SWITCHED DECISION TREE FOR SIMULTANEOUS LEARNING OF MULTIPLE DATASETS

DEĞİŞTİRİLEBİLİR KARAR AĞACI ALGORİTMASI İLE ÇOKLU VERİNİN EŞ ZAMANLI OLARAK ÖĞRENİLMESİ

AYÇA KULA ARSLAN

PROF. DR. MEHMET ÖNDER EFE

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

January 2024

ABSTRACT

SWITCHED DECISION TREE FOR SIMULTANEOUS LEARNING OF MULTIPLE DATASETS

Ayça Kula Arslan

Master of Science, Computer Engineering

Supervisor: Prof. Dr. Mehmet Önder Efe

January 2024, 83 pages

This paper proposes a framework to learn multiple datasets simultaneously using a single full decision tree structure. The threshold values are changed with respect to the dataset and stored in a matrix called "mask". Therefore, the full-tree model is called as a switched decision tree. First of all, solvable version of the problem is studied in order to find the decision tree parameters using a genetic algorithm. Then, the proposed algorithm is adapted for a real dataset. Obtained results demonstrate the usefulness of the algorithm for representing multiple datasets within a single switched tree structure.

Keywords: Simultaneous learning, decision trees, classification, multiple datasets, genetic algorithm, optimization

ÖZET

DEĞİŞTİRİLEBİLİR KARAR AĞACI ALGORİTMASI İLE ÇOKLU VERİNİN EŞ ZAMANLI OLARAK ÖĞRENİLMESİ

Ayça Kula Arslan

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Prof. Dr. Mehmet Önder Efe

Eş Danışman: Doç. Dr. Adı Soyadı

Ocak 2024, 83 sayfa

Bu makale, tek bir tam karar ağacı yapısını kullanarak birden fazla veri kümesini aynı anda öğrenmek için bir çerçeve önermektedir. Eşik değerleri veri setine göre değiştirilerek "maske" adı verilen bir matriste saklanır. Bu nedenle tam ağaç modeline anahtarlamalı karar ağacı adı verilir.Karar ağacı parametrelerinin genetik bir algoritma kullanılarak belirlenmesi için öncelikle problemin çözülebilir versiyonu üzerinde çalışılarak çözüm bulundu. Daha sonra önerilen algoritma gerçek bir veri seti üzerinde denedi. Elde edilen sonuçlar, algoritmanın tek bir anahtarlamalı ağaç yapısı içerisinde birden fazla veri kümesini temsil etme konusundaki kullanışlılığını göstermektedir.

Keywords: Eş zamanlı öğrenme, karar ağacı, sınıflandırma, çoklu veri seti, genetik algoritma, optimizasyon

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Prof. Dr. Mehmet Önder Efe for his comments and suggestions during my study.

I would like to thank Semih Arslan for his opinions and for his willingness to share his time with me.

I would also like to express my gratitude to my family for their moral support and warm encouragement.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

DT	:	**D**ecision **T**ree
GA	:	**G**enetic **A**lgorithm
FT	:	**F**ull **T**ree
MT	:	**M**ulti **T**ask
WL	:	**W**eak **L**earner
GB	:	**G**radient **B**oosting
GBDT	:	**G**radient **B**oosting **D**ecision **T**ree
ExtraTrees	:	**Ext**remely **Ra**ndomized **Trees**
FNR	:	**F**alse **N**egative **R**ate
FDR	:	**F**alse **D**iscovery **R**ate

1. INTRODUCTION

In the recent years, due to the increasing amount of generated data, machine learning algorithms aim to continually develop the performance of algorithms while using huge amounts of information [5]. Therefore, this paper searches for techniques to efficiently use massive data to increase performance. Learning multiple datasets is a problem in machine learning that needs improvement. Therefore, a decision tree model structure is built to learn multiple datasets simultaneously by increasing the performance.

In literature, transfer learning, multi-label learning and multi-task learning deals with learning multiple datasets. Multi-task learning involves acquiring knowledge in multiple tasks concurrently and enhancing a model's capacity to handle each task by leveraging information from either all or a subset of the tasks [6]. Faddoul et al. [7] presents a multi-task Adaboost framework that employs Multi-Task Decision Trees as weak classifiers and modifies the information gain rule used in decision trees for learning multiple tasks. Moreover, Ying et al. [8] builds a Multi-task Gradient Boosting Machine (MT-GBM) by utilizing GBDT method where the branches of the tree have been split using multi-task losses. Simm et al. [9] proposes a novel approach called MT-ExtraTrees, a tree-based ensemble method for both classification and regression, built upon extremely randomized trees. MT-ExtraTrees is described as a multi-task learning method that employs a binary decision tree ensemble. However, this paper is inspired by the idea of Cheung et al. [10] which presents a method for training a single neural network that simultaneously learns K different tasks. The same number of parameters can be used to train a single task and a multiple-task operation within a single neural network architecture.

In the literature, insufficient research has been identified to address the multi-task problem through the application of decision trees.Furthermore, decision trees are chosen for their merits, emphasizing on their advantages, and notably, the relatively limited research conducted in comparison to neural networks. Decision trees offer numerous benefits,

including robustness to outliers and straightforward interpretability. Therefore, this thesis leverages decision trees to address the challenge posed by multiple datasets.

This thesis proposes a new model to learn multiple datasets simultaneously using a switched decision tree structure. Each dataset is associated with its own specific decision tree, characterized by unique split parameters, including thresholds, features, and classes. However, in this thesis, a single full decision tree structure algorithm is obtained where the feature and classes are fixed and thresholds are switchable with respect to the dataset. The threshold values are changed with respect to the dataset and stored in a matrix called ”mask”. Using the mask structure enables the classification of different datasets within a single decision tree model. The decision tree parameters including, thresholds, features and classes are optimized using genetic algorithm for different datasets. After finalizing the full tree model, the initial step involves constructing a solvable problem to determine the optimal decision tree parameters using a genetic algorithm. Then, the framework is adapted to the real datasets. Subsequently, conclusions are drawn by evaluating performance metrics used for the classification task. The evaluation of the model's performance involves analyzing accuracy, examining cost plots, and studying confusion matrices.

1.1. Scope Of The Thesis

This thesis mainly focuses on learning different datasets at the same time using a single decision tree structure. A single full decision tree structure has been built for classifying different datasets. This full decision tree has a feature and class set that is compatible with different datasets. On the other hand, the full tree also has a threshold set that is switchable for each specific dataset. Additionally, the goal is to design a decision tree for utilizing the genetic algorithm in identifying optimal thresholds, features, and classes. Finally, the objective is to achieve satisfactory accuracy results for each dataset.

1.2. Contributions

In this research, we cover these deficiencies by proposing a novel, simple, and efficient approach. The main contributions of this paper can be summarized as follows:

- We propose a switched full decision tree model for simultaneous learning of multiple datasets.
- Unlike most of the previous works, we use the ”mask” concept where we store thresholds in an array and use each row of values with respect to the datasets.
- For the first time, a unique full decision tree model is built for multiple datasets, and by switching the thresholds of a DT with respect to the dataset the classification results are obtained.
- Our simulation results show that a single full decision tree structure can be obtained for datasets that have the same number of feature values and classes.
- The genetic algorithm is used in identifying optimal thresholds, features, and classes for each dataset.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation, contributions, and the scope of the thesis.
- Chapter 2 provides background knowledge for decision trees and genetic algorithms in order to understand the proposed method.
- Chapter 3 gives a brief literature review of the works that are related to the thesis topic.
- Chapter 4 introduces the proposed method.
- Chapter 5 demonstrates the results of the thesis.
- Chapter 6 states the summary of the thesis and possible future directions.

2. BACKGROUND OVERVIEW

In this section, decision tree and genetic algorithm are explained in detail in order to understand the proposed method. The subjects are explained with simple examples, flowchart or algorithm.

2.1. Decision Tree

A decision tree is a supervised machine learning classifier that has a hierarchical data structure which represents the data and organizes decisions. Decision trees can be used both in classification and regression problems. A decision tree operates as a predictor and represents a map $h : X \rightarrow Y$, by foretelling the assigned label y for a given instance x as it traverses from the tree's root node to a leaf node [11]. It maps the features $x \in X$ of a data point to a predicted label $h(x) \in Y$ [12].

Decision trees can handle both classification and regression tasks. A classification tree is used for classification tasks where the objective is to assign instances to predefined classes or categories. On the other hand, regression trees aim to predict continuous numerical values. Regression trees are employed when dealing with continuous outcomes, with the primary objective being the prediction of numeric values. In this thesis, we will mainly focus on classification trees.

An example of a simple classification using a decision tree is illustrated in figure 2.1. To check if the fruit is mandarin, green plum or watermelon, the decision tree first examines the color of the fruits. If this color is orange, then the tree immediately predicts that the fruit is an orange. Otherwise, the tree turns to examine the sizes of the fruits. If the size of the fruits are small, the decision tree predicts that the fruit is a green plum. Otherwise, the prediction is watermelon.

Decision trees consist of root node, internal nodes, and leaf nodes. The top-level node is the root node that has no parent nodes, which means that it has no incoming edges. The leaf

nodes represent the class labels and these nodes do not have any child nodes (no outgoing edges). The internal nodes are the ones that are between the root and leaf nodes. An internal node might represent a test on the attribute and the branches would lead to different outcomes based on whether the test is true or false.

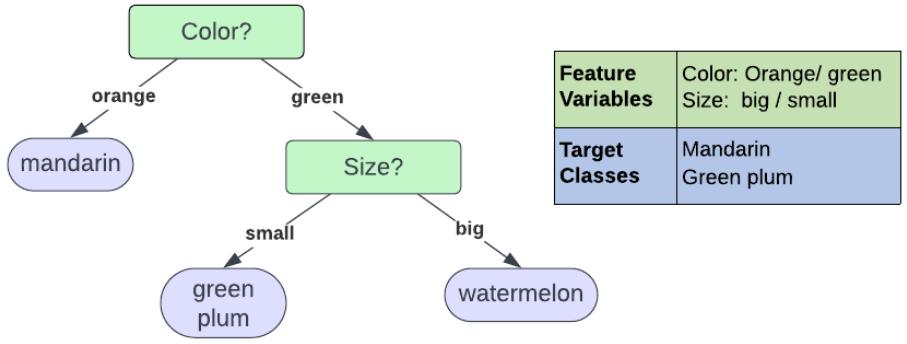

Figure 2.1 Simple Decision Tree

The depth of the tree, the total number of nodes, leaves, and the number of attributes are used as a measure of the complexity of the tree [13]. The complexity of the tree is important because it affects the accuracy of the decision tree [14]. As a result, it is better to build a simpler decision tree with the help of pruning and stopping criteria. Moreover, performance of the DT is directly proportional to the size of the training data. However, an increase in training data leads to a long training time. In practical applications, the most apparent factor is the constraint on resources; thus, training data should be limited in accordance with these resource constraints.

According to [15], it is more efficient to utilize decision trees for problems characterized by the features described below.

- *Attribute-value representation for instances*

  In figure 2.1, for decision tree learning the decision tree is represented by the attributes (color, size) and its values (orange, small).

- *Discrete output target values*

  It is less common to train models on learning target functions with real-valued

values. The more prevalent scenario involves the learning of discrete classifications or categorical outcomes.

- *Training data that includes errors or missing attribute values*

  Decision trees are known for their robustness to errors in the training set. They can effectively handle imperfect data, and their inherent structure often allows them to adapt to variations and outliers without significantly compromising their overall performance. Decision trees are generally capable of handling training data with missing values for certain attributes. A common approach is to estimate the missing attribute values based on the information available from other examples. Assigning the most common value among the training examples at a particular node to fill in missing attribute values is a common strategy. Decision trees can naturally navigate and make decisions based on the available information, allowing them to accommodate datasets with missing values without compromising their overall effectiveness in learning and making predictions.

- *Contains disjunctive expressions*

  Broadly speaking, decision trees encapsulate a series of conjunctions of conditions related to the attribute values of instances. Every route from the tree's starting point to its terminal leaf signifies a specific conjunction of attribute evaluations, with the overall tree embodying a collection of these conjunctions presented as a disjunction. For example figure 2.1, the watermelon leaf corresponds to $(Color = green) \wedge (Size = big)$.

Decision tree induction is building a decision tree based on a provided dataset [16]. Decision tree induction algorithms play a crucial role in diverse domains, including medicine, manufacturing, financial analysis, astronomy, molecular biology etc. [17]. These algorithms have gained popularity due to their numerous advantages. The advantages can be listed as follows:

1. **Comprehensible and easy interpretation:** Decision tree representation is particularly considered user-friendly, as even a compact decision tree can be visualized, utilized,

and comprehended by individuals without specialized expertise. The ability to convert a decision tree into a set of rules enhances its explanatory power.

2. **Fast algorithm speed:** Compared to neural networks and the k-nearest neighbor, the speed of the decision tree algorithm is high [18].

3. **Handle both numeric and categorical input attributes:** Decision trees stand out for their versatility in handling both numerical and categorical attributes, offering an advantage over classification methods like neural networks, k-nearest neighbor, and support vector machines, which may face challenges in directly managing categorical data [19].

4. **Tolerant to missing values:** Decision trees demonstrate the ability to handle datasets that may contain missing values.

On the other hand, decision trees come with certain disadvantages, such as:

1. **Prone to overfitting:** The greedy approach of decision trees, can be a drawback as it tends to cause over–sensitivity to the training set, to irrelevant attributes and noise [20]. Thus, this may lead to overfitting.

2. **Sensitive to changes in data:** Indeed, any modification to the training data leads to a shift in attribute selections, consequently affecting the entire structure of the tree [21].

3. **Induction time and data diversity correlation:** The induction time is directly proportional to the data diversity. The increase in data diversity may be associated with the high number of attribute values. Therefore, this situation will lead to an increase in the number of possible splits. Potentially leading to a notable impact on the time required for the induction process.

In summary, despite disadvantages, decision trees offer many advantages, such as simplicity, interpretability, and effectiveness. Due to this reason, decision tree becomes a valuable method that is used in data analysis and knowledge discovery in various domains.

2.2. Algorithm for Decision Trees

The procedure of constructing a classifier by deriving the structure of a decision tree is commonly referred to as decision tree induction. There are many decision trees algorithms such as ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), CART (Breiman et al., 1984) which can be briefly described as:

1. **ID3 (Quinlan, 1986):** ID3 is the most straightforward decision tree algorithm, employing information gain as its criteria for splitting [16]. ID3 does not incorporate any pruning procedures, and it does not manage numeric attributes or missing values.

2. **CART (Breiman et al., 1984):** In classification and regression trees (CART), the decision tree splitting is based on the towing criteria. It can manage both numerical and categorical attributes.

3. **C4.5 (Quinlan, 1993):** The C4.5 algorithm is a more advanced version compared to ID3, capable of handling numeric attributes and addressing missing values in the data. Additionally, the gain ratio criterion is employed instead of information gain for splitting the decision tree.

These decision tree algorithms are summarized in table 2.1.

	Splitting Criteria	**Prunning Method**	**Attribute Type**	**Missing Values**
ID3	Information Gain	No prunning	Categorical value	Not accepted
CART	Towing	Cost-complexity pruning	Categorical and numeric value	Accepted
C4.5	Gain ratio	Error based pruning	Categorical and numeric value	Accepted

Table 2.1 Summary of decision tree inducers

A typical algorithmic framework for top–down inducing of a decision tree can be written as in algorithm 1. The algorithm is obtained using [22] and [13]. The algorithm mainly consists of two classes:

- **Node Class:** The ”CreateNode” class in algorithm 1, adds a new node to the decision tree. In the algorithm, it can be seen that a node consists of a label and a test condition. If the node is not a leaf node, each node has its feature, threshold values and information about its branching left and right nodes. Otherwise, the node has a non-zero leaf node value. All parameters are initially set to None.

- **Decision Tree Class:** Generally, a decision tree class holds many functions related with finding the best split, finding entropy, information gain and most common label, predicting the classes, recursively growing tree function etc. The constructor of decision tree holds values for minimum sample split and maximum depth which are the hyper-parameters. The minimum sample split is used to define the minimum number of samples required to split a node. Both are used in recursive functions as exit conditions.

The functions are shown in capital letters in the algorithm. If the stopping condition is satisfied, it returns a leaf node (line 1-2). The leaf node is characterized by the most prevalent class within the existing subset of data (line 3). However, if the stopping condition is not satisfied, the algorithm searches for the best split (line 5-7) and creates a new node (line 6). The algorithm explores all viable splitting options and identifies the most advantageous one. Then, the training data is split into subsets. The resulting data subsets obtained from splitting rule iteratively invokes the construction of subtrees using recursive TreeGrowth call (lines 8-9-10). These subtrees are then appended as children to the primary node, resulting in the formation of a comprehensive tree structure (line 10). The culmination of this process is the generation of a decision tree, which serves as the output of the function.

Algorithm 1 Algorithm for Full Decision Tree Model [13]

Input: Training set *S*, attribute set *A*

Output: Decision tree structure

TreeGrowth(S,A) :

1: **if** StoppingCond(S,A) = true **then**
2: *leaf* = CreateNode()
3: *leaf.label* = Classify(S) # Mark the root node in T as a leaf with the most common value of y in S as a label
4: **return** *leaf*
5: **else**
6: *root* = CreateNode()
7: *root.test_cond* = FindBestSplit(S,A)
let V = { *v* | *v* is a outcome of *root.test_cond* }
8: **for** each v ∈ V **do**
9: S_v = {s | root.test_cond(e) = v and s ∈ S }
10: child = TreeGrowth(S_v,A)
add *child* as descendent of *root* and label the edge *(root → child)* as *v*
11: **end for**
12: **return** $root$
13: **end if**

2.2.1. Splitting Criteria

How to split the decision tree is an important question that needs a solution. It is important to determine the attribute that causes a split. Given a training set S and target attribute y, internal node is split with respect to the value $v_{i,j}$ of a single attribute a [13]. Some of impurity based measures are information gain and gini index. There are many methods for splitting, such as information gain, gini index, gain ratio, DKM Criterion, Likelihood Ratio Chi-squared Statistics etc.

In order to split the training data, an attribute has to be chosen as a test condition [22]. For different attribute types (Binary, Nominal, Ordinal, Continous attributes), an appropriate test condition must be chosen.

2.2.1.1. Information Gain Entropy is a measure of impurity in a distribution and evaluates the homogeneity of a dataset. Meaning that for more homogenous dataset, lower entropy is necessary. Lower entropy is desirable because it indicates a more homogeneous subset of data in terms of class labels. For example, if the entropy is 0, it indicates that all members of the training set belong to the same class.

Information gain shows the decrement of entropy after the split of a specific attribute and chooses the best feature for decision-making. When the dataset is split by a specific feature, the high information gain value tells that the feature is a good choice for splitting the node. The information gain formulation uses entropy as:

$$\text{Information Gain}\,(a_i, S) = \text{Entropy}(y, S) - \sum_{v_{i,j} \in \text{dom}(a_i)} \frac{\left|\sigma_{a_i = v_{i,j}} S\right|}{|S|} \cdot \text{Entropy}\left(y, \sigma_{a_i = v_{i,j}} S\right) \tag{1}$$

where entropy is defined as:

$$\text{Entropy}(y, S) = \sum_{c_j \in \text{dom}(y)} -\frac{\left|\sigma_{y = c_j} S\right|}{|S|} \cdot \log_2 \frac{\left|\sigma_{y = c_j} S\right|}{|S|} \tag{2}$$

Entropy and information gain can be calculated from equation 1 and 2, respectively. For example, if we consider a data given as $s = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1]$ where there are two labels (”0” and ”1”). We may calculate entropy as:

$$E(S) = -\frac{7}{10} \log_2 \frac{7}{10} - \frac{3}{10} \log_2 \frac{3}{10} = 0.8813 \tag{3}$$

Moreover, the information gain as:

$$\text{Gain}(S, A) = 0.97095 - \frac{12}{20} * 0.65002 - \frac{8}{20} * 1 = 0.18094 \tag{4}$$

2.2.1.2. Gini Index Gini index takes values between 0 and 1, where 1 indicates inequality. A smaller Gini index indicates a purer distribution. Gini index quantifies the difference in probability distributions among the values of the target attribute [23] which is calculated as:

$$\text{Gini}(y, S) = 1 - \sum_{c_j \in \text{dom}(y)} \left(\frac{\left|\sigma_{y=c_j} S\right|}{|S|} \right)^2 \tag{5}$$

$$\text{Gini Gain}\,(a_i, S) = \text{Gini}(y, S) - \sum_{v_{i,j} \in \text{dom}(a_i)} \frac{\left|\sigma_{a_i=v_{i,j}} S\right|}{|S|} \cdot \text{Gini}\left(y, \sigma_{a_i=v_{i,j}} S\right). \tag{6}$$

2.2.1.3. Gain Ratio According to Quinlan [24], the gain ratio gives better performance results when compared to simple information gain criteria. Gain ratio is found by normalizing the information gain as [16]:

$$\text{Gain Ratio}\,(a_i, S) = \frac{\text{Information Gain}\,(a_i, S)}{\text{Entropy}\,(a_i, S)} \tag{7}$$

2.2.2. Stopping Condition

The decision tree algorithm splits the data and obtains the leaf nodes recursively. In algorithm 1, the stopping condition is shown as "StoppingCond()". However, when the stopping criteria are met then the tree-growing process stops. There are several ways to define the stopping criteria as[20]:

- All training data have the same class label or attribute values
- When the maximum tree depth is reached
- Defining the minimum number of instances required to establish a terminal node (leaf) deployment

- Defining minimum number of samples for a node split
- When the best splitting criterion is not greater than a certain threshold

2.2.3. An Illustrative Example for Splitting

In table 2.2, a tennis dataset has been provided, which is one of the most widely used datasets. The dataset illustrates decisions related to playing tennis based on varying weather conditions. The decision tree obtained is an example of boolean classification, as the target values are either 'yes' or 'no'. The objective of this example is to construct a simple decision tree and use information gain to choose the attributes. The training set S contains a total of 14 data points, comprising 5 days when tennis is not played and 9 days when tennis is played. This distribution leads to an entropy value, which is then used in the calculation of information gain. The entropy value is:

$$\begin{aligned} \text{Entropy}\left([9+,5-]\right) &= -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) \\ &= 0.940 \end{aligned} \tag{8}$$

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overeast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Table 2.2 Weather Conditions and Tennis Playing Decision [1]

In Equation 1, the first term represents the entropy of the training set S, while the second term represents an entropy estimate after splitting the training set S with respect to attribute a. The information gains for each attribute (Outlook, Humidity, Wind) need to be calculated, and the attribute with the highest information gain should be selected as the first test for the decision tree. If the first attribute is chosen as Outlook, the entropy values need to be calculated for each value of Outlook (Sunny, Overcast, Rain) as follows:

$$\begin{aligned} \text{Values(Outlook)} &= \text{Sunny, Overcast, Rain} \\ S_{\text{Sunny}} &:[2+, 3-] \Rightarrow \text{Entropy } \left(S_{\text{Sunny}}\right) = 0.971 \\ S_{\text{Overcast}} &:[4+, 0-] \Rightarrow \text{Entropy } \left(S_{\text{Overcast}}\right) = 0 \\ S_{\text{Rain}} &:[3+, 2-] \Rightarrow \text{Entropy } \left(S_{\text{Rain}}\right) = 0.971 \end{aligned} \tag{9}$$

$$\begin{aligned} \text{Gain}(S, \text{ Outlook }) &= \text{Entropy } (S) - \sum_{i \in \mid \text{ Sunny, Overcast, Rain} \mid} \frac{|S_v|}{|S|} \text{Entropv } (S_v) \\ &= \text{Entropy } (S) - (5/14) \text{ Entropy } \left(S_{\text{Sunny}}\right) - (4/14) \text{ Entropy } \left(S_{\text{Overcast}}\right) \\ &\quad - (5/14) \text{ Entropy } \left(S_{\text{Rain}}\right) = 0.246 \end{aligned} \tag{10}$$

When we continue to do the same calculations for other attributes, we find that Gain $(S,$ Outlook $) = 0.246$, Gain $(S,$ Humidity $) = 0.151$, Gain $(S,$ Temperature $) = 0.029$. Next, we choose the attribute with the highest information gain, which, in this case, is Outlook.

Having identified the root node, it is necessary to add additional nodes beneath the root node (Outlook). In figure 2.3, the next attribute has to be chosen. All examples of outlook = overcast belong to the category of 'yes' for playing tennis. Therefore, the nodes belong under rain and sunny should be found. Because, there are both "yes and "no" samples.

For the S_{sunny} subtree, the information gain is found for each attribute. First, the temperature attribute is tested and the information gain is found as:

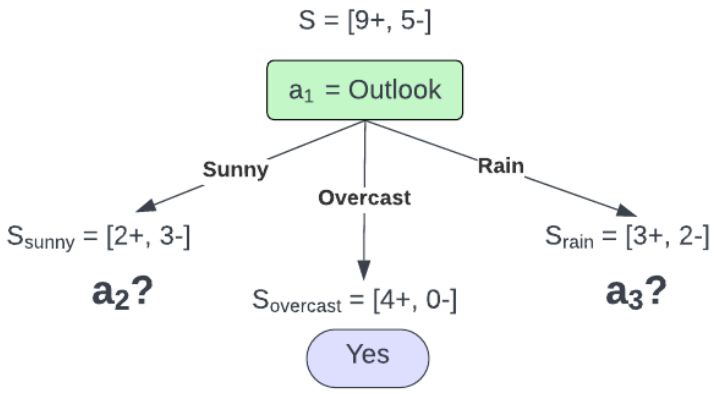

Figure 2.2 Selection of next attribute for play tennis example

$$
\begin{aligned}
&\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = S_{\text{sunny}} - \left[\frac{2}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Temperature=hot})\right. \\
&\left.+\frac{1}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Temperature=cool}) + \frac{2}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Temperature=mild})\right] \\
&= S_{\text{sunny}} - \left[\frac{2}{5}\text{Entropy}\left(\frac{0}{2}, \frac{2}{2}\right) - \frac{1}{5}\text{Entropy}\left(\frac{1}{1}, \frac{0}{1}\right) - \frac{2}{5}\text{Entropy}\left(\frac{1}{2}, \frac{1}{2}\right)\right] \\
&= 0.971 - \left(\frac{2}{5} \times 0 + \frac{1}{5} \times 0 + \frac{2}{5} \times 1\right) \\
&= 0.971 - 0.4 = 0.571
\end{aligned}
\tag{11}
$$

For ”humidity” attribute the information gain is found as:

$$
\begin{aligned}
&\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = S_{\text{sunny}} - \left[\frac{3}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Humidity=high})\right. \\
&\left.+\frac{2}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Humidity=normal})\right] \\
&= S_{\text{sunny}} - \left[\frac{3}{5}\text{Entropy}\left(\frac{0}{3}, \frac{3}{3}\right) - \frac{2}{5}\text{Entropy}\left(\frac{2}{2}, \frac{0}{2}\right)\right] \\
&= 0.971 - \left(\frac{3}{5} \times 0 + \frac{2}{5} \times 0\right) \\
&= 0.971
\end{aligned}
\tag{12}
$$

The information gain for ”wind” can be written as:

$$\begin{aligned}
&\text{Gain}(S_{\text{sunny}}, \text{Wind}) = S_{\text{sunny}} - \left[\frac{3}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Wind=weak})\right. \\
&\left.+\frac{2}{5}\text{Entropy}(S_{\text{sunny}} \wedge \text{Wind=strong})\right] \\
&= S_{\text{sunny}} - \left[\frac{3}{5}\text{Entropy}\left(\frac{1}{3}, \frac{2}{3}\right) - \frac{2}{5}\text{Entropy}\left(\frac{1}{2}, \frac{1}{2}\right)\right] \\
&= 0.971 - \left(\frac{3}{5} \times 0.918 + \frac{2}{5} \times 1\right) \\
&= 0.971 - 0.951 = 0.02
\end{aligned} \tag{13}$$

The attribute ”humidity” yields the maximum information gain for the ”sunny” subtree. Similar calculations should be performed for the ”rain” subtree as well. The final decision tree structure has been obtained as in figure 2.3.

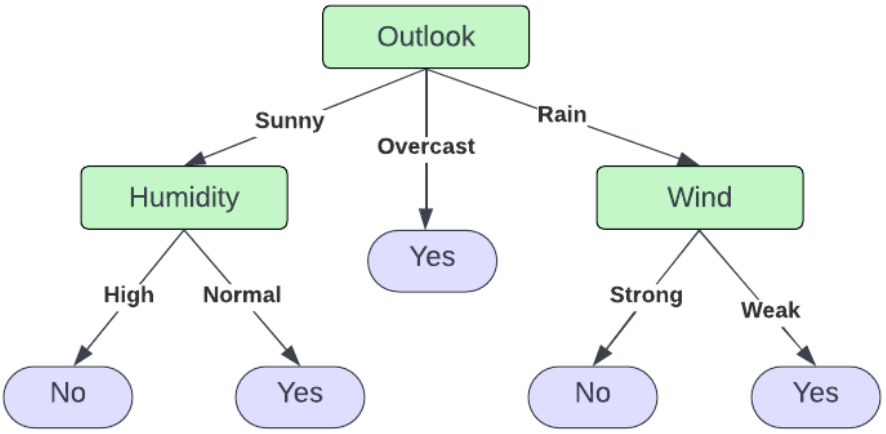

Figure 2.3 Decision tree for play tennis example

2.3. Genetic Algorithm

The genetic algorithm (GA) is a method inspired by biological evolution that is used to solve search and optimization problems[25]. GA’s key elements are that it is a stochastic algorithm (randomness) and in each iteration, there is a population of solutions [26]. The population

constitutes of solutions within the current generation and can alternatively be characterized as a collection of chromosomes as seen in figure 2.4. Chromosome is an individual in population which is defined by a group of parameters, commonly referred to as genes. The chromosomes represents a solution.

Figure 2.4 Population, chromosome and gene

GA is a sub-class of the evolutionary algorithms therefore it uses the terminologies selection, crossover (also called recombination), and mutation. The primary motivation for employing genetic algorithm in this paper is that, in comparison with other traditional methods, it can enhance the classification accuracy in decision trees. Papagelis et al. [27] states that, GA can be used to evolve binary decision trees in order to create simple and accurate decision trees. Moreover, it states that GA chooses the most appropriate features for classification tasks. Moreover, papers [28] and [29] also used genetic algorithm for building decision trees.

The basic flow chart of a genetic algorithm can be shown as in figure 2.5. The algorithm can be summarized as [30]:

1. The cost function and the genetic algorithm variables are defined. Moreover, to converge faster the constraints are defined for some variables.
2. An initial population is randomly created.
3. Next population is created using individuals from the current generation. At each generation the new populations are created as:

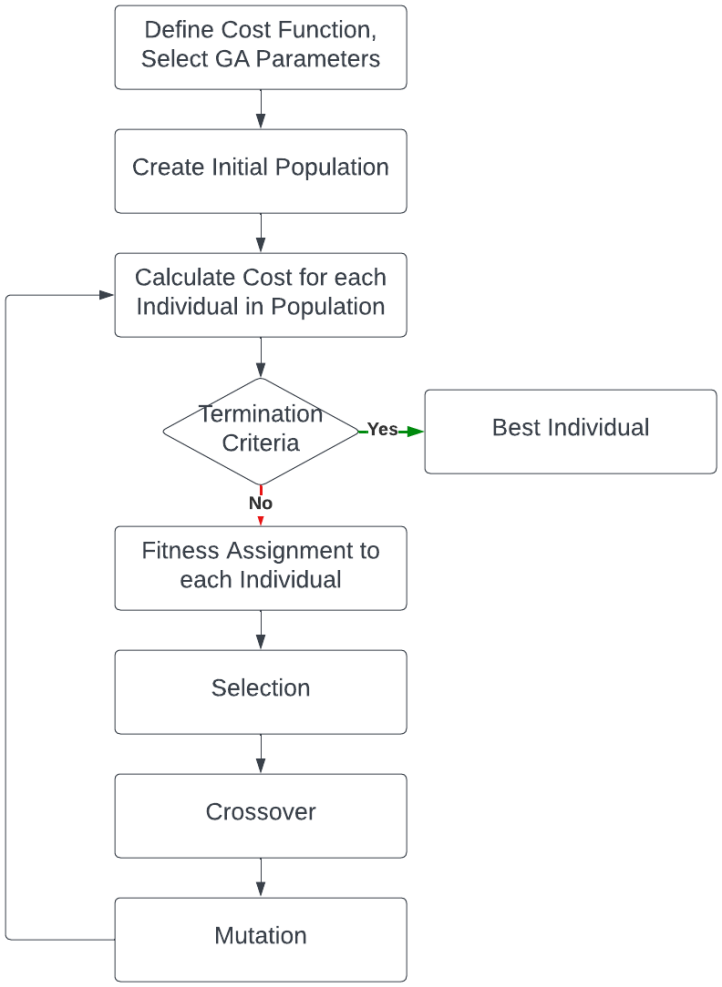

Figure 2.5 Flow chart for genetic algorithm[2]

(a) Each individual's score is computed using its fitness values in a population where these values are called "raw fitness scores". Then, these values are scaled to obtain improved results and values are referred to as "expectation values".

(b) The selection of parents is determined by the individual's "raw fitness scores" or "expectation values."

(c) Elite individuals, possessing the highest fitness or cost values, are directly carried over to the next generation of population. Elite children refers to the succeeding individuals present in the next generation.

(d) Children are obtained using crossover and/or mutation.

(e) The next population/generation is produced by substituting children from to the current population.

4. When the stopping criteria is satisfied, the genetic algorithm concludes its execution.

2.3.1. Components of a Genetic Algorithm

The basic process for a genetic algorithm consists of the selection of parents, generating new children (offsprings) by crossing parents, and then switching old ones with new children in the population [26].

2.3.1.1. Selection Based on the fitness values, the parents or chromosomes are selected from the population. Those who have appropriate fitness values play an active role in the reproduction process [2]. Using the evaluation function the chromosomes are selected from the population. According to [31], " *The fitter the chromosome, the more times it is likely to be selected to reproduce.*". This means that a higher fitness function causes the individual to be selected. This situation affects the convergence rate of GA where higher fitness values relate to higher convergence rates.

There are many methods used for selection such as:

- **Roulette Wheel (Fitness proportionate) Selection :** This selection method is inspired by the game of roulette. In roulette wheel selection, the probability of choosing a population member is linearly proportional to its fitness value [32]. This means that individuals with higher fitness scores have a greater chance of being selected as a parent. The wheel is turned repeatedly to choose additional individuals, gathering a sufficient number of individuals to constitute the next generation, which may lead to the possibility of selecting the same individual more than once.

- **Tournament Selection:** The tournament size is the number of randomly selected individuals from the population. After randomly choosing the tournament size of

individuals, the member with the best fitness is selected. Since sorting takes a significant amount of time, tournament selection is useful for large population sizes.

Random, uniform, rank-based and Boltzman selection are some of the different selection methods.

2.3.1.2. Crossover (Recombination) The parts in each parent's encoded string are exchanged to form children for the next generation. Randomly each parent is chosen for mating (from the mating pool). The crossover point is defined as a randomly selected one or more points in parent chromosomes. These chromosome parts are exchanged with respective to the crossover points. By using the crossover operation, offspring contain some knowledge from each of the parents. The crossover probability defines the frequency of occurrence of crossover. For example, a hundred percentage of crossover means the all children are obtained by crossover. There are many crossover methods such as ordered crossover, partially matched crossover, etc. However, the most general used ones are:

- **Single Point Crossover:** A single point is randomly chosen as a crossover point. Using this point the offspring are obtained by exchanging the parts after the crossover point as shown in figure 2.6. For example, if a parent chromosome contains good genetic knowledge on the head and after the crossover point, each offspring would not have the good attributes together. Therefore, usually two-point or uniform crossover is used to eliminate this negative effect. The outcome of an N-point crossover is directly influenced by the gene placements within the chromosome.

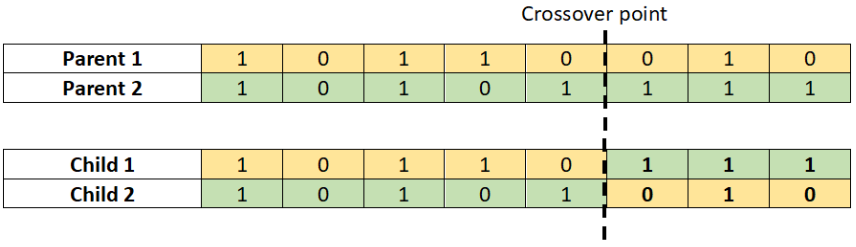

Figure 2.6 Single point crossover

- **Two Point and Multi-Point Crossover:** If the crossover point is more than one, the crossover is defined as a multi-point crossover. While obtaining the offspring, the chromosome parts that lie between these randomly chosen crossover points are swapped. An example of a two-point crossover can be seen in figure 2.7.

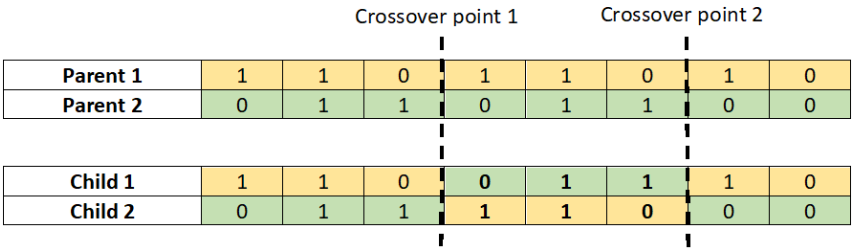

Figure 2.7 Two point crossover

- **Uniform Crossover:** In uniform crossover, some bits are randomly selected and parents are combined at these bits as seen in 2.8. In previous crossover methods, the probability of passing neighbor genes is higher. Moreover, passing neighbor genes can cause undesirable correlations within genes [26]. Therefore, to eliminate this drawback related to gene placements, uniform crossover is recommended.

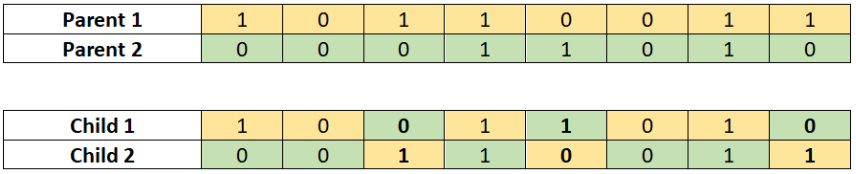

Figure 2.8 Uniform crossover

2.3.1.3. Mutation Mutation is followed after crossover and a mutated child is obtained by randomly changing a bit string of a single parent chromosome [1]. Mutation helps explore all the search space and thus safeguards the algorithm from getting stuck in a local minimum. Also, mutation creates a genetic variation within the population. The mutation operator

works on a single individual or parent, and introduces random changes such as flipping a bit in a binary representation or modifying a value in a numerical representation. The mutation methods can be classified as:

- **Bit Flip Mutation** In the context of bit flip mutation which is employed in binary-encoded Genetic Algorithms, random genes or bits are chosen, and their values are inverted. As illustrated in the figure 2.9, the 0 bit is switched to 1.

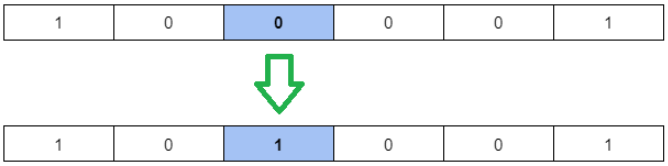

Figure 2.9 Bit flip mutation

- **Swap Mutation** The swap mutation technique is mostly used in permutation encoding where each chromosome shows sequence of numbers.In swap mutation, a pair of genes on the chromosome are randomly chosen and their values are exchanged with each other as seen in figure 2.10.

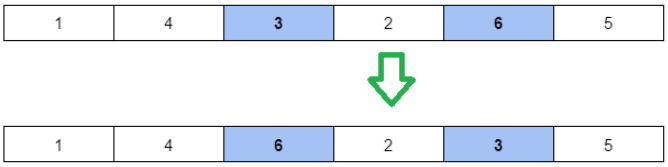

Figure 2.10 Swap mutation

- **Scramble Mutation** Scramble mutation is employed in permutation encoding. In this approach, a subset of genes is randomly selected and then subjected to a random shuffling process as seen in figure 2.11.

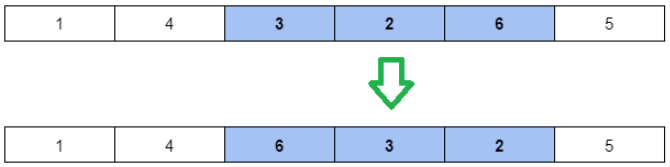

Figure 2.11 Scramble mutation

2.3.2. Selection of Genetic Algorithm Parameters

Choosing GA parameters is a challenging endeavor. Random number generations also affect GA parameter selection to be challenging. Random number generators are primarily used in generating the initial population, performing mating, and managing mutation in genetic algorithms [25]. It is a hard task to test a wide range of intervals for each genetic algorithm parameter. DeJong's investigation into genetic algorithms for function optimization involved a series of parametric studies across a five-function suite of problems. These studies indicated that achieving good performance with genetic algorithms necessitates selecting a high crossover probability, a low mutation probability (inversely proportional with the population size), and a moderate population size[33]. In order to run the algorithm efficiently, certain key points must be taken into account:

- When there is a rough idea of where the solution to a problem might be located, it is advisable to specify the initial population range or search space so that the best estimate of the solution can be reached [34]. This can help narrow down the search and improve the efficiency of the optimization process. If the problem is small and easy it is good to choose a small population size. However, if the problem is complex small population size would not be sufficient. As a side note, choosing a large population size increases the computational time. Moreover, increased population size is directly correlated with an increased probability that the initial population state will include a chromosome presenting the optimal solution [35]. Goldberg's research has demonstrated that the

efficiency of genetic algorithms in converging towards global optima, rather than local ones, is predominantly influenced by the population size [26].

- During each iteration the percentage of mutated bits in the population is commonly referred to as the ”mutation rate.” The genetic algorithm may transform into a random search if the mutation rate is excessively increased and therefore, the diversity of the population is large [32]. Generally, as the population size increases the mutation rate should be decreased. The range for mutation rate is $[0, 1]$.

- Crossover rate represents the number of how much crossover happens among chromosomes within a single generation [36]. The crossover rate takes values between $[0, 1]$. Higher crossover rates contribute to the diversity of the population since new chromosomes are produced, while a diminished crossover rate, in contrast, results in insufficient production of new offspring. According to [25], crossover rate, selection, and crossover methods have less impact on solution convergence when compared with parameters such as population size and mutation rate.

- If elitism is used in the algorithm, it improves the performance. In elitism, a new population is obtained by using the first best or few best chromosomes. Therefore, in each iteration, the top-performing individuals are enforced to stay.

Mutation rate and population size have a critical effect in finding the global minimum in a problem. The number of generations is another hyperparameter. Also, the stopping criteria is a hyper-parameter that is mentioned in the later section.

2.3.3. Search Termination (Convergence Criteria)

The genetic algorithm continues to iterate until the stopping criterion has been reached and therefore, the best individual in the population is found. When the optimal solution is reached the genetic algorithm stops. The category of stopping condition belongs to the direct termination criteria when the predefined condition is met without using any information

derived from the ongoing evolutionary search process [37]. Some of the stopping conditions, which belongs to the category of direct termination, can be defined as:

- **Maximum number of generations**: The algorithm has attained the predefined upper limit on the number of generations. Furthermore, defining this limit effectively restricts the algorithm's run-time and mitigates the consumption of computing resources[32]. GA may endlessly continue to work if the maximum number of iterations is not defined and also other stopping conditions are not satisfied.
- **Stall generations**: The algorithm ceases its operation when there is no enhancement observed in the objective function over a continuous sequence of generations, where the length of this sequence is defined as "Stall generations" [26]. The genetic algorithm will terminate if the population's best fitness remains unchanged for a predetermined number of generations.

A variety of termination criteria may be specified. Moreover, the determination of the optimal number of iterations in a genetic algorithm can be accomplished through a convergence analysis, which can be approached from various perspectives, including scheme theory or Markov chains[38]. If the genetic algorithm does not reach the optimum value, first try changing the population size and mutation rate.

2.3.4. Fitness Function

In every step of the algorithm, the individuals undergo evaluation using a fitness function. The input to the cost function is a chromosome that has to be optimized. The chromosome consists of $Nvar$ variables where for every variable the cost is evaluated using cost function f [25]. Finally, every chromosome will have a cost defined as:

$$\text{cost} = f(\text{ chromosome }) = f\left(p_1, p_2, \ldots, p_{N_{\text{var}}}\right) \tag{14}$$

2.4. Performance Metrics for Classification

In order to effectively show the success of machine learning models, it is crucial to conduct demonstrative presentations or evaluations. For this reason, various performance metrics are suggested in the literature to quantify and evaluate the effectiveness of machine learning models. The performance metric should be selected considering the characteristics of the problem[39]. In determining a suitable performance metric, one must take into account not only the unique characteristics of the task but also the data distribution. The efficiency of the trained classifier should be assessed by testing it on previously unseen data. Subsequently, selecting the relevant performance metric and summarizing the classifier's performance became important [40].

2.4.1. Confusion Matrix

A confusion matrix provides a comprehensive summary of a classifier's performance in classifying test data, outlining the distribution of predicted and actual class labels [41]. The confusion matrix, reveals both accurately and inaccurately classified instances across all classes.

A simple confusion matrix can be defined for a binary classification problem as in table 2.3. In a binary classification scenario, where the target classes are either positive or negative, the corresponding confusion matrix is a 2x2 matrix. The matrix consists of true positive (TP), false positive (FP), false negative (FN), true negative (TN).

		Actual Values	
		Positive (P)	Negative (N)
Predicted Values	Positive (P)	True Positive (TP)	False Positive (FP)
	Negative (N)	False Negative (FN)	True Negative (TN)

Table 2.3 Confusion Matrix

The components of the confusion matrix are:

1. **True Positive (TP):** The model accurately identifies and classifies a certain number of instances as positive that are indeed positive.

2. **True Negative (TN):** The model accurately identifies and classifies a certain number of instances as negative that are indeed negative.

3. **False Positive (FP):** The model incorrectly identifies and categorizes a certain number of instances as positive that are actually negative.

4. **False Negative (FN):** The model incorrectly identifies and categorizes a certain number of instances as negative that are actually positive.

Using the confusion matrix the performance metrics given below can be found [42]:

- $True\ positive\ rate(TPR) = \frac{TP}{FN+TP}$
- $False\ positive\ rate(FPR) = \frac{FP}{TN+FP}$
- $True\ negative\ rate(TNR) = \frac{TN}{TN+FP}$
- $False\ negative\ rate(FNR) = \frac{FN}{FN+TP}$
- $Sensitivity = TPR$
- $Specificity = TNR$
- $Precision = TP/(TP + FP)$
- $Recall = sensitivity = TPR$
- $Type\ I\ error\ rate = FPR$
- $Type\ II\ error\ rate = FNR$
- $False\ discovery\ rate(FDR) = FP/(TP + FP)$

For a multiclass classification task, the confusion matrix is an NxN matrix where N is the N number of classes. The matrix shows the number of correctly and wrongly classified predictions for each class $C_1, C_1, ...C_N$. Along the diagonal of the matrix the correct classifications $C_{11}, C_{22}, ...C_{NN}$ are shown in table 2.4. Moreover, the rest of the part in the matrix show misclassifications.

		Actual			
Predicted		C_1	C_2	$\dots$	C_K
	C_1	C_{11}	C_{12}	$\dots$	C_{1N}
	C_2	C_{21}	C_{22}	$\cdots$	C_{2N}
	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
	C_N	C_{N1}	C_{N2}	$\dots$	C_{NN}

Table 2.4 Predicted vs Observed

A multiclass classification confusion matrix can be turned into a 2x2 matrix (binary-class confusion matrix) for each class in order to calculate performance metrics such as accuracy, precision, recall, etc. Using the basic example in table 2.5, a basic multiclass classification confusion matrix is built and converted to a binary-class confusion matrix in figure 2.12. C_{11} is the number of true positive samples. The sum of C_1 column except C_{11} indicates the false negative samples. The sum of C_1 row except C_{11} indicates the false positive samples.

2.4.2. Accuracy

Percentage of correctly predicted instances relative to the total number of instances in the dataset is defined as accuracy. The most common used performance metric is accuracy

Classes	**Number of Samples**
C1	30
C2	20
C3	15
C4	45

Table 2.5 Example of multi-class classification

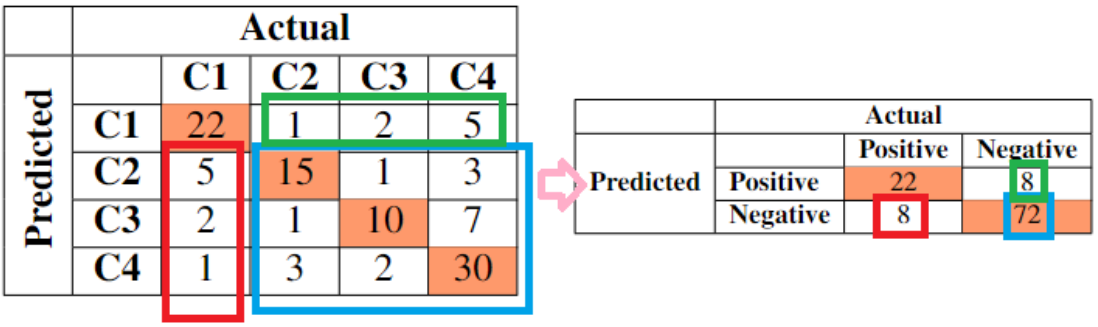

Figure 2.12 Converting multiclass confusion matrix to binary confusion matrix with respect to class C1

where a high accuracy value expresses that the model is making a significant proportion of accurate predictions, meaning it correctly classifies instances most of the time. On the other hand, a low accuracy value shows that the model is making a considerable number of wrong predictions, implying a less reliable performance in classification tasks. Accuracy can be defined as:

$$Accuracy = \frac{TP + TN}{n} \tag{15}$$

where n indicates the total number of samples.

2.4.3. Recall (Sensitivity)

Recall performance metric is also known as Sensitivity or True Positive Rate (TPR) and it is computed as the number of true positives divided by the total number of actual positives.

$$Sensitivity = \frac{TP}{TP + FN} \tag{16}$$

In the context of assessing the efficacy of a diagnostic test in the medical field, sensitivity represents the percentage of individuals with the disease (positive examples), indicating

that the test result is positive [41]. Conversely, specificity quantifies the percentage of individuals without the disease (negative examples), indicating that the test result is negative. In medical scenarios, prioritizing the detection of true positive cases takes precedence over the possibility of triggering false alarms. It is frequently used in conjunction with additional metrics such as precision, specifcity, NPV and accuracy for a comprehensive assessment.

2.4.4. Precision

Positive Predictive Value (PPV) is the fraction of true positives over the number of predicted positives denoted as:

$$Precision = \frac{TP}{TP + FP} \tag{17}$$

It is frequently used in conjunction with additional metrics such as sensitivity, specifcity, NPV and accuracy for a comprehensive assessment. Precision is important where the emphasis is on minimizing the occurrence of False Positives rather than False Negatives.

2.4.5. Specificity

Machine learning classifiers aims to enhance sensitivity while maintaining specificity. Nevertheless, there exists a trade-off between these two metrics, as elevating sensitivity often results in a reduction of specificity and vice versa.

$$Specificity = \frac{TN}{FP + TN} \tag{18}$$

Since Sensitivity, precision, specifcity and NPV fails to account for the entirety of the confusion matrix, none of these four metrics can be employed individually to assess the performance of a machine learning method [39].

2.4.6. F1 Score

By combining the results of both precision and recall, the F1 Score is calculated [43].

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{19}$$

The F_β is a generalized version for F1 Score where β is assigned to 1. When the β value is smaller than 1, it prioritizes precision over recall. Conversely, with a beta value larger than 1, the emphasis shifts towards recall.

$$F_\beta = \frac{(1 + \beta^2) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \tag{20}$$

3. RELATED WORK

Multi-label learning or multi-view learning, and multi-task learning are areas that work with multiple datasets. However, according to the study done in [6], Multi-task learning fits more into the thesis study. The objective of Multi-task learning is to simultaneously acquire knowledge in m tasks, enhancing a model's capacity to handle each task $\{T_i\}_{i=1}^{m}$ by leveraging information from either all or a subset of the tasks[6]. Multi-task learning operates under the assumption that the involved tasks exhibit close interrelationships [44]. It enhances the overall generalization capability by serving a preventive measure against overfitting in each specific task.

Faddou et al. [7] presents a multi-task Adaboost framework that employs Multi-Task Decision Trees as weak classifiers and modifies the information gain rule used in decision trees for learning multiple tasks. AdaBoost is a boosting algorithm where it constructs a "strong classifier" gradually by fine-tuning the weights and integrating one weak classifier at a time using a greedy approach [45]. In this paper a multi-task decision tree algorithm is

defined where an optimal split is found using the maximum value among information gains of IG_J, IG_U and IG_M. IG_J refers to joint information gain which is defined as:

$$IG_J = IG\left(\oplus_{j=1}^{N} Y_j; a\right) \tag{21}$$

Moreover, the unweighted sum of individual task information gains (IG_U) and the maximum value among the individual IGs (IG_M) are used as:

$$IG_U = \sum_{j=1}^{T} IG\left(Y_j; a\right) \tag{22}$$

$$IG_M = \max\left\{IG\left(Y_j; a\right), j = 1, \ldots, N\right\} \tag{23}$$

Unlike traditional decision trees where final decisions are typically made at the leaves, a MT-DT may reach a conclusive decision for certain tasks within an internal test node. This occurs when the internal test node gathers adequate information to classify an instance pertaining to a particular task. As a result, a decision leaf is incorporated into the tree, containing the appropriate classification decision for that specific task and algorithm proceeds with the unresolved tasks. In order to enhance overall classification accuracy the paper focus lies in concurrently addressing N classification tasks. The input of the MT-Adaboost algorithm is the training sample for each classification task $j \in \{1, 2, ...N\}$ whereas the output is the multi-task classifier H. The complete MT-Adaboost algorithm can be seen in algorithm 2.For each boosting iteration ($t = 1, 2, ...T$) the weak learner (WL) returns a multi-task decision tree classifier h_t using the training sample and distribution over $(\mathcal{X}, \mathcal{Y})$ (D). When the classifier h_t accurately categorizes the samples, their weights undergo multiplication by β_t and division by the normalization constant Z_t. The values of β_t fall within the range of $0 \leq \beta_t \leq 1$. Consequently, the weights are reduced following the multiplication process. The ultimate classifier H for a specific task j is defined with respect to the highest $\ln(1/\beta_t)$ weight value.

Algorithm 2 Algorithm for a MT-Adaboost [7]

Input: $S = \cup_{j=1}^{N} \left\{ e_i =< x_i, y_i, j > |\ x_i \in \mathcal{X}; y_i \in \mathcal{Y}_j \right\}$

Output: Classifier H defined as $H_j(x) = \arg\max_{y \in \mathcal{Y}_j} \left(\sum_{i=1}^{i=T} \left(\ln(1/\beta_t) \right) \right), 1 \leq j \leq N$

1: $D_1 = \text{init}(S)$ initialize distribution

2: **for** t=1 to T **do**

3: $h^t = \text{WL}\,(S, D_t)$ {train the weak learner and get an hypothesis MT-DT }
$\{h_t : \mathcal{X} \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N\ \}$

4: Calculate the error of $h^t : \epsilon_t = \sum_{j=1}^{N} \sum_{i:h_j^t(x_i) \neq y_i} D_j\,(x_i)$.

5: **if** $\epsilon_t > 1/2$ **then**

6: Set $T = t - 1$ and abort loop.

7: **end if**

8: $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
{ Update distribution: }

9: **if** $h_j^t\,(x_i) == y_i$ **then**

10: $D_{t+1}\,(e_i) = \frac{D_t(e_i) \times \beta_t}{Z_t}$

11: **else**
$D_{t+1}\,(e_i) = \frac{D_t(e_i)}{Z_t}$

12: **end if**

13: **end for**
{Where Z_t is a normalization constant chosen so that D_{t+1} is a distribution }

14: **return** H

Furthermore, Ying et al. [8] once again employ a boosting technique to address a multi-task learning objective. A multi-task Gradient Boosting Machine (MT-GBM), is built by utilizing GBDT method where decision trees are trained sequentially. In contrast to AdaBoost, the emphasis in gradient boosting is on optimizing a differentiable loss function. In the paper, the tree is split using multi-task losses which is based on Gradients and Hessians.

$$L^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{24}$$

In order to split the node, Ensemble Gradients (G_e) and Hessians (H_e) are used and defined as:

$$G_e = f_G(G_1, G_2, ..., G_3) \tag{25}$$

$$H_e = f_H(H_1, H_2, ..., H_3) \tag{26}$$

In this equation, function f is employed to assess the correlation among the gradient vectors corresponding to each label in the ongoing iteration. On the other hand, for updating the the leaf value Updating Gradients (G_u) and Updating (H_u) are used. Each leaf node corresponds to multiple values as in figure 3.1. Finally, all gradients are gathered and a shared decision tree structure is found in order to learn multiple tasks.

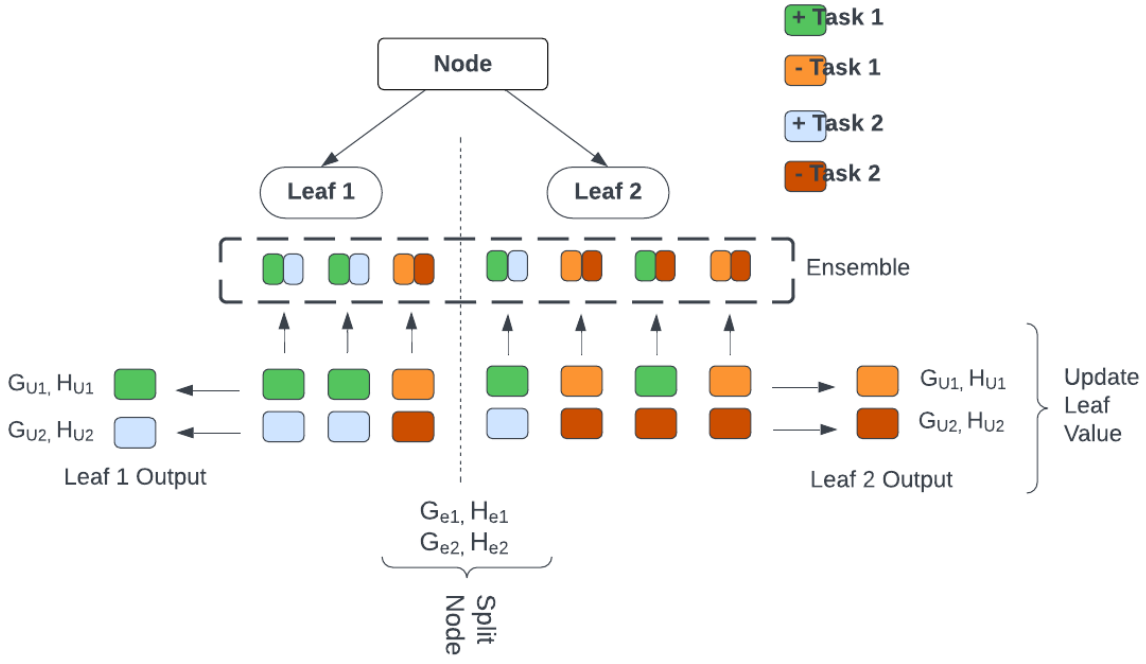

Figure 3.1 Gradients for splitting and updating leaf value

In [9], the author proposes a novel approach called MT-ExtraTrees, a tree-based ensemble method for both classification and regression, built upon extremely randomized trees. ExtraTrees effectiveness in efficiently handling big datasets enhances its attractiveness. MT-ExtraTrees is described as a multi-task learning method that employs a binary decision tree ensemble. The tree incorporates branches exclusively allocated to specific subsets of tasks that exhibit similar behavior.

In addition to introducing a novel algorithm, objective of the studies is to address and mitigate the challenge of catastrophic forgetting. Catastrophic forgetting is an important problem for continual learning. Continual learning characterizes a learning framework wherein the model is systematically exposed to a variety of data and tasks sequentially [46]. In the initial research conducted by McCloskey and Cohen, it was discovered that when training new tasks, neural networks exhibit a proclivity to discard or neglect information previously acquired from tasks that were trained earlier. Catastrophic forgetting has been solved in streaming decision trees by using a class-conditional attribute estimation[5]. There are hardly any resources found except the paper [5], for catastrophic forgetting problem solved for decision trees. However, there are many resources for neural networks such as [47] [48] etc.

However, this paper is inspired by the idea of [10] which presents a method for training a single neural network that simultaneously learns *K* different tasks. The same number of parameters can be used to train a single task and a multiple-task operation within a single neural network architecture. Nevertheless, the paper employed neural networks. However, it will provide valuable insights for future studies for the thesis.

Within the framework of parameter superposition, multiple models are concurrently encapsulated within a unified set of parameter. The set of parameters are denoted as $W_1, W_2, ...W_K$ for each K number of tasks. This means that for each K task, a set of parameters is learned separately and kept in superposition with one another as in figure 3.2. Moreover, the models could also be taken back separately after the superposition.

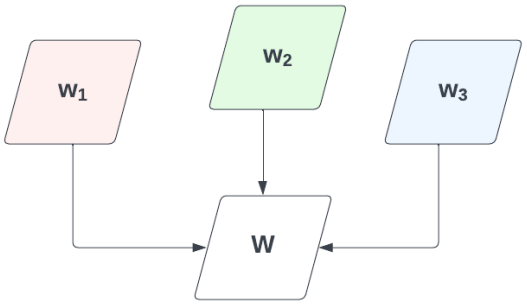

Figure 3.2 Superposition of parameters

For neural networks, the main equation is written as:

$$y = Wx \tag{27}$$

where we multiply the inputs $\left(x \in \Re^N\right)$ with a weight matrix $\left(W \in \Re^{M \times N}\right)$ and compute the output. And for a parameter superposition problem the equation is rewritten as:

$$y_k = W(C_k x) \tag{28}$$

According to [10], in order to reach the specific parameters related to each task "context" information can be used. Under the assumption of small subspace $\Re^N$ exist for each W_K, the transformation of every individual W_K can be obtained using the context information (C_k^{-1}). As seen in figure 3.3, each $W_k C_k^{-1}$ is mutually orthogonal with each other. And, each task-specific parameter can be stored or retrieved by using a context. Since these vectors are in different subspaces and do not overlap with each other, equation can be summarized as:

$$W = \sum_{i=1}^{K} W_i C_i^{-1} \tag{29}$$

Due to its computational and memory advantage, the context vectors are chosen as $c(k)_j \in \{-1, 1\}$ for this literature study. This is called the binary superposition. There are many other superposition types such as rational superposition, complex superposition etc.

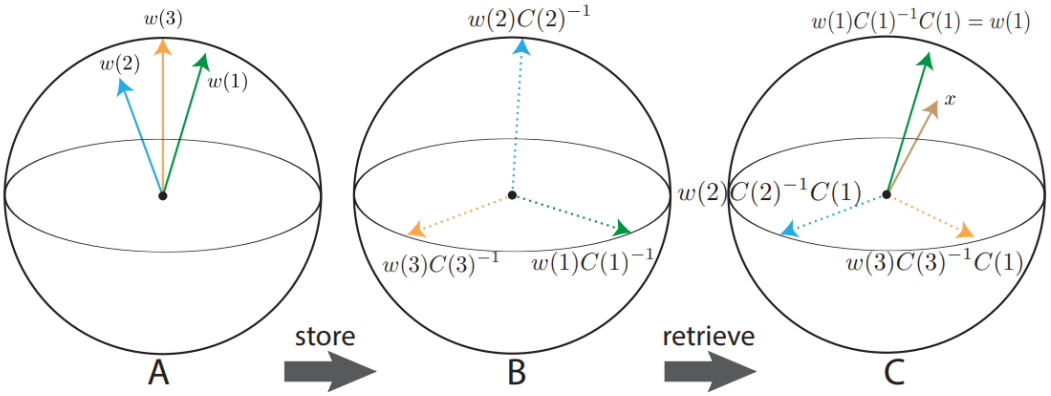

Figure 3.3 Storing and retrieving ω_k using context information [3]

Equations has been applied to the layers of the whole neural network model and the final equation is written as:

$$x^{(l+1)} = g\left(W^{(l)}\left(c(k)^{(l)} \odot x^{(l)}\right)\right) \tag{30}$$

where g() is a non-linearity function. For this study, RELU has been used as a non-linearity function.

The "Input interference" scenario, where the input distribution changes with respect to time, has been studied. Changing the input distribution means obtaining new tasks over time. In order to obtain new tasks over time, permuted MNIST dataset has been used. Permuted MNIST dataset is obtained by permuting image pixels over time which can be seen in figure 3.4.

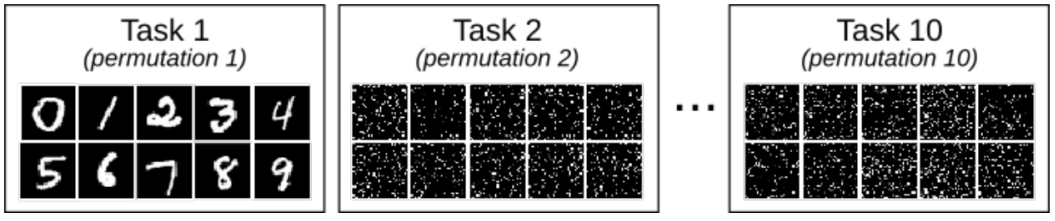

Figure 3.4 Permuted MNIST task[4]

Permuted MNIST dataset has been applied to fully connected networks for different numbers of units respectively as 128, 256, 512, 1024, and 2048. The identical iteration number and

permutation period, as outlined in the paper referenced as [10], are employed in this context. In table 3.1, some of the parameter's values have been shown.

Parameters	**Values**
Activation Function	RELU
Iteration number	50000
Number of units	128, 256, 512, 1024 and 2048
Permutation period	1000
Batch size	128

Table 3.1 Parameters defined for the experiment

The permutation period (P) indicates that the inputs are permuted after every 1000 iterations. This means that, when the number of iterations (N) is 50000, 50 different tasks (T) are obtained. This can be shown as:

$$T = N/P \tag{31}$$

Moreover, for each task, a new context vector has been chosen. The context is randomly chosen from values of $\{-1, 1\}$ for every permutation period. The feature vector has been calculated from equation 28 when the forward function is called. According to [10], more units mean a bigger network and fitting more data which causes the network to be robust to catastrophic forgetting. Therefore, as seen in figure 3.5 larger networks behave better for the parameter superposition model. The classification accuracy is much greater in a network with 2048 units.

For multi-class problems, displaying the cross-entropy loss provides a valuable perspective. As expected, the total loss and cross-entropy loss get smaller as the number of units increases in a network as seen in figures 3.6 and 3.7. The cross-entropy loss decreases when there is a less disparity between the predicted probability and the true label.

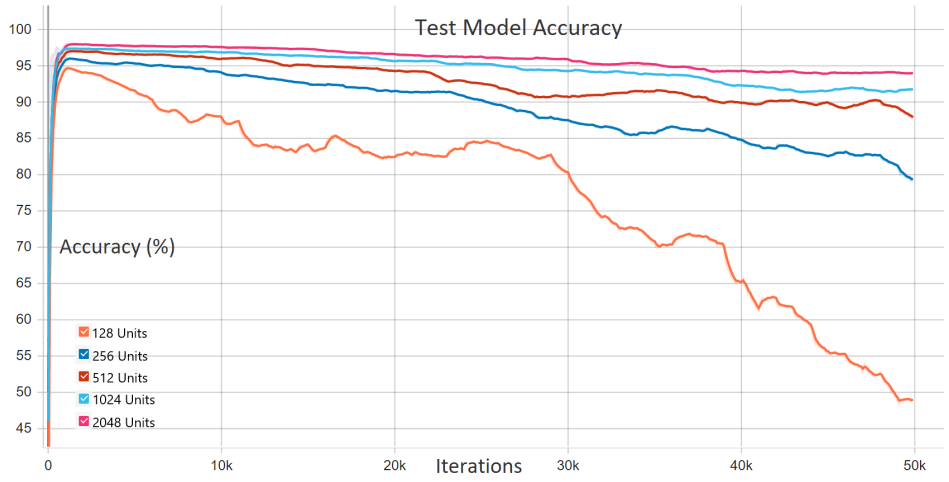

Figure 3.5 Accuracy vs. Number of Units

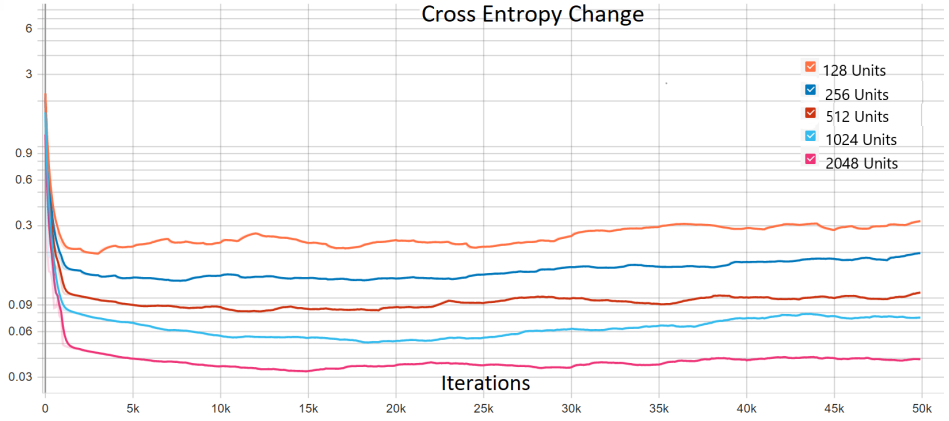

Figure 3.6 Cross Entropy Loss vs. Number of Units

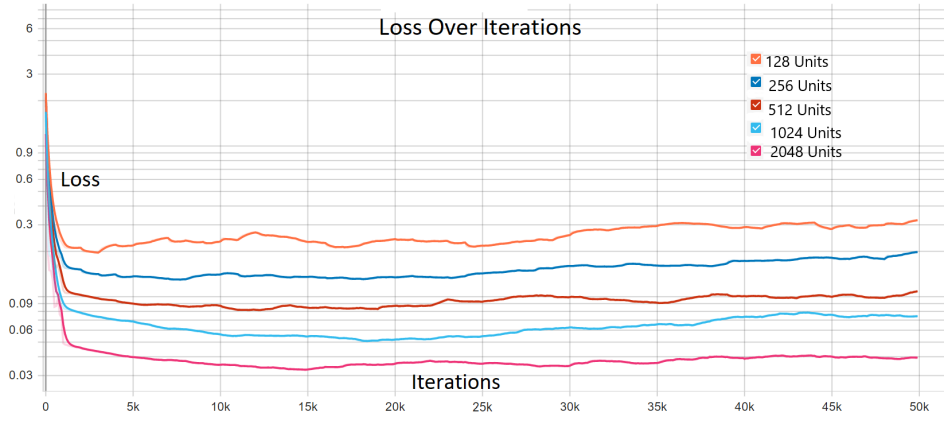

Figure 3.7 Loss vs. Number of Units

4. PROPOSED METHOD

In this section, the full decision tree structure and the switching mechanism are presented. The first part for this section introduces the algorithm responsible for constructing the entire decision tree structure, shedding light on its fundamental components. Subsequently, in the second part of this section, the parameters fine-tuned through the genetic algorithm are presented and a comprehensive understanding of the optimization process is provided.

4.1. Full Decision Tree Structure

A single decision tree structure has to be chosen in order to switch between datasets. At the beginning of the study, a decision tree structure was chosen based on the maximum depth criteria. The maximum depth criteria means choosing the decision tree structure of a dataset that has the maximum depth. Each dataset has its own decision tree and therefore, different depths. The initial concept was to build a decision tree based on the dataset tree which has the highest depth and optimize the thresholds, features, and classes for each dataset. However, it is hard to fit different datasets where there is a specific tree split and path since it represents specific data. Accommodating diverse datasets becomes challenging when aiming to build a specific tree that represents distinct sets of data. As illustrated in Figure 4.1 and Figure 4.2, for instance, the Iris and wheat seeds datasets exhibit disparate tree splits and depths. A dataset featuring an initial right split leading to a leaf node might have resulted in undesirable outcomes when applied to the Iris dataset. This is due to the distinctive node generation pattern of the Iris dataset, which originates from the right side.

A better way to reach the solution is to use a full tree model as in figure 4.3 where the switching mechanism is much more simple. The full tree (FT) is a tree where every node has zero or two children. It was easier to achieve better results in this structure. According to this full-tree model, the depth of the full tree is selected larger than or equal to the maximum depth among datasets.

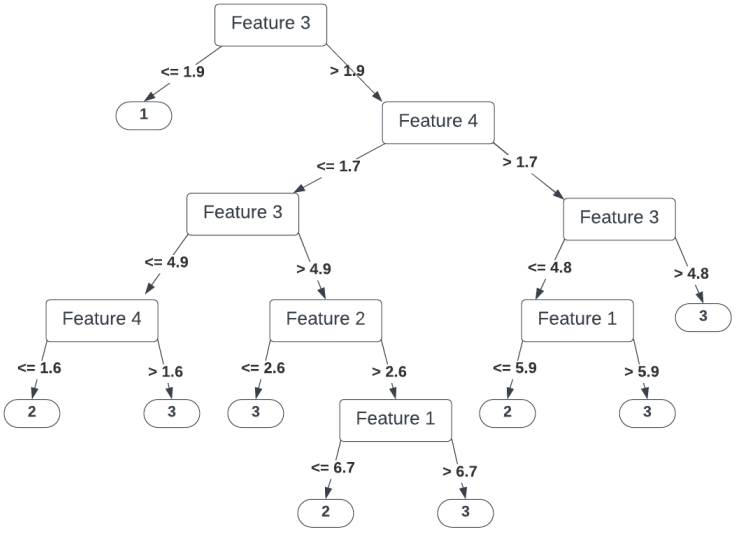

Figure 4.1 Decision tree for Iris dataset

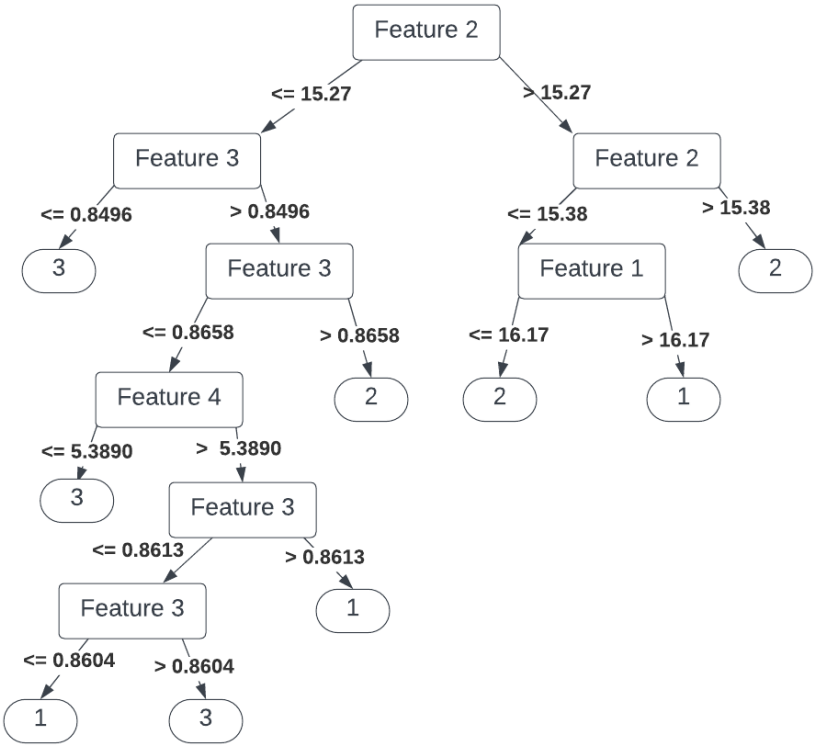

Figure 4.2 Decision tree for Wheat seeds dataset

Given that the decision tree is set to be optimized through a genetic algorithm, it becomes imperative to define a cost function. The cost function of the decision tree model is chosen

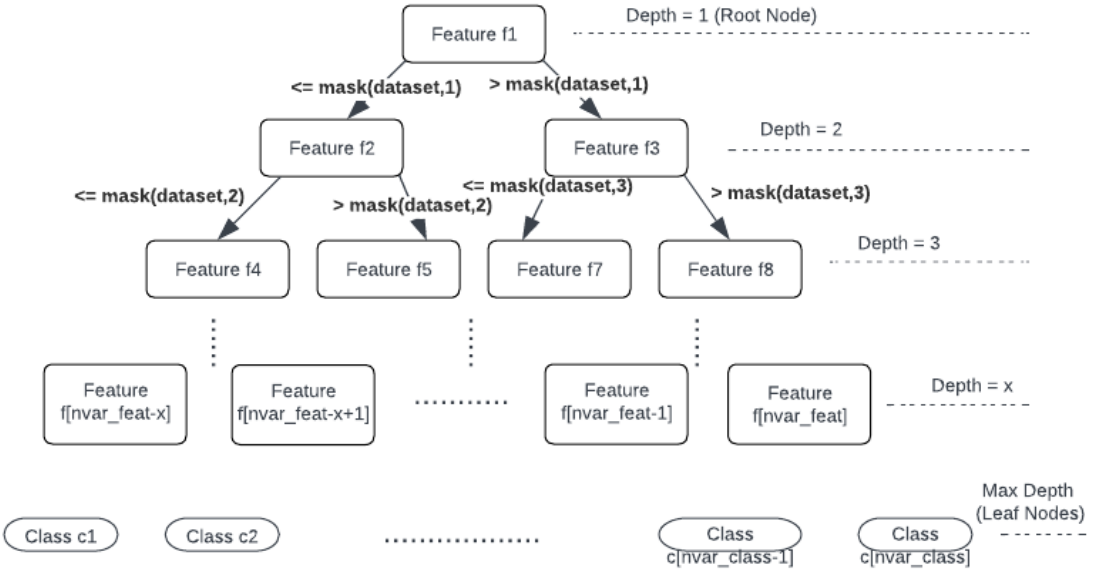

Figure 4.3 Full Tree Structure

as:

$$J = \sum_{i=1}^{m} 100 - \alpha_i \tag{32}$$

where, α is the accuracy percentage, and indicates the percentage of data that are correctly classified. Since the accuracy of the entire data set should be examined, the cost function is written in terms of α_m. Moreover, increasing the total accuracy is critical. Therefore, a summation symbol is added to the definition of the cost function at the outermost part. For the total number of datasets (m) the accuracy must be increased.

The algorithm for the full decision tree can be written as in Algorithm 3. In full tree there is no term as finding best split as in decision tree structure. Every node is split until the maximum depth is reached. The stopping condition of the full decision tree is defined with respect to the maximum depth of the entire tree. The genetic algorithm comprehensively updates the entire full tree structure. Parameters, including class labels, threshold values, and the feature array, are fed into the genetic algorithm, where they undergo a fine-tuning

process. By iteratively reducing the cost function, the algorithm fine-tunes the parameters and enhances the overall performance and efficiency of the optimization process. The genetic algorithm tries to converge towards an effective solution and, therefore, finds the optimal parameters.

Algorithm 3 Algorithm for Full Decision Tree Model [13]

Input: Training set *S*, attribute set *A*

Output: Full decision tree structure

TreeGrowth(S,A) :

if stoppingCond(S,A) = true **then**
  leaf = createNode()
  leaf.label = classLabels
  return *leaf*
else
  root = createNode()
  root.test_cond = split(S,A)
  let V = { *v* | *v* is a outcome of *root.test_cond* }
  for each v ∈ V **do**
    S_v = {s | root.test_cond(e) = v and s ∈ S }
    child = TreeGrowth(S_v,A)
    add *child* as descendent of *root* and label the edge *(root* → *child)* as *v*
  end for
  return $root$
end if

4.2. Switching Model

Every dataset possesses its distinctive decision tree representation, where features, thresholds, classes, and splits can vary. Integrating decision trees from diverse datasets into a unified full tree model demands considerable effort due to these inherent differences. However, the existence of a full tree model with consistent positions for features, class sets

and thresholds underscores the significance of optimization in streamlining this integration process. Especially in mask structure there can be a huge amount of parameters that need to be optimized through a genetic algorithm. Due to its extensive depth, a tree will have large number of threshold values. Consequently, the computation time may become excessively high due to the processing of huge number of threshold values in a tree with significant depth. In genetic algorithm, it is important to define the contraints on the optimization parameters which are the thresholds, classes, and features for each dataset. By constraining the optimization parameters, the algorithm will operate within specified bounds and may reach the optimal solution faster. Before, starting the genetic algorithm process, the initial step involves constructing decision trees for each dataset. Subsequently, for each tree (t_m), the respective depths are determined. Then, the maximum depth among them is chosen. Then, the parameter D, representing the depth of the full tree to be constructed, is determined as follows:

$$D \geq \max\left(\text{depth}\left(t_1\right), \text{depth}\left(t_2\right), \ldots, \text{depth}\left(t_m\right)\right) \tag{33}$$

The full tree depth, D, is a hyperparameter. D should be chosen greater or equal to the maximum depths among datasets as defined in equation 33. After finding this limit value the highest D value can be selected by trial and error. Once the full tree depth is determined, the construction of the tree can proceed accordingly. The established full tree depth will provide valuable insights into the numerical values and constraints for the optimization parameters. First of all the total number of thresholds are defined as (n):

$$n = 2^{D-1} - 1 \tag{34}$$

When the tree depth is large, the exponential calculations result in a significant number of thresholds that need to be optimized. The mask is a matrix that stores threshold values for each dataset which is shown in equation 35. The number of rows in the mask matrix is

equal to the number of datasets, while the number of columns corresponds to the number of thresholds.

$$mask = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}_{m \times n} \tag{35}$$

The total number of threshold variables within a mask structure is defined as:

$$nvar_{mask} = mn \tag{36}$$

The class array consists of dataset labels. The size of the class array is $[1, nvar_{class}]$. Moreover, the number of feature variables should be the same as the number of threshold variables.

$$nvar_{class} = 2^{D-1} \tag{37}$$

$$nvar_{feature} = n \tag{38}$$

It should be noted that the dataset has a certain class and feature numbers. This should not be confused with the variable size here. For example, if the dataset has three classes, the class array will have size of $nvar_{class}$. However, this array consists of the three class labels.

5. EXPERIMENTAL RESULTS

The simulations are performed in MATLAB. First of all, the designed model is tested on a solvable problem, where the datasets are generated via switched decision trees. This ensures

that there is a solution with zero cost. Then, the model is applied to different datasets and an appropriate tree with an associated mask is sought.

5.1. Performance Evaluation

In order to evaluate the classification accuracy and performance of the proposed method for different classification tasks, firstly, data obtained from a solvable problem is used. Then, some of the real-world datasets are used to demonstrate the merits and effectiveness of the proposed scheme. Utilizing parallel processing can enhance processing speed, particularly if there is a multicore processor.

5.1.1. Using a Solvable Problem Dataset

First of all the problem is simplified. Matlab's "meshgrid" command is used to create a two-dimensional array that contains values between -1 to 1. The array can be thought of as a dataset. And, the data has passed through a decision tree where each of the splitting values and features are known. Also, each node split is defined. Finally, the aim is to solve the problem in reverse. The main problem is to find the decision tree using the datasets. In figure 5.1, the problem is visualized. Each leaf contains a class label whereas each internal node has a feature and threshold value. In order to switch between datasets the mask matrix is used. The feature and class labels are stationary. On the other hand, the thresholds are switched with respect to the datasets.

Consequently, the new proposed algorithm has worked as expected on a solvable problem. The cost function is minimized and it has reached zero as shown in figure 5.2. This indicates that the accuracy is maximum and the decision tree parameters for the solvable problem has been found through the proposed algorithm. As the cost function is formulated based on incorrect class predictions, the genetic algorithm endeavors to minimize the occurrences of misclassified classes. In figure 5.3, the accuracies are shown. As expected, while the cost function decreases, the accuracies for each dataset increase. The drop in accuracy at higher

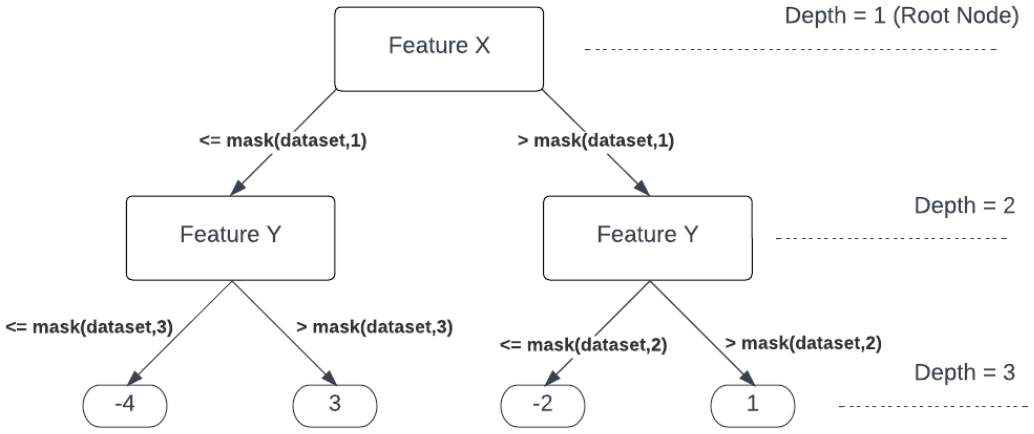

Figure 5.1 Simple decision tree structure for a solvable problem

values is attributed to the stochastic nature inherent in the genetic algorithm.

The genetic algorithm requires the definition of two essential parameters: population size and the number of generations. Since there are fewer variables in the algorithm that need optimization, the population size is set to 150 as seen in table 5.1. The maximum generation number is found by trial and error.

GA Parameters	**Values**
Population Size	150
Max Generations	200
Number of Variables	13
Depth of Full DT	3
Elite count	8
Crossover Fraction	0.8

Table 5.1 GA parameter settings for solvable problem

The mask parameters, feature, and class array are found through the new algorithm. The best cost value indicates the minimum or optimal value reached among all members of the population. On the other hand, the mean fitness shows the average of all members of the

current population. For populations with low diversity, the mean and best value are the same or close to each other.

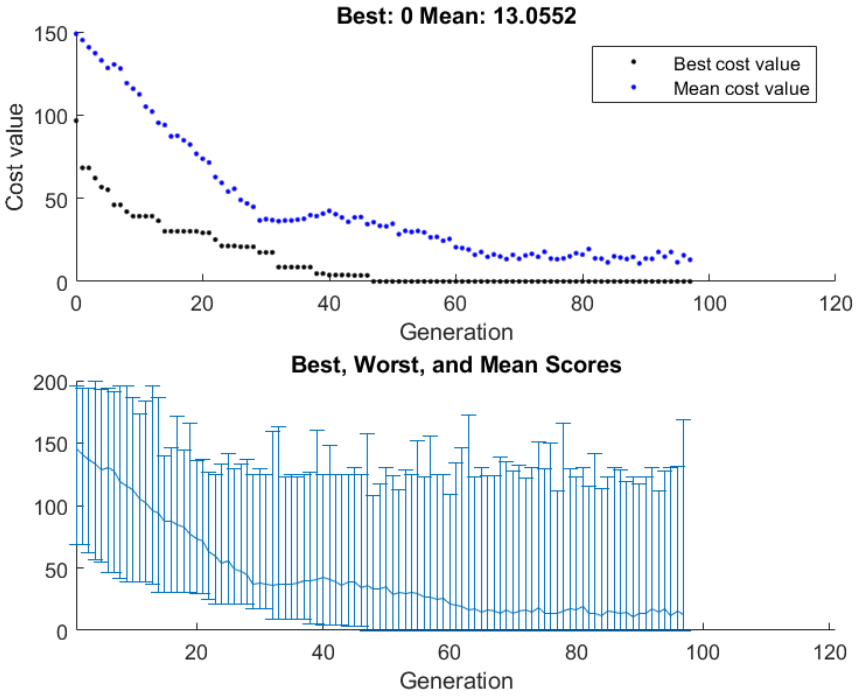

Figure 5.2 Solvable problem simulation

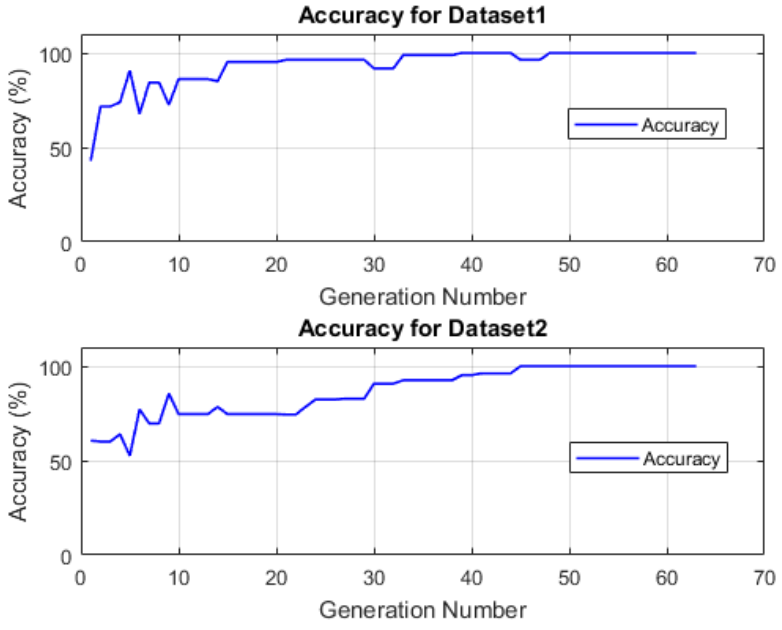

Figure 5.3 Accuracies for dataset 1 and 2 for a solvable problem

5.1.2. Using Real Datasets

A literature review was done for datasets in order to find better results. The datasets used for classification tasks were filtered. For classification tasks, the selected datasets are used as benchmarks in literature. These datasets are found from Kaggle and UCI. For each dataset the knowledge of feature, class size, number of instances and data source was examined and finalized in table 5.2. All of the datasets used in the study has been splitted into two subsets of $\%80$ train data and $\%20$ test data.

Dataset Name	**Feature Size**	**Class Size**	**No. of instances**	**Data Source**
Golf Play Dataset with Extended Features	4	2	14	kaggle
Iris dataset	4	3	150	UCI
Balance Scale	4	3	625	UCI
Car Evaluation	6	4	1728	UCI
Seeds	7	3	210	UCI
Penguin	7	3	335	Kaggle
Diabetes prediction dataset	8	2	100000	kaggle
Tic-Tac-Toe Endgame	9	2	958	UCI
Water Quality and Potability	9	2	3276	kaggle
Breast Cancer Wisconsin (Original)	9	2	699	UCI
(Diabetes, Hypertension and) Stroke Prediction	10	2	40910	kaggle
Heart Failure Prediction Dataset	11	2	918	kaggle
Pumpkin Seeds Dataset	12	2	2500	kaggle
Heart failure clinical records	12	2	299	UCI
Heart Disease Dataset	13	2	303	UCI
(Diabetes,) Hypertension (and Stroke) Prediction	13	2	26083	kaggle
Wine	13	3	178	UCI
Statlog (Heart) Data Set	13	2	270	kaggle
Adult	14	2	48842	UCI
Credit Approval	15	2	690	UCI
Diabetes (Hypertension and Stroke) Prediction	17	2	70692	kaggle

Table 5.2 Datasets from the Kaggle and UCI repository

The order of the datasets is arranged based on the feature size. To observe the results systematically, the initial focus is on datasets with the smallest feature size. Also, if the

class sizes are equal the results should be better. Therefore, the simulations are done with datasets that have the same feature and class size. Since the number of instances increase the computation time, it is good to start with Iris and balance dataset. Due to the high number of instances, the computation time in MATLAB significantly increases.

5.1.2.1. Iris and Balance Dataset Iris dataset is an iris plant dataset that consists of four feature measurements from three different species [49]. The features of the Iris dataset include sepal length, sepal width, petal length, and petal width. These feature measurements consist of continuous-valued numbers. Moreover, the dataset consists of three classes: Iris setosa, Iris versicolor and Iris virginica. In balanced dataset there are approximately equal number of instances across each class. As seen in figure 5.4, Iris is a balanced dataset.

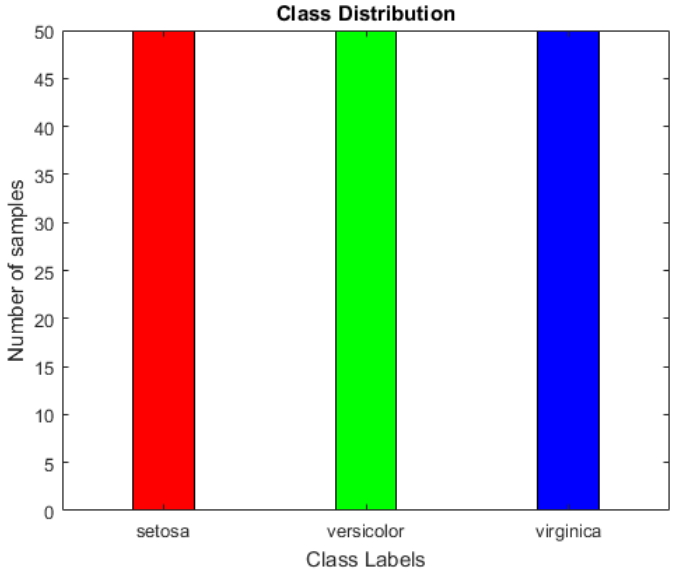

Figure 5.4 Class distribution plot for Iris dataset

On the other hand, the second dataset which is ”balance scale” gives information on whether the balance scale tip is at left, right or balanced. The dataset is classified according to the greatness between (left-distance * left-weight) and (right-distance * right-weight). If the values are equal it is classified as balanced. Moreover, it is important to note that this dataset consists of categorical values for each measurements. It is crucial to emphasize that the dataset is imbalanced which means that it contains unequal distribution of classes. In figure

5.5, it can be observed that class 'B' constitutes the minority, accounting for 8% of the dataset. Since the dataset is imbalanced accuracy performance metric may be misleading. The predictions made for the minority class may not be sufficient.

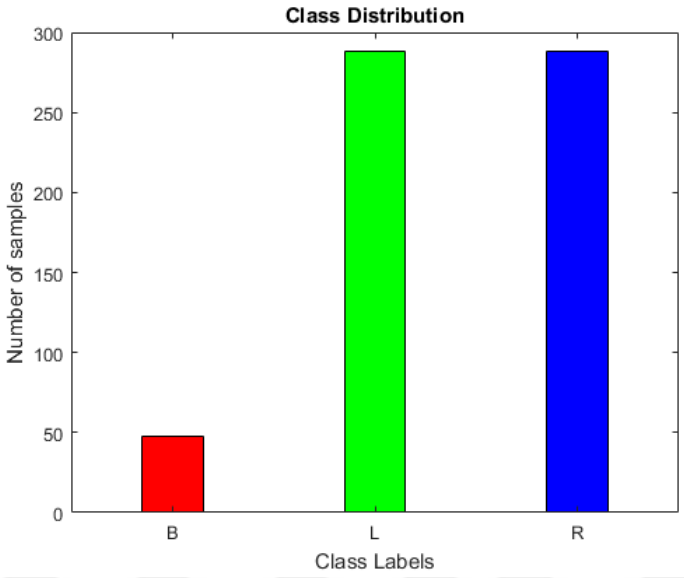

Figure 5.5 Class distribution plot for balance dataset

During the operation of the algorithm for these datasets, the parameters of the genetic algorithm defined in table 5.3 are used. As discussed in section 2.3.2., since there are 509 variables passed into the genetic algorithm, the population size is set to 200. Because of the large number of optimized variables, the maximum generation is chosen as 500.

GA Parameters	**Values**
Population Size	200
Max Generations	500
Number of Variables	509
Depth of Full DT	8
Elite count	10
Crossover Fraction	0.8

Table 5.3 GA parameter settings for datasets Iris and balance

	Training Accuracy (%)	Test Accuracy (%)
Iris Dataset	98.33 %	86.67 %
Balance dataset	75.20 %	70.97 %

Table 5.4 Train and test accuracy for iris and balance dataset

In order to make predictions, the dataset has been splitted into two subsets as: $\%80$ train data and $\%20$ test data. Both train and test accuracy are separately calculated and shown in table 5.4. To estimate the performance on a new data, the test data is not used in the training process. By using this new decision tree structure the classification accuracy's for "balance scale" dataset is approximately $\%75$, whereas for Iris is $\%98$. The model achieved sufficient performance. In single decision tree based models the classification accuracy values for "balance scale" dataset was approximately $\%87$, whereas for Iris it was $\%95 - 100$. The simulation result is shown in figure 5.6. As time progresses, the cost function can only minimize to a certain extent.

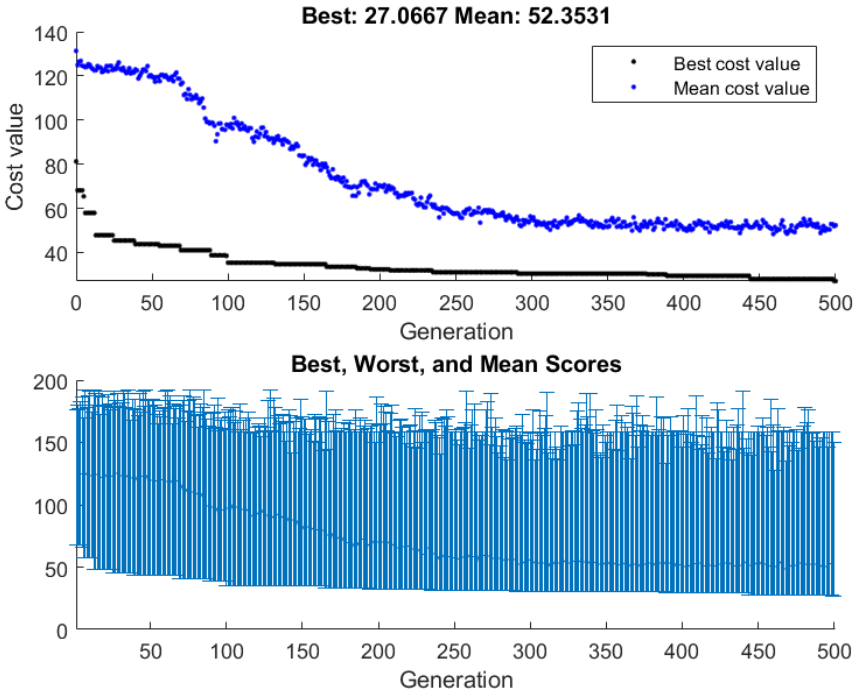

Figure 5.6 Simulation of iris and balance dataset with depth of 8

The study is also examined with different genetic algorithm parameters in order to decrease

the cost value. According to the study [50], decreasing elite count can lead to better results. However, decreasing elite count has made a negative effect on our thesis study. Especially when there were fewer best score values in the population, a lower value was chosen for the elite count. After the change in the value of the elite count, no further decrement has been observed in the cost function.Therefore, the study continued by increasing the depth of the full tree as shown in figure 5.7. However, it took longer time to reach the similar results. Therefore, it is better to choose the full tree depth by adding maximum of three to the maximum depth among dataset depths.

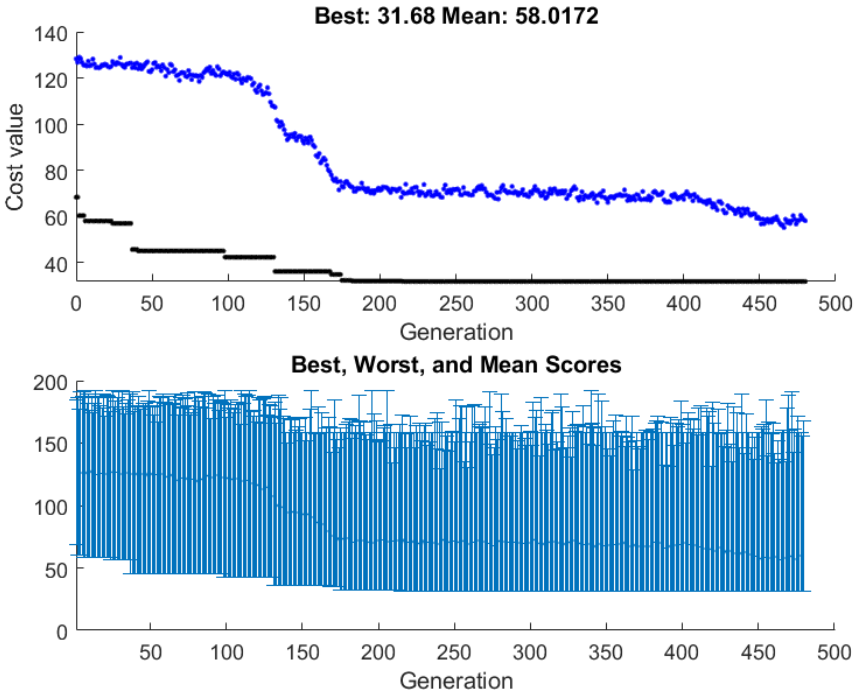

Figure 5.7 Simulation of iris and balance dataset with depth of 10

The confusion matrices for each dataset is shown in figure 5.8 and 5.9 in order to show the classification performance. Also, the recall, precision, FNR and FDR results are shown in figure. In Iris dataset, the classes ”setosa” and ”versicolor” has recall of 100 % . The recall value of 63.6% for ”virginica” indicates that out of 11 samples belonging to this class, only 4 of them were correctly identified. For ”versicolor”, model predicts the class with a precision of 73.3% . Precision indicates that model classifies 15 samples as ”versicolor” whereas 11 samples are actually belonging to this class.

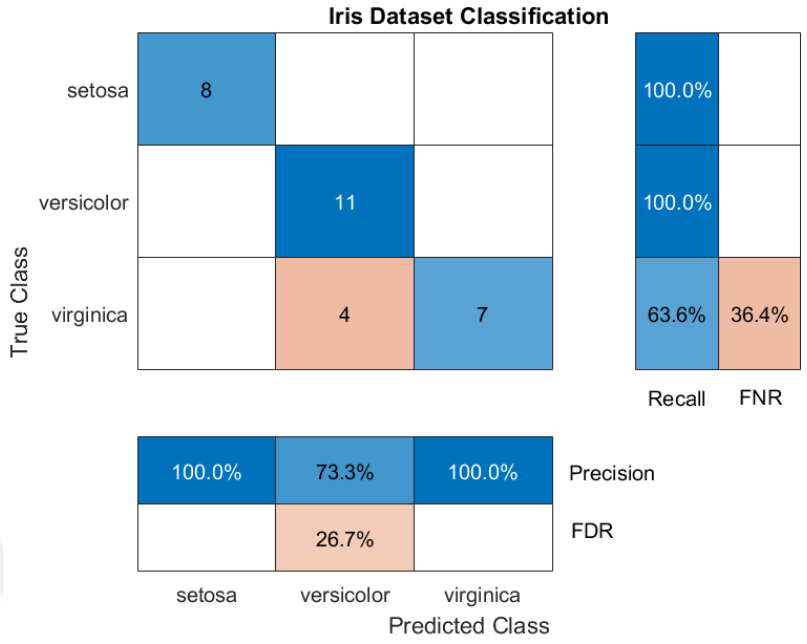

Figure 5.8 Confusion matrix of Iris dataset

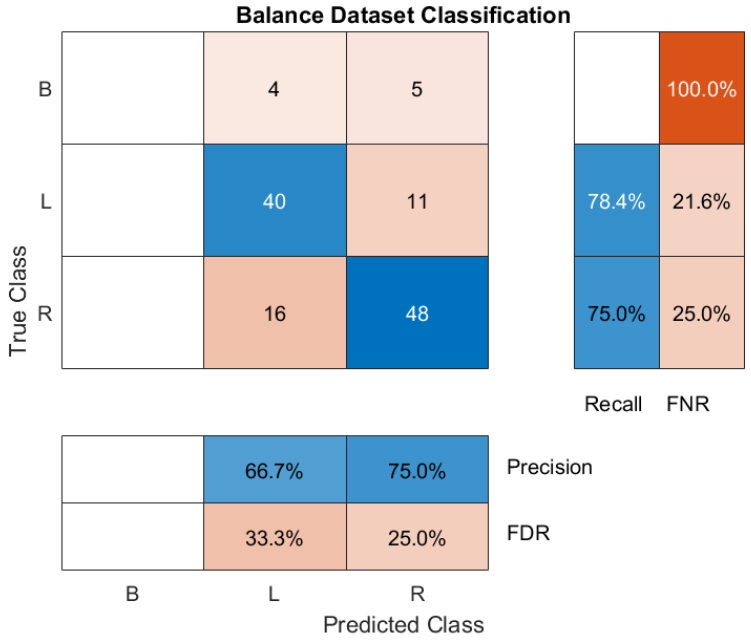

Figure 5.9 Confusion matrix of balance dataset

In balance dataset, as expected, model's performance on the minority class is low due to the less amount of class portion of the minority class. Applying over-sampling to the minority class, using F1-score as an evaluation metric can be used for classifying the imbalanced

dataset with higher performance. As a further study, the cost function can be modified according to desired performance criteria. Therefore, if the goal is to increase the F1 Score, the cost function should be chosen with respect to the F1 score instead of accuracy.

5.1.2.2. Pumpkin Seeds and Heart Failure Clinical Records Dataset Pumpkin Seeds is a dataset that consists of two types of pumpkin seeds which are ”Çerçevelik” and ”Ürgüp Sivrisi” [51]. The dataset has 12 features such as: Major axis length , minor axis length, perimeter, convex area, area, equivalent diameter, aspect ratio, roundness, compactness, extent, eccentricity and solidity. These feature measurements consist of continuous-valued numbers. The dataset can be thought of balanced since the class proportions are nearly the same as seen in figure 5.10.

Figure 5.10 Class distribution plot for pumpkin seeds dataset

On the other hand heart failure clinical records dataset has two boolean classes where it indicates if the patient died (1) or not (0) [52]. Moreover, it has 12 features as: age, anaemia, creatinine phosphokinase level, diabetes, ejection fraction, high blood pressure, platelets, level of serum creatinine, level of serum sodium, sex, smoking and time. As seen in figure 5.11, the class distribution is imbalanced. Category ”1” is the minority class.

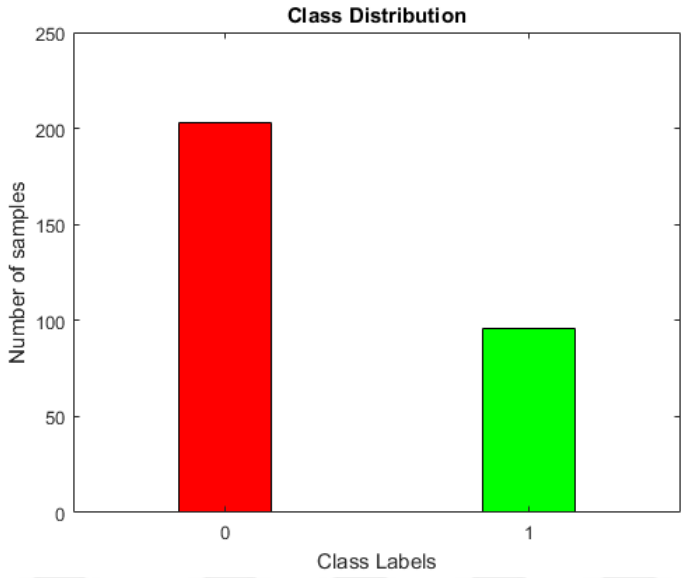

Figure 5.11 Class distribution plot for heart failure record dataset

The genetic algorithm has been initialized using parameters defined in table 5.5. For these datasets, the depth of the full decision tree is smaller. Therefore, there are less number of variables to optimize.

GA Parameters	**Values**
Population Size	200
Max Generations	500
Number of Variables	253
Depth of Full DT	7
Elite count	10
Crossover Fraction	0.8

Table 5.5 GA parameter settings for datasets pumpkin seeds and heart records

After approximately 80 iterations, the algorithm could not converge to a better solution as seen in figure 5.12. It may be stuck on local optimum point. Since finding a decision tree is a NP-complete complex problem, the traditional decision trees may want to reach to local optimum instead of global optimum [53].

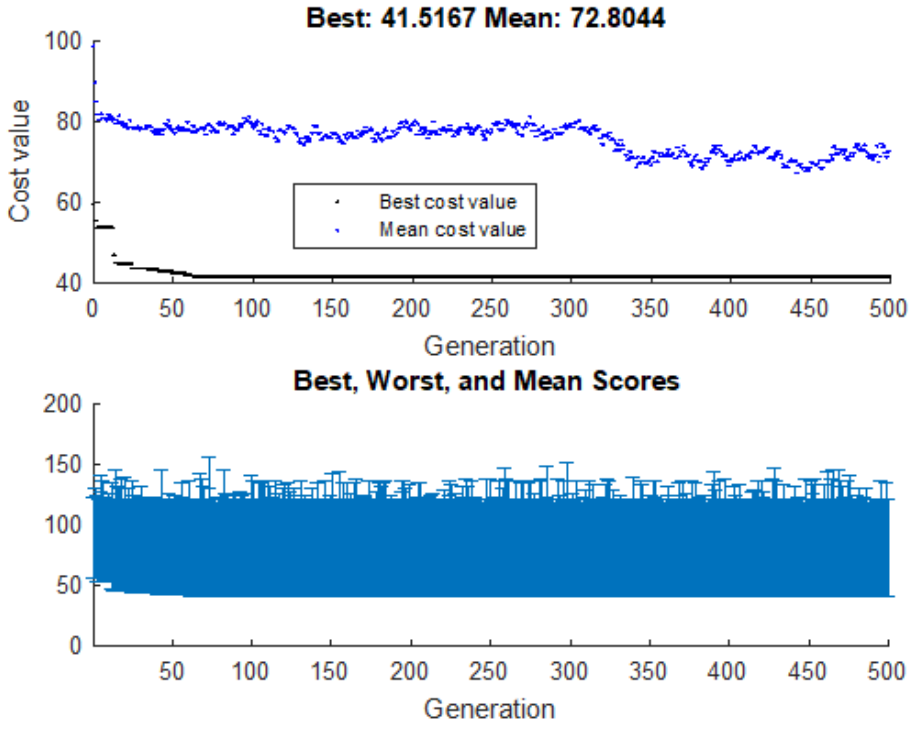

Figure 5.12 Simulation of pumpkin seeds and heart records dataset

It can be seen that in table 5.6, the training and test accuracy is almost the same. This situation shows that the model behaves similar in the test data which is usually considered as a positive outcome.

	Training Accuracy (%)	**Test Accuracy (%)**
Pumpkin Seeds Dataset	87.65 %	87.40 %
Heart records dataset	70.83 %	61.07 %

Table 5.6 Train and test accuracy for pumpkin seeds and heart records dataset

The confusion matrices for the two datasets are given in figure 5.13 and 5.14. The heart failure record dataset comprises approximately 8% of the sample size of the pumpkin seed dataset. This indicates that the heart failure record dataset is relatively small when compared with the pumpkin seed dataset. It is important to mention that the train and test split gives better performance results when the dataset is large. When the dataset is small, there will be insufficient amount of training data to learn the model.

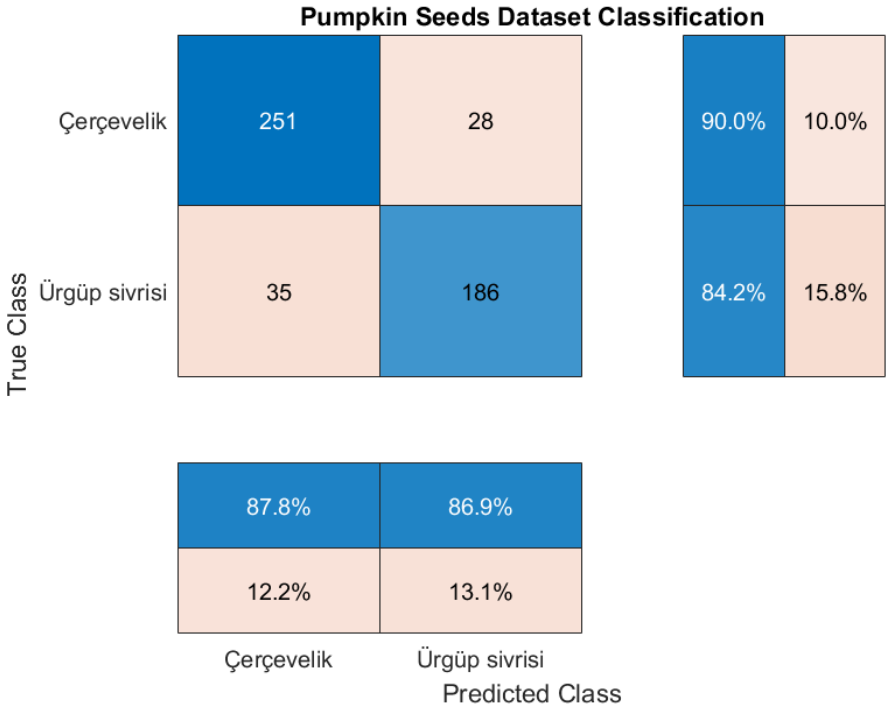

Figure 5.13 Confusion matrix of pumpkin seeds dataset

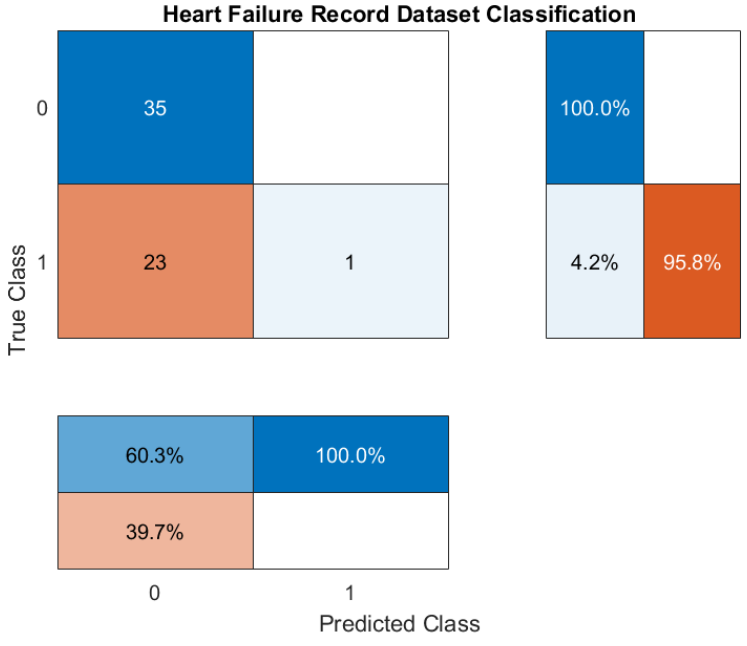

Figure 5.14 Confusion matrix of heart failure record dataset

5.1.2.3. Penguin and Seeds Dataset Penguin dataset contains three types of classes which are Adelie, Chinstrap and Gentoo [54]. The dataset has 7 features such as: island,

culmen length (mm), culmen depth (mm), flipper length (mm), body mass, sex and year. The dataset is considered balanced as seen in figure 5.15.

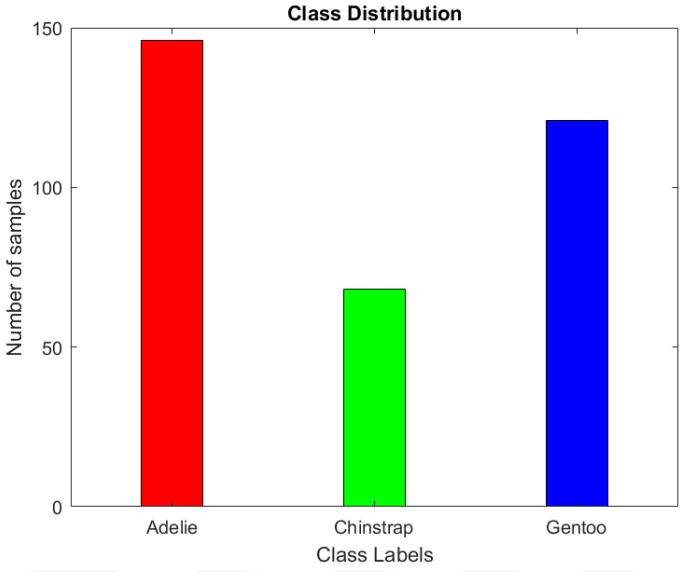

Figure 5.15 Class distribution plot for penguin dataset

Seeds dataset contains three different kinds of wheat which are Kama, Rosa and Canadian [55]. The dataset has 7 features such as: area , perimeter, compactness , length of kernel, width of kernel, asymmetry coefficient and length of kernel groove. As can be seen in figure 5.16 the class distribution is imbalanced.

The genetic algorithm parameters are defined as in table 5.7.

GA Parameters	**Values**
Population Size	200
Max Generations	500
Number of Variables	253
Maximum Depth of DT	7
Elite count	10
Crossover Fraction	0.8

Table 5.7 GA parameter settings for datasets penguin and seeds

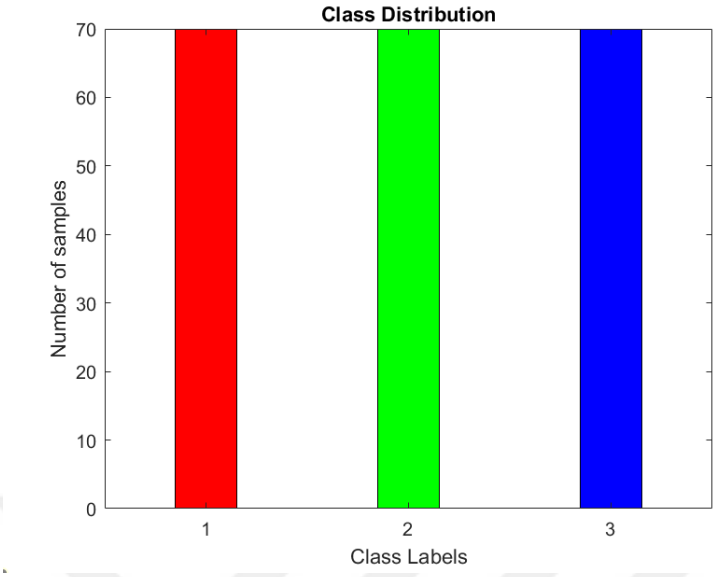

Figure 5.16 Class distribution plot for seeds dataset

The cost function is decreased until the value of 62.58 as seen in figure 5.17. The mean cost value could not attain an appropriate level to induce a decrease for the optimal cost.

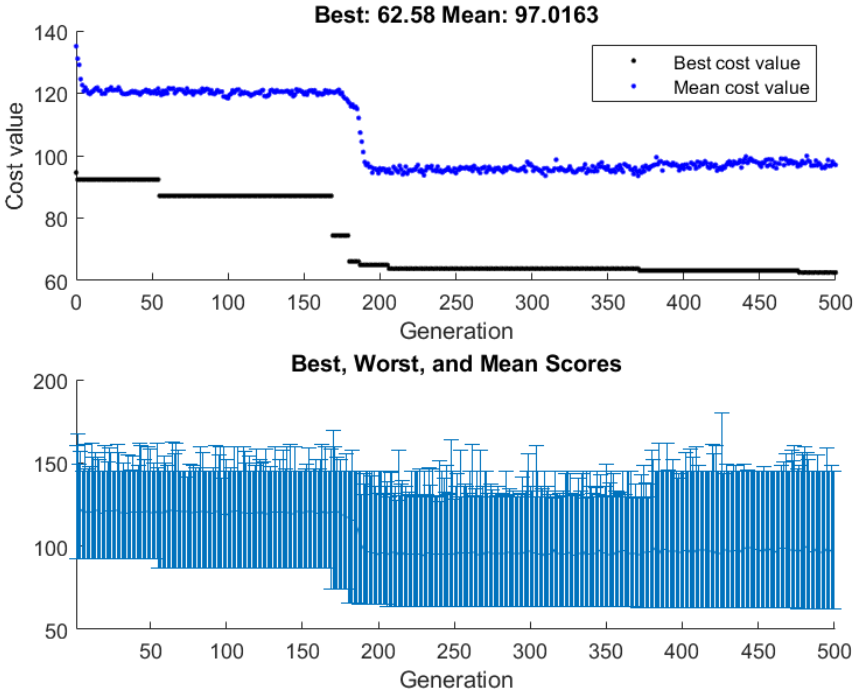

Figure 5.17 Simulation of penguin and seeds dataset

The final results for train and test accuracy is summarized in table 5.8. The test and train accuracy is low because the cost value did not decrease to a satisfactory level.

	Training Accuracy (%)	**Test Accuracy (%)**
Penguin Dataset	73.13 %	71.64 %
Seeds dataset	64.29 %	54.76 %

Table 5.8 Train and test accuracy for penguin and seeds dataset

The confusion matrices in figures 5.18 and 5.19 shows the the predicated classes. There were 13 samples of Chinstrap penguin in the dataset. However, the model did not predict any Chinstrap penguin. The same situation happened for the seed dataset's 3rd category. As a result, the FNR for the both classes are 100 %.

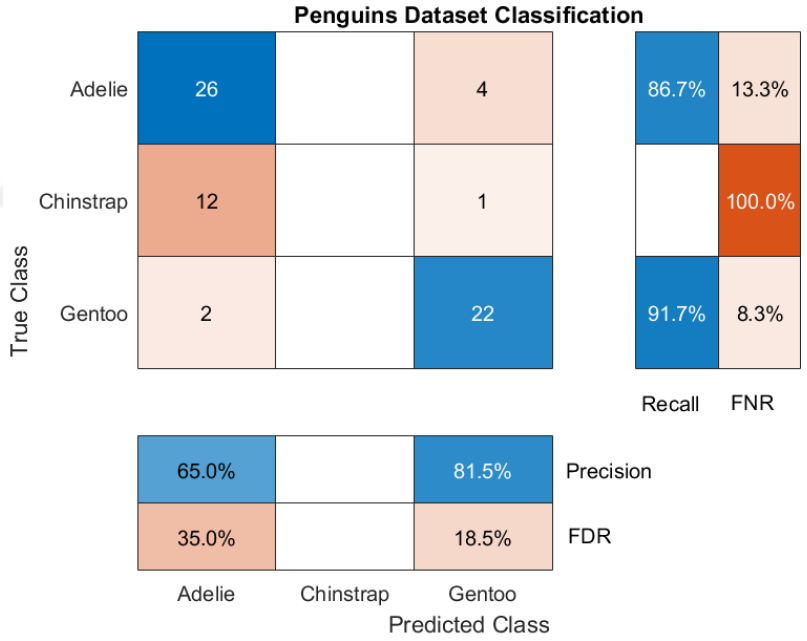

Figure 5.18 Confusion matrix of penguin dataset

The depth of the full tree is a hyperparameter. Therefore, additional simulations were conducted at a greater depth; nevertheless, these revealed a decrease in performance associated with this modification as seen in figure 5.20.

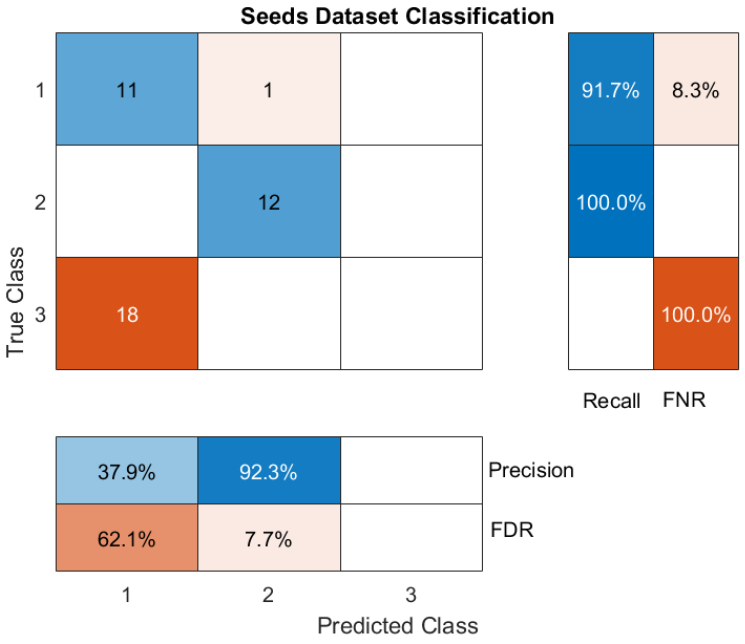

Figure 5.19 Confusion matrix of seeds dataset

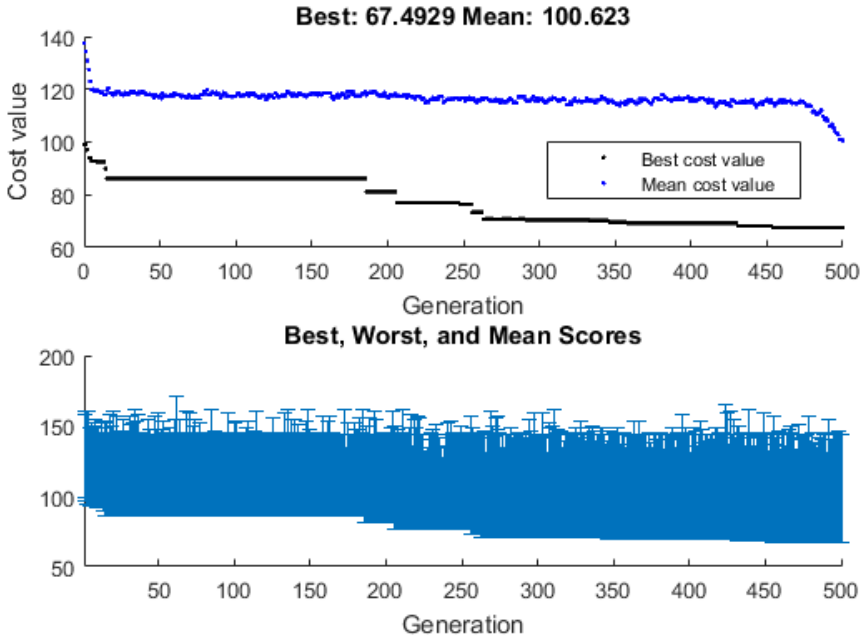

Figure 5.20 Penguin and seeds dataset with depth of 8

6. CONCLUSION

In this thesis, a new switched decision tree structure is proposed, which is built for the classification of multiple datasets within a single switched decision tree structure. A single full decision tree structure has been used and a mask is defined to switch between datasets. The depth of the full decision tree is defined as a hyper-parameter. However, it is shown that the full tree depth should be chosen bigger than the depth of decision trees obtained for each dataset. The simulations showed that it is important to avoid excessively high full tree depths, as they can cause the cost function to remain at elevated values.

The thesis provides insight into the applicability of genetic algorithms for finding critical parameters of decision trees, including thresholds, features, and classes. Experimental studies has been performed on a number of real-world scenarios. Moreover, it is shown that for similar datasets, the classification accuracy is fairly good. Using the related multiple datasets better results can be obtained. If the datasets differ from each other, the lower accuracy values are expected. However, the proposed method demonstrates favorable outcomes even when applied to unrelated datasets.

Further studies can be made to increase the accuracy for different datasets. A new perspective can be added to increase the performance of the model. Moreover, the cost function can be manipulated according to the desired performance metric. For imbalanced datasets, a cost function that contains F1 score can be selected. The code can be implemented in Colab or a GPU environment to speed up training. Overall, obtained results seem promising. The study is generally useful in places where hardware resources are limited.

REFERENCES

[1] Tom M. Mitchell. *Machine learning*. McGraw-Hill, **2021**.

[2] Achille Messac. *Optimization in practice with MATLAB for engineering students and professionals*. Cambridge University Press, **2015**.

[3] Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one, **2019**. doi:10.48550/ARXIV.1902.05522.

[4] Gido M. van de Ven and Andreas S. Tolias. Generative replay with feedback connections as a general strategy for continual learning, **2018**. doi:10.48550/ARXIV.1809.10635.

[5] Lukasz Korycki and B. Krawczyk. Streaming decision trees for lifelong learning. In *ECML/PKDD*. **2021**.

[6] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, **2022**. doi:10.1109/TKDE.2021.3070203.

[7] Jean Faddoul, Boris Chidlovskii, Rémi Gilleron, and Fabien Torre. Learning multiple tasks with boosted decision trees. volume 7523, pages 681–696. **2012**. ISBN 978-3-642-33459-7. doi:10.1007/978-3-642-33460-3_49.

[8] ZhenZhe Ying, Zhuoer Xu, Weiqiang Wang, and Changhua Meng. Mt-gbm: A multi-task gradient boosting machine with shared decision trees, **2022**.

[9] Jaak Simm, Ildefons Magrans, and Masashi Sugiyama. Tree-based ensemble multi-task learning method for classification and regression. *IEICE Transactions on Information and Systems*, E97.D:1677–1681, **2014**. doi:10.1587/transinf.E97.D.1677.

[10] Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one, **2019**. doi:10.48550/ARXIV.1902.05522.

[11] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, **2014**. doi:10.1017/CBO9781107298019.

[12] Alexander Jung. *Machine learning: The basics*. Springer, **2022**.

[13] Lior Rokach and Oded Maimon. *Data mining with decision trees. Theory and applications*, volume 69. **2008**. doi:10.1142/9789812771728_0001.

[14] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman Hall/CRC, **1984**.

[15] Tom M. Mitchell. *Decision Tree Learning*. McGraw-Hill, **2021**.

[16] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, **2005**. doi:10.1109/TSMCC.2004.843247.

[17] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, **2012**.

[18] A Galathiya, Amit Ganatra, and C. Bhensdadia. Improved decision tree induction algorithm with feature selection, cross validation, model complexity and reduced error pruning. 3, **2012**.

[19] Victor E Lee, Lin Liu, and Ruoming Jin. *Decision trees: Theory and algorithms*. Chapman and Hall/CRC, **2014**.

[20] Lior Rokach and Oded Maimon. *Decision Trees*, pages 165–192. Springer US, Boston, MA, **2005**. ISBN 978-0-387-25465-4. doi:10.1007/0-387-25465-X_9.

[21] Mona Al Hamad and Ahmed M. Zeki. Accuracy vs. cost in decision trees: A survey. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 1–4. **2018**. doi:10.1109/3ICT.2018.8855780.

[22] Introduction to data mining (second edition).

[23] S. Singh and P. Gupta. Comparative study id3, cart and c4.5 decision tree algorithm: A survey. *International Journal of Advanced Information Science and Technology*, 3(27):97–103, **2014**.

[24] J. R. Quilan. Decision trees and multi-valued attributes. *Machine intelligence*, pages 305–318, **1988**.

[25] Randy L. Haupt and S. E. Haupt. *Practical genetic algorithms*. John Wiley, **2004**.

[26] S. N. Deepa and S. N. Sivanandam. *Introduction to genetic algorithms*. Springer, **2010**.

[27] A. Papagelis and D. Kalles. Ga tree: genetically evolved decision trees. In *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*, pages 203–206. **2000**. doi:10.1109/TAI.2000.889871.

[28] Z. Bandar, H. Al-Attar, and K. Crockett. Genetic algorithms for decision tree induction. In *Artificial Neural Nets and Genetic Algorithms*, pages 187–190. Springer Vienna, Vienna, **1999**. ISBN 978-3-7091-6384-9.

[29] Rodrigo Barros, Márcio Basgalupp, Andre de Carvalho, and Alex Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42:291–312, **2012**. doi:10.1109/TSMCC.2011.2157494.

[30] How the genetic algorithm works. `https://de.mathworks.com/help/gads/how-the-genetic-algorithm-works.html`. Accessed: 2023-12-26.

[31] Melanie Mitchell. *An introduction to genetic algorithms*. MIT, **1998**.

[32] Eyal Wirsansky. *Hands-on genetic algorithms with python: Applying genetic algorithms to solve real-world deep learning and artifical Intelligence Problems*. Packt Publishing, **2020**.

[33] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Pearson, **2013**.

[34] *Genetic Algorithm and Direct Search Toolbox*. The MathWorks, Inc.

[35] Stanley Phillips Gotshall and Bart Rylander. Optimal population size and the genetic algorithm. **2002**.

[36] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and V. B. Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12), **2019**. ISSN 2078-2489. doi:10.3390/info10120390.

[37] Seyyedeh Newsha Ghoreishi, Anders Clausen, and Bo Nørregaard Jørgensen. Termination criteria in evolutionary algorithms: A survey. In *International Joint Conference on Computational Intelligence*. **2017**.

[38] Martín Safe, Jessica Carballido, Ignacio Ponzoni, and Nélida Brignole. On stopping criteria for genetic algorithms. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 405–413. Springer Berlin Heidelberg, Berlin, Heidelberg, **2004**. ISBN 978-3-540-28645-5.

[39] Isaac Diego, Ana Redondo, Rubén Fernández, Jorge Navarro, and Javier Moguerza. General performance score for classification problems. *Applied Intelligence*, 52, **2022**. doi:10.1007/s10489-021-03041-7.

[40] Mohammad Hossin and Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining Knowledge Management Process*, 5:01–11, **2015**. doi:10.5121/ijdkp.2015.5201.

[41] Kai Ming Ting. *Confusion Matrix*, pages 209–209. Springer US, Boston, MA, **2010**. ISBN 978-0-387-30164-8. doi:10.1007/978-0-387-30164-8_157.

[42] Bradley J Erickson and Felipe Kitamura. Magician's corner: 9. performance metrics for machine learning models, **2021**.

[43] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, **2009**. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2008.08.010.

[44] Zhiyuan Chen, Bing Liu, Ronald Brachman, Peter Stone, and Francesca Rossi. *Lifelong Machine Learning*. Morgan & Claypool Publishers, 2nd edition, **2018**. ISBN 1681733021.

[45] Marcus Brubaker. Adaboost lecture notes. `https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf`.

[46] Prakhar Kaushik, Alex Gain, Adam Kortylewski, and Alan Yuille. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping, **2021**.

[47] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, **2017**. doi:10.1073/pnas.1611835114.

[48] Gido M. van de Ven and Andreas Savas Tolias. Three scenarios for continual learning. *ArXiv*, abs/1904.07734, **2019**.

[49] R. A. Fisher. Iris. UCI Machine Learning Repository, **1988**. DOI: https://doi.org/10.24432/C56C76.

[50] Apoorva Mishra and Anupam Shukla. Analysis of the effect of elite count on the behavior of genetic algorithms: A perspective. In *2017 IEEE 7th International Advance Computing Conference (IACC)*, pages 835–840. **2017**. doi:10.1109/IACC.2017.0172.

[51] Pumpkin Classification. Kaggle, **2022**. `https://www.kaggle.com/code/yatindeshpande/pumpkin-classification`.

[52] Heart failure clinical records. UCI Machine Learning Repository, **2020**. DOI: https://doi.org/10.24432/C5Z89R.

[53] Dariusz Jankowski and Konrad Jackowski. Evolutionary algorithm for decision tree induction. In Khalid Saeed and Václav Snášel, editors, *Computer Information Systems and Industrial Management*, pages 23–32. Springer Berlin Heidelberg, Berlin, Heidelberg, **2014**. ISBN 978-3-662-45237-0.

[54] Kristen Gorman. Penguin dataset. Kaggle, **2021**. `https://www.kaggle.com/code/parulpandey/penguin-dataset-the-new-iris`.

[55] Niewczas Jerzy Kulczycki Piotr Kowalski Piotr Charytanowicz, Magorzata and Szymon Lukasik. seeds. UCI Machine Learning Repository, **2012**. DOI: https://doi.org/10.24432/C5H30K.