Application of genetic programming algorithm for designing decision trees and their ensembles

To cite this article: S A Mitrofanov 2020 IOP Conf. Ser.: Mater. Sci. Eng. 734 012098

View the article online for updates and enhancements.
Application of genetic programming algorithm for designing decision trees and their ensembles

S A Mitrofanov
Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarsky Rabochny Av., Krasnoyarsk, 660037, Russian Federation

E-mail: sergeimitrofanov95@gmail.com

Abstract. Decision tree is a machine learning algorithm that is very effective in classification problems. Decision tree is a topical algorithm because of the good interpretability of the results of their work. However, decision tree has the drawback that standard algorithms don't allow obtaining the optimal structure of the decision tree. To solve this drawback, it's proposed to use the genetic programming algorithm. This algorithm is one of the branches of evolutionary algorithms and has proven itself for the design of intelligent information technologies. Genetic programming, in one of their implementations, searches for solutions in tree space, which is well suited for designing decision trees. Ensembles that are based on decision trees have high efficiency. In this paper, random forest and gradient boosting are considered. In ensembles, it's proposed to combine decision trees that are designed by the genetic programming algorithm. Algorithms was tested on classification problems.

1. Introduction
At the moment, methods based on decision trees are one of the best algorithms for working with structured data [1]. Decision trees are one of the most effective classification methods that has been successfully applied in the practice of data mining [2-4]. The decision trees are widely used due to the intuitive form of the results, which is close to the expert’s formal reasoning. Therefore, increasing the efficiency of methods based on decision trees is an actual area of data analysis.

Learning decision tree algorithms build an “almost optimal” tree, performing a local search in each node in the hope that the resulting tree will be as close as possible to the optimal one. Such algorithms are called “greedy” [5]. In addition, the decision tree should have a high degree of generalization, that is, the algorithm shouldn’t just “remember” the belonging of the objects in the training set to classes, but reflect the dependencies between inputs and outputs [6]. Therefore, in this paper, we consider the possibility of using the genetic programming algorithm. Genetic programming is an effective tool for computer-aided design of information technologies [7]. In one of the variants of genetic programming, individuals are represented in the form of binary trees; therefore, this algorithm is convenient for finding the optimal structure of a decision tree by searching in the space of trees.

The genetic programming algorithm does not perform a local optimum search in each node of the tree; it searches for the optimal structure from the set of all possible structures, performing a stochastic estimate of the gradient of the error function of the decision tree in the tree space.
A self-configuring genetic programming algorithm is used to automate the process of constructing decision trees; it designs decision trees without any imposed restrictions on their structure, in contrast to standard algorithms.

However, decision trees are prone to overfitting regardless of the algorithm of their construction [8]. This drawback can be overcome if the trees are combined into ensembles. The construction of ensembles is one of the most powerful methods of machine learning, which often surpasses other methods in the quality of predictions [9].

In this paper, we consider such methods of constructing ensembles as gradient boosting and random forest.

To increase the efficiency of the compositions, it was proposed to combine the trees designed by the self-configuring genetic programming algorithm in the ensemble [10].

2. Decision trees

The decision tree is a method based on the application of various separation functions of the initial data set, in particular, simple threshold rules [11].

The decision tree is an acyclic graph in which there are two types of vertices: internal and leaf [12].

In the inner node $V_{\text{inner}}$ is a predicate $\beta_V$ that can define an object in the right or left child node $X \rightarrow \{0, 1\}$. A predicate $\beta_V$ is selected from some set of binary predicates $B$. In the simplest case, if all the signs are binary, then these are just the original signs. If the signs are real, then the predicate will be constructed in the form of a binary condition: the value of the sign is greater or less than some value ($\beta_V(x) = x_j \geq \theta_j$, where $\theta_j$ is the threshold value).

At each leaf node $c_V \in Y$, a class label is defined.

Decision trees are an effective method, especially in demand in tasks where interpretation of the results is required, which is understandable to a layman in the field of data analysis. Decision tree learning algorithms have a significant drawback; they can't choose the optimal tree's structure. To combat this drawback, it is proposed to use the genetic programming algorithm.

3. Decision tree design approach

One of the branches of evolutionary algorithms is genetic programming. A characteristic feature of genetic programming algorithms is the tree structure of the individual [13], that is, we can say that this algorithm searches for solutions in the space of trees. This is very suitable for designing decision trees. A tree is a directed graph in which each subsequent node is associated with one and only previous one. Vertices can take values from terminal and functional sets.

![Figure 1. Genetic programming algorithm for designing decision trees.](image-url)
At the initialization stage, a population is formed. The nodes of the tree are randomly filled with threshold functions and class labels (figure 2).

![Figure 2. An example of a functional node with a threshold value p of argument X](image)

Since the pore values can be set up by exhaustive search, as in standard algorithms, it’s impossible due to the non-sequential construction of the tree structure, it’s proposed to use the differential evolution method to set the threshold values.

The set of all threshold values forms a vector \( \bar{x} = (x_1, x_2, ..., x_n) \), where \( n \) is the number of threshold values, that is, the number of terminal nodes. Since for each tree we can represent a set of threshold values by a vector, we can optimize the target function of the classification error using the differential evolution method.

Fitness is calculated using the following formula:

\[
\text{fitness} = 1 - \text{error},
\]

\[
\text{error} = \frac{1}{n} \sum_{i=1}^{n} 1(y_i^* \neq y_i),
\]

where \( n \) is the number of points in the training set. The best is the individual with the highest fitness value.

The stopping criterion is a 100% correct classification of all objects in the training set or the achievement of a given number of generations. Selection and recombination work as in the standard genetic programming algorithm. Mutation consists in changing a tree node to another one of the same types of set. If a node belongs to a functional set, then during mutation the attribute of a threshold function changes, if to a terminal, then the class label changes. When changing the attribute of a threshold function, the threshold value isn’t determined randomly, but is recalculated at the stage of optimization of threshold values. Also, for a greater variety of structures, threshold functions can have comparison signs either more or less (also changes with mutation).

The genetic programming algorithm allows us to choose the structure of the decision tree without any restrictions, which should positively affect the efficiency of the algorithms.

The problem of choosing the optimal settings gives rise to difficulties in the interaction of non-core specialists with evolutionary algorithms. Enumerating all the possible configurations of the algorithm is a very time-consuming process, and often it’s simply impossible. This implies the need for self-tuning evolutionary algorithms [14]. For self-configuring of categorical parameters of the algorithms, the Population-Level Dynamic Probabilities method was used [15]. The adaptation of numerical parameters is carried out according to the Success History Adaptation algorithm [16].

4. Ensembles of decision trees

Another problem of decision trees is their high tendency to overfitting. Using ensemble of decision trees avoids this problem.
Usually, when solving problems of data analysis, the structure is selected and the parameters are adjusted of a separate technology, after which the solution of the problem is trusted in the best method found. However, a certain set of algorithms when used simultaneously, according to the selected collective technology, often allows us to get a better solution. Such a set of algorithms is called an ensemble or composition [17].

An ensemble is a combination of \( N \) algorithms \( b_1(x), \ldots, b_N(x) \) into one. The idea is to train these algorithms, and then average the results:

\[
a(x) = \text{sign} \left( \frac{1}{N} \sum_{n=1}^{N} b_n(x) \right)
\]

This expression allows us to get an answer in classification problems. The ensemble of algorithms \( a(x) \) returns the sign of the average of the results of the \( N \) algorithms \( b_1(x), \ldots, b_N(x) \) that are included in the ensemble. Algorithms \( b_1(x), \ldots, b_N(x) \) are called basic. Such an ensemble of decision trees is called Random Forest.

### 4.1 Random forest

Random forest is machine learning algorithm, which consists in using an ensemble of decision trees and combines the ideas of bagging and the method of random subspaces [18].

In the proposed approach, a given number of trees was generated using the genetic programming algorithm, which were then combined into a standard random forest.

A random forest is an ensemble of deep trees that are built independently of each other. This approach has two significant drawbacks. The first drawback is that deep tree training requires a large amount of computational resources, which is further increased in cases of bulk sampling or a large number of attributes. Limiting the depth of a tree with their subsequent unification into a random forest will not allow achieving the desired results when working with complex patterns.

The second problem of a random forest is associated with the independent construction of trees, which is why the process of building trees is non-directional: each subsequent tree in the ensemble doesn’t depend on the previous ones. For this reason, a large number of trees are needed to solve complex problems.

To solve these problems, it’s possible to use boosting.

### 4.2 Gradient Boost

The main idea of gradient boosting is the sequential construction of basic algorithms, and the construction of each subsequent algorithm is carried out in such a way as to correct the errors of an already formed composition.

Gradient boosting is one of the best ways of directed ensemble construction today. In gradient boosting, the ensemble under construction is the sum of the basic algorithms \( b_1(x) \), and not their averaging [19]:

\[
a_N(x) = \sum_{n=1}^{N} b_n(x)
\]

In the developed approach, at first the decision tree is designed by the genetic programming algorithm. On the training set, classification errors made by the constructed tree are calculated. Based on the errors of the first tree, the second tree is also constructed by the genetic programming algorithm. The next tree will be built by the genetic programming algorithm on the errors of an already formed ensemble, and so on until the number of trees reaches a given number or the error in the training set is equal to zero.

### 5. Solution of classification problems

The proposed approach is designed to increase efficiency in solving classification problems. For testing, eight real tasks were selected from the Machine Learning Repository [20]:

1. Car type recognition.
2. Recognition of the type of urban landscape.
3. Determination of the variety of iris.
4. Diagnosis of Parkinson's disease.
5. Image recognition by segment.
6) Diagnosis of heart disease.
7) Determining the type of soil from satellite images.
8) Determination of biodegradable chemicals.

During the research, the efficiency of decision trees designed by evolutionary algorithms with self-configuring and adaptation of parameters was compared with the results obtained by the decision tree in the RapidMiner data analysis system [21]. Table 1 presents the accuracy of the classification on test problems. Accuracy refers to the proportion of correctly classified objects in the control sample. The genetic programming algorithm was given 100 generations and 100 individuals for the evolutionary cycle. The depth of the trees is from 5 to 9. To optimize the threshold functions by differential evolution, 50 cycles with a population size of 20 were performed. In RapidMiner, the depth of the tree was set to 20. The following notation is introduced in the table: DT GP is results obtained by a decision tree designed by a genetic programming algorithm and DT RapidMiner is results obtained by a standard decision tree implemented in the RapidMiner system.

Table 1. The accuracy of the classification of trees.

| Task number | Classification accuracy | DT GP | DT RapidMiner |
|-------------|-------------------------|-------|---------------|
| Task 1      | 0.762                   | 0.739 |
| Task 2      | 0.73                    | 0.769 |
| Task 3      | 1                       | 0.978 |
| Task 4      | 0.771                   | 0.282 |
| Task 5      | 0.835                   | 0.869 |
| Task 6      | 0.864                   | 0.778 |
| Task 7      | 0.819                   | 0.826 |
| Task 8      | 0.851                   | 0.789 |

Decision trees designed using the genetic programming algorithm are merged into a random forest. The forest was built of 50 trees. The depth of trees designed by genetic programming is from 5 to 9. The random forest settings in RapidMiner are as follows: number of trees is 50, tree depth is 20. The following notation is introduced in Table 2: RF+GP is results obtained by a random forest from decision trees designed by the genetic programming algorithm and RF RapidMiner is results obtained by a standard random forest implemented in the RapidMiner system.

Table 2. Accuracy of random forest classification.

| Task number | Classification accuracy | RF+GP | RF RapidMiner |
|-------------|-------------------------|-------|---------------|
| Task 1      | 0.725                   | 0.613 |
| Task 2      | 0.74                    | 0.578 |
| Task 3      | 1                       | 0.978 |
| Task 4      | 0.811                   | 0.745 |
| Task 5      | 0.876                   | 0.925 |
| Task 6      | 0.897                   | 0.852 |
| Task 7      | 0.84                    | 0.723 |
| Task 8      | 0.82                    | 0.663 |

As for a random forest, trees designed using the genetic programming algorithm are also used for gradient boosting. For both GP and RapidMiner, up to 500 trees were added to the ensemble. The depth of the trees for all algorithms is 3. The results are presented in table 3. The following notation is introduced in the table: GBoos+GP is results obtained by gradient boosting over decision trees designed...
by the genetic programming algorithm and GBoos RapidMiner is results obtained by standard gradient boosting implemented in the RapidMiner system.

| Task number | Classification accuracy |
|-------------|-------------------------|
| GBoost +GP  | GBGBoost RapidMiner     |
| Task 1      | 0.696                   | 0.691               |
| Task 2      | 0.603                   | 0.694               |
| Task 3      | 1                       | 0.978               |
| Task 4      | 0.797                   | 0.78                |
| Task 5      | 0.662                   | 0.778               |
| Task 6      | 0.875                   | 0.778               |
| Task 7      | 0.896                   | 0.826               |
| Task 8      | 0.782                   | 0.789               |

### 6. Conclusion

Designing decision trees using a genetic programming algorithm, as well as combining these trees into ensembles is effective, which is confirmed by tests. The random forest algorithm is turned out to be the most effective compared to other methods, showing an improvement in the result for all tasks except one. This is most likely due to the fact that deep trees are built in a random forest, and it’s more efficient to search for the optimal structure on deep trees. The application of the genetic programming algorithm for the design of intelligent information technologies is an actual and effective area of data analysis.

### References

[1] Michie D, Spiegelhalter D J and Taylor C C 1994 Machine Learning, Neural and Statistical Classification *Ellis Horwood* 289

[2] Quinlan J Ross 1993 C4.5: Programs for Machine learning *Morgan Kaufmann Publishers* p 302

[3] Barsegyan A A, Kupriyanov M S, Stepanenko V V and Kholod I I 2004 *Methods and models of data analysis: OLAP and Data Mining* (BHV-Petersburg, St. Petersburg) p 336

[4] Russel S and Norvig P 2009 *Artificial Intelligence: A Modern Approach* (Pearson Education Limited) p 1099

[5] Barsegyan A A, Kupriyanov M S, Stepanenko V V and Kholod I I 2007 Data analysis technology: Data Mining, Visual Mining, Text Mining, OLAP (BHV-Petersburg, St. Petersburg) p 384

[6] Lipinskiy L V, Kushnareva T V, Popov E A and Dyabkin E V 2014 Hybrid evolutionary algorithm for the automated design of decision trees *Vestnik SibGUA* vol 5 no 57 pp 85–92

[7] Lipinskiy L V and Semenkin E S 2006 Application of genetic programming algorithm in automated design of intellectual information technologies *Vestnik SibGUA* vol 3 no 10 pp 22–26

[8] Ben-Gal I, Dana A, Shkolnik N and Singer 2014 Efficient Construction of Decision Trees by the Dual Information Distance Method *Quality Technology & Quantitative Management* pp 133–147

[9] Schapire R E 1990 The strength of weak learnability *Machine Learning* vol 5 pp 197–227

[10] Mitrofanov S A and Karaseva T S 2017 Symbolic regression problems solving with self-configuring genetic programming algorithm *Materials of 2nd International Conference on International Scientific and Research Conference on Topical Issues in Aeronautics and Astronautics* vol 2 no 13 pp 49–51

[11] Breiman L, Friedman J H, Olshen R A and Stone C T 1984 Classification and Regression Trees/Wadsworth (Belmont, California) p 368
[12] Ayvazyan S A, Enyukov I S and Meshalkin L D 1983 Applied statistics: Fundamentals of modeling and primary data processing (Moscow: Finance and Statistics) p 471
[13] Koza J R 1992 The Genetic Programming Paradigm: Genetically Breeding Populations of Computer Programs to Solve Problems (Cambridge, MA: MIT Press) p 785
[14] Qin A K and Suganthan P N 2005 Self-adaptive differential evolution algorithm for numerical optimization Proceedings of the IEEE congress on evolutionary computation (CEC) pp 1785–91
[15] Gumennikova A V, Emelyanova M N, Semenkin E S and Sopov E A 2003 About evolutionary algorithms for solving hard optimization problems Vestnik SibGAU no 4 pp 14–23
[16] Stanovov V V and Semenkin E S 2012 Study of the effectiveness of various methods of self-tuning of the genetic algorithm Materials of VIII All-Russian Scientific and Practical Conference on International Scientific and Research Conference on Topical Issues in Aeronautics and Astronautics vol 8 pp 319–320
[17] Levin E, Tishby N and Solla S 1990 A statistical approach to learning and generalization in layered neural networks Proc. IEEE (Special Issue on Neural Networks) vol 78 issue 10 pp 1568–74
[18] Breiman L 2001 Random Forests Machine Learning 45: 5. doi:10.1023/A:1010933404324
[19] Friedman Jerome H 2001 Greedy function approximation: A gradient boosting machine Ann. Statist. 29, no. 5, pp 1189-1232. doi:10.1214/aos/1013203451
[20] Tanabe R and Fukunaga A 2013 Success-history based parameter adaptation for Differential Evolution 2013 IEEE Congress on Evolutionary Computation (Cancun) pp 71–78
[21] Gmurman V Ye 2003 Probability Theory and Mathematical Statistics Vyshaya shkola (Moscow, Russia) pp 303–304