GAbDT - Genetic Algorithm based Decision Tree

Gabriel Branco Vitorino

October 2025

Resumo

In this work, we developed a Decision Tree whose construction is guided by a Genetic Algorithm. The Genetic Algorithm is responsible for selecting the most suitable premise for splitting each node of the tree, aiming to minimize the majority error. This approach replaces the traditional exhaustive search, which evaluates all possible splits based on the average between consecutive attribute values, thus enabling the exploration of alternative ways to partition the data, thereby helping to avoid local minima. The algorithm's performance was compared with that of the conventional Decision Tree implementation available in the Scikit-Learn library across different datasets, using a Student's t-test for statistical evaluation.

1 Introduction

A decision tree is a machine learning model that organizes data into a hierarchical structure, where each split of the dataset is guided by decision rules. This structure consists of internal nodes, which represent questions formulated based on data attributes, and leaf nodes, which contain the model's final predictions. The purpose of the tree is to partition the data iteratively, creating increasingly homogeneous subsets with respect to the target variable. This process resembles a sequence of "yes or no" questions, where each answer directs the sample to a specific branch of the tree until a classification is reached.

To determine the best splits, the algorithm examines potential cut points for each attribute, evaluating how many divisions to test according to criteria such as the number of unique values of the variable or the nature of the data (continuous or categorical). For continuous attributes, it considers multiple thresholds as potential split points, while for categorical variables it may test different category groupings. Each split is evaluated according to an impurity measure, such as the Gini index or entropy, and the one that yields the greatest reduction in impurity is selected. This process is repeated recursively until a stopping criterion is met, such as a minimum number of samples per node, a maximum tree depth, or the absence of significant improvements in data separation.

The Genetic Algorithm based Decision Tree (GAbDT) has the potential to avoid local minima, whereas the traditional decision tree algorithm always converges to the same solution, which may correspond either to a global or a local minimum. As a result, in some executions, our algorithm is able to achieve better performance than the the standard approach.

The proposed method is a Decision Tree that employs a Genetic Algorithm to select the best partitions during tree construction. The Genetic Algorithm is used to minimize a fitness function that computes the majority error—or another desired evaluation metric—produced by a given split, thereby attempting to identify the most suitable premise for recursively dividing the instances until the decision tree is fully formed. It is worth noting that the fitness function can be designed to maximize or minimize different metrics, serving as a measure of the impurity of the resulting subsets.

In this approach, each individual in the Genetic Algorithm corresponds to a premise. A premise is defined as a set consisting of an attribute, an operator, and a threshold (['att', 'op', 'ths']), which is applied to partition the training set at each node. The algorithm starts with a population of n premises and evolves across generations by selecting the rules that minimize the majority error. To this end, each premise is applied, the majority-class error is calculated for both resulting groups, and the weighted sum of these errors—proportional to the group sizes—yields the fitness of each solution.

The lower the fitness value, the better the premise partitions the training set, producing more homogeneous subsets. Subsequent generations within the same node are formed by retaining the best individual found so far and by generating offspring through crossover and mutation applied to parents selected by three-way tournament selection. In the final generation, the best individual is chosen as the premise to split the current dataset. The same

procedure is applied recursively to the subsequent nodes until the groups become sufficiently homogeneous (majority error minimized) and the algorithm reaches a leaf node.

2 Methodology

In this section, we provide a detailed explanation of the implemented algorithm, highlighting the internal mechanisms of its learning process.

GAbDT can be divided into three groups of functions. The first group consists of functions responsible for generating the initial population of premises, applying premises to the training dataset, and calculating the fitness function using the majority error resulting from the split, handled by the function `evaluate()`. The main motivation is to enable a broader exploration of possible data split points, rather than relying solely on the mean of consecutive values as the cut-off point, as well as the sequence of data partitions. This is important because a decision tree, by always selecting the best premise at each node, may fail to discover different combinations of splits that could ultimately result in higher accuracy.

The function `criar_individuo(atributos)` generates an individual that is, a splitting premise, randomly. Each premise consists of an attribute selected from the training dataset, a logical operator (e.g., `<`, `>=`, etc.), and a numerical threshold randomly chosen between the minimum and maximum values of that attribute. This approach provides greater flexibility and variability in the creation of decision rules. Based on this function, an initial population of n individuals is generated, each representing a potential premise for partitioning the dataset.

The function `aplicar_premissa(df, premissa)` applies a specific premise to the dataset, producing two subsets: `df_verdadeiro`, containing the instances that satisfy the premise, and `df_falso`, containing the remaining instances. This division allows for evaluating the impact of the premise on class separation.

To quantify the quality of each split, the function `erro_majoritario(df, coluna_classe)` calculates the classification error assuming that all instances belong to the majority class of that subset. Thus, the smaller the error, the more homogeneous the subset is with respect to the target variable, which indicates a good partition.

Finally, the function `evaluate(df, populacao)` applies each premise

from the population to the dataset, computes the majority errors of the resulting subsets, and returns the fitness value of each individual. The total error of a premise is computed as the weighted average of the subset errors, with weights corresponding to the relative size of each subset compared to the original dataset. This function serves as the foundation for guiding the evolutionary process of premises across generations.

The second group of functions implements the genetic operators responsible for evolving the population of premises over generations. These operators aim to search, through mechanisms inspired by natural selection, for premises that produce more effective data partitions, progressively reducing the previously defined fitness function.

The function `torneio(pop_avaliada, k=3)` is used as the selection method. It randomly chooses k individuals from the evaluated population and returns the one with the lowest fitness value. This process ensures moderate selective pressure, favoring better individuals while still maintaining genetic diversity.

To generate new individuals (offspring), the function `crossover_rnd(pai1, pai2)` performs crossover between two selected premises. Each component of the new premise (attribute, operator, and threshold) is inherited randomly from one of the parents. This mechanism enables the recombination of beneficial characteristics from parental premises, promoting the exploration of the feature space.

Additional diversity is introduced by the function `mutacao()`, which performs controlled mutation of each component of a premise with distinct probabilities. Mutation may alter the attribute used, the logical operator, or the threshold value, with the new values being chosen based on the observed ranges of the dataset. The function `calcular_intervalo_atributos(df, atributos)` calculates these ranges for each numerical attribute, serving as the basis for mutation.

The central function in this group is `nova_geracao()`, which implements the complete cycle for creating a new generation of the population. The process begins with elitism, which ensures that the best individual of the current generation is preserved in the new population. The remaining individuals are generated from parents selected via tournament, followed by crossover and mutation. The population size is kept constant, equal to that of the initial population, which represents a computational cost that can be fine-tuned in future adaptations.

This iterative cycle progressively refines the quality of the premises across generations, driving the population toward solutions that produce better

partitions of the dataset.

The final group of functions is associated with the construction of the decision tree. This group includes a function that combines all the previously defined components to recursively build the final decision tree, evolving a solution for each subgroup generated—that is, for each node of the tree. In addition, there is also a function that generates a visual representation of the decision tree, as well as a function that uses the evolved premises to make predictions on a test set.

The function `evoluir_solucoes()` performs the recursive construction of the tree. At each call, it evaluates whether the current node should be terminal (i.e., a leaf) based on two criteria: the maximum tree depth or the minimum number of samples per node. Otherwise, a population of premises is initialized and evolved for a fixed number of generations using the genetic operators described earlier. The fitness function guides this process by seeking to minimize the majority classification error.

At the end of evolution, the best premise found is used to split the dataset into two subsets. Each subset is then evaluated according to its majority error. If this error is below a predefined threshold (`erro_threshold`), the corresponding branch becomes a leaf; otherwise, the evolutionary process continues recursively for that branch.

The output of the function is a tree represented by nested dictionaries. Each node stores whether it is a leaf, the applied premise (if any), the class counts before and after the split, and the resulting left and right branches. This representation makes the model interpretable and easily accessible for inspection.

The function `predict_custom_tree()` enables predictions with the constructed tree. For each instance, the function traverses the tree recursively until reaching a leaf, applying the premise stored at each node. The class assigned to the instance is then the most frequent one at the reached leaf node. Support for multiple relational operators (`<`, `<=`, `>`, `>=`) ensures flexibility in defining decision rules.

In summary, this function organizes the evolutionary process into a hierarchical structure similar to a traditional decision tree, but where the splitting rules are optimized by a genetic algorithm. The result is a non-conventional decision tree, in which partitions are obtained through evolutionary optimization rather than greedy algorithms such as ID3 or CART. Consequently, the model can capture decision rules that are more suitable to the data, even when such rules are not evident through direct deterministic analysis.

2.1 Tests and Applications

During the development of the algorithm, it was applied to well-known and extensively studied datasets in order to compare the performance of our version with the original algorithms. This process was crucial for the implementation and refinement of the adaptations made to GAbDT. The datasets were obtained from the *MLBench* package, which contains a collection of machine learning benchmarking problems, both artificial and real-world, including, for example, several datasets from the UCI (*University of California, Irvine*) repository.

The developed algorithm was then subjected to a more rigorous evaluation through statistical tests applied to standardized datasets widely used in the literature, namely: Waveform 5000 [1], Ionosphere [2], and Diabetes [3]. These applications aim to validate the effectiveness of the algorithms in more relevant and challenging real-world contexts, assessing whether they indeed show improved performance compared to traditional approaches.

The Waveform 5000 dataset is characterized by a simulated multiclass classification problem, consisting of 5,000 samples and multiple continuous variables, presenting a moderate level of noise and complexity [1]. The Ionosphere dataset, in turn, contains radar-collected data intended to distinguish between echoes from metallic and non-metallic objects, comprising 351 samples and 34 attributes [2]. The Diabetes dataset corresponds to a real-world binary classification problem, with 768 samples and 8 predictor variables related to clinical factors for detecting diabetes mellitus [3].

Using these datasets allowed for a comparative analysis of the performance of the developed algorithms across diverse contexts, ranging from noisy artificial data (Waveform 5000) to real biomedical data (Diabetes) and radar data with specific characteristics (Ionosphere). After the initial validation, the algorithms were then applied to a final dataset of personal interest, using Heart Rate Variability (HRV) data. This application aimed not only to verify the effectiveness of the methods in a real-world scenario with greater clinical relevance and complexity but also to detect indications of significant performance improvement relative to traditional methods.

Model training on the HRV data was performed using all attributes highlighted in Table 1 for each training example [4]. The dataset was part of the phd thesis of Santos, Rafael Rodrigues and it is not available publicly.

Our analysis was conducted using a two-class division: Apneic and Non-Apneic, as an alternative to the table shown. To this end, an AHI threshold

Tabela 1: Anthropometric and PSG characteristics by severity class of OSA. Values are n (%) for males and median for the remaining variables [4].

	Non-Apneic (n=47)	**Mild (n=63)**	**Moderate (n=70)**	**Severe (n=111)**
MALES	12 (25.5%)	27 (42.9%)	30 (42.9%)	49 (44.1%)
AGE (years)	41	52	56.5	56
HEIGHT (m)	1.62	1.66	1.65	1.63
WEIGHT (kg)	71.5	82	85.5	98
BMI	26.7	31.04	31.01	34.72
AHI	2.6	10.0	21.1	57.7
T90	0.07	1.10	2.75	19.80
MinSat (%)	88	86	83	74

of 15 was established to separate the two classes, grouping Non-Apneic and Mild cases into one class and Moderate and Severe cases into the other [4]. This approach was chosen due to prior knowledge of the complexity of the four-class problem, which typically requires more complex algorithms such as Random Forests and deep Neural Networks. The binary problem is more suitable given the complexity of the KNN and decision tree algorithms used.

The data were then split into training and test sets, which were provided both to the original algorithms without modifications and to the modified versions (GAbDT and DTKNN), allowing for the evaluation of performance gains in the new versions. Paired Student's t-tests with 10- or 20-fold cross-validation were performed for both algorithms to assess the statistical significance of the observed performance improvements. In all tests, we ensured that the hyperparameters used in both the original and modified versions were identical.

3 Results

In the simpler datasets used during development, statistical tests did not reveal a significant difference between the traditional algorithm and the version implemented in this work. Thus, we can conclude that, for more trivial problems, the developed algorithm perform as well as the traditional Python implementations. This outcome was expected, and this phase was mainly

important for properly tuning the algorithms during the development stage.

The results obtained on the Waveform 5000 [1], Ionosphere [2], and Diabetes [3] datasets are more encouraging and are discussed below. Table 2 presents the results for the GAbDT algorithm on these three datasets. For each dataset, the average accuracy achieved by the standard decision tree and the custom algorithm is reported. The last two columns show the Student's t statistic and the corresponding p-value from paired t-tests using 10-fold cross-validation.

From Table 2, it can be observed that, although GAbDT achieved slightly higher average accuracies on all three datasets, none of the differences were statistically significant at the 95% confidence level according to the p-values obtained from the paired t-tests. That is, from a statistical perspective, it cannot be claimed with certainty that the developed algorithm outperforms the traditional versions in these scenarios, even though it performed better in most folds.

Tabela 2: Comparative results between Scikit-Learn and GAbDT on the datasets used

Dataset	**Average Accuracy Scikit-Learn**	**Average Accuracy GAbDT**	**Student's t**	**p-value**
Diabetes	70.6%	74.1%	1.728	0.118
Ionosphere	86.8%	88.9%	1.369	0.193
Waveform 5000	73.7%	76.1%	3.723	0.047

It is important to highlight that the results indicate a trend of improvement in average accuracies, particularly for the Waveform 5000 dataset, where GAbDT achieved an average accuracy of 76.1% compared to 73.7% for the standard tree, with a p-value of 0.047 representing a statistically significant difference. In the Diabetes dataset, the 3.5 percentage point difference in average accuracy is notable, even though the p-value of 0.118 does not indicate formal statistical significance.

These results reinforce that GAbDT is competitive and shows potential to outperform the traditional algorithm in specific contexts. Considering that GAbDT was developed with a focus on customization and incorporating

evolutionary techniques, its promising performance on more complex datasets indicates that the approach, with some modifications, could achieve even better results. Future investigations will focus on exploring hyperparameter tuning using a validation stage, variations in the representation of individuals in the genetic population, or even different splitting metrics in the tree to further enhance model performance.

We will not discuss execution time in depth, but it ranges from approximately 1 to 5 minutes depending on the dataset and pruning parameters defined for tree construction. This is considerably longer than the induction time of a standard decision tree; however, this delay can be offset by a significant accuracy gain on some datasets, as will be shown later in the HRV results analysis.

After implementation and testing, the algorithm was applied to the HRV dataset. To compare the performance of the developed algorithm with the Python standard, a paired Student's t-test was performed using 20-fold cross-validation. Table 3 shows the results for GAbDT, with an average accuracy of 71.55% for GAbDT versus 62.45% for the Scikit-Learn model. The t-statistic was 3.18 and the p-value 0.004, indicating statistical significance.

The paired Student's t-test was repeated four additional times to confirm the result, and all executions yielded statistically significant differences at a confidence level of at least 95%. The pre-pruning parameters, such as the minimum number of samples per leaf and the maximum depth, were the same for both algorithms (10 and 15, respectively). Additionally, Appendix 1 provides a comparison between a tree generated by Scikit-Learn and one generated by GAbDT for this task.

As observed, GAbDT performs significantly better than the standard algorithm in detecting OSA, although this difference is less evident in simpler, artificially distributed datasets, where both algorithms exhibit statistically similar accuracies. This is likely due to differences in data complexity.

A conventional decision tree always generates the same rules in the same order for a fixed dataset, i.e., it always finds the best split at the first node, the best split at the second node, and so on. In contrast, GAbDT allows exploration of different decision trees and, importantly, different rule orderings. In short, while the rule chosen by a traditional decision tree is always optimal for a given node, a sub-optimal rule, which would never be selected by a standard tree, combined with other rules in subsequent or previous nodes may result in a more optimized split, achieving higher accuracy in the end. GAbDT's ability to explore different tree configurations within the same

Tabela 3: Comparative statistical results between GAbDT and the Scikit-Learn decision tree. Hyperparameters used were *max_ depth* equal to 15 and *min_ samples_ split* equal to 10. All other parameters were kept at the library's default values.

	GAbDT	**Scikit-Learn**
Average Accuracy	70.8%	63.6%
Paired Student's t-test $t = 3.178$, $p = 0.004$		

training set is particularly valuable for ensemble methods, such as random forests, where multiple trees can be generated and majority voting applied to classify a new instance.

Moreover, to maintain model interpretability, a large and representative validation set could be used to select the best tree generated by the genetic algorithm over, for example, 10 iterations, and then test it on the test set. These applications appear promising and will be the focus of future studies.

4 Conclusion

The implementation of the Genetic Algorithm-based Decision Tree (GAbDT) proved to be a promising approach, particularly in the main application to Heart Rate Variability (HRV) data for the detection of Obstructive Sleep Apnea (OSA). Although the results obtained on benchmark datasets showed improvements that were not statistically significant in most cases, albeit quite expressive, for the real-world problem of classifying patients with OSA and the waveform dataset, GAbDT achieved significantly higher performance than the traditional decision tree, with an average accuracy of 70.8% and 76.9% respectively compared to 63.6% and 73.7%, both with p-values smaller than 0.05, such as 0.004 and 0.005 in the Student's t-test.

An accuracy of 70.8% is quite remarkable for this dataset, considering that the best result obtained using cross-validation with a random forest and data balancing by [4] was 76.0%. This demonstrates that the algorithm's ability to explore different sequences of rules, often not considered by traditional approaches, can be advantageous in scenarios of higher complexity and

variability, as observed in clinical data.

In the future, ensemble strategies could be particularly interesting to explore for improving the algorithm's performance, given that GAbDT naturally generates highly varied decision trees in each run, but this could potentially just average out the performance of a traditional tree. Additionally, further adjustments to the genetic algorithm could be implemented to ensure that its execution time is more concise and competitive with a standard decision tree, whose execution is very fast. Methods to enhance the GA's performance itself could also be applied, such as techniques to avoid premature convergence like fitness sharing, the periodic introduction of random individuals into the population, or modifications to the mutation and crossover functions to better meet the problem's requirements.

Referências

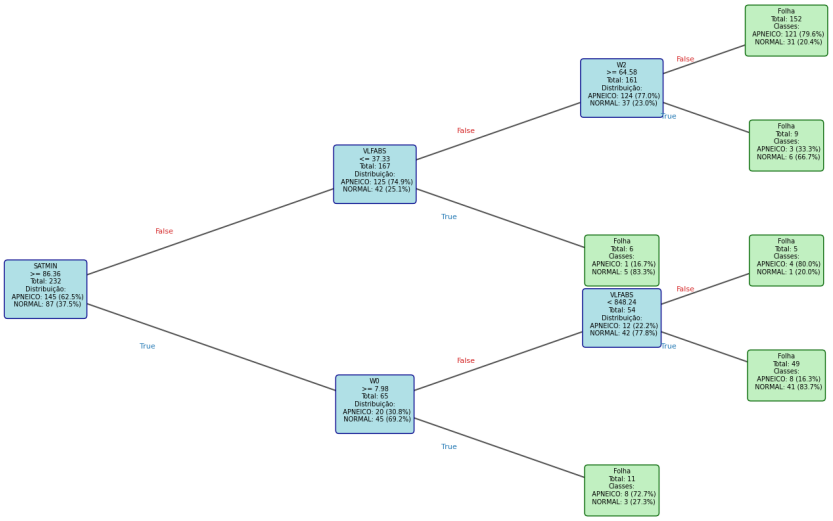
[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Waveform database generator (version 1.0), 1984. UCI Machine Learning Repository.

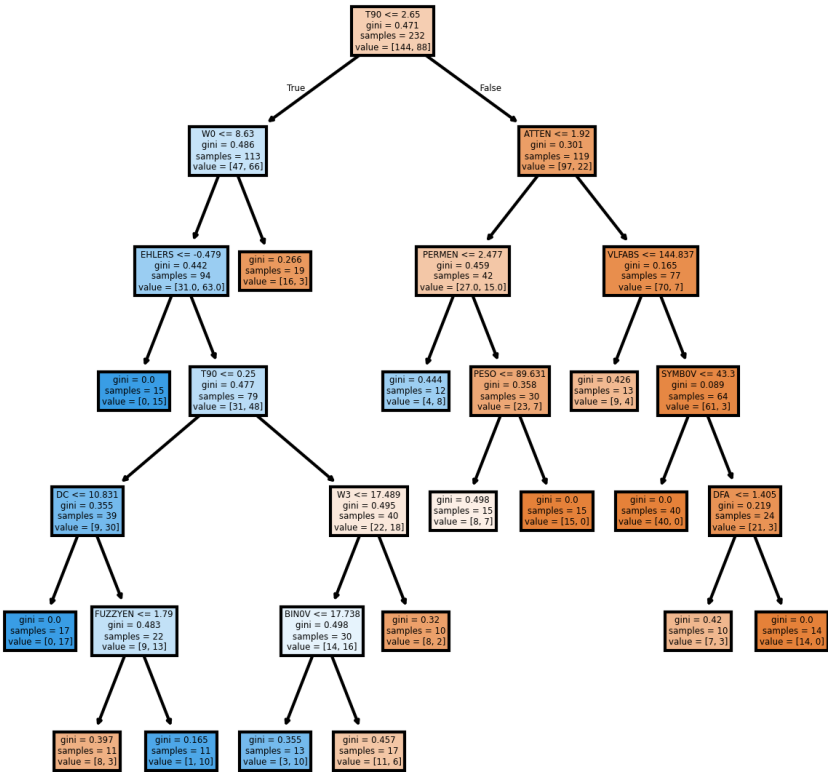
[2] V.G. Sigillito, S.P. Wing, L.V. Hutton, and K.P. Baker. Ionosphere data set, 1989. UCI Machine Learning Repository.

[3] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes. Pima indians diabetes database, 1988. UCI Machine Learning Repository.

[4] Rafael Rodrigues dos Santos, Matheo Bellini Marumo, Alan Luiz Eckeli, Helio Cesar Salgado, Luiz Eduardo Virgílio Silva, Renato Tinós, and Rubens Fazan. The use of heart rate variability, oxygen saturation, and anthropometric data with machine learning to predict the presence and severity of obstructive sleep apnea. *Frontiers in Cardiovascular Medicine*, 12, 2025.

A Apêndice 1


SATMIN
>= 86.36
Total: 232
Distribuição:
APNEICO: 145 (62.5%)
NORMAL: 87 (37.5%)
False
True
VLFABS
<= 37.33
Total: 167
Distribuição:
APNEICO: 125 (74.9%)
NORMAL: 42 (25.1%)
false
True
W0
>= 7.98
Total: 65
Distribuição:
APNEICO: 20 (30.8%)
NORMAL: 45 (69.2%)
false
True
W0
>= 64.58
Total: 161
Distribuição:
APNEICO: 124 (77.0%)
NORMAL: 37 (23.0%)
False
True
Folha
Total: 152
Classes:
APNEICO: 121 (79.6%)
NORMAL: 31 (20.4%)
Folha
Total: 9
Classes:
APNEICO: 3 (33.3%)
NORMAL: 6 (66.7%)
Folha
Total: 6
Classes:
APNEICO: 1 (16.7%)
NORMAL: 5 (83.3%)
VLFABS
< 848.24
Total: 54
Distribuição:
APNEICO: 12 (22.2%)
NORMAL: 42 (77.8%)
False
True
Folha
Total: 5
Classes:
APNEICO: 4 (80.0%)
NORMAL: 1 (20.0%)
Folha
Total: 49
Classes:
APNEICO: 8 (16.3%)
NORMAL: 41 (83.7%)
Folha
Total: 11
Classes:
APNEICO: 8 (72.7%)
NORMAL: 3 (27.3%)


T90 <= 2.65
gini = 0.471
samples = 232
value = [144, 88]
True
False
W0 <= 8.63
gini = 0.486
samples = 113
value = [47, 66]
ATTEN <= 1.92
gini = 0.301
samples = 119
value = [97, 22]
EHLERS <= -0.479
gini = 0.442
samples = 94
value = [31.0, 63.0]
gini = 0.266
samples = 19
value = [16, 3]
PERMEN <= 2.477
gini = 0.459
samples = 42
value = [27.0, 15.0]
VLFABS <= 144.837
gini = 0.165
samples = 77
value = [70, 7]
gini = 0.0
samples = 15
value = [0, 15]
T90 <= 0.25
gini = 0.477
samples = 79
value = [31, 48]
gini = 0.444
samples = 12
value = [4, 8]
PESO <= 89.631
gini = 0.358
samples = 30
value = [23, 7]
gini = 0.426
samples = 13
value = [9, 4]
SYMB0V <= 43.3
gini = 0.089
samples = 64
value = [61, 3]
DC <= 10.831
gini = 0.355
samples = 39
value = [9, 30]
W3 <= 17.489
gini = 0.495
samples = 40
value = [22, 18]
gini = 0.498
samples = 15
value = [8, 7]
gini = 0.0
samples = 15
value = [15, 0]
gini = 0.0
samples = 40
value = [40, 0]
DFA <= 1.405
gini = 0.219
samples = 24
value = [21, 3]
gini = 0.0
samples = 17
value = [0, 17]
FUZZYEN <= 1.79
gini = 0.483
samples = 22
value = [9, 13]
BIN0V <= 17.738
gini = 0.498
samples = 30
value = [14, 16]
gini = 0.32
samples = 10
value = [8, 2]
gini = 0.42
samples = 10
value = [7, 3]
gini = 0.0
samples = 14
value = [14, 0]
gini = 0.397
samples = 11
value = [8, 3]
gini = 0.165
samples = 11
value = [1, 10]
gini = 0.355
samples = 13
value = [3, 10]
gini = 0.457
samples = 17
value = [11, 6]