UNIVERSITÄT
KOBLENZ · LANDAU
Department 4: Computer Science

Comprehensive study of hyperparameter tuning of ML models (Decision Tree and Random Forest)

Project internship
in computer science
with an application subject in business informatics

Submited By:

Parth Thummar
Chirag Khokhar
Ravi Babariya
Jaydipkumar Savaliya

Reviewer: Prof. Dr. Thomas Burkhardt, Abteilung Finanzierung, Finanzdienstleistungen und eFinance, Fachbereich 4: Institut für Management, Universität Koblenz-Landau
Second reviewer: Dipl. Inf. Heiko Neuhaus, Abteilung Finanzierung, Finanzdienstleistungen und eFinance, Fachbereich 4: Institut für Management, Universität Koblenz-Landau

Koblenz, August 2023

Abstract

Decision tree classifiers play a crucial role in data classification. They are simultaneously highly straightforward and well-established. Among various decision tree learning approaches, the top-down method is the most effective. It operates by iteratively introducing new splits based on individual features to minimize a given split criterion. Nevertheless, it is possible to explore alternative split methodologies, including those that consider multiple features simultaneously, to enhance the efficiency of this approach and generate more concise and accurate decision trees. It's worth noting that this kind of work is considerably more challenging and necessitates the application of effective optimization algorithms to discover the best solutions. In the context of this research, a comparative study of the three most widely used optimization methodsâRandom Search, Grid Search, and Genetic Algorithm is presented with respect to decision trees and random forests. The experimental results from this study, conducted using the Rice (Cammeo and Osmancik) and heart disease datasets, serve to elucidate the distinctions between these algorithms and offer insights into their comparative performance. The comparison is mainly based on the accuracy and complexity of the model.

Keywords: Machine Learning, Decision Tree, Random Forest, hyperparameter optimization

Explanation

We certify that We wrote this work independently and did not use any sources or resources other than those specified and that the work No other examination authority has done so in the same or similar form and from this as part of an examination was accepted. All statements, literal or substantive were adopted are marked as such.

We agree with the placement of this work in the library. ja ☒ nein ☐

We agree to the publication of this work on the Internet. ja ☒ nein ☐

Koblenz, October 31, 2023

Contents

Chapter 1

introduction

Machine learning holds significant utility across diverse applications such as computer vision, recommendation systems, autonomous vehicles, advertising, and photo identification, owing to its general-purpose nature and strong performance in the realm of data analytics. Specific types of challenges or datasets show more compatibility with machine learning techniques. Formulating a proficient machine learning model requires meticulous consideration of multiple factors, including selecting an algorithm and optimizing the model's hyperparameters to attain an optimal architecture. Within the framework of a machine learning model, there are two distinct types of parameters. The developer is responsible for initializing and making subsequent modifications during the machine learning process for these parameters. The first group comprises "model parameters." The second category, known as "hyperparameters," must be set by the developer before the training of the model, as they define the fundamental architecture of the model. [YS20, p. 2]

The core of machine learning is optimization, and most problems in machine learning are reduced to optimization challenges. Consider a machine learning analyst tackling a problem with a specific dataset. The analyst selects a suitable model family and preprocesses the data to align it with modeling requirements, initiating the problem-solving process. Typically, the model is trained by solving an initial optimization problem, which seeks to maximize the model's variables or parameters based on the given loss function

and possibly a regularization function [BPH06, p. 1266].

The optimization function involves multiple hyperparameters that are set prior to the learning process. These hyperparameters have a significant impact on how the machine-learning algorithm responds to the data. The developer must identify the set of hyper-parameter values that yield the best results on the data within a reasonable amount of time before proceeding to the training phase. This procedural effort, crucial for enhancing the predictive power of machine learning algorithms, is referred to as "hyperparameter tuning" or "optimization" [WCZ+19, p. 27].

When constructing a model through training, a classification algorithm discerns the relationship between input and output attributes. In the realm of machine learning, the decision tree method is an excellent choice for classification tasks, particularly when the dataset is well-structured with few nodes. Its objective is to make highly accurate predictions of the target class. [CA21, p.20]

A decision tree is a tree-based method in which each path is defined by a sequence of data splits that begins at the root and terminates at a leaf node, producing a Boolean output. It operates as a hierarchical representation of knowledge relationships with its nodes and connections. In the context of classification, nodes act as decision points. Decision trees are among the most effective techniques and find frequent use in various fields, including pattern recognition, image processing, and machine learning. They are particularly valuable for tasks related to grouping and categorization, especially in the context of data mining. Each tree consists of branches and nodes, where each node represents a specific feature. There exist various decision tree algorithms, each has its upper hand with a specific purpose and application in mind. Some of these algorithms include Iterative Dichotomiser 3 (ID3), its successor known as Classification and Regression Tree (CART), and the Generalized and Unbiased method[CA21, p.20].

1.1 Similar work

The study conducted by Tong Yu and Hong Zhu [YZ20] offers a comprehensive review and comparison of various algorithms for hyperparameter tuning, including methods like Grid Search, Random Search, Bayesian Optimization, Tree Parzen Estimators, and more. This research provides a thorough exploration of automated hyperparameter optimization (HPO) for machine learning models, with a specific focus on deep neural networks. The paper delves into critical hyperparameters related to model training and structure, discussing their significance and methods for defining their value ranges. It also evaluates major optimization algorithms, considering their efficiency and accuracy, particularly in the context of deep learning networks. The study further assesses significant services and toolkits for HPO, comparing their support for advanced search algorithms, compatibility with major deep-learning frameworks, and adaptability for custom modules created by users. In conclusion, the paper addresses the challenges that arise when applying HPO to deep learning, compares various optimization algorithms, and highlights prominent approaches for model evaluation under limited computational resources.
This paper is available on Google Scholar by searching for "hyperparameter optimization," and it has been cited by 462 articles, indicating its significant impact and relevance in the field of machine learning and hyperparameter optimization.

The study conducted at [MHC+16] investigated various hyperparameter tuning algorithms. For their research, they employed the J48 decision tree algorithm, specifically the Weka implementation of the algorithm, to compare different tuning algorithms. The algorithms under comparison included Genetic Algorithm, Particle Swarm Optimization (PSO), and Estimation of Distribution Algorithm (EDA). Through experimentation involving 102 datasets, they determined that the Estimation of Distribution Algorithm (EDA) was the most effective optimization technique for their purposes.
This paper is available on Google Scholar, and you can find it by searching for the term "hyperparameter tuning of decision tree." It has been cited 110 times, indicating its relevance and impact in the field of hyperparameter tuning for decision tree algorithms.

In a study conducted at [AMS+21], the Authors did research on heart disease diagnosis techniques with the help of machine learning techniques. However, challenges associated with datasets such as missing data, inconsistent data, and mixed data (containing inconsistent missing data both as numerical and categorical) are often obstacles in medical diagnosis. They performed Feature scaling on 11 ML algorithms and concluded that The SVM and CART algorithms showed the highest accuracy (99%) compared with the other nine algorithms when applied without any scaling techniques. On the other hand, KNN showed the lowest performance with 75% accuracy. However, the overall performance improved up to 12% with the use of the MaxAbs scaling method. The study result also revealed that the overall performance is similar with or without using data scaling techniques for all algorithms except LR, KNN, and SVM. CART achieved an accuracy of 100% when implemented with Robust Scaler and Quantile Transformer methods. On the other hand, the performance of SVM reduced drastically from 99% to around 63% while using Normalization techniques. However, among all data scaling methods, with StandScale, SVM showed a higher accuracy at around 92%. To sum up, with data scaling methods, CART outperformed all other ML algorithms in the heart disease dataset. A previous study conducted by [TSS09] showed that Naive Bayes outperformed all other methods, this study found that NB has the lowest accuracy when used with different scaling approaches.

The Authors in the study [LL19] compared three of the most widely used algorithms for hyperparameter optimization: Grid Search, Random Search, and the Genetic Algorithm. Their investigation focused on assessing the suitability of these algorithms for neural architecture search (NAS). The results revealed that nearly all the algorithms tested exhibited extended search times in the space of distributions. Notably, encoding the number of parameters as a genome yielded the most promising results, although it might still require some time for the evolutionary process to arrive at the optimal model. Furthermore, the authors also addressed the challenges associated with hyperparameter optimization. They concluded that it is challenging to definitively determine which of the algorithms will consistently yield superior results. Grid search or random search might be

effective choices for smaller models with simpler search spaces. However, the evolutionary algorithm appears to be the most suitable option for scenarios involving models with numerous layers and expansive search spaces.

1.2 central question of the work

The aim of this study is to investigate and explore various hyperparameter tuning techniques to improve the performance of a machine learning model, with a specific focus on fine-tuning the parameters of a Decision Tree algorithm. Hyperparameter tuning is a critical aspect of machine learning model development, as it allows for the optimization of a model's performance and its ability to generalize effectively to new and unseen data.

1.3 Structure of the work

In this swift study, various research on the optimization of machine learning models has been reviewed. Chapter 2 of this study provides an extensive discussion of methodologies such as Decision Trees, Random Forest, Random Search, Grid Search, and Genetic Algorithms.

For Decision Trees, the section begins with an introduction to the concept of Decision Trees. Subsequently, pseudocode is presented to illustrate a simple Decision Tree algorithm, facilitating a better understanding of its workings. The chapter further covers different types of Decision Tree algorithms, including ID3, C4.5, CHAID, and CART. It also delves into the fundamental components of Decision Trees, their advantages, and limitations. In the sections dedicated to Random Search and Grid Search, a brief introduction is provided to explain how these search techniques operate and how they discover optimal parameters for machine learning models. In the Genetic Algorithm section, the study follows a structured approach. It begins with an introduction to Genetic Algorithms and their basic structure, outlining the key steps followed in these algorithms. The study then proceeds to explain various components of Genetic Algorithms in detail, providing a comprehensive understanding of their inner workings. This detailed exploration in this

section offers valuable insights into Decision Trees and the optimization techniques discussed in the study. In the third part of the second chapter, the study describes a series of experiments conducted, with a focus on Decision Trees. The study initially observes the effects of parameters on Decision Trees. Subsequently, it applies a Random Search and presents the results obtained. Grid Search is then employed, followed by the utilization of Genetic Algorithm (GA). The results obtained from these experiments are discussed and likely compared to analyze the effectiveness of each technique.

In the sixth part of the second section, the study discusses the methods for model evaluation. It provides an explanation of two common evaluation metrics, accuracy and F1-score. These metrics are essential for assessing the performance of machine learning models and are often used to measure the model's accuracy in making predictions and its ability to balance precision and recall, particularly in classification tasks. This section likely delves into the details of these metrics and their significance in evaluating the model's performance. In the seventh part of the second section of chapter 2, the concept of cross-validation is explained.

In the final chapter, the study concludes with a summary of the findings and offers conclusions based on the research. It may also suggest potential avenues for future studies and research directions. Datasets used in the experiment are also presented.

Chapter 2

Methodology

2.1 Description of the method

In this research paper, our focus is on the optimization of the decision tree because it provides an effective method for making decisions because it finds all possible outcomes. It enables developers to analyze the possible consequences of a decision and as an algorithm accesses more data, it can predict outcomes for future data. The optimization methods that will get used are random search, grid search, and genetic algorithm and the datasets are heart disease [JD88] and Rice (Cammeo and Osmancik) [mis19] which is taken from UCI repository. The reason behind the usage of random search techniques is that whenever the number of combinations increases it is best and less computationally expensive. While Grid search tries all possible combinations and the Genetic algorithm works on a fitness function, so the accuracy depends on the fitness function. For the practical the libraries used are scikit-learn [PVG+11], matplotlib [Hun07] and seaborn [Was21] from Python programming language. Here the scikit learn is used for the computational process and matplotlib is used for visualization and the get access of confusion matrix seaborn is used. In the first phase of practical optimization performed with default parameters after that Random search, Grid search, and at last Genetic algorithm method are used respectively so the conclusion can be made depending on the results of these optimization methods. For research max_depth, criterion, min_samples_split, min_samples_leaf are chosen as hyperparameters for the decision tree because they are the most important parameters.

2.2 Detailed introduction to the topic

In this section, an in-depth explanation is provided for all the methods employed in the study. These detailed explanations aim to give readers a comprehensive understanding of the methodologies and techniques utilized in the research.

2.2.1 Decision Tree

In its most elementary definition, decision tree analysis constitutes a divide-and-conquer approach to classification (and regression, which lies beyond the purview of this review). Decision trees serve as a valuable tool for unearthing features and discerning patterns within extensive databases, pertinent to discrimination and predictive modeling. These attributes, in conjunction with their easily interpretable nature, underscore why decision trees have found extensive application in the realms of exploratory data analysis and predictive modeling for over two decades. It is worth noting that decision trees possess a well-established foundation in the domains of both machine learning and artificial intelligence, and they are gradually carving out a niche for themselves in the chemical and biochemical sciences. [MFL+04, p. 275]

The decision tree algorithm belongs to the supervised learning algorithm family, and it can handle both regression and classification tasks. Its main purpose is to establish a training model that can extract decision rules from the training data, enabling it to predict the class or value of target variables. One of the standout characteristics of the Decision Trees algorithm is its simplicity, making it relatively easier to understand compared to other classification methods. It utilizes a tree structure for this purpose, with each leaf node representing a class label and each internal node representing an attribute. In the context of decision trees, the process of predicting a record's class label begins at the root of the tree. We initiate this by comparing the values of the record's attribute with the attribute specified at the root. Based on this comparison, we follow the branch corresponding to the attribute's value, leading us to the next node in the tree. We continue this process, comparing the attribute values of our record with those of internal nodes in

the tree, until we reach a leaf node that offers a predicted class value. Ultimately, the decision tree model enables us to forecast the target class or value for the given record. Now, let's delve into how to construct a decision tree model [Sax17].

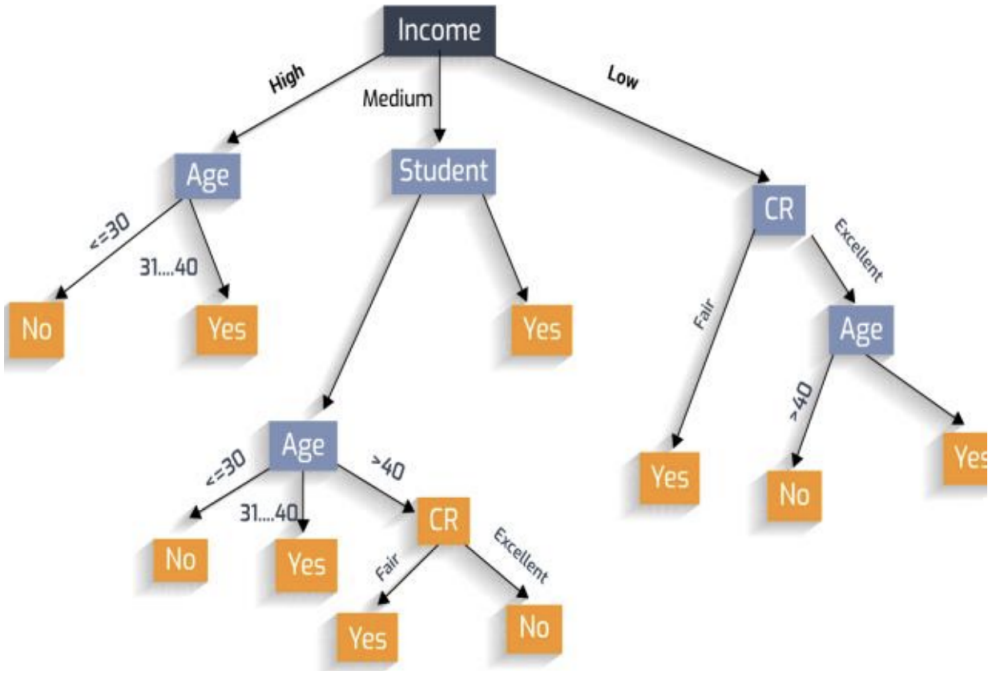


Figure 2.1: Example Of Decision Tree [BDS10, p. 19]

Pseudocode of Decision Tree

Pseudocode for Decision Tree is [Sax17]:

Step 1: Begin with the entire dataset and choose the best attribute as the root node of the tree.

Step 2: Generate subsets of the training data by grouping examples with the same attribute values.

Step 3: Recursively repeat steps 1 and 2 on each subset until you reach leaf nodes in every branch of the tree.

Principle of Decision Tree Construction

1. The entire training set is initially treated as the root of the tree.

2. Categorical feature values are preferred. If the features are continuous, they are discretized (divided into categories or intervals) before model construction.

3. Records are recursively divided based on attribute values.

4. A statistical technique is utilized to determine which attributes should serve as the tree's internal or root nodes.

These principles provide guidance for building a decision tree model, ensuring that it effectively captures the relationships between attributes and the target variable [Sax17].

Assortment of Decision Tree Algorithms:

An array of decision tree algorithms is available to facilitate the creation of these models, each characterized by unique attributes and construction techniques. Among the well-established decision tree algorithms are:

A. Iterative Dichotomiser 3 (ID3)

Within corporate environments, data mining techniques serve as powerful tools, offering valuable insights to inform decision-making processes. Classification plays a pivotal role in numerous corporate domains, including the banking industry, where it is employed for predictive task such as anticipating the class of an object. ID3 is a classification algorithm founded on decision trees. The fundamental principle of ID3 involves the computation of information gain as a splitting metric, with all training samples utilized at each step of the search process. This methodology, although time-consuming, yields lower error probabilities. While some researchers have proposed parallel classification techniques, it's essential to acknowledge that these algorithms are not without their limitations. One significant limitation is the necessity to present each data item in the dataset to every processor, which can be time-consuming, especially when dealing with large-scale datasets. [KIES16, p. 70-71]

B. Classification and Regression Tree (C4.5)

Research in the domains of data mining and machine learning has seen substantial attention directed toward classification algorithms. Among these algorithms, Quinlan's C4.5 system holds a prominent position for several reasons. Firstly, it is an evolution of the earlier ID3 system, representing a significant advancement in machine learning research. Consequently, C4.5 has long been regarded as the foundational platform for the development and assessment of innovative concepts in this field.

On the other hand, the C4.5 tree-induction technique distinguishes itself as one of the swiftest main-memory approaches in the realm of machine learning and data mining. It offers commendable classification accuracy, as demonstrated by research outcomes [Rug02, p. 1]. The most computationally intensive aspect of the C4.5 System is, by far, the tree construction algorithm. The evaluation and simplification processes entail traversing the decision tree to ascertain the classes of cases in the training and test sets or to verify pruning conditions. The time allocated to simplification and evaluation forms only a minor portion of the overall execution time, except for small training sets. Hence, the primary emphasis is on tree construction [Rug02, p. 439].

C. Chi-squared Automatic Interaction Detector (CHAID)

The Bonferroni test serves as the foundation for the algorithmic analysis of the decision tree model, known as Chi-squared Automatic Interaction Detection (CHAID). This approach results in the creation of a tree diagram that elucidates the connections between splitting variables and associated factors, facilitating the identification of homogeneous population segments.

In contrast to logistic regression analysis, which examines risk factors across the entire population and treats various factors equally, CHAID decision tree analysis facilitates the division of the population into subgroups characterized by distinct attributes and allows for the estimation of prevalence within each subgroup. The combination of these two approaches helps pinpoint the target population and enhances the comprehension of the significance of potential determinants[YCC+16, p. 1198].

D. Classification And Regression Tree (CART) :

The CART decision tree, a binary recursive partitioning method, is capable of processing both continuous and nominal attributes as targets and predictors. It doesn't require or advocate for data binning; data are processed in their original form. The data is initially split into two child nodes at the root, and each child is further subdivided into grandchildren. The tree construction process proceeds to its maximum extent without the application of a stopping rule. In other words, the tree-building process continues until no more splits are feasible due to insufficient data. Subsequently, employing the novel approach of cost-complexity pruning, the initially extensive tree is systematically pruned, incrementally trimming it back to the root by evaluating each split. The split with the least impact on overall performance concerning the training data is the next one to be pruned (more than one split can be removed simultaneously). The primary objective of the CART mechanism is to construct a set of nested pruned trees, each of which represents a potential optimal tree. This approach deviates from producing a single tree and focuses on exploring multiple pruned alternatives [WK09, p. 181].

In this discussion, we will focus on the more renowned ones, namely ID3, C4.5, and CART. Each of these algorithms contributes distinct strengths and methodologies to the process of constructing decision trees, accommodating a variety of applications and data types. The selection of the most appropriate algorithm is contingent on the specific requirements and characteristics of the dataset in question [GRJ+17, p. 15-16].

Fundamental Components of Decision Tree:

In a decision tree model, the fundamental components are nodes and branches, and the three most critical phases in the model-building process involve splitting, stopping, and pruning. [SY15, p. 2]

Nodes:

Nodes within decision trees can be categorized into three distinct types:

(a) Root nodes : also known as decision nodes, represent pivotal decisions that partition all records into two or more mutually exclusive subsets.

(b) Internal nodes : Sometimes referred to as chance nodes, indicate potential options that can be selected at specific points within the tree structure. These nodes are connected to both their parent and child nodes through their top and bottom edges, respectively.//
(c) Leaf nodes : often termed end nodes, convey the ultimate outcomes resulting from a series of decisions or occurrences.

Branches:

Branches in decision trees correspond to random events or outcomes that emanate from internal and root nodes. These branches collectively form a hierarchical structure, constituting the foundation of a decision tree model. Each pathway, leading from the root node through internal nodes to a leaf node, represents a distinct classification decision rule. These decision tree pathways can also be expressed in the form of "if-then" rules. For example, one might articulate it as follows: " Outcome j occurs if conditions 1, 2, and so forth, up to condition k, are met." [SY15, p. 2]

Splitting:

Parent nodes are strategically divided into more homogenous child nodes about the target variable. This division is accomplished by employing relevant input variables associated with the target variable. Both continuous input variables, which can be discretized into multiple categories, and discrete input variables are considered in this process. Prior to constructing the model, it is imperative to identify the most crucial input variables. Subsequently, records at the root node and subsequent internal nodes are partitioned into two or more categories, often referred to as "bins," based on the status of these variables. The selection of the most suitable input variables is determined by assessing the purity of the resulting child nodes, measured by the proportion with the target condition. Several criteria, including entropy, the Gini index, classification error, information gain, and gain ratio, are utilized to gauge this purity and guide the variable selection process [SY15, p. 2].

Stopping:

In the development of a statistical model, it is imperative to consider the delicate balance

between the conflicting attributes of robustness and complexity. As a model becomes more complex, its capacity to predict future data tends to diminish. An extreme illustration of complexity is the creation of an exceedingly intricate decision tree model with an extensive spread designed to ensure 100% purity in every record within each leaf node, signifying that each record exhibits the desired outcome. However, this approach represents an extreme scenario. Such a decision tree would be overly tailored to the specific characteristics of the current observations, resulting in a scarcity of records within each leaf node. Consequently, it would lack the capability to predict future cases with a reasonable level of reliability and would exhibit poor generalizability, ultimately lacking robustness. When constructing a decision tree, it is crucial to implement halting rules to prevent the model from becoming excessively complex. This helps avoid issues of overfitting. Common parameters utilized in these halting rules include the minimum number of records required in a leaf, the minimum number of records needed in a node before initiating a split, and the maximum depth, representing the number of steps from any leaf to the root node. The selection of appropriate stopping settings should consider the specific goals of the study and the characteristics of the dataset in use. it is generally recommended that the proportion of records within a leaf node should fall between 0.25% and 1.00% of the entire training dataset. This range helps strike a balance, preventing both overfitting and underfitting of the model [SY15, p. 2].

Pruning:

Stopping rules do not always yield a straightforward outcome. An alternative method for constructing a decision tree model involves initially generating a large tree and subsequently pruning it down to an appropriate size by removing nodes that do not contribute additional information. One widely adopted technique for selecting the optimal subtree from a range of candidates is to consider the percentage of records with prediction errors, or the percentage of records where the anticipated occurrence of the target variable is inaccurate. Another approach to determining the best option is by employing a validation dataset. This entails dividing the sample into two parts and evaluating the model created on the training dataset using the validation dataset to gauge its performance, or, Cross-

validation, involving the division of the sample into ten groups or "folds" and assessing the model created from the first nine folds on the tenth fold, followed by repeating this process for all ten combinations and aggregating the error rates or incorrect predictions, is a valuable technique when working with small datasets. Pruning methods encompass both pre-pruning (also known as ahead pruning) and post-pruning (or reverse pruning). In pre-pruning, techniques like Chi-square tests or multiple-comparison adjustments are employed to prevent the growth of non-significant branches in the decision tree. Post-pruning, on the other hand, occurs after constructing a complete decision tree and aims to remove branches in a way that enhances overall classification accuracy when applied to the validation dataset [SY15, p. 3].

Scoring Criteria for Decision Tree Partitions:

Several scoring standards have been developed to evaluate decision tree partitions. This article will focus on two specific scoring criteria, namely Information Gain (Info Gain) and the Gini Index (Gini), to enhance clarity and comprehension. These criteria are described by the following equation, where Nj represents the number of samples belonging to class j, N(t) is the total number of samples in node t, and Nj(t) is the number of samples in node t that belong to class j. Equation 2.1 encapsulates the concept of information, or Info.

$$Info = -\sum_{j} \frac{N_j(t)}{N(t)} \log_2 \frac{N_j(t)}{N(t)} \tag{2.1}$$

The Info Gain, which signifies the partition value that maximizes the change in information, is selected. We employ the following equation to compute Info Gain, where pk represents the proportion of samples allocated to the kth subspace, and Info(q) denotes the information of the feature subspace q.

$$infoGain = Info(Parent) - \sum_{k} (P_k) Info(Child_k) \tag{2.2}$$

The Gini Index measures the extent to which partitioning the feature space reduces class impurity. The symbol ||g|| represents the normalization of vector g, while p(j) signifies the prior probability that a sample belongs to class j within a set of training samples. In the MATLAB decision tree code for the microarray example, proportional priors are applied to align with the default option, where p(j) is defined as Nj/N(1).

$$impurity = 1 - \sum_{j} ||p(j)N_j(t)/N_j||^2 \tag{2.3}$$

$$Gini = impurity(Parent) - \sum_{k} (p_k) impurity(Child_k) \tag{2.4}$$

In general, multiple scoring standards are often applied sequentially, especially in binary classification systems. The choice of feature selection can vary based on the scoring criteria, depending on factors such as sample size and class distribution. It's important to note that while there is no direct comparison between the absolute values of the two scoring criteria, they each offer unique insights into the classification process [MFL+04, p. 277].

Advantages and Limitations of Decision Trees

Expanding beyond the technical aspects, let's delve into the broader context of decision trees. Within the realm of supervised learning, the decision tree (DT) algorithm stands as a fundamental tool employed in the creation of predictive models. These models leverage decision rules extracted from training data to aid in predicting the assignment of class or value to target variables. The DT algorithm is noted for its versatility, as it can tackle issues related to both regression and classification. However, it's imperative to consider its associated merits and demerits [ZZ08, KMT17, PK20, p. 1955-1959,p. 111-115, p. 246-269],], which are comprehensively delineated below, encompassing various facets of its applicability in practical scenarios.

Advantages of Decision Trees:

Effective Cost Structure: The cost of utilizing decision trees for data prediction exhibits a logarithmic relationship with the quantity of training data points. This cost-effectiveness is particularly advantageous when handling large datasets.

Versatile Data Handling: Decision trees possess the capability to effectively handle both numerical and categorical data. In contrast, certain other methods are tailored exclusively for one type of variable. This flexibility broadens their applicability.

Intuitive Visualization: Decision trees offer the advantage of being easily visualized and comprehended. Their graphical representation facilitates a clear and straightforward understanding of the decision-making process.

Minimal Data Preparation: Unlike many other data analysis techniques, decision trees necessitate minimal data preparation. This streamlines the analysis process by reducing the need for complex data normalization, the creation of dummy variables, and handling missing values.

Transparency of Decision Logic: Decision trees exemplify a "white box" model, making their decision logic easily understandable in terms of Boolean logic, given their primarily binary outputs ("yes" or "no"). This transparency enhances comprehension and interpretation.

Robustness to Assumption Violation: Decision trees exhibit resilience, performing effectively even when there is some deviation from the underlying assumptions of the dataset. This makes them valuable in scenarios where data does not strictly conform to certain assumptions.

Challenges with Decision Trees:

Optimal Decision-Making Challenges: Identifying the most optimal decision-making mechanism presents a significant challenge with decision trees. Suboptimal tree structures may deter optimal decision paths, leading to subpar choices.

Complexity with Multiple Layers: Decision trees with multiple layers can become highly intricate. This complexity may impede interpretation, especially in the case of deep trees with numerous branches.

Increasing Computational Complexity: As datasets grow in size and more train-

ing samples are incorporated, decision tree algorithms may exhibit greater computational complexity. This implies that larger datasets may demand enhanced processing power for accurate results.
In conclusion, decision trees offer numerous desirable qualities, including simplicity, effectiveness, and versatility. Nevertheless, they also exhibit certain limitations, such as the need to manage complexity and optimize decision pathways, particularly when dealing with extensive datasets. These considerations underscore the importance of judiciously applying decision trees and adjusting them as needed for various data analysis and prediction tasks.

2.2.2 Random Forest

A Random Forest represents a supervised machine learning algorithm in which the combined computations of several decision trees yield a single conclusive outcome. Its attractiveness lies in the fusion of simplicity and remarkable efficiency [Log22].

In the realm of classification trees, two well-established methods are boosting [SS98, p. 20-22] and bagging[Bre96, p. 20-22]. In boosting, a series of trees assign additional weight to data points that previous models erroneously predicted. Predictions are ultimately determined through a weighted voting process. In contrast, bagging entails the independent construction of successive trees without reference to their predecessors, as delineated in reference[Bre01, p. 11]. Each tree is fashioned using a bootstrap sample extracted from the dataset, and predictions are established through a simple majority vote.

Conversely, the Random Forest (RF) classifier comprises several trees, each grown in a somewhat random manner (see 2.2). Leaf nodes of these trees indicate estimates of the posterior distribution related to the image classes. As elucidated in [CXW+17, p. 151-155], internal nodes within each tree contain tests that optimally partition the data space for classification purposes. An image traverses each tree for classification within the RF, and the resultant leaf distributions are subsequently amalgamated. To ensure that every tree is grown using a distinct subset, randomness is introduced at two critical

junctures in the training process: firstly, by subsampling the training data, and secondly, by selecting the node tests, as mentioned in [RGCR14, p. 494-495].

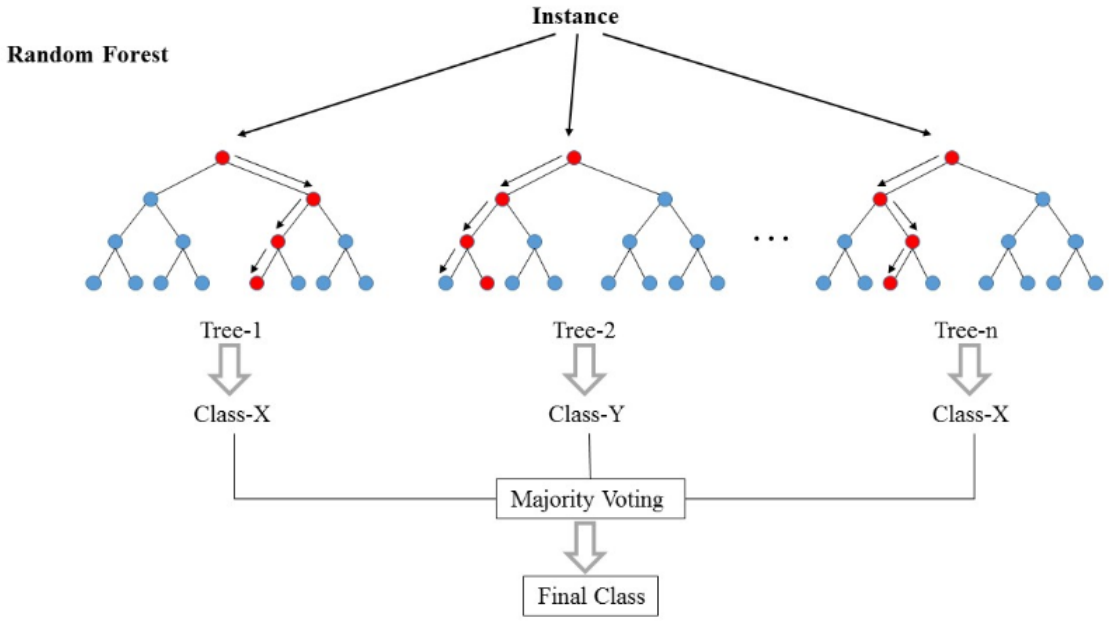


Figure 2.2: A) Random Forest Graph [**?**, p. 965]

The quantity of predictors in the Random Forest model dictates the number of trees required to attain optimal performance. A practical approach involves comparing the predictions made by the entire forest with those generated by a subset of the forest to ascertain the ideal number of trees. The ensemble includes an adequate number of trees when the performance of the subset equals that of the entire forest.

Breiman[Bre01, p. 14-15] suggests experimenting with different values when selecting the 'mtry' parameter, including the default setting, half of the default, and twice the default. This technique aids in determining the optimal 'mtry' value. In situations where a dataset comprises numerous variables but only a small subset is deemed "important," the utilization of a higher 'mtry' value can enhance model performance.

Reliable estimates of proximity and variable importance necessitate a substantial number of trees. It is imperative to recall that the Random Forest algorithm falls within the category of "embarrassingly parallel" techniques. This characteristic enables the simultaneous execution of several Random Forests on distinct machines. As explained in [EB17, p. 144-145], the final prediction can then be obtained by averaging the results of these parallel computations.

In contrast to bagging, the Random Forest (RF) classifier introduces an additional layer of randomness, as explained in [Bre01, p. 10-12]. Apart from generating unique bootstrap samples for every tree, RFs employ a novel method when constructing regression or classification trees. Traditional trees utilize the best possible split from all available variables to determine decisions at each node. In contrast, RF selects the optimal split from a randomly chosen subset of predictors to determine the decision at each node. Despite its apparent counterintuitiveness, this non-traditional approach outperforms several other classifiers, such as discriminant analysis, Support Vector Machines (SVMs), and Neural Networks (NNs). Notably, as mentioned in [Bre01, p. 10-12], RF demonstrates robustness against overfitting.

Furthermore, Random Forest (RF) proves highly user-friendly due to its minimal parameterization. The critical factors to consider are the number of variables in the random subset at each node and the total number of trees in the forest. Significantly, as highlighted in [WLC+15, p. 1134-1135], RF typically exhibits a relatively low sensitivity to changes in these parameter values.

In essence, RF constitutes a group of Decision Trees (DTs), with each tree casting votes for the class assignment of a particular sample. The final class assignment is determined through a majority vote among these trees, as detailed in [SS15b, p. 8376]. As mentioned in [Bre01, p. 15], RF demonstrates outstanding efficiency in handling categorical features. It is also well-suited for managing large training datasets and high-dimensional feature spaces. Due to its adaptability, radio frequency technology finds wide application across

various fields.

A predictive model structured hierarchically with a set of conditions is known as a decision tree (DT). Instances are categorized by following the path of satisfied conditions from the root of the tree to a leaf node associated with a specific class label, as noted in [WB09, p. 451]. DTs can also be readily transformed into a set of classification rules.

The following procedures are adhered to in constructing a Random Forest[JS17, p. 66]:

A. Initiate with a training set containing N cases. Select each case at random to generate a unique dataset distinct from the original.

B. Examine the T attributes within this dataset randomly. Choose a subset of Tt « T attributes at random. Ensure that the optimal combination of t variables is used for node splitting at each decision tree node. Maintaining the stability of t throughout the tree growth process is crucial.

C. Allow each tree to reach its full complexity without pruning.

The performance of the Random Forest model hinges on two primary factors [JS17, p. 66] :

1. An increase in the correlation between any two trees in the forest results in an elevated error rate. Higher error rates correspond to increased correlation.

2. The strength of a tree is reflected in its error rate, with lower error rates indicating stronger trees. Collectively, robust individual trees enhance the overall strength of the forest.

Hyperparameters to Enhance Predictive Capability:

n_estimators: The hyperparameter n_estimators specifies the number of trees the algorithm constructs before averaging its predictions.

max_features: max_features indicates the maximum number of features considered by the Random Forest when splitting a node.

Hyperparameters to Boost Velocity:

random_state: By adjusting the 'random_state' hyperparameter, you can control the level of unpredictability in the data sampling process. Consistency in the 'random_state' value ensures reproducible results when the hyperparameters and training data remain consistent.

oob_score: OOB (Out-of-Bag) scoring is a unique feature of Random Forest, functioning as a form of cross-validation. Approximately one-third of the data in OOB scoring is reserved for evaluating model accuracy, as opposed to training.

n_jobs: The 'n_jobs' hyperparameter informs the engine of the maximum number of processors it can use. A value of -1 allows the algorithm to utilize all available processors for faster computation, while a value of 1 restricts it to a single processor.

2.2.3 Random Search

Random search, in contrast, selects hyperparameter combinations at random and evaluates them as part of the optimization process. The top-performing combinations are then chosen based on the evaluation results [ZZ17, SS15a]. Random search is particularly effective in handling high-dimensional datasets [PR18, p. 6-10]. The mechanics of random search are as follows:

1. Selection of Random Hyperparameter Combinations: The first step in the random search process involves selecting hyperparameter combinations at random, within predefined ranges or bounds specified for each hyperparameter.

2. Evaluation of Combinations: A predefined performance metric or evaluation criterion is used to assess the randomly selected hyperparameter combinations. This evaluation examines the performance of each combination in the given task, such as in a machine learning model.

3. Identification of the Best Outcome: Based on the evaluations, the hyperpa-

rameter combination that yields the most favorable results according to the evaluation criterion is determined. This combination is considered the best set of hyperparameters for the specific problem.

2.2.4 Grid search

Hyperparameter tuning is a crucial aspect of machine learning model development. Two common methods for optimizing hyperparameters are grid search and random search. Grid search involves systematically assessing a model across all conceivable hyperparameter values, while random search employs a random selection of hyperparameter values. While grid search is a comprehensive approach, it can be computationally intensive, especially for models with a high number of hyperparameters. Conversely, random search is computationally more efficient but may not consistently yield the best hyperparameter values. Machine learning researchers typically strive to address problems characterized by convex objective functions with minimal dependence on hyperparameters. Such problems are easier to optimize and tend to generalize well to new, unseen data. However, in practice, hyperparameters often play a pivotal role in guiding the optimization process toward regions in the search space that enhance generalization performance.

Enhanced Grid Search Algorithm

The fundamental concept behind the enhanced grid search algorithm is to begin by defining a search region for the kernel parameter σ and the penalty factor C [JWH12, p. 15-24]. Subsequently, step lengths for C and σ are computed, and a thorough examination of parameters within these step lengths is conducted. This process generates a 2-dimensional grid, with each intersection node on the grid representing a distinct set of (σ, C) values. Identifying the global optimal solution for the parameters may take some time due to the typically extensive parameter search range. Figure 2.3 illustrates the model flow diagram for the grid search-based method.

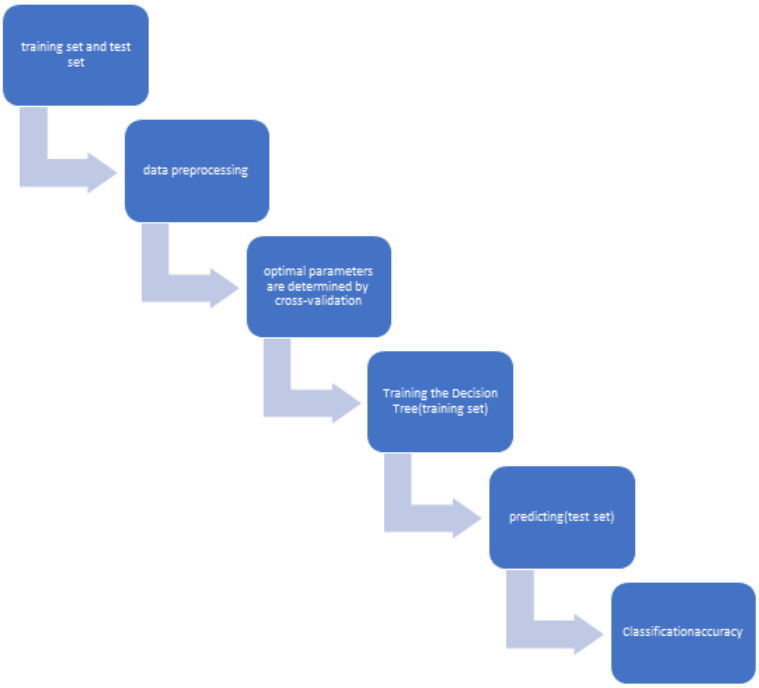


Figure 2.3: Grid Search: A Methodical Approach to Hyperparameter Optimization

Grid search is a meticulous hyperparameter optimization technique that systematically explores predefined subsets of hyperparameters. It operates by establishing both lower and upper bounds and specific limits for these hyperparameters [ISW16, p. 1502-1509]. Combinations of hyperparameter values are created, forming a grid, and these combinations are rigorously assessed to identify the optimal set of values [AO04, p. 60-78]. Grid search offers the advantage of ensuring high precision during parameter optimization [AL19, p. 1529-1533]. The operational principles of grid search are as follows:

1. **Definition of Parameter Bounds :** The initial step involves defining the lower and upper bounds for hyperparameters, outlining the range within which the optimal values will be identified.
2. **Step Size Determination:** This entails determining the number of steps or intervals within the parameter boundaries, indicating the granularity of the search.
3. **Grid Creation:** A grid of potential hyperparameter combinations is constructed,

considering the parameter constraints and step sizes. This grid encompasses a wide range of possible values.

4. Systematic Evaluation: Each combination within the grid is methodically assessed using a predefined performance metric or evaluation criterion. This evaluation may be based on the model's performance on test data, a validation set, or a cross-validation approach.

The selection of the best hyperparameters involves identifying the combination that produces the most favorable outcomes based on the evaluation criterion. This process is referred to as the selection of the best values, representing the arrangement that optimally addresses the given problem.

2.2.5 Genetic Algorithm

Genetic Algorithms (GAs) constitute a potent methodology employed in the exploration and refinement of solutions, drawing inspiration from the evolutionary and genetic selection principles initially postulated by Charles Darwin. GAs utilize a variety of mechanisms to fashion solutions customized to particular problems. They function on a population of chromosomes, usually expressed as strings using a selected alphabet, frequently in binary format. Each chromosome functions as a potential solution to the given problem and is distinguished by a fitness value that quantifies its excellence [McC05, p. 206].

The Genetic Algorithm (GA) procedure commences with the generation of a population consisting of these created chromosomes. Utilizing fitness-based selection and recombination mechanisms, the GA begets successive generations or descendant populations. In the process of recombination, parental chromosomes undergo a merging of their genetic material to generate offspring chromosomes that integrate into the descendant population. This iterative process continues until a predefined stopping criterion is met. In each iteration, subsequent generations progress, ultimately enhancing the fitness of the chromosomes. Through this iterative mechanism, GAs strive to uncover solutions for a diverse range of problem scenarios.[McC05, p. 206]

Problem Representations and Optimization Challenges in Genetic Algorithms

Within the realm of genetic algorithms, two pivotal elements that are tailored to the specific problem are the problem encoding and the evaluation function. In the context of parameter optimization problems, the objective is to fine-tune a set of variables with the aim of either maximizing profits, minimizing costs, or minimizing errors. One can envision this problem as a metaphorical black box adorned with control dials symbolizing distinct parameters. The sole output emanates from an evaluation function that quantifies the performance of a given parameter configuration. The ultimate aim is to configure these parameters to optimize the output, typically expressed as a function, such as F(X1, X2, ..., XM) [Whi94, p. 66-67].

Genetic algorithms are frequently employed in the context of nonlinear problems in which parameters interact, and their collective effects impact the output. This interplay is commonly referred to as "epistasis" within the genetic algorithm community [Whi94, p. 67]. An underlying assumption is that the parameters can be expressed as bit strings, implying that they are discretized within a predefined range. For instance, when employing 10 bits per parameter, there are 1024 distinct values. This discretization is a viable approach for continuous parameters if it provides adequate precision and accurately represents the underlying function. However, coding becomes more complex if a parameter can only take on a specific finite set of values. In such cases, solving coding issues is considered part of designing the evaluation function. [Whi94, p. 67].

The evaluation function is an integral part of the problem description but developing it can sometimes involve creating simulations or performance-based approximations. It must also be relatively fast to compute, as genetic algorithms work with populations of potential solutions. Evaluating a population can be time-consuming, particularly when the population is replaced generationally. For example, if an evaluation takes one hour, evaluating 10,000 potential solutions would take over a year, which could be roughly 50 generations for a population of 200 strings. Therefore, efficiency is a crucial consideration [Whi94, p. 67].

The Genetic Algorithm Structure

Genetic Algorithms (GAs) are search algorithms grounded in the fundamental principles of heredity and natural selection. They belong to the wider domain of evolutionary computation, which encompasses more than just GAs. Within the realm of GAs, a specific problem typically yields a multitude of diverse solutions [AL19, p. 379].The genetic algorithm adheres to a structured framework consisting of various components. This design approach enables the reusability of these components in diverse genetic algorithms, streamlining the implementation process. The fundamental components include chromosome encoding, the fitness function, the selection process, crossover, and the evolution strategy [McC05, p. 206].

The resultant solutions undergo a process of mutation and recombination, mimicking the mechanisms of biological genetics. This process results in the generation of new offspring, and this cycle repeats over multiple generations. The fitness of each individual is gauged by its objective function value, and individuals with higher fitness levels are more inclined to mate and produce progeny. As the process advances toward its ultimate termination criterion, it guarantees the emergence of individuals with enhanced fitness and superior solutions in successive generations[AL19, p. 379] .

The following is a basic steps for a genetic algorithm:

1) Initialize the population: Generate a random population of candidate solutions.
2) Evaluate the fitness of each solution: Use the fitness function to evaluate the fitness of each solution.
3) Select solutions to reproduce: Use the selection process to choose which solutions will reproduce.
4) Create new offspring: Use the combination process to create new offspring from the selected solutions.

5) Replace a portion of the population with the new offspring: Use the evolution scheme to replace a portion of the population with the new offspring.

6) Repeat steps 2-5 until a termination criterion is met: The termination criterion is typically a condition on the fitness of the best solution or the number of generations that have passed.

Algorithm A : Classical Genetic Algorithm (GA)
Initialize a population of candidate solutions.
Evaluate the fitness of each candidate solution.
Repeat for a fixed number of generations or until convergence:
Select parents based on their fitness
Create a new population by combining the genetic material of the selected parents (crossover)
Perform mutation on some of the offspring
Evaluate the fitness of the new population
Select the best-performing candidates to survive and become parents for the next generation
Return the best candidate solution found in the final generation

Algorithm 2.2.5 presents the pseudocode of a classical genetic algorithm.
This pseudocode delineates the fundamental stages of a classical genetic algorithm, encompassing the initialization of the population, the selection of parents, the crossover operation (genetic recombination), mutation, fitness assessment, and the iterative progression of generations. The algorithm persists for a predefined number of generations or until convergence criteria are satisfied, culminating in the identification of the most optimal solution [Kat21, p. 8085-8086].

Genetic Algorithm Components

a) Coding of Chromosomes

In the context of Genetic Algorithms (GAs), populations are composed of chromosomes, which serve as string representations of potential solutions to specific problems. These chromosomes can be envisioned as sequences of letters drawn from the alphabet A, C, G, T, and they bear a conceptual resemblance to biological DNA chromosomes. Genes are individual positions, or loci, within a chromosome; the letter at a specific locus is termed the allele value or simply allele. The phrase "GA encoding" pertains to the chosen representation for a given scenario [McC05, p. 207]. Traditionally, solutions are encoded using a bit-string form in GAs. Gene sequences with allele values represented by characters from the binary alphabet {0, 1} make up bit-string chromosomes. When GAs are used to solve issues, the solution space is usually finite yet too large to evaluate every conceivable solution exhaustively computationally. A GA may, for example, often work with bit strings of length 100, which defines a solution space with 2^{100} (or 10^{30}) members. The way these bit strings are interpreted depends on the specific issue at hand. For example, in one problem, a bit string of length 20 might represent a single integer value in standard binary notation, while in another context, these bits could signify the presence or absence of 20 different factors within a complex process [McC05, p. 207].

That common representation can solve a variety of problems is one of GAs' advantages. This makes it possible to create standardized operators and processing procedures, which makes it easier to adapt GAs to new problem domains. It's crucial to remember that the chromosomal encoding, by itself, doesn't have much information about a given condition. The fitness function, which is the second part of a GA, captures a lot of the importance of a particular chromosome inside a given application[McC05, p. 207].

b) Fitness Criteria in Genetic Algorithms

Fitness criteria assume a pivotal role in the context of genetic algorithms (GAs) at each stage. These criteria serve the purpose of assessing the population, whether it is to ascertain which individuals should advance to the subsequent generation or to establish when

the algorithm should be terminated. The fitness function is tailored to the particular problem and encompasses diverse constraints aligned with the application's prerequisites, all with the aim of yielding optimal outcomes [Kat21, p. 87]. The fitness function forms the cornerstone of a GA and is usually structured in a format akin to: GA(fitness, fitness threshold, population size, fraction to be replaced, mutation rate). In this equation, 'fitness' represents the function that assesses the quality of a solution, 'fitness threshold' establishes the minimum acceptable solution, 'population size' (ps) denotes the number of individuals in the population, 'fraction to be replaced' (f) signifies the proportion of the population to be substituted, and 'mutation rate' (m) defines the frequency at which mutations take place [KSA15, p. 87].

The meticulous design of the fitness function holds paramount importance for GAs, as the algorithm's performance relies significantly on the efficacy of this function. GAs frequently necessitate a substantial number of iterations to achieve optimal outcomes for intricate problems. In situations where the accuracy of the fitness function is uncertain or affected by noise, or when execution time is a constraining factor, fitness values can be approximated [KSA15, p. 87]. Fitness functions can be realized in two primary manners: one in which they are mutable and capable of adaptation over time, and the other in which they remain unchanging throughout the algorithm's execution. The decision on how to implement the fitness function is frequently guided by the distinct characteristics of the problem domain [KSA15, p. 87].

Here, we introduce a common and straightforward example frequently utilized in the genetic algorithm community: the OneMax problem. This problem stands as a classic benchmark. The objective here is to maximize the number of '1s' in a binary string of length 'n'. The encoding for this problem is uncomplicated, as each chromosome is represented by a binary string of length 'n'. The fitness function for the OneMax problem is a simple count of the "1s" within a given chromosome. Consequently, there exist 2^n possible chromosomes in total, with each having a fitness score ranging from 0 to 'n'. In this scenario, a binary string consisting of 'n' '1s' is evidently the optimal chromosome for the OneMax problem of length 'n'. This straightforward problem formulation serves

as a valuable benchmark for evaluating various GA design and implementation strategies [McC05, p. 107].

c) Selection Methods

In genetic algorithms, the selection process is a pivotal step that determines which strings will partake in the reproduction. Selection, often referred to as the reproduction operator, exerts a significant influence on the convergence rate of GAs. Commonly employed selection techniques encompass the roulette wheel, rank-based, tournament, Boltzmann, and stochastic universal sampling methods [Kat21, p. 8096]. Other selection strategies include Truncation Selection, which eliminates the least fit individuals before random selection; Tournament Selection, which selects the fitter of two randomly chosen chromosomes; and Random Stochastic Selection, which chooses chromosomes based on their expected selection frequency. These techniques empower GAs to adapt to a multitude of problem domains and optimization objectives [McC05, p. 208].

d) Recombination (Crossover and Mutation)

Recombination is the procedure through which individuals in the succeeding generation come into being through the amalgamation of chromosomes chosen from a given population. It emulates the amalgamation of genes occurring during the process of procreation. The probability of selecting higher-fitness chromosomes ensures that the succeeding generation will consist of a greater number of highly fit chromosomes. Within recombination, crossover and mutation stand as two indispensable genetic operators. The results of these operators are stochastic and probabilistic in nature [McC05, p. 208].Once the coding strategy is established, we can proceed to the crossover process. Crossover utilizes gene segments from the parent chromosomes to generate a new offspring [AL19, p. 381].

Chromosome 1 : 11011 | 00100110110
Chromosome 2 : 11011 | 11000011110
Offspring 1 : 11011 | 11000011110
Offspring 2 : 11011 | 00100110110

Crossover of Chromosome

Numerous distinct crossover methods are at one's disposal, including the option to select multiple crossover points. The complexity of the crossover process is contingent upon the chromosome encoding. Fine-tuning the choice of crossover method for specific problems can enhance the effectiveness of genetic algorithms [AL19, p. 381].

The fundamental mechanism that sustains genetic diversity across successive populations is mutation. Three widely recognized mutational processes include Scramble Mutation (SM), Simple Inversion Mutation (SIM), and Displacement Mutation (DM). DM introduces randomness while maintaining validity by relocating a substring within a single solution.Exchange mutations and insertion mutations, which exchange or insert portions of a solution respectively [Kat21, p. 8100]. In binary encoding, mutation involves randomly switching selected bits from 0 to 1 or from 1 to 0 [AL19, p. 381].

Original Offspring 1	1101111000011110
Original Offspring 2	1100111000011110
Mutated offspring 1	1101100100110110
Mutated offspring 2	1101101100110110

Table 2.1: Mutation of Offspring

Mutation operators alter one or more allele values to modify the attributes of an individual chromosome. The standard mutation operator is applied to every position on bit-string chromosomes. A randomly generated number, uniformly distributed between 0 and 1, is compared to a predefined "mutation rate." If the random number surpasses the mutation rate, no mutation occurs at that location. However, when the mutation rate is greater than or equal to the random number, the allele value is inverted, changing 0 to 1 or vice versa. It's worth noting that mutation rates are typically maintained at very low levels [McC05, p. 209].

2.2.6 Model Evaluation: Assessing Algorithm Performance

Upon completing the data processing stage, the next critical step is to evaluate the model's performance. Model evaluation is essential for gauging the algorithm's efficacy and accuracy in a specific context. In this study, the Confusion Matrix, a widely-used evaluation technique, was employed to calculate accuracy and F-measure values [XDM16].
The Confusion Matrix is a valuable tool for understanding how well the model's predictions align with actual results. It provides insights into various performance metrics, including false positives, false negatives, true positives, and true negatives. Two key metrics assessed are:

a. Accuracy: This metric quantifies the percentage of correct predictions, offering an overall measure of the algorithm's performance. It evaluates the model's accuracy in predicting both positive and negative cases [XDM16, p. 250-261]

b. F-Measure: This essential metric assesses the algorithm's performance, particularly useful when working with unbalanced datasets. The F-measure combines precision and recall, balancing accurate predictions and positive predictive value [HC18, p. 539-547].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

$$F - Measure = \frac{2PrecisionRecall}{Precision + Recall} \tag{2.6}$$

Equations 2.5 and 2.6 are utilized to calculate accuracy and the F-measure, respectively. These formulas provide a numerical means of assessing the model's effectiveness. The F-measure, by amalgamating precision and recall, furnishes a balanced evaluation of the

model's ability to correctly classify instances, especially in scenarios with potential class imbalance. Meanwhile, a high accuracy value signifies a substantial proportion of accurate predictions. In summary, the F-measure, accuracy, and the Confusion Matrix constitute indispensable tools for algorithm assessment. They offer a comprehensive understanding of a model's performance by considering both accurate and erroneous predictions. These metrics play a pivotal role in determining the suitability of an algorithm for a given task or problem.

2.2.7 Cross-Validation: A Rigorous Evaluation Technique

In the realm of machine learning, cross-validation is a computationally intensive method frequently employed to thoroughly assess an algorithm's or model's performance and ascertain its predictive accuracy [MSW20, XDM16, HC18] This technique involves utilizing all available data instances as both training and testing datasets to ensure a comprehensive evaluation.

Specifically, this investigation utilized K-Fold Cross Validation, a systematic technique for dividing the dataset into K subsets or "folds." K, a user-defined parameter, is chosen, and during each iteration, one fold is reserved as the test set while the remaining (K-1) folds are employed for training [Fus11, p. 137-146].

Key Aspects of K-Fold Cross Validation in this Study:

1. Iterative Training and Testing: K-Fold Cross Validation entails a sequence of iterations, each representing a distinct training and testing cycle. For each iteration, one of the K folds serves as the test set, while the others are used for model training.

2. Averaging Results: Typically, the outcomes from all iterations of cross-validation are averaged to derive an overall performance metric. This approach minimizes the impact of data partitioning and results in a more reliable estimation of algorithm performance.

3. Robustness and Generalization Assessment: K-Fold Cross Validation offers a robust method for assessing the model's generalization capability. It ensures that the al-

gorithm does not overfit the training data and can perform effectively on unseen instances by evaluating it across multiple data subsets.

4. Prediction Error Estimation: The method delivers a dependable estimate of the prediction error, often measured using metrics like accuracy or mean squared error. This aids in gauging the expected performance of the algorithm on new, untested data.

In conclusion, K-Fold cross-validation is a valuable technique that provides a more comprehensive and accurate appraisal of algorithm performance compared to a single train-test split. Utilizing all available data for both training and testing and averaging results over multiple iterations, assists researchers and practitioners in making informed decisions regarding the algorithm's suitability for their specific problem or task.

2.3 Experiment and result

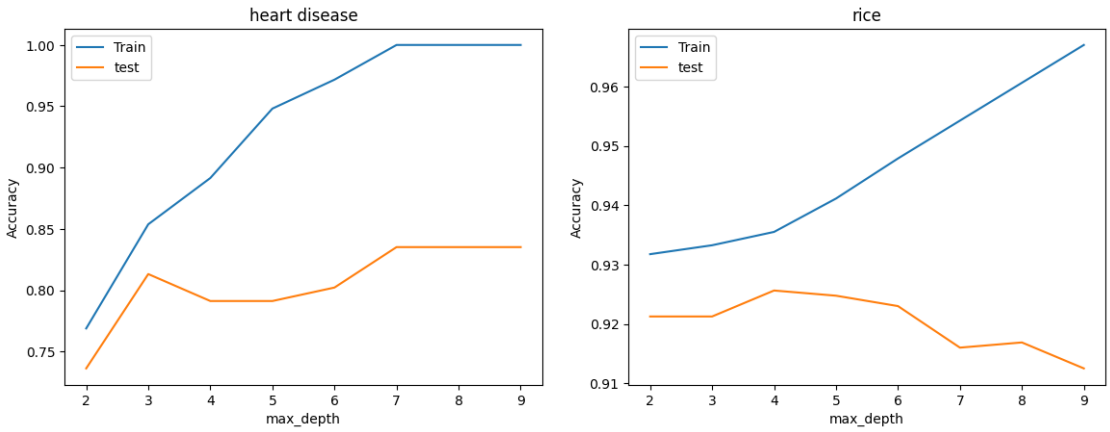
2.3.1 Effects of Hyperparameters of Decision Tree

A hyperparameter is a key variable in any machine learning algorithm as it controls the flow of the ML model. Hyperparameters must be set before training any ML model. Decision Tree has hyperparamers like max_depth, min_samples_split, min_sample_leaf, max_features, min_weight_fraction_leaf, etc [edu]. In this study we consider max_depth, min_samples_split, min_sample_leaf, and critetaion.

a) max_depth :

The maximum depth of the tree is a variable that controls how many times nodes can be split further. If the value is too small then it cannot adapt the pattern in data and if the value is too big then it will simply overfit the data.

Here is the effect of max_depth on DecisionTreeClassifier in sklearn [PVG+11] using experimental data.

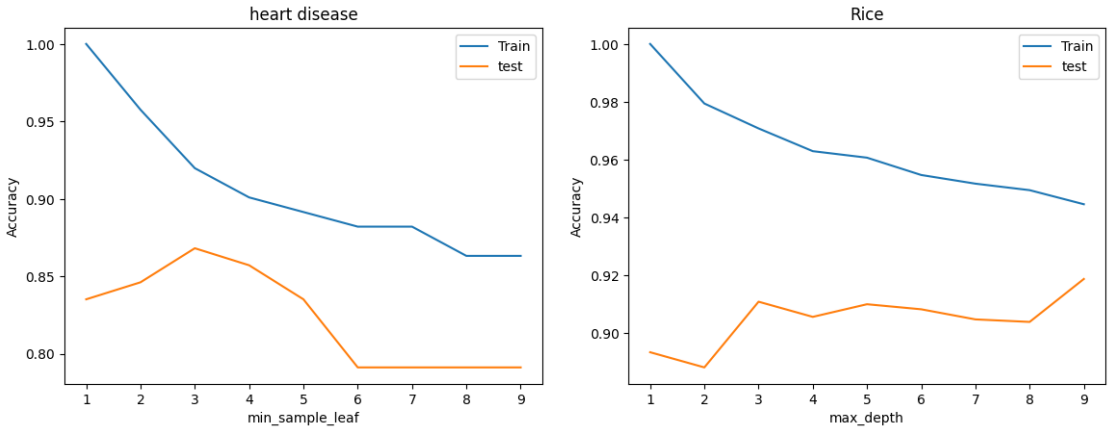


As you can see in the figure training accuracy increase for both dataset as max_depth increase and decrease for Rice. Finding the best max_depth as it does not overfit or underfit data is very important for any dataset.

b) min_samples_leaf :

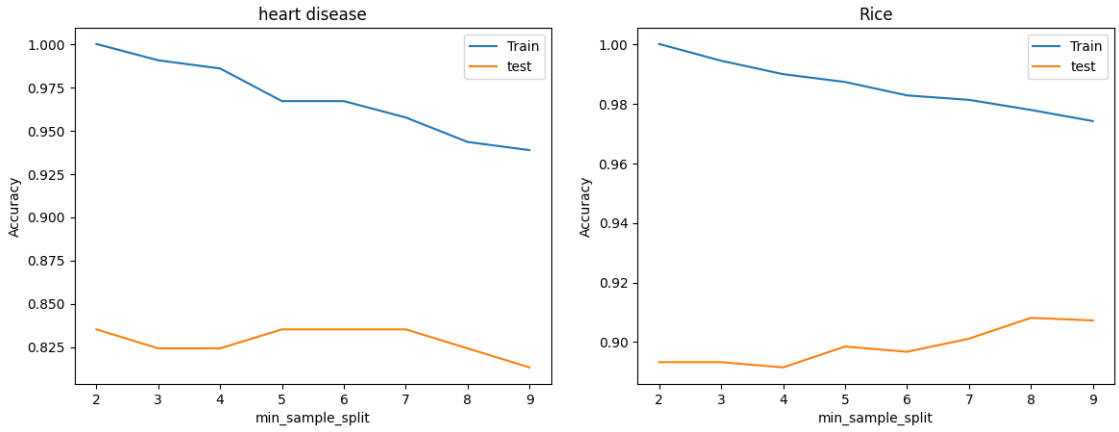
The minimum number of samples required to assign a node as a leaf node. Split at every

node will only be considered if the child has more than min_samples_leaf samples in it. In sklearn DecisionTreeClassifer, min_sample_leaf excepts float and integer as input. In this experiment integer is considered while it is easy to implement. Below are figures that describe how accuracy varies with min_samples_leaf varies.



c) min_samples_split :

Specify the minimum number of samples required in a node for it to be further split during the tree-building process. Before making a split, the algorithm checks if the number of samples in the current node is greater than or equal to the min_samples_split value. If the number of samples in the node is less than min_samples_split, the algorithm will not perform a split, and the current node will become a leaf node. If the number of samples in the node is greater than or equal to min_samples_split, and if the proposed split would result in child nodes with a number of samples greater than or equal to min_samples_split, the split is allowed to happen.



The above-presented figures show how the accuracy of DT varies with respect to the minimum number of samples for split for the used Datasets.

d) Criterion:

Criterion is the Parameter in DT to determine the measurement at the split when constructing the DT. Choice of the criterion is very important for the DT as it defines splitting rules for DT while constructing DT using Training Data. Gini Index and Entropy are the most used criteria in DT.

Gini index:- It is an impurity measure of datasets. It is an alternative of Information Gain [PG16, p. 22].

$$Gini(S) = \sum p_i^2 \tag{2.7}$$

Entropy:-

Entropy is the measure of the dataset that describes the diversity of class in the dataset. If the dataset contains samples from the same samples then entropy will be 0 and if the dataset contains samples from more them one class then it varies between 0 and 1 [APK22, p. 2436].

$$E(X) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.8}$$

Above is a formula for entropy Where pi is the number of samples belonging to class I in sub-node X and c is the Number of class [APK22, p. 2436].

	gini	entropy
Training Accuracy	1.0	1.0
Test Accuracy	83.51%	82.41%

Table 2.2: Heart Disease

	gini	entropy
Training Accuracy	1.0	1.0
Test Accuracy	89.32%	90.20%

Table 2.3: Rice

As described in the above table for the dataset heart disease gini is more effective as compared to entropy as it gives more accuracy. While for the Rice dataset entropy is more effective.

2.3.2 Decision Tree with default settings

In scikit-learn version 0.22 (as of my knowledge cutoff date in January 2022), when you create a DecisionTreeClassifier, the default hyperparameters are indeed as follows:
criterion: "gini"
max_depth: None (unlimited depth)
min_samples_split: 2
min_samples_leaf: 1
It means that when you create a DecisionTreeClassifier without explicitly specifying these hyperparameters, it uses these default values. The first decision tree was constructed using these default parameters for both the "Heart Disease" dataset and the "Rice" dataset. Figure 2.4 is the decision tree created with default parameters for the heart dataset and for rice, it creates DT with more than 50 nodes, and it's hard to place it here.

For the Heart dataset, the accuracy score is 1 for the training dataset and 0.84 for the testing dataset. For the Rice dataset, the accuracy score is 1 for the training dataset and 0.89 for the testing dataset.

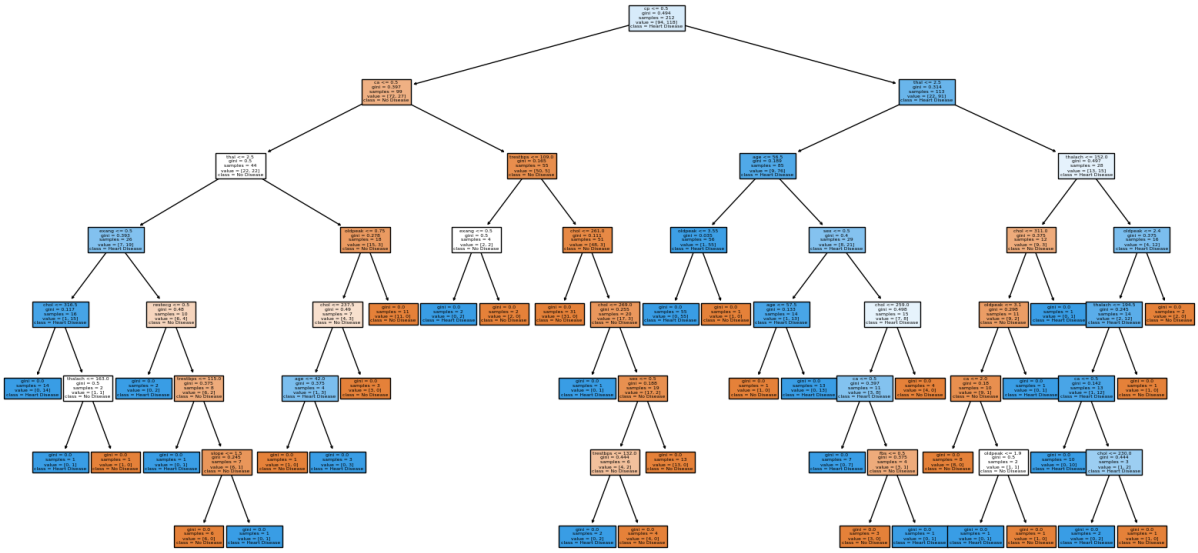


Figure 2.4: DT with default parameters with heart disease dataset

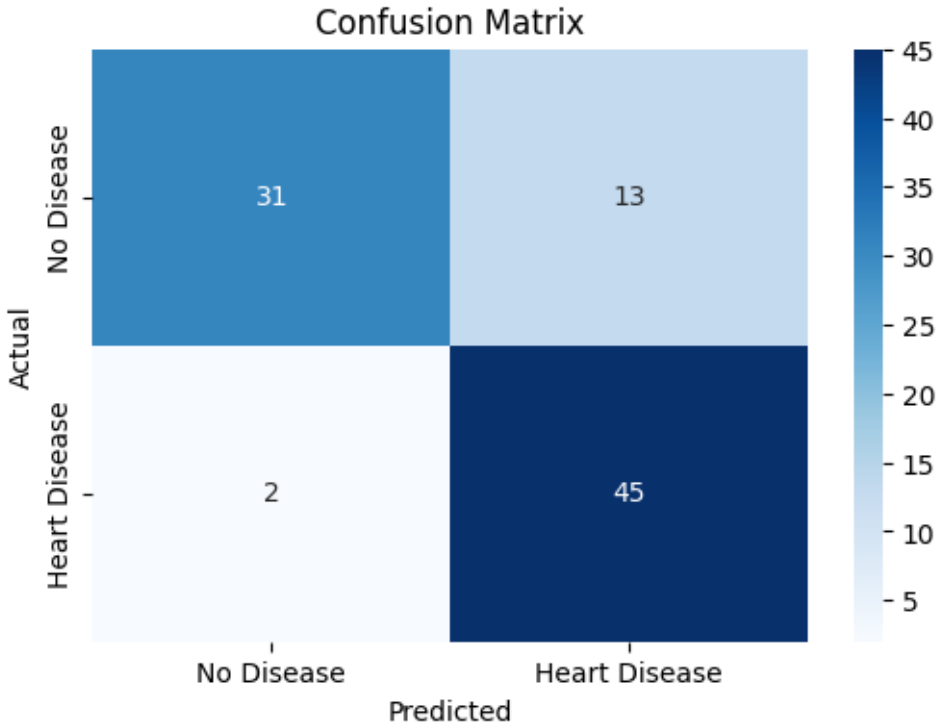


Figure 2.5: Heart disease

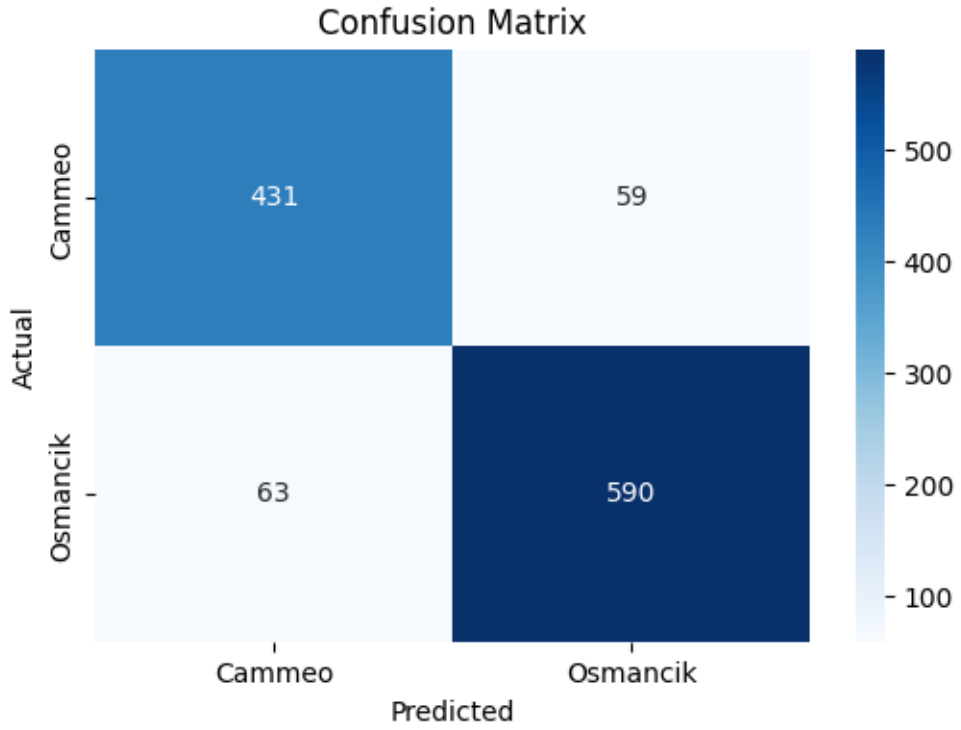


Figure 2.6: Rice

2.3.3 Parameter tuning

Random Search

Random Search is applied to find the best parameter for DT. Random Search selects parameters randomly which is why 10 iteration is conducted for both data set and then the result with the best accuracy is selected for creating the resulting DT Classifier. For conducting this experiment, "RandomizedSearchCV" is employed. It is an implementation of Random Search in Python and is available in the scikit-learn library (3). RandomizedSearchCV requires a Classifier, cross-validation (CV), a scoring function, and a parameter distribution as arguments. It then fits the training data and, based on the scoring function, provides a set of parameters from the parameter distribution. For all 10 iterations, it uses DecisionTreeClassifier as Classifier, it used 5 as cross-validation (CV), and Accuracy as the scoring function. Table 2.4 parameters for parameter distribution.

parameter	values
max_depth	1,2,3,4,5,6,7,8,9
criterion	'gini', 'entropy'
min_samples_split	2,3,4,5,6,7,8
min_samples_leaf	1,2,3,4,5,6,7,8,9,10

Table 2.4: Parameter distribution

max_depth	criterion	min_samples_split	min_samples_leaf	Accuracy
9	gini	5	5	0.835165
9	gini	8	5	0.835165
7	gini	7	9	0.791209
5	gini	2	5	0.835165
6	gini	5	5	0.835165
7	gini	2	6	0.791209
8	gini	2	4	0.857143
8	gini	5	4	0.857143
7	gini	5	3	0.868132
9	gini	5	3	0.868132

Table 2.5: 10 Iterations of Random Search on Heart Disease dataset

max_depth	criterion	min_samples_split	min_samples_leaf	Accuracy
2	gini	3	1	0.921260
2	gini	8	9	0.921260
1	gini	6	1	0.921260
2	gini	5	4	0.921260
4	gini	4	8	0.925634
4	gini	2	8	0.925634
1	entropy	5	10	0.921260
4	gini	7	9	0.925634
2	gini	3	3	0.921260
1	entropy	4	10	0.921260

Table 2.6: 10 Iterations of Random Search on Rice dataset

parameter	values
max_depth	7
criterion	'gini'
min_samples_split	5
min_samples_leaf	3

Table 2.7: best parameters for heart disease selected by random search

parameter	values
max_depth	4
criterion	'gini'
min_samples_split	2
min_samples_leaf	8

Table 2.8: best parameters for Rice selected by random search

As shown in Table 2.5 For the Heart disease dataset, random search produces different combinations of parameters, and accuracy varies with each parameter combination. The best result, with an accuracy of 0.87, is chosen, and the selected parameters are listed in Table 2.7. For the Rice dataset, Table 2.6 displays the results from 10 iterations of random search, and the parameters yielding an accuracy of 0.92 are selected. These parameters are presented in Table 2.8.

Grid Search

As a second method, grid search is also employed to find the best parameters for the Decision Tree (DT). Unlike RandomizedSearchCV, it tests all possible combinations of the given parameters, which can be computationally complicated. In the case of Grid Search, "GridSearchCV" is employed, which is an implementation of grid search in Python available in scikit-learn (sklearn). It takes a Classifier, cross-validation (cv), param_grid, and a scoring function as parameters. Subsequently, it fits the training data and, based on the scoring function, provides a set of parameters from the parameter grid. The same parameters are used for GridSearchCV as those used in RandomizedSearchCV.
Table 2.9 and Table 2.10 provides the value of the best parameters selected by this technique for both datasets. This suggests that the GridSearch found the same optimal set of hyperparameters after performing multiple iterations for both the "Heart Disease" and "Rice" datasets.

parameter	values
max_depth	6
criterion	'gini'
min_samples_split	2
min_samples_leaf	5

Table 2.9: best parameters for Heart Disease selected by Grid search

parameter	values
max_depth	4
criterion	'gini'
min_samples_split	2
min_samples_leaf	8

Table 2.10: best parameters for Rice selected by Grid search

Genetic Algorithm

The third method employs a genetic algorithm (GA). To implement GA in Python, the PyGAD library is utilized. the parameters for PyGAD.GA : num_generations=1000, num_parents_mating=3, num_genes=6, gene_type=int, sol_per_pop=15, init_range_low=1, init_range_high=20, mutation_probability=0.1, crossover_probability=0.8.

The fitness function for the GA is determined by selecting accuracy as the measurement. In each iteration, a decision tree is constructed based on the solution gene, and the accuracy of this decision tree is computed to serve as the fitness value. This accuracy-based fitness function helps evaluate and guide the optimization process for the parameter tuning.

Table 2.9 presents the parameters selected for heart disease and Table 2.12 for Rice at the end of the genetic algorithm optimization process, which achieved an accuracy of 0.90 for one dataset and 0.80 for the other dataset. This indicates that the genetic algorithm was able to identify a set of hyperparameters that led to improved accuracy for the given datasets. The specific parameter values in Table 1 represent the outcome of the genetic algorithm's optimization for these two datasets.

parameter	values
max_depth	6
criterion	'gini'
'min_samples_split	2
'min_samples_leaf	3

Table 2.11: best parameters for Heart Disease selected by GA

parameter	values
max_depth	4
criterion	'gini'
'min_samples_split	2
'min_samples_leaf	7

Table 2.12: best parameters for Rice selected by GA

2.3.4 Parameter tuning of Random Forest

In this section, a supplementary experiment is conducted on the random forest algorithm. An additional parameter, "n_estimators," is introduced to specify the number of trees that will be generated. This experiment focuses exclusively on the Heart Disease dataset.

Table 2.13 displays the parameters employed in the tuning process for the random forest algorithm. The experiment uses only limited parameters for simplicity.

parameter	values
n_estimators	2,3,4,5,6,7,8,9,10,11,12,13
max_depth	1,2,3,4,5,6,7,8,9
criterion	'gini', 'entropy'
min_samples_split	2,3,4,5,6,7,8
min_samples_leaf	1,2,3,4,5,6,7,8,9,10

Table 2.13: Parameter distribution of Random Forest

Initially, a random forest is constructed using default parameters, resulting in an accuracy of 0.86. The figure illustrates the confusion matrix of the random forest.

Subsequently, a Random Search is employed to identify the optimal parameter set for the random forest. The Random Search process is iterated 10 times to find the best

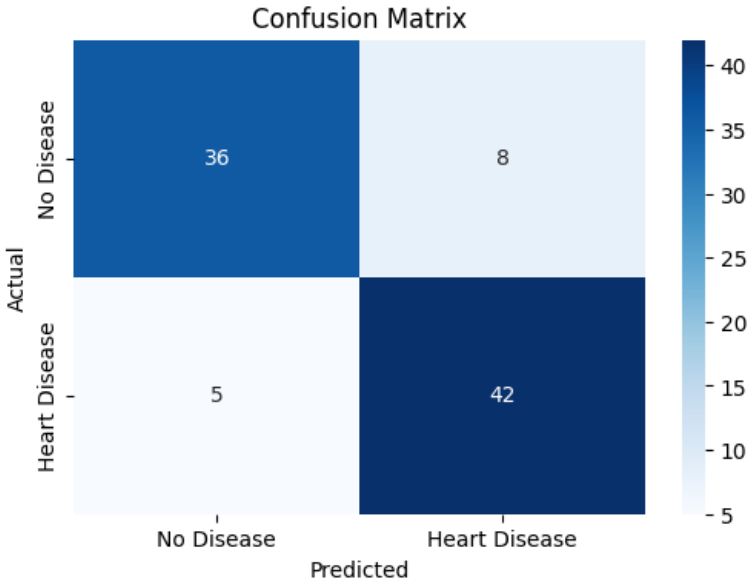


Figure 2.7: Confusion Matrix of Random Forest With Default Parameter

parameter configuration out of the 10 iterations. Table 2.14 presents the results of these 10 iterations.

Random Search achieves an accuracy of 0.87 with parameters 2.15.

n_estimators	max_depth	criterion	min_samples_split	min_samples_leaf	Accuracy
12	4	gini	5	1	0.780220
13	4	entropy	2	1	0.813187
10	4	gini	2	4	0.824176
11	5	gini	4	7	0.824176
13	8	entropy	7	3	0.868132
7	5	gini	8	6	0.802198
13	6	gini	5	4	0.824176
11	8	entropy	7	2	0.868132
12	9	gini	6	7	0.857143
12	8	gini	6	7	0.857143

Table 2.14: 10 Iterations of Random Search on Heart Disease dataset

The highest accuracy achieved through grid search is 0.88. It's worth noting that the same set of parameters used in the grid search for random forest is also applied in conjunction with a Decision Tree model. Finally, a genetic algorithm is employed to determine the optimal parameter configuration for the random forest model, resulting

parameter	values
n_estimators	11
max_depth	8
criterion	entropy
'min_samples_split	7
'min_samples_leaf	2

Table 2.15: best parameters for Heart Disease selected by Random Seach

parameter	values
n_estimators	8
max_depth	6
criterion	gini
'min_samples_split	2
'min_samples_leaf	10

Table 2.16: best parameters for Heart Disease selected by GA

in an accuracy matching the 0.88 accuracy obtained through the grid search using the parameters specified in Table 2.16.

2.3.5 Result

For the heart disease dataset, the default Decision Tree (DT) model achieves an accuracy of 84%. However, after conducting a Random Search and using parameters as specified in Table 2.7, the DT accuracy increases to 87%. In contrast, Grid Search with parameter 2.9 results in an accuracy of 84%, which remains the same as the default setting. Similarly, when employing Genetic Algorithm (GA), the DT model also achieves an accuracy of 87%. In this case, Random Search and GA give parameters with better accuracy than Grid Search. For the rice dataset, when using the default Decision Tree (DT) model, it achieves an accuracy of 89%. However, when parameters found through Random Search are employed, the accuracy significantly improves to 93%. Similarly, when parameters obtained through Grid Search and Genetic Algorithm (GA) are utilized, the DT model also attains an accuracy of 93%. In this case, Random Search, Grid Search, and Genetic Algorithm all lead to the same improved accuracy of 93% compared to the default accuracy of 89% for the rice dataset. In both cases, the parameter tuning process enhances the model's accuracy by approximately 3%. It's important to note that while Random Search

delivered the best result in this specific instance, it doesn't guarantee the best result every time since it selects parameters randomly. On the other hand, the Genetic Algorithm, with sufficient iterations, consistently aims for the best result as it uses a fitness function to guide the selection of solutions in each iteration. This makes the Genetic Algorithm a more reliable approach for optimizing the model's performance over time.

With the default parameters, the Decision Tree (DT) produces a very large tree, as illustrated in Figure 1. However, after applying parameter tuning, the DT model results in a comparatively smaller model with fewer rules. This indicates that the parameter tuning process has effectively pruned the tree and simplified the model, making it more concise and potentially improving its generalization capabilities. Figures 2.11, 2.12, 2.13, 2.8, 2.9, 2.10 show decision tree with parameter puning.

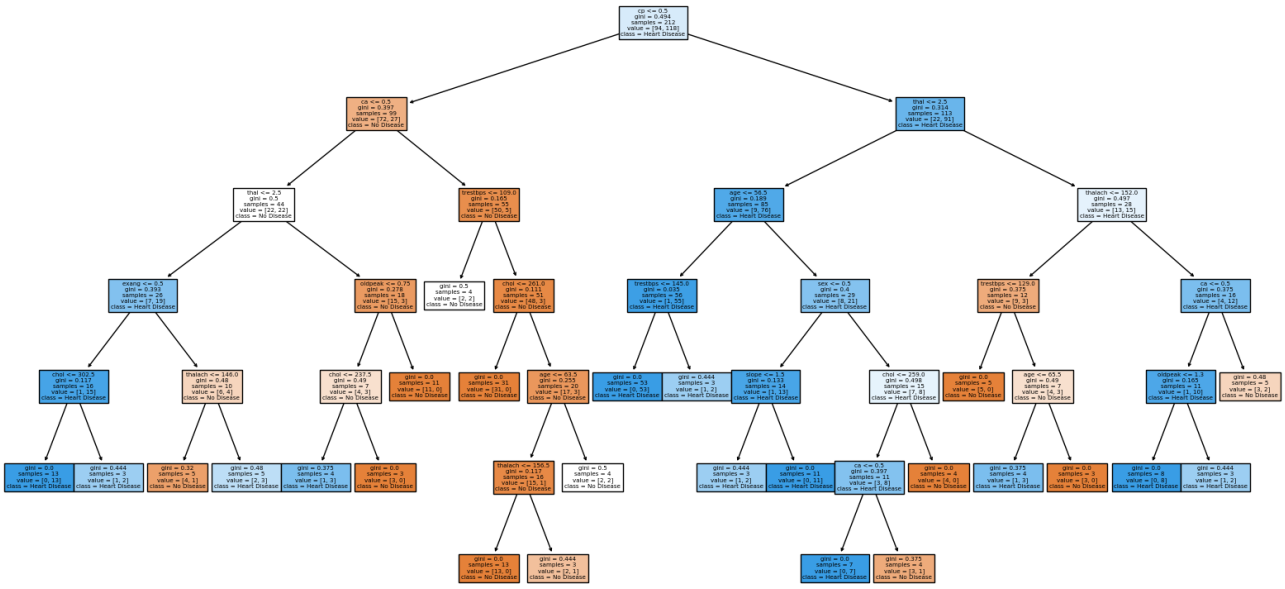


Figure 2.8: DT of heart disease with random search

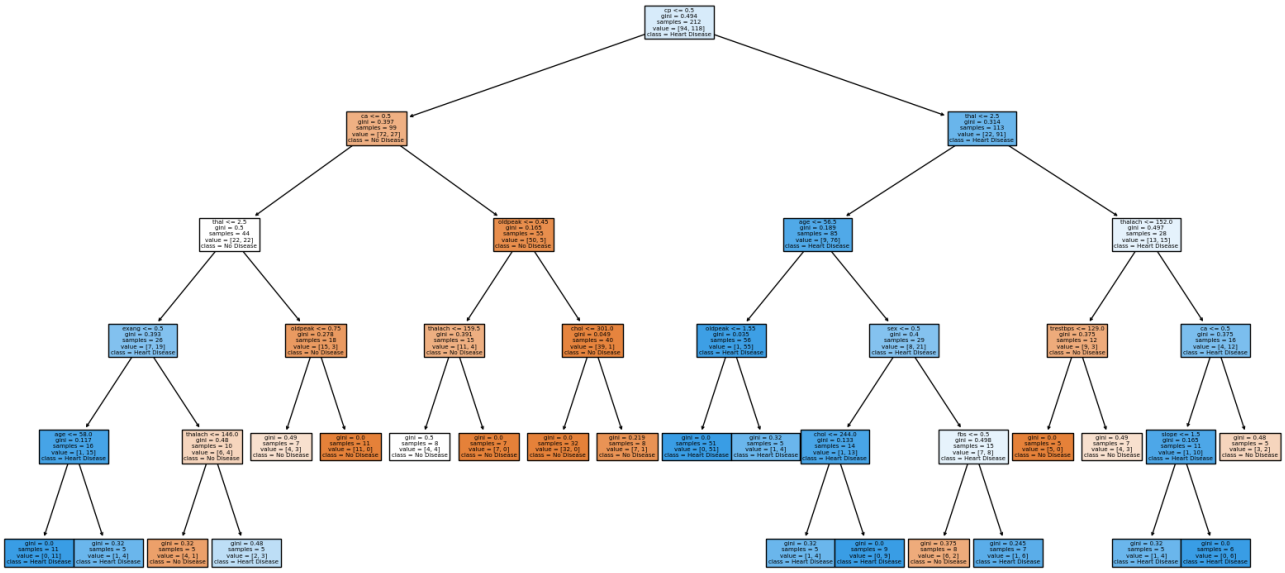


Figure 2.9: DT of heart disease with grid search

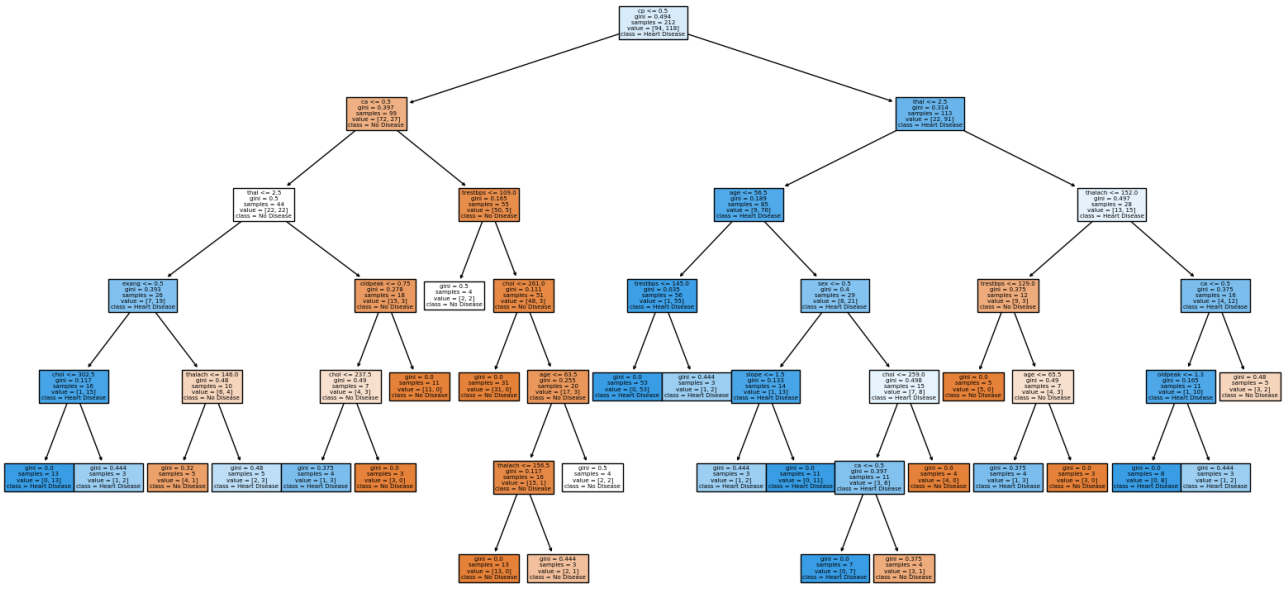


Figure 2.10: DT of heart disease with GA

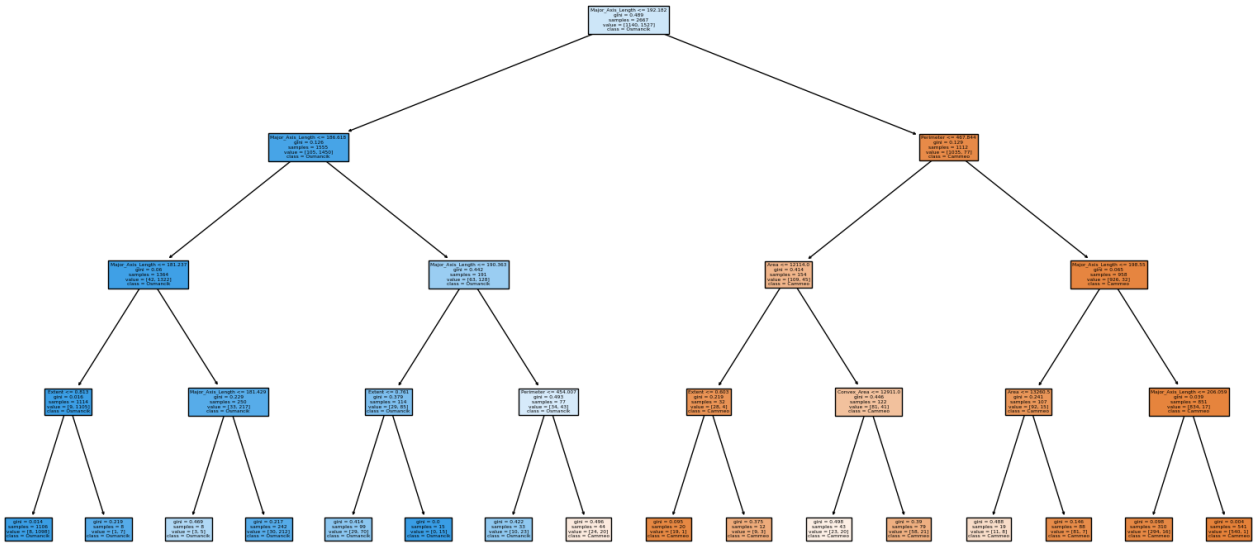


Figure 2.11: DT of Rice with random search

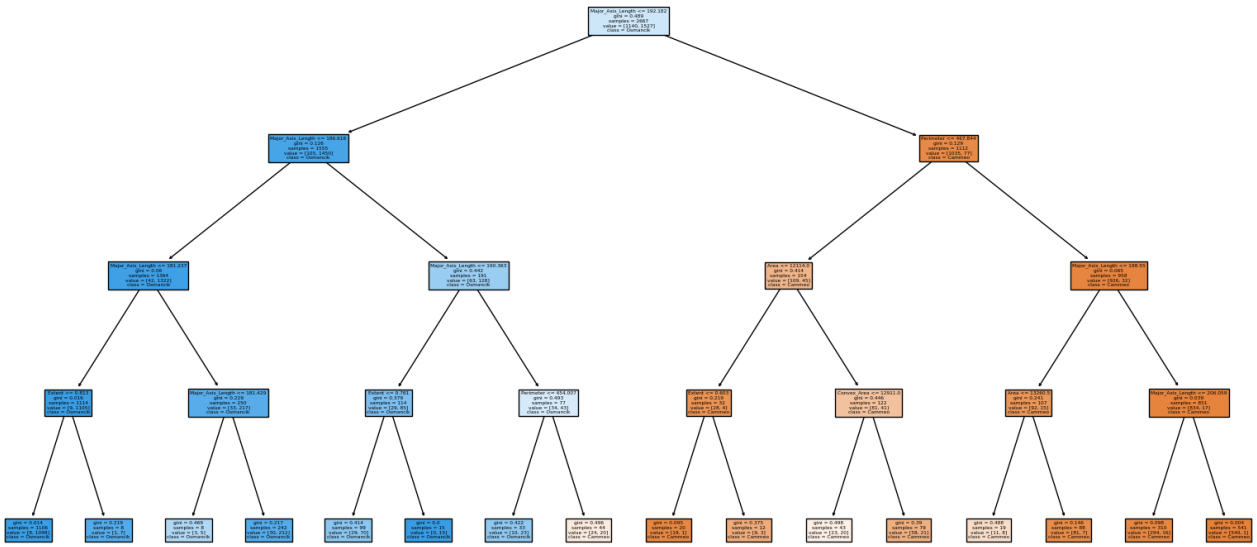


Figure 2.12: DT of Rice with grid search

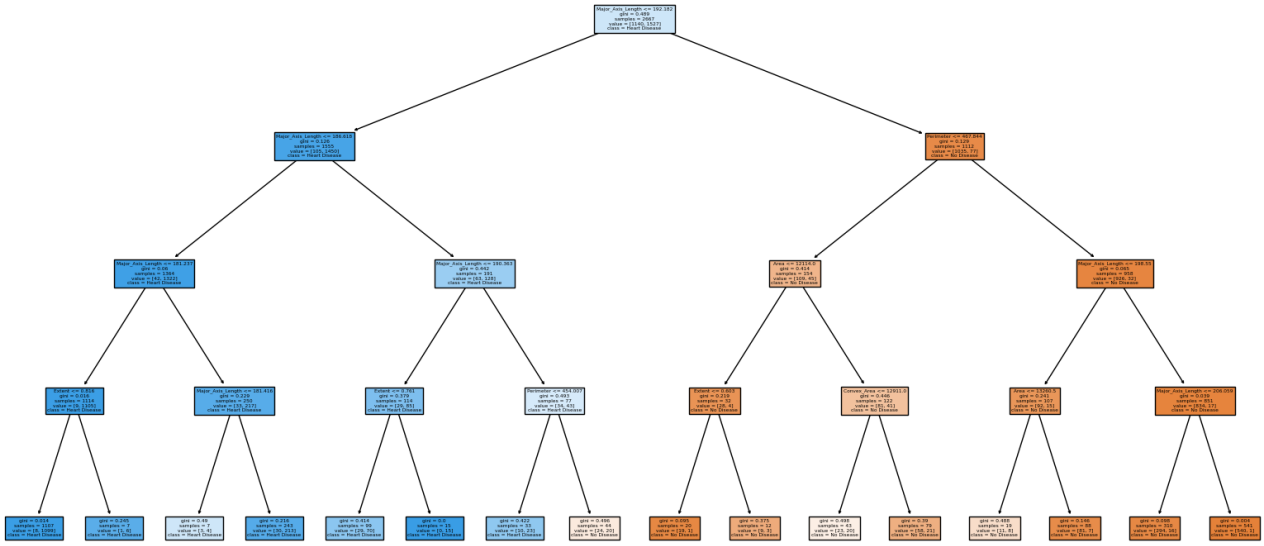


Figure 2.13: DT of Rice with grid search

Chapter 3

Conclusion

This study has demonstrated a straightforward approach to parameter tuning using random search, grid search, and a genetic algorithm. In the second chapter, detailed methodologies for these tuning methods are explained. In the third chapter, by conducting experiments on two datasets, the study was able to improve accuracy and reduce the complexity of decision trees (DT). Machine learning models are valuable for prediction, but without proper parameter tuning, they may yield less accurate results and more complex model structures.

Indeed, all three techniquesRandom Search, Grid Search, and Genetic Algorithmare valuable for optimizing Random Forest models. They each have their strengths and trade-offs in terms of computational cost and performance:

Random Search: This method is the fastest in terms of computation cost, as it explores parameter combinations randomly. While it may not guarantee the absolute best result, it often provides a good compromise between performance and computational efficiency.

Grid Search: Grid Search is typically the slowest among the three because it systematically tests all combinations of parameters. It is, however, effective at finding the best result by covering the entire parameter space, making it a comprehensive but resource-intensive approach.

Genetic Algorithm: Genetic Algorithms offer an attractive balance between performance and computation cost. They can yield the best results with a more efficient use of computational resources compared to Grid Search, making them a powerful option for pa-

rameter optimization. For future research, it is suggested that experiments be conducted with a broader range of datasets, and parameter range with different machine learning models. This can help in assessing the generalizability and effectiveness of hyperparameter tuning techniques across different domains.
Additionally, it's important to note that hyperparameter tuning is data-dependent. If the dataset changes, the hyperparameter tuning process should be repeated to find the optimal parameters for the new training data. This emphasizes the importance of adaptability and ongoing optimization when working with machine learning models.

3.1 Attachment

Here you can describe the data sets in greater detail or insert them completely if there is not too much. The software used can also be described in more detail here if this is deemed necessary.

3.1.1 Datasets:

For this experiment, two datasets from the UCI Machine Learning Repository have been selected. The UCI Machine Learning Repository [MK] is an open-source dataset repository that can be accessed online. It offers datasets for a wide range of problems, including classification, regression, clustering, time series analysis, and more. You can find datasets for various machine learning and data analysis tasks on the official UCI Machine Learning Repository website, which provides a valuable resource for researchers and data scientists.

A) Heart Disease : The "Heart Disease" [JD88] dataset is used for classification problems and consists of 303 instances. It contains a combination of categorical, integer, and real value type features. This dataset is commonly used for tasks related to heart disease diagnosis or prediction.

Figure 3.1 shows the first 10 values of the dataset for the sample, which contains 13 attributes and a target variable (1 or 0) to determine whether a person has the disease or is disease-free. If the target variable is 1, it indicates that the person has the disease. If

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

Figure 3.1: Heart Disease Dataset

the target variable is 0, it signifies that the person is disease-free.

Rice (Cammeo and Osmancik) : The "Rice (Cammeo and Osmancik)" [mis19] dataset is used for the second dataset and is sourced from the UCI library. For this dataset, a total of 3,810 images of rice grains were captured by creators, with a focus on two different species: Cammeo and Osmancik. From these images, seven features were extracted for each of the two rice species. This dataset is designed for tasks related to classifying rice grains into these two distinct species based on their extracted features.

Figure 2 displays 10 values from the dataset, where the first 7 columns represent the

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	class
0	15231	525.578979	229.749878	85.093788	0.928882	15617	0.572896	Cammeo
1	14656	494.311005	206.020065	91.730972	0.895405	15072	0.615436	Cammeo
2	14634	501.122009	214.106781	87.768288	0.912118	14954	0.693259	Cammeo
3	13176	458.342987	193.337387	87.448395	0.891861	13368	0.640669	Cammeo
4	14688	507.166992	211.743378	89.312454	0.906691	15262	0.646024	Cammeo
5	13479	477.015991	200.053055	86.650291	0.901328	13786	0.657897	Cammeo
6	15757	509.281006	207.296677	98.336136	0.880323	16150	0.589708	Cammeo
7	16405	526.570007	221.612518	95.436707	0.902521	16837	0.658888	Cammeo
8	14534	483.640991	196.650818	95.050682	0.875429	14932	0.649651	Cammeo
9	13485	471.570007	198.272644	87.727287	0.896789	13734	0.572320	Cammeo

Figure 3.2: Heart Disease Dataset

attributes for this dataset, and the "class" column is the classification target. The "class"

column contains labels that indicate whether the rice grains belong to the "Cammeo" or "Osmancik" species.

List of Figures

List of Tables

Bibliography

[AL19] Annu Lambora, Kriti C. Kunal Gupta G. Kunal Gupta: Genetic Algorithm-A Literature Review. (2019). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8862255 30, 33, 37, 38

[AMS+21] Ahsan, Md M. ; Mahmud, M. A. P. ; Saha, Pritom K. ; Gupta, Kishor D. ; Siddique, Zahed: Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance. In: *Technologies* 9 (2021), Nr. 3. http://dx.doi.org/10.3390/technologies9030052. – DOI 10.3390/technologies9030052. – ISSN 2227–7080 10

[AO04] Ataei, M. ; Osanloo, M.: Using a combinationof genetic algorithm and the grid search methodto determine optimum cutoff grades of multiplemetal deposits. In: *International Journal of Surface Mining, Reclamation and Environment* 18 (2004) 30

[APK22] Aning, Samuel ; PrzybyÅa-Kasperek, MaÅgorzata: Comparative Study of Twoing and Entropy Criterion for Decision Tree Classification of Dispersed Data. In: *Procedia Computer Science* 207 (2022), 2434-2443. http://dx.doi.org/https://doi.org/10.1016/j.procs.2022.09.301. – DOI https://doi.org/10.1016/j.procs.2022.09.301. – ISSN 1877–0509 44

[BDS10] Bengio, Yoshua ; Delalleau, Olivier ; Simard, Clarence: Decision trees do not generalize to new variations. In: *Computational Intelligence* 26 (2010), Nr. 4, S. 449â467. http://dx.doi.org/10.1111/j.1467-8640.2010.00366.x. – DOI 10.1111/j.1467–8640.2010.00366.x 15, 63

[BPH06] Bennett, Kristin P. ; Parrado-Hernández, Emilio: The interplay of optimization and machine learning research. In: *The Journal of Machine Learning Research* 7 (2006), S. 1265–1281 8

[Bre96] Breiman, Leo: Bagging Predictors. In: *Machine Learning* 24 (1996), 123-140. http://dx.doi.org/10.1007/BF00058655. – DOI 10.1007/BF00058655 24

[Bre01] Breiman, Leo: Random Forests. In: *Machine Learning* 45 (2001), S. 5–32. http://dx.doi.org/10.1023/A:1010933404324. – DOI 10.1023/A:1010933404324 24, 25, 26

[CA21] Charbuty, Bahzad ; Abdulazeez, Adnan: Classification Based on Decision Tree Algorithm for Machine Learning. In: *Journal of Applied Science and Technology Trends* 2 (2021), Mar., Nr. 01, 20 - 28. http://dx.doi.org/10.38094/jastt20165. – DOI 10.38094/jastt20165 8

[CXW+17] Chen, W. ; Xie, X.S. ; Wang, J.L. ; Pradhan, B. ; Hong, H.Y. ; Bui, D.T. ; Duan, Z. ; Ma, J.Q.: A Comparative Study of Logistic Model tree, Random Forest, and Classification and Regression Tree Models for Spatial Prediction of Landslide Susceptibility. In: *Catena* 151 (2017), 147-160. http://dx.doi.org/10.1016/j.catena.2016.11.032. – DOI 10.1016/j.catena.2016.11.032 24

[EB17] Edwin, R. ; Bogdan, Z.: Comparison of Support Vector Machine, Random Forest and Neural Network Classifiers for Tree Species Classification on Airborne Hyperspectral APEX Images. In: *European Journal of Remote Sensing* 50 (2017), S. 144–154 26

[edu] decision Tree Hyperparamers. https://www.educba.com/decision-tree-hyperparameters/ 42

[Fus11] Fushiki, T.: Estimation of prediction error by using K -fold cross-validation. In: *Statistics and Computing* 21 (2011) 40

[GRJ+17] Gupta, Bhumika ; Rawat, Aditya ; Jain, Akshay ; Arora, Arpit ; Dhami, Naresh: Analysis of various decision tree algorithms for classification in Data Mining. In: *International Journal of Computer Applications* 163 (2017), Nr. 8, S. 15â19. http://dx.doi.org/10.5120/ijca2017913660. – DOI 10.5120/ijca2017913660 18

[HC18] Hand, D. ; Christen, P.: A note on using the F-measure for evaluating record linkage algorithms. In: *Statistics and Computing* 28 (2018) 39, 40

[Hun07] Hunter, J. D.: Matplotlib: A 2D graphics environment. In: *Computing in Science & Engineering* 9 (2007), Nr. 3, S. 90–95. http://dx.doi.org/10.1109/MCSE.2007.55. – DOI 10.1109/MCSE.2007.55 13

[ISW16] I. Syarif, A. P. B. ; Wills, G.: SVMparameter optimization using grid search andgenetic algorithm to improve classification performance. In: *Telkomnika* 14 (2016) 30

[JD88] Janosi, Steinbrunn William Pfisterer M. Andras ; Detrano, Robert: *Heart Disease*. UCI Machine Learning Repository, 1988. – DOI: https://doi.org/10.24432/C52P4X 13, 60

[JS17] Jaiswal, Jitendra K. ; Samikannu, Rita: Application of Random Forest Algorithm on Feature Subset Selection and Classification and Regression. In: *World Congress on Computing and Communication Technologies (WCCCT)*, 2017 27

[JWH12] J. W. Han, D. A. Randell G. T. P. Breckon B. T. P. Breckon: The application of support vector machine classification to detect cell nuclei for automated microscopy. In: *Machine Vision and Applications* 23 (2012) 29

[Kat21] Katoch, Chauhan Sumit Singh Kumar V. Sourabh: A review on genetic algorithm: past, present, and future. In: *Multimedia Tools and Applications* 80 (2021), 8091-8126. https://doi.org/10.1007/s11042-020-10139-6 34, 36, 37, 38

[KIES16] Khedr, Ayman E. ; Idrees, Amira M. ; El Seddawy, Ahmed I.: Enhancing Iterative Dichotomiser 3 algorithm for classification decision tree. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6 (2016), Nr. 2, S. 70–79 16

[KMT17] 21) K. Mittal, D. K. ; Tewari, P. C.: An insight into âDecision Tree Analysis. In: *World Wide Journal of Multidisciplinary Research and Development* 41 (2017), Nr. 12, S. 111â115 22

[KSA15] Kour, Haneet ; Sharma, Parul ; Abrol, Pawanesh: Analysis of Fitness Function in Genetic Algorithms. 1 (2015), 09, S. 87–90 36

[LL19] Liashchynskyi, Petro ; Liashchynskyi, Pavlo: Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. In: *CoRR* abs/1912.06059 (2019). http://arxiv.org/abs/1912.06059 10

[Log22] Logunova, Inna: *Random Forest Classifier*. 2022 24

[McC05] McCall, John: Genetic algorithms for modelling and optimisation. In: *Journal of Computational and Applied Mathematics* 184 (2005), Nr. 1, 205-222. http://dx.doi.org/https://doi.org/10.1016/j.cam.2004.07.034. – DOI https://doi.org/10.1016/j.cam.2004.07.034. – ISSN 0377–0427. – Special Issue on Mathematics Applied to Immunology 31, 33, 35, 37, 38

[MFL+04] Myles, Anthony J. ; Feudale, Robert N. ; Liu, Yang ; Woody, Nathaniel A. ; Brown, Steven D.: An introduction to decision tree modeling. In: *Journal of Chemometrics* 18 (2004), Nr. 6, 275-285. http://dx.doi.org/https://doi.org/10.1002/cem.873. – DOI https://doi.org/10.1002/cem.873 14, 22

[MHC+16] Mantovani, Rafael G. ; Horváth, Tomáš ; Cerri, Ricardo ; Vanschoren, Joaquin ; De Carvalho, Andre C.: Hyper-parameter tuning of a decision tree induction algorithm. In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)* IEEE, 2016, S. 37–42 9

[mis19] *Rice (Cammeo and Osmancik)*. UCI Machine Learning Repository, 2019. – DOI: https://doi.org/10.24432/C5MW4Z 13, 61

[MK] Markelle Kelly, Kolby N. Rachel Longjohn L. Rachel Longjohn: *The UCI Machine Learning Repository*. https://archive.ics.uci.edu 60

[MSW20] M. Syukron, R. S. ; Widiharih, T.: Perbandingan Metode Smote Random Forest Dan Smote Xgboost Untuk Klasifikasi Tingkat Penyakit Hepatitis C Pada Imbalance Class Data. In: *Jurnal Gaussian* 9 (2020) 40

[PG16] Pooja Gulati, Manish G. Amita Sharma S. Amita Sharma: Theoretical Study of Decision Tree Algorithms to Identify Pivotal Factors for Performance Improvement: A Review. 141 (2016) 44

[PK20] Priyanka, N.A. ; Kumar, Dharmender: Decision Tree Classifier: A detailed survey. In: *International Journal of Information and Decision Sciences* 12 (2020), Nr. 3, S. 246. http://dx.doi.org/10.1504/ijids.2020.108141. – DOI 10.1504/ijids.2020.108141 22

[PR18] Putatunda, S. ; Rama, K.: A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost. In: *Proc. of the 2018 International Conference on Signal Processing and Machine Learning* (2018) 28

[PVG+11] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830 13, 42

[RGCR14] Rodriguez-Galiano, V.F. ; Chica-Rivas, M.: Evaluation of Different Machine Learning Methods for Land Cover Mapping of a Mediterranean Area

Using Multi-Seasonal Landsat Images and Digital Terrain Models. In: *International Journal of Digital Earth* 7 (2014), S. 492–509. http://dx.doi.org/10.1080/17538947.2012.748848. – DOI 10.1080/17538947.2012.748848 25

[Rug02] Ruggieri, S.: Efficient C4.5 [classification algorithm]. In: *IEEE Transactions on Knowledge and Data Engineering* 14 (2002), Nr. 2, S. 438–444. http://dx.doi.org/10.1109/69.991727. – DOI 10.1109/69.991727 17

[Sax17] Saxena, Rahul: How decision tree algorithm works. In: *URl: http://dataaspirant. com/2017/01/30/howdecision-tree-algorithm-works/.(accessed: 2019-01-28)* (2017) 15, 16

[SS98] Shapire, Robert E. ; Singer, Yoram: BoosTexter: A System for Multi-Label Text Categorization. In: *Machine Learning* 39 (1998), S. 135–168 24

[SS15a] Sinha, P. ; Sinha, P.: Comparative Study of Chronic Kidney Disease Prediction using KNN and SVM. In: *International Journal of Engineering Research and Technology* 4 (2015) 28

[SS15b] Sun, L. ; Schulz, K.: The Improvement of Land Cover Classification by Thermal Remote Sensing. In: *Remote Sensing* 7 (2015), S. 8368–8390 26

[SY15] Song, Yan-Yan ; Ying, LU: Decision tree methods: applications for classification and prediction. In: *Shanghai archives of psychiatry* 27 (2015), Nr. 2, S. 130 18, 19, 20, 21

[TSS09] Tu, My C. ; Shin, Dongil ; Shin, Dongkyoo: Effective diagnosis of heart disease through bagging approach. In: *2009 2nd international conference on biomedical engineering and informatics* IEEE, 2009, S. 1–4 10

[Was21] Waskom, Michael L.: seaborn: statistical data visualization. In: *Journal of Open Source Software* 6 (2021), Nr. 60, 3021. http://dx.doi.org/10.21105/joss.03021. – DOI 10.21105/joss.03021 13

[WB09] Waske, B. ; Braun, M.: Classifier Ensembles for Land Cover Mapping Using Multitemporal SAR Imagery. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 64 (2009), S. 450–457 27

[WCZ+19] Wu, Jia ; Chen, Xiu-Yun ; Zhang, Hao ; Xiong, Li-Dong ; Lei, Hang ; Deng, Si-Hao: Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb. In: *Journal of Electronic Science and Technology* 17 (2019), Nr. 1, 26-40. http://dx.doi.org/https://doi.org/10.11989/JEST.1674-862X.80904120. – DOI https://doi.org/10.11989/JEST.1674–862X.80904120. – ISSN 1674–862X 8

[Whi94] Whitley, Darrell: A genetic algorithm tutorial. In: *Statistics and Computing* 4 (1994). https://doi.org/10.1007/BF00175354 32

[WK09] Wu, Xindong ; Kumar, Vipin: *The top ten algorithms in data mining*. CRC press, 2009 18

[WLC+15] Wang, Z.L. ; Lai, C.G. ; Chen, X.H. ; Yang, B. ; Zhao, S.W. ; Bai, X.Y.: Flood Hazard Risk Assessment Model Based on Random Forest. In: *Journal of Hydrology* 527 (2015), S. 1130–1141 26

[XDM16] X. Deng, Y. D. Q. Liu L. Q. Liu ; Mahadevan, S.: An improved method to construct basic probability assignment based on the confusion matrix for classification problem. In: *Information Sciences* 340 (2016) 39, 40

[YCC+16] Ye, Fang ; Chen, Zhi-Hua ; Chen, Jie ; Liu, Fang ; Zhang, Yong ; Fan, Qin-Ying ; Wang, Lin ; Shi, Qiang: Chi-squared Automatic Interaction Detection Decision Tree Analysis of Risk Factors for Infant Anemia in Beijing, China. In: *Chinese Medical Journal* 129 (2016), Nr. 10, 1193-1199. http://dx.doi.org/10.4103/0366-6999.181955. – DOI 10.4103/0366–6999.181955 17

[YS20] Yang, Li ; Shami, Abdallah: On hyperparameter optimization of machine learning algorithms: Theory and practice. In: *Neurocomputing* 415 (2020), 295-316. http://dx.doi.org/https://doi.org/10.1016/j.neucom.2020.

07.061. – DOI https://doi.org/10.1016/j.neucom.2020.07.061. – ISSN 0925–2312 7

[YZ20] Yu, Tong ; Zhu, Hong: Hyper-parameter optimization: A review of algorithms and applications. In: *arXiv preprint arXiv:2003.05689* (2020) 9

[ZZ08] Zhao, Yongheng ; Zhang, Yanxia: Comparison of decision tree methods for finding active objects. In: *Advances in Space Research* 41 (2008), Nr. 12, S. 1955â1959. http://dx.doi.org/10.1016/j.asr.2007.07.020. – DOI 10.1016/j.asr.2007.07.020 22

[ZZ17] Zhang, L. ; Zhan, C.: Machine Learning in Rock Facies Classification: An Application of XGBoost. In: *Proc. of International Geophysical Conference, Qingdao, China* (2017) 28