Computing (2025) 107:42
https://doi.org/10.1007/s00607-024-01394-8

REGULAR PAPER

A novel ensemble learning method using majority based voting of multiple selective decision trees

Mohammad Azad[1] · Tasnemul Hasan Nehal[2] · Mikhail Moshkov[3]

Received: 8 June 2024 / Accepted: 11 December 2024 / Published online: 31 December 2024

Abstract
Traditional decision tree algorithms are susceptible to bias when certain classes dominate the dataset and prone to overfitting, particularly if they are not pruned. Previous studies have shown that combining several models can mitigate these issues by improving predictive accuracy and robustness. In this study, we propose a novel approach to address these challenges by constructing multiple selective decision trees using the entirety of the input dataset and employing a majority voting scheme for output forecasting. Our method outperforms competing algorithms, including KNN, Decision Trees, Random Forest, Bagging, XGB, Gradient Boost, and ExtraTrees, achieving superior accuracy in five out of ten datasets. This practical exploration highlights the effectiveness of our approach in enhancing decision tree performance across diverse datasets.

Keywords Decision tree · Multiple tree · Ensembles · Majority vote · Machine learning

Mathematics Subject Classification 68T05 · 05C05 · 62H30 · 68T30

✉ Mohammad Azad
mmazad@ju.edu.sa

Tasnemul Hasan Nehal
g202318850@kfupm.edu.sa

Mikhail Moshkov
mikhail.moshkov@kaust.edu.sa

1 Department of Computer Science, College of Computer and Information Sciences, Jouf University, 72441 Sakaka, Saudi Arabia

2 Department of Control and Instrumentation Engineering, System and Control Engineering, King Fahd University of Petroleum and Minerals, 31261 Dhahran, Saudi Arabia

3 Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology (KAUST), 23955-6900 Thuwal, Saudi Arabia

1 Introduction

Decision trees are powerful tools in machine learning, prized for their simplicity and interpretability. However, they come with inherent limitations. A single decision tree algorithm may produce a tree that is overly complex and fails to comprehensively represent the input data. Even minor changes in the input can lead to vastly different trees, rendering decision trees inherently fragile. Consequently, individual decision trees often capture noise in the training data, resulting in poor generalization performance.

To address these shortcomings, ensemble methods have emerged as a popular approach. By aggregating predictions from multiple trees, ensemble methods create more robust and generalizable models. These methods leverage the principle that combining the predictions of multiple weak learners (individual trees) can capture complex relationships in the data more effectively, leading to better performance on unseen data.

Moreover, traditional decision trees may struggle with imbalanced datasets, particularly in accurately classifying minority classes. Ensemble methods offer a solution by weighting or resampling the data to give more importance to minority classes, thereby improving overall prediction accuracy.

One of the key advantages of ensemble methods is their inherent feature selection capability. By evaluating the importance of each feature across multiple trees, ensemble methods naturally perform feature selection, enhancing model interpretability and efficiency. Additionally, ensemble methods are flexible and can be adapted to various problem types and datasets.

In high-performance computing environments and distributed computing frameworks, ensemble methods shine. Their parallelizability and scalability make them well-suited for handling large-scale data and complex modeling tasks.

In this study, we present a novel ensemble algorithm where, we generate multiple decision trees followed by two steps of selection of trees. Finally, we use majority voting technique which is an ensemble technique where predictions are made by combining the results of various models. Our method is not hampered by significant errors or misclassifications from a single model because it depends on the performance of multiple models. This method makes the estimator more susceptible to overfitting and more robust. It is now well acknowledged that using a combination of classifiers created for a specific prediction problem yields better prediction rates than using each classifier separately [1, 2].

In our method, we consider a decision table D containing features f_1, f_2, ..., f_n and each feature column has several unique attributes $a_{i_1}, a_{i_2}, ..., a_{i_n}$. In the first stage of this algorithm, we create the base binary decision tree for this table using the standard approach like CART [3]. Then we generate multiple binary decision trees by changing the node questions of each tree at every level. The node questions were selected by calculating the information gain based on Gini index for each "feature=attribute" ($f_1 = a_{i_1}$) pair of the decision table, ignoring those that have zero gains. We sort the question pairs from maximum information gain to a minimum. In this way, a noble multiple decision tree generation technique was developed which maximizes the computing efficiency in terms of generating trees from best performance to less. In the second stage, we limit the number of decision trees to L which is a user-defined parameter. After that, we filter some trees in order to get K number (where $K <= L$) of best decision trees in terms of accuracy on the validation data set. In the third and final stage, the chosen trees were then used to find the prediction accuracy on the testing data set using a majority voting scheme.

The rest of the paper is structured as follows: Sect. 2 covers the most recent related research in this field. Section 3 provides a description of the suggested approach for our multi-tree algorithm. In Sect. 4, experimental results comparing the proposed method to state-of-the-art approaches are presented. The suggested study is concluded in Sect. 5, which also makes recommendations for future improvements.

2 Related work

Decision trees serve as fundamental tools in machine learning for both classification and regression tasks due to their simplicity, interpretability, and ability to handle nonlinear relationships within data. Seminal decision tree algorithms such as C4.5 [4], CART [5], ID3 [6] have laid the foundation for subsequent research in the field. C4.5 introduced several innovations, including the ability to handle categorical variables, missing values, and pruning techniques to avoid overfitting. CART, on the other hand, focused on binary splits and recursive partitioning to create decision trees, contributing to the development of subsequent algorithms.

Traditional decision tree algorithms are not without their limitations. One notable issue is their susceptibility to bias when certain classes are dominant in the dataset, leading to imbalanced trees and suboptimal predictive performance [7]. Moreover, unpruned decision trees tend to overfit the training data, resulting in poor generalization to unseen instances [3].

In the paper [8], the author presents a similar idea to generate multiple decision trees to change the node questions. However, the author produced only a few number of decision trees on top levels whereas our method can produce a large number of different decision trees. Furthermore, the author only compared their approach for two data sets whereas we compared 10 different data sets. In another paper [9], the author suggests an approach based on multi-criteria decision analysis (MCDA) to assist the DM project team in choosing the best suitable DT. However, the author uses some commercial software to create multiple decision trees and uses different performance criteria to measure and choose among decision trees. The number of created decision trees were small in number. There is another paper [10], where the authors presented meta decision tree algorithm by combining multiple models. However, this paper does not create multiple decision trees based on the data sets rather it creates a small decision tree by combining multiple classifier models.

The authors [11, 12] use "Maximally Diversified Multiple Decision Tree Algorithm (MDMT)" to construct numerous trees where an attribute is examined in a maximum of one tree. Every tree evaluates an entirely distinct collection of qualities compared to the collection of attributes examined in every other tree. Using a conventional technique such as C4.5, MDMT creates the first decision tree on a given data set. After removing all non-class attributes that were examined in the first tree from the data set, C4.5 is once more applied to the updated data set in order to construct the second decision tree. The procedure keeps on until the user-specified number of trees is produced or all of the data set's nonclass attributes are eliminated. We contend that MDMT might not be able to construct a sufficient number of trees for a low dimensional data set. The method might be able to create a large number of decision trees for a high dimensional dataset. But since useful attributes are removed from the data set each time a tree is formed, the quality of the later trees might not be particularly good.

The authors [13] use "Cascading and Sharing Trees (CS4)" takes the number of trees to be generated as a user input. The qualities are then arranged by gain ratios in CS4. The ith tree's root attribute is regarded as the ith best attribute. We contend that the amount of non-class attributes in a data set is the maximum number of trees that CS4 may construct. It is therefore unable to construct a large number of trees for a low dimensional data collection.

In a different study [14], a technique similar to ours was employed to create multiple decision trees. However, differences exist in the splitting mechanism and attribute selection criteria. The differences are: first, we use binary split whereas they use C4.5's splitting mechanism; second the authors considers mainly good attributes based on some goodness crtieria whereas we considers all possible attributes and corresponding splits till we create required number of trees and finally we select and filters trees based on the accuracy performance on the validation data set whereas they didn't use such steps.

In the paper [15], the authors study decisions trees for decision tables with multiple decisions using dynamic programming algorithm for bi-objective optimization and consider three methods to design decision trees and evaluate the number of nodes, and local and global misclassification rates of constructed trees. However, this paper does not consider accuracy of prediction for the testing data set, thus it is far from our research objectives.

In addition, there exist other studies that touch on a different aspect of the general topic of merging various models, but are not directly linked to our research goal. To complete the picture, we mentioned them below.

A group of researchers recently presented the idea of "Ant Colony Optimization" (ACO) as a way to acquire an extended multi-tiered classification model composed of several decision trees (one for every class value). The proposed ACO-based algorithms, Ant-Tree-Mer-Miner ML and Ant-Tree-Mer MI, showed an overall predictive accuracy improvement over 32 benchmark datasets compared to very well-known decision tree induction algorithms (C4.5, CART, etc.), including the original ant-Tree-Mer algorithm [16]. Another group of academics utilized a multi-tree genetic programming method (MTGP) to improve the imputation of missing values in the target domain and to improve symbolic regression performance on incomplete data. Using the MTGP approach to map the source domain close to the target domain, the researchers created many new features from the source domain. This enhanced the performance of symbolic regression and enhanced the imputation of missing data in the target domain [17]. Another research team used an alternative genetic programming strategy, where the automated production of redundant characteristics was the main focus. A multi-tree genetic programming representation is used in the proposed GPRFC (Genetic Programming for Redundant Feature Creation), where each tree couples a source feature with a redundant feature and creates a fitness function based on mutual information to assess feature redundancy. The trials showed that GPRFC could effectively produce redundant features that greatly enhance cluster and benchmark data sets for classification [18]. Another group of researchers showed that automatic data item alignment in web data extraction may be achieved with a multi-feature, Directed Acyclic Graph (DAG) based multi-tree matching technique. By aligning nodes inside the rooted trees of data records, the algorithm was able to align data items. Determining the global alignment for the offspring nodes of aligned parent nodes was a crucial aspect of the procedure [19].

3 Proposed method

We propose a novel approach named multi-tree algorithm to create multiple decision trees for classification and regression tasks that contain 3 major steps (see Fig. 1). In the first step, we construct the "L" number of multiple decision trees on the training data set followed by the step of filtering the "K" number of trees that have a good performance on the validation data set. Finally, we predict the output based on the technique of the majority vote on the testing data set.

3.1 Core architecture of a decision tree

Before we discuss our main algorithm, we discuss the architecture of a single decision tree. Any decision tree consists of leaf and non-leaf nodes connected with branches (see Fig. 2). This study considers only a binary decision tree where each non-leaf node has only two children nodes. Each non-leaf node in the decision tree is labeled by a question (feature = value) and two branches come out from this node (True and False) which further joins to two children nodes (either leaf or non-leaf node). In the case of the leaf node, it is labeled by the decision (value of the target feature).

We associate a partitioning of the data set for each node. We used information gain based on the uncertainty value of the gini index to partition the data set. For example, D data set is divided into D_{left} with probability p_l (False branch) and D_{right}

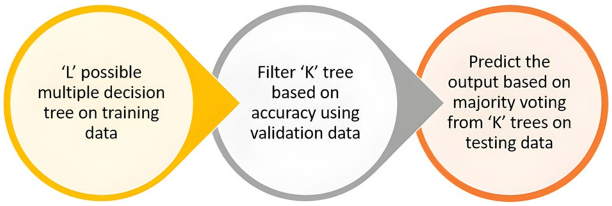

'L' possible multiple decision tree on training data
Filter 'K' tree based on accuracy using validation data
Predict the output based on majority voting from 'K' trees on testing data

Fig. 1 Multi-tree algorithm flowchart

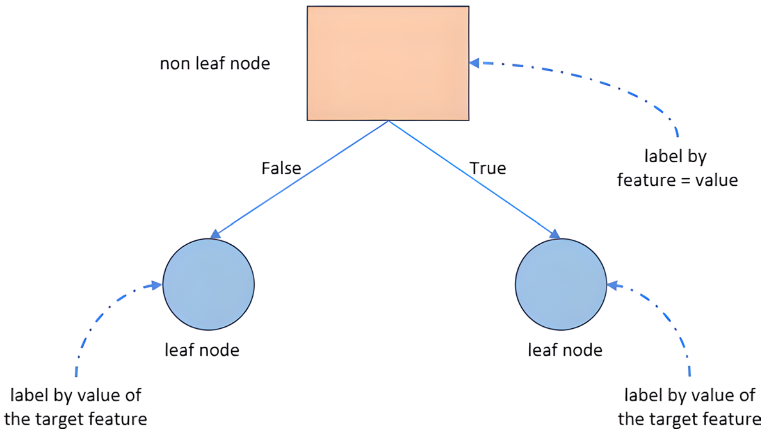


Fig. 2 Decision tree architecture and labeling

with probability p_r (True branch) for the particular feature, then we use the following equation to calculate the information gain (IG). Note that, the calculation of the gini index for a data set S containing m labels (their probabilities $p_1, p_2, \ldots, p_m$) of target features is also given below:

$$gini(S) = 1 - \sum_{i=1}^{m} p_i^2$$

$$IG(D) = gini(D) - p_l \times gini(D_{left}) - p_r \times gini(D_{right})$$

We choose the feature which gives maximum information gain for this partition. Splitting the training data recursively into subsets based on feature value pairs creates the tree until a stopping requirement is satisfied (for example, no more partition is possible). An example of a single decision tree is given in Fig. 3.

3.2 Steps of multi-tree algorithm

Here, we discuss the steps of our multi-tree algorithm in detail. We divide it into three sections considering the three steps. First section, we describe the multiple decision tree construction. In the second section, we filter decision trees that are poorly performed and keep only K decision trees that are good enough for the prediction. Finally, in the third step, we describe the final prediction using the majority voting technique.

3.2.1 Building multiple decision trees

The main idea behind our algorithm is to create decision trees using all the unique training feature value pairs available in the data set. One possible way to do this is by changing the node question (feature=value) for each level of the tree.

Table 1 Example dataset

Age	Income	Buys computer
Youth	High	No
Middle aged	High	Yes
Senior	Medium	No
Youth	Low	Yes

In a traditional decision tree (Fig. 3), the feature value pair that has the maximum score of information gain is considered a node question (denoted by "feature = value") and used to split the data. As a result, a new branch is created, the tree reaches the next level and this continues recursively until the tree reaches the bottom. However, instead of the best node question based on the information gain value for a split, if we use any other feasible node question for that split, we will see a new variation of the branch created below that node question. If the branches continue to build recursively, choosing the best node question for the rest of the tree nodes in the traditional way, we will observe that the final tree is a variation of the base tree. This way, by just changing different node questions in different levels, we can build multiple trees that are variations of the base tree.

The first node question is known as the root of a decision tree. Let us consider a dataset where we have five unique feature value pairs. If we check the information gain, we find that the third feature value has the maximum gain. So traditionally, we will put the third feature as the first node question (i.e., root question), and after completing the tree, it will be considered the base tree. Now if we use all the remaining feature value pairs as a first node question one at a time and let the tree build recursively, then we will get four variations of the base tree. This way, using all the unique feature value pairs of the training data, we can build multiple trees.

Let us consider an example dataset (Table 1) where there are six unique feature value pairs: "Age = youth", "Age = middle", "Age = senior", "Income = high", "Income = medium", and "Income = low". Figure 4 shows how we can

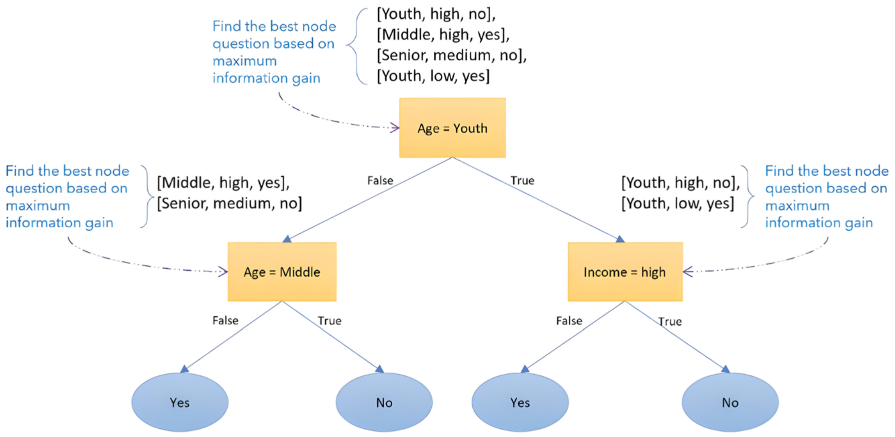

Find the best node question based on maximum information gain
[Youth, high, no], [Middle, high, yes], [Senior, medium, no], [Youth, low, yes]
Age = Youth
False
True
Find the best node question based on maximum information gain
[Middle, high, yes], [Senior, medium, no]
[Youth, high, no], [Youth, low, yes]
Find the best node question based on maximum information gain
Age = Middle
Income = high
False
True
False
True
Yes
No
Yes
No

Fig. 3 Traditional decision tree for the example data set

create multiple trees from this dataset. In Fig. 4a $f_{i_1} = a_{i_1}, f_{i_2} = a_{i_2}, \dots, f_{i_n} = a_{i_n}$ represents all the unique feature value pairs of an input dataset. In the case of the example dataset, $n = 6$. For every six unique feature value pairs, we will get $n = 6$ different trees. Now, if we fix the first node and go to the next level (Fig. 4b) of a tree, we see that the number of remaining unique features is $n - 1$, which is five in our case. Similarly, we can vary one node question at a time and we will get another five trees and the other node questions of the same level will be selected as default by calculating the maximum information gain. The procedure continues at every level at every possible node question of a tree, as shown in Fig. 4c. So for a given dataset of n feature value pairs, the total number of possible multi-trees may be calculated as $n \times (n-1)^2 \times (n-2)^4 \times ... \times 1$. However, we are interested in getting L number of trees to reduce the time complexity of our algorithm. The algorithm is illustrated in Algorithm 1. Note that, we generate trees from top to bottom fashion where we create variations in the top level and then recursively lower levels until we get L decision trees.

Algorithm 1 Multiple Decision Trees Generation

```
Require: $D$ is a training data set
Require: $L$ is the desired number of decision trees
Ensure: A collection of $L$ number of binary decision trees
1: Initialize an empty collection $A$, to store all decision trees.
2: Create a base decision tree and store it to $A$.
3: $No.of.trees \leftarrow 1$
4: Create an empty Queue $Q$, which will contain decision trees along with their level
   where variations will take place
5: Insert the base decision tree and all the variations of the base decision tree at the
   first level to the $Q$
6: while $Q$ is not empty do
7:     Perform dequeue operation from $Q$. As a result, we receive decision tree $T$ and
   its level $l$ of variation
8:     Create many variations of $T$ by changing node questions (different feature value
   pair) in the next level $(l+1)$        ▷ Note that node questions will be sorted
   from maximum information gain to the minimum information gain and one node
   question change will create one variation of $T$
9:     for each new variation $v$ of $T$ do
10:        if $No.of.trees \leq L$ and $v$ is not a duplicate tree in $A$ then
11:            enqueue $v$
12:            store $v$ to $A$
13:            $No.of.trees \leftarrow No.of.trees + 1$
14:        end if
15:    end for
16: end while
17: return $A$
```

Figure 5 shows how using queue data structure, we have achieved our desired goal. Assume that $T_{1 \cdot 1}$ is a variant of the first tree T_1. Now, our algorithm might produce the first decision tree again if we wanted to construct a new variation of

$T_{1 \cdot 1}$. A Duplicate Check function was utilized to eliminate any duplicate decision trees. Duplicate trees in this context refer to trees with comparable node questions and topologies. Figures 9 and 15, for instance, may have comparable structures, but the level two node question on the right side is different. For this reason, these two trees are not regarded as duplicate trees.

For experimentation purposes, we manually hand-drawn all the possible decision trees have for small dataset (Table 1). Surprisingly, we found that it is possible to create 54 decision trees using only six unique feature value pairs. In Appendix A, we have illustrated complete tree diagrams of some of our multi-trees to give more insight into our algorithm.

3.2.2 Filtering step

Algorithm 2 Multi-Tree Filtering Technique

```
Require: $D$ is the set of $L$ number of multiple decision trees
Require: $n\_tolerance$ is the percentage of the accuracy to determine the permissible
    boundary of accuracy for insertion of a new decision tree
Require: $avg\_acc$ is the average value of the validation accuracy of decision trees
    that have been stored in the collection
Require: $K$ is the maximum number of trees in the collection
Require: $V$ is the validation data set
Ensure: A collection of decision trees
 1: Initialize a collection (or Array) $A$ where the decision trees will be stored along
    with their validation accuracy.
 2: Generate the first 3 decision trees from $D$ and save them in $A$ according to their
    validation accuracy from the data set $V$ in non-decreasing order. Calculate $avg\_acc$
    as the average of the validation accuracy of the 3 decision trees.
 3: $tree\_counter \leftarrow 3$
 4: while $tree\_counter \leq K$ do
 5:     Generate a new decision tree $T$ from $D$.
 6:     if validation accuracy of $T$ from the data set $V \geq avg\_acc \times n\_tolerance$ then
 7:         insert $T$ in $A$ so that it retains the ordering of the array according to the
    validation accuracy;
 8:         update the $avg\_acc$ based on the decision trees in $A$;
 9:         $tree\_counter \leftarrow tree\_counter + 1$
10:     end if
11: end while
12: return $A$
```

The decision trees that were created in the previous step may not perform well for the prediction problem. Therefore, we use a validation data set to filter out some decision trees whose performance is not adequate and keep only the K number of best-performing decision trees. Algorithm 2 shows how we filter the trees that are not well performed.

We calculate the average accuracy of the decision trees in the collection. Then, we check the accuracy of a new decision tree whether it is in the range of the tolerable threshold of the average accuracy. If yes then we insert the new decision tree in the collection and update the average accuracy, otherwise, we discard the tree. We continue this process till we get the K number of decision trees.

3.2.3 Majority voting step

The final step in our algorithm is the prediction by the selected k decision trees for the test data set. We use the majority voting technique (see Algorithm 3) where we calculated the predicted output for each row as the maximum votes received from all k decision trees.

Algorithm 3 Predicted output by majority voting

```
Require: $A$ is the collection of multiple decision trees
Require: $D$ is the test data set
Ensure: An array containing the predicted output
 1: Create an array, counter, with a length equal to the number of distinct values of
    the target feature in the test data. Initialize the array counter to 0. ▷ This array
    counts the votes for each value of the target feature
 2: Initialize the predicted output array, $B$, to 0.
 3: for each row $r$ in $D$ do
 4:     for each tree $T$ in $A$ do
 5:         $p \leftarrow$ get the prediction of the row $r$ by the tree $T$
 6:         Find the corresponding index of the prediction $p$ in the array counter and
    increase its value
 7:     end for
 8:     Find the predicted output as the index in the counter array that contains the
    maximum value and insert it in array $B$
 9: end for
10: return $B$
```

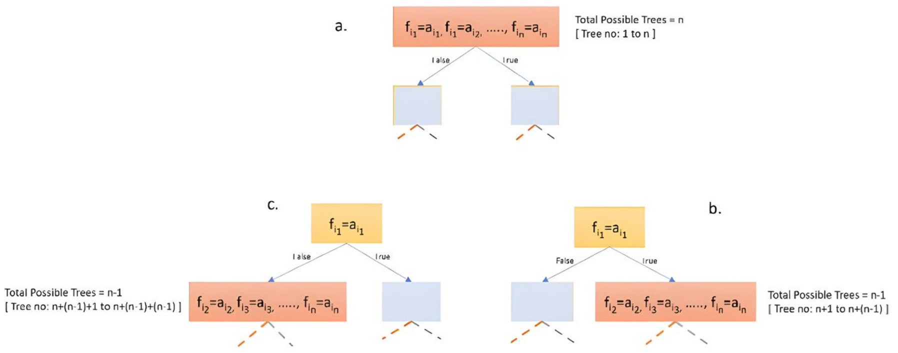
Finally, we compared the actual output and predicted output for all the rows and reported the final accuracy of our method.

4 Experimental results and discussion

In this section, we first go over the various data sets that are utilized for the experiments. Then we discuss various other models that will be used to compare their performance to ours. Lastly, we present and discuss the experimental findings.

4.1 Data sets

We have used 10 data sets for our experiments (see Table 2). Below we provide a short description for each data set.



a.
$f_{i_1}=a_{i_1}, f_{i_1}=a_{i_2}, \ldots, f_{i_n}=a_{i_n}$
Total Possible Trees = n
[Tree no: 1 to n]
False
True
c.
$f_{i_1}=a_{i_1}$
Total Possible Trees = n-1
[Tree no: n+(n-1)+1 to n+(n-1)+(n-1)]
$f_{i_2}=a_{i_2}, f_{i_3}=a_{i_3}, \ldots, f_{i_n}=a_{i_n}$
b.
$f_{i_1}=a_{i_1}$
$f_{i_2}=a_{i_2}, f_{i_3}=a_{i_3}, \ldots, f_{i_n}=a_{i_n}$
Total Possible Trees = n-1
[Tree no: n+1 to n+(n-1)]

Fig. 4 General mechanism of splits and calculation of possible decision trees

The **Breast Cancer data set** [20] contains features that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image. We denote it by "Cancer".

The **Diabetes data set** [21] is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict, based on diagnostic measurements, whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients were females at least 21 years old of Pima Indian heritage. We denote it by "Diabetics".

The **Heart Disease data set** [22] has been used by ML researchers to date. The goal is to predict the presence of heart disease in the patient. We denote it by "Heart".

The **Titanic data set** [23] contains the passenger information of the legendary Titanic ship. The goal is to predict which passenger survived the Titanic shipwreck. We denote it by "Titanic".

The **Netflix data set** [24] provides a snapshot of a sample Netflix user base, showcasing various aspects of user subscriptions, revenue, account details, and activity. We denote it by "Netflix".

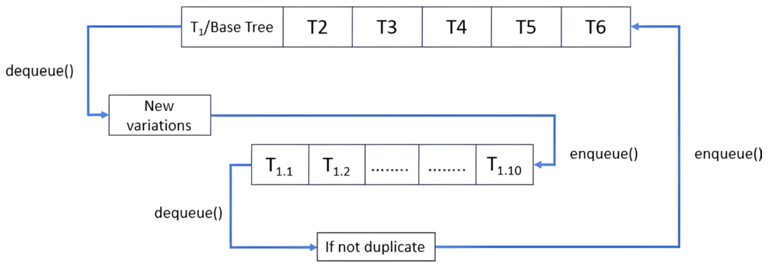


Fig. 5 Queue data structure to create multiple decision trees

Table 2 Dataset characteristics

Dataset name	Total rows	Total features	Data type	Number of classes
Cancer	569	30	Numerical	Binary
Diabetics	768	8	Numerical	Binary
Heart	303	13	Numerical	Binary
Titanic	889	5	Categorical, Numerical	Binary
Netflix	2500	5	Categorical, Numerical	3 classes
CSalary	6698	7	Categorical, Numerical	4 classes
Psychology	1175	19	Categorical, Numerical	Binary
Housing	545	12	Categorical, Numerical	3 classes
CreditRisk	1000	20	Categorical, Numerical	Binary
CarAccept	1728	6	Categorical	4 classes

The **Salary Based on Country and Race data set** [25] contains a total of 6704 entries with 17 feature columns. The objective is to predict 'Education Level' based on country, race, and income salary. We denote it by "CSalary".

The **Psychological Effects of COVID data set** [26] contains answers to various questions provided by people. Most of the questions in the form were provided as multiple-choice questions to avoid any case-sensitive issues. We denote it by "Psychology".

The **Housing Price data set** [27] contains a total of 545 entries with 12 feature columns. The objective is to predict 'furnishing status'. We denote it by "Housing".

The **Credit Card Risk data set** [28] consists of 20 features about the customers, like - duration, credit amount, age, credit history all paid, credit history critical/other existing credit, credit history delayed previously, and so on. Here, the model predicts credit risks as 'good' or 'bad'. We denote it by "CreditRisk".

The **Car Acceptability data set** [29] was derived from a simple hierarchical decision model originally developed for the demonstration of DEX. The model evaluates car acceptability based on overall price (buying and maintaining) and technical characteristics (comfort and safety). We denote it by "CarAccept".

4.2 Data sets preprocessing

We eliminated a feature after first determining whether it had a unique value. Next, using an appropriate mapping, all string or "non-numerical" ordinal values were converted to numeric values. Any row with a 'NaN' value was eliminated. For each of the three dataset partitions—train, validation, and test—a consistent "Label/Target" data ratio was guaranteed. To ensure an effective normalization process, a MinMax() scaling procedure was applied to each division in the dataset. Finally, data balancing is applied only to the training data set using the Synthetic Minority Oversampling Technique (SMOTE). The architecture illustrated in Fig. 6 provides the pipeline used in our methodology, including data pre-processing.

4.3 Other machine learning models

In addition to various datasets, various machine learning models have been employed to evaluate the results against our model.

K-Nearest Neighboring (KNN) is a non-parametric supervised learning classifier that groups individual data points based on closeness (in terms of distance) in order to classify or predict data [30].

Classification and Regression Trees (CART) is a decision tree-based technique that is useful for machine learning problems including both regression and classification. It functions by employing binary splits to recursively divide the training data into smaller subsets. Other tree-based techniques, on the other hand, might permit multiple child nodes. It chooses the features by applying gini impurity functions [5].

Bagging, also known as Bootstrap aggregation, is a type of ensemble learning that involves the parallel training of multiple base models (Decision Tree) on different subsets (bootstrapping) of training data [31].

Gradient Boost is a strong boosting algorithm that transforms several weak learners into strong learners. In gradient boosting, each new learner is trained to minimize the loss function (i.e., mean squared error) of the previous learner by gradient descent [32].

Random Forest A random forest consists of multiple decision trees on various subsets of the data set. The average of the decision trees is used to improve the accuracy of the predicted data set. Instead of using a single decision tree, a random forest collects the result of every tree and expects that the final output will be based on most of the predictions' votes [33].

Extreme Gradient Boosting (XGB) classifier combines the predictions of multiple weak models to generate a more robust prediction for a large scale of data. It is a parallelized and carefully optimized version of the gradient boosting algorithm [34].

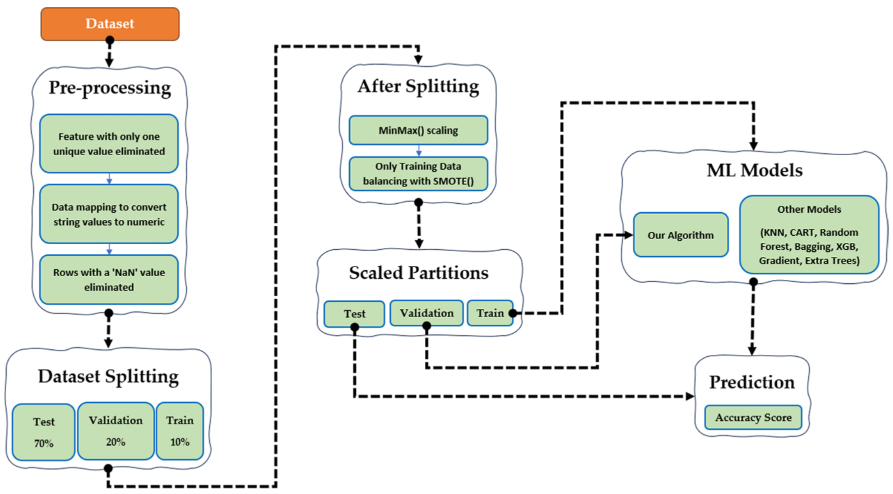

Dataset
Pre-processing
Feature with only one unique value eliminated
Data mapping to convert string values to numeric
Rows with a 'NaN' value eliminated
Dataset Splitting
Test 70%
Validation 20%
Train 10%
After Splitting
MinMax() scaling
Only Training Data balancing with SMOTE()
Scaled Partitions
Test
Validation
Train
ML Models
Our Algorithm
Other Models (KNN, CART, Random Forest, Bagging, XGB, Gradient, Extra Trees)
Prediction
Accuracy Score

Fig. 6 Architecture of our proposed method

Extremely Randomized Tree (Extra Trees) classifiers, also referred to as Extra Tree Classifiers, are ensemble learning techniques that combine the results of different de-correlated tree-based decision trees in a forest to generate their classification result [35].

4.4 Parameters tuning

In this section, we analyze the performance of our method when we tune the parameters *L* and *K*.

4.4.1 L vs. test accuracy

This set of graphs (Fig. 7) shows the changes in the accuracy of our algorithm over different datasets. We tested for five different L values (500, 1000, 1500, 2000, and 2500) while keeping the value of K constant every time. From the graphs, it can be seen that in most cases, the accuracy increases with the increase of the value of L. This indicates that the 'L' parameter is very promising for our algorithm. From the trend of L vs accuracy, it can be concluded that with more computing power, the increase in the value of L will help us achieve much higher accuracy than what we currently achieve.

4.4.2 K vs. test accuracy

Figure 8 provides information on the difference in accuracy for variation of the parameter 'K' for different datasets. In this experiment, we have plotted the accuracy on the Y axis and K = 100, 200 and 300 on the X axis while keeping the value of 'L' constant for each simulation. From the graphs, we can see that there is no direct relationship between the parameter K and the final accuracy. Therefore, we can conclude that K is not a dominant parameter for our algorithm.

4.5 Results and analysis

This section begins with a discussion of the performance metrics. Next, the experimental results are presented, starting with those from real datasets, followed by results from synthetic datasets.

4.5.1 Performance metrics

To evaluate the performance of our algorithm across multiple datasets, we use four performance metrics: accuracy, precision, recall, and F1-score.

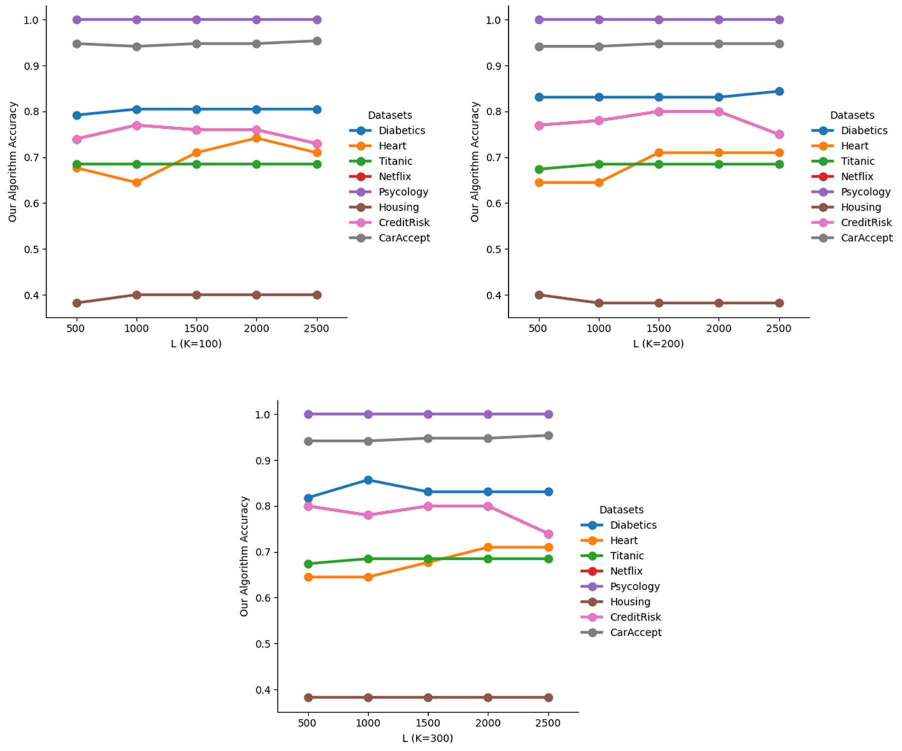


Fig. 7 Performance analysis between parameter (L) vs final accuracy graphs for different datasets

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy measures the proportion of correctly classified instances (both positives and negatives) out of the total instances, providing a general sense of the model's ability to classify correctly. It is especially useful when classes are balanced, meaning the numbers of positive and negative instances are roughly equal. Precision, on the other hand, focuses on the accuracy of positive predictions, calculating the proportion of true positives out of all instances predicted as positive. A high precision score indicates that, when the model predicts a positive, it is likely correct. Recall measures the model's effectiveness in identifying actual positive instances, calculated as the proportion of true positives out of all actual positives. High recall means

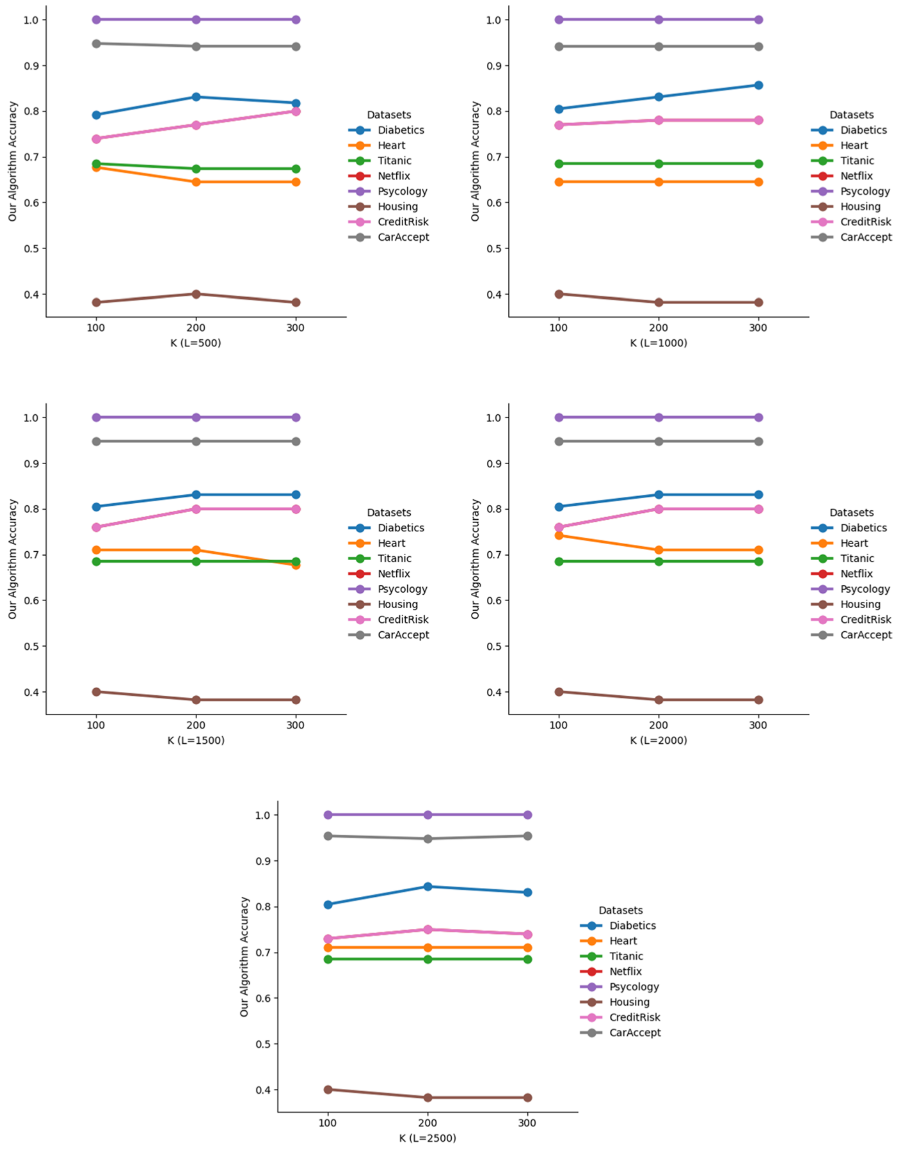


Fig. 8 Performance analysis between parameter (K) vs final accuracy graphs for different datasets

the model captures most of the positive instances, making it valuable in scenarios where missing positives is costly. The F1 Score balances precision and recall, calculated as the harmonic mean of these two metrics, with values ranging from 0 to 1, where 1 signifies the best balance. The F1 Score is particularly useful in cases with imbalanced classes, as it accounts for both false positives and false negatives, providing a comprehensive view of the model's performance when both precision and recall are important.

4.5.2 Evaluation with real dataset

Table 3 shows the optimized values of *L* and *K* used in our algorithm. The most common value for *K* is 300, whereas the most common value for *L* is 500. Table 4 provides comprehensive performance overview of our algorithm compared to other algorithms like BaggingClassifier, XGB, Gradient Boost, and ExtraTrees.

In terms of accuracy, our algorithm achieved high scores in datasets like Cancer (0.965) and Diabetics (0.857, matching Gradient Boost) and led in Netflix (0.800, tied with XGB). However, it performed less optimally in Titanic (0.685) and Housing (0.400), where Gradient Boost (0.764) and RandomForest (0.436) achieved better results. For precision, our algorithm showed strong results in Diabetics (0.842) and Netflix (0.765), equal to Gradient Boost and XGB, respectively, but was lower in Titanic (0.665) and Housing (0.386), where KNN (0.770) and ExtraTrees (0.444) outperformed it. In terms of recall, our algorithm excelled in datasets prioritizing true positive identification, such as Diabetics and CreditRisk (0.864, tied with Gradient Boost and XGB), though it had lower recall in Titanic (0.661) and Housing (0.379), where Gradient Boost (0.753) and ExtraTrees (0.436) performed better. For the F1 Score, which balances precision and recall, our algorithm achieved top scores in Diabetics (0.849), Netflix (0.752), and CreditRisk (0.752), but performed lower in Titanic (0.663) and Housing (0.382), where Gradient Boost (0.751) and ExtraTrees (0.436) excelled. These results indicate that our algorithm is a versatile and competitive model, showing robust performance in datasets like Cancer (F1 of 0.963) and Psychology (F1 of 1.0), though it may benefit from optimization in datasets with complex or noisy patterns, making it a valuable alternative to established models across diverse data conditions.

Table 3 Performance metrics and optimized parameters of our algorithm on real dataset

Dataset name	Accuracy	Precision	Recall	F1-score	K	L
Cancer	0.965	0.957	0.972	0.963	300	500
Diabetics	0.857	0.842	0.864	0.849	300	1000
Heart	0.742	0.739	0.739	0.739	100	2000
Titanic	0.685	0.665	0.661	0.663	100	500
Netflix	0.8	0.765	0.743	0.752	300	500
CSalary	0.594	0.595	0.594	0.594	300	1500
Psychology	1.0	1.0	1.0	1.0	100	500
Housing	0.4	0.386	0.379	0.382	200	500
CreditRisk	0.8	0.765	0.743	0.752	300	500
CarAccept	0.954	0.861	0.903	0.874	100	2500

Table 4 Comparison of our algorithm on real dataset in terms of Accuracy (a), Precision (b), Recall (c), F1-score (d)

Dataset name	KNN	CART	Random forest	Bagging Classifier	XGB	Gradient boost	Extra Trees	Our algorithm
a.								
Cancer	0.965	0.912	1.0	0.947	1.0	1.0	1.0	0.965
Diabetics	0.662	0.818	0.831	0.831	0.805	0.857	0.779	0.857
Heart	0.742	0.548	0.71	0.71	0.645	0.71	0.742	0.742
Titanic	0.764	0.742	0.742	0.753	0.742	0.764	0.753	0.685
Netflix	0.71	0.72	0.77	0.75	0.8	0.78	0.75	0.8
CSalary	0.51	0.587	0.593	0.578	0.6	0.599	0.593	0.594
Psycology	0.881	0.992	1.0	1.0	0.992	0.992	0.992	1.0
Housing	0.455	0.418	0.436	0.436	0.364	0.418	0.436	0.4
CreditRisk	0.71	0.64	0.78	0.7	0.8	0.78	0.75	0.8
CarAccept	0.902	0.942	0.96	0.954	0.96	0.948	0.954	0.954
b.								
Cancer	0.962	0.902	1.0	0.948	1.0	1.0	1.0	0.957
Diabetics	0.657	0.8	0.815	0.813	0.788	0.842	0.765	0.842
Heart	0.739	0.539	0.707	0.708	0.643	0.714	0.757	0.739
Titanic	0.77	0.726	0.726	0.74	0.726	0.75	0.738	0.665
Netflix	0.644	0.674	0.732	0.702	0.765	0.741	0.705	0.765
CSalary	0.543	0.595	0.599	0.58	0.6	0.635	0.595	0.595
Psycology	0.869	0.988	1.0	1.0	0.988	0.988	0.988	1.0
Housing	0.485	0.408	0.429	0.423	0.364	0.42	0.444	0.386
CreditRisk	0.644	0.585	0.75	0.643	0.765	0.741	0.7	0.765
CarAccept	0.817	0.789	0.87	0.824	0.862	0.847	0.881	0.861
c.								
Cancer	0.962	0.921	1.0	0.938	1.0	1.0	1.0	0.972
Diabetics	0.672	0.8	0.836	0.827	0.807	0.864	0.787	0.864

Table 4 (continued)

Dataset name	KNN	CART	Random forest	Bagging Classifier	XGB	Gradient boost	Extra Trees	Our algorithm
Heart	0.739	0.538	0.704	0.71	0.632	0.697	0.727	0.739
Titanic	0.719	0.724	0.724	0.727	0.724	0.753	0.733	0.661
Netflix	0.621	0.686	0.683	0.66	0.743	0.71	0.65	0.743
CSalary	0.51	0.587	0.593	0.578	0.6	0.599	0.592	0.594
Psycology	0.897	0.993	1.0	1.0	0.993	0.993	0.993	1.0
Housing	0.456	0.407	0.412	0.416	0.35	0.393	0.431	0.379
CreditRisk	0.621	0.59	0.69	0.643	0.743	0.71	0.688	0.743
CarAccept	0.891	0.798	0.938	0.869	0.938	0.93	0.936	0.903
d.								
Cancer	0.962	0.908	1.0	0.943	1.0	1.0	1.0	0.963
Diabetics	0.652	0.8	0.821	0.819	0.794	0.849	0.769	0.849
Heart	0.739	0.536	0.705	0.708	0.631	0.698	0.728	0.739
Titanic	0.729	0.725	0.725	0.732	0.725	0.751	0.735	0.663
Netflix	0.628	0.678	0.697	0.671	0.752	0.721	0.662	0.752
CSalary	0.515	0.586	0.595	0.578	0.599	0.605	0.592	0.594
Psycology	0.876	0.991	1.0	1.0	0.991	0.991	0.991	1.0
Housing	0.454	0.406	0.416	0.417	0.353	0.398	0.436	0.382
CreditRisk	0.628	0.586	0.707	0.643	0.752	0.721	0.693	0.752
CarAccept	0.85	0.792	0.897	0.845	0.896	0.878	0.905	0.874

4.5.3 Evaluation with synthetic datasets

4.5.3.1 Dataset creation This study involved the generation of synthetic datasets to explore the effects of varying dimensionality, noise levels, and the presence of outliers on classification performance. Using the *make_classification* function from scikit-learn, datasets were created with controlled parameters: three levels of dimensionality (3, 5, and 7 features), three levels of noise intensity (low, moderate, high), and varying levels of outliers (none, low, and moderate). Each dataset contained 100 samples, balanced equally between two classes. Noise was introduced by adjusting the *flip_y* parameter, which randomly flips a proportion of the class labels, simulating incorrect labeling and reflecting noise intensities of 1%, 10%, and 30%. This mimics real-world data imperfections, such as mislabeling or measurement errors, testing the algorithms' robustness in handling inaccuracies. Outliers were introduced by randomly selecting data points and modifying their values with extreme deviations, simulating real-world anomalies.

The datasets were split into training, validation, and test sets using stratified sampling to maintain class balance, followed by scaling with Min–Max normalization. These controlled manipulations provided a systematic investigation of how changes in dimensionality, noise, and outliers affect model performance, offering insights into the adaptability and robustness of classification algorithms under various data conditions. The processed datasets were saved in structured formats, facilitating further analysis and replication of the study.

4.5.3.2 Evaluation The performance of various synthetic datasets was evaluated across different conditions, including varying dimensionality, noise levels, and the presence of outliers, using standard classification metrics such as accuracy. The results, summarized in the Table 5, indicate that the dataset with moderate dimensionality achieved the highest performance, with perfect score (accuracy = 1), suggesting an ideal scenario for classification. Datasets with low and high dimensionality showed similar performance (accuracy = 0.9), indicating that the complexity of feature space did not significantly impair our algorithm's performance.

Regarding noise levels, the dataset with low noise maintained high performance (accuracy = 0.9), whereas moderate noise introduced substantial degradation (accuracy = 0.7). High noise resulted in a slight improvement compared to moderate noise but still showed decreased overall performance (accuracy = 0.8). This trend highlights the sensitivity of the model to noise, especially at moderate levels, which substantially impacted classification accuracy.

The presence of outliers showed minimal impact on the model's performance, with datasets containing no outliers, low outliers, and moderate outliers all achieving similar performance metrics (accuracy = 0.9). This consistency suggests that our algorithm maintained robust performance in the face of outlier introduction, demonstrating resilience against data anomalies.

Table 5 Performance metrics and optimized parameters of our algorithm on Synthetic Dataset

Dataset name	Accuracy	Precision	Recall	F1-score	K	L
low_dim	0.9	0.917	0.9	0.899	100	500
moderate_dim	1	1	1	1	100	500
high_dim	0.9	0.917	0.9	0.899	100	500
low_noise	0.9	0.917	0.9	0.899	100	500
moderate_noise	0.7	0.708	0.7	0.697	100	500
high_noise	0.8	0.857	0.8	0.792	100	500
no_outliers	0.9	0.917	0.9	0.899	100	500
low_outliers	0.9	0.917	0.9	0.899	100	500
moderate_outliers	0.9	0.917	0.9	0.899	100	500

4.5.3.3 Comparison with other algorithms Our algorithm exhibits strong and consistent performance across a range of synthetic datasets, demonstrating adaptability in terms of accuracy, precision, recall, and F1 score. It performs competitively with established models, particularly in scenarios with moderate dimensionality, low noise, and varying levels of outliers, where it consistently achieves or closely approaches the highest scores across all metrics. While our algorithm maintains high performance in high-dimensional and outlier-prone datasets, its sensitivity to moderate noise is evident, with slight declines in precision, recall, and F1 scores compared to ensemble models like Gradient Boost and ExtraTrees, which generally show greater robustness under noisy conditions. Despite these occasional limitations, our algorithm effectively balances precision and recall, performing on par with traditional algorithms like KNN and XGB, and frequently matching or surpassing models such as RandomForest and BaggingClassifier. These results indicate that our algorithm is a reliable and versatile model for diverse data conditions, with room for optimization in noise handling, reinforcing its potential as a robust alternative in classification tasks across complex data landscapes (Table 6).

4.5.4 Statistical analysis

The statistical analysis of our algorithm was conducted using the Friedman Test, via the STAC platform [36], with a significance level of 0.05 to compare its performance against several established machine learning algorithms, including Gradient Boost, Bagging Classifier, Random Forest, Extra Trees, XGB, K-Nearest Neighbors (KNN), and CART (Decision Trees). Table 7 shows that the test leading to the acceptance of the null hypothesis (H0), which asserts that there is no statistically significant difference between the mean performances of the algorithms. This outcome confirms that our algorithm performs on par with leading models, underscoring its competitiveness and reliability.

Table 6 Comparison of our algorithm on synthetic dataset in terms of Accuracy(a),Precision(b), Recall(c), F1- score(d)

Dataset name	KNN	CART	Random forest	Bagging classifier	XGB	Gradient boost	Extra trees	Our algorithm
a								
low_dim	0.9	0.9	0.9	0.9	0.9	1.0	1.0	0.9
moderate_dim	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0
high_dim	0.9	0.9	0.8	0.8	0.9	0.9	0.9	0.9
low_noise	0.9	0.9	0.9	0.9	0.9	1.0	0.9	0.9
moderate_noise	0.9	0.8	0.8	0.8	0.8	0.8	0.9	0.7
high_noise	0.8	0.8	0.8	0.8	0.8	0.7	0.8	0.8
no_outliers	0.9	0.9	0.9	0.9	0.9	1.0	0.8	0.9
low_outliers	0.9	1.0	0.9	1.0	0.9	1.0	0.9	0.9
moderate_outliers	0.9	1.0	0.9	1.0	0.9	1.0	0.8	0.9
b								
low_dim	0.917	0.917	0.917	0.917	0.917	1.000	1.000	0.917
moderate_dim	0.917	1.000	1.000	1.000	1.000	1.000	1.000	1.000
high_dim	0.917	0.917	0.800	0.800	0.917	0.917	0.917	0.917
low_noise	0.917	0.917	0.917	0.917	0.917	1.000	0.917	0.917
moderate_noise	0.917	0.800	0.800	0.800	0.800	0.800	0.917	0.708
high_noise	0.857	0.857	0.857	0.857	0.857	0.708	0.857	0.857
no_outliers	0.917	0.917	0.917	0.917	0.917	1.000	0.800	0.917
low_outliers	0.917	1.000	0.917	1.000	0.917	1.000	0.917	0.917
moderate_outliers	0.917	1.000	0.917	1.000	0.917	1.000	0.800	0.917
c								
low_dim	0.9	0.9	0.9	0.9	0.9	1.0	1.0	0.9
moderate_dim	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0
high_dim	0.9	0.9	0.8	0.8	0.9	0.9	0.9	0.9
low_noise	0.9	0.9	0.9	0.9	0.9	1.0	0.9	0.9

Table 6 (continued)

Dataset name	KNN	CART	Random forest	Bagging classifier	XGB	Gradient boost	Extra trees	Our algorithm
moderate_noise	0.9	0.8	0.8	0.8	0.8	0.8	0.9	0.7
high_noise	0.8	0.8	0.8	0.8	0.8	0.7	0.8	0.8
no_outliers	0.9	0.9	0.9	0.9	0.9	1.0	0.8	0.9
low_outliers	0.9	1.0	0.9	1.0	0.9	1.0	0.9	0.9
moderate_outliers	0.9	1.0	0.9	1.0	0.9	1.0	0.8	0.9
d								
low_dim	0.899	0.899	0.899	0.899	0.899	1.000	1.000	0.899
moderate_dim	0.899	1.000	1.000	1.000	1.000	1.000	1.000	1.000
high_dim	0.899	0.899	0.800	0.800	0.899	0.899	0.899	0.899
low_noise	0.899	0.899	0.899	0.899	0.899	1.000	0.899	0.899
moderate_noise	0.899	0.800	0.800	0.800	0.800	0.800	0.899	0.697
high_noise	0.792	0.792	0.792	0.792	0.792	0.697	0.792	0.792
no_outliers	0.899	0.899	0.899	0.899	0.899	1.000	0.800	0.899
low_outliers	0.899	1.000	0.899	1.000	0.899	1.000	0.899	0.899
moderate_outliers	0.899	1.000	0.899	1.000	0.899	1.000	0.800	0.899

4.5.4.1 Ranking analysis The ranking analysis using the Friedman test evaluated the relative performance of our proposed algorithm against other established machine learning models across multiple datasets. The results demonstrated that our algorithm consistently ranked within the top four models across key metrics-accuracy, precision, recall, and F1 score. Notably, it achieved the highest rank in 5 out of 10 datasets, indicating its robust performance and adaptability to diverse data types.

4.5.4.2 Post-hoc analysis The post-hoc analysis was conducted to determine if there were statistically significant differences between our algorithm and other top-performing models. In most pairwise comparisons, the adjusted p values supported the acceptance of the null hypothesis, meaning that our algorithm's performance was not statistically different from leading models such as Random Forest, Gradient Boost, and XGB. This finding emphasizes that our algorithm performs comparably to well-established algorithms across various conditions, reinforcing its reliability as a versatile model.

4.5.4.3 Implications of the results The statistical findings demonstrate that our algorithm consistently performs at a level comparable to the best algorithms available, even under varying data conditions. While it ranks just below the top-performing models, its position within the top four confirms its strength and adaptability. The acceptance of the null hypothesis across all comparisons underscores that our algorithm is not only reliable but also a highly competitive choice among advanced machine learning methods. These results validate our algorithm as a practical, high-performing alternative in classification tasks, offering a balance of performance and consistency that rivals established models. As such, our algorithm represents a valuable tool for complex data analysis, capable of delivering results on par with the most recognized techniques in the field.

5 Conclusions

This study introduced a novel multi-tree algorithm that constructs multiple decision trees by modifying each node question, utilizing all available training data, and refining tree construction through various parameter adjustments. Unlike traditional approaches such as Random Forest, our algorithm implements a majority voting scheme for predictions, demonstrating superior performance by achieving the highest accuracy in 5 out of 10 datasets tested. The results highlight the potential of our model as a highly generalized approach, capable of adapting to various data types with consistent performance. Statistical analysis further confirms that our algorithm ranks among the top four models, performing comparably to leading machine learning methods across diverse conditions. The acceptance of the null hypothesis in

Table 7 Statistical analysis in terms of Accuracy (a), Precision (b), Recall (c), F1- score (d)

(a)

Test type	Statistic	*p* value	Significance level	Result
Friedman Test	1.25198	0.27965	0.05	H0 accepted (No significant difference)
	Rank	Algorithm	Rank Score	
	1	GradientBoost	3.42105	
	2	BaggingClassifier	4.15789	
	3	RandomForest	4.36842	
	4	OurAlgorithm	4.42105	
	5	XGB	4.44737	
	6	ExtraTrees	4.5	
	7	KNN	5.28947	
	8	CART	5.39474	
	Comparison	Statistic	Adjusted *p* value	Result
	OurAlgorithm vs CART	1.29142	1.00000	H0 accepted
	OurAlgorithm vs Gradient Boost	1.19208	1.00000	H0 accepted
	OurAlgorithm vs KNN	1.15897	1.00000	H0 accepted
	OurAlgorithm vs Bagging-Classifier	0.33113	1.00000	H0 accepted
	OurAlgorithm vs ExtraTrees	0.13245	1.00000	H0 accepted
	OurAlgorithm vs XGB	0.09934	1.00000	H0 accepted
	OurAlgorithm vs Random-Forest	0.03311	1.00000	H0 accepted

(b)

Test Type	Statistic	*p* value	Significance Level	Result
Friedman Test	1.38324	0.21799	0.05	H0 accepted (No significant difference)
	Rank	Algorithm	Rank Score	
	1	GradientBoost	3.23684	
	2	ExtraTrees	4.18421	
	3	RandomForest	4.28947	
	4	OurAlgorithm	4.47368	
	5	XGB	4.52632	
	6	BaggingClassifier	4.76316	
	7	KNN	5.15789	
	8	CART	5.36842	
	Comparison	Statistic	Adjusted *p* value	Result
	OurAlgorithm vs Gradient Boost	1.55633	0.83741	H0 accepted
	OurAlgorithm vs CART	1.12585	1.00000	H0 accepted

Table 7 (continued)

(b)

Test Type	Statistic	*p* value	Significance Level	Result
	OurAlgorithm vs KNN	0.86095	1.00000	H0 accepted
	OurAlgorithm vs ExtraTrees	0.36425	1.00000	H0 accepted
	OurAlgorithm vs Bagging-Classifier	0.36425	1.00000	H0 accepted
	OurAlgorithm vs Random-Forest	0.23179	1.00000	H0 accepted
	OurAlgorithm vs XGB	0.06623	1.00000	H0 accepted

(c)

Test Type	Statistic	*p* value	Significance Level	Result
Friedman Test	1.40943	0.20714	0.05	H0 accepted (No significant difference)
	Rank	Algorithm	Rank Score	
	1	GradientBoost	3.28947	
	2	XGB	4.28947	
	3	OurAlgorithm	4.31579	
	4	RandomForest	4.36842	
	5	ExtraTrees	4.39474	
	6	BaggingClassifier	4.65789	
	7	CART	5.21053	
	8	KNN	5.47368	
	Comparison	Statistic	Adjusted *p* value	Result
	OurAlgorithm vs KNN	1.45699	1.00000	H0 accepted
	OurAlgorithm vs Gradient Boost	1.29142	1.00000	H0 accepted
	OurAlgorithm vs Decision Trees	1.12585	1.00000	H0 accepted
	OurAlgorithm vs Bag-gingClassifier	0.43047	1.00000	H0 accepted
	OurAlgorithm vs ExtraTrees	0.09934	1.00000	H0 accepted
	OurAlgorithm vs Ran-domForest	0.06623	1.00000	H0 accepted
	OurAlgorithm vs XGB	0.03311	1.00000	H0 accepted

(d)

Test Type	Statistic	*p* value	Significance Level	Result
Friedman Test	1.36261	0.22686	0.05	H0 accepted (No significant difference)
	Rank	Algorithm	Rank Score	

Table 7 (continued)

(d)

Test Type	Statistic	*p* value	Significance Level	Result
	1	GradientBoost	3.23684	
	2	ExtraTrees	4.28947	
	3	RandomForest	4.34211	
	4	OurAlgorithm	4.36842	
	5	XGB	4.52632	
	6	BaggingClassifier	4.65789	
	7	CART	5.26316	
	8	KNN	5.31579	
	Comparison	Statistic	Adjusted *p* value	Result
	OurAlgorithm vs Gradient Boost	1.42387	1.00000	H0 accepted
	OurAlgorithm vs KNN	1.19208	1.00000	H0 accepted
	OurAlgorithm vs CART	1.12585	1.00000	H0 accepted
	OurAlgorithm vs BaggingClassifier	0.36425	1.00000	H0 accepted
	OurAlgorithm vs XGB	0.19868	1.00000	H0 accepted
	OurAlgorithm vs ExtraTrees	0.09934	1.00000	H0 accepted
	OurAlgorithm vs RandomForest	0.03311	1.00000	H0 accepted

multiple comparisons emphasizes its reliability and competitiveness. While there is room for further optimization, particularly with increased computational resources and expanded parameter ranges, our algorithm stands out as a robust, high-performing alternative in classification tasks. These findings validate its relevance and utility as a versatile tool in complex data analysis, offering a balance of accuracy and adaptability that aligns closely with the most established algorithms in the field.

Appendix A Tree variations of example dataset

A shows some possible Muti-Trees built from Table 1 example dataset. The six unique features are used to build six different trees (Figs. 9, 10, 11, 12, 13 and 14). Now to increase the number of trees, node questions of the next level are used accordingly shown in Figs. 15, 16 and 17.

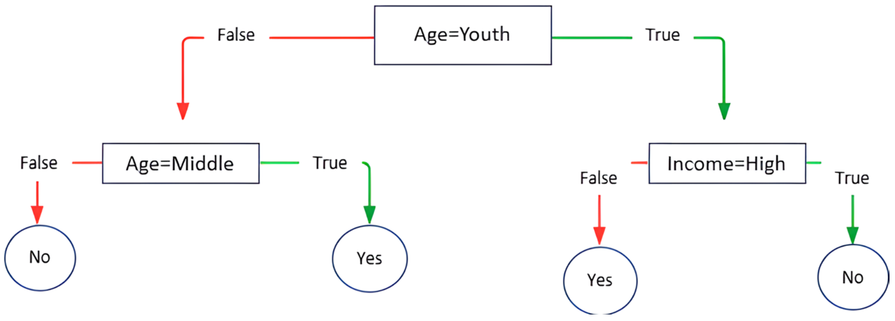


Fig. 9 Example dataset multi-tree variation 1(Level 1)

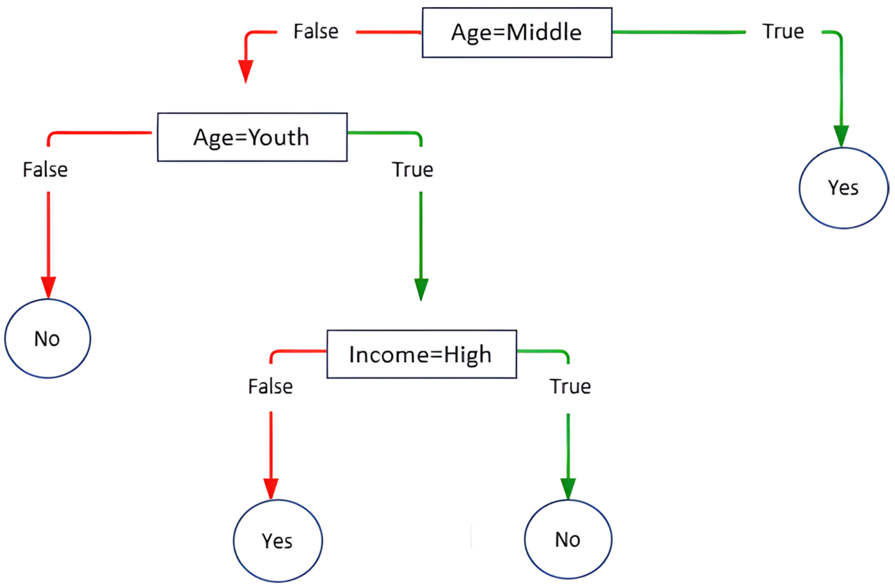


Fig. 10 Example dataset multi-tree variation 2(Level 1)

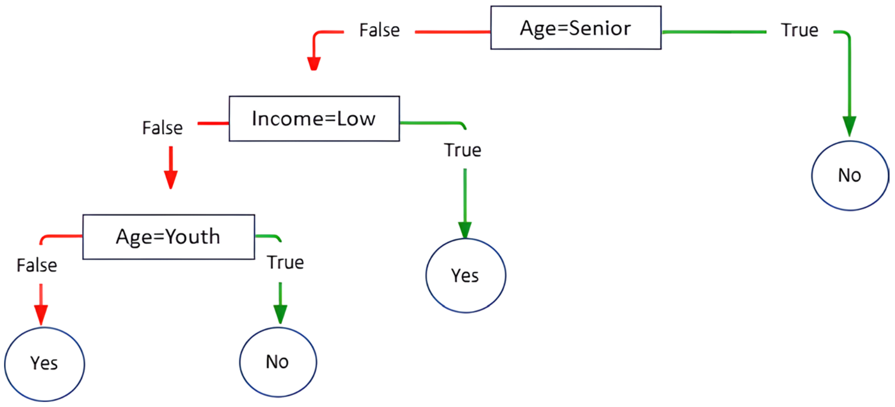


Fig. 11 Example dataset multi-tree variation 3(Level 1)

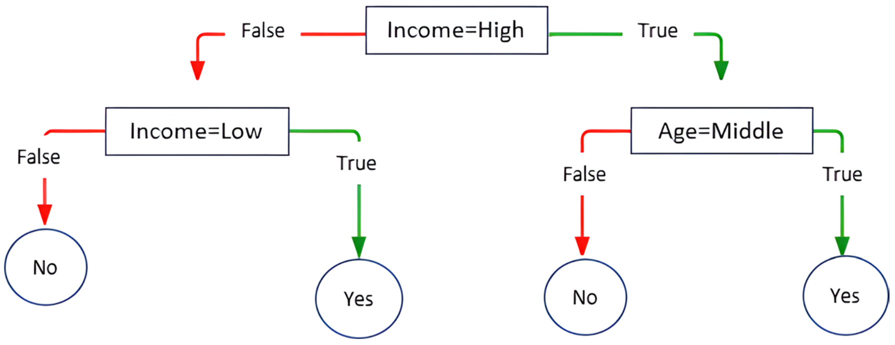


Fig. 12 Example dataset multi-tree variation 4(Level 1)

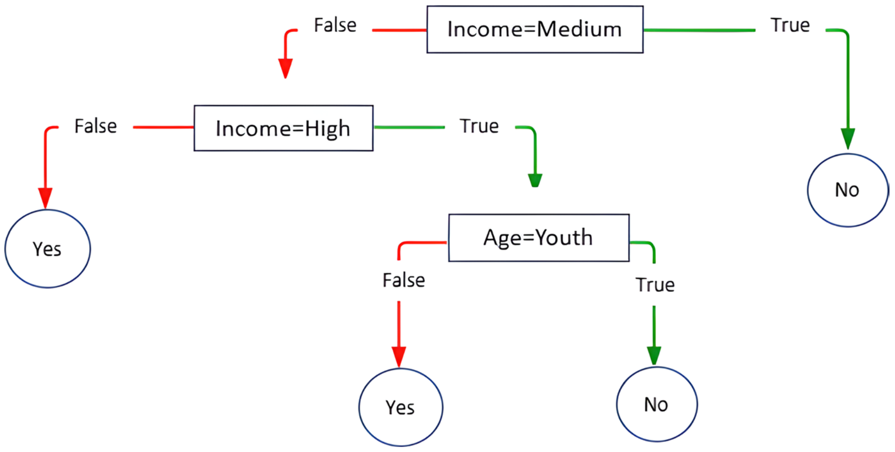


Fig. 13 Example dataset multi-tree variation 5(Level 1)

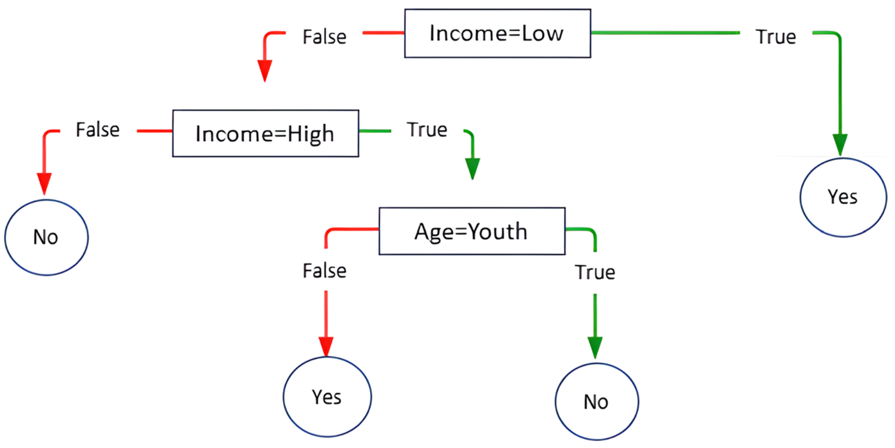


Fig. 14 Example dataset multi-tree variation 6(Level 1)

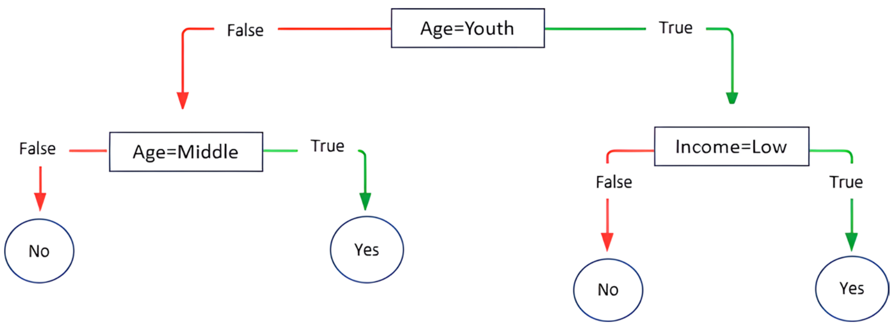


Fig. 15 Example dataset multi-tree variation 7(Level 2)

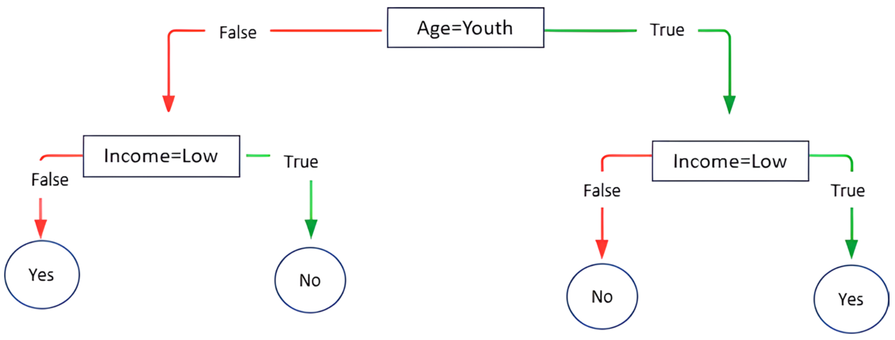


Fig. 16 Example dataset multi-tree variation 8(Level 2)

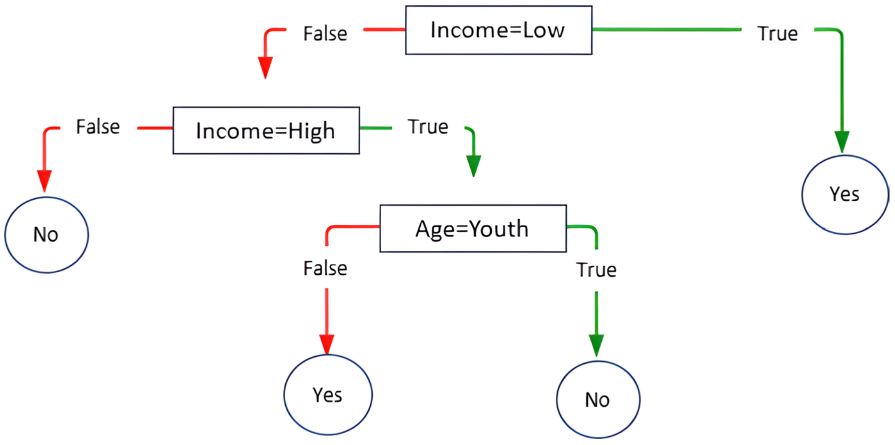


Fig. 17 Example dataset multi-tree variation 9(Level 2)

Acknowledgements The authors would like to thank Jouf University and King Abdullah University of Science & Technology to support this research.

Author contributions MA: Conceptualization, Supervision, Project administration, Writing—original draft, Validation, Writing—review and editing. THN: Software, Data Curation, Writing—original draft. MM: Conceptualization, Funding.

Data availability Available upon request.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethics approval and consent to participate Not applicable.

References

1. Maqsood I, Khan MR, Abraham A (2004) An ensemble of neural networks for weather forecasting. Neural Comput Appl 13:112–122
2. West D, Dellana S, Qian J (2005) Neural network ensemble strategies for financial decision applications. Comput Oper Res 32(10):2543–2559

3. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Biometrics 40:874
4. Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann, Burlington
5. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Chapman and Hall/CRC, Boca Raton
6. Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106
7. Batista GEAPA, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explor Newsl 6(1):20–29. https://doi.org/10.1145/1007730.1007735
8. Kwok SW, Carter C (1990) Multiple decision trees. Mach Intell Pattern Recognit 9:327–335. https://doi.org/10.1016/B978-0-444-88650-7.50030-5
9. Osei-Bryson K-M (2004) Evaluation of decision trees: a multi-criteria approach. Comput Oper Res 31(11):1933–1945. https://doi.org/10.1016/S0305-0548(03)00156-4
10. Todorovski L, Džeroski S (2000) Combining multiple models with meta decision trees. In: Zighed DA, Komorowski J, Żytkow J (eds) Principles of data mining and knowledge discovery. Springer, Berlin, pp 54–64
11. Hu H, Li J, Wang H, Daggard G, Shi M (2006) A maximally diversified multiple decision tree algorithm for microarray data classification. WISB '06. Australian Computer Society, , pp 35–38
12. Hu H, Li J-Y, Wang H, Daggard G, Wang L-Z (2008) Robustness analysis of diversified ensemble decision tree algorithms for microarray data classification. In: 2008 International conference on machine learning and cybernetics, vol 1, pp 115–120. https://doi.org/10.1109/ICMLC.2008.4620389
13. Li J, Liu H (2003) Ensembles of cascading trees. In: Third IEEE international conference on data mining, pp 585–588. https://doi.org/10.1109/ICDM.2003.1250983
14. Islam Z, Giggins H (2011) Knowledge discovery through sysfor: a systematically developed forest of multiple decision trees. In: Proceedings of the ninth Australasian data mining conference, vol 121, pp 195–204
15. Azad M, Chikalov I, Moshkov M (2020) Representation of knowledge by decision trees for decision tables with multiple decisions. Procedia Comput Sci 176:653–659. https://doi.org/10.1016/j.procs.2020.09.037
16. Salama K, Otero F (2014) Learning Multi-tree Classification Models with Ant Colony Optimization. In Proceedings of the International Conference on Evolutionary Computation Theory and Applications (IJCCI 2014) -ECTA; ISBN 978-989-758-052-9, SciTePress 38–48. https://doi.org/10.5220/0005071300380048
17. Al-Helali B, Chen Q, Xue B, Zhang M (2020) Multi-tree genetic programming for feature construction-based domain adaptation in symbolic regression with incomplete data. In: Proceedings of the 2020 genetic and evolutionary computation conference, pp 913–921
18. Lensen A, Xue B, Zhang M (2018) Generating redundant features with unsupervised multi-tree genetic programming. In: Genetic programming: 21st European conference, EuroGP 2018, Parma, Italy, April 4–6, 2018, Proceedings 21. Springer, pp 84–100
19. Shi S, Liu C, Yuan C, Huang Y (2014) Multi-feature and dag-based multi-tree matching algorithm for automatic web data mining. In: 2014 IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT), vol 1, pp 118–125. IEEE
20. William W, Mangasarian O, Street N, Street W (1995) Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B
21. Smith JW, Everhart JE, Dickson W, Knowler WC, Johannes RS (1988) Using the adap learning algorithm to forecast the onset of diabetes mellitus. In: Annual symposium on computer application in medical care, p 261. American Medical Informatics Association
22. Andras J, William S, Matthias P, Robert D (1988) Heart disease. UCI Machine Learning Repository. https://doi.org/10.24432/C52P4X
23. Cukierski W (2012) Titanic: machine learning from disaster. Kaggle, San Francisco
24. Arnav: Netflix Userbase Dataset. Kaggle. https://www.kaggle.com/datasets/arnavsmayan/netflix-userbase-dataset (2023)
25. Spasivska V (2023) Salary dataset based on country and race. Kaggle. https://www.kaggle.com/datasets/veronikanikaaa/salary-data-based-on-country-and-race
26. Hemanth: Psycological Effects of COVID (2023). https://doi.org/10.34740/KAGGLE/DSV/6172485. https://www.kaggle.com/dsv/6172485

27. KUMARdatalab H (2023) Housing price prediction. Kaggle. https://www.kaggle.com/datasets/harishkumardatalab/housing-price-prediction
28. Rijn JV (2023) Credit risk customers. Kaggle. https://doi.org/10.34740/KAGGLE/DS/3119852. https://www.kaggle.com/ds/3119852
29. Bohanec M (1997) Car evaluation. UCI Machine Learning Repository. https://doi.org/10.24432/C5JP48
30. Fix E, Hodges JL (1989) Discriminatory analysis—nonparametric discrimination: consistency properties. Int Stat Rev 57:238
31. Breiman L (2004) Bagging predictors. Mach Learn 24:123–140
32. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29:1189–1232
33. Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010950718922
34. Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. KDD '16. ACM. https://doi.org/10.1145/2939672.2939785
35. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63:3–42
36. Rodríguez-Fdez I, Canosa A, Mucientes M, Bugarín A (2015) STAC: a web platform for the comparison of algorithms using statistical tests. In: Proceedings of the 2015 IEEE international conference on fuzzy systems (FUZZ-IEEE)