

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/3421651>

Top-Down Induction of Decision Trees Classifiers—A Survey

Article in IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews) · December 2005

DOI: 10.1109/TSMCC.2004.843247 · Source: IEEE Xplore

CITATIONS

414

READS

1,091

2 authors:



Lior Rokach

Ben-Gurion University of the Negev

318 PUBLICATIONS 12,000 CITATIONS

[SEE PROFILE](#)



Oded Maimon

Tel Aviv University

234 PUBLICATIONS 6,108 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Price Sensitive Recommended Systems [View project](#)



Data mining applications in Cyber Security [View project](#)

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 top-down manner. The paper suggests a unified algorithmic framework for presenting these algorithms and provides profound descriptions of the various splitting criteria and pruning methodology.

Index Terms—Classification, Decision Trees, Splitting Criteria, Pruning Methods

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], [2], [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.

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 denote the set of input attributes containing n attributes: $A = \{a_1, \dots, a_i, \dots, a_n\}$ 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 $dom(a_i) = \{v_{i,1}, v_{i,2}, \dots, v_{i,|dom(a_i)|}\}$ its domain values, where $|dom(a_i)|$ stands for its finite cardinality. In a similar way, $dom(y) = \{c_1, \dots, c_{|dom(y)|}\}$ 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: $X = dom(a_1) \times dom(a_2) \times \dots \times dom(a_n)$. 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.: $U = X \times dom(y)$.

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 $S(R) = (\langle x_1, y_1 \rangle, \dots, \langle x_m, y_m \rangle)$ where $x_q \in X$ and $y_q \in dom(y)$.

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 $\{\text{True}, \text{False}\}$.

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 $A = \{a_1, a_2, \dots, a_n\}$ 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:

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

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

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

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 *internal node* or *test nodes*. All other nodes are called *leaves* (also known as *terminal nodes* or *decision nodes*).

In the 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.

Figure 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.

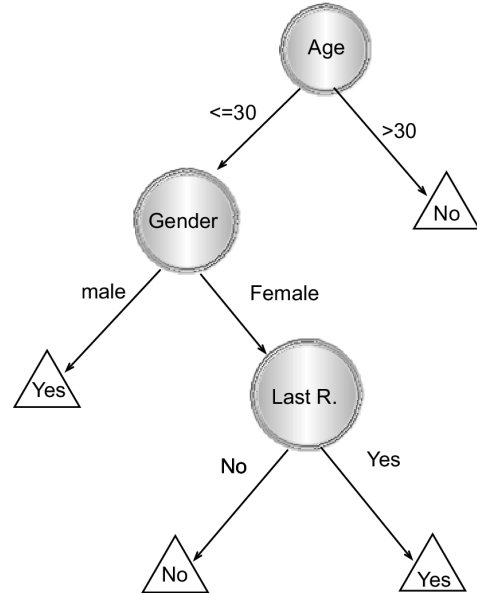


Fig. 1. Decision tree presenting response to direct mailing

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 tree, as it is considered more comprehensible. 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 Figure 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 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.

Figure 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], CART [5]. Some of which consists of two conceptual phases: Growing and Pruning (C4.5 and CART). Other inducers perform only the growing phase.

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.

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

- $\phi(P) \geq 0$

procedure *DTInducer*(S, A, y)

- 1: $T = \text{TreeGrowing}(S, A, y)$
- 2: **Return** $\text{TreePruning}(S, T)$

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, \dots, v_n) gains the best splitting metric.
- 6: **if** best splitting metric \geq threshold **then**
- 7: Label the root node in T as $f(A)$
- 8: **for** each outcome v_i of $f(A)$ **do**
- 9: $\text{Subtree}_i = \text{TreeGrowing}(\sigma_{f(A)=v_i} S, A, y)$.
- 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: $T = \text{pruned}(T, t)$
- 5: **end if**
- 6: **until** $t = \emptyset$
- 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)

- $\phi(P)$ 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:

$$P_y(S) = \left(\frac{|\sigma_{y=c_1} S|}{|S|}, \dots, \frac{|\sigma_{y=c_{|dom(y)|}} S|}{|S|} \right) \quad (3)$$

The goodness-of-split due to discrete attribute a_i is defined as reduction in impurity of the target attribute after partitioning S according to the values $v_{ij} \in \text{dom}(a_i)$:

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

Information Gain [6] is an Impurity Based Criteria that uses the entropy measure (origin from information theory) as the impurity measure.

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

Where:

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

C. Gini Index

Gini Index is an Impurity Based Criteria that measures 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:

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

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

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

D. Likelihood Ratio Chi-Squared Statistics

The likelihood-ratio is defined as [14]:

$$G^2(a_i, S) = 2 \cdot \ln(2) \cdot |S| \cdot InformationGain(a_i, S) \quad (8)$$

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: $(dom(a_i) - 1) \cdot (dom(y) - 1)$.

E. Normalized Impurity Based Criteria

The Impurity Based Criterion described above is biased towards 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 as following:

$$GainRatio(a_i, S) = \frac{InformationGain(a_i, S)}{Entropy(a_i, S)} \quad (9)$$

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:

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

Where:

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

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:

$$\beta^*(a_i, S) = \max \beta(a_i, d_1, d_2, S) \quad (11)$$

s.t.

$$\begin{aligned} d_1 \cup d_2 &= dom(a_i) \\ d_1 \cap d_2 &= \emptyset \end{aligned}$$

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:

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

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

J. Orthogonality Criterion

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

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

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 problems constellation.

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 $\text{dom}(y) = \{c_1, c_2\}$, the criterion is defined as:

$$\begin{aligned}
KS(a_i, d_1, d_2, S) = & \\
& \left| \frac{|\sigma_{a_i \in d_1 \text{ AND } y=c_1} S|}{|\sigma_{y=c_1} S|} - \frac{|\sigma_{a_i \in d_1 \text{ AND } y=c_2} S|}{|\sigma_{y=c_2} S|} \right|
\end{aligned} \quad (14)$$

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 [24], [25], [5], [26], [17], [27], [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 other, 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 greedy search [5], [29] linear programming [30], [31], linear discriminant analysis [30], [18], [32], [33], [34], [35] and others [36], [37], [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, \dots, T_k are 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.:

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

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 . $\text{pruned}(T, t)$ 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, \dots, 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 1-probability error rate estimation with and without pruning.

The 1-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:

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

Where $p_{apr}(y = c_i)$ 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:

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

However this correction still produces 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:

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

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.

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

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 $\text{subtree}(T, t)$ denote the sub tree rooted by the node t . Let $\text{maxchild}(T, t)$ 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:

$$\varepsilon_{UB}(\text{subtree}(T, t), S_t) \quad (20)$$

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

$$\varepsilon_{UB}(\text{subtree}(T, \text{maxchild}(T, t)), S_{\text{maxchild}(T, t)}) \quad (22)$$

According to the lowest value the procedure either leaves the tree as is, prune the node t , or replaces the node t with the sub tree rooted by $\text{maxchild}(T, t)$.

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 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 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:

$$\begin{aligned} \text{Cost}(t) = & \sum_{c_i \in \text{dom}(y)} |\sigma_{y=c_i} S_t| \cdot \ln \frac{|S_t|}{|\sigma_{y=c_i} S_t|} \\ & + \frac{|\text{dom}(y)| - 1}{2} \ln \frac{|S_t|}{2} + \\ & \ln \frac{\pi^{\frac{|\text{dom}(y)|}{2}}}{\Gamma(\frac{|\text{dom}(y)|}{2})} \end{aligned} \quad (23)$$

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 MML (minimum message length) 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 outperform 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:

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

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:

$$\begin{aligned} res(a_i, dom_1(a_i), dom_2(a_i), a_j, dom_1(a_j), dom_2(a_j), S) = \\ \frac{|\sigma_{a_i \in dom_1(a_i)} \text{ AND } a_j \in dom_1(a_j) S|}{|S|} \\ + \frac{|\sigma_{a_i \in dom_2(a_i)} \text{ AND } a_j \in dom_2(a_j) S|}{|S|} \quad (25) \end{aligned}$$

When the first split refers to attribute a_i and splits its domain to $dom_1(a_i)$ and $dom_2(a_i)$. The alternative split refers to attribute a_j and splits its domain to $dom_1(a_j)$ and $dom_2(a_j)$.

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 its mode over all instances having the same target attribute value. Formally,

$$est(a_i, y_q, S) = \arg \max_{v_{i,j} \in dom(a_i)} |\sigma_{a_i=v_{i,j}} \text{ AND } y=y_q S| \quad (26)$$

where y_q denote 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 procedure. It does not 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 is ceased 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 leaf 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]. CHAID (Chisquare-Automatic-Interaction-Detection) 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 QUEST (Quick, Unbiased, Efficient, Statistical Tree) 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 a 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 and 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]

- 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 to handle datasets that may have errors.
- Decision trees are capable to handle datasets that may have missing values.
- Decision trees are considered to be a nonparametric method; meaning decision trees have no assumptions on the space distribution and on the classifier structure.

On the other hand decision trees have disadvantages such as:

- 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 classifier 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 Figure 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].

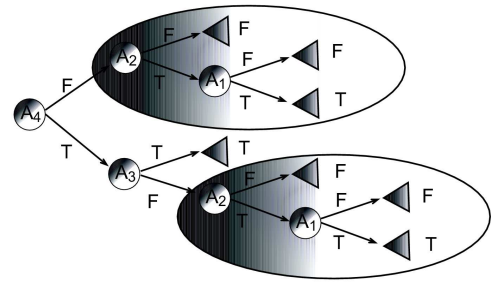


Fig. 3. Illustration of Decision Tree with Replication

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.

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 IFN (Information Fuzzy Network) that is based on information theory.

Figure ?? 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, 10 of them are positively diagnosed with diabetes while 2 of them 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.

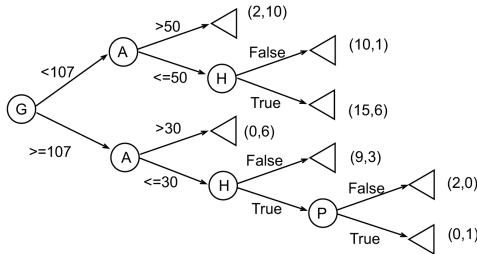


Fig. 4. Illustration of Oblivious Decision Tree

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 to partition the datasets into several disjoint 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 to scale 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 become 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 to induce 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. on Systems, Man and Cybernetics*, 21(3):660-674, 1991.
- [2] S. K. Murthy, Automatic Construction of Decision Trees from Data: A MultiDisciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4):345-389, 1998.
- [3] R. Kohavi and J. R. Quinlan. Decision-tree discovery. In Will Klogsen and Jan M. Zytow, editors, *Handbook of Data Mining and Knowledge Discovery*, chapter 16.1.3, pages 267-276. Oxford University Press, 2002.
- [4] S. Grumbach and T. Milo: Towards Tractable Algebras for Bags. *Journal of Computer and System Sciences* 52(3): 570-588, 1996.

- [5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth Int. Group, 1984.
- [6] J.R. Quinlan, Simplifying decision trees, *International Journal of Man-Machine Studies*, 27, 221-234, 1987.
- [7] T. R. Hancock, T. Jiang, M. Li, J. Tromp: Lower Bounds on Learning Decision Lists and Trees. *Information and Computation* 126(2): 114-122, 1996.
- [8] L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15-17, 1976
- [9] H. Zantema and H. L. Bodlaender, Finding Small Equivalent Decision Trees is Hard, *International Journal of Foundations of Computer Science*, 11(2):343-354, 2000.
- [10] G.E. Naumov. NP-completeness of problems of construction of optimal decision trees. *Soviet Physics: Doklady*, 36(4):270-271, 1991.
- [11] J.R. Quinlan, Induction of decision trees, *Machine Learning* 1, 81-106, 1986.
- [12] J. R. Quinlan, C4.5: Programs For Machine Learning. Morgan Kaufmann, Los Altos, 1993.
- [13] S. B. Gelfand, C. S. Ravishanker, and E. J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(2):163-174, 1991.
- [14] F. Attneave, Applications of Information Theory to Psychology. Holt, Rinehart and Winston, 1959.
- [15] J.R. Quinlan, Decision Trees and Multivalued Attributes, J. Richards, ed., *Machine Intelligence*, V. 11, Oxford, England, Oxford Univ. Press, pp. 305-318, 1988.
- [16] R. Lopez de Mantras, A distance-based attribute selection measure for decision tree induction, *Machine Learning* 6, 81-92, 1991.
- [17] U. M. Fayyad and K. B. Irani. The attribute selection problem in decision tree generation. In proceedings of Tenth National Conference on Artificial Intelligence, pages 104-110, Cambridge, 1992. MA: AAAI Press/MIT Press.
- [18] J. H. Friedman. A recursive partitioning decision rule for nonparametric classifiers. *IEEE Trans. on Comp.*, C26:404-408, 1977.
- [19] E. Rounds, A combined non-parametric approach to feature selection and binary decision tree design, *Pattern Recognition* 12, 313-317, 1980.
- [20] P. E. Utgoff and J. A. Clouse, A Kolmogorov-Smirnoff Metric for Decision Tree Induction, Technical Report 96-3, University of Massachusetts, Department of Computer Science, Amherst, MA
- [21] X. Li and R. C. Dubes, Tree classifier design with a Permutation statistic, *Pattern Recognition* vol. 19, 229-235, 1986.
- [22] P. C. Taylor and B. W. Silverman. Block diagrams and splitting criteria for classification trees. *Statistics and Computing*, 3(4):147-161, December 1993.
- [23] J. K. Martin. An exact probability metric for decision tree splitting and stopping. *An Exact Probability Metric for Decision Tree Splitting and Stopping*, *Machine Learning*, 28 (2-3):257-291, 1997.
- [24] E. Baker AND A. K. Jain. On feature ordering in practice and some finite sample effects. In *Proceedings of the Third International Joint Conference on Pattern Recognition*, pages 45-49, San Diego, CA, 1976.
- [25] M. BenBassat. Myopic policies in sequential classification. *IEEE Trans. on Computing*, 27(2):170-174, February 1978.
- [26] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227-243, 1989
- [27] W. L. Buntine, T. Niblett: A Further Comparison of Splitting Rules for Decision-Tree Induction. *Machine Learning*, 8: 75-85, 1992.
- [28] Loh and Shih, Split selection methods for classification trees. *Statistica Sinica*, 7: 815-840, 1997.
- [29] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1-33, August 1994.
- [30] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
- [31] Bennett and O.L. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3:29-39, 1994.
- [32] J. Sklansky and G. N. Wassel. *Pattern classifiers and trainable machines*. Springer-Verlag, New York, 1981.
- [33] Y. K. Lin and K. Fu. Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition*, 16(1):69-80, 1983.
- [34] W.Y. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant Analysis. *Journal of the American Statistical Association*, 83:715-728, 1988.
- [35] G. H. John. Robust linear discriminant trees. In D. Fisher and H. Lenz, editors, *Learning From Data: Artificial Intelligence and Statistics V*, Lecture Notes in Statistics, Chapter 36, pages 375-385. Springer-Verlag, New York, 1996.
- [36] Paul E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4):377-391, 1989.
- [37] D. Lubinsky. Algorithmic speedups in growing classification trees by using an additive split criterion. *Proc. AI&Statistics93*, pp. 435-444, 1993.
- [38] I. K. Sethi and J. H. Yoo. Design of multicategory, multifeature split decision trees using perceptron learning. *Pattern Recognition*, 27(7):939-947, 1994.
- [39] I. Bratko and M. Bohanec, Trading accuracy for simplicity in decision trees, *Machine Learning* 15, 223-250, 1994.
- [40] T. Niblett and I. Bratko, Learning Decision Rules in Noisy Domains, *Proc. Expert Systems 86*, Cambridge: Cambridge University Press, 1986.
- [41] H. Almuallim: An Efficient Algorithm for Optimal Pruning of Decision Trees. *Artificial Intelligence* 83(2): 347-362, 1996.
- [42] J Rissanen, Stochastic complexity and statistical inquiry. *World Scientific*, 1989.
- [43] J. R. Quinlan and R. L. Rivest. Inferring Decision Trees Using The Minimum Description Length Principle. *Information and Computation*, 80:227-248, 1989
- [44] Manish Mehta, Jorma Rissanen, Rakesh Agrawal: MDL-Based Decision Tree Pruning. *KDD 1995*: 216-221.
- [45] C. Wallace and J. Patrick, Coding decision trees, *Machine Learning* 11: 7-22, 1993.
- [46] M. Kearns and Y. Mansour, A fast, bottom-up decision tree pruning algorithm with near-optimal generalization, in J. Shavlik, ed., 'Machine Learning: Proceedings of the Fifteenth International Conference', Morgan Kaufmann Publishers, Inc., pp. 269-277, 1998.
- [47] F. Esposito, D. Malerba and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476-492, 1997.
- [48] J. Quinlan, Unknown attribute values in induction. In Segre, A. (Ed.), *Proceedings of the Sixth International Machine Learning Workshop* Cornell, New York. Morgan Kaufmann, 1989.
- [49] J. A. Sonquist, E. L. Baker, and J. N. Morgan. Searching for Structure. Institute for Social Research, Univ. of Michigan, Ann Arbor, MI, 1971.
- [50] M. W. Gillo, MAID: A Honeywell 600 program for an automatised survey analysis. *Behavioral Science* 17: 251-252, 1972.
- [51] J. N. Morgan and R. C. Messenger. THAID: a sequential search program for the analysis of nominal scale dependent variables. Technical report, Institute for Social Research, Univ. of Michigan, Ann Arbor, MI, 1973.
- [52] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119-127, 1980.
- [53] G. Pagallo and D. Hassler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 1990.
- [54] H. Almuallim and T.G. Dietterich, Learning Boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69: 1-2, 279-306, 1994.
- [55] Schlimmer, J. C. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the 1993 International Conference on Machine Learning*, pp 284-290, San Mateo, CA, Morgan Kaufman, 1993.
- [56] P. Langley and S. Sage, Oblivious decision trees and abstract cases. in *Working Notes of the AAAI-94 Workshop on Case-Based Reasoning*, pp 113-117, Seattle, WA: AAAI Press, 1994.
- [57] R. Kohavi and D. Sommerfield, Targeting business users with decision table classifiers, in R. Agrawal, P. Stolorz & G. Piatetsky-Shapiro, eds, 'Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining', AAAI Press, pp. 249-253, 1998.
- [58] M. Last, O. Maimon, and E. Minkov, Improving Stability of Decision Trees, *International Journal of Pattern Recognition and Artificial Intelligence*, 16: 2,145-159, 2002.
- [59] J. Catlett. Mega induction: Machine Learning on Vary Large Databases, PhD, University of Sydney, 1991.
- [60] D. J. Fifield. Distributed Tree Construction From Large Datasets, Bachelor's Honor Thesis, Australian National University, 1992.
- [61] P. Chan and S. Stolfo, On the Accuracy of Meta-learning for Scalable Data Mining, *J. Intelligent Information Systems*, 8:5-28, 1997.
- [62] M. Mehta, R. Agrawal and J. Rissanen. SLIQ: A fast scalable classifier for data mining: In *Proc. If the fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.
- [63] J. C. Shafer, R. Agrawal and M. Mehta, SPRINT: A Scalable Parallel Classifier for Data Mining, *Proc. 22nd Int. Conf. Very Large Databases*, T. M. Vijayaraman and Alejandro P. Buchmann and C. Mohan and Nandlal L. Sarda (eds), 544-555, Morgan Kaufmann, 1996.

- [64] J. Gehrke, R. Ramakrishnan, V. Ganti, RainForest - A Framework for Fast Decision Tree Construction of Large Datasets, *Data Mining and Knowledge Discovery*, 4 (2/3) 127-162, 2000.
- [65] K. Alsabti, S. Ranka and V. Singh, CLOUDS: A Decision Tree Classifier for Large Datasets, *Conference on Knowledge Discovery and Data Mining (KDD-98)*, August 1998.
- [66] A. Freitas and S. H. Lavington, *Mining Very Large Databases With Parallel Processing*, Kluwer Academic Publishers, 1998.
- [67] J. Gehrke, V. Ganti, R. Ramakrishnan, W. Loh: BOAT-Optimistic Decision Tree Construction. *SIGMOD Conference 1999*: pp. 169-180, 1999.
- [68] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161-186, 1989.
- [69] P. E. Utgoff, Decision tree induction based on efficient tree restructuring, *Machine Learning* 29 (5): 1997.
- [70] S. L. Crawford. Extensions to the CART algorithm. *Int. J. of Man-Machine Studies*, 31(2):197-217, August 1989.
- [71] Loh and Shih, Families of splitting criteria for classification trees. *Statistics and Computing* 1999, vol. 9, pp. 309-315.
- [72] Lim, Loh and Shih A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning* 2000, vol. 40, pp. 203-228.
- [73] Shih, Selecting the best splits classification trees with categorical variables. *Statistics and Probability Letters* 2001, vol. 54, pp. 341-345.
- [74] W. Muller and F. Wysotzki. Automatic construction of decision trees for classification. *Annals of Operations Research*, 52:231-247, 1994.
- [75] W. Y. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant Analysis. *Journal of the American Statistical Association*, 83:715-728, 1988.
- [76] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45-77, 1995
- [77] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63-90, 1993.
- [78] Rajeev Rastogi and Kyuseok Shim, PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning, *Data Mining and Knowledge Discovery*, 4(4):315-344, 2000.
- [79] J. H. Friedman, Multivariate Adaptive Regression Splines, *The Annual Of Statistics*, 19, 1-141, 1991.