Component-based decision trees for classification

Boris Delibasic[1*] Milos Jovanovic[1] Milan Vukicevic[1] Milija Suknovic[1] Zoran Obradovic[2]

[1] Faculty of Organizational Sciences, University of Belgrade, Jove Ilica 154, Belgrade, Serbia

[2] Information Science and Technology Center, Temple University, Philadelphia, PA 19122, USA

***Abstract*—** Typical data mining algorithms follow a so called "black-box" paradigm, where the logic is hidden from the user not to overburden him. We show that "white-box" algorithms constructed with reusable components design can have significant benefits for researchers, and end users as well. We developed a component-based algorithm design platform, and used it for "white-box" algorithm construction. The proposed platform can also be used for testing algorithm parts (reusable components), and their single or joint influence on algorithm performance. The platform is easily extensible with new components and algorithms, and allows testing of partial contributions of an introduced component. We propose two new heuristics in decision tree algorithm design, namely removal of insignificant attributes in induction process at each tree node, and usage of combined strategy for generating possible splits for decision trees, utilizing several ways of splitting together, which experimentally showed benefits. Using the proposed platform we tested 80 component-based decision tree algorithms on 15 benchmark datasets and present the results of reusable components' influence on performance, and statistical significance of the differences found. Our study suggests that for a specific dataset we should search for the optimal component interplay instead of looking for the optimal among predefined algorithms.

Key Words— Classification, Data mining, Decision tree, Induction, Reusable components, Open-source platform

[*] Corresponding author: T: +381-64-88-93-154, F: +381-11-24-61-221 E:boris.delibasic@fon.bg.ac.rs

1. Introduction

Many decision tree algorithms have been developed, but there is no evidence that any algorithm outperforms all others in every situation. Strong support for this claim is given by No Free Lunch (NFL) theories [27] where authors prove that there is no classification algorithm that outperforms others on every dataset, but one can always find an algorithm that is optimal for a dataset. Therefore, it can be important to broaden the space of available algorithms. Component-based algorithms, derived by combining components from known algorithms or partial algorithm improvements, support this goal.

One problem in using machine learning algorithms is that most users are limited to several available algorithms either incorporated in popular software or frequently used in the research community. Another problem is that algorithms are typically designed in a black-box manner with limited adaptability to datasets through a set of parameters. According to [23] the black-box approach slows down development of data mining algorithms, because new algorithms are developed incrementally and become more complex, therefore reimplementation takes a lot of time. This fact caused a large time gap between the development of algorithms and their application in practice.

The need for standardized algorithm components that can be interchanged between algorithms is reported [23]. The same article also supports the development of open source frameworks that will serve the machine learning and data mining community for fast algorithm development and fair performance comparison between algorithms and their parts. It is also emphasized that such an approach would speed up the development and application of new data mining algorithms because it would enable:

- Combining advantages of various algorithms,

- Reproducing scientific results,
- Comparing algorithms in more details,
- Building on existing resources with less re-implementation,
- Faster adaption in other disciplines and industry,
- Collaborative emergence of standards.

The question of whether the combination of components could improve algorithm performance was asked previously [23]. A positive answer to this question is suggested in our study. Our research supports this problem in three ways:

1. We proposed a framework for storing reusable decision tree algorithm components as well as a structure for combining these components into a generic decision tree (GDT) algorithm.
2. We implemented the components, the GDT algorithm structure, and also a testing framework as open source solutions for a white-box component-based GDT algorithm design which enables efficient interchange of decision tree algorithms components. Our platform (WhiBo) is intended for use by the machine learning and data mining community as a component repository for developing new decision tree algorithms and fair performance comparison of classification algorithms and their parts.
3. We provided statistical evidence that component-based algorithms can outperform, on specific datasets, some commonly used classification algorithms.

2. Related work

A lot of work is being done in developing platforms for machine learning and on software engineering based on reusable components. A brief review of prior work related to our study is contained in this section.

2.1. Machine learning software tools and algorithms

Among the most famous open-source machine learning platforms are Weka [26], R [20] and Rapid Miner [16]. These platforms support various data mining tasks and have capabilities in data preprocessing, model generating, model evaluation, and model exporting. The machine learning algorithms are usually implemented as black boxes, although some effort can be noticed in generalizing algorithms. For example, the authors of Rapid Miner implemented a decision tree which can use different split evaluation criteria (ratio gain from C4.5 [19], information gain from ID3 [18], the Gini impurity measure from CART [4] etc).

In [29] authors propose a DMTL (data mining template library) which consists of generic containers and algorithms for frequent pattern mining. They show that "the use of generic algorithms is competitive with special purpose algorithms".

Some comparison of decision tree design can be found in [17,21]. Key topics important for decision tree construction are discussed in these papers, although no effort is being made towards identifying generic structures and reusable components. There are also many hybrid algorithms in the literature that combine various machine learning algorithms [13, 30]. A popular hybrid approach consists of combining two or more black-box algorithms into one. Frameworks for combining components are, however, rarely found in the literature. One such framework is proposed in [8].

In [11] a framework for fast decision tree construction of large datasets is proposed. The authors analyzed well-known algorithms and improved their performance, but the main goal of their proposed generic decision tree was to improve the scalability of these algorithms on large datasets.

2.2. Reusable components in software engineering

There is no precise way to identify reusable components. Still, reusable design is not new, as it is widely used in software engineering. In software engineering reusable components are defined as triplets consisting of concept, content and context [24]. The concept is the description of what a component does; it describes the interface and the semantics represented by pre and post conditions. The content describes how the component is realized, which is encapsulated and hidden from the end user. The context explains the application domain of the component, which helps to find the right component for a specific problem.

Reusable components, as described at [24] allow, not just the decomposition of an algorithm into smaller units, but also better description of what the component does and when it should be used. This is done better than in classical algorithms that are implemented as a whole, i.e. black boxes, where it is more difficult to describe when the algorithm should be used. Additionally, the user of a white-box algorithm has the ability to adapt and modify an algorithm to specific data or constraints. Here, we believe, further research can be done towards automatic data-driven composition of reusable components into algorithms.

The theory of reusable design, more popularly known as the pattern theory, is grounded in architecture [1], software engineering [10], organizational design [5] and many other areas. The main purpose of these researches is sharing of good ideas and easier maintenance of built systems.

An open list of features that every reusable component should have is proposed in [25], but this list can be used merely as a guideline, and not as a formal tool for components identification. Because there are still no unique agreements of what a reusable component is, we can not imply that the components we identified are the last word in decision tree design.

3. Generic decision trees based on reusable components

In our study, the main principle of component identification was to discriminate algorithm design structure from specific algorithm solutions. The design structure was saved in a generic algorithm shell while the specific solutions were identified as reusable components.

Reusable components (RCs) were identified in well-known algorithms as well as in partial algorithm improvements. We analyzed several decision tree induction algorithms, namely ID3 [18], C4.5 [19], CART [4], and CHAID [12]. The choice of these algorithms was guided by how much they are available within popular software, together with a survey that points out more popular algorithms [28]. Further, we analyzed partial algorithm improvements in [14] and [15]. We identified reusable components classified according to frequently occurring sub-problems in decision tree design. RCs are solutions for sub-problems within the process of inducing the decision tree. For each of these sub-problems, we isolated several solutions as RCs from the original algorithms. The sub-problems play an important role in the decision tree design because for each sub-problem there are many possible solutions, i.e. RCs, and the designer of the algorithms can choose which RC should be used for solving a specific sub-problem.

Table 1 shows basic RCs for tree growth from four algorithms analyzed in this paper which have the same generic structure, but differ in specific solutions for certain sub-problems. The components identified in cells of Table 1 are reusable because there are no barriers for interchanging these components for a sub-problem, as RCs have the same input and output specifications.

From Table 1 we can notice that algorithms can be easily upgraded to handle sub-problems they couldn't originally. For example, the CHAID algorithm cannot handle numerical

(continuous) data, but in a component-based design it can easily adapt RCs for creating possible numerical splits from algorithms that include this feature.

Benefit from the component-based approach includes identifying RCs in algorithms and providing these for use in other algorithms. For example, CHAID includes a RC that groups attribute categories and splits tree nodes on grouped categories. In decision tree growth, instead of trying to branch a categorical attribute on all categories (as in ID3 or C4.5), or make binary groupings (as in CART), CHAID tries to estimate the optimal grouping of attribute categories using the chi-square test for estimating category differences. For an attribute with 5 values, CHAID can decide to group the values into 2, 3, 4 or 5 separate value groups, based on the difference in class distribution.

This behavior can be used independently within any decision tree induction algorithm, as a RC. It could solve the problem of an overly detailed number of categories of an attribute that are not informative for a decision. Nevertheless, we are not aware of any algorithm, beside CHAID, that uses this reasoning to calculate near-optimal category grouping. This approach apparently has been forgotten by the machine learning community.

RCs are combined following a GDT structure presented in Fig. 1. This structure, we believe, suits the analyzed algorithms soundly. It is important to notice that the units in Fig. 1 should not be regarded as algorithmic steps, but as sub-problems that define an algorithm structure.

In Table 2 we show RCs identified in algorithms and partial algorithm improvements we've analyzed and grouped according sub-problems they belong to. The last column in Table 2 shows whether RCs are currently implemented in the open-source platform we propose, and that will be explained later.

The sub-problem “Remove insignificant attributes” (RIA) was inspired by attribute selection in [14]. Although in their paper only one attribute is selected for splitting, we modified this idea to opt out attributes that should not be candidates for tree nodes, thus improving the speed of an algorithm. The component “F test/ Chi square test” uses statistical tests to heuristically find statistically insignificant attributes in every tree node separately.

For “Create split (numerical)” (CSN) we identified one RC, namely “binary”, that is used to divide numerical attribute values into two distinct subsets as proposed at [9]. This RC can be found in C4.5 and CART.

For “Create split (categorical)” (CSC) three RCs were identified. “Binary” is used in CART for generating all possible binary splits, and “Multiway” in ID3 and C4.5 for generating splits on as many branches as there are categories in the attribute. “Significant” was proposed in CHAID and it is used to find near-optimal groupings of categories.

For “Evaluate split” (ES) we identified five RCs. “Chi square” was used in CHAID, “information gain” in ID3, “gain ratio” in C4.5, and “gini” in CART. We also identified the component “distance measure” in [15] which represents a partial algorithm improvement.

For “Stop criteria” we used typical RCs identified in most decision tree algorithms.

The “Prune tree” (PT) components were found in CART and C4.5 algorithm. “Reduced error pruning”, “pessimistic error pruning” and “error-based pruning” prune algorithms are employed in C4.5 while “cost-complexity pruning” is used in CART.

The proposed GDT structure allows the reproduction of analyzed algorithms, although it is not the main goal of the generic algorithms design. A structure which doesn’t allow replication of the original algorithms, but allows idea sharing could still be quite useful. Fig. 2 illustrates how C4.5 can be reproduced using RCs.

Inputs and outputs for every sub-problem are defined at Table 3. These definitions allow generic tree design. In fact, similar to existing algorithms on a higher level of granularity, RCs are also specified by their inputs, outputs, and parameters.

The GDT algorithm is shown in Fig. 3. The proposed GDT can be extended with more RCs or with additional sub-problems that could fit the GDT structure.

4. WhiBo: an open-source framework

We implemented the proposed component-based framework, WhiBo, as a plug-in for Rapid Miner that enables the creation of generic decision tree algorithms for classification. In this platform every constructed algorithm represents one Rapid Miner operator that can be used in the environment together with other operators. That means that algorithms generated from WhiBo are fully integrated with other Rapid Miner operators like IO, data preprocessing, performance evaluation, visualization, learners etc.

The WhiBo generic tree user interface (shown at Fig. 4) contains four panels:

- The left panel contains an array of buttons. Every button represents a concrete sub-problem in decision-tree algorithms construction.
- The central panel allows users to choose an RC for solving a selected sub-problem. Additionally, users can choose multiple RCs in certain sub-problems (e.g. multiple "Stop criteria" RCs, multiple "Create splits" RCs). This panel also enables users to define parameters for selected components.
- The right panel shows the designed GDT structure (selected RCs and their parameters).
- The top panel contains options for creating new, saving current or opening existing component-based algorithms.

For illustration purposes, we will construct two component-based algorithms that differ in a single component. The algorithms will then be applied on the "car" dataset from UCI repository [3].

Fig. 5 shows the definitions of these two algorithms: the one shown on the left panel is the classical CART algorithm and on the right is CART with "multiway" instead of "binary" split for "Create split (categorical)".

The question is will this "slight" difference in algorithm design have effect on the resulting tree model? Fig. 6 shows tree models generated with algorithms defined in Fig. 5.

The complexity of the tree models is obviously affected by this change. This complexity differences can be attributed to change in a single RC. In section five we give evidence that changes in a single component can result in statistically different classification accuracy.

To illustrate the robustness of WhiBo environment, in the next subsection we provide an example of extending a repository by a new RC and sub-problem.

WhiBo can be found at the following web page http://whibo.fon.bg.ac.rs. Data mining and machine learning researchers are invited to join our efforts to exchange components of decision trees and other machine learning algorithms in an open way based on the proposed WhiBo platform, as to establish a standard for interchange of components among decision tree based classification algorithms, as well as other machine learning algorithms.

5. Experiments

Using WhiBo we designed 80 component-based algorithms. The algorithms were created varying RCs from four sub-problems (RIA, CSC, ES, and PT) while the RC used for CSN was constant in all algorithms ("bin"). Algorithms were created by combining RCs shown in Table 4.

Two parameters for splitting and merging in the "chs" component were set to 5% in all algorithms. Sixteen of these eighty algorithms include the RC "all". It is a RC that uses, in each induction step, several methods for splitting categories (in our experiment: "bin" + "mul" + "sig"). In our experiments we wanted to test if broadening the space of candidate splits improves the accuracy of the decision tree algorithms.

We performed three types of experiments in which we tested:

1. Statistical significant differences among 80 component-based algorithms on 15 datasets with a total of 3160x15 pair-wise comparisons,
2. Accuracy, time, tree complexity (weighted average tree depth, number of nodes, etc) of 80 algorithms on 15 datasets, and
3. The "chs/anf" RC's trade-off between time and accuracy.

We conducted these experiments on benchmark datasets chosen from the UCI repository [3]. Acknowledging existing critiques for the usage of such a repository [22], this study still uses the repository since it doesn't try to find a superior algorithm, but merely shows differences in accuracy among components on different datasets. The goal of our research is fundamentally different from what's criticized, since our aim was to find the best algorithm for a single dataset, rather than a superior algorithm that suits all needs. The chosen datasets are shown in Table 5 (column labeled as "Significant differences" will be explained in Section 5.1), while Table 6 lists some basic properties of the datasets.

5.1. Statistical significance of component interchange

The goal of the first set of experiments was to determine if there are statistically significant differences in accuracy among component-based algorithms on a selected dataset. If differences were significant, this would provide evidence that component exchange between algorithms can

help significantly improve accuracy. So, this would suggest that for a specific dataset we should search for the optimal component interplay instead of looking for the optimal among predefined algorithms.

For this experiment we used combined 5 iterations 2-fold cross-validation F-test [2] because it was shown to have considerable statistical power, while keeping the type I error low [7] compared to other popular significance tests.

We compared pair-wise accuracy of 80 algorithms on 15 datasets. In other words, for each dataset we made 3160 pair-wise comparisons searching for a statistically significant difference in prediction accuracy. Significant differences in accuracy were noticed on 13 datasets, while differences above 5% were noticed on six datasets. Datasets in Table 5 are sorted according to the fraction of significant differences found on classification accuracies between algorithms. For example, in car 58% of algorithms pairs showed statistically significant difference in accuracy. On the remaining 2 datasets no statistically significant differences in algorithm accuracy were noticed. In 3160x15= 47,400 tests a total of 15% significant differences between algorithm accuracy were found. This significance was measured with 95% of confidence causing at most 5% false positives.

We were searching for the significantly most accurate algorithm on each dataset, i.e. the winner algorithm on a dataset. We compared algorithms in pairs and calculated a summary score for each algorithm. Algorithms, compared in pairs, received 1 point if there were no significant differences in algorithms accuracy ("a draw"). If there were significant differences the more accurate algorithm got 2 points ("a victory"), and the beaten algorithm 0 points ("a loss"). This way we made for each dataset a scoring for all algorithms. Algorithms had an average score of 79 with 6.4 standard deviation and 83.03 median. The distribution of the algorithms accuracy

scores is shown in Fig. 7. Two groups of algorithms scores are clearly visible where differences within groups are no more than 5 points (i.e. the group on the left achieved from 82 to 87 points, and the group on the right from 69 to 74 points), whereas scores between groups differ at least 8 points.

We grouped algorithms in two classes:

1. Best algorithms, scored in range [82, 87] points, and
2. Worst algorithms, scored in range [69, 74] points.

We labeled each of the 80 algorithms with the class the algorithm belongs to, according to significance scoring, and performed a decision tree algorithm to find rules by which algorithms were assigned to a class. In Table 7 we show extracted rules that can soundly describe the scoring classes with 100 % accuracy.

From Table 7 we see which components were parts of "best" and "worst" class of algorithms. We can conclude that "Remove insignificant attributes" and "Prune tree" have no influence on algorithms being classified as "best" or "worst". Algorithms containing "multiway" are always part of the "worst" algorithms assembly, and also algorithms that combine "all" with "chi-square", "gini" or "information gain".

Popular algorithms reconstructed with RCs are classified according rules in Table 7:

1. C4.5 (!-M-GR-P, !-M-GR-!): **worst**
2. CART (!-B-G-!): **best**, and
3. CHAID (!-S-C-!): **best**.

This picture, though, can be hugely changed on a specific dataset. An algorithm (!-M-C-!) including "multiway" is part of the best algorithms on "aba" dataset, although on average it is classified as "worst".

Table 8 illustrates ten interesting examples of significant differences found in algorithms that differ in only one RC. In row 4 we can see that an algorithm significantly looses accuracy when it uses the "chi-square/anova f test" RC. On the other hand, in rows 7 through 10 algorithms improve accuracy when using this RC.

In our experiments, the "car", "nur", and "tic" datasets showed the largest fraction of significant differences in pair-wise accuracies among 80 algorithms. What's more interesting is that these differences occur between the same algorithms. The statistically significant differences found in "car" are 78% similar to those found in the "nur" dataset, and 70% to those found in "tic". This suggests that algorithms behave similarly on these datasets, which could be due to some intrinsic dataset properties.

Significant differences observed when replacing RCs motivate analysis of classification algorithms on the level of components.

5.2. Performance analysis

In the previous experiment we showed that changing RCs in algorithms can result in statistically significant differences in algorithm accuracy. Our second experiment aims to explore these differences. Here, besides accuracy, we have also recorded additional properties that describe the complexity of the resulting tree model (weighted average tree depth, number of nodes, run time, etc).

These experiments follow a similar scheme to the previous ones, only this time we explore the overall accuracy of component-based algorithms, rather than score from significance of the pair-wise comparisons.

The 80 algorithms' accuracies are distributed as on Fig. 8. The accuracies vary in average between 83.22% and 79.37%. The results are reported for 10-fold cross-validation test with stratified sampling.

The 80 algorithms' average achieved accuracy on all datasets was 81.45, with standard deviation 1.17. We classified algorithms' accuracies into three classes using average accuracy and standard deviation, as algorithms were in average similarly accurate and there weren't well-separated groups of algorithms accuracy:

1. The best algorithms (accuracy > 82.62 (average + stand. deviation)),
2. Average algorithms (accuracy [80.28, 82.62]), and
3. The worst algorithms (accuracy < 80.28 (average – stand. deviation)).

We then used a decision tree algorithm on the dataset (inputs 80 algorithms components, output accuracy class) and discovered 8 rules for the three classes of algorithms, shown in Table 9. The tree showed an accuracy of 97.25 %, with two average algorithms being misclassified as the best.

From Table 9 we can also notice that the PT component had no influence on classification accuracy of algorithms. We see that the RC "chi-square/anova f test" (chs/anf) improves algorithms accuracy in general. Also we notice that "multiway" (mul) algorithms perform badly on average, as do "all" combined with "chi-square" (chs), "gini" (gin), and "information gain" (inf). The best algorithms used for RIA are "chs/anf", for CSC "binary" (bin) and "significant" (sig), or for CSC "all" with ES RCs "distance measure" (dis) and "gain ratio" (gai). This indicates that some RCs are more preferable than others on average.

Using this rule we classified the popular algorithms reconstructed with RCs as:

1. C4.5 (!-M-GR-P, !-M-GR-!): **worst**
2. CART (!-B-G-!): **average**, and

3. CHAID (!-S-C-!): **average**.

We notice that the three famous algorithms are not part of "the best" algorithms class in general and that C4.5 performed on the selected 15 datasets in average very bad. Although the "chs/anf" RC raises accuracy in general, according to the results presented in section 5.1, this raise of accuracy is not statistically significant. We must also keep in mind that average accuracy is not representative for all datasets.

The distribution of "car" dataset accuracies is shown on Fig. 9. Algorithms achieved average accuracy 94.48% with standard deviation 3.63%. We classified algorithms by accuracy into three classes: [98,09 – 96.76], [91.84-91.49], and [89-88.71] where accuracies inside groups differ no more than 1.33, and accuracies between neighboring groups differ at least 2.49.

We present rules from a decision tree model (100% accurate) in Table 10. Using this rules we classified the popular algorithms reconstructed with RCs as:

1. C4.5 (!-M-GR-P, !-M-GR-!): **worst**
2. CART (!-B-G-!): **best**, and
3. CHAID (!-S-C-!): **best**.

On the "car" dataset the component "chs/anf", which was in average part of the best accuracy group, had no influence on an algorithm being classified in the best accuracy group.

Accuracy of algorithms varies between datasets. A RC performing well on one dataset can perform poor on another dataset. However, we noticed that 80 algorithms had similar accuracy patterns on "nur", "car", and "tic" dataset. We show accuracy patterns for algorithms using "chs/anf" (Fig. 10) and for algorithms not using this RC (Fig. 11).

On these three datasets, and possibly some other datasets, algorithms behavior could follow the same pattern. This opens the question whether these datasets are somehow similar. If they

were similar in some measurable way, it would be possible to predict algorithms' performance, based on the performance on similar datasets. This could potentially be used to aid the algorithm selection process.

Algorithms using "mul" RC on these three datasets results in accuracy loss. Similar accuracy loss was observed for "chs" and "gin" when combined with "all". We can also notice that "chs/anf" reduces accuracy when used with "mul", and "all" on "tic", but improves accuracy on "car".

We showed that effectiveness of using a RC in an algorithm is dataset dependent, and that there are datasets where algorithms perform similarly. This indicates that RC performance should be related to dataset properties.

One further point in our research was that component-based design enables easier analysis of partial algorithm improvements. We only give some indications for this. Figs 12 (a) and (b) show average accuracy and average run time of algorithms using CSC RCs. Run time was measured on the level of algorithms.

From Figs 12 (a) and (b) we notice that algorithms including "bin" had the best accuracy in average, but second worst run time. We expected benefit in average accuracy using "all" because it generates the widest space of candidate splits. However, there wasn't. Moreover, "all" only outperformed "mul", and in addition had the longest processing time. Fig. 12 (a) show that "sig" has comparable accuracy with "bin", but computes faster as shown at Fig. 12 (b).

Although algorithms using "all" showed on average not as the best alternative there are, however, datasets where "all" is part of the top ranking algorithms. On the "car" dataset "all" belongs to the group of most accurate algorithms when combined with "distance", while when combined with "chi", "inf", or "gin", was performing bad, or average when combined with

"chs/anf", on this dataset (Table 10). This indicates biases present in evaluation measures towards "deep" or "shallow" trees.

The "sig" RC seems as an excellent choice in average, because it performs similar to "bin" but requires less computational time.

We explored interactions between "Create splits (categorical)" RCs and "Evaluate split" RCs. These are shown on Fig.s 13-14. We used "all" to test if there are biases between ES RCs and CSC RCs. As Fig. 13 shows, "all", combined with "chs", "gin" or "inf", performs similarly to "mul", while "all" combined with "dis" or "gai" performs similarly to "bin" and "sig".

This indicates that "chs", "gin", and "inf" are biased towards choosing "mul" splits, while "gai" is biased towards bin. This is also indicated by the results summarized in Fig. 14, where a similar pattern of behavior can be noticed on the average number of tree nodes.

We measured tree complexity, also, with *weighted average tree depth* (WATD) which can be calculated as the average product of leaf's depth and their corresponding number of cases (instances)

$$WATD = \frac{\sum_{i=1}^{l} d_i * c_i}{\sum_{i=1}^{l} c_i}$$

where d_i is a leaf's depth, c_i is the number of cases in leaf i, and l is the total number of leafs. This measure indicates the average length of the tree path needed to classify an example. The results are shown in Fig. 15. We see that "gai" is consistently part of the "deepest" trees while "inf" produces the "shallowest" trees.

In Fig. 16 we show that algorithms that evaluate split based on "gai" consistently required most processing time, but they were also accurate. On the other hand algorithms with split evaluation based on "dis" were fast and accurate (Fig. 13).

Finally, we show algorithms' average accuracies on fifteen datasets. The datasets in Table 11 are sorted according to the number of significant differences found in the previous section. The column "Max-Min" shows the difference between the best performing algorithm and worst performing algorithm on a dataset. These differences must be used with caution, because they tell as not much about significant differences between algorithm accuracy found in data (Table 5).

5.3. Analysis of tradeoff between accuracy and speed when removing insignificant attributes

Algorithms including the "chs/anf" RC performed better in average by accuracy. We wanted to test how this RC influences accuracy and computational speed. This time we didn't remove attributes that were insignificant at a predefined threshold (e.g. 5%) but sorted all attributes by significance, in each induction step, and removed a defined percentage of the least significant attributes. We tested the 80 algorithms with four experiments:

1. Experiment 1: In each node we removed 40% of the least significant attributes. They weren't used further for split evaluation.
2. Experiment 2: In each node we removed 60% of the least significant attributes.
3. Experiment 3: In each node we removed 80% of the least significant attributes.
4. Experiment 4: In each node we found only the most significant attribute like in [13] and used it for splitting.

In Tables 12 and 13 we show the averaged results of our experiments. The best average values for each dataset, i.e. row are bolded, while the worst values are underlined. On one hand, we can make the conclusion that reducing the number of attributes in each induction step reduces on most datasets computational speed. It would be expected that Experiment 4 needs the least

computational time, while Experiment 1 should need the most time. This, however, doesn't happen on all datasets because the time needed to calculate the significance of attributes was not compensated by reduced calculations performed after usage of this RC. A more detailed analysis, which is out of the scope of this paper, would be needed to analyze these findings more thoroughly.

On the other hand, RIA reduces average accuracy of decision tree classifiers (Table 11). The difference between the best and poorest accurate average accuracy achieved on experiments is shown in the rightmost column in Table 13. These differences are small compared to the results shown in Table 11.

In contrary to what would be expected, there are some datasets where accuracy improves when choosing only the most significant attribute (e.g. "tic", "cmc", and "adv"). So, using this RC is recommended on most datasets, because it can reduce computational time, but still not decreasing accuracy too much.

Obviously, one can always find a dataset where this doesn't hold, so it is important to find for each dataset an appropriate RC interplay.

6. Conclusion and future research

Reusable component design is a relatively new research topic in data mining. Although classical algorithms are well-established and widely used, we show that there are still many insufficiently exploited research possibilities within these algorithms when looking at the components level.

We proposed a white-box decision tree design approach that is aimed to help the cost-effective design of classification algorithms that could perform better in specific situations. We showed

that there is significant statistical evidence that such a white-box approach can produce more useful algorithms.

Three experiments reported in this article provide evidence that a component-based approach can outperform existing decision tree algorithms on specific datasets.

We conclude that:

1. Component-based algorithms are useful for testing of performance influences of each algorithm part, and enables easy construction of new algorithms, that can show better performance.
2. Algorithm (and component) performance is influenced by the interplay between the components on a specific dataset, so using the same "black-box" algorithms generally does not give the best results. Experimentally, well known algorithms also did not rank best on average, which should inspire usage of component-based algorithms.
3. "Remove insignificant attributes" RC (when used with 5% threshold), that is used in each induction step, improves algorithms accuracy in average.
4. It can be beneficial to use "all" RC (as a union of "binary", "multiway", and "significant") in algorithms design.
5. RCs "sig" and "dis" seem as the best alternative for algorithms design if the trade-off between accuracy and time is compared.
6. "Remove insignificant attributes" RC (when used for removing a defined percentage of least significant attributes) has a good trade-off between accuracy and time reduction, so it is recommended to be used.
7. One can always find a dataset where the previous conclusions do not hold.

The big question "why" a particular RC performs better on specific datasets remains an issue for further research. Solving this could help better describe and improve parts of algorithms. Further experiments are needed to explore the relationship between dataset properties and RC performance. There is also the possibility of analyzing how RCs are interacting to improve performance.

This study was focused on decision trees, but it is also applicable to other kinds of classification models and other families of machine learning algorithms. For example, a generic algorithm for partitioning clustering is already proposed in [6].

Another research direction for the future is to solve the problem of finding the most appropriate algorithm for a problem when the number of possible algorithms is huge. By expanding the number of RCs and sub-problems of a generic algorithm, finding the most appropriate algorithm will a challenging task. We believe that meta-heuristics, like genetic algorithms or variable neighborhood search, could be used in further research.

Our research supports the call for standardization of components and algorithms [23]. Standardization would enable easier and faster interchange of algorithm ideas and implementations. We offer WhiBo as a framework towards this standardization.

Acknowledgements

This research was partially funded by a grant from the Serbian Ministry of Science and Technological Development, project ID TR12013, 2008-2009.

References

[1] C. Alexander, *The timeless way of building*, Oxford University Press, 1979.

[2] E. Alpaydin, Combined 5 x 2 cv F test for comparing supervised classification learning algorithms, *Neural Computation* **11** (1999), 1885-1892.

[3] A. Asuncion and D.J. Newman, *UCI machine learning repository*, University of California, School of Information and Computer Science. [www.ics.uci.edu/~mlearn/MLRepository.html], 2007.

[4] L. Breiman, J. Friedman. C.J. Stone and R.A. Olshen, *Classification and regression trees*, CRC Press, 1984.

[5] J.O. Coplien and N.B. Harrison, *Organizational patterns of agile software development*, Prentice Hall PTR, 2005.

[6] B. Delibasic, K. Kirchner, J. Ruhland, M. Jovanovic and M. Vukicevic, Reusable components for partitioning clustering algorithms, *Artificial Intelligence Review* (2009). Doi: 10.1007/s10462-009-9133-6

[7] T. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, *Neural Computation* **10** (1998), 1895-1923.

[8] N. Drossos, A. Papagelis and D. Kalles (2000) Decision tree toolkit: A component-based library of decision tree algorithms, *Lecture Notes in Computer Science* (2000), 381-387.

[9] U.M. Fayyad and K.B. Irani, On the handling of continuous-valued attributes in decision tree generation, *Machine Learning* **8** (1992), 87-102.

[10] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: Abstraction and reuse of object-oriented design, in: Proceedings of the 7th European conference on object-oriented programming, G. Goos and J. Hartmanis, ed., Springer, Berlin/Heidelberg, 1993, pp. 406-431.

[11] J. Gehrke, R. Ramakrishnan and V. Ganti, RainForest: A Framework for fast decision tree construction of large datasets, *Data Mining and Knowledge Discovery* **4** (2000), 127-168.

[12] G.V. Kass, An exploratory technique for investigating large quantities of categorical data, *Applied Statistics* **29** (1980), 119-127.

[13] R. Kohavi, Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid, in: *KDD 96*, 1999, pp. 202-207.

[14] W.Y. Loh and Y.S. Shih, Split selection methods for classification trees, *Statistica Sinica* **7** (1997), 815-840.

[15] R.L. Mantaras, A distance-based attribute selection measure for decision tree induction, *Machine Learning* **6** (1991), 81-92.

[16] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz and T. Euler, YALE: Rapid prototyping for complex data mining tasks. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2006, pp. 935-940.

[17] S.K. Murthy, Automatic construction of decision trees from data: A multidisciplinary survey, *Data Mining and Knowledge Discovery* **2**, 345-389.

[18] J.R. Quinlan, Induction of decision trees, *Machine Learning* **1** (1986), 81-106.

[19] J.R. Quinlan, *C4.5 programs for machine learning*, Morgan Kaufmann, 1993.

[20] R Development Core Team, R: A language and environment for statistical computing, *R Foundation for Statistical Computing*, Vienna, Austria, 2008 [http://www.R-project.org].

[21] S.R. Safavian and D. Landgrebe, A survey of decision tree classifier methodology, *IEEE Transactions on System, Man, and Cybernetics* **21** (1991), 660-674.

[22] S. Salzberg, On comparing classifiers: A critique of current research and methods, *Data Mining and Knowledge Discovery* **1** (1999), 1-12.

[23] S. Sonnenburg, M.L. Braun, C.S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.R. Mueller, F. Pereira, C.E. Rasmussen, G. Raetsch, B. Schoelkopf and A. Smola, The need for open source software in machine learning, *Journal of Machine Learning Research* **8** (2007), 2443-2466.

[24] W. Tracz, Where does reuse start?, *ACM SIGSOFT Software Engineering Notes* **15** (1990), 42-46.

[25] T. Winn and P. Calder, Is this a pattern?, *IEEE Software* **19** (2002), 59-66.

[26] I. Witten and E. Frank, *Data mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[27] D.H. Wolpert, The lack of a priori distinctions between learning algorithms, *Neural computation* **8**(1996), 1341–1390.

[28] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z.H. Zhou, M. Steinbach, D.J. Hand and D. Steinberg, Top 10 algorithms in data mining, *Knowledge information systems* **14** (2008), 1-37.

[29] M.J. Zaki, N. Parimi, N. De, F. Gao, B. Phophaakdee, J. Urban, V. Chaoji, M. Al Hasan and S Salem, Towards generic pattern mining, *Lecture Notes in Computer Science* (2005), 91-97.

[30] Z.H. Zhou and Z.Q. Chen, Hybrid decision tree, *Knowledge Based Systems* **15** (2002), 515-528.

Sub-problem	Algorithm			
	ID3	C4.5	CART	CHAID
Create split (numerical)	Multi-way	Multi-way	Binary	Signi-ficant
Create split (categorical)	None	Binary	Binary	None
Evaluate split	Infor-mation gain	Gain ratio	Gini	Chi^2

Table 1. Basic sub-problems and reusable components for tree growth of ID3, C4.5, CART, and CHAID

Sub-problem	**Reusable component**	**Abbreviation and code (in brackets)**	**Available in WhiBo**
Remove insignificant attributes	CHI SQUARE/ANOVA F TEST	chs/anf (C)	X
Create split (Numerical)	BINARY	bin (B)	X
Create split (Categorical)	BINARY	bin (B)	X
	MULTIWAY	mul (M)	X
	SIGNIFICANT	sig (S)	X
Evaluate split	CHI SQUARE	chs (C)	X
	INFORMATION GAIN	inf (I)	X
	GAIN RATIO	gai (GR)	X
	GINI	gin (G)	X
	DISTANCE MEASURE	dis (D)	X
Stop criteria	MINIMAL GAIN	mga	
	MAXIMAL TREE DEPTH	mtd	X
	MINIMAL NODE SIZE	mns	X
	MINIMAL LEAF SIZE	mls	X
Prune tree	REDUCED ERROR PRUNING	rep	
	PESSIMISTIC ERROR PRUNING	pep (P)	X
	ERROR-BASED PRUNING	ebp	
	COST COMPLEXITY PRUNING	ccp	

Table 2. Sub-problems and RCs identified for GDT

Sub-problem	Input	Output
Remove insignificant attributes	Dataset in current node	Dataset in current node (reduced)
Create split	Dataset in current node	A split candidate
Evaluate split	A split candidate	The best split in current node
Stop criteria	Current tree model	Signal for stopping tree growth in current node
Prune tree	Current tree model	Pruned tree model

Table 3. Sub-problems with their inputs and outputs

Sub-problems	Reusable components				
RIA	chs/anf	none (!)			
CSC	mul	bin	sig	all	
ES	gai	inf	gin	dis	chs
PT	pep	none (!)			

Table 4. RCs used for creation of 80 algorithms

ID	Dataset	Significant differences
car	Car evaluation	58 %
nur	Nursery	43 %
tic	Tic-tac-toe endgame	39 %
aba	Abalone	19 %
cmc	Contraceptive method choice	18 %
spe	SPECT Heart	18 %
con	Connect-4	5 %
cre	Credit approval	5 %
cov	Cover type	5 %
adu	Adult	4 %
kin	King-rook vs. king-pawn	4 %
len	Lenses	2 %
vot	Congressional voting records	1 %
thy	Thyroid disease	0 %
adv	Internet Advertisements	0 %

Table 5. Fifteen benchmark datasets

ID	No. cat. attrib.	No. num. attrib.	No. records	No. classes
Car	6	0	1728	4
Nur	8	0	12960	4
Tic	9	0	958	2
Aba	1	7	4177	3
cmc	2	7	1473	3
Spe	22	0	187	2
Con	42	0	67557	3
Cre	9	6	690	2
Cov	10	44	581012	7
Adu	8	6	32561	2
Kin	36	0	3196	2
Len	5	0	24	3
Vot	16	0	435	2
Thy	22	6	2800	3
Adv	1555	3	2369	2

Table 6. Basic properties of benchmark datasets

Rule	CSC	ES	Class
1	"bin", "sig"		best
2	"all"	"dis", "gai"	best
3	"all"	"chs", "gin", "inf"	worst
4	"mul"		worst

Table 7. Rules extracted from decision trees that classified algorithms according to significance scores

	Winner	Looser	Dataset
1	!-A-D-!	!-A-I-!	car
2	!-A-D-!	!-A-G-!	car
3	C-S-C-!	C-B-C-!	nur
4	!-S-G-!	C-S-G-!	nur
5	**!-M-GR-!**	C-M-GR-!	tic
6	!-A-GR-!	**!-M-GR-!**	tic
7	C-B-G-!	**!-B-G-!**	cmc
8	C-S-C-!	**!-S-C-!**	spe
9	C-S-D-!	!-S-D-!	con
10	C-S-D-!	!-S-D-!	aba

Table 8. Some comparisons of classifiers differing significantly in accuracy

Rule	RIA	CSC	ES	Class
1	"chs/anf"	"all"	"dis", "gai"	Best
2	"chs/anf"	"bin", "sig"		Best
3	!	"all"	"dis", "gai"	Average
4	!	"bin", "sig"		Average
5	"chs/anf"	"all"	"chs", "gin", "inf"	Average
6	"chs/anf"	"mul"		Average
7	!	"all"	"chs", "gin", "inf"	Worst
8	!	"mul"		Worst

Table 9. Rules extracted from decision trees that classified algorithms by accuracy on 15 datasets

Rule	RIA	CSC	ES	Class
1		“bin”,“sig”		Best
4		“all”	“dis”, “gai”	Best
2	“chs/anf”	“mul”		Average
5	“chs/anf”	“all”	“chs”, “gin”, “inf”	Average
3	!	“mul”		Worst
6	!	“all”	“chs”, “gin”, “inf”	Worst

Table 10. Rules extracted from decision trees that classified algorithms by accuracy on the “car” dataset

Dataset	Average	Max	Min	Max-Min
car	94.48%	98.09%	88.71%	9.38%
nur	98.86%	99.88%	97.01%	2.87%
tic	88.46%	94.26%	78.71%	15.55%
aba	72.84%	78.00%	63.50%	14.50%
cmc	50.59%	55.19%	46.23%	8.95%
spe	87.88%	91.99%	82.25%	9.74%
con	62.96%	66.66%	58.97%	7.69%
cre	81.67%	84.35%	77.54%	6.81%
cov	60.97%	64.59%	57.77%	6.82%
adu	76.84%	80.34%	74.18%	6.16%
kin	99.58%	99.69%	99.47%	0.22%
len	75.33%	78.33%	71.67%	6.67%
vot	93.06%	94.46%	91.70%	2.76%
thy	94.52%	95.71%	92.86%	2.86%
adv	83.67%	85.00%	80.00%	5.00%

Table 11. Results of 10-fold cross-validation of 80 component-based algorithms on 15 datasets

Dataset	1. 40% attributes removal	2. 60% attributes removal	3. 80% attributes removal	4. Using the most significant attribute
car	00:15.1	00:12.7	**00:11.5**	00:13.7
nur	03:09.6	03:05.0	**02:42.6**	02:50.0
tic	00:09.6	00:08.1	**00:05.8**	**00:05.8**
aba	01:03.2	00:39.4	**00:28.4**	00:29.9
cmc	01:55.5	01:51.9	01:34.2	**01:14.1**
spe	00:02.2	00:01.8	**00:01.4**	**00:01.4**
con	01:47.3	01:25.3	00:48.8	**00:33.9**
cre	14:14.9	08:13.7	02:29.8	**02:22.8**
adu	07:40.6	04:26.5	01:40.7	**01:30.6**
kin	01:04.7	02:57.9	00:45.8	**00:41.5**
cov	01:56.8	02:44.5	02:05.4	**01:01.3**
len	00:00.0	00:00.0	00:00.0	00:00.0
vot	00:04.5	00:04.1	00:03.4	**00:03.1**
thy	00:59.7	00:54.2	00:46.2	**00:01.3**
adv	00:32.6	00:33.7	00:25.6	**00:17.8**

Table 12. Average run time of 80 algorithms on 15 datasets in four experiments

Dataset	1. 40% attributes removal	2. 60% attributes removal	3. 80% attributes removal	4. Using the most significant attribute	Max - Min
car	**93.69%**	93.66%	93.60%	93.60%	0.09%
nur	**99.01%**	98.95%	98.97%	98.98%	0.06%
tic	90.26%	90.34%	90.65%	**90.66%**	0.40%
aba	**75.75%**	75.41%	75.10%	75.13%	0.65%
cmc	48.73%	48.53%	48.96%	**48.97%**	0.44%
spe	**84.97%**	84.74%	84.20%	83.42%	1.55%
con	61.78%	61.63%	**62.72%**	61.24%	1.48%
cre	**79.95%**	79.34%	79.30%	79.56%	0.65%
cov	60.72%	**60.81%**	60.18%	59.10%	1.71%
adu	**75.94%**	75.57%	74.86%	74.83%	1.11%
kin	**99.60%**	98.95%	99.58%	99.53%	0.65%
len	**72.92%**	72.75%	72.75%	72.75%	0.17%
vot	**92.45%**	92.42%	**92.45%**	92.39%	0.06%
thy	96.11%	96.20%	**96.37%**	90.12%	6.25%
adv	83.50%	83.50%	83.00%	**84.29%**	1.29%

Table 13. Average accuracy of 80 algorithms on 15 datasets in four experiments

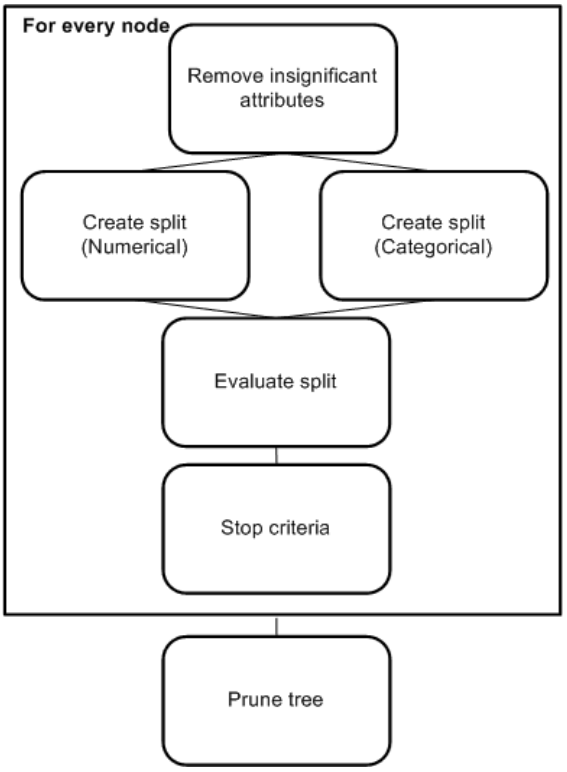


Fig. 1. Generic decision tree structure

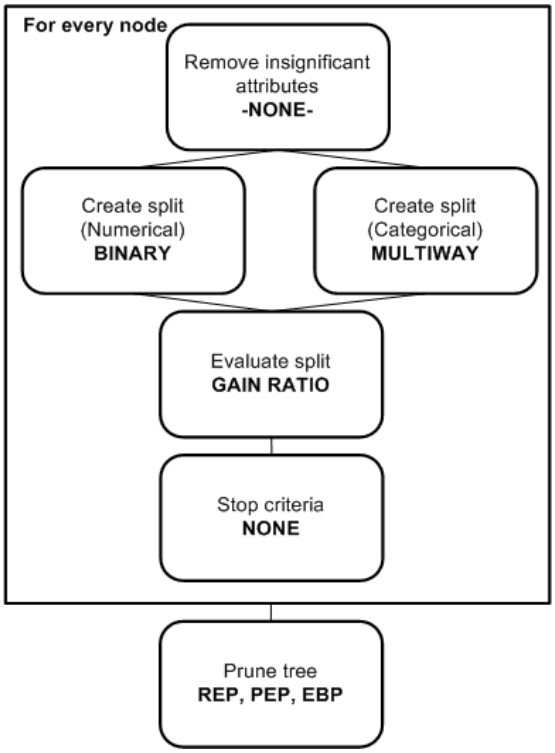


Fig. 2. Example of an algorithm (C4.5) designed with RCs

Generic decision tree algorithm

Input: Dataset
Output: Decision tree model

Step 1. Optionally, use “Remove insignificant attributes” to eliminate uninformative attributes in the current tree node.
Step 2. Use “Create split” to create candidate tree split in the current tree node.
Step 3. Use “Evaluate split” to measure the “goodness” of the candidate split.
Step 4. If the candidate split is better than the best split, remember new candidate split as best.
Step 5. Repeat steps 2-4 until no more candidate splits are produced by “Create split”.
Step 6. If “Stop criteria” is met, create a leaf node and add it to the tree model.
Otherwise, split the dataset according to the best split, and recursively return to step 1 for each new branch of the node.
Step 7. Optionally, after the tree model is built use “Prune tree” to shorten branches which are uninformative according to the prune criteria.

Fig. 3. The GDT algorithm

Fig. 4. WhiBo GDT user interface

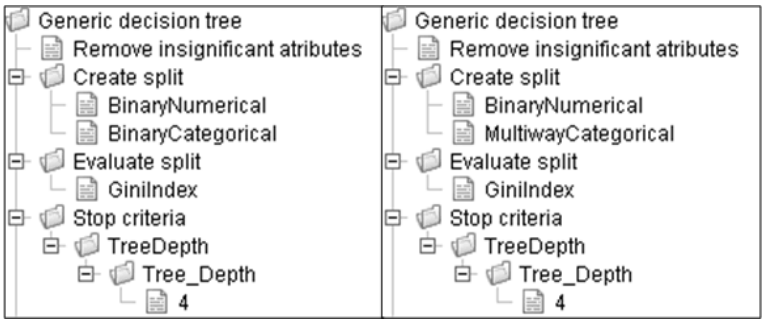
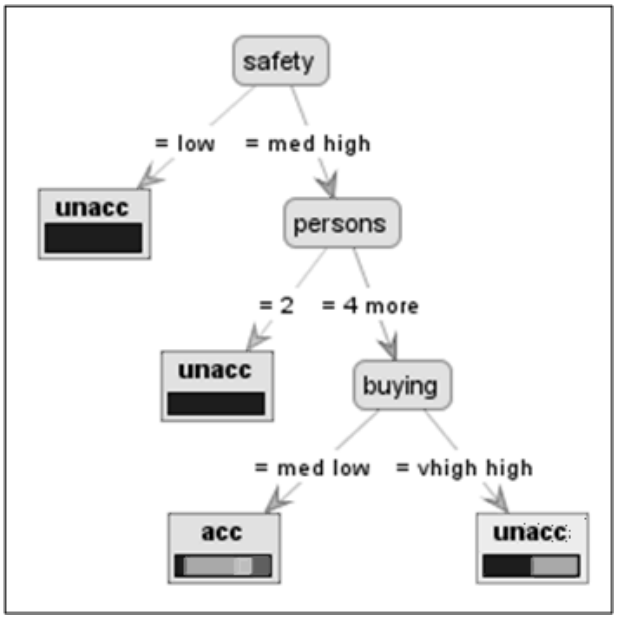
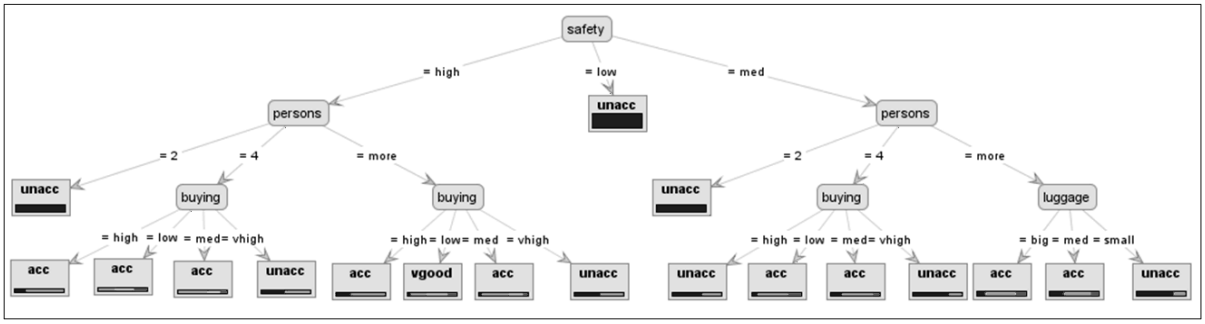


Fig. 5. Definitions of two algorithms differing in a single component (binary vs. multiway split)



6 (a) The model obtained by the first algorithm



6 (b) The model obtained by the second algorithm

Fig. 6. Two classification models differing in a single component

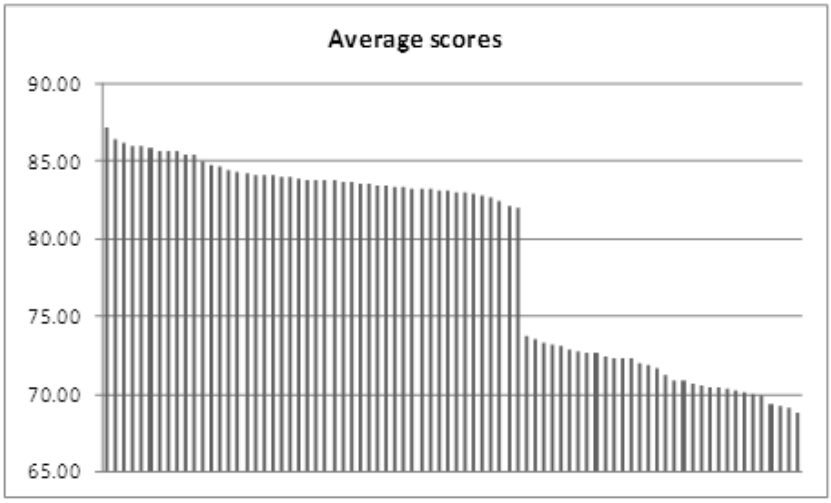


Fig. 7. Distribution of 80 algorithms' scores

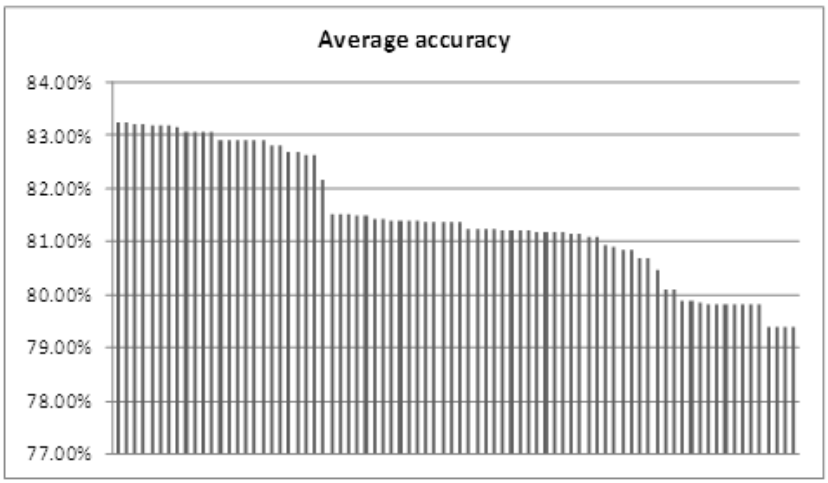


Fig. 8. Distribution of 80 algorithms' accuracies

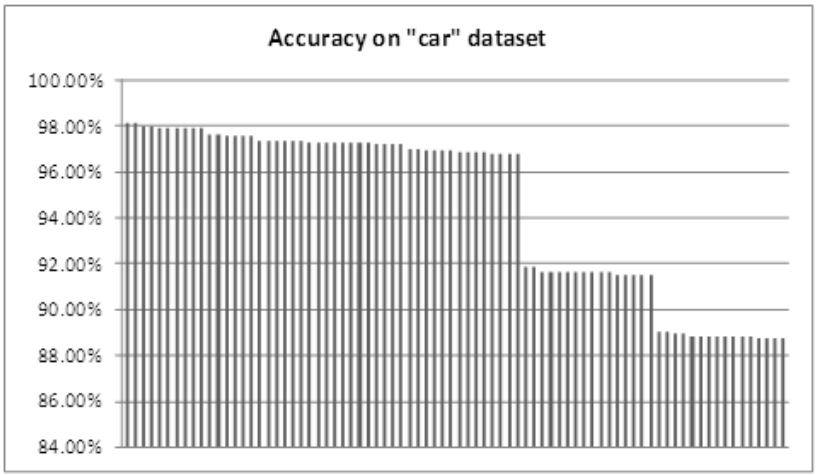


Fig. 9. Distribution of 80 algorithms' accuracies on the "car" problem

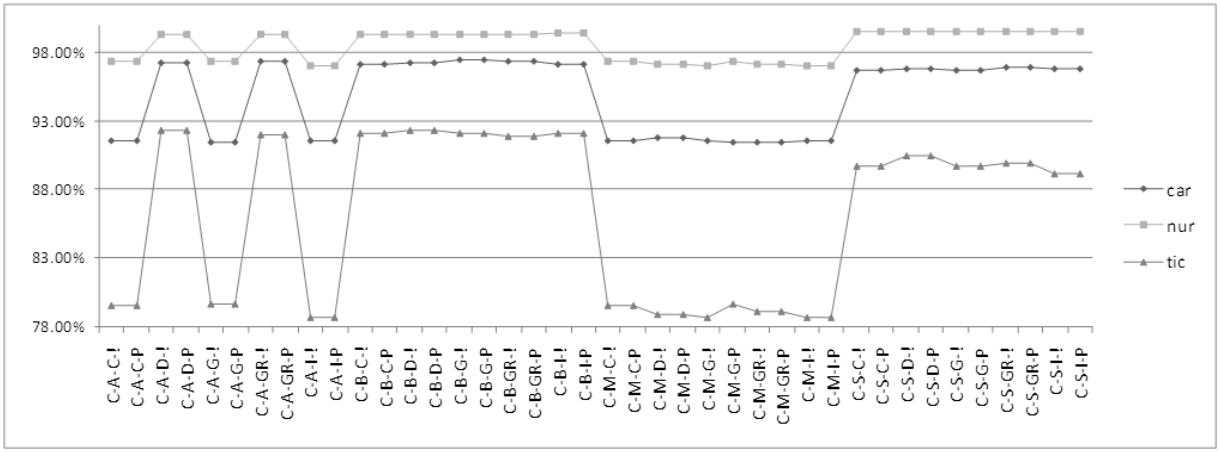


Fig. 10. Accuracy of 40 algorithms (including “chs/anf”) on “car”, “nur”, and “tic” problems

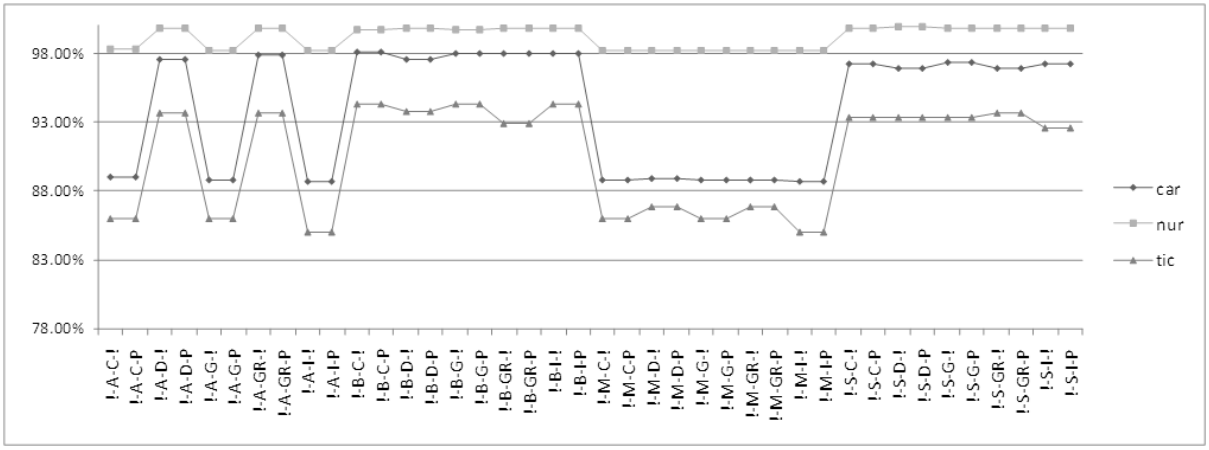


Fig. 11. Accuracy of 40 algorithms (not including “chs/anf”) on “car”, “nur”, and “tic” problems

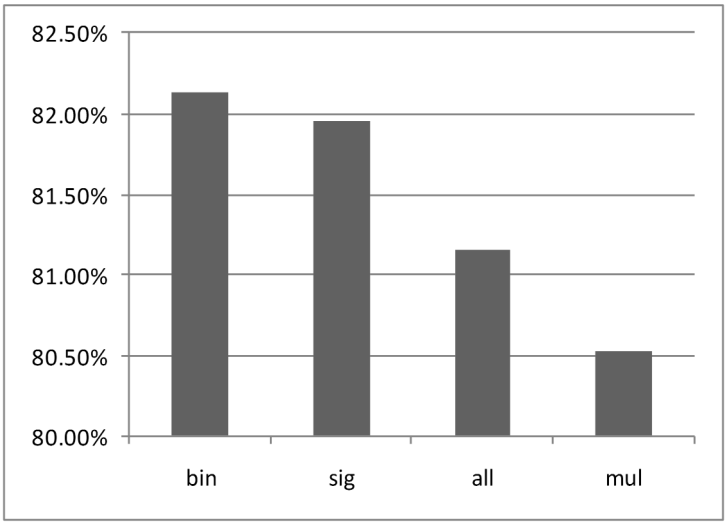


Fig. 12 (a) Average accuracy grouped by CSC RCs

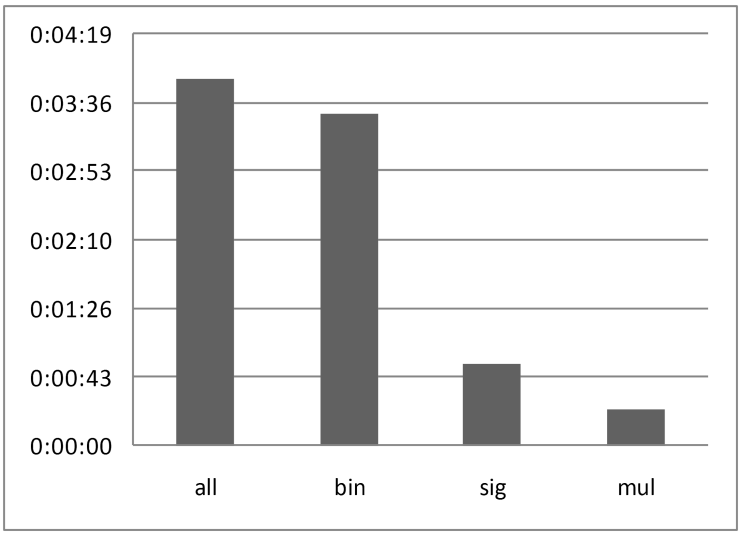


Fig. 12 (b) Average run time (sec) grouped by CSC RCs

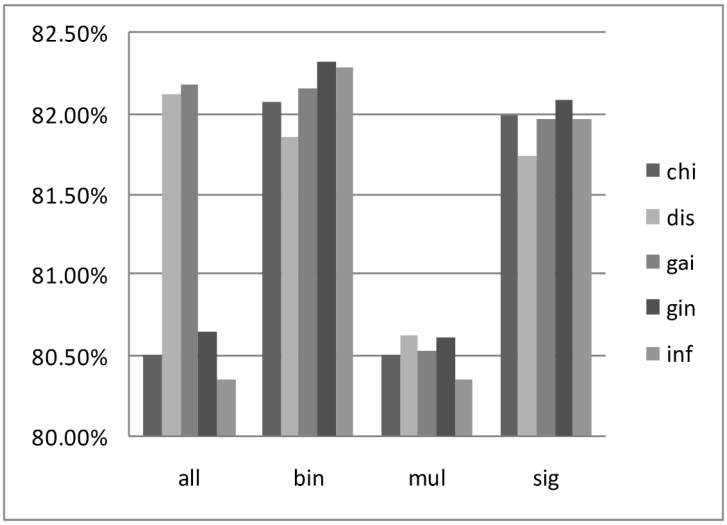


Fig. 13. Average accuracy when using different CSC and ES RCs

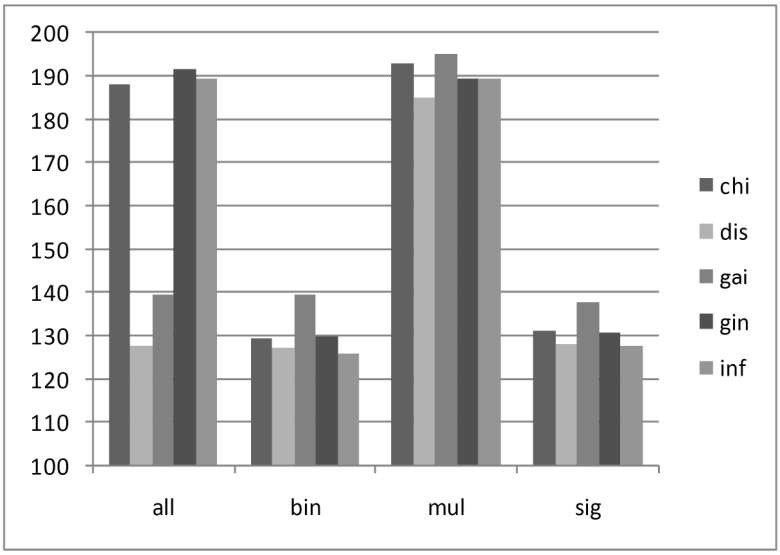


Fig. 14. Average total number of nodes of decision tree models with various CSC and ES RCs

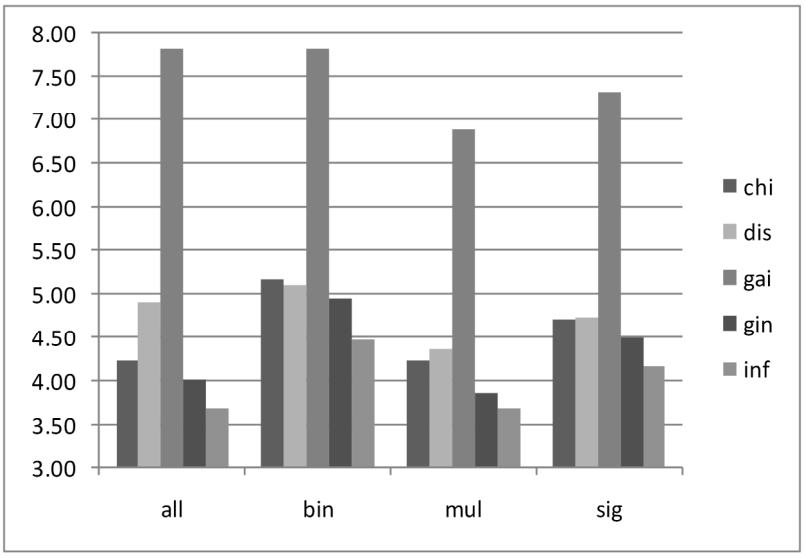


Fig. 15. Weighted average three depths when using different CSC and ES RCs

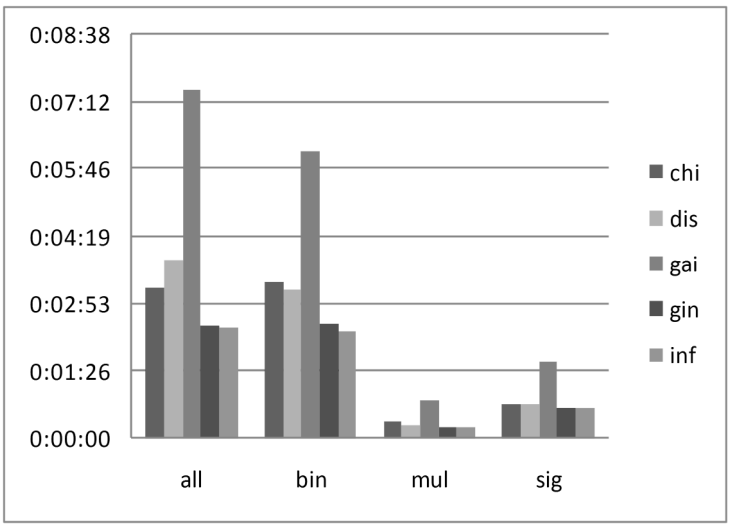


Fig. 16. Average run time when using different CSC and ES RCs