Evolutionary Algorithm for Decision Tree Induction

Dariusz Jankowski[1] and Konrad Jackowski[1,2]

[1] Wroclaw University of Technology, Department of Systems and Computer Networks, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
[2] IT4Innovations, VSB – Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava - Poruba, Czech Republic

Abstract. Decision trees are among the most popular classification algorithms due to their knowledge representation in form of decision rules which are easy for interpretation and analysis. Nonetheless, a majority of decision trees training algorithms base on greedy top-down induction strategy which has the tendency to develop too complex tree structures. Therefore, they are not able to effectively generalise knowledge gathered in learning set. In this paper we propose EVO-Tree hybrid algorithm for decision tree induction. EVO-Tree utilizes evolutionary algorithm based training procedure which processes population of possible tree structures decoded in the form of tree-like chromosomes. Training process aims at minimizing objective functions with two components: misclassification rate and tree size. We test the predictive performance of EVO-Tree using several public UCI data sets, and we compare the results with various state-of-the-art classification algorithms.

Keywords: classification, decision tree induction, evolutionary algorithms.

1 Introduction

Decision tree (DT) has been widely used to build classification models, due to its simple representation that resembles the human reasoning. There are many well-known decision-tree algorithms: Quinlan's ID3 [1], C4.5 [2] and Breiman et al.'s Classification and Regression Tree (CART) [3]. Decision trees have proved to be valuable tools for classification tasks. They have various advantages: presenting an interpretable output as a sequence of easy-to-understand tests, can handle numerical and categorical data, hierarchical decomposition allows better use of available features. One of the main difficulties of inducing a recursive partitioning structure is obtaining right sized tree (height and balance). Traditional top-down greedy strategy for decision tree can create over-complex trees that do not generalise well from the training data (overfitting problem). Several techniques have been suggested for obtaining right sized trees. The most popular of these is pruning. Since generating the optimal model tree is a NP-complete problem traditional top-down greedy strategy for decision trees may have a tendency to converge towards local optima rather than the global optimum of the problem [4].

In this work, we propose use of the evolutionary algorithms (EAs) paradigm as an alternate heuristic to generate model trees. EAs has been successfully applied to decision tree induction, e.g. Papagelis and Kalles use genetic algorithm to directly evolve decision trees [5]. Worth mentioning is using by them a tree structure to represent decision trees instead of traditional binary strings representations.

In presented EVO-Tree algorithm we decide to use global metrics of tree quality, that is size and accuracy. Such approach reduce complexity of the final classifier by removal of sections of a classifier that may be based on erroneous data without reducing predictive accuracy. Also we can focus on what criteria an induced tree must satisfy rather than how to induce a tree (which impurity measure select, how prune, etc.). We test the predictive performance of our approach using public UCI data sets, and we compare the results with state-of-arts classification algorithms.

The paper is organized as follows. In section 2 the review of decision tree algorithm is presented. Section 3 describes principles of evolutionary algorithms. Section 4 shows how evolutionary algorithm can be used to inductively generate decision trees. In section 5 experimental results show the validity of the approach. In section 6 we conclude and prioritize the directions for further work.

2 Decision Tree

The tree is a structure build from elements called nodes and branches. Three types of nodes can be distinguish: a root, internal and terminal (leaf). The root and the internal nodes denote tests on the attributes and each branch represents the outcome of a test. Each leaf node holds a class label i.e. the final decision. The general algorithm for building decision tree has two phases: growth and pruning.

In the first phase a decision tree is built by selecting the best test attribute as the root of the decision tree. Based on the test the learning set is split into two subsets (two children nodes). Then, repeat recursively the procedure on each branch to induce the remaining levels of the decision tree until all instances in a leaf belong to the same class. Different algorithms use different metrics to determine the best way to generate the test in the node and split the records. The most common impurity measures (metrics) are: Gini Index, Information Gain, and Gain Ratio [6].

The second, pruning phase may be done only on the fully grown tree. The goal is to reduce the size of decision trees by removing "insignificant" nodes or even subtrees (sections of a nodes that may be based on noisy or erroneous data). A tree that is too large risks overfitting the training data and poorly generalizing to new samples.

3 Evolutionary Algorithm

The Evolutionary algorithm (EA) is search heuristic that mimics the process of natural biological evolution. The idea behind EA is the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem. The first population of individuals is usually randomly initialized, and it evolves toward better and better regions of the search space by means of randomized processes of selection (which is deterministic in some algorithms), mutation, and crossover (also called recombination). The environment delivers quality information (fitness value) about the search points. In selection process the best individuals have a higher probability to reproduce more often than those of lower fitness. The recombination mechanism allows mixing information from two individuals and passing it to their offspring. Mutation causes small random changes in the individuals. After the evolution is completed, the fittest individual represents a "near-optimal" solution for the problem.

Most of decision tree induction algorithms relies on a greedy, top down, recursive partitioning strategy [7]. Such approach does not guarantee an globally optimal decision tree. One of the method to escape local minima in the search space is using randomized search techniques such as evolutionary algorithms. In [8] competitive co-evolution for DT induction is applied and a tree-encoding scheme is used. Works, that inheritance from the genetic algorithms conception, use fixed-length string representations (called "linear chromosomes") for coding individuals [9, 10]. The main disadvantage of such approach is hard implementation for nonbinary decision trees.

In [11, 12] authors finding with EA best combination of attributes for each split to induce oblique DTs. Such type of tree differs from traditional DTs but build an optimal oblique DT is also an NP-complete problem, what motivate authors to avoid greedy strategy.

Instead of evolve full decision trees EAs are sometimes using to improve specific components of decision tree classifier, for instance: tree pruning [13], calculating the cost of classification [14] or controlling parameters of trees [15].

4 The EVO-Tree Algorithm

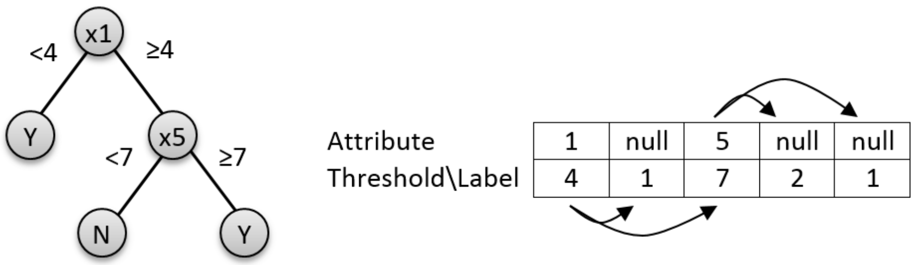
EVO-Tree (EVOlutionary Algorithm for Decision Tree Induction) is a novel multi-objective evolutionary algorithm proposed to evolve binary decision trees for classification. In multi-objective EAs different objectives are aggregated and combined into one objective function using a fixed weight when more than one objective needs to be optimized. With such a weighted aggregation, one solution is obtaining in one run (see 4.3). Algorithm 1 shows the pseudocode of the EVO-Tree algorithm.

Algorithm 1. EVO-Tree pseudocode (adapted from [16])

```
1: Randomly generate initial population of trees
2: Compute the fitness value of each tree
3: repeat
4:   Select individuals based on fitness
5:   Apply crossover and mutation to selected individuals, creating new trees
6:   Compute the fitness value of each new tree
7:   Update the current population (new individuals replace previous individuals)
8: until (stopping criteria)
```

4.1 Representation

The representation for the genotype–phenotype mapping is crucial element of a search space algorithm. We believe that tree-based representation for the candidate solution is the best one and natural. Each individual is stored in breadth-first order as an implicit data structure in two arrays (see Fig. 1). All nominal data and class labels are maps to an integer, so each component is a numeric (integer, real or null). Assuming that node has an index *i*, its left child can be found at indices *2i* and the right child at *2i+1* respectively (see Fig. 1). Terminal nodes assume null values and class number. The root has always index one. This method do not wastes space (even if the tree is not a complete binary tree) by using sparse matrices in Matlab environment. Advantages of such representation are more compact storage and better locality of reference in a context of memory utilization.



Attribute	1	null	5	null	null
Threshold\Label	4	1	7	2	1

Fig. 1. Decision tree and corresponding structure

4.2 Initial Population

Trees are generated randomly. The growth of the trees is dictated by a parameter that indicates the maximum tree depth from root to leaf inclusive, but no less than 2 levels. Generation algorithm starts with a root node and two children. Next with probability P_{split} decides whether the children are split and another child nodes are created or the node becomes a terminal node (leaf). For the leaf node random class label is assigned. If the node is further expanded then algorithm randomly select attribute and split value. If selected attribute has *k* split values, it is replaced by *k-1* calculated thresholds. First we sort all values of the attribute then with the following formula $\left(\frac{i^{th}+i^{th+1}}{2}\right)$ thresholds are set and one randomly selected. Advantage of such synthetic thresholds is protection against noisy or erroneous data. The default value of P_{split} is 0.7. Population size can be controlled by a *pop_size* parameter (details in chapter 5.1).

4.3 Fitness Evaluation

The goal of any classification system is best predictive accuracy for new unlabelled samples. For decision trees also size of the final tree is important. A tree that is too large risks overfitting the training data and poorly generalizing to new samples.

On the other hand a small tree might not capture important information about the sample space. To solve this problem in classical top-down induction algorithms various techniques for pruning decision trees were developed.

In our approach another solution is proposed. The fitness function (FF) is balanced between the number of correctly classified instances and size of the tree, with two extra parameters α_1, α_2 , to tune their relative weight:

$$FF = \alpha_1 f_1 + \alpha_2 f_2 \tag{1}$$

where

$$f_1 = 1 - \frac{Total\ no.of\ Samples\ Correctly\ Classified\ in\ Training\ Set}{Total\ no.of\ samples\ in\ Training\ Set} \tag{2}$$

$$f_2 = \frac{Tree_{current\_depth}}{Tree_{target\_depth}} \tag{3}$$

Parameters α_1, α_2 are the relative importance of the complexity term (default values are 0.99 and 0.01 respectively). There is no one optimal value of α_1, α_2 therefore tuning this parameters may lead to the improvement of the results. F_1 is the classification error estimated on the learning set. The growth of the trees is controlled by a parameter f_2 that penalizes the size of an individual (in depth of the tree) and allow to obtain the desired size of tree. The value of $\mathrm{Tree_{target\_depth}}$ should be provided by the user by tuning it to the specific problem that is solved. This parameter was set to 21.

The fitness function is the function we want to optimize. Used Matlab toolbox tries to find the minimum of the fitness function so the best fitness value for a population is the smallest fitness value for any individual in the population.

4.4 Selection

During each generation the algorithm uses the current population to create the children that make up the next generation based on the fitness function. Some well-known selection methods are: tournament selection, roulette wheel selection, rank-based selection etc. Each individual can be selected more than once, in which case it spread its genes to more children. Stochastic uniform is default selection method in our algorithm. In this technique each parent corresponds to a section of the line of length proportional to its fitness value. The algorithm moves along the line in steps of equal size. At each step, the algorithm select a parent from the section it lands on. This prevents the EA from converging too fast which allows the algorithm to better search the solution space.

The elitism technique is also implemented. Three individuals with the best fitness values in the current generation are kept to the next generation. These individuals are called elite children.

4.5 Crossover

The algorithm creates crossover child by combining pairs of parents in the current population. First, two individuals are chosen by selection algorithm. From both trees random node is selected according to randomly selected number which can take values from 1 (root node) to n (total number of tree nodes). After identifying the sub-trees according to the randomly selected number in both parents, a new individual (offspring) is created by replacing sub-tree form first parent by the one from second parent. Figure 2 illustrates the crossover operation.

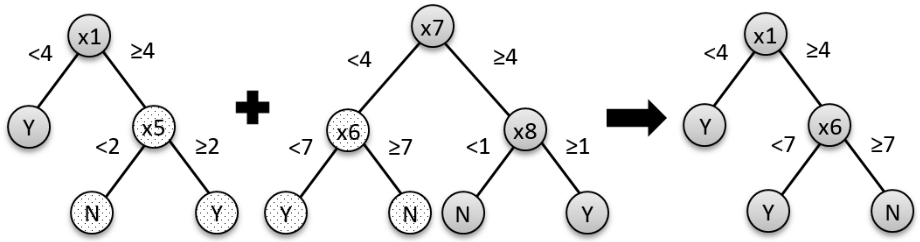


Fig. 2. Tree single point crossover

4.6 Mutation

Mutation options specify how the genetic algorithm makes small random changes in the individuals in the population to create mutation children. Mutation provides genetic diversity and enable the genetic algorithm to search a broader space. In our algorithm we implemented node condition mutation, which randomly change both attribute and split value of a randomly selected node (see Fig. 3). For terminal node class assignment is changed.

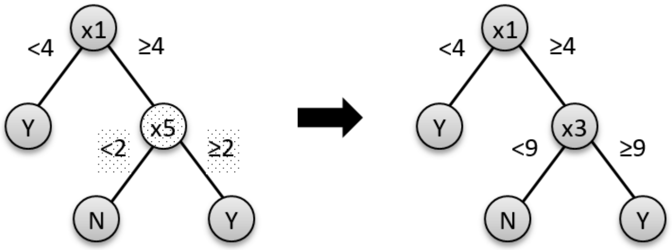


Fig. 3. Tree node mutation

4.7 Stopping Criteria

The algorithm terminates if the fitness of the best individual in the population does not improve during the fixed number of generations. This status indicates, that the algorithm has converged. Additionally, the maximum number of generations is specified, which allows limiting the computation time in case of a slow convergence. Default values one can find in chapter 5.1.

5 Experiments

In this section experimental validation of the proposed approach on real-life UCI datasets is described. Our main objectives was: effectiveness evaluation of EVO-Tree algorithm in comparative tests and identification of application domains.

5.1 Default Parameters

The parameters value are important and decide whether the algorithm will find a near-optimum solution and how efficiently. Choosing correctly the parameters is a time-consuming task. We decide to deterministic parameter control (user-specified). The default parameters of EVO-Tree used in the simulations were as follow: number of generations - 500, population size - 400, crossover/mutation probability - 0.6/0.4, selection method - stochastic uniform with elitism, stop criterion - 500 generations or 100 generations without improvement.

5.2 Datasets

There are two assumptions about data: first, there are no unknown or missing values and second, the classification of all instances is known. The datasets are briefly described in Table 1.

Table 1. Datasets specification

Dataset	**Number of Instances**	**Number of Attributes**	**Number of Classes**
abalone	210	7	28
ecoli	336	8	8
page-blocks	5473	10	5
winequality-red	1599	12	6
winequality-white	4898	12	7
breast tissue	106	10	6
seeds	210	7	3

5.3 Experimental Analysis and Results

All experiments were carried out in Matlab and KNIME framework using as a base classifiers implemented in Weka. The pool of compared classifiers consisted of 6 algorithms: EVO-Tree, Naive Bayes, Multilayer Perceptron, SVM, decision tree (C4.5), Random Tree. All experiments were carried out using 5x2 cross-validation and presented as averaged results (see Tab. 2). The advantage of this method is that all observations are used for both training and validation but never at the same time.

Additionally, we applied average ranks (AR) ranking method to resolve the issue which algorithm is the best on an unseen datasets. For each dataset we order the algorithms according to the measured accuracy and assign ranks (best algorithm has rank 1 and so on). The final ranking is obtained by averaging all values (see Tab. 3).

Table 2. Measured accuracy

Classifier	Breast tissue	Ecoli	Page blocks	Seeds	Segment	Wine q. red	Wine q. white	Abalone
EVO-Tree	0,621	0,810	0,955	0,901	0,908	0,566	0,530	0,248
J48	0,651	0,793	0,967	0,896	0,953	0,557	0,539	0,207
LibSVM	0,200	0,688	0,911	0,900	0,528	0,548	0,531	0,237
Multlayer Perceptron	0,243	0,540	0,268	0,324	0,193	0,267	0,187	0,042
Naive Bayes	0,636	0,843	0,898	0,909	0,802	0,519	0,444	0,232
Radom Tree	0,608	0,773	0,959	0,875	0,935	0,570	0,548	0,195

The abalone dataset requires a separate comment because it is very resistant to good performance under machine learning techniques. None of the supervised learning techniques that are presented here achieve good results. The reason is large number of classes and the highly overlapped data that does not split easily onto a particular class.

Table 3. Ranking generated for tested classifiers based on accuracy on all datasets

Classifier	Breast tissue	Ecoli	Page blocks	Seeds	Segment	Wine q. red	Wine q. white	Abalone	Avg. Rank
J48	1	3	1	4	1	3	2	4	2.38
EVO-Tree	3	2	3	2	3	2	4	1	2.50
Random Tree	4	4	2	5	2	1	1	5	3.00
Naive Bayes	2	1	5	1	4	5	5	3	3.25
LibSVM	6	5	4	3	5	4	3	2	4.00
Multilayer Perceptron	5	6	6	6	6	6	6	6	5.88

Our experiments seems to support the following conclusions:

- There is very little difference in the accuracy between the algorithms, however we can notice that all three decision tree-based algorithm in almost all cases outperformed other ones. We suppose that thus fact is caused by two factors:
  - First one can be found by analysis of tested benchmark datasets. According to Table 1 all of them consists of large number of attributes. Keeping in mind that decision trees algorithms has natural ability to select the most valuable features we suppose that high accuracy of them are caused by implicitly performed feature selection which eliminated irrelevant attributes.

  - The second reason is, in our opinion, high relevancy of rule based model for particular datasets. It is known, that it is hard to find out optimal model for given decision problem as there are very few guidance on relation between the classifier model and characteristic of datasets. In our cases, decision tree based models appeared to be the most appropriate.
- Results for Abalone dataset has to be commented more profoundly, because J48 and Random tree gain surprisingly low accuracy tested on this dataset. We suppose that is caused by the fact that the dataset has is the most difficult ones. It consists of 28 classes and very few samples (only 210). Such a small number of samples caused the classical decision tree memorized the learning set loosing generalisation ability. Contrary to them, EVO-Tree get very high performance what, according to our assumptions, allows to draw conclusion that proposed algorithm effectively counteract overfitting.
- Overall high accuracy of EVO-Tree on Abalone dataset shows also high ability of this algorithm to deal with dataset described by few features comparing to large number of attributes. Apparently, EVO-Tree training procedure much more effectively extract useful information and create decision rules from such difficult datasets comparing to classical decision tree algorithms. We believe, that it proves effectiveness of application of EA which optimize the tree structure in a manner that allow to avoid falling into local minima, what is natural features of classical decision tree induction algorithms.

6 Conclusion and Future Work

In this article we describe and evaluate a novel evolutionary algorithm for decision tree induction. Because it is the first publication about the EVO-Tree, mostly we focus on description of the algorithm that evolve decision trees as alternative to greedy top-down approaches. Additionally, we present experimental results that prove usefulness of our algorithm. Proposed approach reduce tree size (measured by the number of decision rules) and classification error at the same time. While traditional decision trees induced in top-down greedy strategy require pruning to remove sections of the tree that provide little power to classify instances.

The presented system is constantly improved and currently we are working on adjusting the parameters of the algorithm, which can largely influence whether the algorithm will find a near-optimum solution. Another direction of our research is generating multivariate (oblique) decision trees. The main difference between multivariate and the traditional univariate decision trees is that the first one uses linear combinations of the features at each non-leaf node for testing (which divides the attribute space with hyperplanes). Oblique decision trees are usually much smaller and often more accurate. Furthermore, EVO-Tree can provide a groundwork for an online learning algorithm appropriate for processing data streams with ability to adapt the decision model to concept drift. The idea is to have a *window* and forgetting mechanism to maintain a relevant set of examples. Those are our first goals for the future work and we believe that above extensions improve significantly EVO-Tree algorithm.

Acknowledgement. This work was supported by the statutory funds of the Department of Systems and Computer Networks, Faculty of Electronics, Wroclaw University of Technology dedicated for Young Scientists (grant no. B30036/K0402).

This paper has been elaborated in the framework of the project Opportunity for young researchers, reg. no. cz.1.07/2.3.00/30.0016, supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic.

References

1. Quinlan, J.: Learning efficient classification procedures and their application to chess end games. Mach. Learn (1983)
2. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
4. Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is NP-complete. Inf. Process. Lett. 5, 15–17 (1976)
5. Papagelis, A., Kalles, D.: GATree: Genetically Evolved Decision Trees. In: IEEE Int. Conf. Tools with Artif. Intell., pp. 203–206 (2000)
6. Bramer, M.: Principles of Data Mining. Springer (2007)
7. Rokach, L., Maimon, O.: Data mining with decision trees: theory and applications. World Scientific Publishing Company (2008)
8. Aitkenhead, M.J.: A co-evolving decision tree classification method. Expert Syst. Appl. 34, 18–25 (2008)
9. Kennedy, H.C., Chinniah, C., Bradbeer, P., Morss, L.: The contruction and evaluation of decision trees: A comparison of evolutionary and concept learning methods. In: Corne, D.W. (ed.) AISB-WS 1997. LNCS, vol. 1305, pp. 147–161. Springer, Heidelberg (1997)
10. Bandar, Z., Al-Attar, H., McLean, D.: Genetic algorithm based multiple decision tree induction. In: ICONIP 1999. ANZIIS 1999 & ANNES 1999 & ACNN 1999. 6th International Conference on Neural Information Processing. Proceedings (Cat. No.99EX378), pp. 429–434. IEEE (1999)
11. Dumitrescu, D., András, J.: Generalized Decision Trees Built With Evolutionary Techniques. Stud. Informatics Control 14, 15 (2005)
12. Czajkowski, M., Kretowski, M.: Global induction of oblique model trees: An evolutionary approach. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS (LNAI), vol. 7895, pp. 1–11. Springer, Heidelberg (2013)
13. Chen, J., Wang, X., Zhai, J.: Pruning Decision Tree Using Genetic Algorithms. In: 2009 International Conference on Artificial Intelligence and Computational Intelligence, pp. 244–248. IEEE (2009)
14. Lomax, S., Vadera, S.: A survey of cost-sensitive decision tree induction algorithms. ACM Comput. Surv. 45, 1–35 (2013)
15. Bratu, C.V., Savin, C., Potolea, R.: A Hybrid Algorithm for Medical Diagnosis. In: EUROCON 2007 - The International Conference on Computer as a Tool, pp. 668–673. IEEE (2007)
16. Basgalupp, M.P., Carvalho, A., Barros, R.C., Freitas, A.: A Survey of Evolutionary Algorithms for Decision-Tree Induction. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 1–10 (2011)