Automatic Design of Decision-Tree Algorithms with Evolutionary Algorithms

Rodrigo C. Barros rcbarros@icmc.usp.br
Universidade de São Paulo, São Carlos, Brazil

Márcio P. Basgalupp basgalupp@unifesp.br
Universidade Federal de São Paulo, São José dos Campos, Brazil

André C. P. L. F. de Carvalho andre@icmc.usp.br
Universidade de São Paulo, São Carlos, Brazil

Alex A. Freitas A.A.Freitas@kent.ac.uk
University of Kent, Canterbury, UK

doi:10.1162/EVCO_a_00101

Abstract

This study reports the empirical analysis of a hyper-heuristic evolutionary algorithm that is capable of automatically designing top-down decision-tree induction algorithms. Top-down decision-tree algorithms are of great importance, considering their ability to provide an intuitive and accurate knowledge representation for classification problems. The automatic design of these algorithms seems timely, given the large literature accumulated over more than 40 years of research in the manual design of decision-tree induction algorithms. The proposed hyper-heuristic evolutionary algorithm, HEAD-DT, is extensively tested using 20 public UCI datasets and 10 microarray gene expression datasets. The algorithms automatically designed by HEAD-DT are compared with traditional decision-tree induction algorithms, such as C4.5 and CART. Experimental results show that HEAD-DT is capable of generating algorithms which are significantly more accurate than C4.5 and CART.

Keywords

Decision trees, hyper-heuristics, automatic algorithm design, supervised machine learning, data mining.

1 Introduction

Classification is a machine learning task that aims at building class prediction models, taking into account a set of predictive attributes (also known as features). The outcome of a classification model is the assignment of class labels to new instances whose only known information are the values of the predictive attributes (i.e., whose class labels are unknown). The set of instances whose class labels are known is named the training set—$\{\mathbf{x}^i, y^i\}_{i=1}^N$—where $\mathbf{x}^i$ is the ith vector of predictive attributes $\mathbf{x}^i = (x_1^i, x_2^i, x_3^i, \ldots, x_n^i)$ and y^i is the corresponding class label.

A classification algorithm usually operates in two steps. In the first step, the training set $\{\mathbf{x}^i, y^i\}_{i=1}^N$ is used as the input so that a classification model that represents the relationship between predictive attributes and class labels can be built. In the second step, the classification model is used to classify new instances whose class labels are unknown.

Manuscript received: July 30, 2012; revised: January 8, 2013; accepted: January 16, 2013.

Decision trees are one of the most employed classification methods to date. In fact, a relatively recent poll from the kdnuggets website[1] pointed out decision trees and decision rules as the most used classification methods by researchers and practitioners, reaffirming their importance in machine learning classification tasks. They are an efficient nonparametric method whereby the input space is split into local regions in order to predict the class labels (Alpaydin, 2010).

A decision tree can be seen as a graph $G = (V, E)$ consisting of a finite, nonempty set of nodes (vertices) v and a set of edges E. To be considered a decision tree, this graph must satisfy the following properties (Safavian and Landgrebe, 1991):

- The edges must be ordered pairs (v, w) of vertices, that is, the graph must be directed;
- There can be no cycles within the graph, that is, the graph must be acyclic;
- There is exactly one node, known as root, with no entering edges;
- Every node, except for the root, has exactly one entering edge;
- There is a unique path—a sequence of edges of the form $(v_1, v_2), (v_2, v_3), \ldots, (v_{l-1}, v_l)$—from the root to each node;
- When there is a path from node v to w, $v \neq w$, v is a proper ancestor of w and w is a proper descendant of v. A node with no proper descendant is called a leaf (or a terminal). All other nodes are called internal nodes (except for the root).

Root and internal nodes hold a test over a given predictive attribute (or a set of predictive attributes) from the dataset. The edges correspond to the possible outcomes of the test, eventually leading to the leaf nodes, which are responsible for holding class labels. For predicting the class label of a certain instance, one has to navigate through the decision tree. Starting from the root, one has to follow through the edges according to the results of the tests over the predictive attributes. When reaching a leaf node, the information it contains (class label) is usually assumed to be the prediction outcome.

As can be seen, decision trees are a natural alternative to powerful black-box methods, such as support vector machines (SVM) and neural networks (NNs), considering their comprehensible nature that resembles the human reasoning. Heuristic top-down decision-tree induction algorithms present several advantages over other learning algorithms, such as robustness to noise, low computational cost, and ability to deal with redundant attributes (Tan et al., 2005). In Figure 1, an example of a general decision tree for classification is presented. Circles denote the root and internal nodes, while boxes with round corners denote the leaf nodes.

The number of decision trees that can be grown from the same dataset increases exponentially with the number of attributes in the dataset. Induction of an optimal decision tree from a dataset is considered to be a hard task. For instance, Hyafil and Rivest (1976) have shown that constructing a minimal binary tree with regard to the expected number of tests required for classifying an unseen object is an NP-complete problem. Hancock et al. (1996) have proved that finding a minimal decision tree consistent with the training set is NP-Hard, which is also the case of finding the minimal equivalent

[1] http://www.kdnuggets.com/polls/2007/data_mining_methods.htm

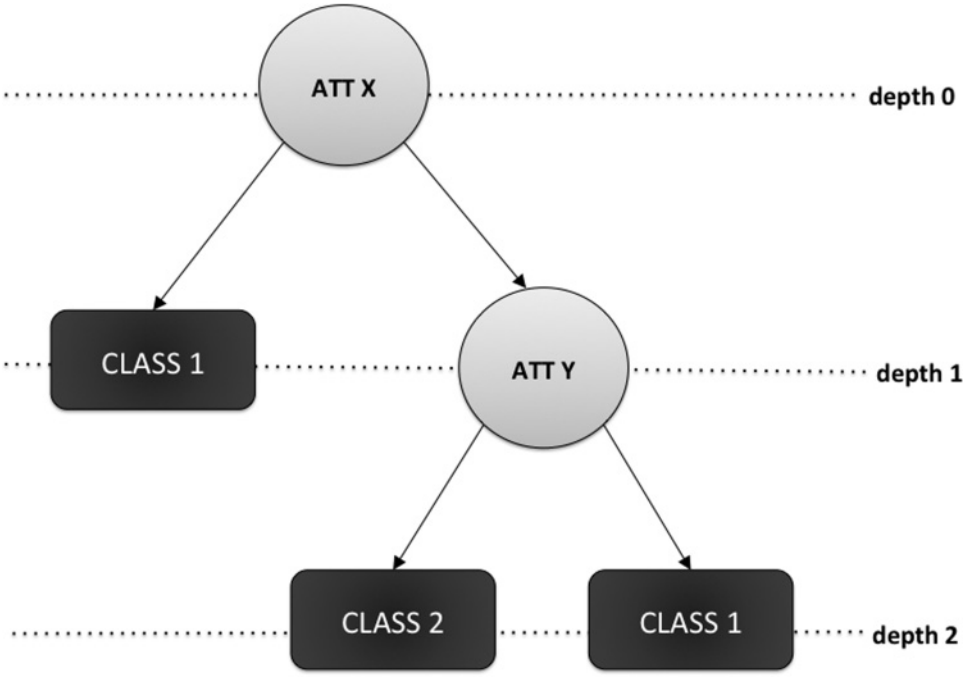

Figure 1: Example of a general decision tree for classification.

decision tree for a given decision tree (Zantema and Bodlaender, 2000) and building the optimal decision tree from decision tables (Naumov, 1991). These studies indicate that growing optimal decision trees (a brute-force approach) is only feasible for very simple problems.

Hence, heuristics are necessary for dealing with the problem of growing decision trees. Several approaches developed in the last decades are capable of providing reasonably accurate—albeit suboptimal—decision trees in a reduced amount of time, such as bottom-up induction (Barros, Cerri, et al., 2011), hybrid induction (Kim and Landgrebe, 1991), evolutionary induction (Basgalupp, Barros, et al., 2009; Basgalupp, de Carvalho, et al., 2009; Barros et al., 2010, 2012; Barros, Ruiz, et al., 2011), and ensemble of trees (Breiman, 2001). Notwithstanding, no strategy has been more successful in generating accurate and comprehensible decision trees with low computational effort than the greedy top-down induction strategy. Due to its popularity, a large number of approaches has been proposed for each one of the design components of top-down decision-tree induction algorithms. For instance, new measures for node splitting tailored to a vast number of application domains have been proposed, as well as many different strategies for selecting multiple attributes for composing the node rule. There are also studies in the literature that survey the numerous approaches for pruning a decision tree (Breslow and Aha, 1997; Esposito et al., 1997). It is clear that by improving these design components, we can obtain more robust top-down decision-tree induction algorithms.

Bearing in mind that the manual improvement of decision-tree design components has been carried out for the past 40 years, we believe that automatically designing decision-tree induction algorithms could provide a faster, less-tedious—and at least

equally effective—strategy for improving decision-tree induction algorithms in the years to come. This belief is partly supported by a proof of concept that evolutionary algorithms can effectively design new data mining algorithms that are competitive with conventional ones, as reported in Pappa and Freitas (2010). In that work, a grammar-based genetic programming system was investigated to automatically design a rule induction algorithm, which is another type of classification algorithm, and the automatically-designed algorithms were shown to be competitive with state of the art manually-designed rule induction algorithms.

In order to automatically design decision-tree induction algorithms, we propose a hyper-heuristic evolutionary algorithm called HEAD-DT. It evolves a set of design components of top-down decision-tree induction algorithms, thus every individual in the evolutionary algorithm is a full decision-tree induction algorithm, and not just a decision tree. We first envisioned the problem of automatically designing decision-tree induction algorithms in Barros, Basgalupp, et al. (2011). In this new study, we investigate the effectiveness of automatically designing decision-tree induction algorithms in both benchmarking and bioinformatics data. This paper is an extended version of a previous conference paper (Barros et al., 2012a), providing a more detailed description of the proposed approach and also presenting new computational results and their analysis for 10 real-world bioinformatics (gene expression) datasets.

This paper is organized as follows. Section 2 offers a background on decision trees for readers unfamiliar with the topic. Section 3 presents in detail the HEAD-DT algorithm. Sections 4 and 5 describe the experimental methodology and the obtained results, respectively, whereas Section 6 discusses relevant aspects regarding the effectiveness and efficiency of HEAD-DT. Section 7 presents related work and Section 8 concludes the paper with our final thoughts and future work suggestions.

2 Background on Decision Trees

Automatically generating decision trees has been the object of study of most research fields in which data exploration techniques have been developed (Murthy, 1998). Disciplines such as engineering (pattern recognition), statistics, decision theory, and more recently artificial intelligence (machine learning), have a large number of studies dedicated to the generation and application of decision trees. In statistics, we can trace the origins of decision trees to works that proposed building binary segmentation trees for understanding the relationship between predictors and dependent variables. Some examples are AID (Sonquist et al., 1971) and CHAID (Kass, 1980). Decision-tree induction algorithms, and induction methods in general, arose in machine learning to avoid the knowledge acquisition bottleneck in expert systems (Murthy, 1998).

Specifically regarding top-down induction of decision trees (by far the most popular approach of decision-tree induction), Hunt's concept learning system (CLS; Hunt et al., 1966) can be regarded as the pioneering work for inducing decision trees. Systems that directly descend from Hunt's CLS are ID3 (Quinlan, 1986), ACLS (Patterson and Niblett, 1983), and Assistant (Kononenko et al., 1984).

At a high level of abstraction, Hunt's algorithm can be recursively defined in only two steps. Let $\mathbf{X_t}$ be the set of training instances associated with node t and $y = \{y_1, y_2, \ldots, y_k\}$ be the class labels in a k-class problem (Tan et al., 2005):

1. If all instances in $\mathbf{X_t}$ belong to the same class y_i then t is a leaf node labeled as y_i.
2. If $\mathbf{X_t}$ contains instances that belong to more than one class, an attribute test condition is selected to partition the instances into subsets. A child node is

created for each outcome of the test and the instances in $\mathbf{X_t}$ are distributed to the children based on the outcomes. Recursively apply the algorithm to each child.

Hunt's simplified algorithm is the basis for most current top-down decision-tree induction algorithms. Nevertheless, its assumptions are too stringent for practical use. For instance, it would only work if every combination of attribute values is present in the training data, and if the training data are inconsistency-free (each combination has a unique class label).

Hunt's algorithm was improved in many ways. Its stopping criterion, for example, as expressed in step 1, requires all leaf nodes to be pure (i.e., belonging to the same class). In most practical cases, this constraint leads to very large decision trees, which tend to suffer from overfitting. Possible solutions to overcome this problem are prematurely stopping the tree growth when a minimum level of impurity is reached, or performing a pruning step after the tree has been fully grown. Another design issue is how to select the attribute test condition to partition the instances into smaller subsets. In Hunt's original approach, a cost-driven function was responsible for partitioning the tree. Subsequent algorithms, such as ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993), make use of information theory based functions for partitioning nodes in purer subsets.

These improvements were achieved by human investigation of several design alternatives. We believe that evolutionary algorithms (EA) are capable of evolving new efficient decision-tree induction algorithms by automatically finding a good combination of different heuristics, many of them present in known induction algorithms. In the next section, we present an EA designed to evolve the design components of decision-tree induction algorithms, HEAD-DT.

3 HEAD-DT

HEAD-DT is a hyper-heuristic algorithm able to automatically design top-down decision-tree induction algorithms. Hyper-heuristics can automatically generate new heuristics (or algorithms) suited to a given problem or class of problems. This is accomplished by combining, through an EA, components or building-blocks of human designed heuristics (Burke et al., 2009, 2010).

HEAD-DT is a regular generational EA in which individuals are collections of building blocks (heuristics) from decision-tree induction algorithms. In Figure 2, we present its evolutionary scheme.

Each individual in HEAD-DT is encoded as an integer string (see Figure 3), and each gene has a different range of supported values. We divided the genes into four categories that represent the major building blocks (design components) of a decision-tree induction algorithm: (1) split genes; (2) stopping criteria genes; (3) missing values genes; and (4) pruning genes. We detail each category next.

3.1 Split Genes

These genes are concerned with the task of selecting the predictive attribute to split the data in the current node of the decision tree. A decision rule based on the selected attribute is thus generated, and the input data are filtered according to the outcomes of this rule, and the process continues recursively. HEAD-DT uses two genes to model the split component of a decision-tree algorithm: split criterion and split type.

3.1.1 Split Criterion Gene

The first gene, split criterion, is an integer that indexes one of the 15 splitting criteria implemented: information gain (Quinlan, 1986), Gini index (Breiman et al., 1984), global

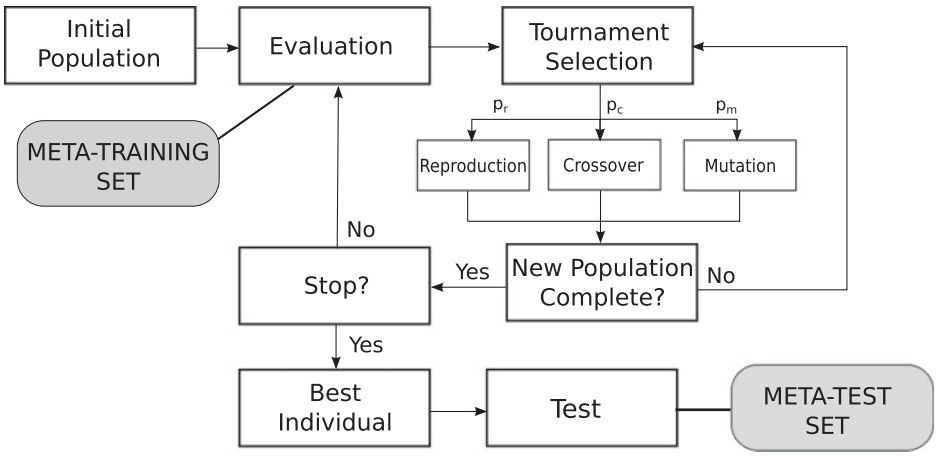

Figure 2: HEAD-DT evolutionary scheme.

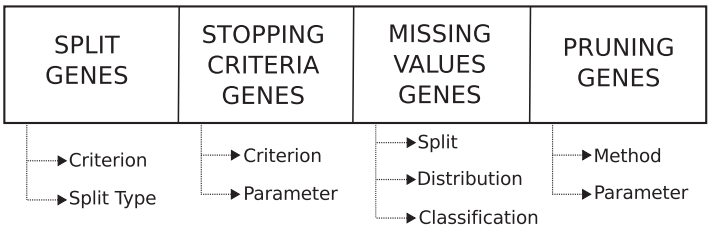

Figure 3: Linear genome for evolving decision-tree induction algorithms.

mutual information (Gleser and Collen, 1972), G statistics (Mingers, 1987), Mantaras criterion (De Mántaras, 1991), hypergeometric distribution (Martin, 1997), Chandra-Varghese criterion (Chandra and Varghese, 2009), DCSM (Chandra et al., 2010), χ^2 (Mingers, 1989), mean posterior improvement (Taylor and Silverman, 1993), normalized gain (Jun et al., 1997), orthogonal criterion (Fayyad and Irani, 1992), twoing (Breiman et al., 1984), CAIR (Ching et al., 1995), and gain ratio (Quinlan, 1993).

The most well-known univariate criteria are based on Shannon's entropy (Shannon, 1948). Entropy is known to be a unique function that satisfies the four axioms of uncertainty. It represents the average amount of information when coding each class into a code word with ideal length according to its probability. The first splitting criterion based on entropy was the global mutual information (GMI; Gleser and Collen, 1972). Afterward, the well-known information gain (Quinlan, 1986) became a standard after appearing in algorithms like ID3 (Quinlan, 1986) and Assistant (Kononenko et al., 1984). It belongs to the class of the so-called impurity-based criteria. Quinlan (1986) acknowledged the fact that the information gain is biased toward attributes with many values. To deal with this problem, he proposed the gain ratio (Quinlan, 1993), which basically normalizes the information gain by the entropy of the attribute being tested. Several variations of the gain ratio were proposed, for example, the normalized gain (Jun et al., 1997). Alternatives to entropy-based criteria are the class of distance-based measures, that is, criteria that evaluate separability, divergency, or discrimination

between classes. Examples are the Gini index (Breiman et al., 1984), the twoing criterion (Breiman et al., 1984), and the orthogonality criterion (Fayyad and Irani, 1992), among others. We also included lesser-known criteria, such as CAIR (Ching et al., 1995) and mean posterior improvement (Taylor and Silverman, 1993), as well as the more recent Chandra-Varghese (Chandra and Varghese, 2009) and DCSCM (Chandra et al., 2010), to enhance the diversity of options for generating splits in a decision tree.

3.1.2 Split Type Gene

The second gene, split type, is a binary gene that indicates whether the splits of a decision tree are going to be necessarily binary or perhaps multi-way. In a binary tree, every split has only two outcomes, which means that nominal attributes with many categories will be divided into two subsets, each representing an aggregation over several categories. In a multi-way tree, nominal attributes are divided according to their number of categories, that is, one edge (outcome) for each category. In both cases, numeric attributes always partition the tree into two subsets, ($\leq$threshold and $>$threshold).

3.2 Stopping Criteria Genes

The second category of genes is related to the stopping criteria component of decision-tree induction algorithms. The top-down induction of a decision tree is recursive and it continues until a stopping criterion (also known as pre-pruning) is satisfied. HEAD-DT uses two genes to model the stopping criteria component of decision trees: criterion and parameter.

3.2.1 Stopping Criterion Gene

The first gene, stopping criterion, selects among the five following different strategies for stopping the tree growth:

- Reaching class homogeneity: when all instances that reach a given node belong to the same class, there is no reason to split this node any further. This criterion can be combined with any of the following criteria;
- Reaching the maximum tree depth: a parameter tree depth can be specified to avoid deep trees. Range: [2, 10] levels;
- Reaching the minimum number of instances for a non-terminal node: a parameter of the minimum number of instances for a non-terminal node can be specified to avoid (or at least alleviate) the data fragmentation problem in decision trees. Range: [1, 20] instances;
- Reaching the minimum percentage of instances for a non-terminal node: same as above, but instead of the actual number of instances, we set the minimum percentage of instances. The parameter is thus relative to the total number of instances in a dataset. Range: [1%, 10%] the total number of instances;
- Reaching a predictive accuracy threshold within a node: a parameter of the accuracy reached can be specified for halting the growth of the tree when the predictive accuracy within a node (majority of instances) has reached a given threshold. Range: {70%, 75%, 80%, 85%, 90%, 95%, 99%}.

3.2.2 Stopping Parameter Gene

The second gene, stopping parameter, dynamically adjusts a value in the range $[0, 100]$ to the corresponding strategy. For instance, if the strategy selected by the gene stopping criterion is reaching a predictive accuracy threshold within a node, the following mapping function is executed:

$$param = ((value \bmod 7) \times 5) + 70 \tag{1}$$

This function maps from $[0, 100]$ to $\{70, 75, 80, 85, 90, 95, 100\}$, which is almost what was defined as the range of the strategy of reaching a predictive accuracy threshold. The final step subtracts 1 from the resulting parameter value if it is equal to 100. Similar mapping functions are executed dynamically to adjust the ranges of the gene stopping parameter.

3.3 Missing Values Genes

Handling missing values is an important task in decision-tree induction. Missing values can harm the tree induction process and also its use for the classification of new examples. During the tree induction, there are two moments in which we need to deal with missing values: splitting criterion evaluation and instances distribution. To deal with these scenarios, HEAD-DT uses three genes to model the missing values component of decision trees: mv split, mv distribution, and mv classification.

3.3.1 mv Split Gene

During the split criterion evaluation for a node t based on an attribute a_i, we implemented the following strategies: (1) ignore all instances whose value of a_i is missing (Friedman, 1977; Breiman et al., 1984); (2) imputation of missing values with either the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t (Clark and Niblett, 1989); (3) weight the splitting criterion value (calculated in node t with regard to a_i) by the proportion of missing values (Quinlan, 1989); and (4) imputation of missing values with either the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t whose class attribute is the same as the instance whose a_i value is being imputed (Loh and Shih, 1997).

3.3.2 mv Distribution Gene

For deciding which child node a training instance x_j should go to, considering a split for node t over a_i, we adopted the following options: (1) ignore instance x_j (Quinlan, 1986); (2) treat instance x_j as if it has the most common value of a_i (mode or mean), regardless of its class (Quinlan, 1989); (3) treat instance x_j as if it has the most common value of a_i (mode or mean) considering the instances that belong to the same class as x_j; (4) assign instance x_j to all partitions (Friedman, 1977); (5) assign instance x_j to the partition with the largest number of instances (Quinlan, 1989); (6) weight instance x_j according to the partition probability (Quinlan, 1993; Kononenko et al., 1984); and (7) assign instance x_j to the most probable partition, considering the class of x_j (Loh and Shih, 1997).

3.3.3 mv Classification Gene

For classifying an unseen test instance x_j, considering a split in node t over a_i, we used the strategies: (1) explore all branches of t combining the results (Quinlan, 1987a); (2) take the route to the most probable partition (largest subset); (3) halt the classification process and assign the instance x_j to the majority class of node t (Quinlan, 1989).

3.4 Pruning Genes

Pruning is usually performed in decision trees to enhance tree comprehensibility by reducing its size while maintaining (or even improving) its predictive accuracy. It was originally conceived as a strategy for tolerating noisy data, although it was found to improve decision tree predictive accuracy in many noisy datasets (Breiman et al., 1984; Quinlan, 1986, 1987b). We designed two genes in HEAD-DT for pruning: method and parameter.

3.4.1 Pruning Method Gene

The first gene, pruning method, indexes one of the following five approaches for pruning a decision tree (and also the option of not pruning at all):

- Reduced-error pruning (REP) is a conceptually simple strategy proposed by Quinlan (1987b). It uses a pruning set (part of the training set) to evaluate the goodness of a given subtree from T. The idea is to evaluate each non-terminal node t regarding the classification error in the pruning set. If this error decreases when we replace the subtree $T^{(t)}$ rooted on t by a leaf node, then $T^{(t)}$ must be pruned. Quinlan also imposes a constraint: a node t cannot be pruned if it contains a subtree that yields a lower classification error in the pruning set. The practical consequence of this constraint is that REP should be performed in a bottom-up fashion. The REP pruned tree T' presents an interesting optimality property: it is the smallest most accurate tree resulting from pruning original tree T (Quinlan, 1987b). Apart from this optimality property, another advantage of REP is its linear complexity, since each node is visited only once in T. An obvious disadvantage is the need for a pruning set, which means one has to divide the original training set, resulting in fewer instances to grow the tree. This disadvantage is particularly serious for small datasets.

- Also proposed by Quinlan (1987b), pessimistic error pruning (PEP) uses the training set for both growing and pruning the tree. The apparent error rate, that is, the error rate calculated for the training set, is optimistically biased and cannot be used to decide whether pruning should be performed or not. Quinlan proposes adjusting the apparent error according to the continuity correction for the binomial distribution in order to provide a more realistic error rate. PEP is computed in a top-down fashion, and if a given node t is pruned, its descendants are not examined, which makes this pruning strategy very computationally efficient. However, Esposito et al. (1997) point out that the introduction of the continuity correction in the estimation of the error rate has no theoretical justification, since it was never applied to correct over-optimistic estimates of error rates in statistics.

- Originally proposed in Niblett and Bratko (1986) and further extended in Cestnik and Bratko (1991), minimum error pruning (MEP) is a bottom-up approach that seeks to minimize the expected error rate for unseen cases. It uses an ad hoc parameter m for controlling the level of pruning. Usually, the higher the value of m, the more severe the pruning. Cestnik and Bratko (1991) suggest that a domain expert should set m according to the level of noise in the data. Alternately, a set of trees pruned with different values of m could be offered to the domain expert, so he or she can choose the best one according to his or her experience.

- Cost-complexity pruning (CCP) is the post-pruning strategy of the CART system, detailed in Breiman et al. (1984). It has two steps: (1) generate a sequence of increasingly smaller trees, beginning with T and finishing with the root node of T, by successively pruning the subtree yielding the lowest cost complexity, in a bottom-up fashion; and (2) choose the best tree among the sequence based on its relative size and predictive accuracy (either on a pruning set, or provided by a cross-validation procedure in the training set). The idea behind step 1 is that the pruned tree T_{i+1} is obtained by pruning the subtrees that show the lowest increase in the apparent error (i.e., error in the training set) per pruned leaf. Regarding step 2, CCP chooses the smallest tree whose error (either on the pruning set or on cross-validation) is not higher than one standard error (SE) larger than the lowest error observed in the sequence of trees.

- Finally, error-based pruning (EBP), proposed by Quinlan and implemented as the default pruning strategy of C4.5 (Quinlan, 1993), is an improvement over PEP. It is based on a far more pessimistic estimate of the expected error. Unlike PEP, EBP performs a bottom-up search, and it carries out not only the replacement of non-terminal nodes by leaves but also grafting of subtree $T^{(t)}$ onto the place of parent t. For deciding whether to replace a non-terminal node by a leaf (subtree replacement), to graft a subtree onto the place of its parent (subtree raising) or not to prune at all, a pessimistic estimate of the expected error is calculated by using an upper confidence bound. An advantage of EBP is the new grafting operation that allows pruning useless branches without ignoring interesting lower branches. A drawback of the method is the presence of an ad hoc parameter, *CF*. Smaller values of *CF* result in more pruning.

3.4.2 Pruning Parameter Gene

The second gene, pruning parameter, is in the range [0, 100] and its value is dynamically mapped by a function, according to the pruning method selected (similar to the stopping parameter gene). For REP, the parameter is the percentage of training data to be used in the pruning set (varying within the interval [10%, 50%]). For PEP, the parameter is the number SEs to adjust the apparent error in the set {0.5, 1, 1.5, 2}. For MEP, the parameter m may range within [0, 100]. For CCP, there are two parameters: the number of SEs (in the same range as PEP) and the pruning set size (in the same range as REP). Finally, for EBP, the parameter *CF* may vary within the interval [1%, 50%].

3.5 Fitness Evaluation

During the fitness evaluation, HEAD-DT employs a meta-training set for assessing the quality of each individual throughout evolution. It also employes a meta-test set, which is used to assess the quality of the evolved decision-tree induction algorithm (the best individual in Figure 2). There are two distinct approaches for dealing with the meta-training and meta-test sets:

1. Evolving a decision-tree induction algorithm tailored to one specific dataset.
2. Evolving a decision-tree induction algorithm from multiple datasets.

In the first case (see Figure 4), we have a specific dataset for which we want to design a decision-tree induction algorithm. The meta-training set is thus composed of the training data we have available for the dataset at hand, and the meta-test set

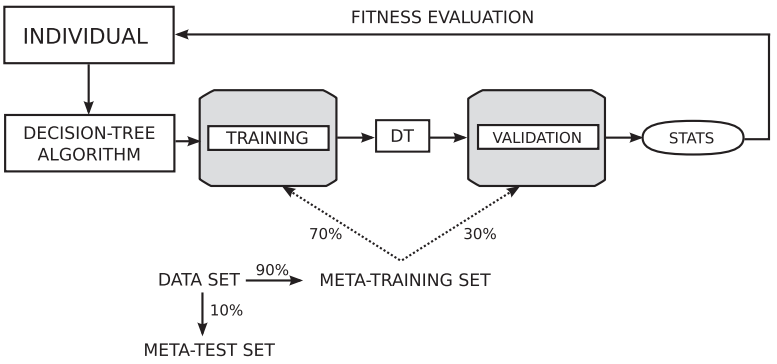

Figure 4: Fitness evaluation from one dataset in the meta-training set.

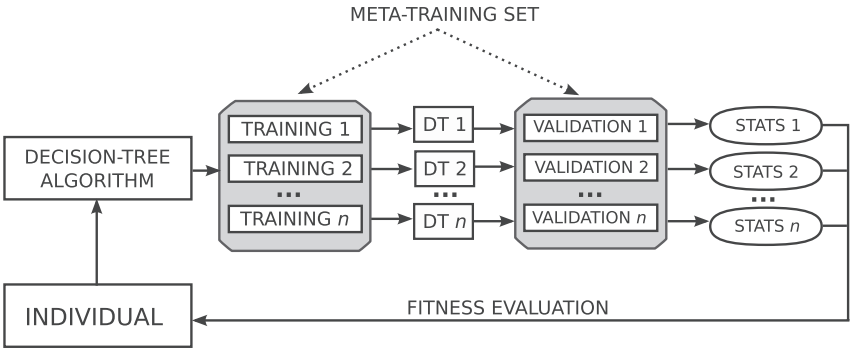

Figure 5: Fitness evaluation from multiple datasets in the meta-training set.

is comprised of the test data (belonging to the same dataset) we have available for evaluating the performance of the algorithm.

In the second case (see Figure 5), we have multiple datasets comprising the meta-training set, and possibly multiple (but different) datasets comprising the meta-test set. This strategy can be employed with two different objectives: (1) designing a decision-tree induction algorithm that performs reasonably well for a wide variety of datasets (i.e., datasets with very different structural characteristics and/or from very distinct application domains, e.g., finance, medicine, bioinformatics, etc.); and (2) designing a decision-tree induction algorithm that is tailored to a particular application domain or to a specific statistical profile. In both cases, the main goal is to generate induction algorithms tailored to a set of datasets, which may be from the same domain or not.

In this paper, we focus on the first fitness approach—generating one decision-tree induction algorithm per dataset. The second approach is a topic left for future research. In the employed approach, we first divide the dataset into 90% for the meta-training set and 10% for the meta-test set (one of the 10-fold cross-validation splits), in a stratified fashion. Next, the meta-training set is further divided into two sets: a training set (70% of the meta-training set) and a validation set (30% of the meta-training set). The training set is thus used to build the decision tree for each individual, and the validation set is used for evaluating the performance of each individual (through a fitness function) at each

generation. Once the evolution is over, we evaluate the best individual (decision-tree algorithm) by building the decision tree from the full meta-training data and evaluating it over the meta-test data. The results achieved in the meta-test set correspond to a single split of the 10-fold cross-validation procedure, and therefore the remaining nine splits must be executed. Results provided by the 10 different splits of the cross-validation procedure are then averaged to generate the final result for each dataset.

As a fitness function, we investigate both predictive accuracy and F-measure (harmonic mean of precision and recall; Tan et al., 2005). These performance measures can be defined in terms of four variables: true positives (tp), true negatives (tn), false positives (fp), and false negatives (fn). True positives (negatives) are the number of instances that were correctly predicted as belonging to the positive (negative) class, whereas false positives (negatives) are the number of instances that were incorrectly predicted as belonging to the positive (negative) class. With these four variables, we can define accuracy, precision, recall, and F-measure, respectively, as:

$$\text{accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \tag{2}$$

$$\text{precision} = \frac{tp}{tp + fp} \tag{3}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{4}$$

$$\text{F-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{5}$$

Note that this formulation assumes that the classification problem at hand is binary, that is, composed of two classes: positive and negative. Nevertheless, it can be trivially extended to account for multi-class problems. For instance, one can compute the measure for each class—assuming each class to be the positive class in turn—and (weight-) average the per-class measures. A discussion about the pros and cons of accuracy and F-measure as predictive performance measures will be presented later.

3.6 Evolutionary Cycle

The evolution of individuals in HEAD-DT follows the scheme presented in Figure 2. The nine-gene linear genome of an individual in HEAD-DT is composed of the building blocks described in the earlier sections: [split criterion, split type, stopping criterion, stopping parameter, pruning strategy, pruning parameter, mv split, mv distribution, mv classification], where mv stands for missing value.

The first step of HEAD-DT is the generation of the initial population, in which a population of n individuals is randomly generated (random number generation within the gene's acceptable range of values). The individuals from the current population participate in a pairwise tournament selection procedure for defining those that will undergo genetic operators. Individuals may participate in either uniform crossover, random uniform gene mutation, or reproduction, the three mutually-exclusive genetic operators employed in HEAD-DT. In addition, HEAD-DT employs an elitism strategy, in which the best i% individuals are retained from one generation to the next. Note that these operators are the conventional operators used in the genetic algorithms literature.

One possible individual encoded by the nine-gene linear string representation is [4, 1, 2, 77, 3, 91, 2, 5, 1], which accounts for Algorithm 1.

Algorithm 1 Example of an algorithm automatically designed by HEAD-DT

```
1) Recursively split nodes with the G statistics criterion;
2) Create one edge for each category in a nominal split;
3) Perform step 1 until class-homogeneity or the maximum tree depth of 7 levels
   ((77 mod 9) + 2) is reached;
4) Perform MEP pruning with m = 91;
5) When dealing with missing values:
     5.1) Impute missing values with mode/mean during split calculation;
     5.2) Distribute missing-valued instances to the partition with the largest
          number of instances;
     5.3) For classifying an instance with missing values, explore all branches
          and combine the results.
```


4 Experimental Methodology

4.1 Datasets

To assess the relative performance of the algorithms automatically designed by HEAD-DT, we employ well-known UCI public datasets (Frank and Asuncion, 2010). In addition, we test the effectiveness of the automatically-generated algorithms tailored to each of 10 real-world gene expression datasets (de Souto et al., 2008). The 30 datasets used for testing the effectiveness of HEAD-DT are summarized in Table 1, where the fifth and sixth columns describe the number of numerical and nominal attributes, respectively. The other column names are self-explanatory.

4.2 Baseline Algorithms and Statistical Analysis

We compare the resulting decision-tree induction algorithms with the most well-known and widely-used decision-tree induction algorithms to date: C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984). Since HEAD-DT is a nondeterministic method, its results are averaged over five different runs (varying the random seed in each run).

In order to provide some reassurance about the validity and nonrandomness of the obtained results, we present the results of statistical tests by following the approach proposed by Demšar (2006). In brief, this approach seeks to compare multiple algorithms on multiple datasets, and it is based on the use of the Friedman test with a corresponding post hoc test. The Friedman test is a nonparametric counterpart of the well-known ANOVA, as follows. Let R_i^j be the rank of the jth of k algorithms on the ith of N datasets. The Friedman test compares the average ranks of algorithms, $R_j = \frac{1}{N}\sum_i R_i^j$. The Friedman statistic, given by:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{6}$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are large enough.

Iman and Davenport (1980) showed that Friedman's χ_F^2 is undesirably conservative and derived an adjusted statistic:

$$F_f = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2} \tag{7}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

Table 1: Summary of the 30 datasets used in the experiments.

Source	Dataset	# instances	# attributes	# numeric	# nominal	% missing	# classes
UCI	abalone	4,177	8	7	1	0.00	30
	anneal	898	38	6	32	0.00	6
	arrhythmia	452	279	206	73	0.32	16
	audiology	226	69	0	69	2.03	24
	bridges_version1	107	12	3	9	5.53	6
	car	1,728	6	0	6	0.00	4
	cylinder_bands	540	39	18	21	4.74	2
	glass	214	9	9	0	0.00	7
	hepatitis	155	19	6	13	5.67	2
	iris	150	4	4	0	0.00	3
	kdd_synthetic	600	61	60	1	0.00	6
	segment	2,310	19	19	0	0.00	7
	semeion	1,593	265	265	0	0.00	2
	shuttle_landing	15	6	0	6	28.89	2
	sick	3,772	30	6	22	5.54	2
	tempdiag	120	7	1	6	0.00	2
	tep.fea	3,572	7	7	0	0.00	3
	vowel	990	13	10	3	0.00	11
	winequality_red	1,599	11	11	0	0.00	10
	winequality_white	4,898	11	11	0	0.00	10
Gene expression	alizadeh-2000-v1	42	1,095	1,095	0	0.0	2
	armstrong-2002-v1	72	1,081	1,081	0	0.0	2
	armstrong-2002-v2	72	2,194	2,194	0	0.0	3
	bittner-2000	38	2,201	2,201	0	0.0	2
	liang-2005	37	1,411	1,411	0	0.0	3
	ramaswamy-2001	190	1,363	1,363	0	0.0	14
	risinger-2003	42	1,771	1,771	0	0.0	4
	tomlins-2006	104	2,315	2,315	0	0.0	5
	tomlins-2006-v2	92	1,288	1,288	0	0.0	4
	yeoh-2002-v1	248	2,526	2,526	0	0.0	2

If the null hypothesis of similar performances is rejected, we proceed with the Nemenyi post hoc test for pairwise comparisons. The performance of two classifiers is significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{8}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

4.3 Parameters

The baseline algorithms CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993) were used with the default parameter values proposed by their authors, which typically represent robust values that work well across different datasets. None of the algorithms, including HEAD-DT, had their parameter values optimized to individual datasets, since the goal was to evaluate a parameter settings' generalization ability across a wide range of datasets, as usual in the supervised machine learning literature.

Table 2: Parameter values for HEAD-DT.

Parameter description	Value
Number of individuals	100
Number of generations	100
Tournament selection size	2
Elitism rate	5%
Crossover rate	90%
Mutation rate	5%
Reproduction rate	5%

HEAD-DT was configured with the parameter values described in Table 2. These values were defined in previous nonexhaustive empirical tests, and no further attempt of optimizing these values was made. A more robust parameter optimization procedure is left for future research.

5 Results

In this section, we present the results obtained by HEAD-DT with the strategy of generating one algorithm per dataset. Recall that, in this case, both the meta-training and meta-test sets contain data from a specific dataset. We used 10-fold cross-validation to evaluate the results provided by HEAD-DT and by the baseline algorithms C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984).

For the UCI datasets, we employed predictive accuracy (see Equation (2)) as fitness function, whereas for the gene expression (GE) datasets we employed the F-Measure (see Equation (5)), considering that most GE datasets are quite imbalanced. The results for the UCI datasets were also reported in our previous work (Barros et al., 2012a), but the results for the GE datasets are a new contribution of this paper.

5.1 Results for the UCI Datasets

Table 3 shows the classification accuracy of CART, C4.5, and HEAD-DT for the 20 UCI datasets. It illustrates the average accuracy over the 10-fold cross-validation runs $\pm$ the *SD* of the accuracy obtained in these runs (best absolute values in bold). It is possible to see that HEAD-DT generates more accurate trees in 13 out of the 20 datasets. CART provides more accurate trees in two datasets, and C4.5 in none. In the remaining five datasets, no method was superior to the others.

To evaluate the statistical significance of the predictive accuracy results, we calculated the average rank for CART, C4.5, and HEAD-DT to be 2.375, 2.2, and 1.425, respectively. The average ranks suggest that HEAD-DT is the best performing method regarding accuracy. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 20}{3 \times 4}\left[2.375^2 + 2.2^2 + 1.425^2 - \frac{3 \times 4^2}{4}\right] = 10.225 \tag{9}$$

and the Iman's F statistic is given by:

$$F_f = \frac{(20-1) \times 10.225}{20 \times (3-1) - 10.22} = 6.52 \tag{10}$$

The critical value of $F(k-1,(k-1)(n-1)) = F(2,38)$ for $\alpha = 0.05$ is 3.25. Since $F_f > F_{0.05}(2,38)$ $(6.52 > 3.25)$, the null-hypothesis is rejected. We proceed with a

Table 3: Classification accuracy of CART, C4.5, and HEAD-DT—UCI datasets.

Dataset	CART	C4.5	HEAD-DT
abalone	**0.26 ± 0.02**	0.22 ± 0.02	0.20 ± 0.02
anneal	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
arrhythmia	**0.71 ± 0.05**	0.65 ± 0.04	0.65 ± 0.04
audiology	0.74 ± 0.05	0.78 ± 0.07	**0.80 ± 0.06**
bridges_version1	0.41 ± 0.07	0.57 ± 0.10	**0.60 ± 0.12**
car	0.97 ± 0.02	0.93 ± 0.02	**0.98 ± 0.01**
cylinder_bands	0.60 ± 0.05	0.58 ± 0.01	**0.72 ± 0.04**
glass	0.70 ± 0.11	0.69 ± 0.04	**0.73 ± 0.10**
hepatitis	0.79 ± 0.05	0.79 ± 0.06	**0.81 ± 0.08**
iris	0.93 ± 0.05	0.94 ± 0.07	**0.95 ± 0.04**
kdd_synthetic	0.88 ± 0.00	0.91 ± 0.04	**0.97 ± 0.03**
segment	0.96 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
semeion	0.94 ± 0.01	0.95 ± 0.02	**1.00 ± 0.00**
shuttle_landing	0.95 ± 0.16	0.95 ± 0.16	**0.95 ± 0.15**
sick	0.99 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
tempdiag	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
tep.fea	0.65 ± 0.02	0.65 ± 0.02	0.65 ± 0.02
vowel	0.82 ± 0.04	0.83 ± 0.03	**0.89 ± 0.04**
winequality_red	0.63 ± 0.02	0.61 ± 0.03	**0.64 ± 0.03**
winequality_white	0.58 ± 0.02	0.61 ± 0.03	**0.63 ± 0.03**

post-hoc Nemenyi test to find which method provides the best accuracy. The critical difference *CD* is given by:

$$CD = 2.343 \times \sqrt{\frac{3 \times 4}{6 \times 20}} = 0.74 \tag{11}$$

The difference between the average rank of HEAD-DT and C4.5 is 0.775, while the difference between HEAD-DT and CART is 0.95. Since both the differences are larger than *CD*, the performance of HEAD-DT is significantly better than both C4.5 and CART regarding predictive accuracy.

Table 4 shows the classification F-measure of CART, C4.5, and HEAD-DT for the same UCI datasets. It can be seen that HEAD-DT generates better trees (regardless of the class imbalance problem) in 16 out of the 20 datasets. CART generates the best tree in two datasets, while C4.5 does not induce the best tree for any dataset.

We calculated the average rank for CART, C4.5, and HEAD-DT to be 2.5, 2.225 and 1.275, respectively. The average rank suggests that HEAD-DT is the best performing method regarding the F-measure. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 20}{3 \times 4}\left[2.5^2 + 2.225^2 + 1.275^2 - \frac{3 \times 4^2}{4}\right] = 16.525 \tag{12}$$

and the Iman's *F* statistic is given by:

$$F_f = \frac{(20-1) \times 16.525}{20 \times (3-1) - 16.525} = 13.375 \tag{13}$$

Since $F_f > F_{0.05}(2, 38)$ $(13.375 > 3.25)$, the null hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 0.95, while the difference between

Table 4: Classification F-measure of CART, C4.5, and HEAD-DT—UCI datasets.

Dataset	CART	C4.5	HEAD-DT
abalone	**0.22 ± 0.02**	0.21 ± 0.02	0.20 ± 0.02
anneal	0.98 ± 0.01	0.98 ± 0.01	**0.99 ± 0.01**
arrhythmia	**0.67 ± 0.05**	0.64 ± 0.05	0.63 ± 0.06
audiology	0.70 ± 0.04	0.75 ± 0.08	**0.79 ± 0.07**
bridges_version1	0.44 ± 0.06	0.52 ± 0.10	**0.56 ± 0.12**
car	0.93 ± 0.97	0.93 ± 0.02	**0.98 ± 0.01**
cylinder_bands	0.54 ± 0.07	0.42 ± 0.00	**0.72 ± 0.04**
glass	0.67 ± 0.10	0.67 ± 0.05	**0.72 ± 0.09**
hepatitis	0.74 ± 0.07	0.77 ± 0.06	**0.80 ± 0.08**
iris	0.93 ± 0.05	0.93 ± 0.06	**0.95 ± 0.05**
kdd_synthetic	0.88 ± 0.03	0.90 ± 0.04	**0.97 ± 0.03**
segment	0.95 ± 0.01	0.96 ± 0.09	**0.97 ± 0.01**
semeion	0.93 ± 0.01	0.95 ± 0.02	**1.00 ± 0.00**
shuttle_landing	0.56 ± 0.03	0.56 ± 0.38	**0.93 ± 0.20**
sick	0.98 ± 0.00	0.98 ± 0.00	**0.99 ± 0.00**
tempdiag	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
tep.fea	0.60 ± 0.02	0.61 ± 0.02	0.61 ± 0.02
vowel	0.81 ± 0.03	0.82 ± 0.03	**0.89 ± 0.03**
winequality_red	0.61 ± 0.02	0.60 ± 0.03	**0.63 ± 0.03**
winequality_white	0.57 ± 0.02	0.60 ± 0.02	**0.63 ± 0.03**

HEAD-DT and CART is 1.225. Since both the differences are larger than *CD* (0.74), the performance of HEAD-DT is significantly better than both C4.5 and CART regarding the F-measure.

Table 5 shows the size of decision trees (measured as the number of tree nodes) generated by C4.5, CART, and HEAD-DT. It can be seen that CART generates smaller trees in 15 out of the 20 datasets. C4.5 generates smaller trees in two datasets, and HEAD-DT in only one dataset. The statistical analysis is given below:

$$\chi_F^2 = \frac{12 \times 20}{3 \times 4}\left[1.2^2 + 2^2 + 2.8^2 - \frac{3 \times 4^2}{4}\right] = 25.6 \tag{14}$$

$$F_f = \frac{(20-1) \times 33.78}{20 \times (3-1) - 25.6} = 33.78 \tag{15}$$

Since $F_f > F_{0.05}(2, 38)$ ($33.78 > 3.25$), the null hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 0.8, while the difference between HEAD-DT and CART is 1.6. Since both the differences are larger than *CD* (0.74), HEAD-DT generates trees which are significantly larger than both C4.5 and CART for these UCI datasets.

5.2 Results for the Gene Expression Datasets

In this section, we present the results for 10 microarray GE datasets (de Souto et al., 2008). Microarray technology enables expression level measurement for thousands of genes in parallel, given a biological tissue. Once combined, a fixed number of microarray experiments comprise composed a gene expression dataset. The considered datasets are related to different types or subtypes of cancer (e.g., prostate, lung, and skin) and

Table 5: Tree size of CART, C4.5, and HEAD-DT trees—UCI datasets.

Dataset	CART	C4.5	HEAD-DT
abalone	**44.40 ± 16.00**	2088.90 ± 37.63	4068.12 ± 13.90
anneal	**21.00 ± 3.13**	48.30 ± 6.48	55.72 ± 3.66
arrhythmia	**23.20 ± 2.90**	82.60 ± 5.80	171.84 ± 5.18
audiology	**35.80 ± 11.75**	50.40 ± 4.01	118.60 ± 3.81
bridges_version1	**1.00 ± 0.00**	24.90 ± 20.72	156.88 ± 14.34
car	**108.20 ± 16.09**	173.10 ± 6.51	171.92 ± 4.45
cylinder_bands	4.20 ± 1.03	**1.00 ± 0.00**	211.44 ± 9.39
glass	**23.20 ± 10.56**	44.80 ± 5.20	86.44 ± 3.14
hepatitis	**6.60 ± 8.58**	15.40 ± 4.40	71.80 ± 4.77
iris	**6.20 ± 1.69**	8.00 ± 1.41	20.36 ± 1.81
kdd_synthetic	**1.00 ± 0.00**	37.80 ± 4.34	26.16 ± 2.45
segment	**78.00 ± 8.18**	80.60 ± 4.97	132.76 ± 3.48
semeion	34.00 ± 12.30	55.00 ± 8.27	**19.00 ± 0.00**
shuttle_landing	1.00 ± 0.00	1.00 ± 0.00	5.64 ± 1.69
sick	**45.20 ± 11.33**	46.90 ± 9.41	153.70 ± 8.89
tempdiag	5.00 ± 0.00	5.00 ± 0.00	5.32 ± 1.04
tep.fea	13.00 ± 2.83	**8.20 ± 1.69**	18.84 ± 1.97
vowel	**175.80 ± 23.72**	220.70 ± 20.73	361.42 ± 5.54
winequality_red	**151.80 ± 54.58**	387.00 ± 26.55	796.00 ± 11.22
winequality_white	**843.80 ± 309.01**	1367.20 ± 58.44	2525.88 ± 13.17

cover the two flavors in which the technology is generally available: single channel and double channel. The classification task consists of classifying different instances according to their gene (attribute) expression levels. Note that the classification of microarray data is challenging because the number of instances is much smaller than the number of attributes, which makes classification algorithms applied to such data prone to overfitting.

Table 6 shows the classification accuracy of CART, C4.5, and HEAD-DT regarding the 10 GE datasets. It illustrates the average accuracy over the 10-fold cross-validation runs ± the *SD* of the accuracy obtained in these runs (best absolute values in bold). One can see that HEAD-DT generates more accurate trees in nine out of the 10 datasets. In the remaining dataset (yeoh-2002-v1), all methods provide the same decision tree (and thus, obtain the same accuracy value).

We calculated the average rank for CART, C4.5, and HEAD-DT to be 2.25, 2.65, and 1.1, respectively. The average rank suggest that HEAD-DT is the best performing method regarding accuracy. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 10}{3 \times 4}\left[2.65^2 + 2.25^2 + 1.1^2 - \frac{3 \times 4^2}{4}\right] = 12.95 \tag{16}$$

and the Iman's F statistic is given by:

$$F_f = \frac{(10-1) \times 12.95}{10 \times (3-1) - 12.95} = 16.53 \tag{17}$$

$F(k-1,(k-1)(n-1)) = F(2,18)$ for $\alpha = 0.05$ is 3.55. Since $F_f > F_{0.05}(2,18)$ $(16.53 > 3.55)$, the null hypothesis is rejected. We proceed with the post hoc pairwise Nemenyi

Table 6: Classification accuracy of CART, C4.5, and HEAD-DT—GE datasets.

Dataset	CART	C4.5	HEAD-DT
alizadeh-2000-v1	0.71 ± 0.16	0.68 ± 0.20	$\mathbf{0.76 \pm 0.18}$
armstrong-2002-v1	0.90 ± 0.07	0.89 ± 0.06	$\mathbf{0.91 \pm 0.11}$
armstrong-2002-v2	0.85 ± 0.05	0.83 ± 0.05	$\mathbf{0.90 \pm 0.10}$
bittner-2000	0.53 ± 0.18	0.49 ± 0.16	$\mathbf{0.56 \pm 0.19}$
liang-2005	0.71 ± 0.14	0.76 ± 0.19	$\mathbf{0.79 \pm 0.17}$
ramaswamy-2001	0.60 ± 0.09	0.61 ± 0.05	$\mathbf{0.63 \pm 0.08}$
risinger-2003	0.52 ± 0.15	0.53 ± 0.19	$\mathbf{0.60 \pm 0.29}$
tomlins-2006-v1	0.58 ± 0.20	0.58 ± 0.18	$\mathbf{0.58 \pm 0.17}$
tomlins-2006-v2	0.59 ± 0.15	0.58 ± 0.17	$\mathbf{0.61 \pm 0.09}$
yeoh-2002-v1	0.99 ± 0.02	0.99 ± 0.02	0.99 ± 0.02

Table 7: Classification F-measure of CART, C4.5, and HEAD-DT—GE datasets.

Dataset	CART	C4.5	HEAD-DT
alizadeh-2000-v1	0.67 ± 0.21	0.63 ± 0.26	$\mathbf{0.72 \pm 0.23}$
armstrong-2002-v1	0.90 ± 0.07	0.88 ± 0.06	$\mathbf{0.91 \pm 0.11}$
armstrong-2002-v2	0.84 ± 0.05	0.83 ± 0.05	$\mathbf{0.89 \pm 0.11}$
bittner-2000	0.49 ± 0.21	0.45 ± 0.19	$\mathbf{0.51 \pm 0.23}$
liang-2005	0.67 ± 0.21	0.77 ± 0.22	$\mathbf{0.78 \pm 0.21}$
ramaswamy-2001	0.56 ± 0.10	0.56 ± 0.07	$\mathbf{0.59 \pm 0.08}$
risinger-2003	0.48 ± 0.19	0.53 ± 0.21	$\mathbf{0.57 \pm 0.29}$
tomlins-2006-v1	0.56 ± 0.20	0.56 ± 0.20	0.53 ± 0.16
tomlins-2006-v2	0.55 ± 0.13	0.57 ± 0.17	$\mathbf{0.57 \pm 0.10}$
yeoh-2002-v1	0.99 ± 0.02	0.99 ± 0.02	0.99 ± 0.02

test. The critical difference *CD* is given by:

$$CD = 2.343 \times \sqrt{\frac{3 \times 4}{6 \times 10}} = 1.05 \tag{18}$$

The difference between the average rank of HEAD-DT and C4.5 is 1.55, while the difference between HEAD-DT and CART is 1.15. Since both differences are larger than *CD* (1.05), the performance of HEAD-DT is significantly better than both C4.5 and CART regarding the predictive accuracy in the GE datasets.

Next, Table 7 shows the classification F-measure of CART, C4.5, and HEAD-DT for the same GE datasets. The experimental results show that HEAD-DT generates better trees (regardless of the class imbalance problem) in eight out of the 10 datasets. CART and C4.5 generate the best tree for tomlins2006-v1, while all methods generate the same tree for yeah-2002-v1.

We calculated the average rank for CART, C4.5, and HEAD-DT to be 2.2, 2.75, and 1.05, respectively. The average rank suggest that HEAD-DT is the best performing method regarding the F-measure. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 10}{3 \times 4}\left[2.75^2 + 2.2^2 + 1.05^2 - \frac{3 \times 4^2}{4}\right] = 15.05 \tag{19}$$

Table 8: Tree size of CART, C4.5, and HEAD-DT trees—GE datasets.

Dataset	CART	C4.5	HEAD-DT
alizadeh-2000-v1	**3.20 ± 0.63**	5.00 ± 0.00	4.00 ± 1.29
armstrong-2002-v1	**3.00 ± 0.00**	3.60 ± 0.97	4.32 ± 1.74
armstrong-2002-v2	**5.00 ± 0.00**	7.20 ± 0.63	6.20 ± 0.99
bittner-2000	**3.80 ± 1.40**	5.00 ± 0.00	4.08 ± 2.18
liang-2005	**2.20 ± 1.03**	5.00 ± 0.00	3.96 ± 1.41
ramaswamy-2001	**28.20 ± 7.38**	45.20 ± 2.74	39.04 ± 11.31
risinger-2003	**5.00 ± 1.89**	8.60 ± 0.84	6.40 ± 2.53
tomlins-2006-v1	**12.60 ± 2.80**	17.00 ± 1.33	12.80 ± 5.82
tomlins-2006-v2	**7.40 ± 3.10**	16.40 ± 0.97	11.32 ± 4.75
yeoh-2002-v1	3.00 ± 0.00	3.00 ± 0.00	3.48 ± 2.04

and the Iman's F statistic is given by:

$$F_f = \frac{(10-1) \times 15.05}{10 \times (3-1) - 15.05} = 27.36 \tag{20}$$

Since $F_f > F_{0.05}(2, 18)$ ($27.36 > 3.55$), the null hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 1.7, while the difference between HEAD-DT and CART is 1.15. Since both the differences are larger than *CD* (1.05), the performance of HEAD-DT is significantly better than both C4.5 and CART regarding the F-measure.

Finally, Table 8 presents the average tree size for CART, C4.5, and HEAD-DT regarding the GE datasets. CART provides the smaller trees for all datasets (it generates the same tree as C4.5 in dataset *yeah-2002-v1*).

We calculated the average rank for CART, C4.5, and HEAD-DT to be 1.3, 2.4, and 2.3, respectively. The average rank suggests that CART generates smaller trees than both C4.5 and HEAD-DT. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 10}{3 \times 4}\left[2.4^2 + 2.3^2 + 1.3^2 - \frac{3 \times 4^2}{4}\right] = 7.40 \tag{21}$$

and the Iman's F statistic is given by:

$$F_f = \frac{(10-1) \times 7.40}{10 \times (3-1) - 7.40} = 5.29 \tag{22}$$

Since $F_f > F_{0.05}(2, 18)$ ($5.29 > 3.55$), the null hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 0.1, while the difference between HEAD-DT and CART is 1.0. The size of trees generated by CART are significantly smaller than those generated by C4.5 and HEAD-DT. However, there are no significant differences between HEAD-DT and C4.5 regarding the tree size.

5.3 Summary

In the experiments performed for this study, the decision-tree induction algorithms automatically designed by HEAD-DT induced decision trees with better predictive performance than both CART and C4.5, which are very popular and very effective manually-designed decision-tree induction algorithms. In both experiments (with UCI and GE datasets), HEAD-DT consistently presented better results with statistical assurance. The

performance evaluation measures we employed in these experiments—accuracy and F-measure—are among the most well-known and used measures in data mining and machine learning classification problems.

Regarding tree complexity, which was measured as the total number of nodes in a tree (tree size), we observed that CART usually induced much smaller trees, probably due to its particular pruning method (cost complexity pruning). Nevertheless, we saw that smaller trees do not necessarily translate into better performance, since the trees generated by the automatically-designed algorithms are significantly more accurate than those generated by CART.

We conclude from these experiments that HEAD-DT is indeed an effective alternative to state of the art decision-tree induction algorithms C4.5 and CART. Next, we broaden our discussion by commenting on other issues involved in decision-tree induction, and also on the trade-off between HEAD-DT's predictive performance and computational efficiency.

6 Discussion

In this section, we discuss three important topics to support the empirical analysis presented in the previous section: (1) the accuracy dilemma and when preferring F-Measure for evaluating decision-tree induction algorithms; (2) the empirical (and theoretic) time complexity of HEAD-DT, and also of the decision-tree induction algorithms it generates; and (3) an example of a decision-tree induction algorithm automatically designed by HEAD-DT.

6.1 Accuracy versus F-Measure

The fact that HEAD-DT designs algorithms that induce significantly more accurate decision trees than C4.5 and CART is very encouraging. Notwithstanding this result, we must point out that accuracy may be a misleading performance measure. For instance, suppose we have a dataset whose class distribution is very skewed: 90% of the instances belong to class A and 10% to class B. An algorithm that always classifies instances as belonging to class A would achieve 90% accuracy, even though it never predicts a class-B instance. In this case, assuming that class B is equally important (or even more so) than class A, we would prefer an algorithm with lower accuracy, but which could eventually correctly predict some instances as belonging to the rare class B.

The previous example illustrates the importance of not relying solely on accuracy when designing an algorithm. F-measure is the harmonic mean of *precision* and *recall*, and thus rewards solutions that present a good trade-off between these two measures. Hence, for imbalanced-class problems, F-measure should be preferred over accuracy.

In the experimental analysis performed in the previous section, recall that we optimized solutions toward accuracy (in the UCI datasets) and F-measure (in the GE datasets) by varying the HEAD-DT fitness function. We observed that, regardless of the measure being optimized, HEAD-DT was able to generate robust solutions for balanced and imbalanced datasets.

6.2 HEAD-DT Complexity Analysis

Regarding execution time, it is clear that HEAD-DT is slower than either C4.5 or CART. Considering that there are 100 individuals executed for 100 generations, there is a maximum (worst case) of 10,000 fitness evaluations of decision trees.

Algorithm 2 Decision-tree induction algorithm designed by HEAD-DT for the *semeion* dataset

```
1) Recursively split nodes using the Chandra-Varghese criterion;
2) Aggregate nominal splits in binary subsets;
3) Perform step 1 until class-homogeneity or the minimum number of 5 instances is reached;
4) Perform MEP pruning with m = 10;
5) When dealing with missing values:
     5.1) Calculate the split of missing values by performing unsupervised imputation;
     5.2) Distribute missing values by assigning the instance to all partitions;
     5.3) For classifying an instance, explore all branches and combine the results.
```


We recorded the execution time of both breeding operations and fitness evaluation (one thread was used for breeding and another for evaluation). The total time of breeding is absolutely negligible (a few milliseconds in a full evolutionary cycle), regardless of the dataset being used (breeding does not consider any domain-specific information). Indeed, breeding individuals in the form of an integer string is known to be quite efficient in the EA research field.

Fitness evaluation, on the other hand, is the bottleneck of HEAD-DT. In the largest UCI dataset (winequality_white), HEAD-DT took 2.5 hr to be fully executed (one iteration of the cross-validation procedure, in a full evolutionary cycle of 100 generations). In the smallest UCI dataset (shuttle_landing), HEAD-DT took only 0.72 s to be fully executed, which means the fitness evaluation time can largely vary according to the dataset size.

It should be noted that, even in cases where a run of HEAD-DT took several hours, this is still a very short time in the context of algorithm design, since, in general, even a machine learning researcher with expertise in decision-tree induction would take much longer to design a novel decision-tree algorithm that is competitive with C4.5 and CART.

The computational complexity of algorithms such as C4.5 and CART is $O(m \times n \log n)$ (m is the number of attributes and n the number of instances), plus a term regarding the specific pruning method. Considering that breeding takes negligible time, we can say that in the worst case scenario, HEAD-DT time complexity is $O(i \times g \times m \times n \log n)$, where i is the number of individuals and g is the number of generations. In practice, the number of evaluations is much smaller than $i \times g$, due to the fact that repeated individuals are not reevaluated. In addition, individuals selected by elitism and by reproduction (instead of crossover) are also not reevaluated, saving computational time.

6.3 Example of an Evolved Decision-Tree Algorithm

In order to illustrate a novel decision-tree induction algorithm designed by HEAD-DT, let us consider the semeion dataset, in which HEAD-DT managed to achieve maximum accuracy and F-measure (which was not the case for CART and C4.5). The algorithm designed by HEAD-DT is presented in Algorithm 2. It is indeed novel, since no algorithm in the literature combines components such as the Chandra-Varghese criterion with MEP pruning. Furthermore, it chooses MEP as its pruning method, which is a surprise considering that MEP is usually a neglected pruning method in the decision-tree literature.

The main advantage of HEAD-DT is that it automatically searches for the suitable components (with their own biases) according to the dataset being investigated. It is

hard to believe that a human researcher would combine such a distinct set of components like those in Algorithm 2 to achieve 100% accuracy in a particular dataset.

7 Related Work

To the best of our knowledge, no work to date has attempted to automatically design full decision-tree induction algorithms (except of course our previous work, Barros et al., 2012a, which has been extended in this paper, as mentioned in the Introduction). The most related approach to this work is HHDT (Vella et al., 2009). It proposes an EA for evolving heuristic rules in order to determine the best splitting criterion to be used in nonterminal nodes. While this approach is a first step to automate the design of decision-tree induction algorithms, it evolves a single component of the algorithm (the choice of splitting criterion), and thus should be further extended in order to be able to generate full decision-tree induction algorithms.

A somewhat related approach is the one presented by Delibasic et al. (2011). The authors propose a framework for combining decision-tree components, and test 80 different combination of design components on 15 benchmark datasets. This approach is not a hyper-heuristic, since it does not present a heuristic to choose among different heuristics. It simply selects a fixed number of component combinations and tests them all against traditional decision-tree induction algorithms (C4.5, CART, ID3, and CHAID). We believe that our strategy is more robust, since by using an EA, it can search for solutions in a much larger search space. Currently, HEAD-DT searches in the space of more than 127 million different candidate decision-tree induction algorithms.

8 Conclusions and Future Work

In this paper, we presented HEAD-DT, a hyper-heuristic EA that automatically designs top-down decision-tree induction algorithms. Top-down decision-tree induction algorithms have been manually improved over 40 years of research, leading to a large number of proposed approaches for each of their design components. Since the human manual combination of all available design components of these algorithms would be infeasible, we believe the evolutionary search of HEAD-DT constitutes a robust and efficient solution for the design of new and effective algorithms.

We performed a thorough experimental analysis in which the algorithms automatically designed by HEAD-DT were compared to the state of the art decision-tree induction algorithms CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993). In this analysis, we evaluated the effectiveness of HEAD-DT in two scenarios: (1) general performance in 20 well-known UCI datasets with very different characteristics, from very different application domains; and (2) domain-specific performance in 10 real-world microarray gene expression datasets. We assessed the performance of HEAD-DT through two distinct performance evaluation measures (predictive accuracy and F-measure), and a tree complexity measure (tree size). In both scenarios, the experimental results suggested that HEAD-DT can generate decision-tree induction algorithms with predictive performance significantly higher than CART and C4.5, although sometimes generating larger trees. Bearing in mind that an accurate prediction system is widely preferred over a significantly less accurate (but simpler) system, we believe that HEAD-DT arises as a robust algorithm for future applications of decision trees.

As future work, we intend to develop a multi-objective fitness function, considering the trade-off between predictive performance and parsimony. In addition, we plan to investigate whether a more sophisticated search system, such as grammar-based genetic

programming, can outperform our current HEAD-DT implementation. Optimizing the evolutionary parameters of HEAD-DT is also a topic left for future research.

Acknowledgment

The authors would like to thank Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Brazil, for funding this research.

References

Alpaydin, E. (2010). *Introduction to machine learning*, 2nd ed. Cambridge, MA: MIT Press.

Barros, R. C., Basgalupp, M. P., Ruiz, D. D., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2010). Evolutionary model tree induction. In *Proceedings of the ACM Symposium on Applied Computing (SAC 2010)*, pp. 1131–1137.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2011). Towards the automatic design of decision tree induction algorithms. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary computation*, GECCO '11, pages 567–574, New York, NY, USA. ACM.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2012a). A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms. In *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference, GECCO '12*, pp. 1237–1244.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2012b). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 42(3):291–312.

Barros, R. C., Cerri, R., Jaskowiak, P. A., and de Carvalho, A. C. P. L. F. (2011). A bottom-up oblique decision tree induction algorithm. In *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications*, pp. 450–456.

Barros, R. C., Ruiz, D. D., and Basgalupp, M. P. (2011). Evolutionary model trees for handling continuous classes in machine learning. *Information Sciences*, 181:954–971.

Basgalupp, M. P., Barros, R. C., de Carvalho, A. C. P. L. F., Freitas, A. A., and Ruiz, D. D. (2009). Legal-tree: A lexicographic multi-objective genetic algorithm for decision tree induction. In *Proceedings of the ACM Symposium on Applied Computing (SAC 2009)*, pp. 1085–1090.

Basgalupp, M. P., de Carvalho, A. C. P. L. F., Barros, R. C., Ruiz, D. D., and Freitas, A. A. (2009). Lexicographic multi-objective evolutionary induction of decision trees. *International Journal of Bioinspired Computation*, 1(1–2):105–117.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Breslow, L., and Aha, D. (1997). Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(1):1–40.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Colaborative Computational Intelligence* (pp. 177–201). Berlin: Springer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. (2010). A classification of hyper-heuristic approaches. In *Operations research & management science* (pp. 449–468). Berlin: Springer.

Cestnik, B., and Bratko, I. (1991). On estimating probabilities in tree pruning. In *Proceedings of the European Working Session on Learning on Machine Learning (EWSL'91)*, pp. 138–150.

Chandra, B., Kothari, R., and Paul, P. (2010). A new node splitting measure for decision tree construction. *Pattern Recognition*, 43(8):2725–2731.

Chandra, B., and Varghese, P. P. (2009). Moving towards efficient decision tree construction. *Information Sciences*, 179(8):1059–1069.

Ching, J., Wong, A., and Chan, K. (1995). Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):641–651.

Clark, P., and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4):261–283.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92.

de Souto, M., Costa, I., de Araujo, D., Ludermir, T., and Schliep, A. (2008). Clustering cancer gene expression data: A comparative study. *BMC Bioinformatics*, 9(1):497.

Delibasic, B., Jovanovic, M., Vukicevic, M., Suknovic, M., and Obradovic, Z. (2011). Component-based decision trees for classification. *Intelligent Data Analysis*, 15:1–38.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Esposito, F., Malerba, D., and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

Fayyad, U., and Irani, K. (1992). The attribute selection problem in decision tree generation. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 104–110.

Frank, A., and Asuncion, A. (2010). UCI machine learning repository. Available at http://archive.ics.uci.edu/ml

Friedman, J. H. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, 100(4):404–408.

Gleser, M., and Collen, M. (1972). Towards automated medical decisions. *Computers and Biomedical Research*, 5(2):180–189.

Hancock, T., Jiang, T., Li, M., and Tromp, J. (1996). Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122.

Hunt, E. B., Marin, J., and Stone, P. J. (1966). *Experiments in induction*. New York: Academic Press.

Hyafil, L., and Rivest, R. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17.

Iman, R., and Davenport, J. (1980). Approximations of the critical region of the Friedman statistic. *Communications in Statistics A, Theory and Methods*, 9(6):571–595.

Jun, B., Kim, C., Song, Y.-Y., and Kim, J. (1997). A new criterion in selection and discretization of attributes for the generation of decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):1371–1375.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127.

Kim, B., and Landgrebe, D. (1991). Hierarchical classifier design in high-dimensional numerous class cases. *IEEE Transactions on Geoscience and Remote Sensing*, 29(4):518–528.

Kononenko, I., Bratko, I., and Roskar, E. (1984). Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, Yugoslavia.

Loh, W., and Shih, Y. (1997). Split selection methods for classification trees. *Statistica Sinica*, 7:815–840.

Martin, J. (1997). An exact probability metric for decision tree splitting and stopping. *Machine Learning*, 28(2):257–291.

Mingers, J. (1987). Expert systems; Rule induction with statistical data. *Journal of the Operational Research Society*, 38:39–47.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342.

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389.

Naumov, G. E. (1991). NP-completeness of problems of construction of optimal decision trees. *Soviet Physics: Doklady*, 36(4).

Niblett, T., and Bratko, I. (1986). Learning decision rules in noisy domains. In *Proceedings of the 6th Annual Technical Conference on Expert Systems*, pp. 25–34.

Pappa, G. L., and Freitas, A. A. (2010). *Automating the design of data mining algorithms: An evolutionary computation approach*. Berlin: Springer.

Patterson, A., and Niblett, T. (1983). *ACLS user manual*. Glasgow, UK: Intelligent Terminals Ltd.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. (1987a). Decision trees as probabilistic classifiers. In *Proceedings of the Fourth International Machine Learning Workshop*, pp. 31–37.

Quinlan, J. R. (1987b). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234.

Quinlan, J. R. (1989). Unknown attribute values in induction. In *Proceedings of the 6th International Workshop on Machine Learning*, pp. 164–168.

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

Safavian, S., and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660–674.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(1):379–423, 625–656.

Sonquist, J. A., Baker, E. L., and Morgan, J. N. (1971). *Searching for structure*. Ann Arbor, MI: Institute for Social Research, University of Michigan Press.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to data mining*. Reading, MA: Addison-Wesley.

Taylor, P. C., and Silverman, B. W. (1993). Block diagrams and splitting criteria for classification trees. *Statistics and Computing*, 3:147–161.

Vella, A., Corne, D., and Murphy, C. (2009). Hyper-heuristic decision tree induction. *World Congress on Nature & Biologically Inspired Computing*, pp. 409–414.

Zantema, H., and Bodlaender, H. (2000). Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science*, 11(2):343–354.