Protection Measurements for Distributed Decision Trees

Kevin Wallis[1], Fabian Schillinger[2], Christoph Reich[1], Christian Schindelhauer[2]
University of Applied Sciences Furtwangen[1]
University of Freiburg[2]
Germany

Abstract

In Industry 4.0 machine learning approaches are a state-of-the-art for predictive maintenance, machine condition monitoring, and others. Distributed decision trees are one of the learning algorithms for such applications. A new approach of node based parallelization for the construction is presented and allows to classify data through a network of nodes. Attacks on the nodes are discussed based on different attack scenarios and attack classifications are presented. A thorough analysis of protection measurements is given, such that classification is not maliciously modified by an attacker. Different countermeasures are proposed and analyzed. A quorum-based system allows for a good balance between computational overhead and robustness of the algorithm.

Keywords-distributed decision tree, distributed data valida-tion network, node-based parallelization, industrial-security

1. Introduction

Predictive maintenance, job scheduling optimization, or machine condition monitoring are typical applications in Industry 4.0. Machine learning algorithms and techniques are often used to optimized these applications. The algorithms span a large range from Neural Networks, over K-Nearest Neighbors, to decision trees. For data-driven approaches high quality of the data is absolutely necessary to provide meaningful predictions and classifications and later, training of Machine Learning models. Quality of the data is highly influenced by its correctness, as wrong data produces other data, than for example missing data. The correctness can be verified during the preprocessing phase of the workflow. A Distributed Data Validation Network [1] can be used to provide high data quality. The data is evaluated, whether it is in the norm or deviates from it. In other approaches, the algorithm for the data validation procedure is assumed to be trustworthy and does not produce errors, neither is assumed, that the procedure is attacked by third parties. In distributed systems, however, attacks and faults occur. Nodes can fail and stop producing output or they may produce faulty output. This work focuses in distributed decision trees as a main technique for Industry 4.0 Machine Learning classification with a strong perspective on possible faults and attacks on the procedure. Further, various protection measures are provided and discussed in detail. It is found, that different nodes in a distributed decision tree, or other distributed classification algorithms, have different impacts on the outcome. The outcome is affected by the procedure in general and the accuracy of the predicion in detail. The impact can be reduced by applying the right protecting measures, as discussed later.

Our Contribution

A preliminary version of this paper was presented at the World Congress on Internet Security (WorldCIS-2020) [2]. Our contribution consists of a different approach to distribute the decision-making process of a decision tree on a group of nodes in a network. The approach, called node-based parallelization, in contrast to existing approaches spans the decision tree across the nodes in a network, such that every node has a single classification r ule. I n o ur w ork, nodes in a distributed system are seen as possibly faulty or even attacked. Based on this observation, the different impacts of nodes on the decision procedure and outcome are discussed. Further, various protection measures are presented to overcome the influence of attacked or faulty nodes. It is analyzed how this impact can be reduced with the correct protection measures. Both the advantages and the disadvantages are considered and listed.

Organization of the Paper

The paper is structured as follows: In Section II related work is presented. In Section III the backgrounds on decision trees, followed by the backgrounds on distributed decision trees, in Section III-B, and the approach of node-based parallelization are described. The considered attack model is provided in Section IV. Further, Section V describes approaches to mitigate the chances of successful attacks. These approaches are compared, and a detailed analysis of quorum-based decisions is given. Finally, Section VI concludes the work and gives an outlook on future research.

2. Related Work

Decision trees are a well-known and well-researched area of science. Breiman et al. [3] describe essential details like terminology, structure, generation, etc. of decision trees. These results are used as the foundation for our research.

A wide variety of research questions exists concerning decision trees. These include a) unbalanced [4] or noisy input parameters [5] b) the use of the correct tree structure [6][7] c) balancing the tree, so that the distribution of the nodes

is as optimal as possible and thus the minimum number of steps is taken in every decision making process d) optimal decision conditions at the decision nodes [8] and e) runtimes for traversing the tree.

Furthermore, Desai et al. [9][10] deal with the distribution of decision trees. However, we have to distinguish that subtrees are distributed and not single nodes as in our approach. Additionally, the influence of corrupted nodes on the integrity of the decision result was not considered in these works.

In comparison to previous research questions, we deal with the security of a distributed decision tree, where each node is a machine in a network. We investigate how corrupted nodes affect the decision making of the distributed decision tree.

"Quorum systems are the key mathematical abstraction for ensuring consistency in fault-tolerant and highly available distributed computing."[11] Each quorum system is the intersection of two or more quorums [12]. In contrast, one of our protection measurements replaces single nodes with quorums and thereby significantly improving the integrity of the distributed decision tree.

3. Decision Tree and Distributed Decision Tree

Machine learning with decision tree methods is a com-mon way of regression or classification. Classification with decision trees uses supervised learning methods. This paper focuses on classification use cases. In Fig. 1 a decision tree with three decision nodes t_1, t_2 and t_3 and four leaf nodes t_4, t_5, t_6 and t_7 is shown. Decision nodes select, which decision path is taken, and leaf nodes represent the selected class at the end of a decision path. The union $N = D \cup L$ of the decision nodes $D = \{t_1, t_2, t_3\}$ and all leave nodes $L = \{t_4, ..., t_7\}$ is the set of all nodes of the decision tree.

In the following the decision tree definition from Breiman et al. [3] is used. The measurement vector $\boldsymbol{x}$, which consists of several features $(x_1, x_2, ..., x_n)$ is assigned to a class j by the decision tree. The set of all possible input vectors is defined as $X = \bigcup_i \boldsymbol{x_i}$ and the set of all possible classes is defined as $C = \{1, 2, ..., J\}$. Furthermore, the classification of $\boldsymbol{x}$ is done by a classification rule defined as $d(\boldsymbol{x}) = j$.

A. Decision Tree Creation

To generate a decision tree, a prepared data set, called the training data set, is used. The learning data set $\mathscr{L} = \{(\boldsymbol{x_1}, j_1), ..., (\boldsymbol{x_N}, j_N)\}$ consists of a set of tuples, where each tuple corresponds to a measurement vector $\boldsymbol{x}$ and the associated class j. Based on the relationship between the total measurement vectors and their respective classes, the best possible conditions for decision nodes are determined. Different methods are used for the calculation, such as the consideration of the entropy or the gini index [13][14][15].

The creation of decision trees is based on Tan et al. [16]. They defined four methods: `create_node`, `find_best_split`, `classify`, and `stopping_cond`. Furthermore, they defined a skeleton decision tree induction algorithm (see Listing 1), which uses the four methods. The creation of a leaf or decision node is done by `create_node`. When the `stopping_cond` is met the created node will be a leaf node and needs therefore be classified by `classify` otherwise another splitting condition needs to be found by `find_best_split`.

```
def tree_growth(training_data,attributes):
    if stopping_cond(training_data,attributes):
        # Create leaf node with the
        # corresponding label
        leaf = create_node()
        leaf.label = classify(training_data)
        return leaf
    else:
        # Create decision node with the
        # condition statement
        root = create_node()
        root.test_cond = find_best_split(
            training_data,attributes)

        # Iterate all labels and split
        # the training_data
        for label in possible_labels():

            # Combine label training_data
            training_data_label = []
            for data in training_data:
                if root.test_cond(data) == label:
                    training_data_label.push(data)

            # Recursive call with the
            # training_data_label subset
            child = tree_growth(
                training_data_label,attributes)
            # Append child - specific label
            root.childs[label] = child

        return root
```

Listing 1: Skeleton Decision Tree Induction Algorithm

B. Distributed Decision Tree

Distributed decision trees are algorithms that reduce the run-times for classification or regression of a large data set $\mathcal{X} \subseteq X$, by using parallelization. How parallelization is applied can be grouped into four approaches: horizontal, vertical, task-based, or node-based parallelization, as summarized in [17]. In our approach, a tree is created, using the available nodes in the network $n_i \in N$ for the classification. Comparable to the task-based approach a node n_i in the network serves as a node t_i in the distributed decision tree. I.e. $|N| = |T|$, and $n_i = t_i$, for all $n_i \in N$ and $t_i \in T$. The main difference is, that not a subtree is constructed on t_i, but only a single split of the data is performed. A global model of the decision tree exists, such that for each node t_i the feature x_j of the measurement vector $\boldsymbol{x}$ and the

classification rule $d(x_j)$ is defined. After applying $d(x_j)$ at a node t_i, the measurement data is forwarded to the next nodes, according to the decision and the global model.

4. Classification of an Attack

For the classification of an attack two properties of the attack are taken into account. These are the knowledge of an attack and the vulnerable areas. The classification of an attack j is based on the combination of attack knowledge k with vulnerable areas a:

$$c_j = (k, a) \tag{1}$$

A. Faulty and Malicious Nodes

When considering distributed decision trees, where each node is located on a different machine. A machine can be faulty or behave maliciously. This results in misbehaving nodes. The correct classification of data depends on the decision nodes. Faults can lead to misclassifications. Therefore, approaches to mitigate such faulty behavior have to be taken. In distributed systems failures are often distinguished as crash, omission, and byzantine failures. The implications for distributed decision trees are the following:

- Crash Failure: nodes stop working. This results in data, which may not be classified. When considering a stream of data, at some point no classification will be made, whereas all prior data might be classified correctly.
- Omission Failure: nodes or communication channels stop temporarily. Comparable to crash failures, some data might not be classified.
- Byzantine Failure: nodes behave arbitrarily. This means that data can be classified correctly because the faulty node chooses the right classification or branch. In other cases, the same data can be classified incorrectly or might even stay unclassified.
- Attack: machines can be attacked and therefore an intruder can control all classifications. An attacked machine can behave comparable to the previous failures. It may stop classification, pause the classification, or misclassify some data.

B. Classification of Attack Knowledge

An attack on a decision tree uses available information about the decision tree. The information can either be publicly available, obtained by information theft such as espionage, or collected otherwise. The more information is available to an attacker, the greater its impact on the decision tree can be. Depending on the amount and type of information, the knowledge of an attack is divided into four categories:

- *Oblivious:* No public or secret information is available for the attack.
- *Decision Tree Structure:* The decision tree structure, with the individual decision and leaf nodes, is accessible to the attack.
- *Data:* The incoming data to be classified is known. A distinction is made between training data, where the class is also given, and real data.
- *All:* Combines the two sets of information from Model and Data. Thus the complete model and the incoming data are known.

C. Classification of Vulnerable Areas

A decision tree can be divided into different areas, whereby the areas can overlap. Depending on which area an attack has access to, its impact on the decision tree changes. Possible reasons why an attack only has access to one area are a) the nodes are distributed in the network and an attacker only has access to a certain part of the network b) the nodes are installed on different operating systems and the attack can only be carried out on a certain operating system and c) some of the nodes are protected with additional protection, like a stronger firewall, which makes them prone to the attack. In the following different attack areas are listed and shown in Fig. 1.

- *None:* The decision tree is well protected so that it cannot be attacked.
- *Single:* A single node (e.g. t_1) can be attacked.
- *Level:* An entire decision tree level (e.g. t_2 and t_3) has weak points and can be attacked. When multiple levels can be attacked they are categorized as *Subset*.
- *Subtree:* A complete subtree (e.g. t_2, t_4 and t_5) is vulnerable. This can be a parent node and its two child nodes, that is a subtree over two levels or a parent node with child nodes over several levels.
- *Subset:* A subset of nodes has weak points and is vulnerable. The subset can consist of subsequent nodes or nodes that are not related to each other. Thus any possible area of attack (*None*, *Single*, *Level*, *Subtree*, or *All*) can be represented.
- *All:* The attack uses a vulnerability that affects every node.

D. Corruption Factor of an Attack

Let M be a multiset containing all measurements $\boldsymbol{x_i}$ classified by a decision tree. The number of occurrences of an element $\boldsymbol{x_i}$ is called $m(\boldsymbol{x_i})$ where $m(\boldsymbol{x_i}) \in \mathbb{N}_0^+$. Furthermore, the number of all elements in M is defined as:

$$n = |M| = \sum_{x \in M} m(\boldsymbol{x}) \tag{2}$$

Of the given M, all n elements pass the root node, and depending on the initial split the successive nodes are only passed by a subset n_t of the classified measurements.

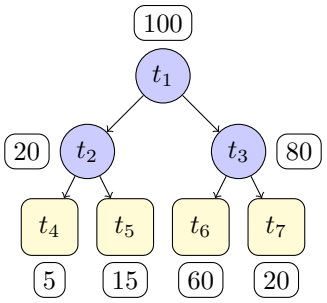


Figure 1: Example of decision tree node occurrences

In Fig. 1 a CART binary decision tree is assumed with $n = 100$ input measurements. From the initially 100 input measurement at the root decision node t_1, decision node t_2 is only passed by a subset of $n_{t_2} = 20$ and t_3 by $n_{t_3} = 80$ measurements. The ratio of passes of node t, also called the corruption factor of an attack on a single node t, is defined as:

$$\eta(t) = \frac{n_t}{n} \tag{3}$$

Therefore, the root node t_1 has a corruption factor of 1.0, t_2 has 0.2 and t_3 has 0.8. A maximum of one node per tree level can be passed per $\boldsymbol{x}$ and the pass ratio per node is $\eta(t) \geq 0$. A decision path $\boldsymbol{p}$ is a tuple of nodes $(t_1, ..., t_n)$ where t_1 is the root node and t_n is a leaf node. All possible paths of the decision tree are called $P = \{\boldsymbol{p_1}, ..., \boldsymbol{p_m}\}$. Let T be a set of parent nodes $\{t_1, ..., t_m\}$ that are attacked. Subsequent nodes are not part of T, as they do not influence the corruption factor.

The total corruption factor $\eta(T)$ on the decision tree of a given set of nodes T is calculated from the sum of the individual nodes $t \in T$.

$$\eta(T) = \sum_{t \in T} \eta(t) \tag{4}$$

Furthermore, the sum of all leaf node corruption factors is:

$$\eta(L) = \sum_{t \in L} \eta(t) = 1 \tag{5}$$

The security objective of decision tree integrity indicates how much impact an attack has on the tree. Thus, the integrity ι of the decision tree is defined as:

$$\iota(T) = 1 - \eta(T) \tag{6}$$

Besides a binary decision node, the extension of a decision node with a set of further decision nodes is also considered (see Fig. 2c). A decision node $t_7 = q_7^1$ is extended by three additional decision nodes q_7^2, q_7^3 and q_7^4. $Q = \{q_7^1, q_7^2, q_7^3, q_7^4\}$ is called a quorum.

The corruption factor of an attack on a quorum i of decision nodes $Q = \{q_i^1, q_i^2, q_i^3, ...\}$ is the same among all nodes. Thus the following definition applies:

$$\eta(t) = \eta(Q) = \eta(q), \quad \forall\, q \in Q \tag{7}$$

E. Relationship between Corruption Factor and Success Probability of an Attack

The corruption factor, i.e. the ratio of corrupted paths to normal paths in the decision tree, indicates the success probability γ of an attack on a certain node t. Assuming node t is corrupted after the attack, the following applies:

$$\gamma(t) = \eta(t) \tag{8}$$

When a set of parent nodes $T = \{t_1, ..., t_m\}$ is corrupted by an attack the attack success probability is equal to the corruption factor:

$$\gamma(T) = \sum_{t \in T} \gamma(t) = \eta(T), \quad T \neq Q \tag{9}$$

By replacing a node t by a quorum Q, the probability of success depends on the number of corrupted nodes C in the quorum:

$$\gamma(t) = \gamma(Q) = \begin{cases} \eta(t), & |C| \geq \lfloor \frac{|Q|}{2} \rfloor + 1 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Lemma 4.1: If a parent node set T contains corrupted nodes $C \subseteq T$ the attack success probability will decrease when a single corrupted node c is replaced by a quorum Q with more non-corrupted nodes than corrupted ones.

Proof: Let $\eta(c) > 0$. Then the attack success probability without a quorum replacement is $\gamma(T) = \sum_{t \in T} \gamma(t)$. Replacing $c \in T$ by a quorum Q with $|C| < \lfloor \frac{|Q|}{2} \rfloor + 1$ leads to $\gamma(c) = 0$ and therefore $\gamma(T) > \sum_{t \in T \setminus \{c\}} \gamma(t) + 0$. ■

5. Protection Measures and Analysis

In the following, we consider attacks on nodes, because the behavior of attacked nodes may resemble that of nodes with failures. Attacked nodes can be inner nodes in the decision tree, then incorrect classification c an h appen, because the data is propagated to the wrong branch. When leaf nodes are attacked the concrete final r esult o f the classification i s m odified by th e no de. Th e no de decides the final c lassification an d th erefore, ca n ch oose th e wrong label. To mitigate the effects of attacked nodes different approaches can be used. To compare these approaches the success probabilities of attacks are analyzed. First, an analysis is performed without any protection measures. Second, different protective mechanisms are shown and supported by theorems and proofs. In the end, a detailed analysis of the quorum protection mechanism is given.

For the analysis of the different protections measurements python scripts [1] are used. The generic algorithm for the evaluation is shown in Listing 2. As input parameters a `combinations_generator`, a `path_generator`, a `corruption_factor_calculator` and a `corruption_validator` with their necessary parameters are given. The analysis algorithm iterates every possible tree combination, where a tree combination is a decision tree with a defined number of corrupted nodes. Furthermore, for every corruption combination all possible paths through the tree are analysed and the corrupted parent nodes for each combination are accunmulated.

```
def analyse_corruption_factors(
    combinations_generator, combinations_params,
    paths_generator, path_params,
    corruption_factor_calculator,
    corruption_factor_calculator_params,
    corruption_validator):

corruption_factors = {}
paths = list(paths_generator(**
    path_generator_params))

# Here is the analysis algorithm
for tree in combinations_generator(**
    combinations_generator_params):
  corrupted_parent_nodes = set()
  for path in paths:
    for node in path:
      if corruption_validator(tree[node]):
        corrupted_parent_nodes.add(node)
        break

  # Calculate corruption factor
  corruption_factor = {
      corruption_factor_calculator(
      corrupted_parent_nodes, **
      corruption_factor_calculator_params): 1}

  # Combine corruption factors
  corruption_factors = dict(collections.Counter(
      corruption_factors) + collections.Counter(
      corruption_factor))

  yield corruption_factors
```

Listing 2: Corruption factor analysis algorithm

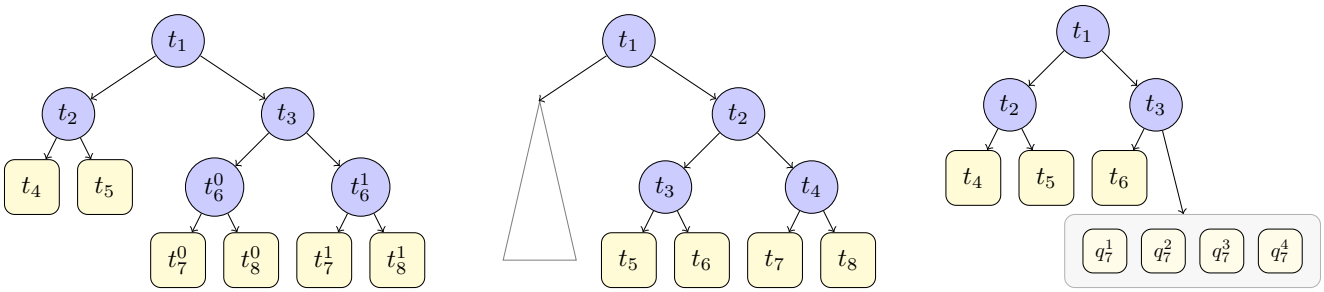
A. Attacker Success without Protection Measures

In a distributed decision tree, the impact of an attack depends on the corrupted node. The more decisions are made on a node, the higher the corruption factor of that node. In the following, four different balanced binary decision trees are analyzed and four different balanced binary trees are simulated because of their size. The decision trees differ in the number of total nodes in the tree $|N| \in \{3, 7, 15, 31\}$ for the analyzed trees and $|N| \in \{63, 127, 255, 511\}$ for the simulated trees. For the analysis, a certain number of corrupted nodes c is set and all corruption combinations are considered. The simulation corrupts the nodes randomly and calculates the corruption factor accordingly. If the number of simulation runs increases, the simulation result will be more accurate. Currently, 100,000 runs are performed per tree topology. This results are the average success probability of an attack $\gamma_{average}$, respectively the average integrity $\iota_{average} = 1 - \gamma_{average}$ of a distributed decision tree. Fig. 3a shows the analysis results of the different decision trees. It shows that with only a few corruption nodes $c < \frac{|N|}{2}$ already a low decision tree integrity $\iota_{average} < 0.5$ exists.

B. Splitting Nodes with High Impact

Some nodes in the tree have a higher corruption factor than other nodes. The influence of an attacker can be reduced if such nodes are split into multiple sub-nodes. Then, each sub-node has to perform the same operation to classify data, but for each sub-node, the amount of computations is reduced. I.e. a node t_i, with n classifications, is split into two sub-nodes t_i^0, t_i^1. Then, t_i^0 and t_i^1 perform $\frac{n}{2}$ classifications each. Introducing sub-nodes therefore, results in a lower chance for an attacker to control nodes with higher corruption factors. In Fig. 2a an example for split nodes is shown. Here, node t_6 is split into t_6^0 and t_6^1, resulting in four final classifications t_7^0, t_8^0, t_7^1, and t_8^1, where the classifications t_n^i and t_n^j are equal and correspond to the same original classification. When the splitting is performed on the root node, the average success of an attacker is decreased the most, because this introduces a new level of nodes. This approximately doubles the nodes in the tree from $2^l - 1$ to $2^{l+1} - 1$, where l is the number of layers. E.g. a tree with 3 layers and 7 nodes, receives a new layer with 8 nodes. When considering a single attacked node, the average success in this example is decreased from $\frac{3}{7}$ to $\frac{4}{15}$. The values calculates as follows: on the first layer an attacked root node provides an success of 1, as every classification is influenced. On the second layer l_2, the $n_2 = 2$ nodes have a combined success of 1. As one of the nodes must provide a success of s_1, with $0 \leq s_1 \leq 1$, whereas the other node provides a success of attack of $s_2 = 1 - s_1$. In the next layer l_3, there are $n_3 = 2 \cdot n_2$ nodes with a combined attacker success of 1. Therefore, the total attacker success is $\frac{3}{7}$: every node is picked with probability $\frac{1}{7}$, therefore nodes from the root layer are picked with probability $\frac{1}{7}$, from the second layer with probability $\frac{2}{7}$, and on the third layer with probability $\frac{4}{7}$. The average success, then is calculated as $1 \cdot \frac{1}{7} + 0.5 \cdot \frac{2}{7} + 0.25 \cdot \frac{4}{7} = \frac{3}{7}$, for 7 nodes. In general, the attacker success with one attacked node is $\frac{l}{2^l - 1}$, where l is the number of layers. Introducing another layer reduces the impact at the cost of doubling the used nodes. When introducing node splitting at a leaf node only 2 additional nodes are introduced. This decreases the attacker's success from $\frac{l}{2^l - 1}$ to $\frac{1}{1+2^l}(l + \frac{1}{2^l})$. Here, the improvement gets

[1] https://github.com/Blackjack92/Distributed-Decision-Trees-Corruption-Analysis

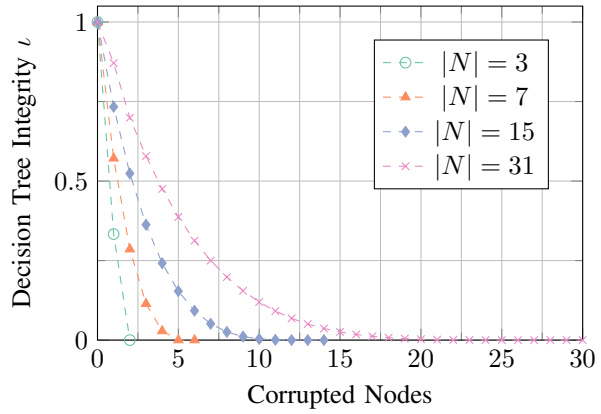


(a) Splitting nodes, such that sibling nodes calculate the same model

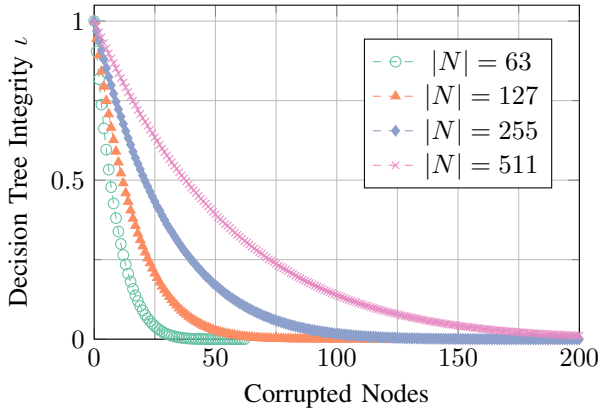
(b) Inserting additional subtrees without impact

(c) Quorum-based decision

Figure 2: Different protection measures in distributed decision trees



(a) Analysed averaged decision tree integrity



(b) Simulation averaged decision tree integrity

Figure 3: Corrupted nodes and their impact on unprotected distributed decision trees

smaller, the larger the tree is.

This approach therefore is the most efficient, w hen the attacker does not know about the tree. Nodes with higher impact are split, at the cost of doubling the number of nodes. Additional nodes and their impact on the distributed decision tree integrity is shown in Table I. The upper part of the table considers two corrupted nodes and the lower part considers about 20% of corrupted nodes. For some node numbers (e.g. 7) it is not possible to reach exactly 20%, which is why the nearest corrupted node number is used as alternative. From the second part of the table it can be seen that the integrity of the distributed decision tree, despite the corruption percentage of 20% remaining the same, deteriorates. Thus, the scaling of the nodes is only efficient if the number of corrupted nodes does not increase to the same extent.

Table I: Correlation between number of nodes and distributed decision tree integrity

Number of Nodes	*Number of Corrupted Nodes*	*Percentage of Corrupted Nodes*	*Integrity*
7	2	28.6%	28.57
15	2	13.3%	52.38
31	2	6.5%	69.89
63 (simulation)	2	3.2%	81.74
127 (simulation)	2	1.6%	89.29
255 (sinmulation)	2	0.8%	93.79
511 (simulation)	2	0.4%	96.48

Number of Nodes	*Number of Corrupted Nodes*	*Percentage of Corrupted Nodes*	*Integrity*
7	1	14.3%	57.14
15	3	20%	36.26
31	6	19.4%	31.27
63 (simulation)	13	20.6%	23.45
127 (simulation)	25	19.7%	20.70
255 (sinmulation)	51	20%	16.35
511 (simulation)	102	20%	13.27

Theorem 5.1: The corruption of a node t_l in a lower tree level has a bigger corruption factor than the corruption of a child node t_h of t_l in a higher tree level. The root node has tree level 0 and is on the lowest level, the leaf nodes are at the highest tree level.

Proof: A parent node t_l has two children t_{h1} and t_{h2}. Assume that $\eta(t_h) = \eta(t_{h1}) = \alpha \cdot \eta(t_{h2})$ and $t_h > 0$, $\alpha > 0$. It follows that the corruption factor is $\eta(t_l) = \eta(t_{h1}) + \eta(t_{h2}) = \eta(t_h) \cdot (1 + \alpha)$ which leads to $\eta(t_l) > \eta(t_h)$. ■

C. Inserting Subtrees without Impact

Another method to reduce the attacker's success is by introducing stubs. Here, a stub is a subtree for the decision tree where the corruption factor is 0. This means the nodes inside a stub have no values to classify and therefore do not perform computations. A stub can consist of one or multiple nodes. Because all classifications are performed outside of the stubs, attacked nodes inside a stub do no influence the outcome of the overall results of the trees. Stubs are displayed in Fig. 2b. Here, node t_1 has a classification rule, where all values are forwarded to node t_2 and none are forwarded to the stub. When a stub is appended to a leaf node the attacker success is reduced from $\frac{l}{2^l-1}$ to $\frac{1}{n+2^l}(l+\frac{1}{2^{l-1}})$, where n is the number of nodes in the stub and l is the number of layers.
This approach is efficient when the attacker has no knowledge about the tree and does not know which nodes are in the stub. The disadvantage is, that these nodes cannot participate in the decision procedure.

It is important to consider that simply shifting visits to another node does not provide any advantage across all possibilities. As an example, take nodes t_2 and t_3 in Fig. 1. If t_2 is treated as a node with 0 visits and t_3 as a node with 100 visits, then the corruption of t_2 has no effect on the integrity of the decision tree. Conversely, the corruption of t_3 has even more of an impact. Therefore, the calculation of the decision tree integrity with $t_2 = 20$ and $t_3 = 80$ is equal to $t_2 = 0$ and $t_3 = 100$.

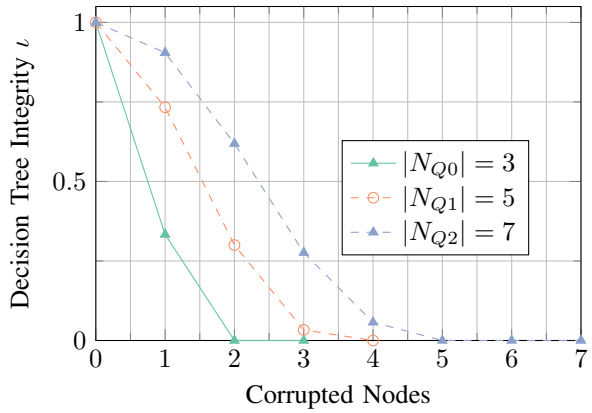
D. Quorum-Based Decisions

A different protection measure is by introducing quorum-based decisions (see Fig. 2c). To find a quorum, a set of nodes is picked that have to make the same decision. I.e. instead of one node performing the classification, multiple nodes perform the classification and have to decide on an outcome. Two outcomes have to be considered. Either no attack occurred, then the results of the classification are the same for all nodes. In the second case, an agreement on one of the different results has to be found. In a byzantine quorum, a predefined threshold τ has to be met to agree on the final value. The threshold can be chosen as $\tau = 0.5 \cdot |Q|$, where $|Q|$ is the size of the quorum. I.e. a majority of the quorum members is sufficient for a decision. Finding the nodes for a quorum can be made by a predefined rule, or by randomly choosing nodes from the network. This balances the computation across the network and prevents attackers, with knowledge about the tree, from using this knowledge to control a majority inside a quorum. When considering a majority vote inside each quorum Q of size $|Q|$, an attacker with less than $0.5 \cdot |Q|$ attacked nodes in total has a success probability of 0, when a quorum is used at every node, because no quorum decision can be outvoted by the attacker. When considering only small numbers of quorums, this approach is still very efficient against an attacker with knowledge about the tree.

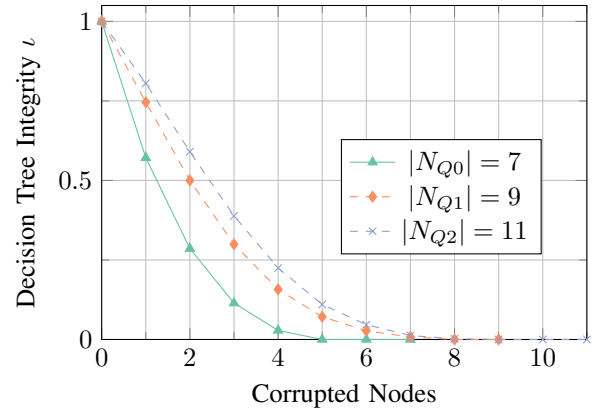
Theorem 5.2: Replacing a decision node t by a quorum decision of multiple nodes $Q = \{q^1, q^2, ..., q^n\}$ decreases the success probability of an attack.

Proof: Let $\gamma(t)$ be the probability of success for an attack on a node t. Assuming that an attack corrupts one node at a time, the success probability for an attack on a single node is $\gamma(t) = 100\%$. If a single node is replaced by the nodes $Q = \{q^1, q^2, ..., q^n\}$, an attack must corrupt more than $\frac{n}{2}$ nodes to be successful. It follows that as long as $n > 2$ and each node has equal co-decision rights in a quorum decision $\gamma(T) = 0\%$. ■
The use of quorums significantly increases the number of nodes and thus potentially corruptible nodes. At least two additional nodes must be inserted per quorum. If a decision path contains a quorum, additional computing power is required per node in the quorum. The additional computing power must not be neglected and must be taken into account when building a decision tree. A comparison between decision trees without protection measurement and quorum protection measurement is shown in Fig. 4a and Fig. 4b. The index represents the number of quorums used in the decision tree, i.e. Q0 is no quorum, Q1 is a quorum of three nodes, and Q2 is two quorums of three nodes each. Fig. 4a has three and Fig. 4b has seven origin nodes. The decision of the tree is the same because no additional decision constraints are added. The quorums are placed at every possible position in the decision tree and then the average decision tree integrity is calculated. It can be seen that without the use of a protection measurement, a corrupted node has a stronger effect than with a protection measurement. The reason for this is that a corrupted node has too little influence in the quorum, so it cannot decide on its own.

Further analyses are shown in Table II. Section one of the table shows how the number of nodes (without quorum) impacts the integrity of the decision tree with one corrupted node. For smaller decision trees with 7 or 15 nodes, adding a full layer of nodes has a large effect. For 7 to 15 nodes it is 26% and for 15 to 31 nodes it is 20%. The higher the number of nodes, the less the effect of adding a complete node layer. Section two shows the effect of a single quorum on the integrity of the decision tree. Identical to section one, the impact of adding a quorum is stronger on smaller decision trees. Based on the ratio of the additional nodes of a quorum to the total number of nodes, this behavior is understandable. The third section shows how 20% of corrupted nodes affect a decision tree protected with a single quorum. As before, it can be seen that a single quorum has little effect in larger decision trees. For this purpose, Table II is compared with Table I. With 63 nodes and 13 corrupted nodes, the difference in integrity is approximately 1.5%. The last section of the table sets the number of quorums equal to the number of corrupted nodes. Note that the amount of



(a) Comparison of averaged decision tree integrity without protection and with quorum protection for a decision tree with three origin nodes



(b) Comparison of averaged decision tree integrity without protection and with quorum protection for a decision tree with seven origin nodes

Figure 4: Corrupted nodes and their impact on quorum protected distributed decision trees

additional nodes due to a quorum is larger than the number of corrupted nodes, since there are at least two additional nodes in each quorum. For quorums with a total size of three nodes, it can be seen that the integrity of the decision tree is reduced, even if the ratio between the number of quorums and the number of corrupted nodes remains constant.

The positioning of the quorum nodes in the decision tree is of crucial importance. An initial consideration of this is given in Table III. These simulation results are based on 10,000 iterations each. The table distinguishes between quorum positioning closer to the root node (top) or closer to the leaf nodes (bottom). In the first section, one quorum and one corrupted node are assumed. A top positioning of the quorum always results in a better integrity of the decision tree. Assuming only one corrupted node, the optimal positioning of a quorum would be at the root position. Thus, an attack never has the success probability of 100%. The second table section considers about 20% corrupted nodes and also shows that the top positioning

Table II: Impact of quorums inside distributed decision nodes

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
7	0	1	57.16
15	0	1	73.49
31	0	1	83.93
63	0	1	90.42

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
7	1	1	71.43
15	1	1	78.04
31 (simulation)	1	1	85.36
63 (simulation)	1	1	90.91

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
7	1	1	71.43
15	1	3	43.78
31 (simulation)	1	6	34.76
63 (simulation)	1	13	24.98

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
7	1	1	71.43
15	3	3	56.68
31 (simulation)	6	6	50.71
63 (simulation)	13	13	43.13

has better performance in terms of decision tree integrity. Finally, the table shows how the integrity of the nodes also deteriorates with top or bottom positioning over the increase of corrupted nodes despite the increase of the quorum number (compare Table II). Furthermore, it can be seen that there is a significant difference of $> 10\%$ between the positioning types, which is why top positioning should always be preferred to bottom positioning.

Despite all the advantages of the top positioning compared to the bottom positioning, the top positioning does not correspond to the optimal positioning. This non-intuitive positioning problem is defined here as the Decision Tree Paradox or Decision Tree Problem. The simplest example is a decision tree with three nodes, one root and two leaf nodes, three corrupted nodes and two quorums. There are two positioning possibilities a) root and leaf nodes are defined as quorums and b) the two leaf nodes are defined as quorums. In case a) the decision tree integrity is 27.14% and in case b) the integrity is 28.57%. Thus, it is shown that there are cases where the top positioning is not optimal. One reason for the phenomenon is that in case b) there is only one way to achieve a corruption of 100%. This is when the root node is corrupted $\binom{6}{2} = 15$ possibilities. In case a) there are besides the possibilities to corrupt the root node $\binom{3}{3}\binom{4}{0} + \binom{3}{2}\binom{4}{1} = 13$ another 3 possibilities to reach a 100% corruption using the leaf nodes. This gives a total of 16 possibilities for a 100% corruption of the decision tree.

Table III: Impact of quorums inside distributed decision nodes

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
15 (top)	1	1	82.3
15 (bottom)	1	1	77.18
31 (top)	1	1	87.98
31 (bottom)	1	1	84.97
63 (top)	1	1	92.37
63 (bottom)	1	1	90.42

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
15 (top)	1	3	48.62
15 (bottom)	1	3	42.63
31 (top)	1	6	38.75
31 (bottom)	1	6	34.34
63 (top)	1	13	27.90
63 (bottom)	1	13	24.93

Number of Nodes	*Number of Quorums*	*Number of Corrupted Nodes*	*Integrity*
15 (top)	3	3	65.25
15 (bottom)	3	3	53.94
31 (top)	6	6	59.98
31 (bottom)	6	6	47.11
63 (top)	13	13	54.06
63 (bottom)	13	13	39.37

6. Conclusion

Decision trees are a central method for classifying data. Especially speed, traceability, and the possibility to distribute computations across multiple machines are advantages when using decision trees. The more features the input data has, the more likely it is that the distribution of nodes is profitable. However, the integrity of the actual decision-making process must be protected. For this purpose, we discussed three different protection measures a) splitting single nodes into several subnodes and thereby reducing the influence of a single node b) adding new subtrees that do not affect the original decision making, and c) securing nodes with quorums and thereby ensuring that the attack must have corrupted the majority in the quorums.

Moreover, a decision tree with protection mechanisms is compared to a decision tree without protection mechanisms. This shows that the number of nodes and the usage of quorums makes a big difference, especially in individual decision tasks. Furthermore, the position (top or bottom based) of a quorum inside the decision tree is crucial. The trade-off between the higher number of nodes and security must be taken into account.

Further research questions deal with an additional trust score which increases the trustworthiness of specific nodes, a deep dive into the Decision Tree Paradox and a thorough analysis of round-based quorum decisions. In a round-based quorum decision, a new quorum is created every time a quorum did not agree on the same result. Moreover, a comparison to distributed neural networks can be considered.

7. References

[1] K. Wallis, F. Schillinger, C. Reich, and C. Schindelhauer, "Safeguarding data integrity by cluster-based data validation network," in *2019 Third World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pp. 78–86, IEEE, 2019.

[2] K. Wallis, F. Schillinger, C. Reich, and C. Schindelhauer, "Corrupted Nodes and their Impact on a Distributed Decision Tree," *The World Congress on Internet Security (WorldCIS-2020)*, 2020.

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

[4] N. V. Chawla, "C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure," in *Proceedings of the ICML*, vol. 3, p. 66, 2003.

[5] A. Ghosh, N. Manwani, and P. Sastry, "On the robustness of decision tree learning under label noise," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 685–697, Springer, 2017.

[6] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[7] M. J. Aitkenhead, "A co-evolving decision tree classification method," *Expert Systems with Applications*, vol. 34, no. 1, pp. 18–25, 2008.

[8] W. N. H. W. Mohamed, M. N. M. Salleh, and A. H. Omar, "A comparative study of reduced error pruning method in decision tree algorithms," in *2012 IEEE International conference on control system, computing and engineering*, pp. 392–397, IEEE, 2012.

[9] A. Desai and S. Chaudhary, "Distributed decision tree," in *Proceedings of the 9th Annual ACM India Conference*, pp. 43–50, 2016.

[10] A. Desai and S. Chaudhary, "Distributed decision tree v. 2.0," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 929–934, IEEE, 2017.

[11] M. Vukolić *et al.*, "The origin of quorum systems," *Bulletin of EATCS*, vol. 2, no. 101, 2013.

[12] D. Malkhi and M. Reiter, "Byzantine quorum systems," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, (New York, NY, USA), p. 569–578, Association for Computing Machinery, 1997.

[13] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93, 2004.

[14] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[15] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali, "A comparative study of decision tree id3 and c4. 5," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 2, pp. 13–19, 2014.

[16] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[17] A. Murdopo, "Distributed decision tree learning for mining big data streams," *Master of Science Thesis, European Master in Distributed Computing*, 2013.