Evolutionary Learning of Linear Trees with Embedded Feature Selection

Marek Krętowski and Marek Grześ

Faculty of Computer Science
Białystok Technical University
Wiejska 45a, 15-351 Białystok, Poland
{mkret, marekg}@ii.pb.bialystok.pl

Abstract. In the paper a new evolutionary algorithm for global induction of linear trees is presented. The learning process consists of searching for both a decision tree structure and hyper-plane weights in all non-terminal nodes. Specialized genetic operators are developed and applied according to the node quality and location. Feature selection aimed at simplification of the splitting hyper-planes is embedded into the algorithm and results in elimination of noisy and redundant features. The proposed approach is verified on both artificial and real-life data and the obtained results are promising.

1 Introduction

Decision trees are, besides decision rules, one the most popular forms of knowledge representation in data mining systems [8] and clones of the classical induction algorithms are included in almost all exploratory tools. The popularity of the decision tree approach can be explained by their ease of application, fast operation and what may be the most important, their effectiveness. Furthermore, the hierarchical structure of a tree classifier, where appropriate tests from consecutive nodes are sequentially applied, closely resembles a human way of decision making which makes decision trees natural and easy to understand even for the not experienced analyst.

There are two main types of decision trees [17]: more common *univariate* trees, where tests in non-terminal nodes use single features, and *linear* (oblique) ones, where splits are based on the dividing hyper-planes. The most known example of the first group is *C4.5* [20] with its commercial version *C5.0*, whereas *LTree* and *OC1* [16] can be treated as good representatives of the second group. There exist also heterogeneous systems, like the well-known *CART* [3], where both forms of tests are permitted.

In univariate trees an inequality test is equivalent to partitioning the feature space with an axis-parallel hyper-plane. However in many real-life problems decision borders are not axis-parallel and the use of only simple tests may lead to over-complicated classifiers (so called "staircase effect"). In such a situation, a piece-wise linear solution offered by an oblique tree is much more natural and appropriate. The richer representation of linear trees gives one the opportunity

L. Rutkowski et al. (Eds.): ICAISC 2006, LNAI 4029, pp. 400–409, 2006.
© Springer-Verlag Berlin Heidelberg 2006

to find simpler and more accurate classifiers. However, it should be mentioned that induction algorithms for this type of decision tree are computationally more complex.

Nature-inspired techniques like evolutionary computation [15] are known to be especially useful in difficult optimization tasks and they are successfully applied to various data mining tasks [9]. As for decision tree learning there are two main approaches to the induction: top-down and global. The first one is based on a greedy recursive procedure of test searching and sub-node creation until the stop condition is met. In contrast to this classical method, the global algorithm searches for both the tree structure and tests at the moment. Evolutionary methods were applied for both induction types and both tree types. In the framework of univariate trees, most of the research was concentrated on global induction (e.g. [11,18,19,14]), whereas for linear trees mainly top-down methods were developed, where only splitting hyper-planes in internal nodes were evolutionary searched (e.g. [5,4,12]). In [2] genetic programming was applied to induce classification trees with limited oblique splits.

In this paper we focused on the global learning of linear decision trees with embedded feature selection. In real applications, data gathered in operational databases, which are used as a learning set, often contain irrelevant or redundant features. The overall performance of the decision tree can be improved by noisy feature elimination and test simplification. Furthermore, the ability to understand and properly interpret the classifier can be increased.

The rest of the paper is organized as follows. In the next section the proposed approach is described. Section 3 presents experimental verification with both artificial and real-life data. In the last section, the paper is concluded and future research directions are outlined.

2 Evolutionary Algorithm for Global Induction of Oblique Decision Trees

The structure of the proposed approach follows the typical evolutionary algorithm framework as described in [15]. The algorithm can be seen as a continuation and significant extension of the work presented in [13]. Due to lack of space, we gave most of our attention to new issues introduced in this work: a new scheme of applying genetic operators and embedding feature selection into the induction process.

2.1 Preliminaries

A learning set is composed of M N-dimensional feature vectors $\mathbf{x}^j = [x_1^j, ..., x_N^j]^T$ $(j = 1, ..., M)(\mathbf{x}^j \in R^N)$ belonging to one of K classes. The feature space can be divided into two regions by a hyper-plane:

$$H(\mathbf{w}, \theta) = \{\mathbf{x} : \langle \mathbf{w}, \mathbf{x} \rangle = \theta\}, \tag{1}$$

where $\mathbf{w} = [w_1, ..., w_N]$ ($\mathbf{w} \in R^N$) is a weight vector, θ is a threshold and $\langle \mathbf{w}, \mathbf{x} \rangle$ represents an inner product. A linear decision tree is a binary tree with splitting hyper-planes in internal nodes and class labels in leaves.

2.2 Genetic Operators

In the majority of evolutionary approaches there are two types of genetic operators applied to individuals: *mutation*, which affects single chromosome and *cross-over*, which enables exchange of genetic information among two chromosomes. In a typical setup, a mutation-like operator is applied with equal and relatively small probability to any gene of the chromosome. In [13] we followed this scheme and every node of the tree has the same chance of being modified. However, it seems that this approach is not really appropriate for the non-linear and hierarchical structure of the decision tree. It is evident that modification of the test in the root node is very crucial because it affects all descendant nodes, whereas mutation of a node near leaves has only a local impact. It was also observed that for small trees mutations were very rare and this significantly slowed down the induction process. Furthermore, mutations of certain nodes, like leaves with feature vectors from only one class, are not profitable at all and should be avoided.

A new complex mutation-like operator is introduced to avoid the aforementioned shortcomings. It is applied with a given probability (default 0.5) to a tree and it guarantees that at least one node will be mutated. First, the type of the node (a leaf or an internal node) is randomly chosen with equal probability. Then the ranked list of nodes is created and a mechanism analogous to ranking linear selection [15] is applied to decide which node will be affected. While concerning leaves, the number of feature vectors from other classes than the decision assigned to the leaf (i.e. number of objects misclassified in this node) is used to put them in order. Additionally, homogeneous leaves (with objects only from one class) are excluded from the list. As a result, leaves which are better in terms of the classification accuracy are mutated with lower probability. As for internal nodes, locations (the level) of the node in the tree is taken into account. It allows us to mutate with higher probability nodes, which are situated on the lower levels of the tree. In Fig. 1 an example of constructing the ranking lists of leaves and internal nodes is presented.

When one node is chosen, possibilities of applying different variants of the mutation are checked and one of them is randomly chosen according to its relevance to the node and to the given probability. Among considered possibilities are:

- changing the role of the node (i.e. pruning an internal node to a leaf or replacing a leaf by a sub-tree);
- the dipolar operator introduced in [12]. It starts with the random choosing of one dipole[1] from the set of not-divided mixed dipoles and divided pure

[1] A *mixed* dipole is a pair of feature vectors from different classes, while in *pure* dipole both objects belong to the same class.

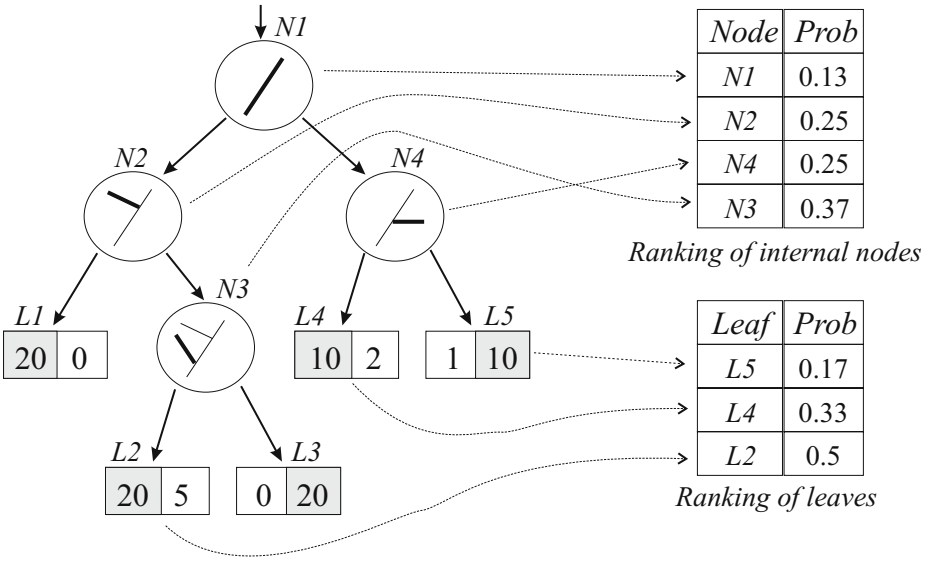


Fig. 1. The construction of two ranking lists of nodes (separate list for leaves and internal nodes) for the mutation operator

ones. If the mixed dipole is drawn, the hyper-plane is shifted to cut it. A new position is obtained by modification of only one randomly chosen weight (or threshold). In case of the pure dipole, the hyper-plane is shifted to avoid separation of objects from the same class and similarly as for mixed dipoles, only one coefficient is modified.

- the classical modification of a single weight of the hyper-plane; in relation to feature selection the probability of dropping a feature (i.e. assigning zero value to a weight) was increased because in the real value representation the number zero has extremely low probability;
- tests from the node and one of its sons are interchanged. This variant can be applied only if at least one son of the considered node is also a non-terminal node.
- one sub-tree can be replaced by another sub-tree from the same node;
- a test can be replaced by an entirely new test, which is chosen in a dipolar way [12]. One mixed dipole is randomly chosen and a hyper-plane is placed to split it. More precisely the hyper-plane is perpendicular to the segment connecting opposite ends of the dipole. The same method is applied for finding splitting hyper-planes during creation of an initial population.

In the presented system, there is also an operator analogous to the standard cross-over. One node is randomly chosen in each of two affected individuals and an exchange encompasses a sub-tree or is limited only to nodes (their hyper-planes). Additionally if both nodes are non-terminal ones, the typical one-point cross-over is applied on weight vectors and thresholds. In other cases nodes are

just substituted. It should be however noted, that according to the results [9] obtained in the framework of genetic programming, where a solution is also encoded in a tree-like structure, the context-insensitive cross-over operator has a rather destructive effect on the offspring. For that reason, the cross-over operator is applied with relatively low probability (default value is equal 0.2) in our system.

The application of a genetic operator to a tree makes its part rooted in the modified node invalid in relation to locations of the input feature vectors. For this reason renewed determination of the locations is necessary. This process can lead to a situation where certain nodes in the sub-tree are empty (or almost empty) and have to be removed. Additionally maximization of fitness is performed by pruning lower parts of the sub-tree on the condition it improves the value of the fitness.

The problem which is directly connected with feature selection is "underfitting" the training data [6], which often occurs near the leaves of the tree. The number of feature vectors used to search for a splitting hyperplane has to be significantly greater than the number of features used. In the presented system, the maximal number of features in a test is restricted using the number of available training objects (default value is 5 objects for each non-zero feature weight).

2.3 Fitness Function

A key issue for any decision tree induction algorithm is finding the appropriate balance between the re-classification accuracy and the generalization power related to the classifier complexity. In the classical top-down approach, the overfitting problem is mitigated by applying a post-pruning algorithm, but such a method has only a limited possibility to restructure the tree [7]. In contrast, the proposed evolutionary algorithm represents the global approach, where the search for an optimal tree structure is a built-in element of the process, thanks to a suitably defined fitness function. The fitness function, which is maximized, has the following form:

$$Fitness(T) = Q_{Reclass}(T) - \alpha \cdot (Comp(T) - 1.0), \tag{2}$$

where $Q_{Reclass}(T)$ is the re-classification quality, $Comp(T)$ is the complexity measure of the tree T and α is the relative importance of the complexity term (default value is 0.005) and a user supplied parameter. Subtracting 1.0 eliminates the penalty related to the complexity of the classifier when the tree is composed of only one leaf - the root node.

The complexity term $Comp(T)$, which is crucial for effective feature selection, should reflect both the tree size (the number of nodes) and the complexity of tests. This can be obtained by expressing $Comp(T)$ as a sum of test complexities in the internal nodes and the number of leaves (for any leaf we assume that

the complexity is equal 1.0). For the hyper-plane $H(\mathbf{w}, \theta)$ the test complexity $Comp(\mathbf{w})$ can be defined as follows:

$$Comp(\mathbf{w}) = (1 - \beta) + \beta \frac{n(\mathbf{w})}{N}, \tag{3}$$

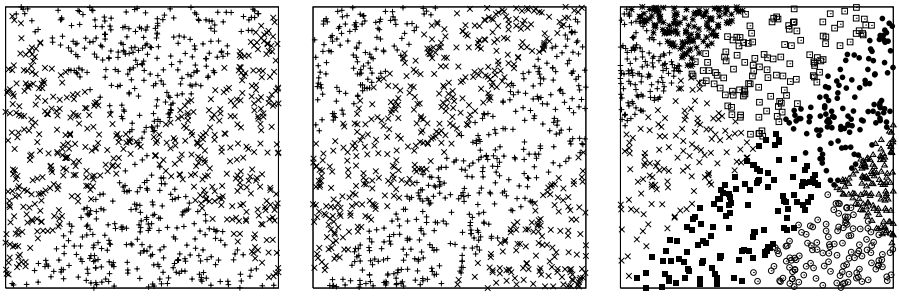
where $n(\mathbf{w})$ is the number of non-zero weights in the hyper-plane and β is a user supplied parameter designed for controlling the impact of test complexity on the tree size (default value = 0.5). When $n(\mathbf{w}) = N$ the test complexity is equal 1, which means that no feature is eliminated from the test. If this condition is met for all tests, $Comp(T)$ is equal to the number of nodes.

It should be noted that, when concerning a specific dataset, by tuning α and β parameters better results can be obtained in terms of accuracy or classifier complexity.

3 Experimental Results

Two groups of experiments are performed to validate the proposed approach (denoted in tables as *GDT*). For the purpose of comparison, results obtained by the well-known *OC1* system [16] are also presented. Both systems were run with default values of parameters. Presented in tables results correspond to averages of 10 runs and were obtained by using test sets (when available) or by 10-fold stratified cross-validation. The average number of leaves is given as a complexity measure of classifiers.

In the first group, artificial datasets with analytically defined decision borders are analyzed. Analogous experiments are described *e.g.* in [16], but original datasets are not available, hence similar configurations were generated by using a random number generator (see Fig. 2). All these datasets are two-dimensional, except *LS5* and *LS10* problems which are defined with 5 and 10 features correspondingly. The number of feature vectors in the learning sets is 1000.



a) b) c)

Fig. 2. Examples of 2-dimensional artificial datasets: a) *rotated chessboard*, b) *zebra1* and c) *zebra3*

Table 1. Results obtained for artificial datasets. Classification accuracy [%] is given as the quality measure and number of leaves as the tree size.

	without noise				50% noise				100% noise			
	GDT		OC1		GDT		OC1		GDT		OC1	
Dataset	Quality	Size	Quality	Size	Quality	Size	Quality	Size	Quality	Size	Quality	Size
chess2	98.4	4	99.3	6	97.3	4.0	89.9	16	96.5	4.0	90.7	22
zebra1	99.1	4.1	98.2	8	98.7	4.1	97.8	8	98.3	4.0	96.7	8
zebra3	97.4	9.0	95.1	15	93.5	9.7	96.3	8	93.9	8.4	90.4	8
LS2	99.8	2	99.7	2	99.9	2	99.9	2	99.8	2	98.8	6
LS5	99.5	2	98.7	2	98.7	2	99.4	4	99	2	98.4	2
LS10	98.1	2.0	96	4	97.3	2.0	93.5	4	97.3	2.0	93.5	4

In order to check the robustness of induction algorithms and especially the efficiency of the embedded feature selection, noisy features were added. They were generated randomly without taking into account class labels and obviously they are irrelevant to classification. Two levels of added noise are analyzed: the number of noisy features is equal to the half of the original amount in the features (denoted as "50% noise") and the number of additional features is equal to the number of features in the original dataset ("100% noise").

Results of experiments with artificial datasets are gathered in the Table 1. For all domains *GDT* performed very well, both in terms of classification accuracy and tree complexity. Compared to *OC1* it was able to find simpler trees with competitive accuracy. It is should be emphasized that *GDT* was also much more robust for added noise than its rival. It can be observed that the accuracy of our system is only slightly decreased, whereas *OC1* performed significantly worse in noisy scenarios.

One of the important innovations introduced in the paper is a new way of applying mutation-like operators (i.e. mutation is applied to the tree and not independently to nodes). It is rather difficult to analyze the impact of any single

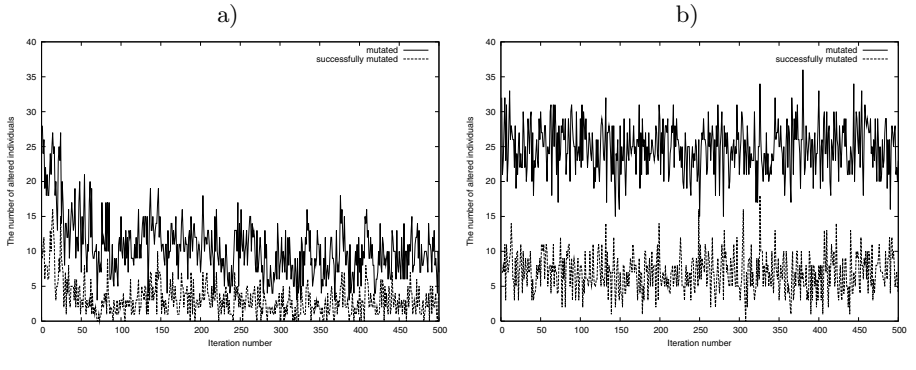


Fig. 3. The performance of two mutation strategies on *LS5* dataset: a) mutation of nodes (5%), b) mutation of the tree (50%)

Table 2. UCI datasets description. In brackets the number of feature vectors in the testing set is provided when such a set is available.

Dataset	Number of examples	Number of features	Number of classes
breast-w	683	9	2
bupa	345	6	2
iris	150	4	3
page-blocks	5473	10	5
pima	768	7	2
sat	4435(2000)	36	7
vehicle	846	18	4
waveform	600(3000)	21	3

operator modification only by looking at the final classification results and more detailed simulations are necessary. In Fig. 3 frequency and efficiency of two types of mutation are compared in one typical run of the algorithm. All other parameters except the mutation type are exactly the same. Performance of two mutation strategies is presented on *LS5* dataset because the optimal decision tree for this problem is composed of only one internal node. It can be easily observed that with the convergence of the search the mutation of nodes becomes less frequent and effective, whereas the mutation of the tree performs equally effectively and can finely fit the dividing hyper-plane position.

In the second series of experiments, a few datasets taken from UCI Machine Learning Repository [1] are analyzed to assess the performance of the proposed system in solving real-life problems. In order to avoid the problem of coding nominal features and treating missing values the datasets with only continuous-valued features and without missing values were chosen. Table 2 presents characteristics of investigated datasets. Obtained results are gathered in Table 3.

The proposed system performed well on almost all analyzed datasets, but its superiority over *OC1* is not so evident as for artificial datasets. The worst

Table 3. Results obtained for real-life datasets from UCI repository. Classification accuracy [%] is given as the quality measure and number of leaves as the tree size.

	GDT		OC1	
Dataset	Quality	Tree size	Quality	Tree size
breast-w	96.7	2.0	95.3	3.0
bupa	68.8	3.5	67.5	6.9
iris	97.0	3.0	96.7	3.0
page-blocks	95.2	3.0	97.0	12.0
pima	75.6	2.1	72.6	5.1
sat	83.7	6.3	85.4	45.0
vehicle	67.6	8.2	70.2	15.4
waveform	82.4	4.2	78.0	3.0

result was obtained by the *GDT* system with a default set of parameters for the *vehicle* dataset. It was verified that significantly better classification accuracy can be easily obtained just by relaxing the stoping condition. Finally, it was once more confirmed that the global approach finds more compact trees with at least comparable accuracy.

As can be expected, the induction times of EA-based system are longer that its top-down rival. However, even for *page-blocks*, which is the biggest dataset composed of more than 5000 feature vectors, the computation time is equal to about 58 min as measured on a standard PC (PIV 3GHz, 1GB RAM). It seems that such an amount of time is acceptable in most analytical applications.

4 Conclusion

In the paper a new evolutionary algorithm for global induction of linear decision trees is proposed. In contrast to the classical top-down approaches, both the structure of the classifier and all hyper-planes in internal nodes are searched during one run of the algorithm. The modified scheme of applying genetic operators leads to a more effective search process that is not sensitive to the tree size. The presented system is able to detect and eliminate noisy or irrelevant features from tests, thanks to feature selection embedded into the induction algorithm. Experimental validation of our approach shows that resulting trees are simpler and more compact with at least the same classification accuracy as existing counterparts.

The presented approach is constantly improved and currently we are working on introducing also univariate tests (tests with nominal outcomes and inequality tests for continuous-valued features) into our system. This should allow the algorithm to better adapt to the problem solved and to locally choose the most suitable test representation.

While investigating evolutionary algorithms there is always a strong motivation for speeding them up. Because they are well suited for parallel architecture we are contemplating re-implementing our system in a distributed environment. This is especially important in the context of modern data mining applications, where huge learning sets are analyzed.

Acknowledgments. This work was supported by the grant W/WI/5/05 from Białystok Technical University.

References

1. Blake, C., Keogh, E., Merz, C.: *UCI repository of machine learning databases*,[http://www.ics.uci.edu/˜mlearn/MLRepository.html]. Irvine, CA: University of California, Dept. of Computer Science (1998).
2. Bot, M., Langdon, W.: Application of genetic programming to induction of linear classification trees. In *EuroGP 2000*. Lecture Notes in Computer Science 1802, (2000) 247–258.

3. Breiman, L., Friedman, J., Olshen, R., Stone C.: *Classification and Regression Trees*. Wadsworth Int. Group (1984).
4. Cantu-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **7**(1) (2003) 54–68.
5. Chai, B., Huang, T., Zhuang, X., Zhao, Y., Sklansky, J.: Piecewise-linear classifiers using binary tree structure and genetic algorithm. *Pattern Recognition* **29**(11) (1996) 1905–1917.
6. Duda, O., Heart, P., Stork, D.: *Pattern Classification*. 2^{nd} edn. J. Wiley (2001).
7. Esposito, F., Malerba, D., Semeraro, G.: A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(5) (1997) 476-491.
8. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy R., (Eds.): *Advances in Knowledge Discovery and Data Mining*, AAAI Press, (1996).
9. Freitas A.: *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer (2002).
10. Gama, J., Brazdil, P.: Linear tree. *Inteligent Data Analysis* **3**(1) (1999) 1–22.
11. Koza, J.: Concept formation and decision tree induction using genetic programming paradigm, In *Proc. of PPSN 1.*, Lecture Notes in Computer Science 496 (1991) 124–128.
12. Kręt owski, M.: An evolutionary algorithm for oblique decision tree induction, In: *Proc. of ICAISC'04.*, Lecture Notes in Computer Science 3070, (2004) 432–437.
13. Kręt owski, M., Grześ, M.: Global induction of oblique decision trees: an evolutionary approach, In: *Proc. of IIPWM'05.*, Springer, Advances in Soft Computing, (2005) 309–318.
14. Kręt owski, M., Grześ, M.: Global learning of decision trees by an evolutionary algorithm, In: *Information Processing and Security Systems.*, Springer, (2005) 401–410.
15. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. 3^{rd} edn. Springer (1996).
16. Murthy, S., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research* 2 (1994) 1–33.
17. Murthy, S.: Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery* 2 (1998) 345–389.
18. Nikolaev, N., Slavov, V.: Inductive genetic programming with decision trees. *Intelligent Data Analysis* 2 (1998) 31–44.
19. Papagelis, A., Kalles, D.: Breeding decision trees using evolutionary techniques. In: *Proc. of ICML'01.*, Morgan Kaufmann (2001) 393–400.
20. Quinlan, J.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann (1993).