Top-Down Induction of Decision Trees Classifiers—A Survey

Lior Rokach and Oded Maimon

***Abstract*—Decision trees are considered to be one of the most popular approaches for representing classifiers. Researchers from various disciplines such as statistics, machine learning, pattern recognition, and data mining considered the issue of growing a decision tree from available data. This paper presents an updated survey of current methods for constructing decision tree classifiers in a top-down manner. The paper suggests a unified algorithmic framework for presenting these algorithms and describes the various splitting criteria and pruning methodologies.**

***Index Terms*—Classification, decision trees, pruning methods, splitting criteria.**

I. Introduction

SUPERVISED methods are methods that attempt to discover relationship between the input attributes and the target attribute. The relationship discovered is represented in a structure referred to as a *model*. Usually, models can be used for predicting the value of the target attribute knowing the values of the input attributes. It is useful to distinguish between two main supervised models: *classification models* (*classifiers*) and *regression models*.

Regression models map the input space into a real-valued domain, whereas classifiers map the input space into predefined classes. For instance, classifiers can be used to classify mortgage consumers to good (fully payback the mortgage on time) and bad (delayed payback).

There are many alternatives to represent classifiers. The decision tree is probably the most widely used approach for this purpose. Originally, it has been studied in the fields of decision theory and statistics. However, it was found to be effective in other disciplines such as data mining, machine learning, and pattern recognition. Decision trees are also implemented in many real-world applications.

Given the long history and the intense interest in this approach, it is not surprising that several surveys on decision trees are available in the literature, such as [1]–[3]. Nevertheless, this survey proposes a profound but concise description of issues related specifically to top-down construction of decision trees, which is considered the most popular construction approach. This paper aims to organize all significant methods developed into a coherent and unified reference.

Manuscript received July 15, 2003; revised Octover 10, 2004. This paper was recommended by Associate Editor S. Lakshmivarahan.

The authors are with the Department of Industrial Engineering, Tel-Aviv University, Ramat Aviv 69978, Israel (e-mail: liorr@eng.tau.ac.il).

Digital Object Identifier 10.1109/TSMCC.2004.843247

II. Preliminaries

In a typical supervised learning, a training set of labeled examples is given and the goal is to form a description that can be used to predict previously unseen examples.

The training set can be described in a variety of languages. Most frequently, they are described as a *bag instance* of a certain *bag schema*. The bag schema provides the description of the attributes and their domains. Formally, bag schema is denoted as $R(A \cup y)$. Where A denotes the set of input attributes containing n attributes: $A = \{a_1, \ldots, a_i, \ldots, a_n\}$ and y represents the class variable or the target attribute.

Attributes (sometimes called field, variable or feature) are typically one of two types: nominal (values are members of an unordered set), or numeric (values are real numbers). When the attribute a_i is nominal it is useful to denote by $\mathrm{dom}(a_i) = \{v_{i,1}, v_{i,2}, \ldots, v_{i,|\mathrm{dom}(a_i)|}\}$ its domain values, where $|\mathrm{dom}(a_i)|$ stands for its finite cardinality. In a similar way, $\mathrm{dom}(y) = \{c_1, \ldots, c_{|\mathrm{dom}(y)|}\}$ represents the domain of the target attribute. Numeric attributes have infinite cardinalities.

The set of all possible examples is called the instance space. The instance space is defined as a Cartesian product of all the input attributes domains: $X = \mathrm{dom}(a_1) \times \mathrm{dom}(a_2) \times \ldots \times \mathrm{dom}(a_n)$. The universal instance space (or the labeled instance space) U is defined as a Cartesian product of all input attribute domains and target attribute domain, i.e.: $U = X \times \mathrm{dom}(y)$.

The training set is a bag instance consisting of a set of m tuples (also known as *records*). Each tuple is described by a vector of attribute values in accordance with the definition of the bag schema. Formally, the training set is denoted as $S(R) = (\langle x_1, y_1 \rangle, \ldots, \langle x_m, y_m \rangle)$ where $x_q \in X$ and $y_q \in \mathrm{dom}(y)$.

Usually, it is assumed that the training set tuples are generated randomly and independently according to some fixed and unknown joint probability distribution D over U. Note that this is a generalization of the deterministic case when a supervisor classifies a tuple using a function $y = f(x)$.

This paper uses the common notation of bag algebra to present projection (π) and selection (σ) of tuples (see for instance [4]).

Originally, the machine learning community has introduced the problem of *concept learning*. To learn a concept is to infer its general definition from a set of examples. This definition may be either explicitly formulated or left implicit, but either way it assigns each possible example to the concept or not. Thus, a concept can be formally regarded as a function from the set of all possible examples to the Boolean set {true, false}.

Other communities, such as the data mining community prefer to deal with a straightforward extension of the *concept learning*, know as *the classification problem*. In this case we

search for a function that maps the set of all possible examples into predefined set of class labels and not limited to the Boolean set.

An *inducer*, is an entity that obtains a training set and forms a classifier that represents the generalize relationship between the input attributes and the target attribute.

The notation I represents an inducer and $I(S)$ represents a classifier which was induced by performing I on a training set S.

Most frequently, the goal of the classifiers Inducers is formally defined as:

Given a training set S with input attributes set $A = \{a_1, a_2, \ldots, a_n\}$ and target attribute y from a unknown fixed distribution D over the labeled instance space, the goal is to induce an optimal classifier with minimum generalization error.

Generalization error is defined as the misclassification rate over the distribution D. In case of the nominal attributes it can be expressed as

$$\sum_{\langle x,y\rangle \in U} D(x,y) \cdot L\left(y, I(S)(x)\right) \tag{1}$$

where $L(y, I(S)(x)$ is the loss function defined as

$$L\left(y, I(S)(x)\right) = \begin{cases} 0, & \text{if } y = I(S)(x) \\ 1, & \text{if } y \neq I(S)(x). \end{cases} \tag{2}$$

In case of numeric attributes the sum operator is replaced with the appropriate integral operator.

The classifier generated by the inducer can be used to classify an unseen tuple either by explicitly assigning it to a certain class (crisp classifier) or by providing a vector of probabilities representing the conditional probability of the given instance to belong to each class (probabilistic classifier).

III. Decision Tree Representation

A Decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a *rooted tree*, meaning it is a *directed tree* with a node called *root* that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an *internal* or *test node*. All other nodes are called *leaves* (also known as *terminal nodes* or *decision nodes*).

In a decision tree, each internal node splits the instance space into two or more subspaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case each test considers a single attribute, such that the instance space is partitioned according to the attribute's value. In the case of numeric attributes the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target value having a certain value.

Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path.

Fig. 1 describes a decision tree that reasons whether or not a potential customer will respond to a direct mailing. Internal nodes are represented as circles whereas leaves are denoted as triangles. Note that this decision tree incorporates both nominal and numeric attributes. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral characteristics of the entire population of potential customers—with respect to direct mailing. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values.

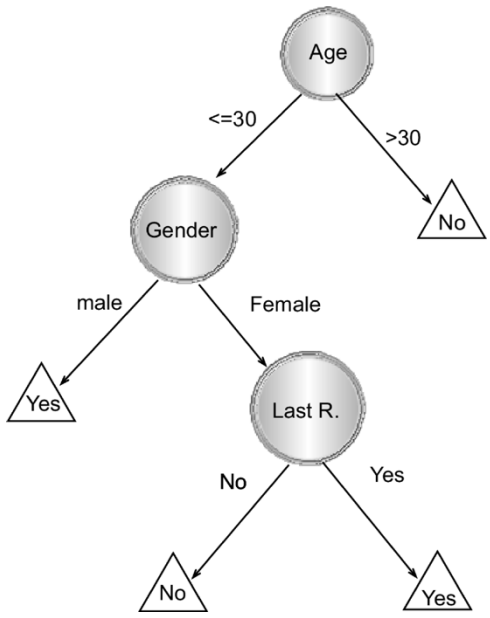

Fig. 1. Decision tree presenting response to direct mailing.

In case of numeric attributes, decision trees can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of the axes.

Naturally, decision-makers prefer less complex decision trees, since they may be considered more comprehensive. Furthermore according to Breiman *et al.* [5] the tree complexity has a crucial effect on its accuracy performance. The tree complexity is explicitly controlled by the stopping criteria used and the pruning method employed. Usually, the *tree complexity* is measured by one of the following metrics:

- the total number of nodes;
- total number of leaves;
- tree depth;
- number of attributes used.

Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value. For example, one of the paths in Fig. 1 can be transformed into the rule: "If customer age ≤ 30, and the gender of the customer is "male," then the customer will respond to the mail." The resulting rule set can then be simplified to improve its comprehensibility to a human user, and possibly its accuracy [6].

IV. Algorithmic Framework for Decision Trees

Decision Tree inducers are algorithms that automatically construct a decision tree from a given dataset. Typically, the goal is

to find the optimal decision tree by minimizing the generalization error. However, other target functions can be also defined, for instance: minimizing the number of nodes or minimizing the average depth.

Induction of an optimal decision tree from a given data is considered to be a hard task. Hancock *et al.* [7] have showed that finding a minimal decision tree consistent with the training set is NP-Hard. Hyafil and Rivest [8] have showed that constructing a minimal binary tree with respect to the expected number of tests required for classifying an unseen instance is NP-complete. Even finding the minimal equivalent decision tree for a given decision tree [9] or building the optimal decision tree from decision tables is known to be NP-Hard [10].

The last references indicate that using optimal decision tree algorithms is feasible only in small problems. Consequently, heuristics methods are required for solving the problem. Roughly speaking, these methods can be divided into two groups: top-down and bottom-up with clear preference in the literature to the first group.

Fig. 2 presents a typical algorithmic framework for top-down inducing of a decision tree. Note that these algorithms are greedy by nature and construct the decision tree in a top-down, recursive manner (also known as "divide and conquer"). In each iteration, the algorithm considers the partition of the training set using the outcome of a discrete function of the input attributes. The selection of the most appropriate function is made according to some splitting measures. After the selection of an appropriate split, each node further subdivides the training set into smaller subsets, until no split gains sufficient splitting measure or a stopping criteria is satisfied.

There are various top-down decision trees inducers such as ID3 [11], C4.5 [12], and CART [5]. Some consist of two conceptual phases: Growing and pruning (C4.5 and CART). Other inducers perform only the growing phase.

procedure $DTInducer(S, A, y)$
1: $T = TreeGrowing(S, A, y)$
2: Return TreePruning(S,T)

procedure $TreeGrowing(S, A, y)$
1: Create a tree T
2: **if** One of the Stopping Criteria is fulfilled **then**
3: Mark the root node in T as a leaf with the most common value of y in S as the class.
4: **else**
5: Find a discrete function $f(A)$ of the input attributes values such that splitting S according to $f(A)$'s outcomes $(v_1,\ldots,v_n)$ gains the best splitting metric.
6: **if** best splitting metric $\geq$ treshold **then**
7: Label the root node in T as $f(A)$
8: **for** each outcome v_i of f(A) **do**
9: $Subtree_i = TreeGrowing(\sigma_{f(A)=v_i} S, A, y)$.
10: Connect the root node of T to $Subtree_i$ with an edge that is labelled as v_i
11: **end for**
12: **else**
13: Mark the root node in T as a leaf with the most common value of y in S as the class.
14: **end if**
15: **end if**
16: Return T

procedure $TreePruning(S, T, y)$
1: **repeat**
2: Select a node t in T such that pruning it maximally improve some evaluation criteria
3: **if** $t \neq \emptyset$ **then**
4: $T = pruned(T, t)$
5: **end if**
6: **until** t=Ø
7: Return T

Fig. 2. Top-down algorithmic framework for decision trees induction. The inputs are S (training set), A (input feature set) and y (target feature).

V. Univariate Splitting Criteria

A. Overview

In most of the cases the discrete splitting functions are univariate. Univariate means that an internal node is split according to the value of a single attribute. Consequently, the inducer searches for the best attribute upon which to split. There are various univariate criteria. These criteria can be characterized in different ways, such as according to the origin of the measure: information theory, dependence, and distance, and according to the measure structure: impurity based criteria, normalized impurity based criteria and binary criteria.

The following sections describe the most common criteria in the literature.

B. Impurity Based Criteria

Given a random variable $\dot{x}$ with k discrete values, distributed according to $P = (p_1, p_2, \ldots, p_k)$, an impurity measure is a function $\phi : [0,1]^k \rightarrow R$ that satisfies the following conditions:

- $\phi(P) \geq 0$;
- $\phi(P)$ is minimum if $\exists i$ such that component $P_i = 1$;
- $\phi(P)$ is maximum if $\forall i, 1 \leq i \leq k, P_i = 1/k$;
- $\phi(P)$ is symmetric with respect to components of P;
- $\phi(P)$ is smooth (differentiable everywhere) in its range.

Note: if the probability vector has a component of 1 (the variable x gets only one value), then the variable is defined as pure. On the other hand, if all components are equal the level of impurity reach maximum.

Given a training set S. The probability vector of the target attribute y is defined as

$$P_y(S) = \left(\frac{|\sigma_{y=c_1} S|}{|S|}, \ldots, \frac{|\sigma_{y=c_{|\mathrm{dom}(y)|}} S|}{|S|} \right). \tag{3}$$

The goodness-of-split due to discrete attribute a_i is defined as a reduction in impurity of the target attribute after partitioning S according to the values $v_{i,j} \in \mathrm{dom}(a_i)$

$$\Delta\Phi(a_i, S) = \phi(P_y(S)) - \sum_{j=1}^{|\mathrm{dom}(a_i)|} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot \phi\left(P_y\left(\sigma_{a_i=v_{i,j}} S\right)\right). \tag{4}$$

Information gain [6] is an impurity based criteria that uses the entropy measure (origin from information theory) as the impurity measure

$$\text{Information Gain } (a_i, S) = \text{Entropy } (y, S) - \sum_{v_{i,j} \in \text{dom}(a_i)} \frac{\left|\sigma_{a_i = v_{i,j}} S\right|}{|S|} \cdot \text{Entropy } \left(y, \sigma_{a_i = v_{i,j}} S\right) \quad (5)$$

where

$$\text{Entropy } (y, S) = \sum_{c_j \in \text{dom}(y)} -\frac{\left|\sigma_{y=c_j} S\right|}{|S|} \log_2 \frac{\left|\sigma_{y=c_j} S\right|}{|S|}.$$

C. Gini Index

Gini index is an impurity-based criteria that meausures the divergence between the probability distributions of the target attribute's values. The Gini index has been used in various works (see [5] and [13]). The Gini index is defined as

$$\text{Gini } (y, S) = 1 - \sum_{c_j \in \text{dom}(y)} \left(\frac{\left|\sigma_{y=c_j} S\right|}{|S|} \right)^2. \quad (6)$$

Consequently, the evaluation criteria for selecting the attribute a_i is defined as

$$\text{Gini Gain } (a_i, S) = \text{Gini } (y, S) - \sum_{v_{i,j} \in \text{dom}(a_i)} \frac{\left|\sigma_{a_i = v_{i,j}} S\right|}{|S|} \cdot \text{Gini } \left(y, \sigma_{a_i = v_{i,j}} S\right). \quad (7)$$

D. Likelihood Ratio Chi-Squared Statistics

The likelihood ratio is defined as [14]:

$$G^2(a_i, S) = 2 \cdot \ln(2) \cdot |S| \cdot \text{Information Gain } (a_i, S). \quad (8)$$

This ratio is useful for measuring the statistical significance of the information gain criteria. The zero hypothesis (H_0) is that the input attribute and the target attribute are conditionally independent. If H_0 holds, the test statistic is distributed as χ^2 with degrees of freedom equal to: $(\text{dom}(a_i) - 1) \cdot (\text{dom}(y) - 1)$.

E. Normalized Impurity Based Criteria

The Impurity Based Criterion described above is biased toward attributes with larger domain values. Namely, it prefers input attributes with many values over attributes with less values [11]. For instance, an input attribute that represents the national security number, will probably get the highest information gain. However, adding this attribute to a decision tree will result with a poor generalized accuracy.

For that reason, it is useful to "normalize" the impurity-based measures, as described in the following sections.

F. Gain Ratio

Quinlan [12] proposes the *gain ratio* measure that "normalize" the information gain follows:

$$\text{Gain Ratio } (a_i, S) = \frac{\text{Information Gain } (a_i, S)}{\text{Entropy } (a_i, S)}. \quad (9)$$

Note that this ratio is not defined when the denominator is zero. Also the ratio may tend to favor attributes for which the denominator is very small. Consequently, it is suggested in two stages. First the information gain is calculated for all attributes. Then taking into consideration only attributes that have performed at least as good as the average information gain, the attribute that has obtained the best ratio gain is selected. Quinlan [15] has showed that the gain ratio tends to outperform simple information gain criteria both from the accuracy aspect as well as from classifier complexity aspects.

G. Distance Measure

Lopez de Mantras [16], introduced a distance measure. Like Gain Ratio this measure also normalizes the impurity measure. However, it suggests normalizing it in a different way

$$DM(a_i, S) = \frac{\Delta\Phi(a_i, S)}{-\sum_{v_{i,j} \in \text{dom}(a_i)} \sum_{c_k \in \text{dom}(y)} b \cdot \log_2 b} \quad (10)$$

where

$$b = \frac{\left|\sigma_{a_i = v_{i,j} \text{ and } y = c_k} S\right|}{|S|}$$

H. Binary Criteria

The binary criteria are used for creating binary decision trees. These measures are based on the division of the input attribute domain into two subdomains.

Let $\beta(a_i, d_1, d_2, S)$ denote the binary criterion value for attribute a_i over sample S when d_1 and d_2 are its corresponded subdomains. The value obtained for the optimal division of the attribute domain into two mutually exclusive and exhaustive subdomains, is used for comparing attributes, namely

$$\begin{aligned} &\beta^*(a_i, S) = \max \beta(a_i, d_1, d_2, S) \\ &s.t. \\ &d_1 \cup d_2 = \text{dom}(a_i) \\ &d_1 \cap d_2 = \emptyset. \end{aligned} \quad (11)$$

I. Twoing Criteria

Breiman *et al.* [5] point out that the Gini index may encounter problems when the domain of the target attribute is relatively

wide. In this case they suggest using binary criterion called twoing criterion. This criterion is defined as

$$\begin{aligned}&\text{twoing}\,(a_i,d_1,d_2,S)\\&=0.25\cdot\frac{|\sigma_{a_i\in d_1}S|}{|S|}\cdot\frac{|\sigma_{a_i\in d_2}S|}{|S|}\\&\cdot\left(\sum_{c_i\in\mathrm{dom}(y)}\left|\frac{\left|\sigma_{a_i\in d_1\text{and }y=c_i}S\right|}{|\sigma_{a_i\in d_1}S|}-\frac{\left|\sigma_{a_i\in d_2\text{and }y=c_i}S\right|}{|\sigma_{a_i\in d_2}S|}\right|\right)^2.\end{aligned}\tag{12}$$

When the target attribute is binary the Gini and twoing criteria are equivalent. For multiclass problems, the twoing criteria prefers attributes with evenly divided splits.

J. Orthogonality Criterion

Fayyad and Irani [17] have presented the orthogonality (ORT) criterion. This binary criteria is defined as

$$\mathrm{ORT}(a_i,d_1,d_2,S)=1-cos\theta(P_{y,1},P_{y,2})\tag{13}$$

where $\theta(P_{y,1},P_{y,2})$ is the angle between two distribution vectors $P_{y,1}$ and $P_{y,2}$ of the target attribute y on the bags $\sigma_{a_i\in d_1}S$ and $\sigma_{a_i\in d_2}S$, respectively.

Fayyad and Irani [17] showed that this criterion performs better than the information gain and the Gini index for specific problem constellations.

K. Kolmogorov–Smirnov Criteria

Friedman [18] and Rounds [19] have suggested a binary criterion that uses Kolmogorov–Smirnov distance. Assuming a binary target attribute, namely $\mathrm{dom}(y)=\{c_1,c_2\}$, the criterion is defined as

$$KS(a_i,d_1,d_2,S)=\left|\frac{\left|\sigma_{a_i\in d_1\text{ and }y=c_1}S\right|}{|\sigma_{y=c_1}S|}-\frac{\left|\sigma_{a_i\in d_1\text{ and }y=c_2}S\right|}{|\sigma_{y=c_2}S|}\right|.\tag{14}$$

Utgoff and Clouse [20] suggest extending this measure to handle target attribute with multiple classes and missing data values. Their results indicate that the suggested method outperforms the gain ratio criteria.

L. Other Univariate Splitting Criteria

Additional univariate splitting criteria can be found in the literature, such as permutation statistic [21], mean posterior improvement [22], and hypergeometric distribution measure [23].

M. Comparison of Univariate Splitting Criteria

Comparative studies of the splitting criteria described above, and others, have been conducted by several researchers during the last thirty years, such as [5], [17], [24]–[28], [71], [73]. Most of these comparisons are based on empirical results, although there are some theoretical conclusions.

Most of the researchers point out that in most of the cases the choice of splitting criteria will not make much difference on the tree performance. Each criterion is superior in some cases and inferior in others, as the "no-free-lunch" theorem suggests.

VI. Multivariate Splitting Criteria

In multivariate splitting criteria several attributes may participate in a single node split test. Obviously, finding the best multivariate criteria is more complicated than finding the best univariate split. Furthermore, although this type of criteria may dramatically improve the tree's performance, these criteria are much less popular than the univariate criteria.

Most of the multivariate splitting criteria are based on linear combination of the input attributes. Finding the best linear combination can be performed using a greedy search [5], [29] linear programming [30], [31], linear discriminant analysis [18], [30], [32]–[35] and others [36]–[38].

VII. Stopping Criteria

The growing phase continues until a stopping criteria is triggered. The following conditions are common stopping rules.

All instances in the training set belong to a single value of y.

The maximum tree depth has been reached.

The number of cases in the terminal node is less than the minimum number of cases for parent nodes.

If the node were split, the number of cases in one or more child nodes would be less than the minimum number of cases for child nodes.

The best splitting criteria is not greater than a certain threshold.

VIII. Pruning Methods

A. Overview

Employing tightly stopping criteria tends to create small and under-fitted decision trees. On the other hand, using loosely stopping criteria tends to generate large decision trees that are over-fitted to the training set. Pruning methods originally suggested by Breiman *et al.* [5] were developed for solving this dilemma. According to this methodology a loosely stopping criterion is used, letting the decision tree to overfit the training set. Then the overfitted tree is cut back into smaller tree by removing sub branches that are not contributing to the generalization accuracy. It has been shown in various studies that employing pruning methods can improve the generalization performance of a decision tree especially in noisy domains.

Another key motivation of pruning is "trading accuracy for simplicity" as presented by Bratko and Bohanec [39]. When the goal is to produce a sufficiently accurate compact concept description, pruning is highly useful. Within this process the initial decision tree is seen as a completely accurate one. Thus, the accuracy of a pruned decision tree indicates how close it is to the initial tree.

There are various techniques for pruning decision trees. Most of them perform top down or bottom up traversal of the nodes. A node is pruned if this operation improves a certain criteria. The following sections describe the most popular techniques.

B. Cost-Complexity Pruning

Breiman *et al.*'s pruning method [5], cost complexity pruning (also known as weakest link pruning or error complexity pruning) proceeds in two stages. In the first stage, a sequence of trees $T_0, T_1, \ldots, T_k$ is built on the training data where T_0 is the original tree before pruning and T_k is the root tree.

In the second stage, one of these trees is chosen as the pruned tree, based on its generalization error estimation.

The tree T_{i+1} is obtained by replacing one or more of the sub-trees in the predecessor tree T_i with suitable leaves. The sub-trees that are pruned are those that obtain the lowest increase in apparent error rate per pruned leaf

$$\alpha = \frac{\varepsilon\left(\text{pruned}\ (T,t), S\right) - \varepsilon(T,S)}{|\text{leaves}\ (T)| - |\text{leaves}\ \ (\text{pruned}\ (T,t))|} \tag{15}$$

where $\varepsilon(T,S)$ indicates the error rate of the tree T over the sample S and $|\text{leaves}(T)|$ denote the number of leaves in T. pruned (T,t) denote the tree obtained by replacing the node t in T with a suitable leaf.

In the second phase the generalization error of each pruned tree $T_0, T_1, \ldots, T_k$ is estimated. The best pruned tree is then selected. If the given dataset is large enough the authors suggest to break it into training set and pruning set. The trees are constructed using the training set and evaluated on the pruning set. On the other hand, if the given dataset is not large enough they propose to use cross-validation methodology, despite the computational complexity implications.

C. Reduced-Error Pruning

Quinlan [6] has suggested a simple procedure for pruning decision trees known as reduced-error pruning. While traversing over the internal nodes from the bottom to the top, the procedure checks for each internal node, whether replacing it with the most frequent class does not reduce the tree's accuracy. In this case, the node is pruned. The procedure continues until any further pruning would decrease the accuracy.

In order to estimate the accuracy Quinlan proposes to use a pruning set. It can be shown that this procedure ends with the smallest accurate subtree with respect to a given pruning set.

D. Minimum-Error Pruning (MEP)

The minimum-error pruning has been proposed by Niblett and Bratko [40]. It performs bottom-up traversal of the internal nodes. In each node it compares the l-probability-error rate estimation with and without pruning.

The l-probability-error rate estimation is a correction to the simple probability estimation using frequencies. If S_t denote the instances that have reached node t, then the error rate obtained if this node was pruned is

$$\varepsilon'(t) = 1 - \max_{c_i \in \text{dom}(y)} \frac{|\sigma_{y=c_i} S_t| + l \cdot p_{\text{apr}}(y = c_i)}{|S_t| + l} \tag{16}$$

where $p_{\text{apr}}(y = c_i)$ is the *a priori* probability of y getting the value c_i, and l denote the weight given to the *a priori* probability. A node is pruned if it does not increase the m probability-error rate.

E. Pessimistic Pruning

Quinlan's pessimistic pruning [12] avoids the need of pruning set or cross validation and uses the pessimistic statistical correlation test instead.

The basic idea is that the error ratio estimated using the training set is not reliable enough. Instead a more realistic measure known as continuity correction for binomial distribution should be used

$$\varepsilon'(T,S) = \varepsilon(T,S) + \frac{|\text{leaves}\ (T)|}{2 \cdot |S|}. \tag{17}$$

However, this correction still produces an optimistic-error rate. Consequently, Quinlan suggests pruning an internal node t if its error rate is within one standard error from a reference tree, namely

$$\varepsilon'\left(\text{pruned}\ (T,t), S\right) \leq \varepsilon'(T,S) + \sqrt{\frac{\varepsilon'(T,S) \cdot (1 - \varepsilon'(T,S))}{|S|}}. \tag{18}$$

The last condition is based on statistical confidence interval for proportions. Usually, the last condition is used such that T refers to a sub-tree whose root is the internal node t and S denote the portion of the training set that refer to the node t.

The pessimistic pruning procedure performs top-down traversing over the internal nodes. If an internal node is pruned then all its descendants are removed from the pruning process, resulting in a relatively fast pruning.

F. Error-Based Pruning (EBP)

Error-based pruning is an evolution of the pessimistic pruning. It is implemented in the well-known C4.5 algorithm.

As in pessimistic pruning the error rate is estimated using the upper bound of the statistical confidence interval for proportions

$$\varepsilon_{UB}(T,S) = \varepsilon(T,S) + Z_\alpha \cdot \sqrt{\frac{\varepsilon(T,S) \cdot (1 - \varepsilon(T,S))}{|S|}} \tag{19}$$

where $\varepsilon(T,S)$ denote the misclassification rate of the tree T on the training set S. Z is the inverse of the standard normal cumulative distribution and α is the desired significance level.

Let subtree (T,t) denote the sub tree rooted by the node t. Let maxchild (T,t) denote the most frequent child node of t (namely, most of the instances in S reach this particular child) and let S_t denote all instances in S that reach the node t.

The procedure performs bottom-up traversal over all nodes and compares the following values:

$$\varepsilon_{UB}\left(\text{subtree}\ (T,t), S_t\right) \tag{20}$$

$$\varepsilon_{UB}\left(\text{pruned}\ \ (\text{subtree}\ (T,t), t), S_t\right) \tag{21}$$

$$\varepsilon_{UB}\left(\text{subtree}\ \ (T, \text{maxchild}\ (T,t)), S_{\text{maxchild}\ (T,t)}\right). \tag{22}$$

According to the lowest value the procedure either leaves the tree as is, prune the node t, or replaces the node t with the subtree rooted by maxchild (T,t).

G. Optimal Pruning

Bratko and Bohanec [39] and Almuallim [41] address the issue of finding optimal pruning.

Bohanec and Bratko [39] introduce an algorithm guaranteeing optimality called optimal pruning (OPT). This algorithm finds the optimal pruning based on dynamic programming, with complexity of $\Theta(|\text{leaves}\ (T)|^2)$ where T is the initial decision tree.

Almuallim [41] introduced an improvement of OPT called OPT-2, which also performs optimal pruning using dynamic programming. However, the time and space complexities of OPT-2 are both $\Theta(|\text{leaves}\ (T*)| \cdot |\text{internal}\ (T)|)$, where $T*$ is the target (pruned) decision tree and T is the initial decision tree.

Since the pruned tree is habitually much smaller than the initial tree and the number of internal nodes is smaller than the number of leaves, OPT-2 is usually more efficient than OPT in terms of computational complexity.

H. Minimum Description Length Pruning

Rissanen [42], Quinlan and Rivest [43] and Mehta *et al.* [44] used the minimum description length (MDL) for evaluating the generalized accuracy of a node. This method measures the size of a decision tree by means of the number of bits required to encode the tree. The MDL method prefers decision trees that can be encoded with fewer bits. Mehta *et al.* [44] indicate that the cost of a split at a leaf t can be estimated as

$$\text{Cost}(T) = \sum_{c_i \in \text{dom}(y)} |\sigma_{y=c_i} S_t| \cdot \ln \frac{|S_t|}{|\sigma_{y=c_i} S_t|} + \frac{|\text{dom}(y)| - 1}{2} \ln \frac{|S_t|}{2} + \ln \frac{\pi^{\frac{|\text{dom}(y)|}{2}}}{\Gamma\left(\frac{|\text{dom}(y)|}{2}\right)} \quad (23)$$

where $|S_t|$ denote the number of instances that have reached to node t.

The splitting cost of an internal node is calculated based on the cost aggregation of its children.

I. Other Pruning Methods

There are other pruning methods reported in the literature. Wallace and Patrick [45] proposed a minimum message length (MML) pruning method. Kearns and Mansour [46] provide a theoretically-justified pruning algorithm.

Mingers [26] proposed the critical value pruning (CVP). This method prunes an internal node if its splitting criterion is not greater than a certain threshold. By that it is similar to a stopping criterion. However, contrary to a stopping criterion a node is not pruned if at least one of its children does not fulfill the pruning criterion.

J. Comparison of Pruning Methods

Several studies aim to compare the performance of different pruning techniques [6], [26], [47].

The results indicate that some methods (such as cost-complexity pruning, reduced-error pruning) tend to over-pruning, i.e., creating smaller but less accurate decision trees. Other methods (like error-based pruning, pessimistic-error pruning and minimum-error pruning) bias toward under-pruning.

Most of the comparisons concluded that the "no-free-lunch" theorem applies in this case also, namely, there is no pruning method that in any case outperforms other pruning methods.

IX. Other Issues

A. Weighting Instances

Some decision trees inducers may give different treatments to different instances. This is performed by weighting the contribution of each instance in the analysis according to a provided weight (between 0 to 1).

B. Misclassification Costs

Several decision trees inducers can be provided with numeric penalties for classifying an item into one class when it really belongs in another.

C. Handling Missing Values

Missing values are a common experience in real world data sets. This situation can complicate both induction (a training set that some of its values are missing) as well as classification (new instance that miss certain values).

This problem has been addressed by several researchers such as Friedman [18], Breiman *et al.* [5] and Quinlan [48]. Friedman [18] suggests handling missing values in the training set in the following way. Let $\sigma_{a_i=?}S$ indicate the subset of instances in S whose a_i values are missing. When calculating the splitting criteria using attribute a_i, simply ignore all instances that their values in attribute a_i are unknown, namely instead of using the splitting criteria $\Delta\Phi(a_i, S)$ it uses $\Delta\Phi(a_i, S - \sigma_{a_i=?}S)$.

On the other hand, Quinlan [48] argues that in case of missing values the splitting criteria should be reduced proportionally as nothing has been learned from these instances. In other words instead of using the splitting criteria $\Delta\Phi(a_i, S)$ it uses the following correction

$$\frac{|S - \sigma_{a_i=?}S|}{|S|} \Delta\Phi\left(a_i, S - \sigma_{a_i=?}S\right). \quad (24)$$

In a case where the criterion value is normalized (like in the case of gain ratio), the denominator should be calculated as if the missing values represent an additional value in the attribute domain.

Once a node is split, Quinlan suggests adding $\sigma_{a_i=?}S$ to each one of the outgoing edges with the following corresponded weight: $|\sigma_{a_i=v_{i,j}}S|/|S - \sigma_{a_i=?}S|$.

The same idea is used for classifying a new instance with missing attribute values. When an instance encounters a node where its splitting criteria can be evaluated due to a missing value, it is passed through to all outgoing edges. The predicted class will be the class with the highest probability in the weighted union of all the leaf nodes at which this instance ends up.

Another approach known as *surrogate splits* was presented by Breiman *et al.* [5] and is implemented in the CART algorithm. The idea is to find for each split in the tree a surrogate split which uses a different input attribute and which most resembles the original split. If the value of the input attribute used in the original split is missing, then it is possible to use the surrogate

split. The resemblance between two binary splits over sample S is formally defined as

$$res(a_i, \mathrm{dom}_1(a_i), \mathrm{dom}_2(a_i), a_j, \mathrm{dom}_1(a_j), \mathrm{dom}_2(a_j), S) = \frac{\left|\sigma_{a_i \in \mathrm{dom}_1(a_i) \text{ and } a_j \in \mathrm{dom}_1(a_j)} S\right|}{|S|} + \frac{\left|\sigma_{a_i \in \mathrm{dom}_2(a_i) \text{ and } a_j \in \mathrm{dom}_2(a_j)} S\right|}{|S|}. \quad (25)$$

When the first split refers to attribute a_i and splits its domain to $\mathrm{dom}_1(a_i)$ and $\mathrm{dom}_2(a_i)$. The alternative split refers to attribute a_j and splits its domain to $\mathrm{dom}_1(a_j)$ and $\mathrm{dom}_2(a_j)$.

Loh and Shih [28] suggest estimating the missing value based on other instances. On the learning phase if the value of a nominal attribute a_i in tuple q is missing, then it is estimated by it mode over all instances having the same target attribute value. Formally

$$est(a_i, y_q, S) = \underset{v_{i,j} \in \mathrm{dom}(a_i)}{\arg\max} \left|\sigma_{a_i = v_{i,j} \text{ and } y = y_q} S\right| \quad (26)$$

where y_q denotes the value of the target attribute in the tuple q. If the missing attribute a_i is numeric then instead of using mode of a_i it is more appropriate to use its mean.

X. Decision Trees Inducers

A. ID3

Quinlan [11] has proposed the ID3 algorithm. It is considered as a very simple decision tree algorithm. ID3 uses information gain as splitting criteria. The growing stops when all instances belong to a single value of target feature or when best information gain is not greater than zero. ID3 does not apply any pruning procedures. Nor does it handle numeric attributes neither missing values.

B. C4.5

C4.5 is an evolution of ID3, presented by the same author [12]. It uses gain ratio as splitting criteria. The splitting ceases when the number of instances to be splitted is below a certain threshold. error-based pruning is performed after the growing phase. C4.5 is capable to handle numeric attributes. It can induce from a training set that incorporates missing values by using corrected gain ratio criteria as presented in Section IX.

C. CART

CART stands for classification and regression trees. It was developed by Breiman *et al.* [5] and is characterized by the fact it constructs binary trees, namely each internal node has exactly two outgoing edges. The splits are selected using the twoing criteria and the obtained tree is pruned by cost-complexity pruning. When provided CART can consider misclassification costs in the tree induction. It also enables users to provide prior probability distribution.

An important feature of CART is its ability to generate regression trees. Regression trees are trees where their leaves predicts a real number and not a class. In case of regression CART looks for splits that minimize the prediction squared error (the least-squared deviation). The prediction in each leaf is determined based on the weighted mean for node.

D. CHAID

Researchers in applied statistics have developed starting from early seventies several procedures for generating decision trees, such as: AID [49], MAID [50], THAID [51] and CHAID [52]. Chisquare-automatic-interaction-detection (CHIAD) was originally designed to handle nominal attributes only. For each input attribute a_i, CHAID finds the pair of values in V_i that is least significantly different with respect to the target attribute. The significant difference is measured by the p value obtained from a statistical test. The statistical test used depends on the type of target attribute. If the target attribute is continuous, an F test is used, if it is nominal, then a Pearson chi-squared test is used, if it is ordinal, then a likelihood-ratio test is used.

For each selected pair CHAID checks if the p value obtained is greater than a certain merge threshold. If the answer is positive it merges the values and searches for an additional potential pair to be merged. The process is repeated until no significant pairs are found.

It then selects the best input attribute to be used for splitting the current node, such that each child node is made of a group of homogeneous values of the selected attribute. Note that no split is performed if the adjusted p value of the best input attribute is not less than certain split threshold. This procedure stops also when one of the following conditions is fulfilled.

Maximum tree depth is reached.

Minimum number of cases in node for being a parent is reached, so it can not be split any further.

Minimum number of cases in node for being a child node is reached.

CHAID handles missing values by treating them all as a single valid category. CHAID does not perform pruning.

E. QUEST

Loh and Shih [28] have presented the quick, unbiased, efficient, statistical tree (QUEST) algorithm. QUEST supports univariate and linear combination splits. For each split, the association between each input attribute and the target attribute is computed using the ANOVA F-test or Levene's test (for ordinal and continuous attributes) or Pearson's chi-square (for nominal attributes). If the target attribute is multinomial, two-means clustering is used to create two super-classes. The attribute that obtains the highest association with the target attribute is selected for splitting. Quadratic discriminant analysis (QDA) is applied to find the optimal splitting point for the input attribute. QUEST has negligible bias and it yields binary decision trees. Ten-fold cross-validation is used to prune the trees.

F. Reference to Other Algorithms

Table I describes other decision trees algorithms available in the literature. Obviously, there are many other algorithms which are not included in this table. Nevertheless most of these algorithms are variation of the algorithmic framework presented above. A profound comparison of the above algorithms and many others has been conducted in [72].

TABLE I
ADDITIONAL DECISION TREES INDUCERS

Algorithm	Description	Reference
CAL5	Designed specifically for numerical-valued attributes	[74]
FACT	An earlier version of QUEST. Uses statistical tests to select an attribute for splitting each node and then uses discriminant analysis to find the split point.	[75]
LMDT	Constructs a decision tree based on multivariate tests that are linear combinations of the attributes.	[76]
T1	A one-level decision tree that classifies instances using only one attribute. Missing values are treated as a "special value". Support both continuous an nominal attributes.	[77]
PUBLIC	Integrates the growing and pruning by using MDL cost.	[78]
MARS	A multiple regression function is approximated using linear splines and their tensor products.	[79]

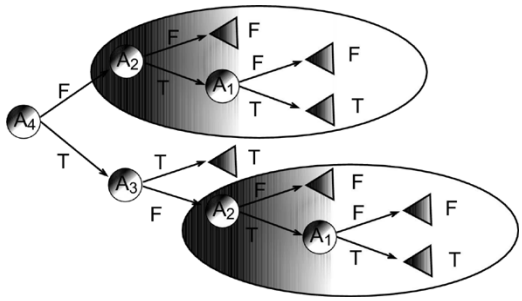

Fig. 3. Illustration of decision tree with replication.

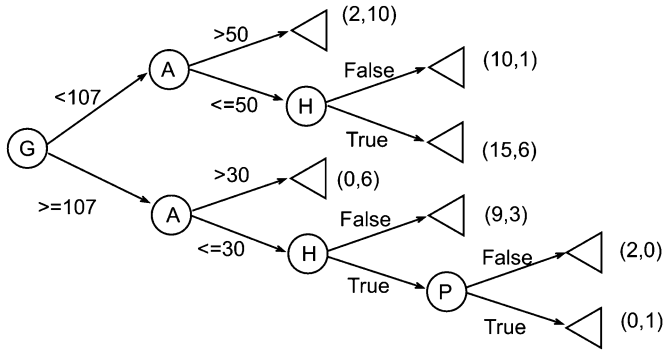

Fig. 4. Illustration of oblivious decision tree.

XI. ADVANTAGES AND DISADVANTAGES OF DECISION TREES

Several advantages of the decision tree as a classification tool have been pointed out in the literature.

- Decision trees are self-explanatory and when compacted they are also easy to follow. Furthermore decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible.
- Decision trees are capable to handle both nominal and numeric input attributes.
- Decision tree representation is rich enough to represent any discrete-value classifier.
- Decision trees are capable of handling datasets that may have errors.
- Decision trees are capable of handling datasets that may have missing values.
- Decision trees are considered to be a nonparametric method. This means that decision trees have no assumptions about the space distribution and on the classifier structure.

On the other hand decision trees have disadvantages such as the following.

- Most of the algorithms (like ID3 and C4.5) require that the target attribute will have only discrete values.
- As decision trees use "divide and conquer" method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present. One of the reasons for that is that other classifiers can compactly describe a classifer that would be very challenging to represent using a decision tree. A simple illustration of this phenomenon is the replication problem [53] of decision trees. Since most decision trees divide the instance space into mutually exclusive regions to represent a concept, in some cases the tree should contain several duplications of the same subtree in order to represent the classifier. For instance if the concept follows the following binary function: $y = (A_1 \cap A_2) \cup (A_3 \cap A_4)$ then the minimal univariate decision tree that represents this function is illustrated in Fig. 3. Note that the tree contains two copies of the same subtree.
- The greedy characteristic of decision trees leads to another disadvantage that should be point it. This is its over-sensitivity to the training set, to irrelevant attributes and to noise [12].

XII. SPECIAL CASES OF TOP-DOWN DECISION TREES INDUCTION

A. Oblivious Decision Trees

Oblivious decision trees are decision trees in which all nodes at the same level test the same attribute. Despite its restriction, oblivious decision trees are found to be effective as a feature selection procedure. Almuallim and Dietterich [54] as well as Schlimmer [55] have proposed forward feature selection procedure by constructing oblivious decision trees, whereas Langley and Sage [56] suggested backward selection using the same means. Kohavi and Sommer [57] have showed that oblivious decision trees can be converted to a decision table.

Recently, Last *et al.* [58] have suggested a new algorithm for constructing oblivious decision trees, called information fuzzy network IFN() that is based on information theory.

Fig. 4 illustrates a typical oblivious decision tree with four input features: glucose level (G), age (A), hypertension (h), and pregnant (p) and the Boolean target feature representing whether that patient suffers from diabetes. Each layer is uniquely associated with an input feature by representing the interaction of

that feature and the input features of the previous layers. The number that appears in the terminal nodes indicates the number of instances that fit this path. For example: regarding patients whose glucose level is less than 107 and their age is greater than 50, ten of them are positively diagnosed with diabetes while two of them are not diagnosed with diabetes.

The decision tree is built by a greedy algorithm, which tries to maximize the mutual information measure in every layer. The recursive search for explaining attributes is terminated when there is no attribute that explains the target with statistical significance.

B. Decision Trees Inducers for Large Datasets

With the recent growth in the amount of data collected by information systems there is a need for decision trees that can handle large datasets.

Catlett [59] has examined two methods for efficiently growing decision trees from a large database by reducing the computation complexity required for induction. However, the Catlett method requires that all data will be loaded into the main memory before induction. Namely, the largest dataset that can be induced is bounded by the memory size.

Fifield [60] suggests parallel implementation of the ID3 algorithm. However, like Catlett it assumes that all dataset can fit in the main memory.

Chan and Stolfo [61] suggest partitioning the datasets into several disjoin datasets, such that each dataset is loaded separately into the memory and used to induce a decision tree. The decision trees are then combined to create a single classifier. However, the experimental results indicate that partition may reduce the classification performance, meaning that the classification accuracy of the combined decision trees is not as good as the accuracy of a single decision tree induced from the entire dataset.

Mehta *et al.* [62] have proposed SLIQ an algorithm that does not require loading the entire dataset into the main memory, instead it uses secondary memory (disk) namely a certain instance is not necessarily resident in main memory all the time. SLIQ creates a single decision tree from the entire dataset. However, this method also has upper limit for the largest dataset that can be processed because it uses a data structure that scales with the dataset size and this data structure is required to be resident in main memory all the time.

Shafer *et al.* [63] have presented a similar solution called SPRINT. This algorithm induces decision trees relatively quickly and removes all of the memory restrictions from decision tree induction. SPRINT scales any impurity based split criteria for large datasets.

Gehrke *et al.* [64] introduced RainForest; a unifying framework for decision tree classifiers that are capable of scaling any specific algorithms from the literature (including C4.5, CART, and CHAID). In addition to its generality, RainForest improves SPRINT on a factor of three. In contrast to SPRINT, however, RainForest requires a certain minimum amount of main memory, proportional to the set of distinct values in a column of the input relation. However, this requirement is considered modest and reasonable.

Other decision tree inducers for large datasets can be found in the works of Alsabti *et al.* [65], Freitas and Lavington [66], and Gehrke *et al.* [67].

C. Incremental Induction

Most of the decision trees inducers require rebuilding the tree from scratch for reflecting new data that has became available. Several researches have addressed the issue of updating decision trees incrementally.

Utgoff [68], [69] presents several methods for updating decision trees incrementally. An extension to the CART algorithm that is capable of inducing incrementally is described in Crawford [70]).

XIII. Conclusion

This paper presented an updated survey of top-down decision trees induction algorithms. It has been shown that most algorithms fit into a simple algorithmic framework whereas the differences concentrate on the splitting criteria, stopping criteria and the way trees are pruned.

References

[1] S. R. Safavin and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 3, pp. 660–674, Jul. 1991.

[2] S. K. Murthy, "Automatic construction of decision trees from data: a multidisciplinary survey," *Data Mining Knowl. Disc.*, vol. 2, no. 4, pp. 345–389, 1998.

[3] R. Kohavi and J. R. Quinlan, "Decision-tree discovery," in *Handbook of Data Mining and Knowledge Discovery*, W. Klosgen and J. M. Zytkow, Eds. London, U.K.: Oxford Univ. Press, 2002, ch. 16.1.3, pp. 267–276.

[4] S. Grumbach and T. Milo, "Toward tractable algebras for bags," *J. Comp. Syst. Scie.*, vol. 52, no. 3, pp. 570–588, 1996.

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

[6] J. R. Quinlan, "Simplifying decision trees," *Int. J. Man-Mach. Studies*, vol. 27, pp. 221–234, 1987.

[7] T. R. Hancock, T. Jiang, M. Li, and J. Tromp, "Lower bounds on learning decision lists and trees," *Inform. Comput.*, vol. 126, no. 2, pp. 114–122, 1996.

[8] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Inform. Process. Lett.*, vol. 5, no. 1, pp. 15–17, 1976.

[9] H. Zantema and H. L. Bodlaender, "Finding small equivalent decision trees is hard," *Int. J. Found. Comput. Sci.*, vol. 11, no. 2, pp. 343–354, 2000.

[10] G. E. Naumov, "NP-completeness of problems of construction of optimal decision trees," *Sov. Phys.: Doklady*, vol. 36, no. 4, pp. 270–271, 1991.

[11] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.

[12] ——, *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann, 1993.

[13] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 2, pp. 163–174, Feb. 1991.

[14] F. Attneave, *Applications of Information Theory to Psychology*. New York: Holt, Rinehart and Winston, 1959.

[15] J. R. Quinlan, "Decision trees and multivalued attributes," in *Machine Intelligence*, J. Richards, Ed. London, U.K.: Oxford Univ. Press, 1988, vol. 11, pp. 305–318.

[16] R. Lopez de Mantras, "A distance-based attribute selection measure for decision tree induction," *Mach. Learn.*, vol. 6, pp. 81–92, 1991.

[17] U. M. Fayyad and K. B. Irani, "The attribute selection problem in decision tree generation," in *Proc. 10th Nat.Conf. Artificial Intelligence*, Cambridge, MA, 1992, pp. 104–110.

[18] J. H. Friedman, "A recursive partitioning decision rule for nonparametric classifiers," *IEEE Trans. Comput.*, vol. C26, no. 4, pp. 404–408, Apr. 1977.

[19] E. Rounds, "A combined nonparametric approach to feature selection and binary decision tree design," *Pattern Recognit.*, vol. 12, pp. 313–317, 1980.

[20] P. E. Utgoff and J. A. Clouse, "A Kolmogorov-Smirnoff metric for decision tree induction," Dept. Comp. Sci., Univ. Massachusetts, Amherst, Tech. Rep. no. 96-3.

[21] X. Li and R. C. Dubes, "Tree classifier design with a permutation statistic," *Pattern Recognit.*, vol. 19, pp. 229–235, 1986.

[22] P. C. Taylor and B. W. Silverman, "Block diagrams and splitting criteria for classification trees," *Statist. Computing*, vol. 3, no. 4, pp. 147–161, Dec. 1993.

[23] J. K. Martin, "An exact probability metric for decision tree splitting and stopping," *Mach. Learn.*, vol. 28, no. 2–3, pp. 257–291, 1997.

[24] E. Baker and A. K. Jain, "On feature ordering in practice and some finite sample effects," in *Proc. 3rd Int. Joint Conf. Pattern Recognition*, San Diego, CA, 1976, pp. 45–49.

[25] M. BenBassat, "Myopic policies in sequential classification," *IEEE Trans. Comput.*, vol. C–27, no. 2, pp. 170–174, Feb. 1978.

[26] J. Mingers, "An empirical comparison of pruning methods for decision tree induction," *Mach. Learn.*, vol. 4, no. 2, pp. 227–243, 1989.

[27] W. L. Buntine and T. Niblett, "A further comparison of splitting rules for decision-tree induction," *Mach. Learn.*, vol. 8, pp. 75–85, 1992.

[28] T. Loh and T. Shih, "Split selection methods for classification trees," *Statistica Sinica*, vol. 7, pp. 815–840, 1997.

[29] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif.Intell. Res.*, vol. 2, pp. 1–33, Aug. 1994.

[30] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[31] P. Bennett and O. L. Mangasarian, "Multicategory discrimination via linear programming," *Optimization Meth. Softw.*, vol. 3, pp. 29–39, 1994.

[32] J. Sklansky and G. N. Wassel, *Pattern Classifiers and Trainable Machines*. New York: Springer-Verlag, 1981.

[33] Y. K. Lin and K. Fu, "Automatic classification of cervical cells using a binary tree classifier," *Pattern Recognit.*, vol. 16, no. 1, pp. 69–80, 1983.

[34] W. Y. Loh and N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis," *J. Amer. Statist. Assoc.*, vol. 83, pp. 715–728, 1988.

[35] G. H. John, "Robust linear discriminant trees," in *Learning From Data: Artificial Intelligence and Statistics V*, D. Fisher and H. Lenz, Eds. New York: Springer-Verlag, 1996, ch. 36, pp. 375–385.

[36] P. E. Utgoff, "Perceptron trees: a case study in hybrid concept representations," *Connect. Sci.*, vol. 1, no. 4, pp. 377–391, 1989.

[37] D. Lubinsky, "Algorithmic speedups in growing classification trees by using an additive split criterion," in *Proc. AI Statistics*, 1993, pp. 435–444.

[38] I. K. Sethi and J. H. Yoo, "Design of multicategory, multifeature split decision trees using perceptron learning," *Pattern Recognit.*, vol. 27, no. 7, pp. 939–947, 1994.

[39] I. Bratko and M. Bohanec, "Trading accuracy for simplicity in decision trees," *Mach. Learn.*, vol. 15, pp. 223–250, 1994.

[40] T. Niblett and I. Bratko, "Learning decision rules in noisy domains," in *Expert Systems*. Cambridge, MA: Cambridge Univ. Press, 1986.

[41] H. Almuallim, "An efficient algorithm for optimal pruning of decision trees," *Artif. Intell.*, vol. 83, no. 2, pp. 347–362, 1996.

[42] J. Rissanen, *Stochastic Complexity and Statistical Inquiry*, Singapore: World Scientific, 1989.

[43] J. R. Quinlan and R. L. Rivest, "Inferring decision trees using the minimum description length principle," *Inform. Comput.*, vol. 80, pp. 227–248, 1989.

[44] R. L. Mehta, J. Rissanen, and R. Agrawal, *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining* , 1995, pp. 216–221.

[45] C. Wallace and J. Patrick, "Coding decision trees," *Mach. Learn.*, vol. 11, pp. 7–22, 1993.

[46] M. Kearns and Y. Mansour, "A fast, bottom-up decision tree pruning algorithm with near-optimal generalization," in *Proc. 15th Int. Conf. Machine Learning*, J. Shavlik, Ed., 1998, pp. 269–277.

[47] F. Esposito, D. Malerba, and G. Semeraro, "A comparative analysis of methods for pruning decision trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 5, pp. 476–492, May 1997.

[48] J. Quinlan, "Unknown attribute values in induction," in *Proc. 6th Int. Machine Learning Workshop*, A. Segre, Ed., Cornell, New York, 1989, pp. 164–168.

[49] A. Sonquist, E. L. Baker, and J. N. Morgan, "Searching for structure," Inst. Social Research, Univ. Michigan, Ann Arbor, MI, 1971.

[50] M. W. Gillo, "MAID: A Honeywell 600 program for an automatised survey analysis," *Behav. Sci.*, vol. 17, pp. 251–252, 1972.

[51] M. W. Morgan and R. C. Messenger, "THAID: A sequential search program for the analysis of nominal scale dependent variables," Inst. Social Research, Univ. Michigan, Ann Arbor, MI, 1973.

[52] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Appl. Statist.*, vol. 29, no. 2, pp. 119–127, 1980.

[53] G. Pagallo and D. Hassler, "Boolean feature discovery in empirical learning," *Mach. Learn.*, vol. 5, no. 1, pp. 71–100, 1990.

[54] H. Almuallim and T. G. Dietterich, "Learning Boolean concepts in the presence of many irrelevant features," *Artif. Intell.*, vol. 69, no. 1–2, pp. 279–306, 1994.

[55] J. C. Schlimmer, "Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning," in *Proc. Int. Conf. Machine Learning*, San Mateo, CA, 1993, pp. 284–290.

[56] P. Langley and S. Sage, "Oblivious decision trees and abstract cases," in *Proc. Working Notes of the AAAI-94 Workshop on Case-Based Reasoning*, Seattle, WA, 1994, pp. 113–117.

[57] R. Kohavi and D. Sommerfield, "Targeting business users with decision table classifiers," in *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining*, R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, Eds., 1998, pp. 249–253.

[58] M. Last, O. Maimon, and E. Minkov, "Improving stability of decision trees," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 16, no. 2, pp. 145–159, 2002.

[59] J. Catlett, "Mega Induction: Machine Learning on Vary Large Databases," Ph.D. thesis, Univ. Sydney, Sydney, Australia, 1991.

[60] D. J. Fifield, "Distributed Tree Construction From Large Datasets," B.S. honor thesis, Australian Nat. Univ., Canberra, Australia, 1992.

[61] P. Chan and S. Stolfo, "On the accuracy of meta-learning for scalable data mining," *J. Intell. Inform. Syst.*, vol. 8, pp. 5–28, 1997.

[62] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: a fast scalable classifier for data mining," in *Proc. 5th Int. Conf. Extending Database Technology (EDBT)*, Avignon, France, Mar. 1996, pp. 18–32.

[63] J. C. Shafer, R. Agrawal, and M. Mehta, "SPRINT: a scalable parallel classifier for data mining," in *Proc. 22nd Int. Conf. Very Large Databases*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds., 1996, pp. 544–555.

[64] J. Gehrke, R. Ramakrishnan, and V. Ganti, "RainForest—a framework for fast decision tree construction of large datasets," *Data Mining Knowl. Discov.*, vol. 4, no. 2/3, pp. 127–162, 2000.

[65] K. Alsabti, S. Ranka, and V. Singh, "CLOUDS: a decision tree classifier for large datasets," in *Proc. Conf. Knowledge Discovery and Data Mining (KDD-98)*, Aug. 1998, pp. 2–8.

[66] A. Freitas and S. H. Lavington, *Mining Very Large Databases With Parallel Processing*. Norwell, MA: Kluwer, 1998.

[67] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh, "BOAT-optimistic decision tree construction," in *Proc. SIGMOD Conf.*, 1999, pp. 169–180.

[68] P. E. Utgoff, "Incremental induction of decision trees," *Mach. Learn.*, vol. 4, pp. 161–186, 1989.

[69] ——, "Decision tree induction based on efficient tree restructuring," *Mach. Learn.*, vol. 29, no. 5, 1997.

[70] S. L. Crawford, "Extensions to the CART algorithm," *Int. J. Man-Mach. Stud.*, vol. 31, no. 2, pp. 197–217, Aug. 1989.

[71] S. L. Loh and S. L. Shih, "Families of splitting criteria for classification trees," *Statist. Comput.*, vol. 9, pp. 309–315, 1999.

[72] S. L. Lim, S. L. Loh, and S. L. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Mach. Learn.*, vol. 40, pp. 203–228, 2000.

[73] S. L. Shih, "Selecting the best splits classification trees with categorical variables," *Statist. Probability Lett.*, vol. 54, pp. 341–345, 2001.

[74] W. Muller and F. Wysotzki, "Automatic construction of decision trees for classification," *Ann. Oper. Res.*, vol. 52, pp. 231–247, 1994.

[75] W. Y. Loh and N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis," *J. Amer. Statist. Assoc.*, vol. 83, pp. 715–728, 1988.

[76] C. E. Brodley and P. E. Utgoff, "Multivariate decision trees," *Mach. Learn.*, vol. 19, pp. 45–77, 1995.

[77] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, pp. 63–90, 1993.

[78] R. Rastogi and K. Shim, "PUBLIC: a decision tree classifier that integrates building and pruning," *Data Mining Knowl. Discov.*, vol. 4, no. 4, pp. 315–344, 2000.

[79] J. H. Friedman, "Multivariate adaptive regression splines," *Annu. Statistics*, vol. 19, pp. 1–141, 1991.

Lior Rokach received the B.Sc., M.Sc. and Ph.D. degrees in industrial engineering from Tel-Aviv University, Tel-Aviv, Israel.

He is a recognized expert in business intelligence, and served in several leading positions in this field. His research interests include data mining, data warehousing and medical informatics. He has recently co-authored the book *Decomposition Methodology for Knowledge Discovery and Data Mining* (Singapore: World Scientific, 2005) and co-edited the handbook *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Research Scientists and Practitioners* (New York: Springer, 2005).

Oded Maimon received the B.Sc. degrees in industrial engineering and mechanical engineering and the M.Sc. degree in operations research, both from The Technion, Haifa, Israel, and the Ph.D. degree from Purdue University, West Lafayette, IN.

He is a Professor and former Chair of the Industrial Engineering Department, Tel-Aviv University. Before joining Tel-Aviv University, he was a Research Scientist at the Massachussetts Institute of Technology, Cambridge, and a Project Leader at Digital Equipment Corporation. He has recently co-authored the book *Decomposition Methodology for Knowledge Discovery and Data Mining* (Singapore: World Scientific, 2005) and co-edited the handbook *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Research Scientists and Practitioners* (New York: Springer, 2005).