Optimization of Fuzzy Rule Induction Based on Decision Tree and Truth Table: A Case Study of Multi-Class Fault Diagnosis

Abdelouadoud Kerarmi[1][a], Assia Kamal-Idrissi[1][b] and Amal El Fallah Seghrouchni[1,2][c]

[1]*Ai Movement - International Artificial Intelligence Center of Morocco - Mohammed VI Polytechnic University, Rabat, Morocco*

[2]*Lip6, Sorbonne University, Paris, France*

Keywords: Fuzzy Logic, Decision Tree, C4.5 Algorithm, Truth Table, Rule Induction, Knowledge Representation, Multi-Classification, Combinatorial Complexity, Fault Diagnosis.

Abstract: Fuzzy Logic (FL) offers valuable advantages in multi-classification tasks, offering the capability to deal with imprecise and uncertain data for nuanced decision-making. However, generating precise fuzzy sets requires substantial effort and expertise. Also, the higher the number of rules in the FL system, the longer the model's computational time is due to the combinatorial complexity. Thus, good data description, knowledge extraction/representation, and rule induction are crucial for developing an FL model. This paper addresses these challenges by proposing an Integrated Truth Table in Decision Tree-based FL model (ITTDTFL) that generates optimized fuzzy sets and rules. C4.5 DT is employed to extract optimized membership functions and rules using Truth Table (TT) by eliminating the redundancy of the rules. The final version of the rules is extracted from the TT and used in the FL model. We compare ITTDTFL with state-of-the-art models, including FURIA, RIPPER, and Decision-Tree-based FL. Experiments were conducted on real datasets of machine failure, evaluating the performances based on several factors, including the number of generated rules, accuracy, and computational time. The results demonstrate that the ITTDTFL model achieved the best performance, with an accuracy of 98.92%, less computational time outperforming the other models.

1 INTRODUCTION

Classification is a key element of machine learning. It aims to assign labels to new data based on prior knowledge. Various approaches for data classification can be found in the literature. Rule induction can be found among these approaches, it aims to assign labels to data using predefined rules that can be obtained from various methods, including Decision Tree algorithms (DT) and association rule mining. Such rule sets may be used in Rule-Based Systems (RBS) (Durkin, 1990), which can be adopted for classification tasks to support decision-makers. In a broader context, RBS uses predefined rules often shaped by expert knowledge through classical IF-THEN rules (Varshney and Torra, 2023). However, it fails to cover the imprecision and uncertainty presented in the expert's knowledge. Therefore, Fuzzy Rule-Based Systems (FRBS) emerged to deal with this imprecision and uncertainty by exemplifying a distinct subset of these rules based on the theory of fuzzy sets (Zadeh, 1965). FRBS were born by combining FL with RBS, they are a practical application of FL and also known as a fuzzy inference system. FL is an Artificial Intelligence (AI) branch that embraces decision-making and logical reasoning. This technique has become a powerful tool for modeling complex dynamic systems by dealing with the vagueness and uncertainty in information in various domains imitating human reasoning, including multi-classification problems. However, FL has some limitations that must be addressed concerning the identification of the fuzzy sets of quantitative attributes, their membership functions and fuzzy rules, which are mostly manually generated (Elbaz et al., 2019). These fundamental FL steps require expert knowledge that can be subjective (Tran et al., 2022). In addition, a huge database can ultimately lead to combinatorial complexity and rule base expansion, making FL system design difficult to maintain and sustain in real-time (Hentout et al., 2023). Noting that the com-

[a] https://orcid.org/0000-0003-3056-3229
[b] https://orcid.org/0000-0001-6396-9685
[c] https://orcid.org/0000-0002-8390-8780

Kerarmi, A., Kamal-Idrissi, A. and Seghrouchni, A.
Optimization of Fuzzy Rule Induction Based on Decision Tree and Truth Table: A Case Study of Multi-Class Fault Diagnosis.
DOI: 10.5220/0012378900003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 2*, pages 312-323
ISBN: 978-989-758-680-4; ISSN: 2184-433X

binatorial complexity of fuzzy rules can be exponential in the worst case, it corresponds to the cartesian product of all possible combinations of fuzzy sets. Let us assume a fuzzy rule with two input variables x and y and one output z. The rule then is written as: **if** $x \in I_x$ **and** $y \in I_y$ **then** $z \in I_z$, where $|I_x| = n$, $|I_y| = m$ and $|I_z| = k$, that are represented as linguistic terms where x, y are the *antecedents*, then there are $n \times m \times k$ possible combinations. Thus, this combinatorial complexity and the rule size should be handled, requiring an accurate rule induction method.

Indeed, FL placed in the sight of rule induction interests (Hüllermeier, 2011). New approaches have been found in the literature. The most adopted one combines FL with DTs, a method used for rule induction to extract knowledge from the dataset based on information theory and thus generate fuzzy rules. DT has demonstrated its efficacy in many areas, such as regression, classification, and feature subset selection tasks. DT and FL are very interpretable, their primary keys lies in their high interpretability when compared to other approaches. This interpretability is often prioritized over alternative methods that might achieve greater accuracy but are notably less interpretable (Bertsimas and Dunn, 2017). Additionally, they possess swift induction processes and demand low computational resources (Cintra et al., 2013).

To the extent of the authors' knowledge, only some studies use the DT model to generate rules for the FL model, it can classify data and provide valuable information into classes based on features. Nevertheless, the number of generated rules from a complicated dataset, the classification accuracy and computational time in this context fell short of the desired performance levels. To overcome these limitations, This paper proposes a new Integrated Truth Table in Decision Tree-based FL model (ITTDTFL) that generates optimized fuzzy sets and rules. C4.5 DT is employed for optimized membership functions and rules extraction using TT by eliminating the redundancy of the rules. The TT technique is presented previously (Kerarmi et al., 2022). The latter smartly and automatically generates a relatively small number of understandable fuzzy rules and membership functions, leading to better results in terms of time complexity interpretability. The data used corresponds to real industrial datasets collected from pumps. We exploit this information generated from the DT to build an FL model that can merge the advantages of the DT and TT techniques to create a robust and efficient model for classification issues. Integrating TTs to reduce the number of fuzzy rules and optimize membership functions while improving accuracy adds significant value to the ITTDTFL model.

The remainder of this paper is organized as follows. In Section 2, we review the relevant literature. Section 3 describes the methodology. Section 4 discusses the experiment's results. Finally, in Section 5, we conclude the paper and outline directions for future research.

2 RELATED WORK

Since FL appears to be a robust model for classification issues in the literature, many researchers have tried to improve it. The most widely used rule-based generation approach is data clustering, which aims to group data into clusters based on a similarity measure. From these clusters, fuzzy sets can be obtained. In (Chiu, 1997), a subtractive clustering method with fuzzy rules extracted from the presented data groups the data point with many neighboring data points as a cluster center, and the neighboring data points are linked to this cluster. Another clustering method in (Gómez-Skarmeta et al., 1999) called fuzzy clustering is used to generate fuzzy rules, where data elements can be assigned to multiple clusters, and each data point is assigned to membership levels, denoting the extent of its association with one or more clusters. Also, in (Reddy et al., 2020) an adaptive genetic algorithm is used to optimize generated rules from a fuzzy classifier to predict heart disease. Numerous studies have proposed the use of optimization algorithms for fuzzy rule generation. One study, for example, suggests using a genetic algorithm (Angelov and Buswell, 2003), which simultaneously estimates the structure of the rule base and the parameters of the fuzzy model from the available data. A hybrid intelligent optimization algorithm is proposed in (Mousavi et al., 2019) to generate and classify fuzzy rules and select the best rules in an if–then FRBS. A method based on subtractive clustering using a genetic algorithm for optimized fuzzy classification rules generation from data is presented in (Al-Shammaa and Abbod, 2014). Other inductive learning algorithms based on FL models, such as fuzzy grid-based CHI algorithm (Chi et al., 1996) and the genetic fuzzy rule learner SLAVE (Gonzblez and Pérez, 1999) were also proposed. In addition, particle swarm optimization is employed to generate the antecedents and consequences of the other models like fuzzy rule base (Prado et al., 2010). Another method that uses DT to generate fuzzy rules and employs a genetic algorithm to optimize these rules was developed in (Kontogiannis et al., 2021), the model achieved an accuracy of 89.2%, generating 281 rules. A similar method proposed in (Ren et al.,

2022) converts the path generated from traversing a DT based on the ID3 algorithm into a set of fuzzy rules. Authors in (Tran et al., 2022) also proposed a Node-list Pre-order Size Fuzzy Frequent (NPSFF) algorithm for fuzzy rule mining, which has proven efficient in other important metrics, notably computational time and memory consumption. Besides clustering and data optimization algorithms, several methods have been proposed for rule generation (Mutlu et al., 2018). However, two algorithms are taking over the literature in rule induction for Classification issues, RIPPER (Cohen, 1995) and FURIA (Hühn and Hüllermeier, 2009), they are still references for comparison with other algorithms, notably C4.5 and other genetic algorithms. A full comparison between FURIA, RIPPER, C4.5, fuzzy grid-based CHI algorithm, and the genetic fuzzy rule learner SLAVE models is presented in (Hühn and Hüllermeier, 2010). These models were run on 45 real-world classification datasets from the UCI, Statlib repositories, agricultural domain, and others. RIPPER and C4.5 gave good results in classification accuracies, but FURIA was the best. In previous work (Kerarmi et al., 2022), the authors proposed to use TT in FL. The Integrated TT in FL (ITTFL) model aims to represent the logic between machine states and generate optimized rules of FL. A series of tests were conducted to justify the choice of the type of membership function used. The results showed that the Trapezoidal membership function gave more accurate results than the Triangular and Gaussian membership functions. Trapezoidal membership functions cover a greater degree of each variable belonging to a given set. However, this approach does not deal with identifying fuzzy sets and their membership functions which are also required to be accurate to have a robust FL model, it requires an absolute classification model of the data. These approaches face computational time and interpretability drawbacks, crucial metrics now considered mandatory. Although methods with higher accuracy exist, interpretability is often preferred. For this reason, DT and FL which are considered very interpretable, are chosen to fill this gap. In brief, FL has known several improvements by integrating different techniques such as DTs, Genetic Algorithms, and Neural Networks. However, these approaches introduce drawbacks such as increased complexity and computational time. Particularly for DT, its greedy behavior where each branch is independently determined, can fail to capture dataset features accurately and lead to duplicated sub-trees and poor performance in classifying future data points.

3 METHODOLOGY

The ITTDTFL model is an extension of the previously proposed model by the authors (Kerarmi et al., 2022) to optimize the fuzzy rules generation based on the TT technique (ITTFL). The ITTDTFL uses a DT to extract knowledge and then optimizes the generated fuzzy rules and membership functions using TT. This section introduces the FL and the DT models and then describes the proposed ITTDTFL model. Figure 1 depicts the architecture of the ITTFL, DTFL and ITTDTFL models.

3.1 Background

3.1.1 Fuzzy Logic Model

FL model is based on fuzzy sets where the linguistic notions and membership functions define the truth-value of such linguistic expressions (Zadeh, 1965). A fuzzy set A in a universe of discourse X is characterized by a membership function $\mu_A(x)$ that assigns a value in the interval [0, 1] to each element x in X. This membership function represents the degree to which x belongs to the set A. The FL System consists of four steps (Hentout et al., 2023): Fuzzification, Fuzzy knowledge base, Inference engine, and Defuzzification.

1. **Fuzzification:** Consisting of converting numerical inputs into linguistic variables represented by membership degrees in fuzzy sets.
2. **Fuzzy Knowledge Base:** Representing the relationship between the input variables x and the output variables y. A fuzzy rule has the following form: "IF *antecedent* THEN *consequent*," where both the *antecedent* and *consequent* involve linguistic variables and fuzzy sets.
3. **Inference Engine:** Performing logical deductions and drawing conclusions based on the rules and knowledge contained in the system's knowledge base.
4. **Defuzzification:** Aggregating fuzzy output, which represents the system's conclusion or decision, is converted into a crisp that can be easily understood.

3.1.2 C4.5 Algorithm

C4.5 Algorithm is a DT algorithm developed by Ross Quinlan (Quinlan, 2014). It is an extension of the ID3 (Iterative Dichotomiser 3). The C4.5 algorithm constructs a DT from a given dataset by partitioning data recursively based on feature attributes. It can handle

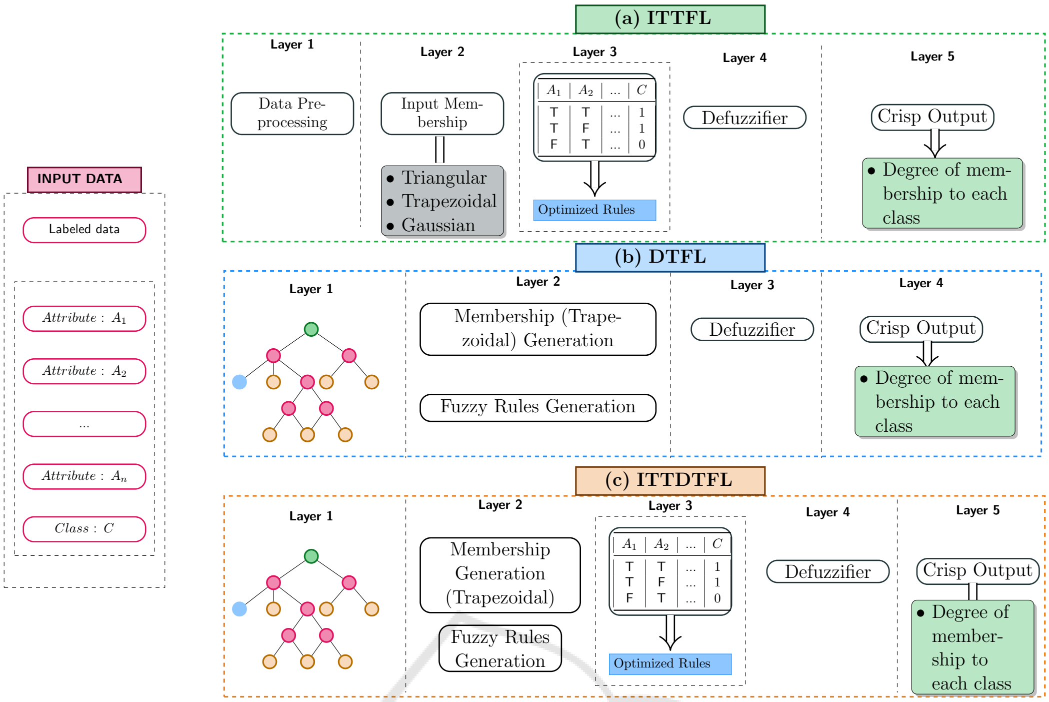


Figure 1: The framework of the three models.

numerical and categorical features and is mainly used for classification tasks. The C4.5 algorithm based on entropy and the information gain allows the generation of a DT. The equation (1) presents the entropy to measure the purity and homogeneity within the data while equation (2) corresponds to information gain to determine the best attribute to use for data splitting at each node (Hssina et al., 2014).

$$\text{Entropy}(S) = -\sum_{i=1}^{n} p_i \times \log(p_i) \tag{1}$$

$$\text{Gain}(S,T) = \text{Entropy}(S) - \sum_{j=1}^{n} (p_j \times \text{Entropy}(p_j)) \tag{2}$$

Where:

- $Entropy(S)$: The entropy of the dataset S.
- p_i: The proportion of instances in S that belong to class i.
- $Gain(S,T)$: The gain achieved by splitting the dataset S using attribute T.
- p_j: The set of all possible values for attribute T.

Here is a brief overview of the C4.5 algorithm steps:

1. The algorithm selects the best attribute for splitting the data starting with the root node. The splitting criterion in C4.5 is based on the entropy and the information gain ratio, which considers the number of choices in a given attribute.
2. The algorithm divides the data according to the selected attribute and then creates child nodes for each possible attribute value.
3. The algorithm recursively repeats steps 1 and 2 for each child node until a stopping condition is satisfied. This condition may be reaching a maximum depth, having a minimum number of samples at a node, or meeting other predefined criteria.
4. The algorithm assigns a class label to each leaf node based on the dominant class of the training samples at that node.
5. The algorithm prunes the tree to reduce overfitting by deleting nodes or merging branches.

3.2 Description of the Proposed Model: ITTDTFL

The ITTDTFL model is an FL-based one that exploits the knowledge extracted from the C4.5 DT without a pruning process to provide all the possible and accurate rules and membership functions for the inference engine. The model's strength relies on using the TT technique to optimize the fuzzy rules and membership functions by merging inclusions of attributes' intervals that build these rules and membership functions.

Figure 2 represents the steps of the ITTDTFL model, whereas the pseudo-code is described in Algorithm 1.

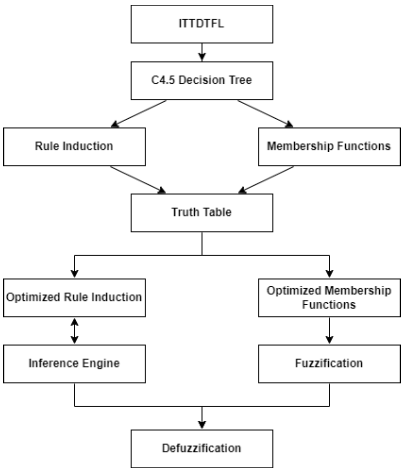


Figure 2: ITTDTFL model flow chart.

Data: dataset D
Result: degree of membership to each class: Class_degree
begin
  D ← Read Data;
  Tree ← decisionTreeC4.5(D, Features, Target);
  TreeRules ← ruleExtraction(Tree, rules, currentRule);
  Intervals ← intervalRuleExtraction(TreeRules, Lists);
  OIntervals, FuzzyRules ← truthTable(Intervals, Lists);
  MembershipFunctions ← (TrapezoidalMF, OIntervals);
  Class_degree ← fuzzuLogicModel(MembershipFunctions, FuzzyRules);
  return Class_degree
end

Algorithm 1: ITTDTFL model.

The algorithm starts by generating a DT without pruning process from the dataset using `decisionTreeC4.5` function. Next, `ruleExtraction` function uses Depth-first search (DFS) to traverse the generated tree from the root node to the deepest leaves, identifying rules along each path (see the description in Algorithm 2). For instance, the output of this step on classifying two attributes (A_1 and A_2), based on one target (C_z), is a set of rules described in Table 1.

The intervals are then extracted using `intervalRuleExtraction` function, where each line is transformed into a rule containing intervals of attributes and the corresponding class. Respecting greater than and smaller than symbols (see Algorithm 3). For example, considering in line 1: '*if (*$A_1 <= 0.05$*) and (*$A_2 > 0.015$*) then class:* C_z *(proba: 100.0%) — based on x samples*', the condition: *(*$A_1 <= 0.05$*)* can be written as A_1 *= [X, 0.05]*, while the condition: *(*$A_2 > 0.015$*)* can be written as A_2 *= [0.015, Y]*, where X, Y represents the Min and Max values that attribute A_1, A_2 can take based on the dataset. Considering a second line, the condition is as follows: ($A_1 > 0.05$) and ($A_2 <= 0.002$) and ($A_2 > 0$) and ($A_2 > 0.001$)..., besides A_1, attribute A_2 also can be transformed to an interval; A_2 *= [0.001, 0.002]*, and so on.

Data: Tree
Result: TreeRules
Function TraverseDecisionTree(node, rules, currentRule) ;
begin
  if *node.class_label is not Null* **then**
    currentRule.append("class: " + node.class_label) ;
    rules.append("if " + currentRule.join(" and ") + " then " + node.class_label) ;
  else
    if *node.attribute is not Null and node.operator is not Null and node.threshold is not Null* **then**
      currentRule.append("(" + node.attribute + " " + node.operator + " " + node.threshold + ")") ;
    end
    for *value, childNode in node.children.items()* **do**
      TraverseDecisionTree(childNode, rules, copy(currentRule)) ;
    end
  end
  return rules
end

Algorithm 2: ruleExtraction.

This step allows the conversion of every line to a new rule line containing intervals for each attribute and its corresponding class, written as: $AttributeX_p$ & $AttributeY_q$ then Class : C_Z. Next, from these lines, `truthTable` function, described in Algorithm 4, is used to create a TT for interval processing. DTs can generate a large number of rules and intervals, and this number is likely to grow as the size and complex-

Table 1: Example of the output of `ruleExtraction` function. A represents the attributes, t is the threshold determined from the tree for each split $t \in \mathcal{R}^+$.

if (A_1 <= t) and (A_2 <= t) then class: C_z (proba: 100.0%) — based on x samples
if (A_1 > t) and (A_2 <= t) and (A_2 > t) and (A_2 > t) then class: C_z (proba: 100.0%) — based on x samples
if (A_1 > t) and (A_2 > t) and.... and (A_2 > t) then class: C_z (proba: 100.0%) — based on x samples
...

Data: TreeRules
Result: Intervals
begin
  Initialize *intervals* dictionary $\{A_1$: [ ], A_2: [ ]$\}$;
  Define patterns: Interval patterns ("$A_1 \leq$ value", "$A_2 >$ value"), Class pattern ("Class : value");
  foreach *line in TreeRules* **do**
    foreach *match in line using patterns* **do**
      Extract *attribute* (A_1 or A_2) and *value*;
      Convert *value* to a floating-point number;
      Update *intervals[A_1]* or *intervals[A_2]* with the extracted *value*;
      Extract the *Class* and the *value*;
      Convert the *value* to string;
      Update *Class* with the extracted *value*;
    end
    A_1_index = [start, end] & A_2_index = [start, end] Class : C_Z;
    save to Intervals;
  end
end

Algorithm 3: intervalRuleExtraction.

ity of the data increases. Therefore, these intervals must pass through the proposed process in order to be reduced to avoid useless computations. The TT approach identifies and merges the inclusions within the extracted intervals. The TT contains attributes, for example, two attributes A_1, A_2, and the classes as columns, while in rows, there are intervals I_{A_1} and I_{A_2} for each state, 1 if the class is true 0 if it is false, as shown in Table 7. First, a grouping process of the attributes by class (where Class = 1), then for each group, it compares the extracted intervals for each attribute. Technically, it creates a list for each row of attributes in the group L_{A_1} & L_{A_2} of intervals. Next, a comparison between intervals of each list is made. A merging process of the inclusions is executed. Taking for example L_{A_1} contains 4 sets or intervals $I_{A_{1_1}}$, $I_{A_{1_2}}$, $I_{A_{1_3}}$ and $I_{A_{1_3}}$ as Figure 3 shows, it is obvious that interval $I_{A_{1_1}}$ includes $I_{A_{1_2}}$ and $I_{A_{1_3}}$ includes $I_{A_{1_4}}$, thus, only $I_{A_{1_1}}$ and $I_{A_{1_3}}$ will be kept and respectively replace intervals $I_{A_{1_2}}$ and $I_{A_{1_4}}$. The use of such an approach has notably reduced the number of intervals as well as the extracted rules.

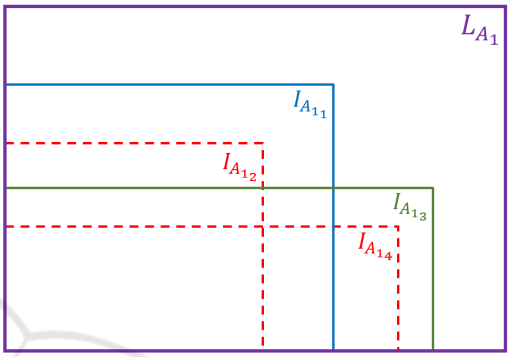


Figure 3: L_{A_1} Intervals inclusion property.

After having the TT's final version, all the intervals left are transformed into Trapezoidal Membership Functions using `membershipF` function. The trapezoidal membership function is a graph representing the degree to which an element belongs to a certain fuzzy set. It has four parameters: *the left and right edges, a lower plateau, and an upper plateau.* These parameters determine the shape of the trapezoid, which represents the fuzzy sets. For example, a fuzzy set of "tall people." the trapezoidal membership function for this set could have the following parameters:

- Left boundary point: 150 cm
- Right boundary point: 200 cm
- Lower plateau: 160 cm
- Upper plateau: 190 cm

We chose this type of membership function based on previous work comparing the Trapezoidal membership function with Triangular and Gaussian membership function (Kerarmi et al., 2022). Based on the results, adopting the Trapezoidal membership function for an FL model gave better results simply because it rates the degree of belonging at 100% that an element belongs to a fuzzy anywhere between the lower and upper plateau. For example, if we use 'Tall' as a linguistic term to describe values that fall within the upper and lower plateaus, it means all people between 160 and 19 cm are Tall. Unlike the Triangu-

Data: Intervals
Result: OIntervals
Initialize an empty truth table, a 2D array with rows for each data point and columns for each class;
while *Reading lines* **do**
    foreach *each line in the lines* **do**
        Initialize an empty row in the truth table with all zeros;
        Extract the values of attribute 1 and attribute 2 from the current line;
        foreach *each class* **do**
            if *attribute 1 is in the class's interval AND attribute 2 is in the class's interval* **then**
                Set the corresponding cell in the truth table to 1;
            end
        end
    end
    Group table by Class = 1;
    Initialize an empty list S for storing optimized intervals;
    $S \leftarrow Group$;
    $L_1 \leftarrow length(I_1)$;
    $L_2 \leftarrow length(I_2)$;
    foreach *two intervals I_1 and I_2 in S* **do**
        if $I_1[1] \leq I_2[1]$ AND $I_1[L_1] \geq I_2[L_2]$ **then**
            $S \leftarrow S \setminus \{I_2\}$
        end
    end
end

Algorithm 4: truthTable.

lar and Gaussian membership functions, an element 100% belongs to a fuzzy set only if it is equal to the median of the fuzzy set. Since the vagueness is also represented in the uncertain belonging degree of an element to a particular fuzzy set, logically, the Trapezoidal membership function is the best.

Rules are extracted from the table, where each row represents a rule. Noting that during the extraction of the rules, we considered extracting these rules in a required format to go straight to the rule base: $rule_n$ = ctrl.Rule($Attribute_p$['$MembershipFunction_x$'] &$Attribute_{p+1}$['$MembershipFunction_y$'],Class['C_Z']).

Finally, all the requirements of an FL model are satisfied, and Fuzzy sets and fuzzy rules are automatically generated and well-optimized. The function `fuzzuLogicMode` is used to build an FL model, the final step in our model.

The model identifies the logic between data, extracts, describes, and represents the knowledge from this data. This part is described in the Experiments Section. For the sake of simplicity, the models return the set of failure classes with their probabilities. This probability can be seen as a measure of the likelihood of an occurrence of a failure in real-time. Table 2 represents an extract of rules from the DT.

Table 2: Short example of extracted rules from the DT.

rule1 = ctrl.Rule(A_1_value['A_{1_0}'], $Class$['C_z'])
rule2 = ctrl.Rule(A_1_value['A_{1_1}'] & A_2_value['A_{2_0}'], $Class$['C_z'])
rule3 = ctrl.Rule(A_1_value['A_{1_2}'] & A_2_value['A_{2_1}'], $Class$['C_z'])
...

4 EXPERIMENTS & RESULTS

In order to evaluate our model performance, we benchmark the algorithms DTFL and ITTDTFL with C4.5, FURIA, and Ripper from WEKA library (Witten et al., 2005) implemented using Python 3.8 in the same environment. The evaluation is done by conducting a series of experiments based on several factors (Hambali et al., 2019), including the number of generated rules, computational time, accuracy in Equation 3, and other metrics such as F1-score which indicates the model's capabilities of avoiding false positives (recall) while identifying positive examples (precision) in Equation 6, Sensitivity/recall which evaluates the model's predictions of true positives of each available category in Equation 7, and Receiver Operating Characteristic Curve (ROC) area which indicates the model performance at distinguishing between the classes. The performance of the five models is extensively demonstrated using three sets of data related to pump failure instances. The description of the data, the experiment protocol, and the obtained results are demonstrated later in this section.

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}} \tag{3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{5}$$

$$\text{F-Measure} = \frac{2(\text{Precision-Recall})}{(\text{Precision}+\text{Recall})} \tag{6}$$

where *TP* is the True Positive, *TN* is the True Negative, *FP* is the False Positive ad *FN* is the False Negative.

4.1 Datasets

Two real datasets were collected from two pumps. The data captures various operational parameters, such as the acceleration time waveform (g), velocity spectrum ($fftv$), and acceleration spectrum ($fftg$), saved as lists of observations, as well as the failure class based on the results of a regularly performed Failure Mode and Effect Critical Analysis method (FMECA) (Kerarmi et al., 2022). It contains seven classes where one is a normal state, and six others correspond to different failures. All the datasets include measurements from several sensors installed throughout the machine. Table 3 represents a statistical data view, while Table 4 describes their form. The third dataset is the combination of the two datasets. These rich and comprehensive datasets provide a detailed view of system behavior and form the basis for performance analysis and classification of potential failures. Additionally, these datasets have been pre-processed to eliminate outliers and missing values before further analysis.

Table 3: Datasets statistical description.

Dataset	Number of rows
1	4016
2	3048
3	7064

Table 4: Datasets description.

<table>
<tr><th>P[1]</th><th>g[2]</th><th>fftv[3]</th><th>fftg[4]</th><th>MSC[5]</th></tr>
<tr><td rowspan="7">P1- P2- P3- P4</td><td rowspan="7">Numerical data</td><td rowspan="7">Numerical data</td><td rowspan="7">Numerical data</td><td>Normal State</td></tr>
<tr><td>Imbalance</td></tr>
<tr><td>Structural fault</td></tr>
<tr><td>Misalignment</td></tr>
<tr><td>Mechanical looseness</td></tr>
<tr><td>Bearing lubrication</td></tr>
<tr><td>Gear fault</td></tr>
<tr><td colspan="5">[1] Sensor position; [2] Acceleration time waveform; [3] Velocity spectrum; [4] Acceleration spectrum; [5] Machine State Class.</td></tr>
</table>

First, We calculate the Root Mean Square (RMS) values of ($fftv$) and ($fftg$) since they are critical factors for machinery status diagnosis for signal normalization and to reduce variability. Equation (7) depicts the formula based on this study (Rzeszucinski et al., 2012). We calculate the RMS of all the lists in $fftv$ and $fftg$ of each machine state class identified by the FMECA method using the following formula:

$$x_{rms} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)} \tag{7}$$

We finally got rows that include the root mean square (RMS) values of $fftv$ and $fftg$ for the seven described machine state classes, this provides essential information for the modeling, training, and testing of the proposed models. Figures 4 and 5 represent the values of $fftv$ and $fftg$ plotted as intervals. The definition of specific values of each state is complicated and challenging, given the inclusions and intersections between intervals, as Figures 4 and 5 show. Therefore, we employed the C4.5 algorithm to extract knowledge from the dataset, this knowledge is represented as rules that determine the path from the root node to a child node containing the class name, with these paths being based on the attributes $fftv$ and $fftg$. Table 5 represents the extracted knowledge from the generated DT. Based on the DT results, each row contains thresholds that are used to create intervals $fftv_n$ and $fftg_m$ of each attribute $fftv$ and $fftg$ for each class C. This knowledge is represented by intervals for classes and converted into a Trapezoidal membership function and Fuzzy rules for the FL model. Table 6 shows an extract of rules from the DT.

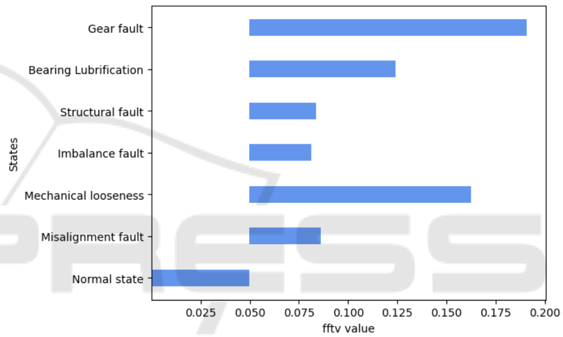


Figure 4: $fftv$ values for each state.

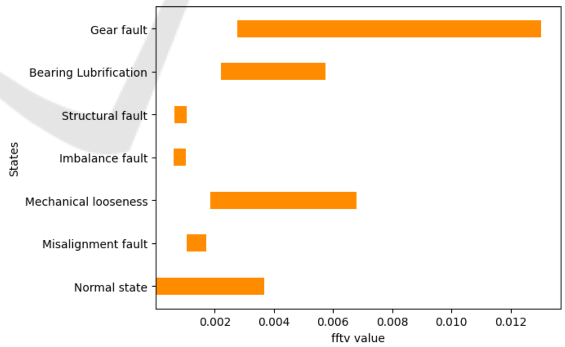


Figure 5: $fftg$ values for each state.

The TT is used directly to merge intervals and avoid rule redundancy, thus considerably reducing the number of rules and membership functions. This has proved effective in the computational time taken for the model ITTDTFL. Table 7 represents the generated TT used for the inclusion merging process.

In the merging process as described in the Methodology Section, the model groups the rows by class where this latter is true equals 1. Then, for each

Table 5: An extract of the knowledge using the DT.

if (fftv <= 0.05) then class: Normal state (proba: 100.0%) — based on 1,513 samples
if (fftv <= 0.05) and (fftg <= 0.002) and (fftg <= 0.001) then class: Misalignment fault (proba: 100.0%) — based on 241 samples
if (fftv <= 0.05) and (fftg <= 0.002) and (fftg <= 0.007) and (fftv <= 0.075) and (fftv <= 0.099) and (fftg <= 0.005) then class: Mechanical looseness fault (proba: 100.0%) — based on 32 samples
if (fftv > 0.05) and (fftg <= 0.002) and (fftg <= 0.001) and (fftv <= 0.073) and (fftv <= 0.081) then class: Structural fault (proba: 100.0%) — based on 8 samples
if (fftv > 0.05) and (fftg <= 0.002) and (fftg <= 0.007) and (fftg <= 0.007) then class: Gear fault (proba: 100.0%) — based on 8 samples
if (fftv > 0.05) and (fftg <= 0.002) and (fftg <= 0.007) and (fftg <= 0.007) and (fftg <= 0.007) then class: Gear fault (proba: 100.0%) — based on 1 samples
if (fftv > 0.05) and (fftg <= 0.002) and (fftg <= 0.007) and (fftv <= 0.075) and (fftg <= 0.003) and (fftv <= 0.05) then class: Gear fault (proba: 100.0%) — based on 1 samples
if (fftv > 0.05) and (fftg <= 0.002) and (fftg <= 0.007) and (fftg <= 0.007) and (fftg <= 0.007) then class: Mechanical looseness fault (proba: 100.0%) — based on 1 samples
...

Table 6: Short example of extracted rules from the DT.

rule1= ctrl.Rule($fftv$['$fftv_1$'], Class['Normal state'])
rule2= ctrl.Rule($fftv$['$fftv_8$'] & $fftg$['$fftg_3$'], Class['Misalignment fault'])
rule3= ctrl.Rule($fftv$['$fftv_{97}$'] & $fftg$['$fftg_{16}$'], Class['Mechanical looseness fault'])
rule4= ctrl.Rule($fftv$['$fftv_{50}$'] & $fftg$['$fftg_1$'], Class['Imbalance fault'])
rule5= ctrl.Rule($fftv$['$fftv_{13}$'] & $fftg$['$fftg_2$'], Class['Imbalance fault'])
rule6= ctrl.Rule($fftv$['$fftv_{74}$'] & $fftg$['$fftg_2$'], Class['Structural fault'])
rule7= ctrl.Rule($fftv$['$fftv_5$'] & $fftg$['$fftg_9$'], Class['Mechanical looseness fault'])
rule8= ctrl.Rule($fftv$['$fftv_{76}$'] & $fftg$['$fftg_8$'], Class['Mechanical looseness fault'])
rule9= ctrl.Rule($fftv$['$fftv_{19}$'] & $fftg$['$fftg_2$'], Class['Structural fault'])
rule10= ctrl.Rule($fftv$['$fftv_{87}$'] & $fftg$['$fftg_{11}$'], Class['Mechanical looseness fault'])

Table 7: TT used for the inclusion merging process.

fftv	fftg	Bearing Lubrication fault	Gear fault	Imbalance fault	Mechanical looseness fault	Misalignment fault	Normal state	Structural fault
$fftv_1$	None	0	0	0	0	0	1	0
$fftv_8$	$fftg_3$	0	0	0	0	1	0	0
$fftv_{97}$	$fftg_{16}$	0	0	0	1	0	0	0
$fftv_{50}$	$fftg_1$	0	0	1	0	0	0	0
$fftv_{13}$	$fftg_2$	0	0	1	0	0	0	0

column of the attributes $fftv$ and $fftg$, each interval is compared to other intervals to check for inclusion; if any inclusion was found, the major interval takes place in the included interval, and so on. These intervals are converted to membership functions, Figure 6 represents membership functions of dataset 3 before optimization. Note that the number of generated membership functions is 130 for fftv and 17 for $fftg$. Algorithm 4 has significantly reduced the number of membership functions, avoiding useless computational effort. The outputs of the optimized membership functions are represented in Figure 7. Finally, by eliminating the redundancies, the TT contains a significantly reduced number of rules and the intervals used for creating membership functions.

4.2 Experiment Protocol

We conducted a series of experiments and split the datasets into training (75%) and testing sets (25%). Table 8 depicts the number of samples used in the training and testing phases and the total number of samples of each dataset.

Table 8: Training/Testing samples.

Dataset	Total number	Training set	Testing set
1	4016	3009	1007
2	3048	2284	764
3	7064	5295	1769

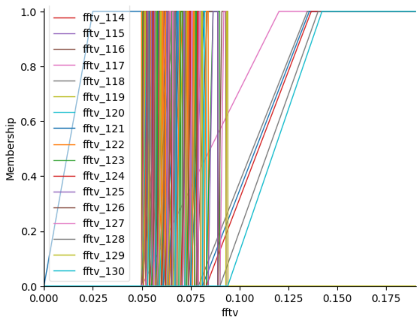
4.3 Results & Discussion

Table 9 shows the performances in terms of the number of generated membership functions for FL-based models, notably DTFL and ITTDTFL models. Table 10 represents the number of generated rules and the computational time of all models, while Table 11 depicts the classification metrics, including the accuracy, sensitivity, F1-score, and ROC area scores.

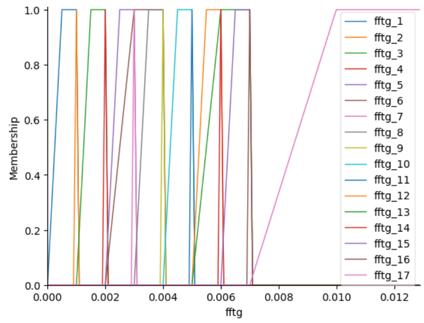
Table 9: Number of generated Membership Functions.

Model	Dataset	Generated Membership Functions number	
		fftv	fftg
DTFL	1	110	19
	2	57	13
	3	130	17
ITTDTFL	1	**20**	**7**
	2	**5**	**10**
	3	**10**	**7**

Regarding computational time and rule number, the FURIA algorithm took 28.73, 8.74, and 45.6 seconds for each dataset to be classified, generating 30, 22, and 31 rules, respectively. RIPPER consistently required longer computational time, from 302.54 seconds to 1277.68 seconds, generating 17, 14, and 20 rules in each experiment. As it is noticed, although the number of rules is relatively small, it took a significant amount of time in order to give results, simply because they need to search for all possible rules that can be used for data classification, as the dataset size increases, it is expected that a model's computational time requirements also proportionally increase. Apparently, RIPPER's requirements have exponen-

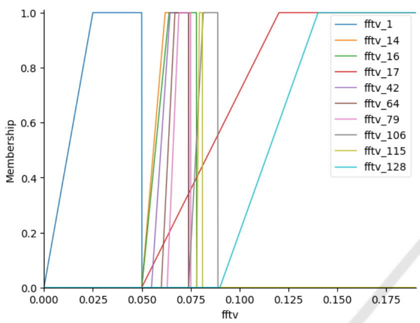


(a) fftv Membership Function

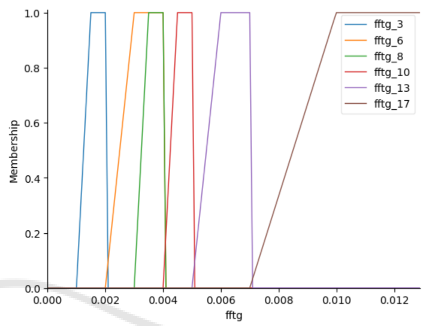


(b) fftg Membership Function

Figure 6: Membership Functions.



(a) fftv Membership Function



(b) fftg Membership Function

Figure 7: Optimized Membership Functions.

tially increased. However, the C4.5 algorithm was the fastest, with 1.2 seconds on average, due to the gain ratio method used for splitting the data, which considers the information gain and the number of values in an attribute. This helps reduce the number of splits required to build the decision tree, making the algorithm faster. In terms of rules, the C4.5 algorithm has generated 23, 10, and 15 rules with the pruning process. However, it fell short of the desired performance levels in terms of classification accuracy. For FL-based models, DTFL also took an important computational time due to the number of generated rules, in the worst case, 287 rules in 364.38 s. Therefore, it is crucial to reduce the number of rules in order to build a faster model. Meanwhile, ITTDTFL has achieved a significant reduction rate of the rules by approximately 86.87%, from 202 to 28 on dataset 1, 89 to 15 on dataset 2, and 287 to 24 on dataset 3. Moreover, the number of the generated membership functions is also notably optimized compared to the DTFL model, as shown in Table 10, ITTDTFL model successively reduced the number of generated membership functions fftv/fftg, from 110/19 to 20/7, 57/13 to 5/10, and 130/17 to 10/7 withing the three datasets. To better represent the differences between models and the critical impact of the number of rules on the computational time, Figure 8 projects the number of rules on the total time taken by the model in the three tests.

Table 10: Number of rules and computational time of each model.

Model	Dataset	Number of generated rules	Runtime (s)
FURIA	1	30	28.73
	2	22	8.74
	3	31	45.62
RIPPER	1	17	674.24
	2	14	302.54
	3	20	1277.68
C4.5	1	23	**1.51**
	2	10	**0.43**
	3	15	**1.79**
DTFL	1	202	127.82
	2	89	23.65
	3	287	364.38
ITTDTFL	1	28	7.45
	2	15	3.1
	3	24	16.08

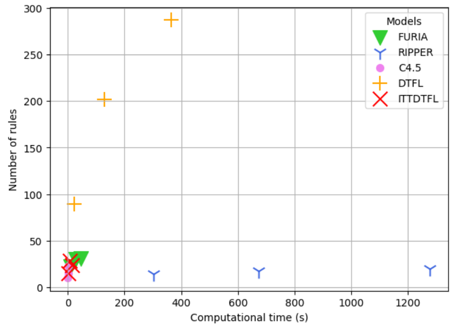


Figure 8: Runtime vs rules number plot.

Table 11: Classification performance of each model.

Model	Dataset	Accuracy (%)	Sensitivity	F1-Score	ROC
FURIA	1	90.28%	0.90	0.90	0.97
	2	93.56%	0.93	0.93	**0.98**
	3	91.59%	0.91	0.91	0.97
RIPPER	1	88.14%	0.88	0.88	**0.98**
	2	93.08%	0.93	0.93	**0.99**
	3	91.14%	0.91	0.91	**0.99**
C4.5	1	88.54%	0.88	0.88	**0.98**
	2	92.78%	0.92	0.92	**0.99**
	3	90.48%	0.90	0.90	**0.99**
DTFL	1	91.45%	0.91	0.90	0.87
	2	95.41%	**0.95**	0.94	0.92
	3	93.15%	0.93	0.93	0.89
ITTDTFL	1	**97.91%**	**0.97**	**0.97**	0.95
	2	**95.94%**	**0.95**	**0.95**	0.90
	3	**98.92%**	**0.98**	**0.98**	0.95

In terms of accuracy, the results obtained from the experiments show that all models did a good job classifying the machine state classes. Considering the number of correct classifications, all models have achieved high accuracy rates, with only a few misclassifications. FURIA, RIPPER, and C4.5 have shown good performances during the different experiments. As expected from these evaluations and others in the literature, FURIA gave the best results, correctly classifying data ranging from 90.28% to 93.56%, for FURIA, 88.14% to 93.08% for RIPPER, while C4.5 correctly classified 88.54% to 92.78%. For FL-based models, the DTFL model also gave good results in terms of accuracy, ranging from 91.45% to 93.49%. Meanwhile, ITTDTFL exhibited excellent accuracies for the three data sets, it attended 95.94%, 97.91%, and 98.92%, accurately classifying the data, enhancing the DTFL model's accuracy by 4.55% and outperformed FURIA, RIPPER, C4.5 successively by 6.92%, 7.5%, 7.32%. These results can be explained by the fact that TT can preserve the most accurate and meaningful membership function corresponding to each class, improving the precision of each fuzzy rule and leading to better classification accuracy. In terms of other metrics, as shown in Table 11, considering 0.9-1.0 is Excellent, and 0.8-0.9 is Good, all the models achieved good to excellent scores in ROC area metric, as well as for sensitivity and F1-score metrics.

To sum up, ITTDTFL successfully optimizes the number of membership functions and accurately induces rules for the FL model. DT is used to generate intervals and rules from the paths of each branch. At the same time, TT eliminates inclusions within generated intervals from these paths, addressing the issue of duplicated sub-trees and enhancing feature capture. This approach results in significantly reduced computational time and improved classification performance. Compared to the related work's results, the ITTDTFL model has significantly outperformed FURIA, RIPPER, C4.5 algorithm, and DTFL model by 4.55% to 7.5% in terms of accuracy and computational time. The ITTDTFL model is very interpretable and easy to manipulate due to its simple structure, domain expert involvement, transparent algorithms, and Human-Understandable rules.

5 CONCLUSIONS & FUTURE WORKS

This paper proposes a fusion between TT, FL, and DT to generate optimized membership functions and rules for FL. This combination shows promising results for the multi-classifications domain. The TT is the key in the ITTDTFL model, it generates accurate and optimized membership functions and rules. The ITTDTFL model has successfully outperformed the most known multi-classification models, such as FURIA, RIPPER, C4.5, and DTFL. A notable advantage of integrating the TT into this process is the significant rule number reduction by 86.87%. This fusion played a significant role in improving the optimized rules' generation and enhancing their precision. Which in turn leads to achieving impeccable accuracies in data classification as well as in computational time. ITTDTFL has successfully reduced the computational time for the DTFL model by 92.87%, enhancing its accuracy by 4.55%. At the same time, passing the other models, FURIA, RIPPER, and C4.5, successively by 6.92%, 7.5%, and 7.32%. Real machine fault datasets were used for the evaluation. It has seven classes and Two complicated attributes (velocity and acceleration spectrums), noting that having more attributes would enhance the precision of the rules and, consequently, the model. This model offers promising potential for delivering accurate results in real-time. Demonstrating its versatility, this model is highly interpretable and can be applied to various classification issues beyond machine condition diagnosis. Thus, the next step is to apply this model to the datasets used in the literature, such as UCI and Statlib repositories, as well as investigate the integration of multi-objective optimization using Evolutionary algorithms, such as genetic algorithms, which will certainly enhance the model's capabilities of accurately classifying the data as well as its computational time requirements.

REFERENCES

Al-Shammaa, M. and Abbod, M. F. (2014). Automatic generation of fuzzy classification rules from data. In *Proc. of the 2014 International Conference on Neural Networks-Fuzzy Systems (NN-FS 14), Venice*.

Angelov, P. P. and Buswell, R. A. (2003). Automatic generation of fuzzy rule-based models from data by genetic algorithms. *Information Sciences*, 150(1-2):17–31.

Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106:1039–1082.

Chi, Z., Yan, H., and Pham, T. (1996). *Fuzzy algorithms: with applications to image processing and pattern recognition*, volume 10. World Scientific.

Chiu, S. (1997). Extracting fuzzy rules from data for function approximation and pattern classification.

Cintra, M. E., Monard, M. C., and Camargo, H. A. (2013). A fuzzy decision tree algorithm based on c4. 5. *Mathware & Soft Computing*, 20(1):56–62.

Cohen, W. W. (1995). Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier.

Durkin, J. (1990). Research review: Application of expert systems in the sciences. *The Ohio Journal of Science*, 90(5):171–179.

Elbaz, K., Shen, S.-L., Zhou, A., Yuan, D.-J., and Xu, Y.-S. (2019). Optimization of epb shield performance with adaptive neuro-fuzzy inference system and genetic algorithm. *Applied Sciences*, 9(4):780.

Gómez-Skarmeta, A. F., Delgado, M., and Vila, M. A. (1999). About the use of fuzzy clustering techniques for fuzzy model identification. *Fuzzy sets and systems*, 106(2):179–188.

Gonzblez, A. and Pérez, R. (1999). Slave: A genetic learning system based on an iterative approach. *IEEE Transactions on fuzzy systems*, 7(2):176–191.

Hambali, A., Yakub, K., Oladele, T. O., and Gbolagade, M. D. (2019). Adaboost ensemble algorithms for breast cancer classification. *Journal of Advances in Computer Research*, 10(2):31–52.

Hentout, A., Maoudj, A., and Aouache, M. (2023). A review of the literature on fuzzy-logic approaches for collision-free path planning of manipulator robots. *Artificial Intelligence Review*, 56(4):3369–3444.

Hssina, B., Merbouha, A., Ezzikouri, H., and Erritali, M. (2014). A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19.

Hühn, J. and Hüllermeier, E. (2009). Furia: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19:293–319.

Hühn, J. C. and Hüllermeier, E. (2010). An analysis of the furia algorithm for fuzzy rule induction. In *Advances in Machine Learning I: Dedicated to the Memory of Professor Ryszard S. Michalski*, pages 321–344. Springer.

Hüllermeier, E. (2011). Fuzzy sets in machine learning and data mining. *Applied Soft Computing*, 11(2):1493–1505.

Kerarmi, A., Kamal-idrissi, A., Seghrouchni, A. E. F., et al. (2022). An optimized fuzzy logic model for proactive maintenance. In *CS & IT Conference Proceedings*, volume 12. CS & IT Conference Proceedings.

Kontogiannis, D., Bargiotas, D., and Daskalopulu, A. (2021). Fuzzy control system for smart energy management in residential buildings based on environmental data. *Energies*, 14(3):752.

Mousavi, S. M., Tavana, M., Alikar, N., and Zandieh, M. (2019). A tuned hybrid intelligent fruit fly optimization algorithm for fuzzy rule generation and classification. *Neural Computing and Applications*, 31:873–885.

Mutlu, B., Sezer, E. A., and Akcayol, M. A. (2018). Automatic rule generation of fuzzy systems: A comparative assessment on software defect prediction. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 209–214.

Prado, R., Garcia-Galán, S., Exposito, J. M., and Yuste, A. J. (2010). Knowledge acquisition in fuzzy-rule-based systems with particle-swarm optimization. *IEEE Transactions on Fuzzy Systems*, 18(6):1083–1097.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Reddy, G. T., Reddy, M. P. K., Lakshmanna, K., Rajput, D. S., Kaluri, R., and Srivastava, G. (2020). Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis. *Evolutionary Intelligence*, 13:185–196.

Ren, Q., zhang, H., Zhang, D., Zhao, X., Yan, L., and Rui, J. (2022). A novel hybrid method of lithology identification based on k-means++ algorithm and fuzzy decision tree. *Journal of Petroleum Science and Engineering*, 208:109681.

Rzeszucinski, P., Sinha, J. K., Edwards, R., Starr, A., and Allen, B. (2012). Normalised root mean square and amplitude of sidebands of vibration response as tools for gearbox diagnosis. *Strain*, 48(6):445–452.

Tran, T. T., Nguyen, T. N., Nguyen, T. T., Nguyen, G. L., and Truong, C. N. (2022). A fuzzy association rules mining algorithm with fuzzy partitioning optimization for intelligent decision systems. *International Journal of Fuzzy Systems*, 24(5):2617–2630.

Varshney, A. K. and Torra, V. (2023). Literature review of the recent trends and applications in various fuzzy rule-based systems. *International Journal of Fuzzy Systems*, pages 1–24.

Witten, I. H., Frank, E., Hall, M. A., Pal, C. J., and Data, M. (2005). Practical machine learning tools and techniques. In *Data mining*, volume 2, pages 403–413. Elsevier Amsterdam, The Netherlands.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.