ISSN 1407-7493 COMPUTER SCIENCE DATORZINĀTNE 2008-7493

INFORMATION TECHNOLOGY AND MANAGEMENT SCIENCE
INFORMĀCIJAS TEHNOLOĢIJA UN VADĪBAS ZINĀTNE

CONSTRUCTION METHODS OF THE DECISION TREES FOR GENETIC PROGRAMMING

Peter Grabusts, Dr.sc.ing, Rezekne Higher Education Institution, 90 Atbrivoshanas alley, Rezekne LV-4600, Latvia, e-mail: peter@ru.lv.

Keywords: genetic programming, decision tree

1. Introduction

In recent years genetic programming has been successfully applied to solve different optimization search problems. It can be used as long as the solution can be encoded in a tree structure. A more efficient way of overcoming the limitations of standard decision tree induction algorithms can be the usage of genetic programming**.** In case of integration with genetic programming and decision tree, each individual of the population in genetic programming can be a decision tree. The functions to be used in the genetic programming are the attributes of the decision tree and classes form the terminal set.

Genetic programming can be considered as a decision tree breeder in which a good decision tree can be generated automatically through evolution.

The aim of the work is to investigate and analyze construction methods of the decision trees with the help of genetic programming.

2. The significance of genetic programming

Genetic programming (GP) is an automated methodology inspired by biological evolution to find computer programs that best perform a user-defined task. It is therefore a particular machine learning technique that uses an evolutionary algorithm to optimize a population of computer programs according to a fitness function determined by a program's ability to perform a given computational task [1].

Genetic programming is a methodology for generating computer programs automatically. Rather than writing programs explicitly, GP applies natural selection and genetic recombination to evolve programs that solve a given problem. GP is founded on the premise that computer programs can be represented as tree structures, as shown in Figure 1.

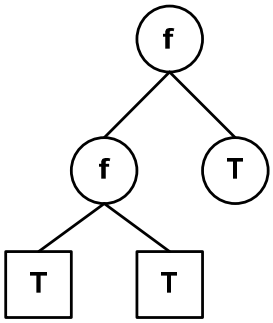


Figure 1. Tree structure example

The functions (f) operate on terminals (T) to produce a result. Functions are operations that take one or more arguments. They can be arithmetic (+, *, /), mathematical (*sin*, *cos*), Boolean (*and*, *or*, *not*), conditional (*if-then-else*), looping (*for*, *repeat*). Terminals are operations that take no arguments but return a value (variables or constant values).

The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs in the scheme computer languages as the solution. A genetic algorithm creates a string of numbers that represent the solution. GP consists of the following four steps:

1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).

2) Execute each program in the population and assign it a fitness value according to how well it solves the problem.

3) Create a new population of computer programs:

- copy the best existing programs;
- create new computer programs by mutation;
- create new computer programs by crossover.

4) The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming [1].

The GP process initially creates a number of computer programs by randomly combining functions and terminals into complete tree structures. This collection of programs, or individuals, is called a population. The GP engine evaluates and assigns a fitness value to each of the individuals in the population. The population is ranked according to fitness. A new population is generated by selecting individuals from the previous generation to participate in creative operations. Individuals are selected for inclusion in the creative process based on their fitness - the higher the fitness, the higher the likelihood of inclusion. Three basic creative operations are used in GP: reproduction, crossover, and mutation. Reproduction is simply the copying of an individual from the previous generation into the next generation without any modification of its structure.

GP substantially differs from other evolutionary algorithms in the implementation of the operators of crossover and mutation.

Mutation is performed by randomly selecting a node in an individual tree structure, and removing that node along with any sub-tree that may exist below it. A new sub-tree is then generated randomly and “grafted in” at the position where the original node was removed. An example of sub-tree mutation is shown in Figure 2 [3].

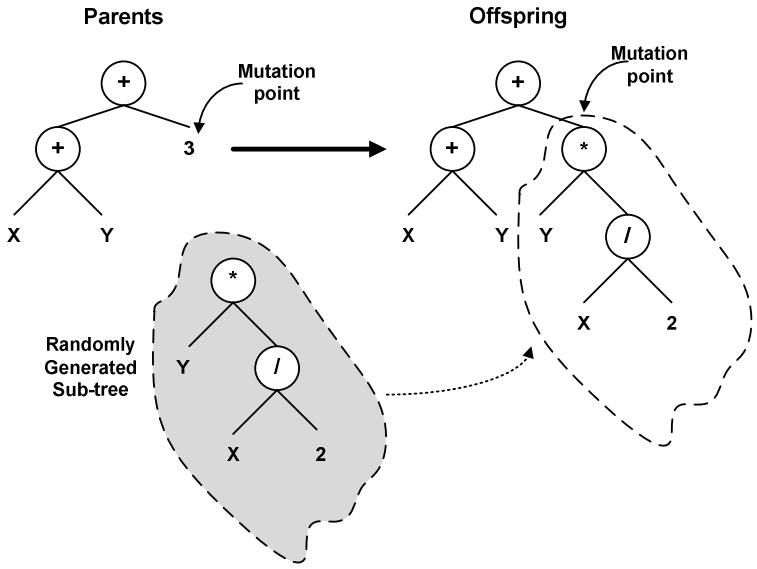


Figure 2. An example of sub-tree mutation

Crossover involves selecting two individuals from the previous generation and selecting a node at random in each of them. The selected nodes, along with any sub-trees that exist below them, are exchanged between the two individuals.

There is no guarantee that GP will find an optimal solution, but a well thought out set of functions and terminals with a reasonable fitness test will usually produce good results [2].

3. Decision trees

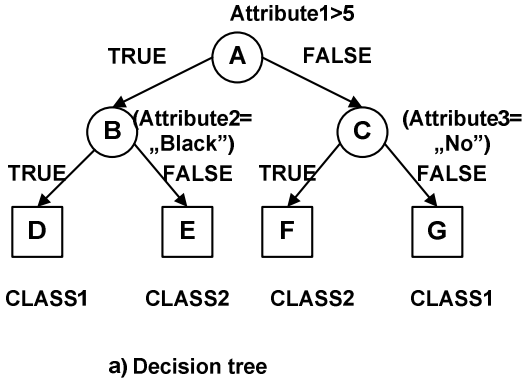
Decision trees and decision rules are data mining methodologies applied in many real-world applications as a powerful solution to classification problems. In general, classification is a process of learning a function that maps a data item into one of several predefined classes. Every classification based on inductive-learning algorithms is given as input a set of samples that consist of vectors of attribute values (also called feature vectors) and a corresponding class. The goal of learning is to create a classification model, known as a classifier, which will predict, with the values of its available input attributes, the class for some entity (a given sample). In other words, classification is the process of assigning a discrete label value (class) to an unlabeled record, and a classifier is a model that predicts one attribute-class of a sample-when the other attributes are given. Different classification methodologies are applied today in almost every discipline where the task of classification, because of the large amount of data, requires automation of the process.

A particularly efficient method for producing classifiers from data is to generate a decision tree. The decision tree representation is the most widely used logic method. There is a large number of decision tree induction algorithms described primarily in the machine-learning and applied statistics literature. They are supervised learning methods that construct decision trees from a set of input-output samples. A typical decision tree learning system adopts a top-down strategy that searches for a solution in a part of the search space. It guarantees that a simple, but not necessarily the simplest, tree will be found. A decision tree consists of nodes where attributes are tested. The outgoing branches of a node correspond to all the possible outcomes of the test at the node.

A well-known tree-growing algorithm for generating decision trees based on univariate splits is Quinlan's ID3 with an extended version called C4.5. Greedy search methods, which involve growing and pruning decision tree structures, are typically employed in these algorithms to explore the exponential space of possible models.

The most important part of the C4.5 algorithm is the process of generating an initial decision tree from the set of training samples. As a result, the algorithm generates a classifier in the form of a decision tree; a structure with two types of nodes: *a leaf*, indicating a class, or *a decision node* that specifies some test to be carried out on a single-attribute value, with one branch and a sub-tree for each possible outcome of the test.

A decision tree can be used to classify a new sample by starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the features outcome for the test at the node is determined and attention shifts to the root of the selected sub-tree. For example, if the classification model of the problem is given with the decision tree in Figure 3a), and the sample for classification in figure 3b), then the algorithm will create the path through the nodes A, C, and F (leaf node) until it makes the final classification decision: CLASS2 [4, 5]. If data samples are represented graphically in an N-dimensional space, where N is the number of attributes, then a logical classifier (decision trees or decision rules) divides the space into regions.



Attribute	Value
Attribute1	5
Attribute2	Black
Attribute3	No

b) An example for classification

Figure 3. Classification of a new sample based on the decision tree model

Each region is labelled with a corresponding class. An unseen testing sample is then classified by determining the region into which the given point falls. Decision trees are constructed by successive refinement, splitting existing regions into smaller ones that contain highly concentrated points of one class. The number of training cases needed to construct a good classifier is proportional to the number of regions. More complex classifications require more regions that are described with more rules and a tree with higher complexity. All that will require an additional number of training samples to obtain a successful classification.

A typical example of decision tree construction for well-known Fisher's IRIS data set is shown in Figure 4 [6, 7].

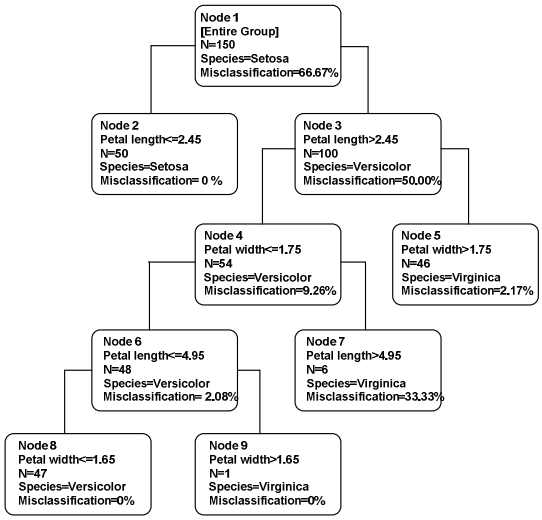


Figure 4. Decision tree for IRIS data set

One well-known limitation of selective induction algorithms is its inadequate description of hypotheses by task-supplied primitive features. To overcome this limitation, constructive inductive algorithms transform the original feature space into a more adequate space by creating new features and augmenting the primitive features with the new ones.

Originally decision trees were proposed for classification in domains with symbolic-valued features. This kind of decision trees may be called univariate or axisparallel, because the tests on each non-leaf node of the tree are equivalent to axis-parallel hyperplanes in the feature space. Another class of decision trees tests a linear combination of the features at each internal node. This kind is called multivariate linear or oblique decision tree, because these tests are equivalent to hyperplanes at an oblique orientation to the axes of the feature space.

One method, first introduced in [8], is based on the combination of primitive features and the augmentation of the feature space before tree generation. For example, as a result of combination of the primitive features x_1 and x_2 one can see a new feature as a new dimension in the feature space. Because the space of possible new features is exponential, the method uses a special kind of feature combination.

As a result, an oblique decision tree is obtained that is equivalent to a non-linear decision tree in the original x-space after a re-transformation of the features. Non-linear decision tree algorithms produce more accurate trees than their axis-parallel or oblique counterparts [8, 9].

4. Decision tree representation for GP

A decision tree representation would be able to correctly handle both numerical and categorical values. Numerical variables and values should only be compared to numerical values or variables and only be used in numerical functions. Similarly, categorical variables and values should only be compared to categorical variables or values. This is a problem for the standard GP operators (crossover, mutation and initialization) which assume that the output of any node can be used as the input of any other node. This is called the closure property of GP which ensures that only syntactically valid trees are created.

A solution to the closure property problem of GP is to use strongly typed genetic programming. Strongly typed GP uses special initialization, mutation and crossover operators. These special operators make sure that each generated tree is syntactically correct even if tree nodes of different data types are used. Because of these special operators, an extensive

function set consisting of arithmetic (+, −, ×, /), comparison (≤, >) and logical operators (and, or, if) can be used.

Another strongly typed GP representation was introduced in 1999 [5]. This linear classification GP algorithm uses a representation for oblique decision trees. An example of a tree can be seen in Figure 5 [5].

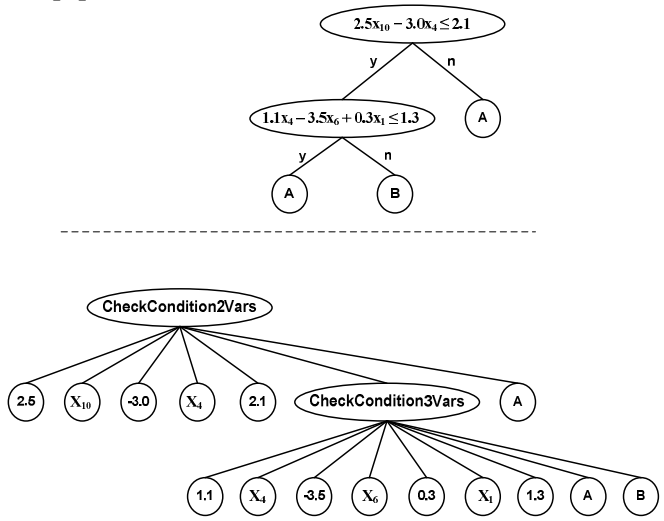


Figure 5. Example decision tree and its representation in the GP

The leftmost children of function nodes (in this case *CheckCondition2Vars* and *CheckCondition3Vars*) are weights and variables for a linear combination. The rightmost children are other function nodes or target classes (in this case A or B). Function node *CheckCondition2Vars* is evaluated as: if $2.5x_{10} - 3.0x_4 \leq 2.1$ then evaluate the *CheckCondition3Vars* node in a similar way; otherwise the final classification is *A* and the evaluation of the decision tree on this particular case is finished.

In 1998 a new representation was introduced - atomic representation that booleanizes all attribute values in the terminal set using atoms. Each atom is syntactically a predicate of the form (*variable*$_i$ *operator constant*) where *operator* is a comparison operator (e.g., ≤ and > for continuous attributes, = for nominal or Boolean attributes).

Since the leaf nodes always return a Boolean value (*true* or *false*), the function set consists of Boolean functions (e.g., *and*, *or*) and possibly a decision making function (*if − then − else*) [10]. An example of a decision tree using the atomic representation can be seen in Figure 6. Input variables are booleanized by the use of atoms in the leaf nodes. The internal nodes consist of Boolean functions and possibly a decision making function.

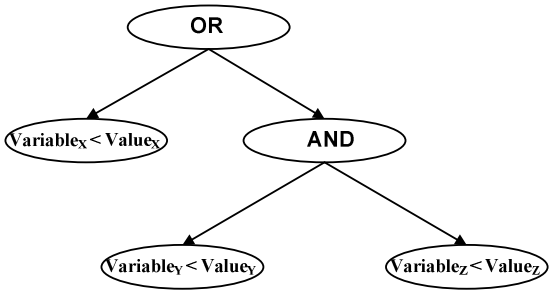


Figure 6. Example decision tree using an atomic representation

To conclude, there are a large number of different possibilities for the representation of decision trees.

5. Conclusions and future work

The main advantage of GP is that it performs a global search for a model, contrary to the local greedy search of most traditional machine learning algorithms. ID3 and C4.5, for example, evaluate the impact of each possible condition on a decision tree, while most evolutionary algorithms evaluate a model as a whole in the fitness function. As a result, evolutionary algorithms cope well with attribute interaction.

To design a decision tree using GP, each individual is defined as a decision tree, which represents both the genotype and the phenotype. The design process is still an evaluation process containing two phases:

- select a part of the training examples at random from the whole training set, and design a decision tree using C4.5, repeat this for all trees in the initial population;
- evolve the tree using GP.

The methodology investigated in the paper will be used in further scientific research that will deal with the construction of the decision tree using genetic programming.

References

1. Koza J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection // Cambridge: MIT Press, 1992.
2. Karr C., Freeman L.M. Industrial Applications of Genetic Algorithms // International Series on Computational Intelligence: CRC Press, 1999.
3. A Field Guide to Genetic Programming. URL: http://www.gp-field-guide.org.uk/ - Visit date September 2008.
4. Kantardzic M. Data Mining: Concepts, Models, Methods, and Algorithms // John Wiley & Sons, 2003.
5. Bot M.C., Langdon W.B. Application of Genetic Programming to Induction of Linear Classification Trees // Proceedings of the 3rd European Conference on GP, 2000.
6. UCI repository of machine learning databases. URL: http://archive.ics.uci.edu/ml/datasets/Iris - Visit date September 2008.
7. Software For Predictive Modeling and Forecasting. URL: http://www.dtreg.com/ - Visit date September 2008.
8. Ittner A., Schlosser M. Non-linear Decision Trees – NDT // Proceedings of 13 International Conference on Machine Learning – IML96, 1996.
9. Cantu-Paz E., Kamath C. Using Evolutionary Algorithms to Induce Oblique Decision Trees // Genetic and Evolutionary Computation Conf. (GECCO), 2000.
10. Hemert J.I. Applying Adaptive Evolutionary Algorithms to Hard Problems // Master's thesis, Leiden University, 1998.

Grabusts Pēteris. Lēmumu koku konstruēšanas metodes ģenētiskajai programmēšanai

Mūsdienās ģenētiskās programmēšanas iespējas tiek plaši izmantotas daudzos optimizācijas un klasifikācijas uzdevumos. Lai ģenētiskās programmēšanas metodes varētu tikt veiksmīgi pielietotas, ir nepieciešams konstruēt attiecīgus lēmumu kokus. Lēmumu koki un likumi uz to pamata ir intelektuālās datu analīzes sastāvdaļa un veiksmīgi tiek pielietoti klasifikācijas problēmu risināšanā. Pastāv vesela virkne dažādu paņēmienu, kas dod iespēju konstruēt lēmumu kokus ģenētiskās programmēšanas iespēju izmantošanai. Literatūras analīze liecina, ka optimālākā metode lēmumu koku konstruēšanā sastāv no diviem etapiem. Sākotnēji tiek veidots lēmumu koks, izmantojot C4.5 algoritmu, kas turpmāk tiek izvērsts ar ģenētiskās programmēšanas operatoru palīdzību. Šajā gadījumā ģenētiskā programmēšana tiek lietota kā globāla meklēšanas stratēģija precīza lēmumu koka atrašanā. Rakstā analizētas mūsdienu pieejas lēmumu koku konstruēšanā un izskatītas ģenētiskās programmēšanas iespējas, kas tiks izmantotas turpmākajā pētnieciskajā darbībā.

Grabusts Peter. Construction methods of the decision trees for genetic programming
Nowadays the possibilities of genetic programming are widely used in many optimization and classification tasks. In order to successfully apply methods of genetic programming, it is necessary to construct correspondent decision trees. Decision trees and decision rules are data mining methodologies applied in many real-world applications as a powerful solution to classification problems. There is a succession of different methods that enable the construction of decision trees for the use of the possibilities of genetic programming. Literature review gives evidence that the most optimal method in the construction of decision trees consists of two phases. Initially a decision tree is created by using C4.5 algorithm, it is further developed with the help of the genetic programming' operators. In this case genetic programming is used as a global stochastic search technique for finding accurate decision trees. The article analyzes present-day approaches to the construction of decision trees and examines the possibilities of genetic programming that will be used in further research.

Грабустс Петерис. Методы построения деревьев решений для генетического программирования
Возможности генетического программирования в наши дни широко применяются во многих задачах оптимизации и классификации. Для успешного применения методов генетического программирования требуется создание соответствующего дерева решений. Деревья решений и законы на их основе успешно применяются в решении многих проблем классификации. Существует множество способов конструирования дерева решений для использования возможностей генетического программирования. Анализ литературы свидетельствует, что оптимальный метод конструирования дерева решений состоит из двух этапов. Изначально с помощью алгоритма C4.5 строится дерево решений, которое затем расширяется с использованием операторов генетического программирования. В этом случае генетическое программирование используется как стратегия глобального поиска для нахождения точного дерева решений. В статье проанализирован современный подход к методам построения дерева решений и рассмотрены возможности генетического программирования для использования в дальнейшей исследовательской работе.