Data mining with genetic algorithms on binary trees

Kenneth Sørensen a,*, Gerrit K. Janssens b

a University of Antwerp, Faculty of Applied Economic Sciences, Middelheimlaan 1, B-2020 Antwerp, Belgium
b Limburg University Centre, Universitaire Campus, gebouw D, B-3590 Diepenbeek, Belgium

Abstract

This paper focuses on the automatic interaction detection (AID)-technique, which belongs to the class of decision tree data mining techniques. The AID-technique explains the variance of a dependent variable through an exhaustive and repeated search of all possible relations between the (binary) predictor variables and the dependent variable. This search results in a tree in which non-terminal nodes represent the binary predictor variables, edges represent the possible values of these predictor variables and terminal nodes or leafs correspond to classes of subjects. Despite of being self-evident, the AID-technique has its weaknesses. To overcome these drawbacks a technique is developed that uses a genetic algorithm to find a set of diverse classification trees, all having a large explanatory power. From this set of trees, the data analyst is able to choose the tree that fulfils his requirements and does not suffer from the weaknesses of the AID-technique. The technique developed in this paper uses some specialised genetic operators that are devised to preserve the structure of the trees and to preserve high fitness from being destroyed. An implementation of the algorithm exists and is freely available. Some experiments were performed which show that the algorithm uses an intensification stage to find high-fitness trees. After that, a diversification stage recombines high-fitness building blocks to find a set of diverse solutions.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Genetic algorithms; Data mining; Binary trees

1. Introduction

Due to the ease of generating, collecting and storing data, we live in an expanding universe of too much data. At the same time, we are confronted with the paradox that more data means less information. Because the cost of processing power and storage has been falling, data have become very cheap. This has opened a new challenge for computer science: the discovery of new and meaningful information.

Traditional on-line transaction processing systems (OLTPs) allow storing data into databases in a quick, safe and efficient way but they fail to deliver a meaningful analysis. While it may be very easy to access the data, it becomes increasingly difficult to access the desired information. Many data are left unexplored because of the lack of sufficiently powerful tools and techniques to turn the data into information and knowledge.

The analysis of data on e.g. a business should provide further knowledge about that business by deriving rules from the data stored. If this can be
achieved, it is obvious that knowledge discovery in databases (KDD) has obvious benefits for any enterprise. Piatetsky-Shapiro et al. (1996) define KDD as “... the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data”. Sometimes KDD is mixed as a term with data mining. However, the term KDD should be employed to the whole extraction of knowledge from data, while the term data mining should be used exclusively for the discovery stage of the KDD process (see e.g. Adriaans and Zantinge, 1996, p. 5).

Hence data mining is concerned with developing algorithms and computational tools and techniques to help people extract patterns from data. Finding the patterns by identifying the underlying rules and features in the data is done in an automatic way.

2. Decision tree analysis for data mining

2.1. Data mining techniques

Two general types of data mining problems exist: (a) prediction and (b) knowledge discovery. Prediction tries to find causal relationships between some fields in the database. These relationships are established by finding predictor variables that explain the variation of other, independent variables. If a causal relationship has been established, action can be undertaken to reach a specific goal e.g. reduce the number of defects of a production line, or improve customer satisfaction. Knowledge discovery problems usually describe a stage prior to prediction, where information is insufficient for prediction (e.g. in Weiss and Indurkhya, 1998, p. 7).

Although many existing techniques from such fields as artificial intelligence, statistics and database systems have been used to this effect, data mining has become an independent new field of research. One of the first approaches used multiple regression. Regression however revealed many drawbacks when applied to data mining problems. One of the shortcomings of multiple regression is that it requires quantitative rather than qualitative data elements. The main reason however for the lack of effectiveness of regression is that it is unable to detect interactions at more than one level. Regression is too weak a technique to reveal many of the complex effects that are present in data, sometimes failing to find many important causal relationships. Decision tree analysis was developed to compensate for some problems that had arisen from using multiple regression as a tool for data mining. Decision trees show the combined dependencies between multiple predictors and the dependent variable as a number of decision branches. Hence, the user can see how the outcome changes with different values of the predictor variables.

Recently, neural networks have been used. These imitate the information-processing methods of the human brain instead of using the statistical theory as a basis for the technique. For a review on data mining techniques, see e.g. Berry and Linoff (1997).

Data mining techniques can be categorised according to the kind of knowledge to be mined. These kinds of knowledge include association rules, characteristic rules, classification rules, discriminant rules, clustering, evolution, and deviation analysis. For a detailed overview of data mining techniques from a database perspective, we refer to Chen et al. (1996). Decision tree analysis seeks for classification rules of the data under study.

2.2. Decision tree analysis

As mentioned before, decision tree analysis techniques were developed to overcome the limitations found in multiple regression.

Decision trees are a simple way of representing knowledge. They classify examples into a finite number of classes. Nodes in a decision tree are labelled with attribute names, edges are labelled with possible values for this attribute and leaves are labelled with different classes. Objects are classified by following a path down the tree, taking the edges corresponding to the values of the attributes in an object.

Work related to automatically constructing and using decision trees for data description, classification and generalisation exists in a wide variety of
disciplines. It has been traditionally developed in the fields of statistics, engineering (logic synthesis, pattern recognition) and decision theory (decision table programming). Recently renewed interest has been generated by research in artificial intelligence (machine learning) and the neuroscience (neural networks).

A decision tree contains zero or more internal or non-terminal nodes and one or more leaf nodes. All internal nodes have two or more child nodes. All non-terminal nodes contain splits, which test the value of a mathematical or logical expression of the attributes. Edges from an internal node to its children are labelled with distinct outcomes of the test at the node. Each leaf node has a class label associated with it.

The decision-tree-based classification method has been influential in machine learning studies as e.g. in ID3 and C4.5 (Quinlan, 1986, 1993). Regression trees can be built e.g. by making use of the CART technique (Breiman et al., 1991) or the automatic interaction detection (AID)-technique (Morgan and Sonquist, 1963; Sonquist et al., 1973).

2.3. Automatic interaction detector: Description, problems and issues

The technique used in this research is the AID-technique. The original idea for this technique was formulated by Morgan and Sonquist (1963). They were interested in developing a statistical analysis technique that would more accurately describe “real world” social and economic events than was possible using standard statistical regression techniques.

The AID-technique is a mechanical, or automatic, system that mimics the steps taken by an experienced data analyst to determine strong data interaction effects. Its basic principle is to explain the variance of a dependent variable through an exhaustive search of all possible relations between predictors and the dependent variable. The results of the search are represented as a binary tree. The nodes represent predictor variables for which a binary split explains most. In a first step, every possible predictor variable is tested to see which one has the strongest predictive power. The population is then split into two classes according to this predictor variable. This process is repeated for the descendental classes, until some stopping criterion is met. The strength of the predictor is measured by the value of

\[ P = \frac{n_1(x_1 - \bar{x}) + n_2(x_2 - \bar{x})}{ns^2} \]

where \( \bar{x}_1 \) and \( \bar{x}_2 \) are the averages in subgroup 1 and 2, \( n_1 \) and \( n_2 \) are the number of subjects in subgroup 1 and 2, \( \bar{x} \) is the population average, \( n \) is the total number of subjects in the population, \( s^2 \) is the population variance. Both of the subgroups formed are then candidates for a new split.

The result is a series of splits, or branches in the data, where each split produces a new data set that is in turn split. The result is a classification tree. The process of splitting ends when (1) a class contains a number of members that is too small to split, (2) the maximum number of splits set by the user of the algorithm have been reached or (3) a class is so homogenous that no split is statistically significant.

One of the main criticisms of the original AID-technique was that it tends to be overly aggressive at finding relationships and dependencies in data and that it could not discriminate meaningful from meaningless relationships. AID would often find relationships that were the result of chance. Later versions of the AID-technique improved on some of its deficiencies (Einhorn, 1972). The AID-technique was combined with statistical hypothesis testing methods. Branches of the AID-technique can be tested at a certain significance level and insignificant splits can be disregarded when showing the classification tree. CHAID and THAID are some of the techniques that were developed and employed a statistical framework to discover more meaningful results (Kass, 1980).

2.4. AID problems/issues

In spite of the fact that AID is a strong technique, it has its disadvantages (Einhorn, 1972; Kass, 1975).

(1) The AID-technique requires large sample sizes, since it possibly divides the population
of subjects into many categories. If some statistical significance is required, the number of subjects in the sample must be large.

(2) The technique does not take correlated variables into consideration. This means that some of the relationships discovered by the technique may be spurious.

(3) An asymmetric variable disturbs the performance of the AID-technique. When coping with an asymmetric dependent variable, the technique tends to repeatedly split off or separate smaller groups. Asymmetric predictors on the other hand decrease their predictive power and their probability to appear in a tree.

(4) The explanatory tree structures are not stable if multiple samples are taken from the same population. These samples from the same population may lead to different trees.

(5) Stopping rules tend to be not very clear. The application of genetic algorithms may solve some of these shortcomings by offering the decision-maker a choice of several “good” trees.

3. Binary classification trees

3.1. Introduction

The specific issue of encoding the binary trees we are confronted with can be compared to the problem discussed by Van Hove and Verschoren (1994b) and later adopted by Robeys et al. (1995). However, our situation is not identical to the one described by the mentioned authors. No restrictions are imposed on the appearance of an explanatory variable or predictor in different subtrees. The only restriction is that an explanatory variable can only appear once in a path from top to leaf. If a variable appears more than once in a path from top to leaf, an empty class will be split off. This is known in data mining as the repetition (or repeated testing) problem. Nevertheless, a predictor variable can appear multiple times in the whole decision tree. This may lead to the replication problem if subtrees are replicated in the decision tree.

Although the AID-technique can work with categorical variables having many possible values, we will restrict the discussion in this paper to binary trees. This assumption restricts the variables that can be used in the AID analysis to binary ones. In later stages, the concepts and techniques discussed here can be easily adapted to include variables with more than two different possible values.

3.2. Binary trees

If \( P \) is a set, we put \( \hat{P} = P \cup \{\ast\} \), where \( \ast \) is a terminal symbol. A classification tree over \( P \) is a quadruple \( A = (V, E, \epsilon, \delta) \) with:

1. \( (V, E) \) a finite binary tree with vertices (nodes) \( V \) and edges \( E \). The set of final nodes (or leaves) is denoted by \( T \). The number of leaves is also called the width of the tree and is denoted by \( w(A) \).
2. \( \epsilon : (V - T) \times \{+, -\} \rightarrow E \) a one-to-one map. For \( v \in V - T \), we call \( \epsilon(v, +) \) the left edge and \( \epsilon(v, -) \) the right edge.
3. \( \delta : V \rightarrow \hat{P} \) is a map satisfying \( \delta(T) \subset \{\ast\} \), and \( \delta(V - T) \subset P \). Thus \( \delta \) labels the elements of \( V \); leaves are labelled by \( \ast \), non-final nodes are labelled by an element of \( P \).

If \( A = (V, E, \epsilon, \delta) \) and if \( v \in V \) then \( A^v = (V_v, E_v, \epsilon_{v-T}, \delta(V)) \) denotes the subtree with top \( v \). It is also a classification tree.

The graphical representation of a classification tree is a picture of the tree. We represent the vertices \( v \) by their label \( \delta(v) \), sketch arrows to represent the edges (\( / \) for a left edge and \( \backslash \) for a right edge), and label the left edges by \( + \) and the right edges by \( - \).

For example:
represents the classification tree \((V, E, \varepsilon, \delta)\), with 
\[ V = \{v_1, v_2, v_3, v_4, v_5\}, \]
where \(v_2, v_4\) and \(v_5\) are final nodes, \(v_1\) is the top, labelled by \(p_1\) and \(\delta(v_3) = p_2\).
Further, we have edges 
\[ E = \{e_1, e_2, e_3, e_4\}, \]
where \(e_1 : v_1 \rightarrow v_2\) and \(e_3 : v_3 \rightarrow v_4\) are left edges, and \(e_2 : v_1 \rightarrow v_3\) and \(e_4 : v_3 \rightarrow v_5\) are right edges. It is clear that this graphical representation offers a useful description of classification trees.

3.3. Complete binary \(n\)-trees

We will call a tree of height \(n\) where the paths from the top node to every leaf node all have equal length \(n\) a complete (binary) \(n\)-tree. An example of a complete 3-tree is

![Diagram of a complete 3-tree]

For convenience, we will often leave out the \(n\)th level when depicting a tree. The previous tree then becomes

![Diagram of a simplified tree]

3.4. Relative position and level of a node in a tree

We say that two nodes in two different trees occupy the same relative position in these trees if the path from the top to this node as determined by the labels of the edges on this path, is the same.
For example, in the next figure, all nodes that have the same index in trees \(A\) and \(B\) are in the same relative position, e.g. \(p_6\) occupies the same relative position as \(q_6\).

3.5. Fitness of trees

To each tree a fitness value \(f(A) \in \mathbb{R}^+\) is assigned which reflects its explanatory power, i.e. indicates how good the classification is. In this fitness function a distinction can be made between two values: the absolute fitness \(f_a(A)\) and the relative fitness \(f_r(A)\). The absolute fitness describes the structural features of the tree, whereas the relative fitness describes the contents of it. Some authors combine absolute and relative fitness into one overall fitness measure. In our research, we do not use a measure of absolute fitness, but instead keep the structure of the trees constant. Experiments have shown that this approach yields better results.

3.5.1. Absolute fitness

The absolute fitness of \(A\) refers to structural features of the tree \((V, E)\), such as height, width, symmetry, etc. For example, as wider trees are more distinctive, it is reasonable that \(f_a(A)\) is made proportional to \(w(A)\). Van Hove and Verschoren use a multiplicative factor \(\ln(w(A))/h\) (where \(h\) is the height of the tree) to measure the absolute fitness of trees (Van Hove and Verschoren, 1994b).

Using a measure of absolute fitness has been proven experimentally to be a very inefficient way to determine the correct structure of the trees. A completely different approach is not to use a measure of absolute fitness, but instead to fix the
absolute fitness of the tree by keeping its structure fixed. One could, for example, use only complete $n$-trees and use special genetic operators that preserve this property. The advantage of this procedure is that the GA does not have to find an optimal structure for the tree, so that it can dedicate all of its power towards finding the optimal contents of the tree. Of course, the disadvantage is that the GA cannot find the optimal structure, which limits its power.

Both approaches mentioned were tested. The results obtained when keeping the tree-structure fixed were much better than the ones obtained when using a measure for the absolute fitness.

3.5.2. Relative fitness

The content and the internal features determine the relative fitness of the tree. In our case the nodes represent binary tests over a feature set. The binary tests classify the subjects into classes, according to their scores on the predictor variables. The relative fitness $f_r(A)$ should be a measure of how good the classification is. If a classification is perfect, the variance within the classes is equal to zero, whereas the variance between the different classes obtains its maximal value.

Various evaluation functions exist like the Gini coefficient or the chi-square test. References related to the evaluation function can be found in Chen et al. (1996).

The fitness function used in the implementation of the proposed algorithm considers the classes of subjects defined by the predictor values in the tree. In the classification tree representation, these classes are represented by an asterisk. Borrowing from one-way anova, the following relative fitness is assigned to each tree:

$$f_r(A) = \frac{\sum_{i=1}^{K} n_i (\bar{x}_i - \bar{x})^2}{\sum_{i=1}^{K} \sum_{j=1}^{n_i} (x_{ij} - \bar{x})^2},$$

where $n_i$ is the number of observations in class $i$, $K$ is the number of classes (equal to $w(A)$), $x_{ij}$ is the $j$th observation in class $i$, $\bar{x}_i$ is the class $i$ sample mean and $\bar{x}$ is the overall sample mean.

Calculating the fitness of a classification tree is a computationally difficult process because each subject in the population has to be classified into one of the $w(A)$ classes. After that, the sum of squares in each class has to be calculated. Taking this fact into account, we want to devise our genetic operators in such a way that the number of fitness function evaluations is reduced. One way to achieve this is to ensure that high-fitness trees are not destroyed by careless crossover or other genetic operators.

4. Genetic algorithm operators

Given a set $P$, assume that a fitness value is assigned to each classification tree over $P$, and that we want to find trees with high fitness. Starting from an arbitrary, randomly generated population of decision or classification trees, the genetic algorithm is used to obtain successive populations (also called generations), containing new elements. The tools are the so-called genetic operators. In a reproduction phase, a number of classification trees are selected. Then various operators such as crossover, switch, translocation as well as certain micro-operators are applied, each with a certain probability. These operators are applied with preference to high-fitness trees, and it is expected that some new elements (the offspring) are of even higher quality. In our operators, we try to stick to the rule that high fitness is a property of trees that should be at least partially conserved by our genetic operators. Every genetic operator defined here preserves the size and shape of the tree, as we assume that only complete $n$-trees are used. Performing our genetic operators on several complete $n$-trees, will only yield other complete $n$-trees.

Van Hove and Verschoren (1994a,b) develop genetic operators on binary trees for pattern recognition problems. Their crossover and translocation operators differ from ours in that no restrictions are imposed on the subtrees that are allowed to be exchanged. These operators yield very bad results for this specific application. We believe the reason for this is that the operators do not take into account the specific aspects of the tree and often destroy the structure of it. For example, the crossover operator used by Van Hove and Verschoren can cross two trees with high fitness to yield two low-fitness trees.
Genetic Programming (GP) (Koza, 1992) uses genetic tree-operators to create computer programs from a high-level problem statement. GP starts with a population of randomly grown trees representing computer programs. Non-terminal nodes generally contain arithmetic operators. Terminal nodes contain the program’s external inputs and random constants. The fitness of a program tree is determined by the extent to which the program it represents is able to solve the program at hand. The GP crossover operator takes two trees and randomly exchanges subtrees rooted at randomly selected nodes in both trees. The GP mutation operator selects a node in a tree, removes the subtree rooted at this node and grows a new random subtree. Other GP operators are beyond the scope of this paper. For a complete description of GP, see e.g. Koza (1992). The GP operators, when applied to classification trees, tend to destroy the fitness of good trees rather than breed high-fitness trees. For this reason, we only use operators that are specifically designed to preserve the fitness of trees with a high-explanatory power.

4.1. Reproduction

The probability \( p(A) \) that an element \( A \) is chosen out of a given population of trees \( T \) is proportional to its fitness:

\[
p(A) = \frac{f(A)}{\sum_{B \in T} f(B)}.\]

Of course, more complex probability functions can be devised, but for our purposes, this simple function is sufficient.

4.2. Crossover operator

Let \( A_1 = (V_A, E_A, \epsilon_A, \delta_A) \) and \( B_1 = (V_B, E_B, \epsilon_B, \delta_B) \) be selected from the population. Further let \( v \in V_A \) and \( v' \in V_B \) be chosen randomly, but at the same relative position in both trees. Crossover produces two new trees: \( A_2 = (V'_A, E'_A, \epsilon'_A, \delta'_A) \) and \( B_2 = (V'_B, E'_B, \epsilon'_B, \delta'_B) \), where \( (V'_A, E'_A) \) is obtained by replacing the subtree \( A'_i \) of \( A_1 \) by the subtree \( B'_i \) of \( B_1 \) and \( B'_2 \) is obtained in a similar way, replacing \( B'_1 \) by \( A'_1 \).

Example. Consider the trees \( A_1 \) and \( B_1 \):

\[
A_1 = (\begin{array}{c}
\text{+} \\
P_1 \\
\text{-} \\
\text{+} \\
P_2 \\
\text{-} \\
P_3 \\
\text{-} \\
P_4 \\
\text{-} \\
P_5
\end{array})
\]

\[
B_1 = (\begin{array}{c}
\text{+} \\
Q_1 \\
\text{-} \\
\text{+} \\
Q_2 \\
\text{-} \\
Q_3 \\
\text{-} \\
Q_4 \\
\text{-} \\
Q_5
\end{array})
\]

If crossover takes place at the nodes labelled by \( p_2 \) and \( q_2 \), then the resulting trees are \( A_2 \) and \( B_2 \):

\[
A_2 = (\begin{array}{c}
\text{+} \\
Q_1 \\
\text{-} \\
\text{+} \\
Q_2 \\
\text{-} \\
Q_3 \\
\text{-} \\
Q_4 \\
\text{-} \\
Q_5
\end{array})
\]

\[
B_2 = (\begin{array}{c}
\text{+} \\
P_1 \\
\text{-} \\
\text{+} \\
P_2 \\
\text{-} \\
P_3 \\
\text{-} \\
P_4 \\
\text{-} \\
P_5
\end{array})
\]

Crossover does not change the population if (a) both selected nodes are final nodes or (b) both selected nodes are top nodes (c) the two subtrees are the same. Therefore we assume that crossover is applied only on nodes that (1) are neither leaves nor top nodes and that (2) have different underlying trees. This crossover operator, although a bit restrictive, works very well in conserving the fitness of high-fitness trees and moreover, it preserves the size and shape of the trees if both trees are complete \( n \)-trees.

Two different ways of applying the crossover operator can be devised:

(1) apply crossover to a pair of nodes in the same relative position in both trees once with a certain probability,
(2) a more aggressive approach is to apply crossover to each pair of nodes that are in the same relative position in both trees with a certain (smaller) probability.

4.3. Switch operator

If \( A = (V, E, \varepsilon, \delta) \) and \( v \in V \) is a non-terminal node then switch at the node \( v \) yields a tree \( A' = (V, E', \varepsilon', \delta) \), where \( \varepsilon'(v, +) = \varepsilon(v, -) \), \( \varepsilon'(v, -) = \varepsilon(v, +) \). For example, if the switch operator is applied to the node with label \( p_1 \) of the following classification tree:

\[
A_1
\]

\[
p_1
\]

\[
p_2
\]

\[
p_3
\]

\[
p_4
\]

\[
p_5
\]

\[
p_6
\]

\[
p_7
\]

then the resulting tree equals

\[
A_2
\]

\[
p_1
\]

\[
p_2
\]

\[
p_3
\]

\[
p_4
\]

\[
p_5
\]

\[
p_6
\]

\[
p_7
\]

Just like the crossover operator, the switch operator does not influence the structure of the tree. Again, two ways of applying the operator to a tree chosen through reproduction can be distinguished:

(1) apply switch at each tree with a certain probability, let the switch operator choose one node and switch it,

(2) a more aggressive version is to let switch take place with a certain probability at each non-terminal node.

4.4. Translocation operator

The translocation operator is a kind of auto-crossover operator. To apply translocation, first select randomly two non-final nodes \( v_1 \) and \( v_2 \) that appear on the same level of a selected tree \( A_1 = (V, E, \varepsilon, \delta) \). Then switch the subtrees \( A_{11} \) and \( A_{12} \), yielding a new tree \( A_2 \). The requirement for the two nodes to be on the same level of the tree is needed to conserve the structure of the tree after translocation.

For example consider the classification tree \( A_1 \), graphically represented as

\[
A_1
\]

\[
p_1
\]

\[
p_2
\]

\[
p_3
\]

\[
p_4
\]

\[
p_5
\]

\[
p_6
\]

\[
p_7
\]

If translocation is applied at the nodes labelled \( p_3 \) and \( p_{13} \), that are at the same level in the tree, then the resulting tree is

\[
A_2
\]

\[
p_1
\]

\[
p_2
\]

\[
p_3
\]

\[
p_4
\]

\[
p_5
\]

\[
p_6
\]

\[
p_7
\]

\[
p_8
\]

\[
p_9
\]

\[
p_{10}
\]

\[
p_{11}
\]

\[
p_{12}
\]

\[
p_{13}
\]

\[
p_{14}
\]

\[
p_{15}
\]
Again, different versions of this operator can be devised, depending on the aggressiveness:

1. give the translocation operator a certain probability to translocate two nodes in every tree,
2. allow the translocation operator to switch each pair of nodes on the same level with a certain, smaller probability.

4.5. Micro-operators

The operators crossover, switch and translocation can be called macro-operators, because they cut and paste entire subtrees within and between trees. It is assumed that the fitness of a tree can be improved by exchanging subtrees between or within two trees or switching the labels of two edges. In many cases however, the fitness of a tree can also be improved with less drastic measures. Therefore, micro-operators are devised. Micro-operators are operators that impact directly on the contents of the nodes of the trees. Depending on the type of content, different micro-operators can be devised. In our case, every non-terminal node is labelled by a predictor variable. Micro-operators change the predictor variable used in a certain node into another one. The operators act only on non-terminal nodes of trees chosen through reproduction. The main difference between macro- and micro-operators is that the latter do not exchange entire subtrees, but only individual predictor variables. Three micro-operators have been devised.

Micro-crossover randomly switches two non-terminal nodes in two different trees, whereas micro-translocation switches two non-terminal nodes in the same tree. Unlike their macro-counterparts, micro-operators do not require the nodes to be in the same relative position or on the same level. Micro-mutation changes a random label in a tree into another one, randomly chosen from the list of predictor variables. Micro-mutation is a special operator in that is the only one able to introduce new splits into the population. The other operators, including the other micro-operators, exchange parts of trees in the population. As a result, they can never introduce a split into a tree that is nowhere in the population. The main reason why micro-mutation is introduced is to ensure that each predictor variable is offered an opportunity to enter the population of trees. If the initial population is small or the number of predictor variables is very large, there is a large probability that some predictor variables never enter into the population of trees. To ensure that all predictor variables have a sufficiently large probability to play a role in the trees retained by the GA, micro-mutation is introduced.

4.5.1. Use of micro-operators

Like macro-operators, micro-operators can be used in different ways, depending on whether the operator is allowed to operate once or several times on each chosen tree. The probability of the operator being applied in the more aggressive case should be much smaller than in the less aggressive case.

The probability of micro-operators occurring is generally chosen very small. The reason for this is that micro-operators may cause a large amount of disruption in the tree they work on, destroying valuable genetic material. A tree with high fitness can be destroyed by the application of a micro-operator close to the root, because every path from the root to a leaf in the subtree defined by \( v \) will be changed by it. A minor change can have large consequences if the node changed is part of a larger subtree structure with a very high-fitness value.

5. The algorithm

5.1. Procedure

Like any GA, the algorithm is initialised by choosing a random population and calculating the fitness for each member in this population.

Given an \( N \)-sized population \( P(t) \) of decision trees, the population \( P(t + 1) \) which also has size \( N \) is obtained as follows:

1. Select two trees \( A \) and \( A' \) out of \( P(t) \), each with a probability proportional to its fitness.
(2) Apply the genetic operators to both trees:
   (2.1) Let crossover take place at randomly chosen vertices with a probability \(p_c\).
   (2.2) Let switch take place at randomly selected edges of both resulting trees with a probability \(p_s\).
   (2.3) Let translocation take place at both trees with probability \(p_t\).
   (2.4) Let micro-mutation take place at a non-final node of both trees with a small probability \(p_{lm}\).
   (2.5) Let micro-crossover take place at randomly chosen nodes of both trees with a small probability \(p_{lc}\).
   (2.6) Finally, apply micro-translocation to both trees with a small probability \(p_{lt}\).
(3) Adjoin both resulting trees to \(P(t + 1)\).

Steps (1), (2.x) and (3) are repeated until \(P(t + 1)\) has size \(N\). If \(N\) is odd, then only one tree is selected in the final step of the generation. The crossover operators are not applied in this case.

The entire procedure is repeated until the stopping criterion is satisfied, usually until a certain number of generations have been generated.

A more aggressive version of the algorithm can be obtained by employing the stronger versions of the micro-and macro-operators. The order of the operators was chosen randomly and does not have a significant influence on the outcome of the algorithm.

5.2. “Trimming” the trees

Because of the way the algorithm works, it is possible that two identical tests appear in a path from leaf to root. This means that the second split generates an empty class. To avoid this, the tree is “trimmed”, i.e. the test appearing closest to the leaf is removed. After trimming, the tree is no longer a complete \(n\)-tree. Trimming is only done at the very end of the algorithm, and therefore trimmed trees are never entered in the population since this would violate our constraint that only complete trees are used in the algorithm.

6. Implementation and experimental results

6.1. genAID

The GA algorithm for data mining was programmed in a user-friendly program genAID. A graphical user interface is provided in which the user can enter the parameters of the algorithm, the predictor and values of the independent variables and the number of trees to retain. The user also has to enter the height of the trees in the population. This height is kept constant by the genetic operators. Trees are trimmed at the end to prevent several copies of the same predictor to appear in one path from top to leaf node.

When the user presses start, the program calculates the best trees and shows them on a new form, together with their fitness value. The trees shown in this form are all different and they all have a high-fitness value. From these trees, the data analyst can choose the one that best serves his needs. The program also calculates the average and maximum fitness for each generation. The result is shown in a graph that is updated while the algorithm runs. An example of such a graph is shown in Fig. 1.

6.2. Intensification and diversification

From the graph in Fig. 1, it is clear that the algorithm works in two stages: an intensification stage and a diversification stage. The first generations are used to allow the algorithm to produce high-fitness trees. This is the intensification part of the algorithm. After these first generations, the population consists of high-fitness individuals (trees with high predictive power). During the next generations, there is no longer a constant increase in both the average and the maximum fitness. In this stage, the algorithm uses the genetic operators to find other high-fitness trees by exchanging high-fitness building blocks between trees and by altering existing trees. The main goal of diversification in this case is not to prevent the algorithm

\[1\] The program genAID can be obtained from the author.
from getting stuck in local optima, but instead it forces the algorithm to find a diverse set of solutions.

Every generation is searched for the trees with highest fitness. These trees are kept in a list, the size of which is determined by the user. At the end of the algorithm, these trees are shown in encoded form, after having been trimmed. The lengths of the intensification and diversification stages are determined by the amount of data that is examined, especially by the number of predictor variables, the size of the population and the height of the trees. In general, the user of the algorithm can visually check whether the diversification stage has been reached.

6.3. Experiments and possible improvements

Experiments were performed for a number of data sets, drawn from a marketing survey. Many binary predictor variables were tested against a few continuous independent variables. Different values for the parameters of the GA were tested. Since the goal of the algorithm is not to maximise some quantifiable function, but instead to provide a set of “good” but diverse results, it is difficult to show numerical results.

However, the results show that the algorithm can be used to determine a number of high-fitness trees that can be used by the data analyst to determine better trees than the one found by AID. Of course, after the trees are shown to the data analyst, it is up to him to determine the trees that best meet his requirements. There is no mechanical procedure to determine which trees are better than others considering other factors than explanatory power as determined by the fitness function. We believe however that the results of the algorithm will provide the data analyst with a better understanding of the studied data.

One possibility to improve the algorithm is to allow for fitness scaling. This can prevent the population from converging too soon in the generation process. For more details on fitness scaling, see Goldberg (1989).

A possible way to increase the performance of the algorithm could be to enter the optimal tree as a seed value. This optimal tree is the tree as it is...
found by the traditional AID-technique. By definition, this tree has a very high-fitness value and is very likely to contain some of the best building blocks available. Inserting this tree into the initial population might decrease the chances of the algorithm to get stuck in local optima too soon (as the global optimum is present in the initial population) and might therefore improve the average solution quality. To ensure that this tree is not by chance eliminated early in the generation process, one could consider re-entering it into the population at regular intervals.

The technique as it is developed only works on binary predictor variables and continuous independent variables. Therefore, only data containing such variables or containing variables that can be reduced to binary values can be examined. To expand the technique to predictor variables with more levels, some of the GA operators require changes. Another solution is to map the non-binary data into binary data. Several techniques have been developed for this purpose, e.g. the IDEAL algorithm by Moreira (2000).

7. Conclusion and further work

In this paper, a technique is developed to perform operations of genetic algorithms on binary trees, representing classification trees of the data mining technique AID. This technique supplies the analyst with a choice of classification trees all having a good explanatory value. From these trees, the analyst can choose the tree that best meets his requirements and is not subject to the disadvantages of the tree resulting from the AID-technique.

The technique uses genetic operators that work directly on the binary classification trees and that are designed not to destroy high-fitness trees in the population. The technique has been implemented and some experiments have been performed on data extracted from a market survey. These results show that the technique produces a set of diverse trees that all show a high-explanatory power. From these trees, the analyst can choose the tree(s) that best serve(s) his purpose.

References