Development and approbation of the improved CART algorithm version

L A Demidova¹ and P O Usachev²

¹MIREA Russian Technological University, Moscow, Russia
²Ryazan State Radio Engineering University, Ryazan, Russia

E-mail: pavel.usachev1@gmail.com

Abstract. This article considers the classification problems using the decision trees. It suggests the improved CART algorithm version called clean CART (CCART). The key feature of this algorithm is the reducing of the hardware memory for the tree storing. The main ideas of the CCART algorithm software implementation have been presented. The experimental results of comparison of the original and the presented algorithms for constructing the decision trees confirm that the proposed CCART algorithm has the stated advantage. Therefore, the proposed CCART algorithm is recommended for use in the ensembles of classifiers construction.

1. Introduction

Currently, the various machine learning algorithms are actively used in solving many applied problems, in particular, the SVM algorithm, the algorithms based on the artificial neural networks, the algorithms based on the decision trees [1].

This acute demand for the machine learning algorithms is due to the need to work with the large volumes of the complex structured data sets in order to quickly extract the information hidden in them and to further use it in the decision making, for example, when solving the problems of classifying new data [2].

Each of these algorithms has its own advantages and disadvantages.

So, the SVM algorithm allows to explicitly build the decision rules using the support vectors identified in the training data set, but it requires some effort to select the optimal values of its parameters and the type of kernel function [3].

The algorithms based on the artificial neural networks do not allow explicitly writing down the decision-making rules, and provide, on the one hand, very flexible tools for solving various types of problems, and on the other hand, require the considerable efforts to select the structure and parameters of the neural network [4].

The advantage of the tree-based algorithms is the simplicity of interpretation of the decisions made, while the fine tuning the parameters of decision trees is also necessary [5].

Moreover, all machine learning algorithms are faced with the need to solve the problem of overfitting [6].

In the proposed study, the decision trees were selected as the subject of the study, on the one hand, in view of the good interpretability of the decision-making process and the possibility of using the trees...
in the variety of ensembles, and on the other hand, due to the possibility of realizing the significant improvements from the point of view of the study authors in organizing the process itself building the trees and storing information in the nodes of the tree [7].

The decision trees are widely used models for solving problems of classification, regression, and clustering [8]. At their core, they represent a set of if-else conditions by which a decision can be made. Such conditions can be displayed in the form of the ordinary questions and therefore we can build decision tree “by hand” (which shows one of the positive qualities of trees: they are clear and understandable for humans), but the use of algorithms to build trees based on any statistics is a much more attractive solution [8].

Nowadays, there are several popular algorithms and their modifications that implement the basic principles of the decision-making based on the trees, namely:

- the CART algorithm (Classification and Regression Tree) and its modification with the closed implementation of IndCART, based on the definition of the rule for splitting a node using the Gini index [9];
- the C4.5 algorithm, which is the modification of the ID3 algorithm and based on the determination of the rule for splitting a node by means of the information gain (Gain ratio).

Since the above mentioned algorithms are very sensitive to data and, as a consequence, are prone to overfitting, the balancing algorithms and/or pruning algorithms are used to work with them [10]. Also, the application of various sampling strategies to data on which a tree will be directly constructed can positively affect to the quality of classification. In this case, it is possible to use the decision trees as the part of the certain ensembles with the aim to increase the classification accuracy and solving the overfitting problem [2]. In the course of research, the CART algorithm, whose advantage is the minimum complexity of the pruning cost, will be considered [3].

The tree-based decision-making algorithms have the following advantages:

- the decision-making results are easily interpreted by a human;
- the algorithm is able to interact with various types of data (for example, with the categorical and numeric data);
- the algorithm performs well even in situations when the initial assumptions are violated;
- the algorithm works with the huge amounts of information.

The disadvantages of the tree-based decision-making algorithms are the following:

- the algorithm is usually greedy (it is trying to maximize any attribute) and, as a result, it cannot ensure the optimality of the whole tree;
- the sensitivity of the algorithm to data;
- the tendency of the algorithm to overfitting;
- when implementing the algorithm for the categorical parameters, the parameters whose number is greater receive the large weights [4].

The disadvantages of the tree-based decision-making algorithms are explained by the fact that the tree is constructed using the so-called greedy procedure [11].

However, these disadvantages can be mitigated by the use of the various truncation methods, in particular, the pruning method or the method with the penalty term, which are actively used, for example, when implementing the CART algorithm [12].

2. Development of the improved CART algorithm version

The CART algorithm is relatively simple. It uses the binary trees, that is, each node has two child nodes. At each stage of the development, the node is split according to the rule (in fact, for each sample label), which divides the node into two parts: the part, which meets split condition, and the part, in which the condition is not fulfilled [13]. After that, the resulting set of rules is evaluated using the split quality assessment function (Gini index, Gain ratio, etc.) and the best split rule is selected. The next step is the recursive construction of the tree to specified stopping condition 14].
As the stopping condition, the method of limiting the tree depth can be used, but since this method has a probability of losing the classification quality, the tree is built to the complete “cleanliness” of the node (only sampling elements belonging to the same class remain) and, if necessary, the branch pruning mechanism is used [15].

Let be $T$ the data set, which keeps the information about objects. Each record in the data set describes the pre-selected characteristics of the certain object, and, also, contains the class label for this object.

To assess the quality of splitting in the CART algorithm, the Gini index can be used, which ensures the reduction of the “impurity” in the node:

$$Gini(T) = 1 - \sum_{i=1}^{n} p_i^2,$$

where $p_i$ is the probability (the relative frequency) of the $i$-th class in the data set $T$, which is used for the classifier development.

If the set $T$ is divided into two parts $T_1$ and $T_2$ with the number of examples in each $N_1$ and $N_2$, respectively, then the partition quality indicator will be the following:

$$Gini_{split}(T) = \frac{N_1}{N} \cdot Gini(T_1) + \frac{N_2}{N} \cdot Gini(T_2).$$

The best split is the one for which the $Gini_{split}(T)$ value is minimal [16].

### 2.1. The clean CART algorithm

The CART algorithm is redundant, since each node must store in itself a subset (on the basis of the data set $T$) which was obtained after splitting (figure 1), and must be mutable for the tree construction possibility.

It would seem that this is not a big problem with the modern computer capacities, but with the growth of these capacities the new opportunities and requirements for the classifiers have appeared. In addition, the presence of data, which is necessary only during the construction of the tree and is not needed when using the classifier, in the nodes can confuse and clog the program code with the unnecessary information [17].

Also, the ability to modify the tree, after it has already been built, can lead to errors and non-deterministic behavior of the program, especially when it comes to asynchrony [18]. Therefore, storing the above data in the nodes is redundant and this problem must be addressed [7].

To eliminate the redundancy in the above sense, it is proposed to use the improved CART algorithm version, which is offered to be called “clean” (CCART, Clean CART). In this CART algorithm version, the process of the tree construction is divided into two stages (figure 2).

The first stage is to build the frame. This frame represents classic binary decision tree with additional features. The frame will store the indices of examples from the training set instead of creating the full-fledged training set entity. It should be noted that these indices are easier to store in the list, because immediately before the node splitting is impossible to say how the examples will be distributed in the child nodes of the tree.

The second stage is implemented taking into account the fact that the tree is built and for the already constructed tree there is no need to store the subset. Moreover, when building the ensembles, the expressiveness of each individual tree is lost in the number of trees used in the ensemble. Since it is assumed that this algorithm will be used in the development of the ensembles, we can conclude that this data can and should be cleared, and the nodes of the tree themselves should be made immutable, while the immutability of the nodes must be made explicit (in the sense that it is necessary not to give the user to change the nodes at the program code level).

The pruning, if necessary, must be done while building the frame tree. Hence, it is necessary to build along the frame the new “cleaned” tree, in which there will be no possibility of changing the nodes. This
increases the time for the building of the tree, but at the same time the tree will turn out to be more lightweight due to the fact that the nodes will not store the subsets, and the non-mutable tree nodes will allow the safe using of the trees in the multi-threaded environment [19].

Figure 1. The example of tree in the CART algorithm.

Figure 2. The example of tree in the CCART algorithm.

The upper-level program description of the CCART algorithm implementation in Java can be represented as the following:

```java
public CCART buildCCART() {
    return initWith(this::createFrameTree)
        .andThen(BinaryFrameTree::getRoot)
        .andThen(CCART.BinaryNode::new)
```

2.2. Using the prepruning method to prevent the classifier overfitting

One of the classifiers based on the decision trees main disadvantage is the possibility of overfitting. Nowadays, the various methods are used to prevent the classifier overfitting, for example, the method which implements the tree depth limit, as well as the prepruning and postpruning (otherwise just pruning) methods [19].

The meaning of the pruning methods is to reduce the size of the tree. After the tree has been built, it can be analyzed and rebuilt accordingly.

For example, the CART algorithm will repeatedly split the data set into smaller and smaller subsets until these final subsets become homogeneous in terms of the final variable.

In practice, this often means that finite subsets (known as tree leaves) are made up of one or more examples. In such situations, the tree may be of the poor quality when forecasting on new data.

The alternative to the pruning method is the prepruning method, whose essence is to assess the quality of splitting and stopping the construction of the tree when the quality is below a given threshold.

In this study, the prepruning method as more appropriate is preferred.

When finalizing the CCART algorithm in order to prevent the overfitting, the same methods apply as when finalizing the original CART algorithm, but taking into account the fact that all of them will be applied to the tree frame.

The prepruning method implements the tree building stop, ensured that the new partition does not provide the proper improvement in the classification quality.

The main attention in the implementation of this method should be given to the choice of the threshold that will be used to stop the building of the tree, and to the choice of the criterion by which we will evaluate the quality of the classifier.

The classification quality is represented with various criteria such as the accuracy, the specificity, the sensitivity, etc. Each of them carries the certain meaning.

For example, the accuracy is simply the number of the correct answers at the data set and the specificity measures the proportion of the actual positives that are correctly identified as such.

The most popular method for the classifier quality evaluation is the cross-validation.

The choice of threshold depends on used quality criterion, and to select it correctly, it is necessary to use the quality rating of the final classifier [20].

Moreover, the quality of the classifier must be assessed at each step of the tree construction. If the increase in quality for the resulting partition is less than the specified threshold, then the tree is completed.

Using the prepruning method can cause the tree to stop building too early and, as a result, the quality of the resulting tree will suffer [21].

In this case, for example, it may turn out that the result of the tree construction following the stop could provide a significant reduction in the number of classifier errors.

The prepruning and pruning methods can be used together, individually or not at all. At its core, the prepruning method is a quick fix for heuristics.

When the prepruning method used together with the pruning method, it can save time.

2.3. The implementation development of the prepruning method for the CCART algorithm

When implementing the prepruning method for the CCART algorithm, the most interesting thing is that to evaluate the quality of the classifier after each partition, it is necessary that the frame tree has the ability to classify objects from the test data set.

This raises the problem that the frame tree does not imply the classifier (because if we do not use the pruning method, we simply do not need this functionality) and it turns out that when we add this logic we violate the single responsibility principle.
The solution to this problem may be the implementation of the approach based on the use of the “decorator” design pattern over the frame tree that knows how to classify the data.

The very essence of using the “decorator” design pattern is the ability to dynamically add the new functionality to objects.

The proposed approach allows not to clutter up the usual frame tree with functionality, which it does not need in cases where the pruning method is not necessary.

Similarly, other tree truncation approaches can be implemented, while using the described design pattern allows to combine the implementations and, for example, prune the several methods of pruning and postpruning at once.

This flexibility allows to achieve the required depth of the tree with simultaneous application of several techniques.

2.4. The API development for the CCART algorithm implementation

The API is the important part in any software library. The trees can be represented in various ways depending on the programming paradigm chosen. A herewith, the most popular is the combining of the object-oriented and functional paradigms.

From the point of view of the object-oriented approach for the tree, it is necessary to create the class hierarchy, which in the future will make it easy to expand the API provided by the library [30]. In turn, from the point of view of the functional approach, as mentioned above, the tree is essentially nothing more than the set of rules or predicates, which can be represented as the functions which receive the object at the input which needs to be classified, and predict the output class.

The implementation of the discussed algorithm was implemented as was described above. Building the hierarchy made it easy to implement the “decorator” pattern, which in turn made it possible to flexibly expand the functionality provided by the tree (in particular, the prepruning method).

Moreover, the tree itself is the function, which allows to use the resulting classifier in the functional style. This opens up the possibility of constructing the compositions in the form of currying or partial application in the functional languages.

Since the CART algorithm has the large number of settings, and the process of creating a tree is not trivial, the best solution in this situation is to create the library with the “builder” design pattern for this algorithm.

The essence of the “builder” design pattern is to create the special entity which will be responsible for configuring and creating the class objects so as not to clutter the constructor with this logic and not mislead the users by providing the large complex constructors (or a large number of them), because even in the such concise programming languages as Python, the use of such constructors usually forces users to turn to the documentation.

By applying the design pattern, the builder manages to comply with the principle of the single responsibility and achieve the clearly determined tree development process.

If the prepruning method is not used, the tree will not know anything about it, there is no unnecessary logic in it, and the builder itself does not provide methods for setting up the prepruning method.

In the opposite case, the user explicitly indicates the need to use the prepruning method and the builder provides the additional methods for configuring this functionality. In addition, the frame tree “decorator”, which is specifically designed for prepruning, will be built, although from the user point of view, the necessary classifiers settings are simply indicated. Using of the “builder” design pattern is declarative and it brings usability.

Below is the example of the fully configured builder using the above proposed API.

```
BinaryDecisionTree.builder()
  .trainDataSet(numericDots)
  .nodeQualityEvaluator(QualityIndicators.GINI_INDEX)
  .withPrePruning()
  .threshold(0.2)
```
.classifierQualityEvaluator(qualityControl::calculateAccuracy)
    .build();

A herewith, the common version of the builder is the following.

BinaryDecisionTree.builder()
    .trainDataSet(categoricalDots)
    .nodeQualityEvaluator(QualityIndicators.GINI_INDEX)

The common version of the tree builder does not provide the methods for configuring the prepruning parameters so as not to distract the user's attention. The methods for setting the prepruning parameters are provided by the special tree builder, which will be returned as the result of calling the tree builder method with PrePruning().

The approach using the above design pattern looks especially effective when applied in the languages with static typing due to the fact that the modern IDEs provide the support in terms of outputting the return type of the method, so working with such the design is very convenient.

Also, the classifier quality evaluation API was developed. This API was described as the functional interface: it means that a matching signature method can be used for the quality evaluation. Moreover, API can be used not only for the prepruning. It brings a lot of flexibility in terms of development and testing classifiers.

The following quality evaluation indexes have been already implemented:

- the accuracy;
- the precision;
- the recall;
- the F-measure;
- the specificity;
- the sensitivity.

In addition, due to the use of the “decorator” design pattern, the several methods can be set at once to assess the quality of the resulting partition, which will be used when implementing the prepruning method. This implies the ability to evaluate each partition at once according to several criteria for evaluating the quality of the classifier and to stop the construction of the tree if the partition did not give the good results on one of them.

For example, we can evaluate the quality of the partition at the same time using the accuracy criterion as well as using the F-measure criterion indicating the threshold of quality which the partition must pass in order to continue building the tree and as soon as the quality of the partition by the criterion is less than the corresponding quality threshold, the tree building will be stopped.

3. Experimental results

The suggestions formulated in the theoretical part for making the constructive improvements in the CCART algorithm are implemented in the open source machine learning library in the Java programming language.

During the experiments using the various data sets, the sizes of the formed trees were measured with application of the standard language tools.

The difference between the sizes of the classical binary decision tree constructed in accordance with the CART algorithm and the binary decision tree constructed using the proposed improved version of this algorithm (i.e., using the CCART algorithm) turned out to be six-fold even for small data sets (in particular, for the classical dataset for classifying the irises). In this case, we can conclude that with the growth of the size of the data set, this difference will increase.

Implementation of the CCART algorithm is used in the specially developed implementation of the classifiers ensembles [22]. The main features of this implementation are that the classifier itself is represented by the interface, that allows the using of classifiers, which were constructed by the different
classification algorithms and, moreover, the algorithms that are based on various classification technologies in one ensemble [23].

It should be noted that the proposed implementation does not focus on the specific data storage structure, but uses the iterator over classifiers.

The essence of the iterator pattern (regardless of programming language in which this pattern is implemented) is to allow developer, without focusing on the data storage structure, to bypass it and perform the corresponding calculations. This gives a huge advantage in the sense that the library in question allows to store the ensembles anywhere [24].

So, to store the ensemble, was used the API collections provided by the Java programming language and the distributed Apache ignite cache, but nothing prevents from storing the ensembles, for example, in a database.

Moreover, the proposed CCART algorithm is particularly effective in the distributed systems, since when working in such systems, the size of the used data sets plays a significant role [25].

Tree sizes were measured using the standard Java language tools (namely, the ObjectSizeCalculator object). It should be noted that Java does not have the analogue sizeOf() from C and the size of the object can be estimated only approximately, but even despite the errors in the measurements, one can judge the difference in the sizes of the resulting trees.

Table 1 shows the measurements of trees built at the artificial data set (2 numerical features which display the coordinates along the x and y axes, 2 classes, 50 examples at the data set) and at the classic data set for classifying the irises (4 numerical features, 3 classes, 150 examples at the data set) [26]. The measurements in the cells of table 1 are given in bytes.

|                  | Artificial data set | Iris data set |
|------------------|---------------------|--------------|
| CART             | 1952                | 11856        |
| CCART            | 216                 | 1256         |

Moreover, the classification accuracy for both implementations is the same.

During the research, the trees, which are provided by the sklearn package for the Python language, were measured. These measurements were made using the pympler package.

According to the results of measurements, it can be concluded that the trees built using the sklearn package are heavier (the difference was about 200 bytes for the data set for classifying the irises, it is possible that it will be even larger as the result of further development and improvement of the CCART algorithm).

4. Conclusion

The research results show that the effectiveness of the proposed CCART algorithm increases with increasing the size of data set. This fact allows to conclude that the use of the CCART algorithm is the good way to reduce the volume which the tree occupies in the computer memory.

Moreover, the developed API gives the users of the library the flexibility to customize the process of constructing the classifier. A herewith, the methods to combat the problem of overfitting of the classifier on the basis of the decision tree were implemented.

The implementation in the Java programming language allowed the development of the flexible software architecture which can be easily expanded if necessary, and provided the ability to use the library, for example, for applications written in Scala.

It is planned to conduct research in order to improve both the presented algorithm and its implementation [27]. This can be done through the introduction of various technologies in tree pruning, new metrics for assessing node quality, approaches aimed at accelerating the construction of a tree. Most
interesting goals at the moment are developing a method for reducing the complexity of presented algorithm and developing a decision tree balancing method [28]. Also, for the presented machine learning library, the decision trees based on the LSTM-models are currently being developed [29].

The proposed algorithm can be recommended for use in the ensembles of classifiers [30].

References
[1] Vapnik V, Bousquet O, and Mukherjee S 2002 Machine Learning, 131-159.
[2] Demidova L, Nikulchev E, and Sokolova Y 2015 International Review on Modelling and Simulations 8(4).
[3] Demidova L, Nikulchev E, and Sokolova Y 2016 International Journal of Advanced Computer Science and Applications (IJACSA) 7 16–24.
[4] Wiem E, Kamal M, Noureddine Z, Jacques B, and Ahmad F 2015 IEEE Aerospace Conference: 1–9.
[5] Demidova L, Sokolova Y, Klyueva I, Stepanov N, and Tyart N 2017 Procedia Computer Science, 103, 222.
[6] Chesalin N, Grodzenskiy S, Nilov M, and Agafonov A 2019 Rossiiskii tekhnologicheskii zhurnal (Russian Technological Journal). 20–29 (in Russ.). https://doi.org/10.32362/2500-316X-2019-7-5-20-29.
[7] Usachev P 2017 Applied Research and Technology ART2017: Proceedings of the regional conference, 385-390 (in Russ).
[8] Vapnik V 1998 Statistical Learning Theory.
[9] Breiman L 2001 Random forests Machine Learning 5–32.
[10] Chistyakov S 2013 Proceedings of the Karelian Scientific Center of the Russian Academy of Sciences № 1. 117–136 (in Russ).
[11] Baytas I M, Xiao C, Zhang X, Wang F, Jain A K, and Zhou J 2017 Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining 65 – 74.
[12] Lima F D S, Amaral G M R, Leite L G M, Gomes J P P and Machado J C 2017 Brazilian Conference on Intelligent Systems 222 – 227.
[13] Dinov Ivo D, Heavner B, Tang M, Glusman G, Chard K, and Darcy M et al. 2016 PLoS ONE 11, (8):e0157077.
[14] Saffari, Leistner C, Santner J J 2009 3rd IEEE ICCV Workshop on On-line Computer Vision, 112-127.
[15] Demidova L, Nikulchev E, and Sokolova Y 2015 IREMOS 446-457.
[16] Sigov A S, Andrianova E G, Zhukov D O, Zykov S V and Tarasov I E 2019 Rossiyskiy tehnologicheskii zhurnal (Russian Technological Journal) 7(1) 5–37 (in Russ). DOI: 10.32362/2500-316X-2019-7-1-5-37.
[17] Kingma D P and Ba J L 2014 (Preprint/arXiv.org; № 1412.6980).
[18] Joshua B. 2017 Minimize mutability Effective Java 3rd Edition; 80-87.
[19] Che Chao, Xiao Cao, Liang Jian, Jin Bo, Zhou Jiayu, and Wang Fei 2017 Proceedings of the 2017 SIAM International Conference on Data Mining 198 – 206.
[20] Freund Y and Schapire R 1997 Journal of Computer and System Sciences 119 – 139.
[21] Rokach L 2008 World Scientific.
[22] Cohen S, Rokach L, and Maimon O 2014 Springer-Verlag New York.
[23] Martin R 2009 Systems Clean code: a handbook of agile software craftsmanship; 181-200.
[24] Drucker H and Cortes C 1996 Advances in Neural Information Processing Systems 8, 479-485.
[25] Ding H, Yang L, and Yang Z 2019. 10.1109/ACCESS.2019.2933676, IEEE Access.
[26] Aydin O and Guldamlaslioglu S 2017 4th International Conference on Electrical and Electronic Engineering (ICCEE).
[27] Anantharaman P, Qiao M and Jadav D 2018 IEEE International Congress on Big Data (BigData Congress) 251-254.
[28] Saxena A and Goebel K NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/tech/dash/pcoe/prognostic-data-repository/), NASA Ames Research Center, Moffett Field, CA.

[29] Freund Y 1995 Information and Computation, 121(2), 256-285.

[30] Bitbucket, https://bitbucket.org/Gibmir/entmoot/src/master/, viewed 29 November 2019.