Trustable Decision Tree Model using Else-Tree Classifier

Truong Tran (truong.tran@psu.edu)
Pennsylvania State University
Marc Pusey
University of Alabama in Huntsville
Ramazan Aygun
Kennesaw State University

Research Article

Keywords: Trustable Machine Learning, Decision Tree, Classification, Minimizing Misclassification

Posted Date: October 5th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2002014/v2

Trustable Decision Tree Model Using Else-Tree Classifier

Truong Tran[1*], Marc Pusey[2] and Ramazan Aygun[3]

[1*]Computer Science, Penn State Harrisburg, Penn State University, Middletown, PA, USA.
[2]Department of Chemistry, The University of Alabama in Huntsville, Huntsville, AL, USA.
[3]Computer Science Department, Kennesaw State University, Kennesaw, GA, USA.

*Corresponding author(s). E-mail(s): truong.tran@psu.edu;
Contributing authors: mlp0041@uah.edu; raygun@kennesaw.edu;

Abstract

With advances in machine learning and artificial intelligence, learning models have been used in many decision-making and classification applications. The nature of critical applications, which require a high level of trust in the prediction results, has motivated researchers to study classification algorithms that would minimize misclassification errors. In our study, we have developed the *trustable machine learning methodology* that allows the classification model to learn its limitations by rejecting the decision on cases likely yield to misclassifications and hence produce highly confident outputs. This paper presents our trustable decision tree model through the development of the *Else-Tree* classifier algorithm. In contrast to the traditional decision tree models, which use a measurement of impurity to build the tree and decide class labels based on the majority of data samples at the leaf nodes, Else-Tree analyzes homogeneous regions of training data with similar attribute values and the same class label. After identifying the longest or most populated contiguous range per class, a decision node is created for that class, and the rest of the ranges are fed into the *else* branch to continue building the tree model. The Else-Tree model does not necessarily assign a class for conflicting or doubtful samples. Instead, it has an else-leaf node, led by the last else branch, to determine

rejected or undecided data. The Else-Tree classifier has been evaluated and compared with other models through multiple datasets. The results show that Else-Tree can minimize the rate of misclassification.

Keywords: Trustable Machine Learning, Decision Tree, Classification, Minimizing Misclassification

1 Introduction

Because incorrect prediction result or misclassification error is costly in many critical classification applications, improving the performance of learning algorithms has paramount importance. For example, cancer detection, tornado prediction, fraud identification, etc., are some examples where misclassifications could not be accepted. Evaluation metrics, such as accuracy, are helping researchers to evaluate and develop better classifiers. Even when an classifier is optimized, the error still exists due to factors like the random cause of the irreducible error and some trade-off between underfitting and overfitting.

To reduce the prediction error, Alpaydin [1] has described an approach that when a data instance cannot be classified accurately, the classification model may reject to make a prediction or postpone the decision to an expert. For example, an incorrect decision made by a predictive model in medical diagnosis can lead to mistaken treatment and may cost a patient's life. In such situations, the predictive model rejecting to make a decision and delegating the burden decision to an expert could be a better option. In our study on protein crystallization analysis, the crystallographer recommended having a third class called "likely-leads" in addition to crystal and non-crystal categories [2–4]. The rationale is the fact that the missed crystals are likely to be classified as "likely-leads" rather than "non-crystals," and human experts can review them and classify them manually. This technique could reduce the error of classifying crystals as non-crystals. However, there are limitations to both approaches. The method proposed by Alpaydin [1] does not provide a specific implementation for classification algorithms, and it is not simple to determine the reject cases for different classification models. On the other hand, adding an additional class label (to reject decisions) to the training data creates a multi-class classification problem, which still has misclassification issues among these classes.

To overcome those problems, we have developed the trustable machine learning methodology, which allows the classifier to learn to identify difficult-to-classify data samples and flags them as 'rejects.' Other data are classified as usual. It is important to note that **none of the data samples are labeled as difficult-to-classify in the training data, but the machine learning model will be trained to identify them**. After training, the classifier can flag difficult-to-classify samples, indicating that it rejects to make a decision on these items and defer them to a human expert with necessary precautions. The

probability of correct classification by an expert is expected to be much higher than that of the classifier in all reject cases. The technique helps the model to yield only the accurate output, thus avoiding misclassifications. We have presented WisdomNet [5] as the neural networks model of trustable machine learning method. For the decision tree (DT) model, we introduced the Else-Tree classifier briefly in our previous work [6] for bioinformatics. In this paper, we present more details of the development and evaluation of the Else-Tree classifier with interpretability.

We propose Else-tree classifier that can minimize the prediction error by not assigning a class label to the sample that could be misclassified. The doubtful samples are passed through the 'else' branch of the tree and marked as 'reject'. To build the else-tree, the algorithm first splits the training data into homogeneous regions of elements with similar attribute value and the same class label. The class label of the data sample is assigned as the value of that region or range. Then the boundaries of those pure regions are used to create decision nodes and children nodes. During testing, if data goes into pure regions, then its class is predicted to be the same as the region. Otherwise, else branch is followed. If the decision tree cannot benefit from splitting, data that goes into the last else node is marked as 'reject' or '?'. Rather than using impurity, our method focuses on the purity of the range of values. Each attribute is analyzed to determine the largest range of regions belonging purely to a specific class. The rest of the ranges are fed into the else branch of the tree for further analysis. In other words, this would mean that the currently selected attribute is not capable of classifying data samples in the 'else' node.

The rest of the paper is organized as follows. Section 2 presents an overview of the DT model and its inherent misclassification problem. The descriptions of Else-Tree and the training algorithm are given in Section 3. The experimental results for testing Else-Tree are given in Sections 4. The discussions about the Else-Tree algorithm and its current limitations are presented in Section 5. The last section concludes the paper.

2 Related Work And Inherent Misclassification Problem of Tree-based Classifiers

The DT classifier is a supervised learning model with a combination of classification rules for classifying data. A tree-based structure is used to organize the rules in the DT model. The model contains decision nodes, branches, and leaf nodes. The first decision node is the root node of the tree. For each decision node in the model, a test is performed on one or more data attributes. The result of the test determines a branch to a subtree. The leaf node represents a class label. For classification, the input data is first fed to the root node. Then, the classification procedure follows a unique path determined by the attribute values of the data to a leaf node, which decides the class membership. The DT algorithm learns the classification rules from the training data. Iterative Dichotomiser 3 (ID3) [7] is the most well-known DT algorithm, which

recursively partitions the training set into disjoint subsets based on the data attributes. At each recursive iteration, the selection of the attribute value to partition the training dataset is decided by some statistical tests. The selected attribute becomes the attribute of a decision node, and the partitioning conditions become the classification rules of that node. Each training subset is used in the next recursive call. If there is no improvement in the statistical tests or no more subsets being split, a leaf node is created, and the current recursive call is terminated. The class label of a leaf node is determined by the majority class of the training data samples in that leaf node.

One factor that has a critical influence on the accuracy of classifications made by a DT is the statistical measures that is used to select the attribute to partition the data at the decision nodes. Many popular decision tree algorithms use an impurity measure to determine the optimal split. For example, ID3 algorithm [7] uses entropy, whereas CART, SLIQ, and SPRINT use Gini index [8–10], and C4.5 algorithm uses gain ratio [11]. However, there is one inherent misclassification problem in these DT algorithms. When all attributes are used and the remaining training data samples cannot be partitioned further, the label of a leaf node is assigned to the majority class. In this situation, the samples belong to the other classes are misclassified.

Let us define the 'pure' and 'impure' sets of data as follows. Let, R be the given set of data samples. We consider D as a subset of set R. There are two possible cases for D:

- Pure set: D contains samples that belong to a single class. The class label of data samples in a pure set is homogeneous.
- Impure set: D contains data samples that belong to a mixture of classes. For the binary classification, the impure set contains data samples of both classes.

An example of the misclassification error is shown in Fig. 1. In this example, consider that a DT algorithm has partitioned a set of data samples into three disjoint subsets in two-dimensional feature space. The class labels of the given data samples are '+' and '-.' As shown in the figure, R_1 and R_2 are two pure subsets of R. All data samples in R_1 belong to the '+' class, and all data samples in R_2 belong to the '- class. Subset R_3 is impure because it has data samples belonging to both '+' and '-' classes. As shown in Fig. 1, R_3 has the majority of samples in '+'. Hence, the DT algorithm may determine the class label for data samples in R_3 as '+'. Any sample that falls into R_3 is predicted as '+'. If an actual '-' data falls into R_3, it shall be incorrectly classified as '+' by the DT classifier. Generally, the classification accuracy correlated with the class label decided based on impure leaf nodes is lower than the accuracy correlated with the label decided by the pure leaf nodes.

The goal of DT algorithms is to find the best attribute at each node for splitting to generate a compact and simple DT. Each decision attribute is determined based on an impurity measure, and DT growth algorithms do not have the purpose of identifying samples that are difficult to classify. Hence

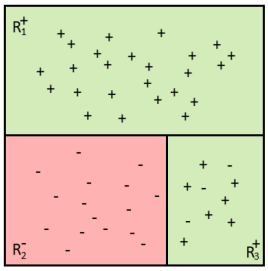


Fig. 1 A simple rectangular decision regions with pure and impure subsets of '+' and '-' samples.

earlier decisions (close to the root node) based on the impurity measure may lead to false class labels at the lower levels of the tree. On the other hand, Else-Tree classifier intrinsically assumes that there are samples hard to classify and start focusing on samples that are easy to classify and sending other samples down the tree to be evaluated using other attributes.

3 Methodology

This section first describes the structure of Else-Tree as an approach of trustable machine learning for the DT classifier. After providing two important functions for Else-Tree growth (finding pure subsets and partitioning), we provide Else-Tree training algorithm.The training algorithm for Else-Tree developed in this study considers the binary classification problem with numeric data attributes.

3.1 Structure of Else-Tree

Else-Tree classifier has a tree structure composed of decision nodes, branches, and leaf nodes. Similar to other DT model, the first decision node in Else-Tree is the root node. Else-Tree has two types of leaf nodes: class-leaf node and else-leaf node. The class-leaf node provides a predictive class label for the input data. The else-leaf node is the flag to indicate that the data item is is a 'reject', which cannot be classified into any predefined class labels. The branches at a decision node in Else-Tree belong to two groups, class-leaf branch and else-branch. The class-leaf branch leads to a class-leaf node having the decision for the input. The else-branch leads to another decision node in the tree, and the last else branch leads to the else-leaf node.

Else-Tree is trained recursively to choose the appropriate attribute and partition the training dataset into disjoint pure subsets. At each recursive call, the attribute selection is determined by the number of samples in the pure subsets. We favor the pure subset containing the most training samples. When an attribute is selected, it is used to create a decision node and partition the training dataset into the largest pure subset for each class label. Then, each class-leaf node is created according to each selected pure subset of the

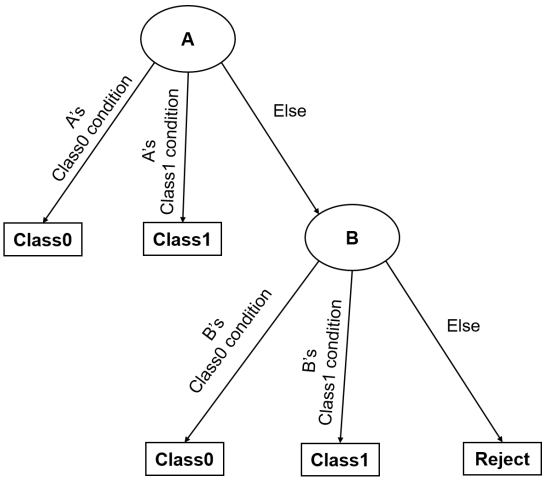


Fig. 2 A sample decision Else-Tree structure for binary classification problem.

training dataset. The remaining samples are fed to the next recursive call using the else-branch. Note that, to avoid overfitting, we require the selected pure subset to contain at least a minimum number of training samples. The training process stops when it cannot generate any pure subset containing an acceptable number of training data samples.

For classifying a new data item, the process starts from the root node and follows a unique route based on the attribute values of the item to a leaf node in Else-Tree. If the classification process leads to a class-leaf node, the data item is classified to a class label identified by the class-leaf node. If the process ends up at the else-leaf, the data item is not classified into any class label and is determined as a reject. The Else-Tree classifier can reduce rate of incorrect predictions by learning the classification rules from the pure subsets of the training data samples and rejecting to make a prediction when the classification process reaches the else-leaf node.

Fig. 2 illustrates the structure of a simple Else-Tree classifier having two decision nodes. The first decision node, also is the root node, caries out a test on the value of attribute A, and the second decision node caries out a test on the value of attribute B. Each decision node has two class-leaves for predicting $Class0$ and $Class1$ labels. The else-branch of the first decision node leads to the second decision node, and the else branch of the second decision node leads to the else-leaf node. When classifying a data sample, the classification starts at the root node and takes a particular path according to the tests on the values of attributes A and B of the data sample. The input data sample will reach the else-leaf node if it does not satisfy any class-leaf conditions. In this case, the data sample is marked as a reject and is not classified to either $Class0$ or $Class1$.

3.2 Finding Pure Subsets for an Attribute

This section describes a helper function that are used in the Else-Tree training procedure. We consider a training dataset R of numerical attributes for a binary classification problem. Let A be the considered attribute to analyze the best pure partition. Note that A is one of the attributes of data samples in R. Algorithm 1 covers the procedures of the Get-Max-Pure-Subset-by-Attribute function.

Algorithm 1 Get-Max-Pure-Subset-by-Attribute(R, A)

Input:

$R = \{x_1, x_2, \cdots, x_n\}$: training dataset of n samples x for binary classification of $Class0$ and $Class1$

A: attribute with m distinct values a_i

Output:

R^0: the largest pure subset of $Class0$.

R^1: the largest pure subset of $Class1$.

1: Sort(R) with respect to A

▷ Create kernel sets $k_1, k_2, \cdots, k_m$

2: $k_i = \{x | x \in R, x_A = a_i\}$ for $1 \leq i \leq m$

3: **for each** k_i **do**

4: **if** $\forall x(x \in k_i \rightarrow x \in Class0)$ **then**

5: $\ell(k_i) =$ Class0 ▷ set class label of k_i to $Class0$

6: **else if** $\forall x(x \in k_i \rightarrow x \in Class1)$ **then**

7: $\ell(k_i) =$ Class1 ▷ set class label of k_i to $Class1$

8: **else**

9: $\ell(k_i) =$ impure ▷ set class label of k_i as 'impure'

10: **end if**

11: **end for**

12: Combine adjacent kernel sets of the same class into larger sets

13: Return (R^0, R^1), the largest pure subset of each class

The step-by-step description of Algorithm 1 is as follows.

Line 1: *Sort R with respect to attribute A.*

All data samples in R are sorted based on the ascending order of the values of attribute A. Let us denote the sorted values of attribute A as $\{a_1, a_2, \cdots a_m\}$, where $a_i < a_{i+1}$, $1 \leq i \leq \mid (m-1) \mid$.

Lines 2-11: *Create kernel sets.*

First, create m subsets $k_1, k_2, \cdots, k_m$ of the input dataset R, where k_i contains data samples whose value of the attribute A equals a_i. We refer to each set k_i as a kernel set of the input dataset. Then, each kernel set k_i is labeled according to the class of its data samples. If all samples in k_i belong to a single class, k_i is a pure set. Otherwise, the set is impure. The pure set is marked with the same class label of its data samples. The impure set is marked

as 'impure.' Note that the class labels of the data samples are unchanged during this process.

Line 12: *Combine adjacent kernel sets.*

Based on the values of the attribute A in order as $\{a_1, a_2, \cdots, a_m\}$, the adjacent kernel sets with the same class label are combined into a single set to create a larger subset of the input data. For instance, if c consecutive sets $k_i, k_{i+1}, \cdots, k_{i+c-1}$ have the same class label, they are combined into a single set $D_{i(i+c-1)}$ with the values of the attribute A in $[a_i, a_{i+c-1}]$.

Line 13: *Select and return the largest pure subset of each class.*

From the combined subsets created in the previous step, the pure subset that contains the most data sample of each class is selected and returned. The selection of a pure subset is guided by the number of data members. For each class, only the pure subset containing the highest number of data samples is selected and returned. If there is no pure subset, the function returns two empty sets.

Example 1: Fig. 3 illustrates two function calls for Get-Max-Pure-Subset-by-Attribute on a sample dataset R having ten samples and two attributes, A and B. The data samples in R belong to $Class0$ and $Class1$. The result of Get-Max-Pure-Subset by-Attribute(R, A) when considering attribute A is shown in Fig. 3a. In this case, the largest pure subset of $Class0$ has five data samples, and the largest subset for $Class1$ has three samples. Fig. 3b shows the results of Get-Max-Pure Subset-by-Attribute(R, B) for attribute B. In this case, both the largest pure subsets of $Class0$ and $Class1$ contain two data samples.

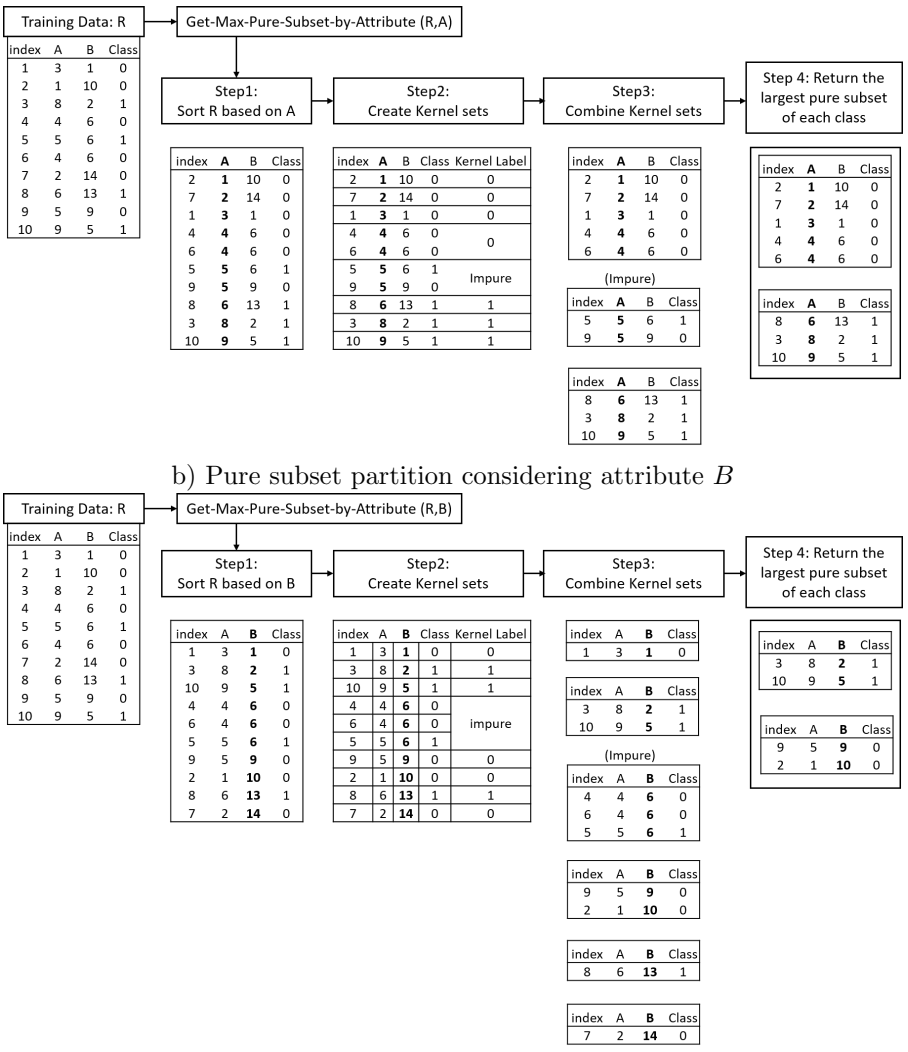


Fig. 3 An example of the Get-Max-Pure-Subset-by-Attribute function.

3.3 Finding Best Pure Partition for Else-Tree

This part presents Find-Best-Pure-Partition function that exhaustively searches for the appropriate attribute to partition the training data. As shown in Algorithm 2, the function processes a training set of binary classification data and returns the best-selected attribute and two corresponding pure maximal sub-datasets. If there is no attribute that can be used to generate a pure subset of the input data samples, the function returns a *NULL* attribute and two empty sets. First, the function loops to process all attributes of the input data. In each iteration, the function considers one attribute at a time and calls the Get-Max-Pure-Subset-by-Attribute function to get the max pure subset of each data class with respect to the attribute. The final selected attribute is the one that yields the pure subset with the highest number of data samples. After the exhaustive search selects the appropriate attribute, the Get-Max-Pure-Subset-by-Attribute is called again with the selected attribute to determine the largest pure subset for each class.

Algorithm 2 Find-Best-Pure-Partition(R))

Input:

R: Binary classification training dataset of $Class0$ and $Class1$ with d numerical attributes $A_1, A_2, \cdots, A_d$

Output:

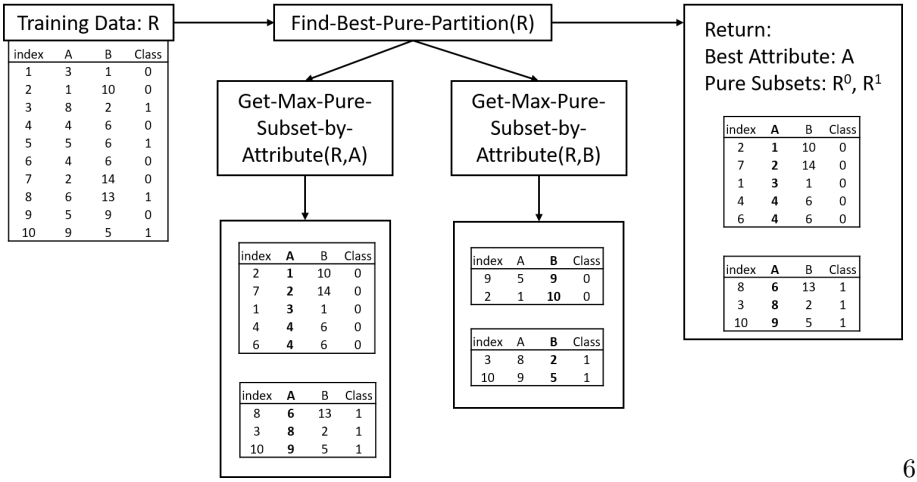
best_Attribute: The best attribute selected

R^0: The largest pure subset of $Class0$.

R^1: The largest pure subset of $Class1$.

1: $R^0 = \emptyset$, $R^1 = \emptyset$
2: max_Size = 0
3: best_Attribute = $NULL$
4: **for each** attribute A_j **do**
5: (R^0, R^1) = Get-Max-Pure-Subset-by-Attribute(R, A_j)
6: **if** max_Size < $|R^0|$ **then**
7: max_Size = $|R^0|$
8: best_Attribute = A_j
9: **end if**
10: **if** max_Size < $|R^1|$ **then**
11: max_Size = $|R^1|$
12: best_Attribute = A_j
13: **end if**
14: **end for**
15: **if** best_Attribute is NOT $NULL$ **then**
16: (R^0, R^1) = Get-Max-Pure-Subset-by-Attribute(R, best_Attribute)
17: **end if**
18: Return (*best_Attribute*, R^0, R^1)

Example 2: Fig. 4 illustrates an example of Find-Best-Pure-Partition function. The input dataset R is the same as in Example 1, having ten data samples with two attributes, A and B, and the class labels are $Class0$ and $Class1$. Find-Best-Pure-Partition function repeatedly calls the Get-Max-Pure-Subset-by-Attribute function to consider each attribute of the data samples in R. As shown in Fig. 4, when considering attribute A, the largest subset of $Class0$ has five data samples, and the largest pure subset of $Class1$ has three data samples. For attribute B, the largest pure subsets of $Class0$ and $Class1$ have two data samples each. Hence, attribute A is selected and returned as the best attribute because it can be used to create the larger pure subset of data samples. As shown in Fig. 4, the two largest pure subsets separated by the values of the selected attribute A are returned by the function.



6

Fig. 4 An example of the Find-Best-Pure-Partition function

3.4 Else-Tree Training Algorithm

This section describes the algorithm to generate an Else-Tree classifier from a training dataset. The recursive training procedure, Else Tree-Growth, is presented in Algorithm 3. The algorithm takes a binary classification dataset R and a pruning threshold $minNum$ as inputs and returns an Else-Tree classification model. A step-by-step description of the Else-Tree-Growth algorithm is given below.

Algorithm 3 Else-Tree-Growth($R, minNum$)

Input:

R: A binary classification dataset with samples belonging to $Class0$ and $Class1$

$minNum$: Pruning threshold as the minimum number of training samples per class-leaf

Output:

Else-Tree classifier model

▷ ***Step 1: General case when set R ≠ empty***

if $R \neq \emptyset$ **then**

  Create a $Node$ which has a *NULL* decision attribute

  $nodeUpdated = false$ ▷ boolean flag indicating $Node$'s status

  $R_{temp} = \emptyset$ ▷ a temporary set to support data split

▷ ***Step 2: Get the best_Attribute and the pure subsets*** R^0 ***and*** R^1

  ($best\_Attribute, R^0, R^1$) = **Find-Best-Pure-Partition**(R)

▷ ***Step 3: Update the tree Node***

  for each pure subset R^i in $\{R^0, R^1\}$ **do**

    if $|R^i| \geq minNum$ **then**

      if *Node's decision attribute is NULL* **then**

        Set the *decision attribute* for $Node = best\_Attribute$

      end if

      Create a *class-leaf* with label = (class label of R^i)

      Add a new *class branch* to *Node*, the branch's condition is decided by testing the *decision attribute*'s values and R^i's boundary

      Add the *class-leaf* to this new *class branch*

      $R_{temp} = R_{temp} + R^i$

      $nodeUpdated = true$

    end if

▷ ***Step 4: Recursive call***

    if $nodeUpdated$ **then**

      $R_{sub} = R - R_{temp}$ ▷ Split data

      Add a new *else branch* to *Node*

      Add subtree **Else-Tree-Growth**($R_{sub}, minNum$) to this new *else branch*

      Return $Node$

    else

      Return **Else-Tree-Growth**($\emptyset, minNum$)

    end if

  end for

else

▷ ***Step 5: Base case when the input set is empty, R = ∅***

  Create an *Else-Leaf* node

  Return *Else-Leaf*

end if

Step 1 (Lines 1-4) *General case:* If the input dataset R is not empty, a node with a *NULL* decision attribute is created followed by Step 2. Otherwise, the algorithm continues with Step 5.
Step 2 (Line 5): The largest pure subsets R^0 of $Class0$ and R^1 of $Class1$, and the best attribute (best_Attribute) is obtained by calling Find-Best-Pure-Partition(R) function followed by Step 3.
Step 3 (Lines 6-16): The current node is updated based on the result of best_Attribute and the pure subsets R^0 and R^1 obtained in Step 2.

Note that when R^0 and R^1 contain just a few training data samples, they may increase the size of Else-Tree and cause overfitting. Else-Tree algorithm applies the pre-pruning technique to avoid this problem. Only the subset containing at least *minNum* data samples is used to create a class-leaf. For example, if $minNum$ is set to two, only the pure subsets with two or more data samples will be used to create a class-leaf.

In this step, the pure subset of each class returned by the Find-Best-Pure-Partition function is compared with *minNum* threshold. If the number of samples in R^i is greater than or equal to *minNum*, the algorithm will update the current node of Else-Tree. First, the decision attribute of the current node is set to the best attribute selected in Step 2. Next, a new class branch is added to the current node. The condition of this class branch is determined based on the values of the decision attribute corresponding to the boundary of the pure subset R^i. Then, a class-leaf with the class label of the data samples in R^i is added to this new class branch. Note that all data samples in pure subset R^i belong to the same class.

When both pure subsets R^0 and R^1 have less than *minNum* training samples, the current node is not updated. Note that Find-Best-Pure-Partition(R) may return R^0 and R^1 as two empty sets when it fails to generate pure subsets of the input data samples. Threshold *minNum* still covers this situation because an empty set has no data samples.

There are two possible cases of the current node status at the end of this step:

- Case 1: The current node is updated. In this case, at least one of the two pure subsets R^0 and R^1 has the number of training samples more than or equal to *minNum*. Hence, at least one class-leaf is added to the node.
- Case 2: The current node is not updated. In this case, both pure subsets R^0 and R^1 have less than *minNum* training samples. There is no decision attribute assigned to the current node, and no class-leaf is created.

These two cases determine the procedures of Step 4.
Step 4 (Lines 17-25) *Recursive Call:* If the current node is updated in Step 3, then an else branch is added to the current node, and a subtree is added to this else branch by recursively calling Else-Tree-Growth on the remaining training samples. Finally, this step returns the current node. In this case, the training samples that do not belong to a class-leaf are allocated into the sub-dataset

R_{sub}. The recursive function call, Else-Tree-Growth(R_{sub}), repeats Step 1 with R_{sub} as the input dataset to create a subtree added to the current node.

On the other hand, if the current node is not updated, then Else-Tree-Growth($\emptyset$) is returned. In this case, Step 1 is repeated with an empty input dataset.

Step 5 (Lines 26-29) *Base Case:* If the input dataset R is empty, an else-leaf is created and returned, then the process is terminated.

Example 3: This part describes an example to illustrate the Else-Tree-Growth algorithm. Consider the training set R with two attributes A and B as shown in Table 1 to train an Else-Tree classifier. R contains 16 data samples of $Class0$ and $Class1$. In this example, the pruning threshold, $minNum$ is set as 2.

The first recursive call of the Else-Tree-Growth algorithm is illustrated in Fig. 5. In Step 1, since the training dataset is not empty, the algorithm continues to build the tree by creating a root node. In Step 2, Find-Best-Pure-Partition(R) function is called as illustrated in Fig. 6. When considering attribute A, we can obtain the largest pure subset of four data samples. For attribute B, the largest pure subset contains three data samples. Hence, A is selected as the best attribute that can be used to partition the training samples. When separated by the values of the selected attribute A, both largest pure subsets R^0 and R^1 contain four training samples. Both R^0 and R^1 have a greater number of data samples than $minNum$ threshold. In Step 3, A is set as the decision attribute of the root node, the boundary conditions of R^0 and R^1 with respect to the values of attribute A are used to create two class leaves, and the remaining training data samples are allocated into the subset R_{sub} for the next recursive call. As shown in Fig. 5, R_{sub} has eight data samples.

In Fig. 5, the data samples in pure subset R^0 have attribute A values in the range of $[1, 2]$. For pure subset R^1, attribute A values are in $[5, 6]$. Hence, the classification rules at the root node can be stated as follows.

- If $1 \leq A \leq 2$ Then $class = Class0$
- If $5 \leq A \leq 6$ Then $class = Class1$
- Otherwise, continue with the next node.

The second recursive call of Else-Tree-Growth algorithm is illustrated in Fig. 7. In this iteration, attribute B is selected as the best attribute. The largest pure subset of $Class0$ has three samples with attribute B values in the range of $[1, 3]$. There are two samples in the largest pure subset of $Class1$ with attribute B values in $[13, 20]$. Hence, both pure subsets are not pruned. The classification rules at this node can be stated as follows.

- If $1 \leq B \leq 3$ Then $class = Class0$
- If $13 \leq B \leq 20$ Then $class = Class1$
- Otherwise, continue with the next node.

The remaining input dataset for the next recursive call contains three training data samples as shown in Fig. 7.

Table 1 The training data used in Example 3.

Index	A	B	Class	Index	A	B	Class
1	2	1	0	9	3	3	0
2	4	15	1	10	1	15	0
3	1	10	0	11	6	18	1
4	5	15	1	12	5	21	1
5	6	15	1	13	4	11	0
6	4	1	0	14	3	13	1
7	2	9	0	15	4	20	1
8	3	9	1	16	3	9	0

The third recursive call of Else-Tree-Growth is illustrated in Fig. 8. According to the remaining training data samples, only one pure subset is obtained, and this pure subset contains just one data sample. Hence, it is pruned by the pruning threshold of $minNum = 2$, and the current node is not updated. The decision attribute of the current node remains empty. Therefore, this recursive call does not return a decision node. Instead, it returns the result of the recursive call Else-Tree-Growth($\emptyset$, 2), which takes in an empty dataset.

In the fourth recursive call shown in Fig. 9, an else-leaf is created and returned because the input dataset is empty. This iteration is the last recursive call of Else-Tree-Growth. After all recursive calls returning their results, Else Tree classifier is formed. The classification rules for this Else-Tree model can be summarized as follows.

Classification rules of the Else-Tree classifier in Example 3:

i. If $1 \leq A \leq 2$ Then $class = Class0$
ii. If $5 \leq A \leq 6$ Then $class = Class1$
iii. Otherwise, continue with the next node.
  iv. If $1 \leq B \leq 3$ Then $class = Class0$
  v. If $13 \leq B \leq 20$ Then $class = Class1$
  vi. Otherwise, Reject.

An example of testing these classification rules is shown in Fig. 11. For example, the input item with $A = 2$ and $B = 2$ satisfies rule (i), $1 \leq A \leq 2$, hence the predicted label for this data item is $Class0$. On the other hand, when feeding an input sample with $A = 4$ and $B = 12$, none of the classification rules leading to a class-leaf are satisfied. Hence, this data sample falls to the else-leaf and is classified as 'reject.'

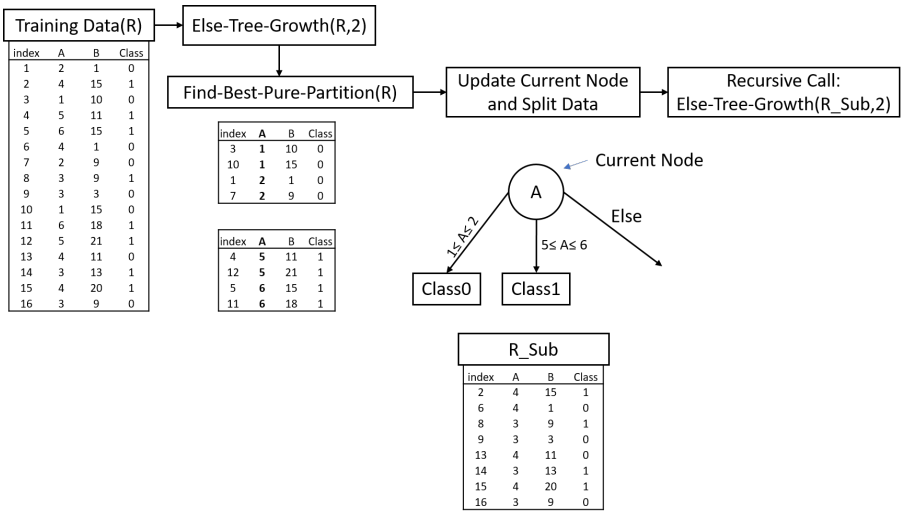


Fig. 5 An illustration of recursive iteration 1 of the Else-Tree-Growth algorithm

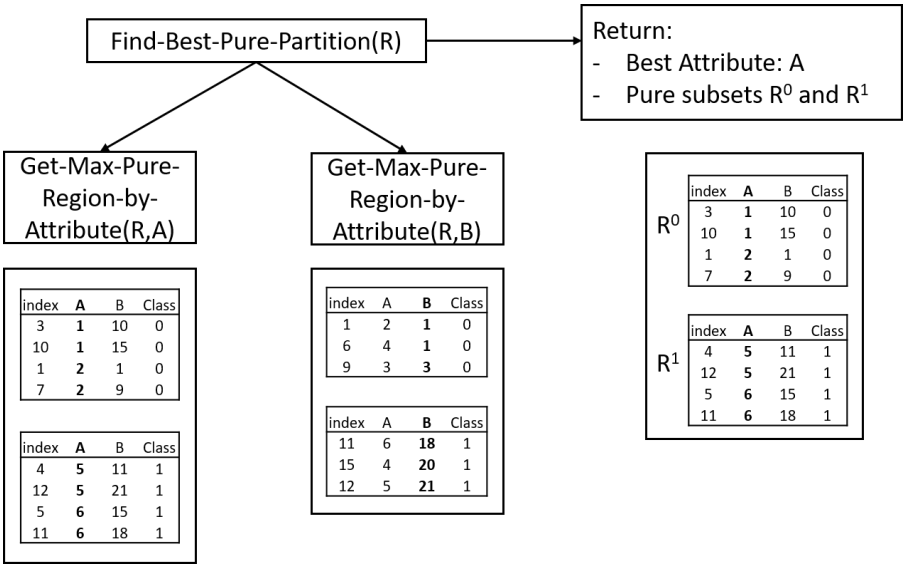


Fig. 6 An illustration of Find-Best-Pure-Partition procedure in recursive iteration 1 of the Else-Tree-Growth algorithm

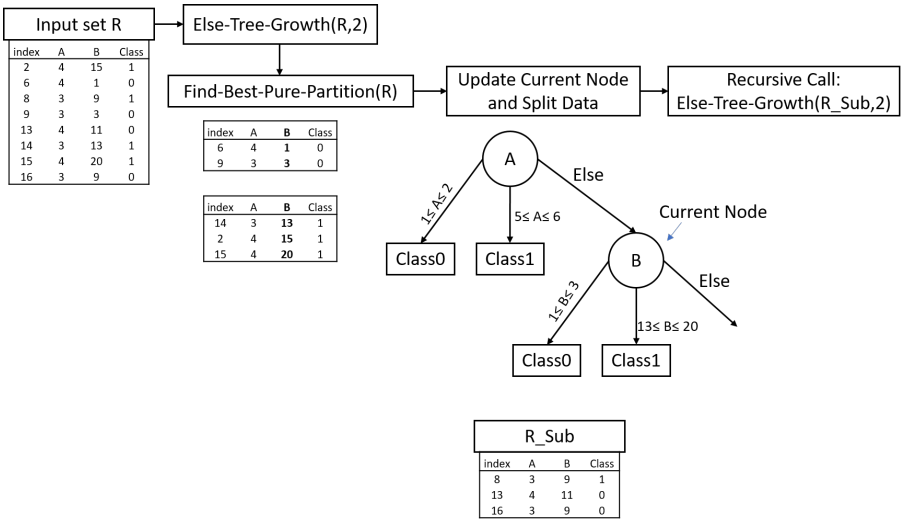


Fig. 7 An illustration of recursive iteration 2 in the Else-Tree-Growth algorithm

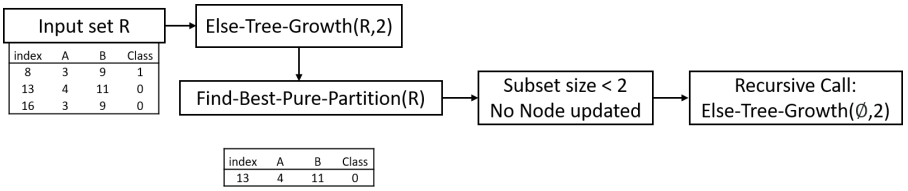


Fig. 8 An illustration of the recursive iteration 3 in the Else-Tree-Growth algorithm

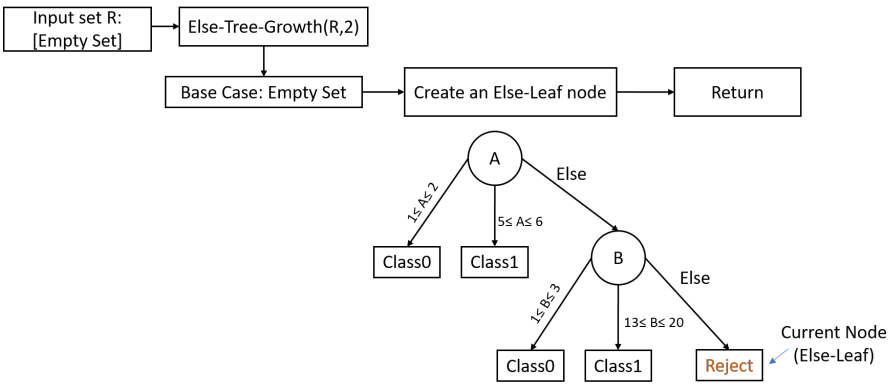


Fig. 9 An illustration of recursive iteration 4 in the Else-Tree-Growth algorithm

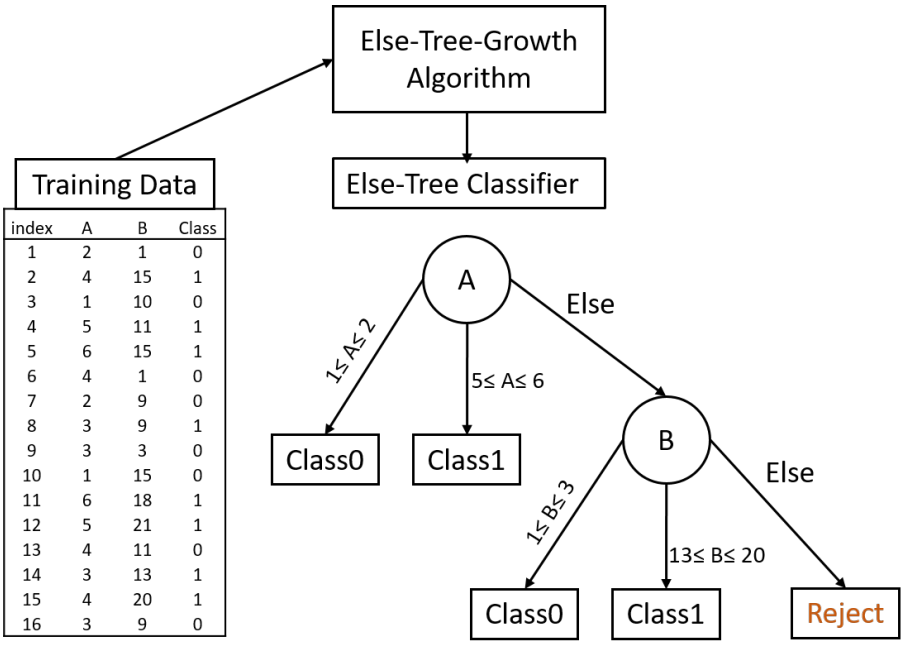


Fig. 10 An illustration of the Else-Tree model after training

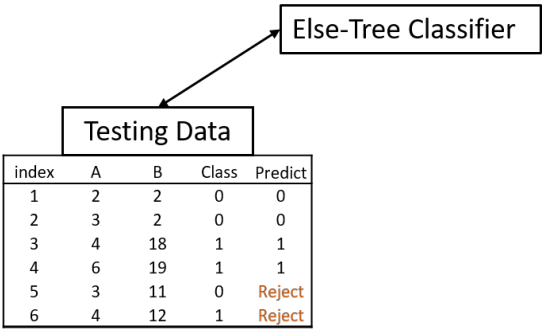


Fig. 11 An example of Else-Tree classification results on a testing dataset

	Predicted Positive	Predicted Negative	Reject
Actual Positive	True Positive (TP)	False Negative (FN)	Reject (R)
Actual Negative	False Positive (FP)	True Negative (TN)	

Fig. 12 The new confusion matrix consider number of 'reject' decisions

4 Experiments and Results

This section presents the evaluation of Else-Tree. There are several metrics that have been created to evaluate the performance of classification models. The widely used metrics are accuracy, recall, precision, F1 score, and cross-entropy. These measurements, other than cross-entropy, are calculated based on the number of correct and incorrect predictions of each class. Besides, false negative and false positive rates are commonly used in classification problems. A confusion matrix (error matrix) is a specific table representing the prediction results of a model. Our new method introduces 'reject' as an outcome of the predicted results. Hence, we present a 'new' confusion matrix and evaluation metrics to cover this outcome option. Figure 12 shows an illustration of our new confusion matrix. True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values are the same as in the typical confusion matrix. The sum of TP, TN, FP, and FN is the number of decided data instances, which are classified by the new model into a predefined class. The new confusion matrix has a column to list the number of reject cases. Some evaluation measures for the new confusion matrix are described below.

- **Number of Rejects (R):** This measure indicates the number of data items that were rejected by the classifier. The classifier has flagged those items as reject and not decided class labels for them. In the new confusion matrix, R (the number of rejects) is a part of the total number of the data, $N = TP + TN + FP + FN + R$.
- **Reject Rate:** The *reject rate* indicates the ratio of data samples that are flagged as reject by the classification model.

$$Reject\ Rate = \frac{R}{TP + TN + FN + FP + R} = \frac{R}{N} \tag{1}$$

In addition, other the measures such as correct rate, error rate, and decided rate can be used when evaluating the model as follows.

- **Correct Rate:** The *correct rate* indicates the fraction of data that are correctly classified by our method.

$$Correct\ Rate = \frac{TP + TN}{TP + TN + FN + FP + R} \tag{2}$$

- **Error Rate:** The *error rate* is the fraction of misclassified results.

$$Error\ Rate = \frac{FP + FN}{TP + TN + FN + FP + R} \tag{3}$$

- **Decided Rate:** The *decided rate* is the ratio of data samples that are decided or predicted into predefined class labels. The sum of error rate and correct rate values equals the decided rate. The reject rate and decided rate are complements of each other.

$$\begin{aligned} Decided\ Rate &= \frac{TP + TN + FN + FP}{TP + TN + FN + FP + R} \\ &= Error\ Rate + Correct\ Rate \\ &= 1 - Reject\ Rate \end{aligned} \tag{4}$$

Our goal is to reduce the error rate while having a small reject rate. In our experiments, we also consider the measurement of accuracy. It is reasonable to consider modifying the calculation of accuracy according to the new confusion matrix. However, in our method, none of the input data was originally marked or labeled as reject. The actual or true number of rejects is unknown. Hence, we do not define the "True Reject" and "False Reject" numbers in the new confusion matrix. The new method does not intend to modify a binary classification into a three-class classification problem. In the new confusion matrix, as illustrated in Figure 12, the sum of TP, TN, FN, and FP is the total number of decided data items, not the number of input data. We suggest keeping the calculation of accuracy as it is, but its interpretation is slightly different. For Else-Tree model, the accuracy, as provided in Equation 5, is the fraction of items among the decided items that are correctly classified.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{5}$$

4.1 Testing Else-Tree Classifier on Classifying Protein Crystallization Trial Images

This section describes our testing of Else-Tree classifier on classifying protein crystallization trial images. These trial images are obtained from our research on protein crystallization analysis.

Protein crystallization is the process of developing protein crystals. This process is influenced by many factors of the crystallization conditions. The crystal of a protein can be used to study the protein structure for medical purposes or drug discoveries. Many screening trials of crystallization conditions are tested to determine the right conditions to crystalize the protein. One technique to evaluate the screening trials is to capture and analyze their images. In literature, there are different methods to capture and utilize the trial images. For example, some researchers analyze the trial images taken in white light, while others use the trace fluorescent labeling (TFL) technique to capture the trial images under fluorescent light [12, 13]. Moreover, different researchers use

various feature extraction techniques and classification models to classify the trial images into different categories of the crystallization observations, such as non-crystal, crystal or tiny crystal, larger crystal, etc. [4]. It is not easy to compare the performance of these studies due to their diversity. However, they all have a certain rate of incorrectly classifying a crystallization condition.

When training an Else-Tree classifier to classify the protein crystallization trial images, the goal is to reduce the error rate to as low as zero. In addition, the reject rate should not be very high either.

4.1.1 Protein Crystallization Trial Image Dataset

We use a feature dataset, named TFL2756, extracted from 2756 images of protein crystallization trials. The images are captured using the TFL technique. Each data sample in TFL2756 contains 52 attributes. The samples belong to two classes, non-crystal (1600 samples) and leading crystal (1156 samples). The non-crystal class is the category of trial images that do not contain protein crystals. The non-crystal class indicates that the screening conditions fail to crystallize the protein. The leading crystal class is the group of images that contain either the clear or likely appearance of protein crystals. The leading crystal class indicates that the screening solution can lead to successful protein crystallization.

4.1.2 Effect of Pruning on Else-Tree Classification Performance

This part evaluates how Else-Tree pruning affects the classification performance of Else-Tree classifier. The classification performance of the C4.5 decision tree (C4.5 DT) is also included for reference purposes. Note that C4.5 DT is an extension of the ID3 algorithm [11].

Else-Tree training algorithm uses the pre-pruning method, which requires the pure subset to have a minimum number of training samples (*minNum*), as described in Section 3.4. This requirement avoids creating the classification rule for a class-leaf based on a small pure subset of training data samples. To examine the effect of the pre-pruning condition, we evaluate Else-Tree with different values of *minNum*.

The C4.5 algorithm requires that any test on the attribute at a tree node must at least split the training data samples into two subsets with a minimum number of data samples in the pure set *minNum* [11]. According to the study in [11], the default value of minNum for C4.5 is set to two. Even though pre-pruning can be used, C4.5 prefers post pruning with a given confidence level (CF) to reduce the error rate. The default value of CF is 25% [11]. In this experiment, we train C4.5 with the default value of CF and different values of *minNum*.

Table 2 summarizes the testing results. In this test, 90% of the data samples in TFL2756 are used for training, and the remaining 10% are used for testing. The parameter *minNum* is initially set to 2, then it is increased to higher

Table 2 Testing results of Else-Tree and C4.5 classifiers for the tree pruning effect

	C4.5				Else-Tree				
minNum	Error Rate	Accuracy	Tree Size	Total Leaves	Error Rate	Accuracy	Reject Rate	Tree Size	Total Leaves
2	2.54%	97.46%	39	20	3.62%	96.38%	0	56	37
13 (0.5%)	2.54%	97.46%	23	12	2.17%	97.75%	3.26%	41	25
25(1%)	2.17%	97.83%	17	9	0%	100%	15.58%	23	13
50(2%)	3.99%	96.01%	11	6	0%	100%	20.65	16	9
75(3%)	4.35%	95.65%	7	4	0%	100%	26.09%	10	6

values. As listed in Table2, *minNum* values of 13, 25, 50, and 75 equivalent to 0.5%, 1%, 2%, and 3% of the number of training data samples, respectively. For each classification model, the following testing results are collected:

- *Classification performance:* The classification results on the test dataset.
- *Tree size:* The total number of nodes on the decision tree, including decision nodes and leaf nodes.
- *Total number of leaves:* The number of leaves on the decision tree.

In Table 2, when training with higher values of minNum, both Else-Tree and C4.5 decision trees reduce their sizes. C4.5 has the testing accuracy in between 95% and 98% and the classification error is about 2% to 5%. When *minNum* equals 25, C4.5 has the highest accuracy of 97.83% and the lowest error rate of about 2.17%. When *minNum* increases to values higher than 25, the classification error of C4.5 increases. On the other hand, as *minNum* increases, the error rate of Else-Tree decreases. However, the reject rate of Else-Tree increases as *minNum* increases, as shown in Table 2. Else-Tree has about 3.62% incorrect classification when *minNum* equals two. The error rate of Else-Tree reduces to zero when *minNum* is set to 25 or higher values. Therefore, pruning is an important process for Else-Tree to reduce incorrect predictions. Hence the pruning threshold should be tested at a low value, then increase until obtaining a good balance between the error rate and reject rate of Else-Tree classifier.

4.1.3 Validation of Classification Results

This section describes the validation of Else-Tree performance on classifying the protein crystallization trial images. To validate the Else-Tree performance, we use the hold-out method to create four different pairs of training and testing sets. Information about these training and testing sets is summarized in Table 3.

- T100: The entire dataset is used for training and testing.
- T90: 90% of the data samples are used for training, and the remaining 10% are used for testing.

Table 3 Summary of the protein crystallization trial feature datasets

	Training			Testing		
Hold-out	Leading Crystal	Non-Crystal	Total	Leading Crystal	Non-Crystal	Total
T100	1156	1600	2756	1156	1600	2756
T90	1035	1445	2480	121	155	276
T80	930	1275	2205	226	325	551
T70	823	1106	1929	333	494	827

- T80: 80% of the data samples are used for training, and the remaining 20% are used for testing.
- T70: 70% of the data samples are used for training, and the remaining 30% are used for testing.

Table 4 presents the False Negative (FN) rate and False Positive (FP) rate of each experiment. The reject rate of Else-Tree is also included. The FN rate is the fraction of the 'leading crystal' images misclassified as 'non-crystal'. The FP rate is the fraction of the 'non-crystal' images predicted as 'leading crystal.' For reference, the FN and FP rates of the C4.5 decision tree and random forest (RF) classifiers are also provided. In this test, Else-Tree is pruned with *minNum* set to be 1.5% of the training size. For example, when using T80 with 2205 training data samples, the pruning threshold of Else-Tree is set to 33 (about 1.5% of 2205). The C4.5 DT is trained with the default confidence level, CF = 25%, and the default minimum number of samples, $minNum = 2$. The RF classifier is trained using 100 trees, and each tree is allowed to try with $\lfloor log_2(n+1) \rfloor$ attributes for each decision tree, where $n = 52$ is the number of attributes in TFL2756.

As listed in Table 4, , the RF classifier is able to avoid FN and FP on T100, where the whole set of TFL2756 is used for training and testing. For other tests, RF has about 0.3% to 0.8% FN and about 1.5% to 1.9% FP. The C4.5 classifier makes FN and FP predictions for all four testing sets. The highest FN rate of C4.5 is about 5% on T90. The lowest FN rate of C4.5 is about 0.2% on T100.

As shown in Table 4, Else-Tree has zero FN on all testing cases. The only case it had FP was around 0.4% on T70. It did not have any FP on T80, T90, and T100. The reject rate of Else-Tree is between 12.1% and 16.7%. These results indicate that Else-Tree can reduce the incorrect prediction when classifying the trial images of protein crystallization.

4.2 Testing Else-Tree on Public Datasets

This part describes the evaluation of Else-Tree on three datasets published in the UCI [14] data repository: Banknote (BANKNOTE), Breast cancer (BREASTCANCER), and Wireless Indoor Localization (WIL). Each data in

Table 4 Testing results with False Positive and False Negative rates for C4.5, RF, and Else-Tree classifiers when classifying protein crystallization trial feature datasets

	FP Rate			FN Rate			Reject Rate
Dataset	C4.5	RF	Else-Tree	C4.5	RF	Else-Tree	Else-Tree
T100	0.7%	0%	0%	0.2%	0%	0%	14.55%
T90	0.6%	1.9%	0%	5.0%	0.8%	0%	16.67%
T80	1.8%	1.5%	0%	2.2%	0.4%	0%	14.51%
T70	1.8%	1.8%	0.4%	0.3%	0.3%	0%	12.09%

∗: *Last column presents the Reject Rate of Else-Tree*

the WIL dataset has seven numeric attributes representing the Wi-Fi signal strengths to determine one of the four indoor locations. We chose the second location as the target class to create a binary classification problem for the WIL dataset.

In this experiment, each dataset is split into the training set (90%) and testing set (10%) using the hold-out method. For reference, we include the testing results of three traditional classification algorithms, Naive Bayes, Support Vector Machine (SVM), and C4.5 decision tree, as shown in Table 5. The parameters of C4.5 are the same as in Section 4.1.3. The SVM uses a linear kernel and a complexity parameter value of one. Overall, SVM works well for all three datasets and obtains test accuracy between 98.25% and 99.50%. C4.5 has about 93% accuracy on the BREASTCANCER data and above 98% accuracy on the WIL and BANKNOTE data. The Naive Bayes has the lowest testing accuracy of about 85.5% on the BANKNOTE data. Note that all of these three traditional classifiers make some incorrect predictions in this test.

Table 6 summarizes the test results of Else-Tree. For each dataset, Else-Tree is tested with different values of the pruning threshold *minNum*. In summary, Else-Tree can reduce the error rate to as low as zero while rejecting to make a prediction on some data samples. For the WIL dataset, Else-Tree has zero error rate and 5% reject rate with $minNum = 7$. With higher values of *minNum*, Else-Tree rejects more WIL data samples. For instance, with $minNum = 11$, the Reject Rate is about 9.5%. For the BANKNOTE dataset, the Else-Tree model with $minNum = 45$ can achieve zero error rate with about 8.7% reject rate. For the BREASTCANCER dataset, Else-Tree has about 1.75% incorrect predictions, and about 5.26% reject rate when pruning with $minNum = 5$. Else-Tree can reach zero incorrect prediction on this data with $minNum = 35$. However, in this case, the reject rate is above 31%. Note that in this experiment, the C4.5 decision tree has about 93% accuracy and 7% incorrect prediction of the BREASTCANCER dataset. The Else-Tree classifier pruned with $minNum = 5$ still makes fewer incorrect predictions than the C4.5 model.

Table 5 Testing accuracy of Naive Bayes, SVM, and C4.5 DT on WIL, BANKNOTE, and BREASTCANCER datasets

Data	Naive Bayes	SVM	C4.5 DT
WIL	97.5%	99.50%	98.00%
BANKNOTE	85.5%	98.55%	98.55%
BREASTCANCER	94.74%	98.25%	92.98%

Table 6 The testing results of Else-Tree on WIL, BANKNOTE, and BREASTCANCER datasets

Data	minNum	Accuracy	Error Rate	Reject Rate
WIL	2	98.0%	2.0%	0%
WIL	5	99.48%	0.50%	4.0%
WIL	7	100%	0	5.0%
WIL	9	100%	0	8.5%
WIL	11	100%	0	9.5%
BANKNOTE	2	98.54%	1.45%	0.72%
BANKNOTE	5	98.54%	1.45%	0.72%
BANKNOTE	25	98.54%	1.45%	0.72%
BANKNOTE	35	99.24%	0.72%	5.07%
BANKNOTE	45	100%	0	8.69%
BANKNOTE	55	100%	0	8.69%
BREASTCANCER	2	94.64%	5.26%	1.75%
BREASTCANCER	5	98.15%	1.75%	5.26%
BREASTCANCER	25	97.77%	1.75%	21.05%
BREASTCANCER	35	100%	0%	31.57%
BREASTCANCER	45	100%	0%	38.59%

5 Discussion

The Else-Tree algorithm splits the training dataset into disjoint pure subsets to create classification rules, which rely on the quality of the training data samples. As a type of a decision tree, Else-Tree is sensitive to noise. Moderate changes in the data may yield large changes in the structure of the Else-Tree model. In addition, when the training dataset is not a good representation of the testing dataset, misclassifications may occur when classifying the testing data.

The current version of Else-Tree places the else-leaf under the last decision node of the tree to indicate the reject cases. This is a reasonable structure because it allows testing all decision nodes on the upper levels before rejecting

a data item. The pruning threshold is a critical parameter that affects the classification performance of an Else-Tree model. This threshold is a pre-specified minimum number of training data samples in the selected pure subset. When the pruning threshold is set to a very low value, it can lead to a very deep tree structure with little predictive performance and possible overfitting. On the other hand, if the pruning threshold is set to a high value, only the large pure subsets are selected to create the prediction rules leading to the rejection of many data samples. Else-Tree should be tested with different pruning threshold values to create a classifier with a low error rate and low reject rate. In the future, we plan to develop an ensemble method for Else-Tree classifiers to make it more robust to noise and possibly to reduce both error rate and reject rate.

6 Conclusions

This study contributes to the area of classification problems in machine learning by developing a trustable decision tree model. The Else-Tree classifier and training algorithm demonstrates that it is possible to reduce incorrect prediction on a tree-based classification model. The Else-Tree algorithm has been justified through the classification experiments on the dataset obtained from the study of protein crystallization and three other public datasets. All experiment results show that Else-Tree can minimize the rate of misclassifications.

The novelty of Else-Tree model is the creation of *else* branch and *else-leaf* node. Else-Tree is trained to reduce error by not assigning a predefined class label to difficult-to-classify data. Instead, Else-Tree sends doubtful data samples to the else branch to find a better attribute for classification. The else-leaf node acts like a flag to indicate the case when the model does not decide a class label for the prediction data. When classifying input data, if the classification rules end up a the else-leaf node, the model determines the data sample as a reject to avoid a possible misprediction. Even though it is hard to claim that misclassifications can be avoided for all applications, this method presents a notable step in minimizing the misclassification problem. In the future, we plan to study more applications of Else-Tree and reduce the faction of rejected cases while minimizing incorrect classifications.

Declarations

- Ethics Approval and Consent to participate: Not applicable.
- Consent for publication: Not applicable
- Human and Animal Ethics: Not applicable
- Availability of supporting data: The protein crystallization trial image feature dataset is not publicly available, but are available from the corresponding author on reasonable request.
- Competing interests: No, the authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

- Funding: This research was supported by the National Institutes of Health (GM116283) grant.
- Authors' contributions: T.T. wrote the main manuscript text. All authors reviewed the manuscript. M.P. provided the protein crystallization trial data and helped with conceptualization and funding acquisition. T.T developed the methodology, designed and implemented algorithms, performed experiments, and prepared all figures and tables. R.A. contributed to the development of the methodology, supervision of the study, and updating and rewriting of the manuscript.
- Acknowledgments: This paper is an extension of our workshop paper, and we have permission from T. X. Tran, M. L. Pusey and R. S. Aygun, "Else-Tree Classifier for Minimizing Misclassification of Biological Data," 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2018, pp. 2301-2308, doi: 10.1109/BIBM.2018.8621322. The authors would like to thank Dr. Heggere Ranganath, retired professor emeritus at the Univerisity of Alabama in Huntsville, for his recommendations in this study.

References

[1] Alpaydin, E.: Introduction to Machine Learning. The MIT Press, Cambridge, MA (2014)

[2] Subedi, S., Dinc, I., Tran, T.X., Sharma, D., Shrestha, B.R., Pusey, M.L., Aygun, R.S.: Visual-x2: interactive visualization and analysis tool for protein crystallization. Network Modeling Analysis in Health Informatics and Bioinformatics **9**(1), 15 (2020). https://doi.org/10.1007/s13721-020-0220-6

[3] Tran, T.X., Aygun, R.S., Pusey, M.L.: Classifying protein crystallization trial images using subordinate color channel. In: 2017 IEEE Int. Conf. on Bioinformatics and Biomedicine (BIBM), pp. 1546–1553 (2017). https://doi.org/10.1109/BIBM.2017.8217890

[4] Tran, T.X., Pusey, M.L., Aygun, R.S.: Protein crystallization segmentation and classification using subordinate color channel in fluorescence microscopy images. Journal of Fluorescence **30**, 637–656 (2020)

[5] Tran, T.X., Aygun, R.S.: Wisdomnet: trustable machine learning toward error-free classification. Neural Computing and Applications (2020). https://doi.org/10.1007/s00521-020-05147-4

[6] Tran, T.X., Pusey, M.L., Aygun, R.S.: Else-tree classifier for minimizing misclassification of biological data. In: 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 2301–2308 (2018). IEEE

[7] Quinlan, J.R.: Induction of decision trees. Machine learning **1**(1), 81–106 (1986)

[8] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC press, ??? (1984)

[9] Mehta, M., Agrawal, R., Rissanen, J.: Sliq: A fast scalable classifier for data mining. In: Proc. Int. Conf. on Extending Database Technol., pp. 18–32. Springer, ??? (1996)

[10] Shafer, J., Agrawal, R., Mehta, M.: Sprint: A scalable parallel classi er for data mining. In: Proc. 1996 Int. Conf. Very Large Databases, pp. 544–555 (1996)

[11] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)

[12] Forsythe, E., Achari, A., Pusey, M.L.: Trace fluorescent labeling for high-throughput crystallography. Acta Crystallographica Section D **62**(3), 339–346 (2006). https://doi.org/10.1107/S0907444906000813

[13] Pusey, M., Barcena, J., Morris, M., Singhal, A., Yuan, Q., Ng, J.: Trace fluorescent labeling for protein crystallization. Acta Crystallographica Section F **71**(7), 806–814 (2015). https://doi.org/10.1107/S2053230X15008626

[14] UCI: UCI data repository. https://archive.ics.uci.edu/ml/index.php