programming discrimination of two linearly inseparable sets. Optimization Methods and Software, 1, 23-24

3. Breiman, L. Friedman, J. H. Olshen, R.A. & Stone, C. J. (1984). Classification and Regression Trees.

4. Breslow L. A., Aha D. W. (1997). Simplification Decision Trees: A Survey. NCARAI Technical Report No. AIC-96-014.

5. Brodley C. E, Utgoff P. E. (1995). Multivariate Decision Trees. Machine Learning 19, 45-77.

6. Hampson, S. E., & Volper, D. J. (1986). Linear Function neurons: Structure and Training. Biological Cybernetics, 53,203-21

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

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

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