A Decision Tree Induction Algorithm for Efficient Rule Evaluation Using Shannon's Expansion

Herrera-Semenets, Vitali; Bustio-Martínez, Lázaro; Hernández-León, Raudel; van den Berg, Jan

DOI
10.1007/978-3-031-47765-2_18

Publication date
2023

Document Version
Final published version

Published in
Advances in Computational Intelligence - 22nd Mexican International Conference on Artificial Intelligence, MICAI 2023, Proceedings

A Decision Tree Induction Algorithm for Efficient Rule Evaluation Using Shannon's Expansion

Vitali Herrera-Semenets[1], Lázaro Bustio-Martínez[2(✉)], Raudel Hernández-León[1], and Jan van den Berg[3]

[1] Advanced Technologies Application Center (CENATAV), La Habana, Cuba
{vherrera,rleon}@cenatav.co.cu
[2] Department of Engineering Studies for Innovation, Iberoamerican University, Mexico City, Mexico
lazaro.bustio@ibero.mx
[3] Intelligent Systems Department, Delft University of Technology, Delft, The Netherlands
j.vandenberg@tudelft.nl

Abstract. Decision trees are one of the most popular structures for decision-making and the representation of a set of rules. However, when a rule set is represented as a decision tree, some quirks in its structure may negatively affect its performance. For example, duplicate sub-trees and rule filters, that need to be evaluated more than once, could negatively affect the efficiency. This paper presents a novel algorithm based on Shannon's expansion, which guarantees that the same rule filter is not evaluated more than once, even if repeated in other rules. This fact increases efficiency during the evaluation process using the induced decision tree. Experiments demonstrated the viability of the proposed algorithm in processing-intensive scenarios, such as in intrusion detection and data stream analysis.

Keywords: Decision Tree · Rule-Based Systems · Data Processing

1 Introduction

Rules are principles or regulations defined to guide actions or behaviors. An advantage of using rules is that it is possible to represent human knowledge naturally. This advantage could be why rule-based systems share roots with cognitive science and artificial intelligence (AI) [14]. Rule-based applications are extensively used in firewall systems [7], clinical decision support systems [10], and intrusion detection systems [6]. In these situations, immediate action is usually required after the reception of new data or, in other words, these systems require (near) real-time data processing [5].

A rule consists of two parts: the premise α and the conclusion β. The premise can be composed of some so-called *filters* and the conclusion can be represented

H. Calvo et al. (Eds.): MICAI 2023, LNAI 14391, pp. 241–252, 2024.
https://doi.org/10.1007/978-3-031-47765-2_18

by a label [8]. A single filter X_i can be represented as $f_i \oplus v$, where f_i is a conditional feature, v is a value from the domain of f_i, and $\oplus$ is a relational operator from the set of relations $\{<, \leq, =, \neq, >, \geq, \in\}$. A premise can be formed from one or more filters by joining them with logical operators. A rule like that is usually denoted as $\alpha \rightarrow \beta$, and it is applicable if its premise is satisfied.

Efficiently evaluating multiple rules on a real-time data stream could be solved using a parallel or sequential approach, replicating the data flow or not. This could be more or less efficient depending on the hardware resources available for the task. However, the rule set may contain overlapping rules [12], or simply rules that share the same filters, which means that the same filter must be evaluated several times.

Having a data structure that allows evaluating all the rules simultaneously while evaluating each filter only once would be very useful and can be applied in many common-life situations. For example, decision trees are one of the most popular data structures for decision-making and representing a set of rules. However, when a rule set is represented as a decision tree, duplicate sub-trees are usually kept in the obtained structure, which could affect spatial efficiency and evaluation time [8].

The main goal of this research paper is to propose a new decision tree induction algorithm for rules evaluation without omitting any filters. The proposed algorithm follows the theoretical principles of Shannon's expansion [13], which allows to obtain a more concise rule representation, and avoids duplicate sub-trees while the same filter is evaluated only once. This is very valuable in such cases where evaluating rules in (near) real-time is required, for example in intrusion detection and data streams processing tasks.

The remainder of this paper is structured as follows. First, the background about the motivation underlying this research, decision tree induction, and Shannon's expansion is described in Sect. 2. Second, the proposed strategy is presented in Sect. 3. Third, in Sect. 4, the experimental results using different settings are presented and discussed. Finally, the conclusions and future work are outlined in Sect. 5.

2 Background

When a decision tree is induced from a predefined rules set (which is not modified frequently), the temporary cost is not essential. However, in scenarios where data stream processing in (near) real-time is required, or where the time taken to add a new rule may have negative implications from a financial or business point of view, the temporary cost of inducing the decision tree acquires greater importance [2]. In addition, once the data used to create the decision tree changes significantly, restructuring the decision tree becomes a necessary task. However, it is difficult to manipulate or restructure decision trees. This is because a decision tree is a representation of procedural knowledge, which imposes an order of evaluation on the attributes [1]. One way to deal with this situation is to induce the decision tree again. This process could be performed on-demand without any noticeable delay if the amount of rules is small [1].

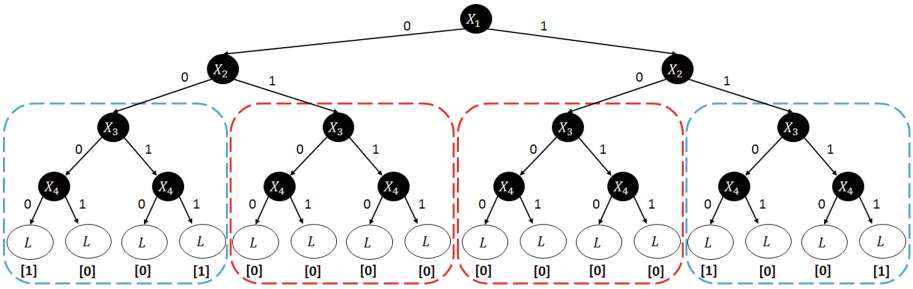


Fig. 1. Example of an induced decision tree.

There are two approaches widely addressed in the literature for decision tree induction [4]. In the most commonly used approach, the decision tree is induced from a data set [11]. In the other approach, the decision tree is induced from a rule set [1].

The primary goal of this paper differs from the latter approach. The intention here is to induce a decision tree that allows evaluating the same rules used for its induction without omitting any of the filters that make up each rule. In such a way, all rules of the rule set are applied when an instance is processed.

A decision tree is used to specify a decision procedure, where the proper sequence of filters can be evaluated [8]. The tree is traversed top-down, *i.e.*, from the root node to a leaf node, evaluating the appropriate filter at each node. Depending on the result (which can be true "1" or false "0"), the corresponding branch is selected, and the next filter to which the branch leads is evaluated. The final decision is determined after reaching a leaf node. This idea has been described in several works [8]. Figure 1 shows the induced decision tree to determine the truth-value of the following premise: $(X_1 \Leftrightarrow X_2) \wedge (X_3 \Leftrightarrow X_4)$.

As it can be seen in Fig. 1, two paths lead to two neighboring leaves labeled with 0. The paths leading to them are identical, regardless of the last edges; in fact, one is labeled with 0 and the other with 1. However, regardless of the value taken by X_4, the decision is the same $\beta = 0$. Therefore, it makes no sense to keep two separate branches leading to the same conclusion (see both sub-trees framed in red), so they can be merged and removed. This allows the final decision label to move up and the tree to be simplified. From a logical point of view, the process that allows a reduction is based on the application of backward dual resolution [8]. Such a process is applied recursively, from bottom to top, until no further reduction is possible. It is valid to highlight that only neighboring nodes can be merged and reduced, and only if they have the same decision value. Although the backward dual resolution is applied, two equal sub-trees are still present (see both sub-trees framed in blue in Fig. 1). This means that the tree obtained can be reduced even more. The literature describes a process to break down a Boolean function by Shannon's expansion [13], which can be helpful for handling the previous situation.

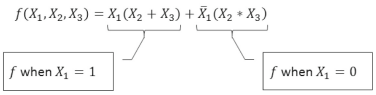


Fig. 2. Example of Shannon's expansion.

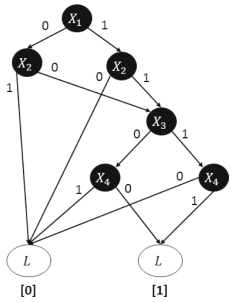


Fig. 3. Representation of the premise $(X_1 \Leftrightarrow X_2) \wedge (X_3 \Leftrightarrow X_4)$ applying the Shannon's expansion.

Shannon's expansion theorem states that every Boolean function of many variables $f(X_1, X_2, ..., X_i, X_n)$ can be decomposed as the sum of two terms, one with a particular variable X_i set to 0, and one with it set to 1. This can be seen in Eq. 1, where for $X_i = 1$ takes place $X_i f$ and for $X_i = 0$ takes place $\overline{X}_i f$.

$$f(X_1, X_2, ..., X_n) = \overline{X}_i f(X_i, ..., X_{i-1}, 0, X_{i+1}, ..., X_n) + X_i f(X_i, ..., X_{i-1}, 1, X_{i+1}, ..., X_n) \quad (1)$$

For example, given a function $f(X_1, X_2, X_3) = X_1 \times X_2 + X_1 \times X_3 + X_2 \times X_3$, its expansion in terms of X_1 is shown in Fig. 2.

If the example depicted in Fig. 2 is used for a decision tree induction, a rule could represent a Boolean function, where each variable X_j is a filter. In this sense, the operators $\times$ and $+$ are represented by the logical operators $\wedge$ and $\vee$ respectively. Therefore, once X_i is evaluated, there are two paths to follow: one path for the case where it is met $(X_i = 1)$ or the other path for when it is not met $(X_i = 0)$. Whatever the way forward, X_i is no longer evaluated. In theory, using Shannon's expansion allows inducing a decision tree where all the filters of the rules are evaluated optimally. This means that for every possible path that can be traversed from the root node to the leaf nodes, it is guaranteed that the filters are evaluated only once, even if the same filter is present several times. Also, Shannon's expansion allows each subtree to be explicitly displayed only once since the repeated occurrences are merged. As shown in Fig. 3, it is possible to obtain a more concise representation of the premise $(X_1 \Leftrightarrow X_2) \wedge (X_3 \Leftrightarrow X_4)$.

As can be seen in Fig. 3, the number of leaf nodes (highlighted with the label L) and filter evaluation nodes (highlighted with the label X_i) is considerably reduced regarding the representations obtained in Fig. 1. The following section

Algorithm 1: Shannon's expansion-based decision tree induction

```
   Input: R: Rule list
   Output: DT: Decision tree
1  DT ← New_Decision_Tree()
2  currentNode ← 0
3  foreach r in R do
4  |  tmpRule.Disjunction_List ← Create_Disjunction_List(r)
5  |  DT[currentNode].Rules.Add(tmpRule)
6  end
7  Set_Root_Node(DT[currentNode])
8  while DT.amountNodes != currentNode do
9  |  if DT[currentNode].Rules != empty then
10 |  |  result_0 ← Shannon(DT[currentNode].Rules, 0, DT[currentNode].Filter)
11 |  |  next_0 ← Search(DT, result_0, 0, currentNode)
12 |  |  DT[currentNode].next_by_zero ← next_0
13 |  |  result_1 ← Shannon(DT[currentNode].Rules, 1, DT[currentNode].Filter)
14 |  |  next_1 ← Search(DT, result_1, 1, currentNode)
15 |  |  DT[currentNode].next_by_one ← next_1
16 |  |  DT[currentNode].Rules ← empty
17 |  end
18 |  else
19 |  |  DT[currentNode].next_by_zero ← 0
20 |  |  DT[currentNode].next_by_one ← 0
21 |  end
22 |  currentNode++
23 end
24 return DT
```

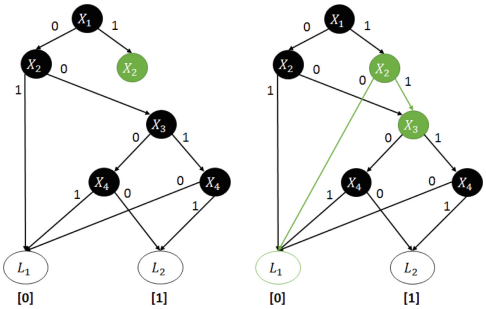
describes the decision tree induction algorithm proposed in this work based on Shannon's expansion principle.

3 Proposal

This paper proposed a novel algorithm based on Shannon's expansion theorem. The proposed algorithm receives a list of rules R, which is used to induce the decision tree DT. As shown in lines 1–2 of Algorithm 1, the first step is to initialize DT and place the *currentNode* position in the root node. Then each premise is represented as a list of disjunctions, where each disjunction will contain the filters that compose it (see lines 3–6 of Algorithm 1) [8]. For example, the premise $\alpha = (X_1 \vee X_2) \wedge X_3$ is represented by a list of two disjunctions: (1) X_1, X_2 and (2) X_3. In this context, the variable X_i is called *filter*.

Each node in DT stores a list of rules with their respective disjunctions that need to be evaluated. Also, each node has a list with the rule(s) IDs that are satisfied and those IDs that are no longer satisfied at that moment. Note that this is a different feature regarding the examples analyzed in the previous section, where the final decision is determined after reaching a leaf node. In line 7 of Algorithm 1, the position of the current node is declared as the root node. Moreover, in this step, the first filter of the first rule is selected to be evaluated at the root.

Then, the loop that recursively builds the decision tree begins (see lines 8–23 of Algorithm 1). In this loop, if there are rules to be inserted in the current node,



(a) Tree induced by edge 0 of the root node

(b) Leaf node L_1 and node X_3 meet the search criteria

Fig. 4. Decision tree induction using Shannon's expansion.

the Shannon's expansion is applied (see lines 9–17 of Algorithm 1). The lines 10–12 of Algorithm 1 are performed for the case where the filter evaluated in the current node is equal to zero. In line 10, the Shannon's expansion is carried out, and in line 11 of Algorithm 1 it is searched if there is any node that evaluates the same rules resulting from the expansion. The search result is set as the next node to scroll through zero (see line 12 of Algorithm 1). In the next three lines, the same three previous steps are executed, but this time for the case where the filter evaluated in the current node is equal to one (see lines 13–15 of Algorithm 1).

Let us further illuminate this step by giving an example using an example: the premise $(X_1 \Leftrightarrow X_2) \wedge (X_3 \Leftrightarrow X_4)$, introduced in the previous section, is used here. After having the tree induced by edge 0 of the root node, it is ready to perform its induction by edge 1 of the root node (see Fig. 4(a)). After applying the Shannon's expansion for the case in which the filter X_2 is fulfilled, a structure is obtained that stores the filters that remain to be evaluated (X_3 and X_4) and the rule (s) IDs that are met and those that are not met. Then, it is sought if there is any node in the decision tree that meets the result of the Shannon's expansion. As it can be seen in Fig. 4(b), the subtree with root node X_3 meets the search criteria; therefore, a new node is not created, but edge 1 of node X_2 is redirected towards the found node X_3. The same happens with edge 0 of X_2, which is redirected to the leaf node L_1.

In line 16 of Algorithm 1, after expanding the tree by 0 and by 1, the list of rules of the current node is emptied. If the current node is a leaf node, the next positions to move through 0 and 1 are defined as zero (see lines 18–21 of Algorithm 1). Line 22 increases the *currentNode* pointer to the next node in the list. Once all nodes are visited, a decision tree is returned (see line 24 of Algorithm 1).

Algorithm 2 describes how Shannon's expansion performs. For this, each rule is checked (see lines 2–13 of Algorithm 2); verifying whether its disjunctions are

Algorithm 2: Shannon

Input: R: Rule list, $result\_Type$: Expanded by 1 or 0, $currentFilter$: Filter evaluated on the current node
Output: $result$: Shannon's expansion result

```
1  result ← New_Result()
2  foreach r in R do
3      tmpRule ← New_Rule()
4      foreach disj in r.Disj_List do
5          tmpDisj ← New_Disjunction()
6          Searching_Filter(tmpDisj, currentFilter, disj.Filter_List, result_Type)
7          if len(tmpDisj.Filter_List > 0) then tmpRule.Disj_List.Add(tmpDisj)
8          else if result_Type == 0 then tmpRule.Disj_List ← empty, break
9      end
10     if len(tmpRule.Disj_List > 0) then result.Rules.Add(tmpRule)
11     else if result_Type == 1 then result.rules_in_one.Add(r.ID)
12     else result.rules_in_zero.Add(r.ID)
13 end
14 return result
```

checked (see lines 4–9 of Algorithm 2); and verifying for each disjunction whether the filters are checked (see line 6 of Algorithm 2).

If the expansion is performed for the scenario in which the current node's filter is satisfied ($result\_Type = 1$), and if any of the filters within a disjunction matches the current node's filter ($currentFilter$), the disjunction is replaced by a new one devoid of filters, and the execution exits the Searching_Filter function. This occurs because, with a single filter match, the entire disjunction is satisfied. In cases where the filters are not identical or the current node's filter is not met ($result\_Type = 0$), the examined filter is added to the temporary disjunction ($tmpDisj$).

After checking all filters, if at least one filter was added to $tmpDisj$, then $tmpDisj$ is added to the list of disjunctions $tmpRule.Disj\_List$ that holds the temporary rule $tmpRule$ (see line 7 of Algorithm 2). If no filter was added to $tmpDisj$ and it is expanded for the case in which the current node filter is not satisfied, the list $tmpRule.Disj\_List$ is emptied, and the disjunctions are no longer checked (see line 8 of Algorithm 2). In line 10 of Algorithm 2, if at least one disjunction was added to $tmpRule.Disj\_List$, then, $tmpRule$ is added to the list of rules $result.Rules$ of the expansion result. If no disjunction was added to $tmpRule.Disj\_List$ and it is expanded for the case in which the current node filter is satisfied, then, the identifier of the checked rule is added to the list of rules $result.rules\_in\_one$ that are satisfied (see line 11 of Algorithm 2). If no disjunction was added, but it is expanded for the case in which the current node filter is not met, then the identifier of the checked rule is added to the list of rules $result.rules\_in\_zero$ that are not satisfied (see line 12 of Algorithm 2). In line 14, after checking all the rules, the result of the Shannon's expansion containing the rules to be evaluated, the identifiers of the rules that were satisfied and those that were not, is returned.

Algorithm 3 describes how to search for a node that meets the result of the Shannon's expansion. The search is performed in a range from the position of the

Algorithm 3: Search

Input: DT: Decision tree, $result$: Shannon's expansion result, $result\_Type$: Expanded by 1 or 0, $currentNode$: Current node position
Output: N: Node

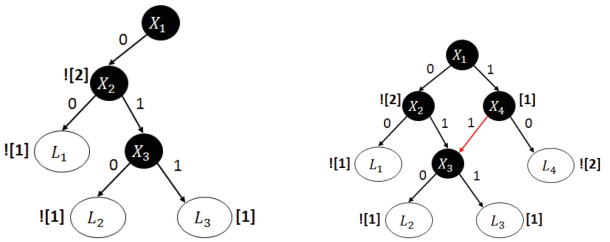
```
1  N ← 0
2  foreach N in range(currentNode, DT.amountNodes) do
3      if result.Rules == DT[N].Rules then
4          if result_Type == 1 then
5              if result.rules_in_one == DT[N].rules_in_one then return N
6          end
7          else if result.rules_in_zero == DT[N].rules_in_zero then return N
8      end
9  end
10 DT.Append(New_Node())
11 filter ← result.Rules[0].Disjunction_List[0].GetFilter(0)
12 DT[DT.amountNodes].Filter ← filter
13 DT[DT.amountNodes].Rules ← result.Rules
14 if result_Type == 1 then DT[DT.amountNodes].rules_in_one ← result.rules_in_one
15 else DT[DT.amountNodes].rules_in_zero ← result.rules_in_zero
16 DT.amountNodes++
17 return N + 1
```

current node to the last node in the list. This decision avoids finding a previously created node that could create a contradiction during the decision tree induction.

For example, Fig. 5(a) shows the decision tree induced by the zero edge of the root node using two rules: (1) $X_1 \vee (X_2 \wedge X_3) \rightarrow 1$ and (2) $X_1 \wedge X_4 \wedge X_3 \rightarrow 2$. The six nodes depicted in Fig. 5(a) occupy the first six positions in the list of nodes that make up the decision tree (see Fig 5(c)). When the edge one of the root begins to be built, a node is generated with the filter X_4, in which the rule 1 is already satisfied, therefore after this node, it would only be necessary to evaluate the filter X_3 of rule 2 (see Fig. 5(b)). If the search for X_3 is performed from the beginning of the list of nodes, in position 3, there is a node that fulfills the search (see Fig. 5(c)). Therefore, an edge from X_4 to X_3 is created, which generates a contradiction in the decision tree, since the rule 1 is satisfied in the node that evaluates the filter X_4, and if it continues along the created edge, it can reach the leaf node L_2 where rule 1 is not satisfied.

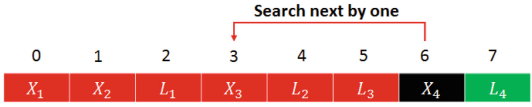
To avoid the contradictions described in Fig. 5(b), the search is carried out from the current position of the node forward (see lines 2–9 of Algorithm 3). If the rules of the Shannon's expansion result are equal to those of the node being compared, then it is checked if the node contains the same rules identifiers as the Shannon result, taking into account whether the expansion was performed for the case in which the filter is met or not (see lines 3–8 of Algorithm 3). If a node that satisfies these conditions is found, it is returned (see lines 5 and 7 of Algorithm 3).

If no node was found in the search process described before, then a new node is created, and the filter to be evaluated is assigned (see lines 10–12 of Algorithm 3). In line 13, the rules that remain to be evaluated from Shannon's expansion result are added to the new node. If the rules were expanded by 1 (this is the case in which the node filter is satisfied), the rules' identifiers that



(a) Building the zero edge of the root node

(b) Contradiction when creating an edge between X_4 and X_3



(c) Position of each node in the list of nodes that make up the decision tree.

Fig. 5. Example of contradiction in a decision tree.

were satisfied in the Shannon's expansion result are added to the new node (see line 14 of Algorithm 3). If the rules were expanded by 0 (this is the case in which the node filter is not satisfied), the rules' identifiers that were not satisfied in the Shannon's expansion result are added to the new node (see line 15 of Algorithm 3). Then, the number of nodes in the decision tree is increased, and the position of the new node is returned (see lines 16 and 17 of Algorithm 3).

4 Experimental Results

The experimental results are focused on evaluating the spatial and time efficiency of the proposed algorithm, given by the number of resulting nodes in the decision tree and the estimated time to build it respectively. Considering the example presented in Sect. 2, where it is shown that the use of the Shannon expansion allows the construction of more compact decision trees than the double-backward resolution method, it is evident that the proposed algorithm offers greater efficiency in this regard. Taking this into account, it would not be fair to make comparisons with the double-backward resolution method, which is why it was not included in the experiments.

The experiments were performed on a PC equipped with an Intel Quad-Core at 2.5 GHz CPU and 4 GB of RAM. The performance of the proposed algorithm was evaluated using 25 different rules.

As can be seen in Table 1, while the number of rules and filters increases, the spatial and time efficiency of the algorithm is negatively affected. Notice that every possible path from the root node to any leaf node must evaluate all the filters of the processed rules. Therefore, if the number of rules and filters grows,

Table 1. Efficiency achieved by the proposed algorithm using different amounts of rules.

Number of rules	Number of filters	Number of nodes	Time (s)
5	10	21	0.001
10	29	597	0.01
15	39	1 562	2
20	58	17 892	143
25	68	33 225	370

then also grows the number of paths needed for evaluating all the rules. This directly affects the number of nodes that are generated and the time needed for inducing the decision tree. However, this process can be conducted offline, so once it is finished, the decision tree can be replaced.

The following experiment consists of evaluating the advantage offered, in terms of time efficiency, by using the decision tree induced by the proposed algorithm over the rules set used to induce it. In this sense, we proceed to identify a data set composed by a high number of instances that allows us to highlight the differences in processing times.

The KDD'99 data set was used for this experiment [3]. Such data set provides connection records (instances) generated by a simulation of a military network. The training collection contains 4 898 431 instances and was used to evaluate the rule sets and the induced decision trees.

As it can be seen in Table 2, five sets of rules were created for this experiment. Note in Table 2 that some rules share equal filters, and the more rules there are in the set, the greater the number of equal filters. This fact can affect the performance when the rules are evaluated individually since the same filters must be processed more than once.

The results achieved show that the induced decision tree turns out to be more efficient concerning the processing time expressed in Table 2 than the rules set. Therefore, there is a fundamental advantage of representing rules set as a decision tree using the proposed algorithm. When a rules set is evaluated, if there are equal filters in different rules, they are evaluated several times (for each rule). In contrast, when any path is traversed from the root to a leaf node of the decision tree, all the rules are evaluated without repeating a single filter, which makes the evaluation process more efficient. This fact is very appreciated in situations where processing data in (near) real-time is required.

The improvement achieved in terms of time efficiency, at first glance, it may seem insignificant. However, in some scenarios (such as intrusion detection or data stream evaluation) the volume of information generated in one second can be considerably large. For example, a 1 Gb/s Ethernet interface can deliver anywhere between 81 274 and 1 488 096 packets/s. In perspective, a 10 Gb/s Ethernet interface can deliver 10 times more packets, that is, between 812 740 and 14 880 960 packets/s [9]. Therefore, the improvement in efficiency offered

Table 2. Time taken to process the KDD'99 training collection with different rule sets and decision trees.

Number of rules	Number of equal filters	Evaluation strategy	Time (s)
5	1	Decision tree	72
		Rules set	73
10	6	Decision tree	108
		Rules set	115
15	9	Decision tree	129
		Rules set	143
20	15	Decision tree	148
		Rules set	166
25	21	Decision tree	163
		Rules set	189

by the induced decision tree could be fundamental in scenarios that require data processing in (near) real-time.

For processing high-speed data streams, a parallel or distributed approach to evaluating the decision tree could be considered. Note that even if the decision tree is parallelized, it would still be more efficient than parallelizing the rules set, since for each process there would be a decision tree that evaluates all the rules optimally. While when parallelizing the rules set, each process would have a rule, therefore to classify an instance it would have to iterate over each process until all the rules were evaluated.

It is valid to notice that no experiments were conducted to estimate the classification accuracy since the decision tree evaluates exactly the same rules used to induce it. Therefore, the results in terms of classification accuracy are the same using the decision tree or the rules set.

5 Conclusions

The algorithm proposed in this research allows inducing a decision tree from rules set where all the filters of the rules are evaluated optimally. In other words, the induced decision tree allows that by any path traveled from the root node to some leaf node, each rule filter is evaluated only once, even if it is repeated in more than one rule. This aspect allows for improving the efficiency of the evaluation process, which makes it feasible to use it in scenarios that require data stream processing in real-time or very close to it.

The experiments conducted show that when the number of nodes and filters increases, the spatial and time efficiency of the decision tree building process is negatively affected. As future work, it is intended to address this issue with an incremental proposal. Another aspect to address in future work is the use

of multiple decision trees to boost up the speed of the evaluation process by applying parallel or distributed programming paradigms.

Acknowledgement. This research was supported by the Universidad Iberoamericana (Ibero) and the Institute of Applied Research and Technology (InIAT) by the project "Detection of phishing attacks in electronic messages using Artificial Intelligence techniques."

References

1. Abdelhalim, A., Traore, I., Nakkabi, Y.: Creating decision trees from rules using rbdt-1. Comput. Intell. **32**(2), 216–239 (2016)
2. Ahmim, A., Maglaras, L., Ferrag, M.A., Derdour, M., Janicke, H.: A novel hierarchical intrusion detection system based on decision tree and rules-based models. In: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 228–233. IEEE (2019)
3. Al-Daweri, M.S., et al.: An analysis of the kdd99 and unsw-nb15 datasets for the intrusion detection system. Symmetry **12**(10), 1666 (2020)
4. Charbuty, B., Abdulazeez, A.: Classification based on decision tree algorithm for machine learning. J. Appl. Sci. Technol. Trends **2**(01), 20–28 (2021)
5. Herrera-Semenets, V., Pérez-García, O.A., Gago-Alonso, A., Hernández-León, R.: Classification rule-based models for malicious activity detection. Intell. Data Anal. **21**(5), 1141–1154 (2017)
6. Herrera-Semenets, V., Pérez-García, O.A., Hernández-León, R., van den Berg, J., Doerr, C.: A data reduction strategy and its application on scan and backscatter detection using rule-based classifiers. Exp. Syst. Appl. **95**, 272–279 (2018)
7. Jaïdi, F.: A novel concept of firewall-filtering service based on rules trust-risk assessment. In: Madureira, A.M., Abraham, A., Gandhi, N., Silva, C., Antunes, M. (eds.) Proceedings of the Tenth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2018), pp. 298–307. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-17065-3_30
8. Ligêza, A.: Logical Foundations for Rule-Based Systems. Springer, Heidelberg (2006)
9. Schudel, G.: Bandwidth, packets per second, and other network performance metrics. Abgerufen am **10**, 2010 (2010)
10. Soufi, M.D., Samad-Soltani, T., Vahdati, S.S., Rezaei-Hachesu, P.: Decision support system for triage management: a hybrid approach using rule-based reasoning and fuzzy logic. Int. J. Med. Informatics **114**, 35–44 (2018)
11. Yates, D., Islam, M.Z., Gao, J.: SPAARC: a fast decision tree algorithm. In: Islam, R., et al. (eds.) AusDM 2018. CCIS, vol. 996, pp. 43–55. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-6661-1_4
12. Zhang, G., Gionis, A.: Diverse rule sets. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD20), pp. 1532–1541. Association for Computing Machinery, New York (2020)
13. Zhang, J., Yang, G., Hung, W.N., Zhang, Y., Wu, J.: An efficient NPN Boolean matching algorithm based on structural signature and Shannon expansion. Clust. Comput. **22**(3), 7491–7506 (2019)
14. Zhao, J., Wu, M., Zhou, L., Wang, X., Jia, J.: Cognitive psychology-based artificial intelligence review. Front. Neurosci. **16**, 1024316 (2022)