22

Evolutionary Algorithms in Decision Tree Induction

Francesco Mola[1], Raffaele Miele[2] and Claudio Conversano[1]
[1]*University of Cagliari,*
[2]*University of Naples Federico II*
Italy

1. Introduction

One of the biggest problem that many data analysis techniques have to deal with nowadays is Combinatorial Optimization that, in the past, has led many methods to be taken apart. Actually, the (still not enough!) higher computing power available makes it possible to apply such techniques within certain bounds. Since other research fields like Artificial Intelligence have been (and still are) dealing with such problems, their contribute to statistics has been very significant.
This chapter tries to cast the Combinatorial Optimization methods into the Artificial Intelligence framework, particularly with respect Decision Tree Induction, which is considered a powerful instrument for the knowledge extraction and the decision making support. When the exhaustive enumeration and evaluation of all the possible candidate solution to a Tree-based Induction problem is not computationally affordable, the use of Nature Inspired Optimization Algorithms, which have been proven to be powerful instruments for attacking many combinatorial optimization problems, can be of great help.
In this respect, the attention is focused on three main problems involving Decision Tree Induction by mainly focusing the attention on the Classification and Regression Tree-CART (Breiman et al., 1984) algorithm. First, the problem of splitting complex predictors such a multi-attribute ones is faced through the use of Genetic Algorithms. In addition, the possibility of growing "optimal" exploratory trees is also investigated by making use of Ant Colony Optimization (ACO) algorithm. Finally, the derivation of a subset of decision trees for modelling multi-attribute response on the basis of a data-driven heuristic is also described. The proposed approaches might be useful for knowledge extraction from large databases as well as for data mining applications. The solution they offer for complicated data modelling and data analysis problems might be considered for a possible implementation in a Decision Support System (DSS).
The remainder of the chapter is as follows. Section 2 describes the main features and the recent developments of Decision Tree Induction. An overview of Combinatorial Optimization with a particular focus on Genetic Algorithms and Ant Colony Optimization is presented in section 3. The use of these two algorithms within the Decision Tree Induction Framework is described in section 4, together with the description of the algorithm for modelling multi-attribute response. Section 5 summarizes the results of the proposed

method on real and simulated datasets. Concluding remarks are presented in section 6. The chapter also includes an appendix that presents J-Fast, a Java-based software for Decision Tree that currently implements Genetic Algorithms and Ant Colony Optimization.

2. Decision tree induction

Decision Tree Induction (DTI) is a tool to induce a classification or regression model from (usually large) datasets characterized by N observations (records), each one containing a set $\mathbf{x}$ of numerical or nominal variables, and a variable y. Statisticians use the terms "splitting predictors" to identify $\mathbf{x}$ and "response variable" for y. DTI builds a model that summarizes the underlying relationships between $\mathbf{x}$ and y. Actually, two kinds of model can be estimated using decision trees: classification trees if y is nominal, and regression trees if y is numerical. Hereinafter we refer to classification trees to show the main features of DTI and briefly recall the main characteristics of regression trees at the end of the section.

DTI proceeds by inducing a series of follow-up (usually binary) questions about the attributes of an unknown observation until a conclusion about what is its most likely class label is reached. Questions and their alternative answers can be represented hierarchically in the form of a decision tree. It contains a root node and some internal and terminal nodes. The root node and the internal ones are used to partition observations of the dataset into smaller subsets of relatively homogeneous classes. To classify a previously unlabelled observation, say i^* $(i^*=1,\ldots,N)$, we start from the test condition in the root node and follow the appropriate pattern based on the outcome of the test. When an internal node is reached a new test condition is applied, and so on down to a terminal node. Encountering a terminal node, the modal class of the observations in that node is the class label of y assigned to the (previously) unlabeled observation. For regression trees, the assigned class is the mean of y for the observations belonging to that terminal node.

Because of their top-down binary splitting approach, decision trees can easily be converted into IF-THEN rules and used for decision making purposes.

DTI is useful for knowledge extraction from large databases and data mining applications because of the possibility to represent functions of numerical and nominal variables as well as of its feasibility, predictive ability and interpretability. It can effectively handle missing values and noisy data and can be used either as an explanatory tool for distinguishing observations of different classes or as a prediction tool to class labels of previously unseen observations.

Some of the well-known DTI algorithms include ID3 (Quinlan, 1983), CART (Breiman et al., 1984), C4.5 (Quinlan, 1993), SLIQ (Metha et al., 1996), FAST (Mola & Siciliano, 1997) and GUIDE (Loh, 2002). All these algorithms use a greedy, top-down recursive partitioning approach. They primarily differ in terms of the splitting criteria, the type of splits (2-way or multi-way) and the handling of the overfitting problem.

DTI uses a greedy, top-down recursive partitioning approach to induce a decision tree from data. In general, DTI involves the following tasks: decision tree growing and decision tree pruning.

2.1 Tree growing

As for the growing of a decision tree, DTI use a greedy heuristic to make a series of locally optimum decisions about which value of a splitting predictor to use for data partitioning. A

test condition depending on a splitting method is applied to partition the data into more homogeneous subgroups at each step of the greedy algorithm.
Splitting methods differ with respect to the type of splitting predictor: for nominal splitting predictors the test condition is expressed as a question about one or more of its attributes, whose outcomes are "Yes"/"No". Grouping of splitting predictor attributes is required for algorithms using 2-way splits. For ordinal or continuous splitting predictors the test condition is expressed on the basis of a threshold value υ such as $(x_i \leq \upsilon?)$ or $(x_i > \upsilon?)$. By considering all the possible split points υ, the best one υ^* partitioning the instances into homogeneous subgroups is selected.
In the classification problem, the sample population consists of N observations deriving from C response classes. A decision tree (or classifier) will break these observations into k terminal groups, and to each of these a predicted class (being one of the possible attributes of the response variable) is assigned. In actual application, most parameters are estimated from the data. In fact, denoting with t some node of the tree (t represents both a set of individuals in the sample data and, via the tree that produced it, a classification rule for future data) from the binary tree it is possible to estimate $P(t)$ and $P(i\,|\,t)$ for future observations as follows:

$$P(t) = \sum_{i=1}^{C} \pi_i P\{x \in t \,|\, \tau(x) = i\} \approx \sum_{i=1}^{C} \pi_i \left(n_{iA}/n_i\right) \tag{1}$$

$$P(i|t) = P\{\tau(x) = i \,|\, x \in t\} = \pi_i P\{x \in t \,|\, \tau(x) = i\} \Big/ P\{x \in t\} \approx \pi_i \left(n_{it}/n_i\right) \Big/ \sum_{i=1}^{C} \pi_i \left(n_{it}/n_i\right) \tag{2}$$

where π_i is the prior probability of each class i $(i \in 1,2,\ldots,C)$, $\tau(\mathbf{x})$ is the true class of an observation x_i ($\mathbf{x}$ is the vector of predictor variables), n_i and n_t are the number of observations in the sample that respectively are class i and node t, and n_{it} is the number of observations in the sample that are class i and node t.
In addition, by denoting with R the risk of misclassification, the risk of t (denoted with $R(t)$) and the risk of a model (or tree) T (denoted with $R(T)$) are measured as follows:

$$R(t) = \sum_{i=1}^{C} P(i|t) L(i, \tau(t)) \tag{3}$$

$$R(T) = \sum_{j=1}^{k} P(t_j) R(t_j) \tag{4}$$

where $L(i,j)$ is the loss matrix for incorrectly classifying an i as a j (with $L(i,i)=0$), and $\tau(t)$ is the class assigned to t once that t is a terminal node and $\tau(t)$ is chosen to minimize $R(t)$ and t_j are terminal nodes of the tree T. If $L(i,j)=1$ for all $i \neq j$, and the prior probabilities τ are set to be equal to the observed class frequencies in the sample, then $P(i\,|\,t)=n_{it}/n_t$ and $R(T)$ is the proportion of misclassified observations.
When splitting a node t into t_r and t_l (left and right sons), the following relationship holds: $P(t_l)\ R(t_l) + P(t_r)\ R(t_r) \leq P(t)\ R(t)$. An obvious way to build a tree is to chose that split maximizing ΔR, i.e., the decrease in risk. To this aim, several measures of impurity (or diversity) of a node are used. Denoting with f some impurity function, the local impurity of a node t is defined as:

$$\varepsilon(t) = \sum_{i=1}^{C} f(p_{it}) \tag{5}$$

where p_{it} is the proportion of those in t that belong to class i for future samples. Since $\varepsilon(t)=0$ when t is pure, f must be concave with $f(0)=f(1)=0$. Two candidates for f are the information index $f(p) = -p \log(p)$ and the Gini index $f(p)= -p(1-p)$, that slightly differ for the two class problem where nearly always choose the same split point. Once that f has been chosen, the split maximizing the impurity reduction is:

$$\Delta\varepsilon = p(t)\varepsilon(t) - p(t_l)\varepsilon(t_l) - p(t_r)\varepsilon(t_r) \tag{6}$$

Data partitioning proceeds recursively until a stopping rule is satisfied: this usually happens when the number of observations in a node is lower than a previously-specified minimum number necessary for splitting, as well as when the same observations belong to the same class or have the same response class.

2.2 FAST splitting algorithm

The goodness of split criterion based on (6) expresses in different way some equivalent criteria which are present in most of the tree-growing procedures implemented in specialized software; such as, for instance, CART (Breiman et al., 1984), ID3 and C4.5 (Quinlan, 1993).

In many situations the computational time required by a recursive partitioning algorithm is an important issue that can not be neglected. In this respect, a fast algorithm is required to speed up the procedure. In view of that, it is worth considering a two-stage splitting criterion which takes into account of the global role played by a splitting predictor in the partitioning step. A global impurity reduction factor of any predictor x_i is defined as:

$$\mathrm{E}_{y|x_s}(t) = \sum_{g \in G_s} \varepsilon_{y|g}(t)\, p(g \mid t) \tag{7}$$

where $\varepsilon_{y|g}(t)$ is the impurity of the conditional distribution of y given the s-th attribute of x_s and G is the number of attributes of x_s $(g \in G)$. The two-stage criterion finds the best splitting predictor(s) as the one (or those) minimizing (7) and, consequently, the best split point among the candidate splits induced by the best predictor(s) minimizing the (6) by taking account only the partitions or splits generated by the best predictor. This criterion can be applied either *sic et simpliciter* or by considering alternative modelling strategies in the predictor selection (an overview of the two-stage methodology can be found in Siciliano & Mola, 2000).

The FAST splitting algorithm (Mola & Siciliano, 1997) can be applied when the following property holds for the impurity measure:

$$\mathrm{E}_{y|x_s}(t) \le \mathrm{E}_{y|h}(t) \qquad \forall\, h \neq g;\; h \in G \tag{8}$$

and it consists of two basic rules:

- iterate the two-stage partitioning criterion by using (7) and (6): select one splitting predictor at a time and consider, at each time, the previously unselected splitting predictors;

- stop the iterations when the current best predictor in the order *x(k)* at iteration *k* does not satisfy the condition $\mathrm{E}_{y|x_k}(t) \leq \mathrm{E}_{y|h^*_{(k-1)}}(t)$, where $s^*_{(k-1)}$ is the best partition at the iteration $(k-1)$.

The algorithm finds the optimal split with substantial time savings in terms of the reduced number of partitions or splits to be tried out at each node of the tree. Simulation studies show that the relative reduction in the average number of splits analyzed by the FAST algorithm with respect to the standard approaches in binary trees increases as a function of both the number of attributes of the splitting predictor and of the number of observations at a given node. Further theoretical results about the computational efficiency of FAST-like algorithms can be found in Klaschka et al. (1998).

2.3 Tree pruning

As for the pruning step, it is usually required in DTI in order to control for the size of the induced model and to avoid in this way data overfitting. Typically, data is partitioned into a training set (containing two-third of the data) and a test set (with the remaining one-third). Training set contains labelled observations and it is used for the tree growing. It is assumed that the test set contains unlabelled observations and it is used for selecting the final decision tree: to check whether a decision tree, say *T*, is generalizable, it is necessary to evaluate its performance on the test set in terms of misclassification error by comparing the true class labels of the test data against those predicted by *T*. Reduced-size trees perform poorly on both training and test sets causing underfitting. Instead, increasing the size of *T* improves both the training and test errors up to a "critical size" from which the test errors increase even though the corresponding training errors decrease. This means that *T* overfits the data and cannot be generalized to class prediction of unseen observations. In the machine learning framework, the training error is named resubstitution error and the test error is known as the generalization error.
It is possible to prevent overfitting by haltering the tree growing before it becomes too complex (pre-pruning). In this framework, one can assume the training data is a good representation of the overall data and use the resubstitution error as an optimistic estimate of the error of the final DTI model (optimistic approach). Alternatively, Quinlan (1987) proposed a pessimistic approach that penalizes complicated models by assigning a cost penalty to each terminal node of the decision tree: for C4.5, the generalization error is $R(t)/n_t+\varepsilon$, where, for a node t, n_t is the number of observations and $R(t)$ is the misclassification error. It is assumed that $R(t)$ follows a Binomial distribution and that ε is the upper bound for $R(t)$ computed from such a distribution (Quinlan, 1993).
An alternative pruning strategy is based on the growing of the entire tree and the subsequent retrospective trimming of some of its internal nodes (post-pruning): the subtree departing from each internal node is replaced with a new terminal node whose class label derives from the majority class of observations belonging to that subtree. The latter is definitively replaced by the terminal node if such a replacement induces an improvement of the generalization error. Pruning stops when no further improvements can be achieved. The generalization error can be estimated through either the optimistic or pessimistic approaches.
Other post-pruning algorithms, such as CART, use a complexity measure that accounts for both the tree size and the generalization error. Once the entire tree is grown using training

observations, a penalty parameter expressing the gain/cost trade off for trimming each subtree is used to generate a sequence of pruned trees, and the tree in the sequence presenting the lowest generalization error (0-SE rule) or the one with a generalization error within one standard error of its minimum (1-SE rule) is selected. Let α be a number in $[0,+\infty]$, called complexity parameter, measuring the "cost" of adding another variable to the model. Let $R(T_0)$ be the risk for the zero split tree. Define:

$$R_\alpha(T) = R(T) + \alpha|T| \tag{9}$$

to be the cost for the tree, and define T_α to be that subtree of the entire tree having the minimal cost. Obviously, T_0 is the entire tree and T_∞ is the zero splits model. The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ minimizing $R_\alpha(T)$. The tuning parameter $\alpha \geq 0$ governs the trade off between the tree size and its goodness of fit to the data. Large values of α result in small trees, and conversely for smaller values of α. Of course, with $\alpha=0$ the solution is the full tree T_0. It is worth noticing that, by adaptively choosing αI, it exists a unique smallest subtree T_α minimizing $R_\alpha(T)$. A weakest link pruning approach is used to find T_α: it consists in successively collapsing the internal node producing the smallest per-node increase in $R(T)$, continuing this way until the single-node (root) tree is produced. This gives a (finite) sequence of subtrees, and it is easy to show that this sequence must contains T_α (see Breiman et al (1984) for details).
Usually, pruning algorithms can be combined with V-fold cross-validation when few observations are available. Training data is divided into V disjoint blocks and a tree is grown V times on V-1 blocks estimating the error by testing the model on the remaining block. In this case, the generalization error is the average error made for the V runs. The estimation of αI is achieved by V-fold cross-validation: the final choice is the $\hat{\alpha}$ minimizing the cross-validated $R(T)$ and the final tree is $T_{\hat{\alpha}}$.

Cappelli et al. (2002) improved this approach introducing a statistical testing pruning to achieve the most reliable decision rule from a sequence of pruned trees.

2.4 Regression tree

In the case the response variable is numeric, the outcome of a recursive partitioning algorithm is regression tree. Here, the splitting criterion is SS_t- $(SS_l$ - $SS_r)$, where SS_t is the residual sum of squares for the parent node, and SS_l and SS_r are the residual sum of squares for the left and right son, respectively. This is equivalent to choosing the splits maximizing the between-groups sum-of-squares in a simple analysis of variance. In each terminal node, the mean value of the response variable μ_y of cases belonging to that node is considered as the fitted value whereas the variance is considered as an indicator of the error of a node. For a new observation y_{new} the prediction error is $(y_{new}$ - $\mu_y)$. In the regression tree case, cost-complexity pruning is applied with the sum of squares replacing the misclassification error.

2.5 DTI enhancements

A consolidated literature about the incorporation of parametric and nonparametric models into trees appeared in recent years. Several algorithms have been introduced as hybrid or functional trees (Gama, 2004), among the machine learning community. As an example, DTI is used for regression smoothing purposes in Conversano (2002): a novel class of

semiparametric models named Generalized Additive Multi-Mixture Models (GAM-MM). Other hybrid approaches are presented in Chan and Loh (2004), Su et al. (2004), Choi et al. (2005) and Hothorn et al. (2006). Nevertheless, relatively simple procedures combining DTI models in different ways have been proposed in the last decade in the statistics and machine learning literature and their effectiveness in improving the predictive ability of the traditional DTI method has been proven in different fields of application.

The first, rather intuitive, approach is Tree Averaging. It is based on the generation of a set of candidate trees and on their subsequent aggregation in order to improve their generalization ability. It requires the definition of a suitable set of trees and their associated weights and classifies a new observation by averaging over the set of weighted trees (Oliver and Hand, 1995). Either a compromise rule or a consensus rule can be used for averaging.

An alternative method consists in summarizing the information of each tree in a table cross-classifying terminal nodes outcomes with the response classes in order to assess the generalization ability through a statistical index and select the tree providing the maximum value of such index (Siciliano, 1998).

Tree Averaging is very similar to Ensemble methods. These are based on a weighted or non weighted aggregation of single trees (the so called weak learners) in order to improve the overall generalization error induced by each single tree. They are more accurate than a single tree if they have a generalization error that is lower than random guessing and if the generalization errors of the different trees are uncorrelated (Dietterich, 2000).

A first example of Ensemble method is Bootstrap Aggregating, which is also called Bagging (Breiman, 1996). It works by randomly replicating the training observations in order to induce single trees whose aggregation by majority voting provides the final classification. Bagging is able to improve the performance of unstable classifiers (i.e. trees with high variance). Thus, bagging is said to be a reduction variance method.

Adaptive Boosting, also called AdaBoost (Freud & Schapire, 1996) is an Ensemble method that uses iteratively bootstrap replication of the training instances. At each iteration, previously-misclassified observations receive higher probability of being sampled. The final classification is obtained by majority voting. Boosting forces the decision tree to learn by its error, and is able to improve the performance of trees with both high bias (such as single-split trees) and variance.

Finally, Random Forest (Breiman, 2001) is an ensemble of unpruned trees obtained by randomly resampling training observations and variables. The overall performance of the method derives from averaging the generalization errors obtained in each run. Simultaneously, suitable measures of variables importance are obtained to enrich the interpretation of the model.

3. Combinatorial optimization

Combinatorial Optimization can be defined as the analysis and solution of problems that can be mathematically modelled as the minimization (or maximization) of an objective function over a feasible space involving mutually exclusive, logical constraints. Such logical constraints can be seen as the arrangement of a bunch of given elements into sets. In a mathematical form:

$$\min_{T \in F} \{\alpha(T)\} \quad \text{or} \quad \max_{T \in F} \{\alpha(T)\} \tag{10}$$

where T can be seen as an arrangement, F is the collection of feasible arrangements and $\alpha(T)$ measures the value of the members of F.
Combinatorial Optimization problems are of great interest because many real life decision-making situations force people to choose over a set of possible alternatives with the aim of maximizing some utility function. On the one hand, the discreteness of the solutions space offers the great advantage of concreteness and, indeed, elementary graphs or similar illustrations can often naturally be used to represent the meaning of a particular solution to a problem. On the other end, those problems carry a heavy burden in terms of dimensionality. If more than few choices are to be made, the decision-making process has to face with the evaluation of a terribly big expanse of cases. This dualism (intuitive simplicity of presentation of a solution versus complexity of solutions search) has made this area of combinatorics attractive for researchers from many fields, ranging from engineering to management sciences.
Elegant procedures to find optimal solutions have been found for some problems, but for most of them only a bunch of properties and algorithms have been developed that still do not allow to reach a complete resolution. This is the case of Computational Statistics, in which computationally-intensive methods are used to "mine" large, heterogeneous, multi-dimensional datasets in order to discover knowledge in the data.
To give an example, the objective of Cluster Analysis is to find the "best" partition of the dataset according to some criterion, which is always expressed as an objective function. This means that all possible and coherent partitions of the dataset should be generated and the objective function has to be calculated for each of them. In many cases, the number of possible partitions grows too rapidly with respect to the number of units, making such strategy practically unfeasible. Another example is the apparently simple problem of calculating the variance for interval data, for which the maximum and the minimum of the variance function have to be searched over the multidimensional cube defined by all the intervals in which the statistical units are defined.
These are examples of statistical problems that cannot be faced with the total enumeration and evaluation of the solutions. In order to try to tackle with this kind of problems, a lot of theory has been developed. One case is when some properties about the objective function are available. These allow to calculate some kind of upper (or lower) bound that a set of possible solutions could admit. In this case, the search could be performed just on the set of possible solutions whose upper bound is higher. If one solution whose effective value is higher than the bounds of all the other sets is found, it would not be necessary to continue the search, being all the other subsets not able to provide better solutions. This is the case of the aforementioned problem of finding the upper bound of variance for interval data, because it can be verified that the maximum is necessarily reached in one of the vertices of the multidimensional cube, so that exploring the whole cube is not necessary. Such a situation allows to restrict the solutions space to a set of 2^n possible solutions, where n is the number of statistical units. Unfortunately, this does not solve the problem because the solutions space becomes enormous even in presence of small datasets (with just 30 units the number of solutions to evaluate is greater than one thousand millions).
The FAST algorithm is another example of a partial enumeration approach, in which a measure of the upper bound of the predictive power of a solutions set is defined and exploited in order to get the same results of the CART greedy approach by using a reduced amount of computations.

Another way to proceed is to make use of non exact procedures, often called heuristics. Those algorithms do not claim to find the global optimum, but are able to converge rapidly towards a local one. Non exact algorithms (that will be called heuristics in the rest of this chapter) are certainly not recent. What has changed, in time, is the respectability associated to them, due to the fact that many heuristics have been proved to rival their counterparts in elegance, sophistication and, particularly, usefulness. Many heuristics have been proposed in the literature, but only two kinds of them will be briefly described in this context due to their role in the problems that will be faced in the next sections. These are: Greedy procedures and Nature Inspired optimization algorithms. In Greedy procedures the optimization process selects, at each stage, an alternative that is the best among all the feasible alternatives without taking into account the impact that such choice will have on the subsequent decisions. The CART algorithm makes use of a greedy procedure to grow a tree in which the optimality criterion is maximised just locally, that is, for each node of the tree but not considering the tree as a whole. This approach clearly results in a suboptimal tree but allows, at least, to obtain a tree in a reasonable amount of time. Whereas, the so-called Nature Inspired heuristics, which are also called "Heuristics from Nature" (Colorni et al., 1993), are Inspired by natural phenomena or behaviour such as Evolution, Ants, Honey-Bees, Immune systems, Forests, etc. Some important Nature Inspired heuristics are: Simulated Annealing (SA), TABU Search (TS) algorithms, Ant Colony Optimization (ACO) and Evolutionary Computation (EC). ACO and EC are described in the following since they are used throughout the chapter.

Ant Colony Optimization represents a class of algorithms that were inspired by the observation of real ant colonies. Observation shows that a single ant only applies simple rules, has no knowledge and it is unable to succeed in anything when it is alone. However, an ant colony benefits from the coordinated interaction of each ant. Its structured behaviour, described as a "social life", leads to a cooperation of independent searches with high probability of success. ACO were initially proposed by Dorigo (1992) to attack the Traveling Salesman Problem. A real ant colony is capable of finding the shortest path from a food source to its nest by using pheromone information: when walking, each ant deposits a chemical substance called pheromone and follows, in probability, a pheromone trail already deposited by previous ants. Assuming that each ant has the same speed, the path which ends up with the maximum quantity of pheromone is the shortest one.

Evolutionary computation (Fogel and Fogel, 1993) incorporates algorithms that are inspired from evolution principles in nature. The methods of evolutionary computation algorithms are stochastic and their search methods imitate and model some natural phenomena, namely:

1. the survival of the fittest
2. genetic inheritance

Evolutionary computing can be applied to problems when it is difficult to apply traditional methods (e.g., when gradients are not available) or when traditional methods lead to unsatisfactory solutions like local optima (Fogel, 1997). Evolutionary algorithms work with a population of potential solutions (i.e. individuals). Each individual is a potential solution to the problem under consideration and it is encoded into a data structure suitable to the problem. Each encoded solution is evaluated by an objective function (environment) in order to measure its fitness. The bias on selecting high-fitness individuals exploits the acquired fitness information. The individuals will change and evolve to form a new

population by applying genetic operators. Genetic operators perturb those individuals in order to explore the search space. There are two main types of genetic operators: Mutation and Crossover. Mutation type operators are asexual (unary) operators, which create new individuals by a small change in a single individual. On the other hand, Crossover type operators are multi-sexual (multary) operators, which create new individuals by combining parts from two or more individuals. As soon as a number of generations have evolved, the process is terminated according to a termination criterion. The best individual in the final step of the process is then proposed as a (hopefully suboptimal or optimal) solution for the problem.

Evolutionary computing are further classified into four groups: Genetic Algorithms (GA), Evolutionary Programming, Evolution Strategies and Genetic Programming. Although there are many relevant similarities between these evolutionary computing paradigms, profound differences among them also emerge (Michalewicz, 1996). These differences generally involve the level in the hierarchy of the evolution being modelled, that is: the chromosome, the individual or the species. There are also many hybrid methods that combine various features from two or more of the methods described in this section.

Genetic Algorithms (GAs), that will be used in the follwing, are part of a collection of stochastic optimization algorithms inspired by the natural genetics and the theory of the biological evolution. The idea behind genetic algorithms is to simulate the natural evolution when optimizing a particular objective function. GAs have emerged as practical, robust optimization and search methods in the last three decades. In the literature, Hollands' genetic algorithm is called Simple Genetic Algorithm (Vose, 1999). It works with a population of individuals (chromosomes), which are encoded as binary strings (genes).

4. Genetic algorithms and heuristics in DTI

4.1 Genetic algorithm for complex predictors

The CART methodology looks for the best split by making use of a brute-force (enumerative) procedure. All the possible splits from all the possible variables are generated and evaluated. Such a procedure must be performed anytime a node has to be split and can lead to computational problems when the number of modalities grows.

Let us first consider how a segmentation procedure generates and evaluates all possible splits. Nominal unordered predictors (Nup) are more complicated to handle than ordered ones because the number of possible splits that can be generated grows exponentially with the number of attributes m. The number of possible splits is $(2^{m-1}-1)$. The computational complexity of a procedure that generates and evaluates all the splits from a nominal unordered predictor is $O(2^n)$. In this respect, it is evident that such enumerative algorithm becomes prohibitive when the number of attributes is high. This is one of the reasons why some software do not accept Nups with a number of attributes higher than a certain threshold (usually between 12 and 15).

One of the possible way to proceed is to make use of a heuristic procedure, like the one proposed in this section. In order to design a Genetic Algorithm to solve such a combinatorial problem, it is necessary to identify:

- a meaningful representation (coding) for the candidate solutions (the possible splits)
- a way to generate the initial population
- a fitness function to evaluate any candidate solution

- a set of useful genetic operators that can efficiently recombine and mutate the candidate solutions
- the values of the parameters used by the GA (population size, genetic operators parameters values, selective pressure, etc.);
- a stopping rule for the algorithm.

The aforementioned points have been tackled as follows. As for the coding, it has been chosen the following representation: a solution is coded in a string of bits (chromosomes) called x, where each bit (gene) is associated to an attribute of the predictor according to the following rule:

$$x_i = \begin{cases} 0 & if \quad i \quad goes \quad to \quad left \\ 1 & if \quad i \quad goes \quad to \quad right \end{cases} \tag{11}$$

The choice of the fitness function is straightforward: the split evaluation function of the standard recursive partitioning algorithm is used (i.e. the maximum decrease in node impurity). Since the canonical (binary) coding is chosen, the corresponding two parents single-point crossover and mutation operators and, as a stopping rule can be used. In addition, a maximum number of iterations is chosen on the basis of empirical investigations. The rest of the GA features are similar to the classic ones: elitism is used (at each iteration the best solution is kept in memory) and the initial population is chosen randomly.

4.2 An ACO algorithm for exploratory DTI

When growing a Classification or a Regression Tree, CART first grows the so-called exploratory tree. Such tree is grown using data of the training set. Then, it is validated by using the test set or by cross-validation.

In this section, the attention is focused on the exploratory tree-growing procedure. In this phase, in theory, the best possible tree should be built, which is the tree having the lowest global impurity measure among all the generable trees. It has been shown (Hyafil and Rivest, 1976) that constructing the optimal tree is a NP-Complete problem. In other words, in order to use a polynomial algorithm, it is only possible to get suboptimal trees. For such a reason, the recursive partitioning algorithms make use of greedy heuristics to reach a compromise between the tree quality and the computational effort. In particular, most of the existing methods for DTI use a greedy heuristic, which is based on a top-down recursive partitioning approach in which, any time, the split that maximizes the one step impurity decrease is chosen. This kind of greedy approach, that splits the data locally (i.e., in a given node) and only once for each node, allows to grow a tree in a reasonable amount of time. On the other hand, this rule is able to generate only a suboptimal tree because anytime a split is chosen a certain subspace of possible trees is not investigated anymore by the algorithm. If the optimal tree is included in one of those subspaces there is no chance for the algorithm of finding it.

Taking these considerations into account, we propose an Ant Colony Optimization algorithm to try to find best exploratory tree. In order to attack a problem with ACO the following design task must be performed:

1. Represent the problem in the form of sets of components and transitions or by means of a weighted graph, on which ants build solutions

2. Appropriately define the meaning of the pheromone trails: that is, the type of decision they bias.
3. Appropriately define the heuristic reference for each decision an ant has to take while constructing a solution.
4. If possible, implement an efficient local search algorithm for the problem to be solved. The best results from the application of the ACO algorithms to NP-hard combinatorial optimization problems are achieved by coupling ACO with local optimizers (Dorigo and Stutzle, 2004)
5. Choose a specific ACO algorithm and apply it to the problem to be solved, taking the previous issues into account
6. Tune the parameters of the ACO algorithm. A good starting point is to use parameter settings that were found to be good when applying the same ACO algorithm to similar problems or to a variety of other problems

The most complex task is probably the first one, in which a way to represent the problem in the form of a weighted graph must be found. We use a representation based on the following idea: let us imagine having two nominal predictors $P_1 = \{a_1, b_1, c_1\}$ and $P_2 = \{a_2, b_2\}$ with, respectively, two and three attributes. Such simple predictors are considered only to explain the idea, because of the combinatorial explosion of the phenomenon. In this case, the set of all possible splits, at a root node, is the following:

- $S_1 = [a_1] - [b_1, c_1]$
- $S_2 = [a_1, b_1] - [c_1]$
- $S_3 = [a_1, c_1] - [b_1]$
- $S_4 = [a_2] - [b_2]$

Any time a split is chosen, it generates two child nodes. For such nodes, the set of possible splits is, in the worst case, equal to 3 (the same as the parent node except the one that was chosen for splitting). This consideration leads to the representation shown in Figure 1 in which, for simplicity, only the first two levels of the possible trees are considered.

It is easy to imagine how the complexity grows when we deal with predictors that generate hundreds or even thousands of splits (which is a common case).

In Figure 1, the space of all possible trees is represented by a connected graph. Moving from a level to another one corresponds to split a variable. The arcs of such a graph have the same meaning of the arcs of the TSP graph (transition from a state to another one or, even better, addition of a component to a partial solution). In this view, it would be correct to deposit pheromone on them. The pheromone trails meaning, in this case, corresponds to the desirability to choose the corresponding split from a certain node.

As for the heuristic information, it is possible to refer to the decrease in impurity deriving from adding the corresponding node to the tree. Such a measure has a meaning which is similar, in some way, to the one that visibility has in the TSP . An arc is much more desirable as higher the impurity decrease is. As a result, to make analogies with the TSP, such impurity decrease can be seen as an inverse measure of the distance between two nodes.

Once the construction graph has been built, and pheromone trails meaning and heuristic function have been defined, it is possible to attack that problem using an ACO algorithm. It is important to note that, because of the specificity of the problem to be modelled (ants can move into a connected graph and there is a measure of “visibility”), the search of the best tree can be seen as a shortest path research, like in TSP. In the latter, ants are forced to pass only one time for each city while, in our case, ants are forced to choose paths that

correspond to binary trees, since the solutions to build must be in the form of tree structures. All the ants will start from the root node and will be forced to move from one node to another in order to build a tour that corresponds to a tree.

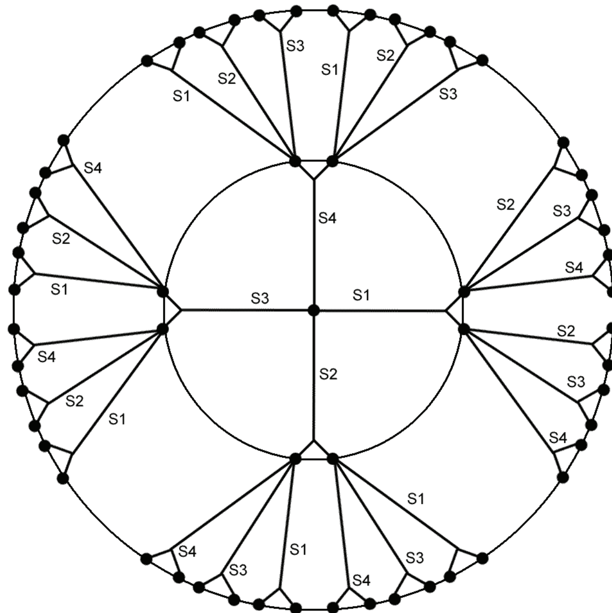

Fig. 1. An example of ACO algorithm for exploratory DTI: each path corresponds to a 2-levels tree.

It is important to notice the basics of the ant moves in the graph shown in Figure 1. At each step, the ant looks for the heuristic information (impurity decrease) and the pheromone trail of any possible direction and decides for the one to choose (and, therefore, the associated split) on the basis of the selected ACO algorithm. Once the ant arrives to a terminal node, it recursively starts to move back to the other unexplored nodes.

In different ACO algorithms, pheromone trails are initialized to a value obtained by manipulating the quality measure (the path's length for the TSP case) of a solution obtained with another heuristic (Dorigo suggests the nearest-neighbour heuristic). In our case, the greedy tree induction rule solution quality is used. Elitism will also be implemented and the chosen parameters (due to the strong similarity with TSP) are the same that have been used successfully for the TSP problem.

4.3 Identification of a parsimonious set of decision trees in multi-class classification

In many situations, the response variable used in classification tree modelling rarely presents a number of attributes that allow to apply the recursive partitioning algorithm in the most accurate manner.

It is well known that:

a) a multi-class response, namely a nominal variables with several classes, usually causes prediction inaccuracy;

b) multi-class and numeric predictors play often the role of splitting variables in the tree growing process in disadvantage of two-classes ones, causing selection bias.

To account for the problems deriving from the prediction inaccuracy of tree-based classifiers grown for multi-class response, as well as to reduce the drawback of the loss of interpretability induced by ensemble methods in these situations, Mola and Conversano (2008) introduced an algorithm based on a *Sequential Automatic Search of a Subset of Classifiers* (SASSC). It produces a partition of the set of the response classes into a reduced number of disjoint subgroups and introduces a parameter in the final classification model that improves its prediction accuracy, since it allows to assign each new observation to the most appropriate classifier in a previously-identified reduced set of classifiers. It uses a data-driven heuristic based on cross-validated classification trees as a tool to induce the set of classifiers in the final classification model.

SASSC produces a partition of the set of the response classes into a reduced number of *super-classes*. It is applicable to a dataset **X** composed of N observations characterized by a set of J (numeric or nominal) splitting variables x_j $(j=1,\ldots,J)$ and a response variable y presenting K classes. Such response classes identify the initial set of classes $C^{(0)}=(c_1,c_2,\ldots,c_K)$. Partitioning **X** with respect to $C^{(0)}$ allows to identify K disjoint subsets $\mathbf{X}^{(0)}_k$, such that: $\mathbf{X}^{(0)}_k = \{\mathbf{x}_s : y_s \in c_k\}$, with $s=1,\ldots,N$. In practice, $\mathbf{X}^{(0)}_k$ is the set of observations presenting the k-th class of y. The algorithms works by aggregating the K classes in pairs and learns a classifier to each subset of corresponding observations. The "best" aggregation (*super-class*) is chosen as the one minimizing the generalization error estimated using V-fold cross-validation.

Suppose that, in the ℓ-th iteration of the algorithm such a best aggregation is found for the pair of classes c_{i^*} and c_{j^*} (with $i^* \neq j$ and $i^*, j^* \in (1,\ldots,K)$) that allows to aggregate the subsets $\mathbf{X}_{i^*}$ and $\mathbf{X}_{j^*}$. Denoting with $T_{(i^*,j^*)}$ the decision tree minimizing the cross-validated generalization error $\delta^{()}_{cv}$, the heuristic for selecting the "best" decision tree can be formalized as follows:

$$\left(i^*, j^*\right) = \underset{(i,j)}{\arg\min}\left\{\delta_{cv}^{(\ell)}\left(T_{(i,j)} \mid \mathbf{X}_i \cap \mathbf{X}_j\right)\right\} \tag{12}$$

The SACCS algorithm is analytically described in Table 1. It proceeds by learning all the possible decision trees obtainable by joining in pairs the K subgroups, and by retaining the one satisfying the selection criteria introduced in (12). After the ℓ-th aggregation, the number of subgroups is reduced to $K^{(-1)} - 1$, since the subgroups of observations presenting the response classes c_{i^*} and c_{j^*} are discarded from the original partition and replaced by the subset $\mathbf{X}^{()}_{(i^*,j^*)} = \mathbf{X}_{(i^*)} \cap \mathbf{X}_{(j^*)}$ identified by the super-class $c^{()} = (c_{(i^*)} \cap c_{(j^*)})$. The initial set of classes C is replaced by $C^{()}$, the latter being composed of a reduced number of classes since some of the original classes form the superclasses coming out from the ℓ aggregations. Likewise, also $\mathbf{X}^{()}_k$ is formed by a lower number of subsets as a consequence of the ℓ aggregations.

The algorithm proceeds sequentially in the iteration $\ell+1$ by searching for the most accurate decision tree over all the possible ones obtainable by joining in pairs the $K^{()}$ subgroups. The sequential search is repeated until the number of subgroups reduces to one in the K-th

iteration. The decision tree learned on the last subgroup corresponds to the one obtainable applying the recursive partitioning algorithm on the original dataset.
The output of the procedure is a sequence of sets of response classes $C^{(1)},\ldots,C^{(K-1)}$ with the associated sets of decision trees $T_{(1)},\ldots,T^{(K-1)}$. The latter are derived by learning $K-k$ trees $(k = 1, \ldots, K-1)$ on disjoint subgroups of observations whose response classes complete the initial set of classes $C^{(0)}$: these response classes identify the super-classes relating to the sets of classifiers $T_{(k)}$. An overall generalization error is associated to each $T_{(k)}$: such an error is also based on V-fold cross-validation and it is computed as a weighted average of the generalization errors obtained from each of the $K-k$ decision trees composing the set. In accordance to the previously introduced notation, the overall generalization errors can be denoted as $\Theta^{(1)}_{cv}, \ldots, \Theta^{(k)}_{cv}, \ldots, \Theta^{(K-1)}_{cv}$. Of course, by decreasing the number of trees composing a sequence $T_{(k)}$ (that is, when moving k from 1 to $K-1$) the corresponding $\Theta^{(k)}_{cv}$ increases since the number of super-classes associated to $T_{(k)}$ is also decreasing. This means that a lower number of trees are learned on more heterogeneous subsets of observations, since each of those subsets pertain to a relatively large number of response classes.

Input:	$C=\{c_1,\ldots,c_K\}_{c_i\cap c_j=\varnothing;i\neq j;i,j\in(1,\ldots,K)}$
Set:	$C^{(0)}=C;\quad K^{(0)}=K;\quad \mathbf{X}_k^{(0)}=\{\mathbf{x}_s : y_s\in c_k\}_{s=1,\ldots,N;k=1,\ldots,K}$
For: ℓ **in** 1 **to** K	
	$c^{(\ell)}=\{c_{i*}\cap c_{j*}\}:\theta_{cv}^{(\ell)}\left(T_{(i*,j*)}\mid \mathbf{X}_{i*}\cap\mathbf{X}_{j*}\right)=\min$
	$K^{(\ell)}=K^{(\ell-1)}-1$
	$C^{(\ell)}=\left\{c_1,\ldots,c_{K^{(\ell)}-2+1}=c^{(\ell)}\right\}$
	$\mathbf{X}_k^{(\ell)}=\{\mathbf{x}_s : y_s\in c_k\}_{k=1,\ldots,K^{(\ell)}-1}$
end For	
Output:	$C^{(1)},\ldots,C^{(K-1)};\quad T_{(1)},\ldots,T_{(K-1)};\quad \Theta_{cv}^{(1)},\ldots,\Theta_{cv}^{(K-1)}$

Table 1. The SASSC algorithm

Taking this inverse relationship into account, the analyst can be aware of the overall prediction accuracy of the final model on the basis of the relative increase in $\Theta^{(k)}_{cv}$ when moving from 1 to $K-1$. In this respect, he can select the suitable number of decision trees to be included in the final classification model accordingly. Supposing that a final subset of g decision trees has been selected $(g \ll K-1)$, the estimated classification model can be represented as:

$$\hat{f}(\mathbf{x})=\sum_{i=1}^{g-1}\sum_{m_i=1}^{M_i}\hat{\psi}_i\hat{c}_{k,i}I\left(\left(x_1,\ldots,x_p\right)\in R_{m_i}\right) \qquad (13)$$

The parameter ψ is called "vehicle parameter". It allows to assign a new observation to the most suitable decision tree in the subset g. It is defined by a set of $g-1$ dummy variables. Each of them equals 1 if the object belongs to the i-th decision tree ($i = 1,\ldots, g-1$) and zero otherwise. The M_i regions, corresponding to the number of terminal nodes of the decision

tree i, are created by splits on predictors $(x_1,\ldots,x_p)$. The classification tree i assigns a new observation to the class $\hat{c}_{k,i}$ of y according to the region R_{m_i}. I is an indicator function with value 1 if an observation belongs to R_{m_i} and value 0 if not. R_{m_i} is defined by the inputs used in the splits leading to that terminal node. The modal class of the observations in a region R_{m_i} (also called the m-th terminal node of the i-th decision tree) is usually taken as an estimate for $\hat{c}_{k,i}$. This notation is consistent with that used in Hastie et al. (2001).

The estimation of τ_i is based on the prediction accuracy of each decision tree in the final subset g. A new observation is slipped into each of the g trees. The assigned class $\hat{c}_{k,i}$ is found with respect to the tree whose terminal node better classifies the new observation. In other words, a new observation is assigned to the purest terminal node among all the g decision trees.

Another option of the algorithm is the possibility to learn decision trees to select the suitable pair of response classes satisfying (12) using alternative splitting criteria. As for CART, it is possible to refer to both the Gini index and Twoing as alternative splitting rules. It is known that, unlike Gini rule, Twoing searches for the two classes that make up together more than 50% of the data and allows us to build more balanced trees even if the resulting recursive partitioning algorithm works slower. As an example, if the total number of classes is equal to K, Twoing uses 2^{K-1} possible splits. Since it has been proved (Breiman et al., 1984, pag.95) that the decision tree is insensitive to the choice of the splitting rule, it can be interesting to see how it works in a framework characterized by the search of the most accurate decision treers like the one introduced in SASSC.

5. Application on real and simulated datasets

Genetic Algorithm. The proposed GA has been applied on two datasets for which the optimal best split could be calculated and for a more complex one, for which it is not possible to proceed with such a brute force strategy.

The first test has been done on the "Mushroom" dataset, available from the UCI Machine Learning Repository (source http://archive.ics.uci.edu/ml/). This dataset has a two-class response variable ("is the mushroom poisonous?") and set of categorical and numerical predictors. One of them (gill colour) has 12 categories (attributes), which can be evaluated exhaustively. The GA algorithm could find the global best solution (which was extracted by using the Rpart package of the R software) in less than 10 iterations. The algorithm has then been tested on a simulated dataset which was obtained by uniformly generating a response variable with 26 modalities and a nominal unordered predictor with 16 modalities for 20,000 observations. By letting be 16 the number of modalities of the splitting predictor it was possible, also in this case, to find the (global) best split by making use of the exhaustive enumeration. Such experimental studies showed that the most efficient configuration of the GA was the following:

- By randomly selecting the initial population (no other solutions have been tried, in fact).
- By setting the number of solutions building the population to be equal to the number of necessary genes (the number of categories of the predictor).
- By setting a crossover proportion of 0.80.
- By setting a mutation probability equal to 0.10.
- By selecting the rank for choosing the solutions to be recombined.

For this kind of problem (20,000 units, 16 categories for the response variable and 26 categories for the splitting predictor) the global optimum was reached in less than 30 iterations.
When the complexity of the problem grows many iterations seems to be required, though such number never appeared to grow exponentially.
The GA has been tested also on the "Adult" dataset available from the UCI Machine Learning website. This dataset has been extracted from the US Census Bureau Database (source: http://www.census.gov/) with the aim of predicting whether a person earns more than 50,000 dollars per year. Such dataset has 325,614 observations and some categorical unordered splitting predictors with many attributes. In particular, the native-country predictor has 42 attributes.

State	Goes to	State	Goes to
United-States	Left	Cuba	Left
Jamaica	Right	India	Left
Unknown Country	Left	Mexico	Right
South	Left	Puerto-Rico	Right
Honduras	Right	England	Left
Canada	Left	Germany	Left
Iran	Left	Philippines	Left
Italy	Left	Poland	Left
Columbia	Right	Cambodia	Left
Thailand	Left	Ecuador	Right
Laos	Right	Taiwan	Left
Haiti	Right	Portugal	Right
Dominican-Republic	Right	El-Salvador	Right
France	Left	Guatemala	Right
China	Left	Japan	Left
Yugoslavia	Left	Peru	Right
Outlying-US	Right	Scotland	Left
Trinadad-Tobago	Right	Greece	Left
Nicaragua	Right	Vietnam	Right
Hong	Left	Ireland	Left
Hungary	Left	Holland-Netherlands	Right

Table 2. The split provided by the GA for the native-country in the Adult dataset

The GA has been run with the aim of trying to find a good split by making use of the native-country splitting predictor that both R and SPSS, for instance, refused to process. As previously mentioned, 30 iterations seemed to be not enough because, in many runs of the algorithm, the "probably best" solution appeared after iteration 80. The solution provided by the algorithm is shown in Table 2. It gives an idea of the complexity of the problem.
The corresponding decrease in the node impurity is 0.3628465. The algorithm has been tested over many simulated dataset and the number of required iterations for the algorithm to reach convergence has been shown to linearly grow as a function of the number of attributes of the splitting predictor (the number of observations in the dataset appeared to be uninfluential).

Ant System. The strong complexity of the decision tree growing procedure (Hyafil & Rivest, 1976) does not allow to exhaustively enumerate and evaluate all the possible generable trees, even from very small datasets. In this respect, it is not possible to check whether the chosen heuristic is able to find the global optimum (in the same manner as it has been previously done for the genetic algorithm).
In the first experiment the algorithm has been tested on a simulated dataset of 500 observations with 11 nominal unordered predictors (with a number of attributes that ranges between 2 and 9) and 2 numeric (continuous) predictors. It could be seen that, when the required tree depth increases, the differences between the global impurity of the tree obtained by the CART greedy heuristic and the one obtained by the Ant System tend to increase. Table 3 reports such results.

Tree Depth	CART	Ant System
4	0.158119	0.153846
5	0.147435	0.121794
6	0.100427	0.085477
7	0.079059	0.059829
8	0.044871	0.029911

Table 3.Global impurity of the decision trees extracted by the proposed algorithm on a simulated dataset

Figure 2 shows the result obtained on the “Credit” dataset that can be found in the SPAD software (source: www.spadsoft.com). This dataset has 468 observations on which 11 nominal variables have been observed together with a two-class response variable. The aim would be to predict such response variable (“is a customer good or bad?).
The first decision tree is the one found by the CART heuristic and the second one has been extracted after 200 iterations of the Ant System algorithm.
Table 4 shows the global impurity of the trees extracted by the CART and Ant heuristics.

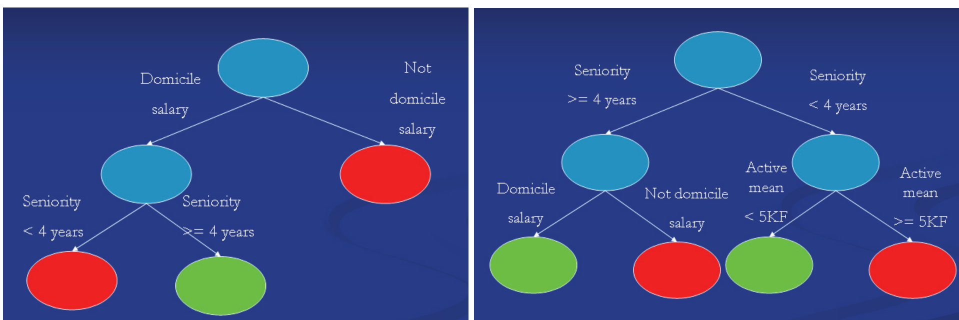

Fig. 2. Decision Trees for the Credit dataset obtained using the CART heuristic (left panel) and after 200 iterations of the Ant System algorithm (right panel).

The algorithms presented here are in an early stage of development. In these examples, an Ant System has been proposed to attack the problem of finding the best exploratory decision tree and it came out that the Ant System-based decision trees performed better than the ones found by the CART greedy heuristic. Even if the improvements weren’t too large

(from 2% to 5% in all of the simulation studies) such algorithm could be still useful for the situations in which high accuracy is required from the decision tree would. Ant System, on the other hand, is the simplest (yet less efficient) ACO technique, so that the use of more powerful ACO algorithms (which is currently under development) would reasonably bring better results. It is well known that ACO algorithms reach their maximum efficiency when coupled with local search techniques or can improve their efficiency by making use of candidate lists.

Tree Depth	CART	Ant System
2	0.2948	0.2734
3	0.2435	0.2301
4	0.2029	0.1816
5	0.1773	0.1517
6	0.1645	0.1539

Table 4. Global impurity of the decision trees extracted by the proposed algorithm on the Credit dataset

SASSC algorithm. In the following, the SASSC algorithm is applied on the “Letter Recognition” dataset from the UCI Machine Learning Repository (source http://archive.ics.uci.edu/ml/). This dataset is originally analyzed in Frey & Slate (1991), who did not achieve a good performance since the correct classified observations did never exceed 85%. Later on, the same dataset is analyzed in Fogarty(1992) using nearest neighbours classification. Obtained results give over 95.4% accuracy compared to the best result of 82.7% reached in Frey & Slate(1991). Nevertheless, no information about the interpretability of the nearest neighbour classification model is provided and the computational inefficiency of such a procedure is deliberately admitted by the authors.

In the Letter Recognition analysis, the task is to classify 20, 000 black-and-white rectangular pixel displays into one of the 26 letters in the English alphabet. The character images are based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 numerical attributes that have to be submitted to a decision tree. Dealing with $K = 26$ response classes, SASSC provides 25 sequential aggregations. Classification trees aggregated at each single step were chosen according to 10-fold cross validation. A tree was aggregated to the sequence if it provided the lowest cross validated generalization error with respect to the other trees obtainable from different aggregations of (subgroups of) response classes.

The results of the SASSC algorithm are summarized in Figure 3. It compares the performance of the SASSC model formed by g=2 up to g=6 superclasses with that of CART using, in all cases, either Gini or Twoing as splitting rules. Bagging (Brieman, 1996) and Random Forest (Breiman, 2001) are used as benchmarking methods as well. Computations have been carried out using the R software for statistical computing.

The SASSC model using 2 superclasses consistently improves the results of CART using the Gini (Twoing) splitting rule since the generalization error reduces to 0.49 (0.34) from 0.52 (0.49). As expected, the choice of the splitting rule (Gini or Twoing) is relevant when the number of superclasses g is relatively small $(2 \leq g \leq 4)$, whereas it becomes negligible for higher values of g (results for $g \geq 5$ are almost identical). Focusing on the Gini splitting criterion, the SASSC’s generalization error further reduces to 0.11 when the number of subsets increases to 6. For comparative purposes, Bagging and Random Forest have been

trained using 6 and 10 classifiers respectively and, in these cases, obtained generalization errors are worse than those deriving from SASSC with g = 6. As for Bagging and Random Forest, increasing the number of trees used to classify each subset of randomly drawn objects improves the performance of these two methods in terms of prediction accuracy. The reason is that their predictions derive form ("in-sample") independent bootstrap replications. Instead, cross-validation predictions in SASSC derives from aggregations of classifications made on "out-of-sample" observations that are excluded from the tree growing procedure. Thus, it is natural to expect that cross-validation predictions are more inaccurate than bagged ones. Of course, increasing the number of subsets of the response classes in SASSC reduces the cross-validated generalization error but, at the same time, increases the complexity of the final classification model. In spite of a relatively lower accuracy, interpretability of the results in SASSC with *g = 6* is strictly preserved.

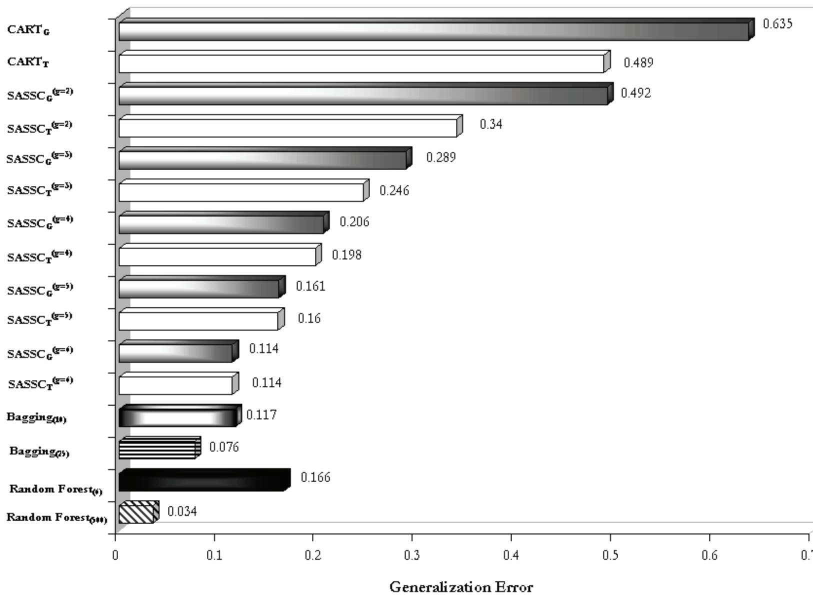

Figure 3. The generalization errors for the Letter Recognition dataset provided by alternative approaches: as for SASSC, subscript *G(T)* indicates the Gini (Twoing) splitting rule, whereas apex *g* indicates the number of superclasses (i.e., decision trees) identified by the algorithm. The subscript for Bagging and Random Forest indicates the number of trees used to obtain the classification by majority voting.

6. Discussion and conclusions

In the last two decades, computational enhancements highly contributed to the increase in popularity of DTI algorithms. This cause the successful use of Decision Tree Induction (DTI) using recursive partitioning algorithms in many diverse areas such as radar signal classification, character recognition, remote sensing, medical diagnosis, expert systems, and speech recognition, to name only a few. But recursive partitioning and DTI are two faces of

the same medal. While the computational time has been rapidly reducing, the statistician is making more use of computationally intensive methods to find out unbiased and accurate classification rules for unlabelled objects. Nevertheless, DTI cannot result in finding out simply a number (the misclassification error), but also an accurate and interpretable model. Software enhancements based on interactive user interface and customized routines should empower the effectiveness of trees with respect to interpretability, identification and robustness. The latter considerations have been the inspiration for the algorithms presented in this chapter aimed at the improvement of DTI effectiveness. They lead to easily interpretable solutions for rather complicated data analysis problems and can be fruitfully used in different fields of Knowledge Discovery from Databases (KDD) and data mining such as, for example, web mining and Customer Relationship Management (CRM).

A Genetic Algorithm for multi-attribute predictor splitting is proposed in this chapter. It can be said that the proposed GA works very well in presence of treatable splitting predictors, for which the exhaustive enumeration is affordable. The algorithm always reaches the global optimum very quickly. This makes possible to think positively, even if nothing can be said, of course, about the case in which the number of attributes gets too large for the exhaustive enumeration and evaluation. Obtained results can be considered definitely useful in those cases where there are no other ways to attack the problem. Future research directions will include exhaustive enumerations on bigger datasets on a grid computing infrastructure.

In addition an Ant Colony Optimization algorithm is also proposed for exploratory tree growing. Such algorithm could be useful for the situations in which high accuracy is required from the decision tree would. Ant System, on the other hand, is the simplest (yet less efficient) ACO technique, so that the use of more powerful ACO algorithms (which is currently under development) would reasonably bring better results. It is well known that ACO algorithms reach their maximum efficiency when coupled with local search techniques or can improve their efficiency by making use of candidate lists.

Finally, a sequential search algorithm for modelling multi-attribute response through DTI has also been introduced. The motivation underlying the formalization of the SASSC algorithm derives from the following intuition: basically, since standard classification trees unavoidably lead to prediction inaccuracy in the presence of multi-class response, it would be favourable to look for a relatively reduced number of decision trees each one relating to a subset of classes of the response variable, the so called super-classes. Reducing the number of response classes for each of those trees naturally leads to improve the overall prediction accuracy. To further enforce this guess, an appropriate criterion to derive the correct number of super-classes and the most parsimonious tree structure for each of them has to be found. In this respect, a sequential approach that automatically proceeds through subsequent aggregations of the response classes might be a natural starting point.

The analysis of the Letter Recognition dataset demonstrated that the SASSC algorithm can be applied pursuing two complementary goals: 1) a content-related goal, resulting in the specification of a classification model that provides a good interpretation of the results without disregarding accuracy; 2) a performance-related goal, dealing with the development of a model resulting effective in terms of predictive accuracy without neglecting interpretability. Taking these considerations into account, SASSC appears as a valuable alternative to evaluate whether a restricted number of independent classifiers improves the generalization error of a classification model.

7. References

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

Breiman, L. (1996) Bagging Predictors, *Machine Learning*, 24, 123-140.

Breiman,, L. (2001). Random Forests, *Machine Learning*, 45, 5-32.

Cappelli, C., Mola, F., & Siciliano, R. (2002), A Statistical Approach to Growing a Reliable Honest Tree, *Computational Statistics and Data Analysis*, 38, 285-299.

Chan, K. Y. & Loh, W. Y. (2004). LOTUS: An algorithm for building accurate and comprehensible logistic regression trees. *Journal of Computational and Graphical Statistics*, 13, 826–852.

Choi, Y., Ahn, H. & Chen, J.J. (2005). Regression trees for analysis of count data with extra Poisson variation. *Computational Statistics and Data Analysis,* 49, 893–915.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., & Trubian, M. (1996). Heuristics from nature for hard combinatorial problems. *International Transactions in Operational Research*, March, 1-21.

Conversano, C. (2002) Bagged mixture of classifiers using Model Scoring Criteria. *Patterns Analysis & Applications*, 5, 4, 351-362.

Dietterich, T.G. (2000) Ensemble methods in machine learning. In J.Kittler and F.Roli, (Eds.), *Multiple Classifier System*. First International Workshop, MCS 2000, Cagliari, vol. 1857 of Lecture notes in computer science. Springer-Verlag.

Dorigo, M. & Stutzle, T. (2004). *Ant Colony Optimization*. The MIT Press, London. 1-15

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy.

Fogarty, T. (1992) First Nearest Neighbor Classification on Frey and Slate's Letter Recognition Problem (Technical Note). *Machine Learning*, 9, 387-388 .

Fogel, L. J. (1997). A retrospective view and outlook on evolutionary algorithms. In *Fuzzy Days*, 337–342.

Fogel, D. B. & Fogel, L. (1993). Evolutionary computation. *IEEE Transactions on Neural Networks*, 5(1):1–2.

Freund, Y., & Schapire, R. (1996), Experiments with a new boosting algorithm, *Machine Learning: Proceedings of the Thirteenth International Conference*, 148-156.

Frey, P.W. & Slate, D.J. Letter Recognition Using Holland-style Adaptive Classifiers. *Machine Learning*, 6, 161-182.

Gama, J. (2004), Functional trees, *Machine Learning*, 55, 219–250.

Hastie, T., Friedman, J. H., & Tibshirani, R., (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer.

Hothorn, T., Hornik, K. & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework, *Journal of Computational and Graphical Statistics*, 15, 651–674.

Hyafil & Rivest (1976). Constructing optimal binary decision trees is NPcomplete. *IPL: Information Processing Letters*, 15-17.

Klaschka, J., Siciliano, R., & Antoch, J. (1998): Computational Enhancements in Tree-Growing Methods, in: Rizzi, A., Vichi, M. & Bock, H.H. (Eds.), *Advances in Data Science and Classification: Proceedings of the 6th Conference of the International Federation of Classification Society,* Springer-Verlag, Berlin Heidelberg. 295-302

Loh, W.Y. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12, 361-386.

Mehta, M., Agrawal, R. & Rissanen J. (1996). SLIQ. A Fast Scalable Classifier for Data Mining. In *Proceedings of the International Conference on Extending Database Technology EDBT*, 18-32.
Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, third edition.
Miele, R., Mola, F., Siciliano, R. (2005). J-Fast: An Interactive Software for Classification and Regression Trees. In *Proceedings of the Classification and Data Analysis Group (CLADAG) of the Italian Statistical Society*. Parma, Italy, 437-440
Miele, R. (2007). Nature Inspired Optimization Algorithms for Classification and Regression Trees. Ph.D. Thesis. Univeristy of Naples "Federico II".
Mola, F., & Conversano, C. (2008) Sequential Automatic Search of a Subset of Classifiers in Multiclass Learning, in: Brito P. & Aluja-Banet T. (Eds.) *COMPSTAT 2008 Proceedings in Computational Statistics,* Physica-Verlag, to appear.
Mola, F., & Siciliano, R. (1997). A fast splitting algorithm for classification trees. *Statistics and Computing*, 7, 209–216.
Oliver, J.J., & Hand, D. J. (1995). On Pruning and Averaging Decision Trees, in *Machine Learning: Proceedings of the 12th International Workshop*,430-437.
Quinlan, J.R., (1983). Learning Efficient Classification Procedures and Their Application to Chess and Games. In Michalski R.S., Carbonell J.G. & Mitchell T.M. (ed.): *Machine Learning: An Artificial Intelligence Approach*, 1, Tioga Publishing, 463-482.
Quinlan, J.R., (1987). Simplifying decision tree. *International Journal of Man-Machine Studies*, 27, 221–234.
Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
Siciliano, R., (1998). Exploratory versus decision trees. In: Payne, R., Green, P. (Eds.), *COMPSTAT 1998 Proceedings in Computational Statistics*. Physica-Verlag, 113–124.
Siciliano, R. & Mola, F. (2000). Multivariate Data Analysis through Classification and Regression Trees, *Computational Statistics and Data Analysis*, 32, 285-301, Elsevier Science, 2000.
Su, X., Wang, M. & Fan, J. (2004). Maximum likelihood regression trees. *Journal of Computational and Graphical Statistics*, 13, 586–598.
Vose, M. D. (1999). The simple genetic algorithm: foundations and theory. MIT Press, Cambridge, MA.

Appendix: The J-FAST software

The algorithms presented in this chapter have been implemented in the Java language. In order to make it possible to test them on real datasets a Java segmentation framework, called J-FAST, has been developed. The first aim of this software is to take care of all the necessary operations to perform before and after running the recursive partitioning algorithm. These can be summarized as follows: reading data from text files and spreadsheets; processing data before carrying out the tree growing process; specifying the type of recursive partitioning algorithm to be applied (i.e., classification or regression tree) ; interpretation of the results.
The J-FAST program is a Java-based recursive partitioning software, which is particularly research oriented. It mainly consists of a flexible, efficient and transparent cross-platform application for building classification and regression trees using any kind of heuristic in the tree growing process (like the CART greedy algorithm or the FAST branch and bound

heuristic or any other one written by the user). It also allows to interactively visualize and compare the results. J-FAST divides the recursive partitioning procedure into three main sections. The data-importing Graphical User Interface (see Figure 4) allows to read data from Excel-like spreadsheets and plain text files and automatically recognises the nature of the variables by distinguishing the categorical, numerical or alphanumerical columns of a data matrix. J-Fast also allows the user to specify the Decision Tree Induction model by choosing the response variable, as well as which predictor(s) should be treated as ordinal, nominal or as excluded from the analysis.

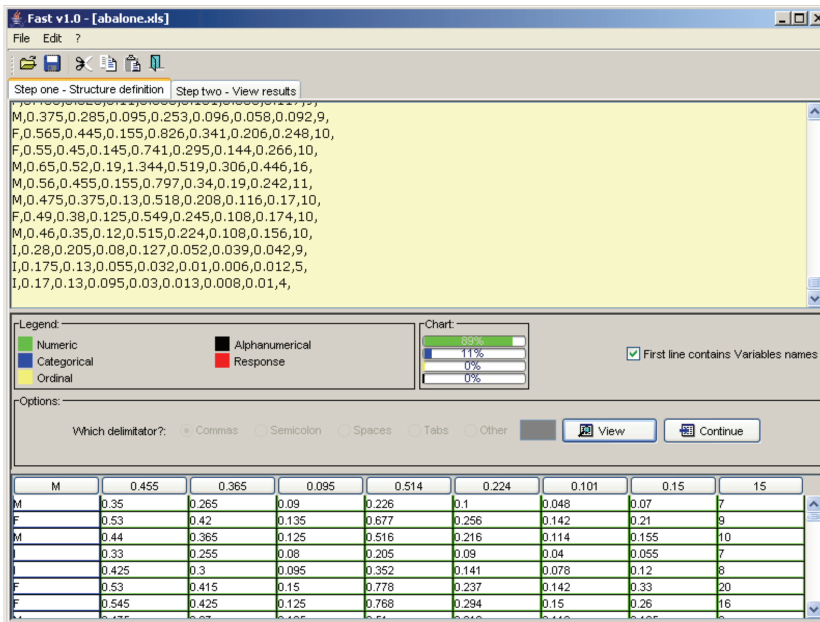

Fig. 4. J-Fast data importing Graphical User Interface

A second GUI visualizes some information about the chosen DTI model and provides some descriptive statistics about the analyzing data. It also allows the user to specify which are the features of the DTI model under specification, such as the learning sample rate, the stopping conditions, the possibility of obtaining a verbose output. It also asks the user to choose between all the recursive partitioning heuristics that are present into the class path. Then, the software starts the tree growing procedure.

The third component of the J-FAST software is the results navigator. It allows the user to interactively display and navigate into the results of the analysis.

The results navigator GUI (see Figure 5) consists of two windows. The first one is the main results window. It visualises the obtained decision tree, charts the misclassification rates and the selected node's information panel (there is a button for visualizing the splitting rule to reach the node, the misclassification rate for the node, etc.). The second component is the Tree Console Window (Figure 6). It contains buttons that allow the user to navigate through the pruning sequence and access directly the best, the trivial and the maximal tree. For each tree in the pruning sequence, the node that is going to be pruned is highlighted. By clicking

on the node, the interface allows to get the data units which fall in that node and to write them into a file in order to continue the analysis of such units using another software. It is also possible, from the second step GUI, to simultaneously start more than one analysis in order to obtain different tree navigators simultaneously on the screen. This feature is particularly useful for comparing trees grown from different datasets or on the same dataset but with using different DTI specifications.

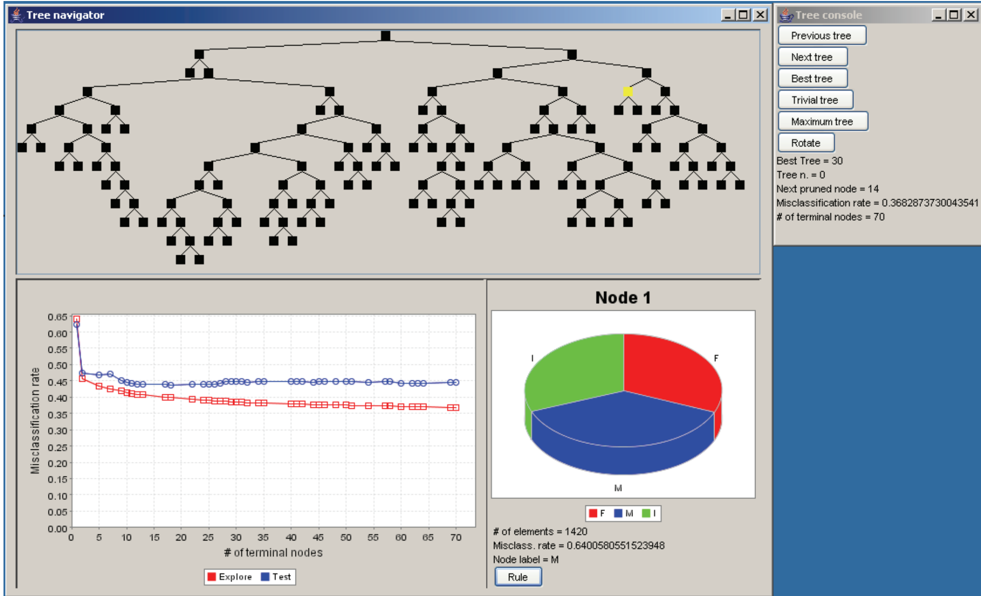

Fig. 5. J-Fast data results navigator Graphical User Interface

J-FAST is more than a simple recursive partitioning software. Because of the fact that it has been mainly designed to support the research activity, it offers many useful functions like the possibility of saving created objects (trees, datasets, nodes, etc.) via the Java serialization mechanism in order to better analyze using other ad-hoc written Java programs (some of them have already been implemented, like a different tree interface called "TreeSurfer").
Interactivity with the R statistical software is also provided: by right-clicking on a node it is possible to send the corresponding data to R in order to continue the analysis. This is particularly useful if another statistical analysis (i.e. a logit model) has to be made on a particular segment (node) extracted from the obtained decision tree.
J-FAST has to be also considered as a Java objects Library (or API - Application Program Interface), for building Classification and Regression Trees. Any researcher which is able to program in Java could use the classes from the J-FAST API in order to get trees without having to write all the necessary code. In addition, the J-FAST platform offers many useful objects. The most important ones are:

- Statistics: it provides univariate and bivariate descriptive statistics.
- DataSet: it stores data for recursive partitioning purposes (response variable, predictors, etc.).
- Split: it specifies the type of split (binary, ternary,etc.)

- TreeGrower: it is a class for growing decision trees
- Pruner: it is class that for decision tree pruning
- TreeViewer: it is a interactive interface class
- Utility: it encompasses many useful function like reading data from plain text files, Excel-like spreadsheets, etc.
- TreeBuild interface: it defines all the rules to follow for the programmer to write his own heuristic.

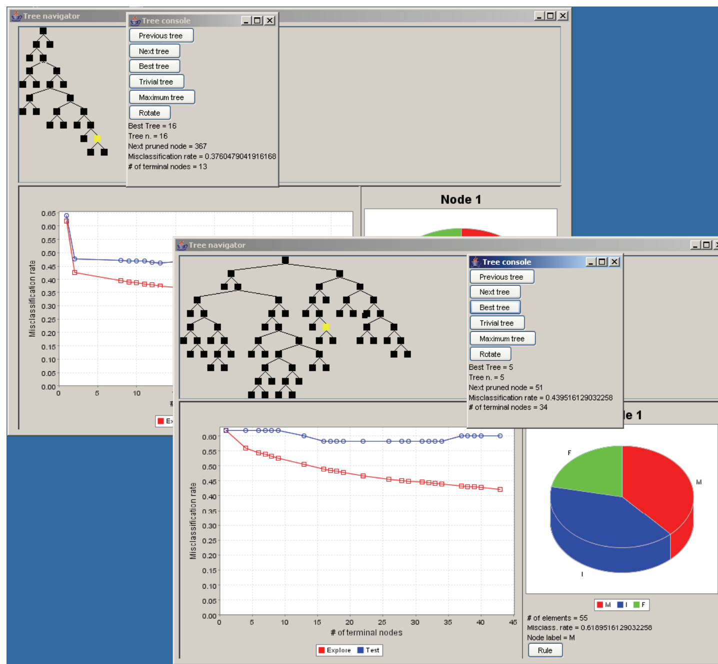

Fig. 6. J-Fast tree console window Graphical User Interface