A search based view of decision trees induction

Rafał Biedrzycki[1], Jarosław Arabas[1]

[1] Institute of Electronic Systems, Warsaw University of Technology,
ul. Nowowiejska 15/19, 00-665 Warszawa

Abstract. This paper addresses the issue of the decision tree induction. We treat this task as a search problem in the space of decision trees. We show that the search space definition allows for easy application of general search methods to solve this task. On the other hand it is possible to interpret standard problem-specific algorithms (e.g. ID3 and C4.5) as instances of specific search methods. We assume a certain metric in the space of decision trees, and define an evolutionary algorithm and a Monte Carlo method to search that space. We provide the experimental comparison of the aforementioned algorithms with ID3 and C4.5.

Keywords. Decision tree, search space, ID3, evolutionary algorithm

1 Introduction

Decision trees are considered to be one of the most popular approaches [6, 9] for representing classifiers. 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 proven 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 approach to decision tree induction is to use algorithms based of one of the impurity functions, like ID3 [14] that uses information gain as splitting criteria. These algorithms are greedy by nature and construct the decision tree in a top-down recursive manner. Unfortunately the quality of generated solutions is very sensitive to the training set. The impurity-based algorithms perform well if a few highly relevant attributes exist, but less so if very complex interactions, noise and irrelevant attributes exist. Another problem is diversified class probability distribution. More about advantages and disadvantages of greedy algorithms we can find in [15].

Impurity based algorithms not guarantee yielding optimal decision trees, which encourages the search for other approaches. One of the possibilities is to use genetic programming (GP) to directly evolve classifier. These approach is

presented in [3, 4, 5, 7, 10, 13] and many others. In most of these approaches each individual is directly represented by the binary tree, mutation may add, remove or change test or class in a node. Crossover operator chooses two random nodes and swaps sub-trees rooted in those nodes. The fitness function is based on the number of correctly classified instances and size of the tree. Unfortunately such GP based approach suffers from explosion of search space size so additional limits to the number of leaves are assumed.

With this paper we try to define the structure of the search space for decision trees design. We show that when the space is defined, it is possible to interpret existing algorithms of tree induction and pruning (e.g. ID3 and C4.5) as the specific search tasks. We are convinced that the presented view can inspire to apply standard search techniques to grow decision trees. We show how to use general search methods (evolutionary algorithm, Monte Carlo search), to generate the decision trees. We provide numerical examples for several well known datasets to prove that it is possible to get the decision trees of comparable quality when applying evolutionary algorithm (EA), or Monte Carlo search (MC) instead of C4.5, latest member of the ID3 family.

It was observed that AI problems could be successfully defined as search tasks [e.g. 3, 8, 11], but according to our best knowledge, this was not directly applied to the decision tree induction.

2 Decision tree

Consider a Cartesian product of a certain number of sets $D=D_1\times D_2\times\ldots\times D_n$ 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 $v(p)$. 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.

3 Space of decision trees

Consider the set of all decision trees and a metric space T 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 test node whose children are category nodes, or in replacing a category node with a test node. Sketch of the fragment of the search space for three binary attributes is depicted in Figure 3.1.

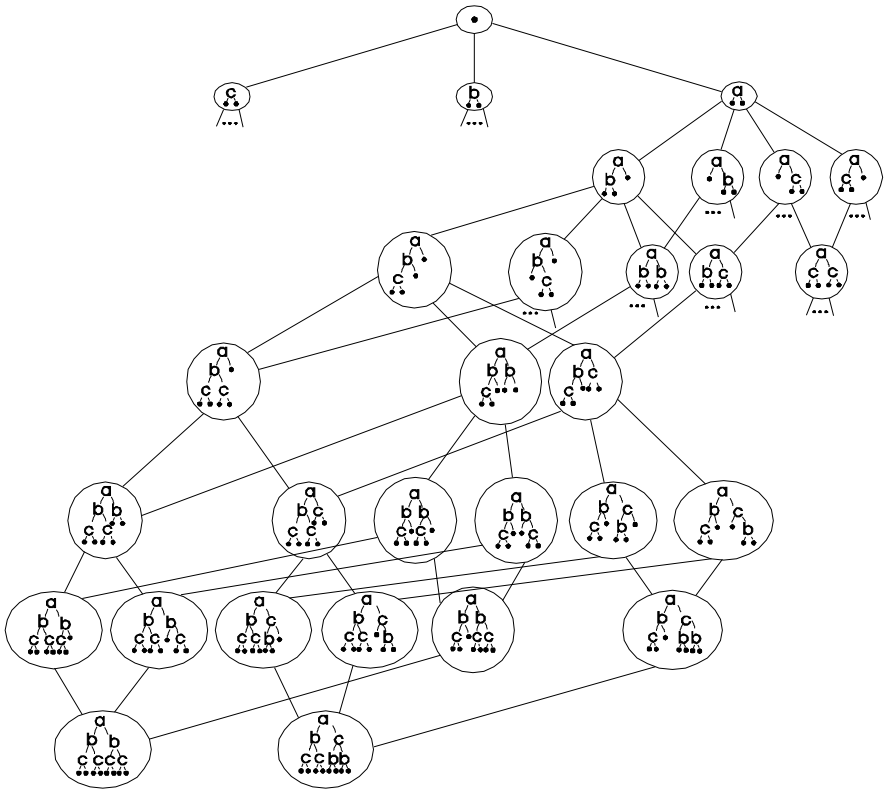

Figure 3.1. Part of space for three attributes. Leaves are marked by bullets

Note 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 a finite upper bound. Moreover, for a pair of trees it may be possible to find two distinct paths between them. The proportion of the number of edges leading to smaller or larger trees is dependent on the number of nodes in the tree. Small trees have larger number of larger trees in their neighborhood, whereas large trees are surrounded by larger number of smaller trees.

The presented space definition has an important asset — it is easy to define popular tree induction algorithms as search task in that space. We could

implement ID3 and the first step of C4.5 algorithm as a search process which starts from an empty tree and goes downwards selecting edges with the use of an impurity function. We could define pruning (the second step of C4.5) as a search process which starts from a certain point of our space (i.e. the result of C4.5 step one) and going towards an empty tree, selecting edges using pessimistic error estimates.

4 Quality function of trees

Each decision tree is a mapping . If a goal is to approximate an unknown mapping then each tree in the search space can be assigned a quality function being the accuracy measure of that approximation. There are many different functions to measure quality of the tree, but the most popular is the classification error and the entropy. A quality function is the objective function to be minimized by a search algorithm.

4.1 Quality function based on error

The classification error is defined as the fraction of misclassified elements from D

$$e(t)=m(\{\mathbf{x}\in D,t(\mathbf{x})\neq c(\mathbf{x})\})/m(D) \tag{4.1}$$

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

$$e_A(t)=m(\{\mathbf{x}\in A,t(\mathbf{x})\neq c(\mathbf{x})\})/m(A) \tag{4.2}$$

4.2 Quality function based on entropy

It is possible to define the quality function based on entropy when the whole tree is treated as an attribute that splits test examples. If so, then we can use equations like those used by the ID3 algorithm

$$E(t(x))=\sum_d P(t(x)=d)E_d(t(x)) \tag{4.3}$$

$$E_d(t(x))=-\sum_c P(c(x)=c|t(x)=d)\log(P(c(x)=c|t(x)=d)) \tag{4.4}$$

where $d.c\in C$ are the decision attribute values, $E(t)$ is the entropy of the tree t and $E_d(t)$ is the entropy of the tree t for category d.

5 Search algorithms

The search algorithms can be divided into three main groups. The first group contains hill climbing methods that choose next point with locally optimal objective function. The second group is random methods where point is randomly chosen. The third group of methods is located between the two first. That group

contains algorithms that choose locally best neighboring point with a certain probability.

We implement ID3 as representative of greedy algorithms, Random as Monte-Carlo-like method and evolutionary algorithm as representative of third group.

5.1 ID3 algorithm

The well known ID3 [14] algorithm was implemented as a search method. The algorithm maintains a single working point from the space of decision trees. The initial working point is an empty tree. In each iteration, a new working point is selected which minimizes the entropy in the neighborhood of the current working point. The algorithm is stopped if in the neighborhood of the current working point there is no other point with smaller entropy.

5.2 Evolutionary algorithm

Evolutionary algorithms (EA) are general search methods inspired by the natural evolution principle. EA maintains a population of points from the search space. In each iteration, the population is processed by the selection and genetic operators (see [2] for detailed description).

In our approach, the population is initialized with empty trees. The nonelitist, tournament selection of size two was used. The mutation is defined as the transition to random neighboring node of tree space. We use very simple function to be minimized, defined as

$$f(t)=e_T(t)+\alpha n(t) \tag{5.1}$$

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.

5.3 Random algorithm

The Random method is our implementation of the Monte Carlo method. We start from an empty tree and repeat the process of randomly choosing a neighboring tree. Assuming that s is the working tree, we select a new working tree t from those neighbors of s which represent a larger tree and such that each terminal node of t is entered by at least one of the training examples. The choice is random, and the selection probability equals

$$P(t|s)=a(s)\left[1-\left(h(t)+1\right)/\left(n+1\right)\right] \tag{5.2}$$

where $h(t)$ is the tree height, n is the number of attributes, and $a(s)$ is a parameter which guarantees that (5.2) defines the probabilities in a proper way. The Random algorithm is used to generate N trees, where N is the user defined parameter, and the best of these is returned as the result.

6 Experiments and results

The algorithms were tested using several datasets from UCI machine learning dataset repository [12] and Orange homepage [1]. A brief characteristic of used datasets is provided in Table 6.1.

Table 6.1. 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

As we can see in Table 6.1 in datasets like “balance scale”, “coil2000” the number of examples belonging to each class is unevenly distributed so these datasets could be hard task for ID3 family. Datasets like “monks-1” and “breast-cancer” are reported to be highly linearly inseparable thus they are also difficult for ID3. The “mushrooms” dataset is known to be easy for ID3 so we could observe how EA and Random manage with it.

In our experiments we assumed that the EA population size equals 20 and the search is terminated after 200 generations (so 4000 trees are examined in total). We assumed α=0.01 in the fitness function formula (4.1). For the Random algorithm we set $N = 4000$.

We treated a well known C4.5 classifier [15] as a reference method. We used the implementation of the C4.5 algorithm provided in WEKA [16] as J48, and EA, ID3 and Random were implemented by ourselves.

Performance of the compared algorithms was analyzed experimentally. Two different methodologies were applied depending on existence of separate test set. When a dataset provided separate test set, we build a tree using the training set and use the test set to verify the tree performance. When there was no separate test set then 10-fold crossvalidation was performed for all algorithms and the average error on the validation set is the performance indicator. ID3 and J48 were run only once, since they are deterministic methods. For EA and Random, 10 independent runs were performed and we report average and standard deviation of their results. The resulting error and tree size values are provided in Table 6.2 and 6.3. In Table 6.4 we provide results when the entropy was assumed as a quality function. Those results were multiplied by 1000 for better readability.

According to Table 6.2 EA in presented version performed generally better than Random and ID3. To our surprise, Random performed comparable or better than ID3. The EA results were comparable to those by J48 except for the "balance scale" and "monks-1" datasets where EA was a winner and moreover,

Table 6.2. Results of the compared algorithms — the percentage of incorrectly classified examples for the test set or the validation set

Error	J48	ID3	EA		Random	
			mean	std. dev.	mean	std. dev.
Balance scale	34.7	29.5	**29.2**	1.20	30.4	1.07
Breast cancer	**28.0**	33.2	29.7	1.48	31.2	2.61
Coil 2000	**6.0**	10.1	**6.0**	0.07	7.4	0.31
Marketing	**67.6**	70.9	68.7	0.46	69.7	0.48
Monks1	24.3	17.4	**9.5**	4.16	14.9	3.98
Mushrooms	**0.0**	**0.0**	0.3	0.22	0.1	0.02

Table 6.3. Results of the compared algorithms — the number of nodes

Size	J48	ID3	EA		Random	
			mean	std. dev.	mean	std. dev.
Balance scale	**41**	485	547	10	162	4
Breast cancer	**7**	381	441	12	223	7
Coil 2000	**1**	31 013	618	127	9 058	980
Marketing	3 582	29 306	**1 181**	22	4 676	82
Monks1	**18**	94	94	17	46	9
Mushrooms	**29**	38	325	26	82	11

Table 6.4. Results of the compared algorithms — entropy of the tree (multiplied by 1000)

Entropy	ID3	EA		Random	
		mean	std. dev.	mean	std. dev.
Balance scale	**226**	233	9	241	7
Breast cancer	225	200	9	**193**	12
Coil 2000	98	**4**	2	42	6
Marketing	815	**700**	8	738	4
Monks1	199	**124**	43	172	32
Mushrooms	**0**	6	4	1	1

even Random performed better than J48. We believe that this is the result of strong functional dependences in these datasets.

As we consider size of trees (Table 6.3), J48 is the winner thanks to the application of pruning. We believe that using pruning for trees generated by EA would let to similar results. We also believe that setting α in equation 5.1 to a larger number would do as well. Even now, there exists datasets, e.g. "marketing", where EA generates trees with similar accuracy but more than 3 times smaller than J48. Random trees are larger than EA trees for large datasets and smaller for small datasets. When we analyze Random algorithm we can see that it stops expanding a node when there are no training example to be used for the split, so for small datasets trees had to be small.

All considered algorithms failed for the "Coil 2000" dataset. J48 achieved minimal error and minimal tree (one leaf). The EA was located on the second place with small trees (~600) when compared to more than 31 thousand of nodes generated by the ID3.

When using entropy as a measure of tree quality (Table 6.4) the superiority of EA over ID3 became even better visible. We could not provide results for C4.5 since it was implemented in Weka.

We also tried to compare our results to the results achieved by the aforementioned GA based approach. Unfortunately, the authors concentrate mainly on datasets with real valued attributes [4, 5, 7, 13], and we were only able to find result of GATree algorithm [13] for the "balance-scale" dataset. The authors report the error value of 28.9%, which is comparable to our results. The resulting tree size of GATree was only 9 nonterminal nodes, but this was achieved by a very strong penalization of the number of nodes.

7 Conclusions

We presented a view of the tree generation as an optimization task in the space of trees. We showed that using general search methods could lead us to produce trees better than ID3 and comparable to J48. We were also able to achieve some unification by defining the standard ID3 algorithm as a search method operating in the same space as EA and Random do.

There are many possible definitions of the space metric and of the quality function which in turn makes the search algorithms work in different ways and yield different results. In this preliminary study we used a simple, poorly tuned EA, a simple space definition and a simple quality function, and even though, the results outperformed those yielded by ID3 and C4.5 in some specific cases. We find this observation very encouraging to lead deeper study in this direction.

References

1. Orange homepage, *http://www.ailab.si/orange/datasets/*
2. Arabas J., (2001). *Lecture Notes on Evolutionary Computation* (in Polish), WNT, Warszawa
3. Bonet B., Geffner H., (1999). Planning as heuristic search: New results. In *ECP*, pages 360–372

4. Bot M., (1999). Application of genetic programming to induction of linear classification trees. Technical report, Vrije Universiteit, Faculteit der Wiskunde en Informatica
5. Cantú-Paz E., Kamath C., (2003). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1)
6. Cichosz P., (2000). *Learning Systems* (in Polish), WNT, Warszawa
7. Eggermont J., Kok J.N., Kosters W.A., (2004). *Genetic Programming for Data Classification: Partitioning the Search Space*", Nicosia, Cyprus, 14-17 March
8. Korf R. E., (1999). Artificial intelligence search algorithms. In *Algorithms and Theory of Computation Handbook*. CRC Press
9. Koronacki J., Ćwik J., (2005). *Statistical Learning Systems* (in Polish), WNT, Warszawa
10. Koza J. R., (1991). Concept formation and decision tree induction using the genetic programming paradigm. In *Parallel Problem Solving from Nature - Proceedings of the 1st Workshop*. Springer
11. Mitchell T. M., (1982). Generalization as search. *Artificial Intelligence*, 18:203–226
12. Newman D.J., Hettich S., Blake C.L., Merz C.J., (1998). *UCI Repository of machine learning databases*, http://www.ics.uci.edu/~mlearn/MLRepository.html
13. Papagelis A., Kalles D., (2000). *GATree: Genetically Evolved Decision Trees*", November
14. Quinlan J. R., (1986). Induction of decision trees. *Machine Learning*, 1:81–106
15. Rokach L., Maimon O., (2005). Top-Down Induction of Decision Trees Classifiers - A Survey, *IEEE Transactions on systems, man and cybernetics - part C: applications and reviews,* November
16. Witten I. H., Frank E., (2000). *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann