Evolutionary and greedy exploration of the space of decision trees

Rafał Biedrzycki[1] and Jarosław Arabas[1]

[1] Warsaw University of Technology, Institute of Electronic Systems, Warsaw, Poland,
email: {R.Biedrzycki, J.Arabas}@elka.pw.edu.pl

Abstract. This paper addresses the issue of the decision trees induction. We define the space of all possible trees and try to find good trees by searching that space. We compare performance of an evolutionary algorithm and standard, problem-specific algorithms (ID3, C4.5).

1 Introduction

Decision trees are considered to be one of the most popular approaches for representing classifiers [5, 8]. Researches from various disciplines such as statistics, machine learning, pattern recognition, and data mining are trying to grow decision trees from data. Unfortunately it has been proved that, even for simple concepts, construction of an optimal decision tree is an NP-complete problem. This leads to the development of several heuristic algorithms.

One of the most popular approaches to decision tree induction is to use algorithms based of one of the impurity functions, like ID3 [13] that uses information gain as splitting criteria. These algorithms are greedy by nature and construct the decision tree in a top-down recursive manner (divide and conquer), so they are very fast. Unfortunately the quality of solutions is very sensitive to the training set. The impurity-based algorithms perform well if few highly relevant attributes exist, but their performance decreases when very complex interactions, noise and irrelevant attributes appear. Another problem for those algorithms is diversified class probability distribution. More about advantages and disadvantages of greedy algorithms can be found in [14].

Impurity based algorithms do not guarantee yielding small decision trees (with good generalization properties) which encourages the search for other approaches. One of the possible solutions is to use genetic programming (GP) to directly evolve classifiers, e.g. [3, 4, 6, 9, 12] and many others. In the most popular approach, each individual is directly represented by the binary tree. Mutation may add a node, remove it, or change test or class in a node. Crossover chooses two random nodes and swaps sub-trees rooted in these nodes. The fitness function is based on the number of correctly classified instances and the size of the tree. This approach suffers from the problem of the rapid growth of tree size, so additional limits are assumed on the number of leaves or the tree depth. There is also no clear agreement on using genetic operators of any specific definition.

With this paper we try to define the structure of the search space for decision trees design to show the background for comparing different tree generations algorithms. We show that once the space is defined, it is possible to formulate the existing algorithms for tree induction and pruning as specific search tasks in that space. We are convinced that the presented view can be an inspiration for the application of standard optimization techniques to generate decision trees. We show how to utilize the knowledge about the structure of the space of decision trees to define the evolutionary algorithm that generates the trees. We provide a numerical example to show that the method is able to yield trees with the error smaller than for the standard tree generation algorithm ID3.

Defining AI problems as a search task is not a new idea [2, 7, 10], but as far as we know no one had tried before to build decision trees by defining the search space and performing direct search in that space.

2 Space of decision trees

2.1 Decision tree definition

Consider a Cartesian product of a certain number of sets $D = D_1 \times D_2 \times ... \times D_n$ and a function $c : D \rightarrow C$, where C is a finite set of elements. Each point from D is a tuple of n values, and each position of the tuple is called a *decision attribute*. The function c is called *classification function*, and C is the set of *classes*, i.e. classification function values. In the paper we will focus on decision problems where sets D_i are finite and ordered.

A decision tree is a tree representation of a classification function $t : D \rightarrow C$. The decision tree consists of nodes and directed edges. Each node can have at most one edge going towards it from some other node which is called a *parent* node. There exists a unique node with no parent and the node is called the *root* of the tree. If a node p is parent to the node q then q is called a *child* of the node p. Nodes with at least one child are called nonterminal or intermediate, and nodes with no children are terminal and are usually called leaves of the tree. Each terminal node p is assigned a certain value of the decision attribute $d(p)$. Each intermediate node p has an attribute $a(p)$ assigned to it. Each edge leading from the nonterminal node with the attribute $a(p)$ is assigned a test based on the value of $a(p)$. In this paper we consider tests of the form $a(p) = v_i$ where $v_i \in D_i$, so the number of children of each nonterminal node equals to the number of the node attribute values.

The decision for a particular vector of conditional attribute values $\mathbf{x} \in D$ is defined by going through the decision tree along a certain path from its root to a leaf $p(\mathbf{x})$ in such a way that along this path all the tests are satisfied, and $t(\mathbf{x}) = d(p(\mathbf{x}))$. For practical reasons it is usually assumed that along each path from the root to a leaf no attribute can be assigned to different nodes. It is however admissible (and highly desirable) that there exist paths for which only a subset of attributes is tested.

2.2 Metric space of decision trees

Consider the set of all decision trees and a metric space over this set. As a metric choose a function that equals a minimum number of elementary changes that have to be performed to get one tree from another one. Assume that an elementary change consists in either deleting a nonterminal node whose children are terminal nodes, or in replacing a terminal node with a nonterminal node. Sketch of the neighborhood structure of the search space is depicted in Figure 1 when three binary attributes are considered. Note

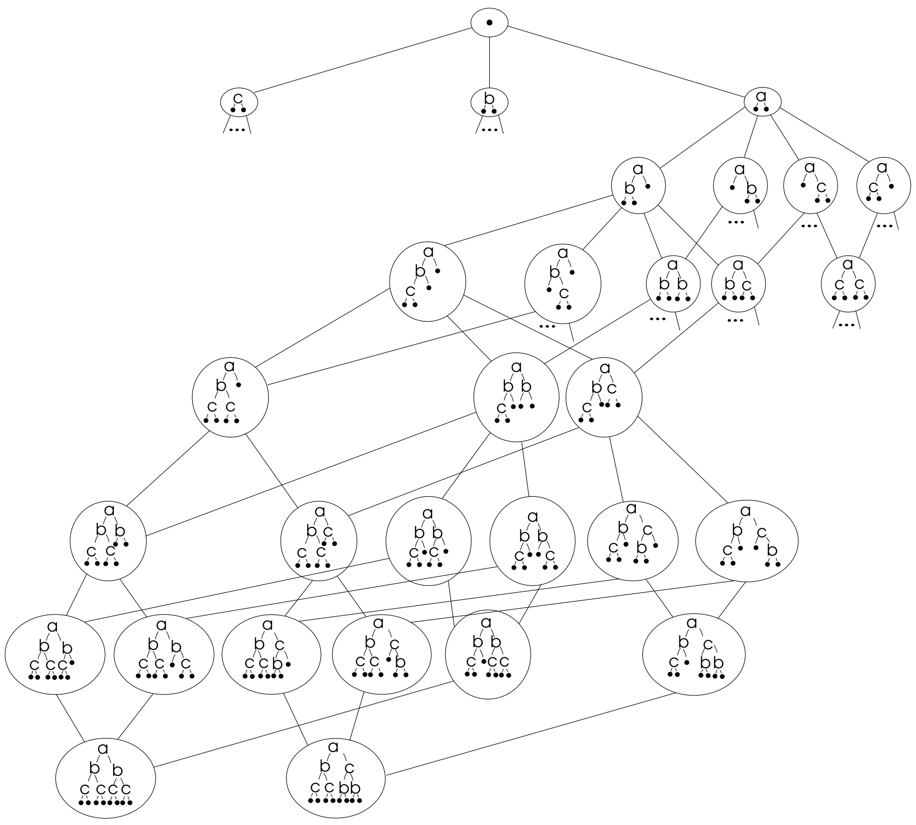

Figure 1. Part of space for three attributes. Leaves are marked by bullets

that it is possible to get any tree from any other one with the finite set of elementary changes, so the search space is consistent and the metric values have an finite upper bound. Note also that for any pair of trees there may exist more than one path between them.

2.3 Quality function

Each decision tree is a mapping $t : D \rightarrow C$. If the goal is to approximate an unknown mapping $c : D \rightarrow C$ then each tree in the search space can be assigned a quality function being the accuracy measure of that approximation. In this paper we assume the quality function being the error — the fraction of misclassified elements from D

$$e(t) = m(\{\mathbf{x} \in D, t(\mathbf{x}) \neq c(\mathbf{x})\})/m(D) \tag{1}$$

where $m(A)$ is the number of elements in the set A. The error value can be estimated using a set $A \subseteq D$ and the estimate is

$$e_A(t) = m(\{\mathbf{x} \in A, t(\mathbf{x}) \neq c(\mathbf{x})\})/m(A) \tag{2}$$

The quality function is the objective function to be minimized in the search space.

Let us discuss the properties of the objective function in the decision trees space. First, for each tree t and its neighborhood $N(t)$ if there exists a tree $t' \in N(t)$ such that $e(t) > e(t')$, then t' contains more nodes than t. Second, for each tree there exists a path to the empty tree along which the error function is monotonic. From the above considerations it is clear that the error function takes a unique maximum in the empty tree and will usually take many local minima.

Unfortunately, it may happen that the best tree $t' \in N(t)$ in the neighborhood of the tree t does not belong to the path from t to the best tree that can be constructed by adding more than a single node to t. This means that it is impossible to find the path to the global minimum by analyzing the local information.

3 Example search in space of decision trees

3.1 Search methods

Having defined the search space of decision trees we are now able to interpret the known tree generation methods as search methods in that space, and we are also able to apply general search methods that are not commonly used for tree generation. To illustrate this consider three methods: an evolutionary algorithm (as a representative of general search methods) and an ID3 and C4.5 algorithms (as reference methods dedicated to tree generation).

Evolutionary algorithm In the evolutionary algorithm each chromosome is a decision tree, and the mutation of a chromosome results in choosing at random a chromosome from its neighborhood, i.e. the mutated tree is either expanded by a nonterminal node, or a nonterminal node shrinks from it. All neighbors are equally probable to be chosen. No crossover is used and the population is initialized with empty trees. The fitness function to be minimized is defined as

$$f(t) = e_T(t) + \alpha n(t) \tag{3}$$

where $e_T(t)$ is the error estimate for a training set T, $n(t)$ is the number of nodes of the tree t, and α is a user defined parameter. The fitness function formula is typical for EA-based tree generation approach (see e.g. [3]).

ID3 method In the ID3 approach, a tree is built from the scratch. The algorithm maintains a single *current* tree t. The ID3 method uses the entropy [13] rather than the function (1) to measure the tree quality. In each iteration a tree t' is searched in the neighborhood of the current tree such that the entropy of t' is minimum in $N(t)$. The algorithm stops if there is no training example for which $t(\mathbf{x}) \neq c(\mathbf{x})$ or if no terminal node can be expanded to reduce the tree error. It is easy to recognize that ID3 is essentially a hard selection based local minimizer, thus it proceeds fast but does not necessarily lead to the best tree.

C4.5 In the C4.5 approach the tree is built in two steps. In the first step the improved ID3 method is used to build a tree. The second step consists in pruning, i.e. reducing the number of nodes in a way that provides only a small increase in the error. Thus the pruning process is essentially a local search in the direction towards the empty tree. The selection of nodes to be deleted and the stop criterion for the pruning process are based on the generalization error estimates — see [14] for details.

3.2 Test problems

The algorithms were tested with the use of several datasets from the UCI machine learning dataset repository [11] and Orange homepage [1]. A brief characteristic of these datasets is provided in Table 1.

Table 1. Brief characteristics of datasets used for experiments. In subsequent columns we give the number of records (# rec.), the number of conditional attributes (# attr.), the average number of conditional attribute values (#val.), the number of classes (#cl.), the percentage of records with equal class (dist.) and the information whether the dataset provides a distinction between the train and the test sets.

dataset	#rec.	#attr.	#val.	#cl.	distr. of classes	test set
Balance scale	625	4	5.0	3	8:46:46	N
Breast cancer	286	9	4.8	2	30:70	N
Coil2000	9,822	84	7.6	2	6:94	Y
Marketing	8,993	13	6.5	9	7:8:9:9:10:11:12:15:19	N
Monks-1	556	6	2.8	2	50:50	Y
Mushrooms	8,124	22	5.3	2	48:52	N

Note that datasets such as 'balance scale' or 'coil2000' are difficult because the number of examples belonging to each class is unevenly diversified — some classes are overrepresented. Datasets like 'monks-1' and 'breast-cancer' are observed to be highly linearly inseparable thus difficult for ID3 [14].

3.3 Experiments and results

In our experiments we used an EA with the noelitist tournament selection (tournament size was set to two). The population size was 20. The method was terminated after 200 generations. We assumed $\alpha = 0.01$ in the fitness function formula (3).

The ID3 algorithm was implemented by ourselves. We also used the implementation of the C4.5 algorithm provided in Weka [15] as J48 method.

Performance of the compared algorithms was analyzed experimentally. We applied different methodology, depending on the properties of datasets. If a dataset provided a separate test set, we used the training set to generate the decision tree and we computed the error for the test set as an indicator of the algorithm's performance.

If a dataset did not provide a separate test set, we applied a slightly more complicated methodology. Each dataset was divided into two parts: 90% of the dataset was treated as a training set, and 10% — as a validation set. We used 10 such divisions with distinct validation sets (10-fold crossvalidation). For each division we built a decision tree, and recorded its error for the validation set. That validation set error averaged over 10 divisions indicated the algorithm's performance.

In Table 2 we report the results of EA, ID3 and C4.5 obtained according to the aforementioned methodology. In case of EA we performed 10 independent runs and reported minimum, average and maximum error. For ID3 and C4.5, a single run is enough to evaluate their quality, since these methods are deterministic in nature.

Table 2. Results of the compared algorithms — the percentage of incorrectly classified examples for the test set or the validation set.

Dataset	J48	ID3	EA		
			average	best	worst
Balance scale	34.7	29.5	**29.3**	27.7	30.9
Breast cancer	**28.0**	33.2	30.0	27.1	33.2
Coil2000	**6.0**	10.1	**6.0**	6.0	6.2
Marketing	**67.6**	70.9	68.8	68.2	69.6
Monks-1	24.3	17.4	**9.5**	5.3	17.4
Mushrooms	**0.0**	**0.0**	0.3	0.0	0.8

Table 3. Results of the compared algorithms — number of nodes.

Dataset	J48	ID3	EA		
			average	best	worst
Balance scale	**41**	485	550	536	561
Breast cancer	**7**	381	437	390	470
Coil2000	**1**	31,013	618	405	789
Marketing	3,582	29,306	**1,176**	1,152	1,196
Monks-1	**18**	94	94	75	126
Mushrooms	**29**	38	333	304	387

According to Table 2, EA in the presented version performed generally better than ID3 when taking into account the error values. Paradoxically, the superiority of EA was especially visible for datasets that were quite small in the number of examples and in the number of attributes. For larger datasets the difference in results was smaller, although EA is still a winner. Considering J48 and EA, the results were generally very alike, but in datasets 'balance scale' and 'monks-1' J48 performed significantly worse because of the harmful influence of pruning.

When analyzing size of generated trees (table 3) we can see that both EA and ID3 yielded trees of a comparable size. For large datasets, however, EA generated much smaller trees. 'Coil-2000' is the dataset where ID3 generated a tree with more than 31 thousands of nodes and error rate more than 10%. As we can see in Table 1, percentage of records in different class for 'coil-2000' dataset is 94:6, so a tree with only one terminal node has the error rate of 6% and J48 was able to find this tree. Although J48 used pruning, EA achieved smaller trees for 'marketing' dataset. We believe that if pruning were applied for EA-generated trees, it could be possible to obtain trees comparable in size to those generated by J48 for all tested datasets.

Finally we would like to mention that execution of the tested algorithms consumed a rather short time, although ID3 was the fastest. This could be easily predicted, since a crucial factor influencing the computation time is the tree error evaluation, so the number of examined trees determines the computation time. For example, solving the classification task for the largest 'coil-2000' dataset took on average 3.51s (ID3) and 29.2s (EA) for the 256MB RAM, Pentium 4 M, 2GHz computer.

We were unfortunately unable to compare our EA-based solution to GP-based approaches mentioned in the introduction. The main reason was that the classification problems with discrete conditional attributes were not extensively studied in those approaches. We were only able to find results of GATree [12] for the 'balance scale' dataset (28.9%). The authors were able to generate trees of comparable classification error but GATree generated much smaller trees (9 nodes). This was achieved by limiting search space using size factor. We believe that a stronger penalty for the number of nodes (larger α in (3)) could lead our EA-based method to get comparable results.

4 Conclusions

We present a view of the tree generation as an optimization task in the space of trees. There are many possibilities of defining the space metric which, when applying a search method, may results in different sequence of candidate trees generated by this method. In the paper we concentrate on a simple metric based on number of operations of adding or deleting a node. The view concentrated on the search space opens the possibility to uniformly treat the distinct tasks of both tree induction and pruning, and to analyze different specific algorithms as instances of general search task. Thus we can think of ID3 as of the greedy search algorithm that starts from the empty tree and goes to the "closest neighbor" tree with the highest improvement to the current tree. We can interpret tree pruning as the search algorithm that starts from a certain nonempty tree and goes towards the empty tree.

We present a simple evolutionary algorithm to perform search in the space of trees. We observe that even though the algorithm was not carefully tuned, it was able to yield very encouraging preliminary results. We plan to perform more extensive study of the presented EA, and to consider different metric definitions and objective functions, including the well known entropy and the minimum codelength rule.

Bibliography

[1] Orange homepage. http://www.ailab.si/orange/datasets/.

[2] Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In *ECP*, pages 360–372, 1999.

[3] Martijn Bot. Application of genetic programming to induction of linear classification trees. Technical report, Vrije Universiteit, Faculteit der Wiskunde en Informatica, November 1999.

[4] Erick Cantú-Paz and Chandrika Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1), February 2003.

[5] Paweł Cichosz. *Systemy uczące się*. WNT, 2000.

[6] Jeroen Eggermont, Joost N. Kok, and Walter A. Kosters. Genetic programming for data classification: Partitioning the search space. In *SAC'04*, Nicosia, Cyprus, 14 -17 March 2004.

[7] Richard E. Korf. Artificial intelligence search algorithms. In *Algorithms and Theory of Computation Handbook*. CRC Press, 1999.

[8] Jacek Koronacki and Jan Ćwik. *Statystyczne systemy uczące się*. WNT, 2005.

[9] John R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496, pages 124–128, Dortmund, Germany, 1-3 1991. Springer-Verlag, Berlin, Germany.

[10] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[11] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/˜mlearn/MLRepository.html.

[12] Athanasios Papagelis and Dimitris Kalles. GATree: Genetically evolved decision trees. In *ICTAI'00*, page 203. IEEE, November 2000.

[13] John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[14] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers – a survey. *IEEE Transactions on systems, man and cybernetics – part C: applications and reviews*, 35(4), November 2005.

[15] Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann, 2000.