Decision Tree Induction for Heart Disease Prediction

Narayan Das Nitol	**150204025**
Fahim Ahmed	**150204039**
Md Fazlay Rabbi	**150204105**
Abrar Jaheen Tahmid	**150204112**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Dhaka, Bangladesh

January 2020

Decision Tree Induction for Heart Disease Prediction

A thesis report
Submitted in partial fulfillment of the requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

Submitted by

Narayan Das Nitol	**150204025**
Fahim Ahmed	**150204039**
Md Fazlay Rabbi	**150204105**
Abrar Jaheen Tahmid	**150204112**

Supervised by

Dr. S. M. Abdullah Al-Mamun

Professor

Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Dhaka, Bangladesh

January 2020

CANDIDATES' DECLARATION

We, hereby, declare that the thesis report presented in this report is the outcome of the investigation performed by us under the supervision of Dr. S. M. Abdullah Al-Mamun, Professor, Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh. The work was spread over two final year courses, CSE4100: Project and Thesis-I and CSE4250: Project and Thesis-II, in accordance with the course curriculum of the Department for the Bachelor of Science in Computer Science and Engineering program.

It is also declared that neither this thesis report nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Narayan Das Nitol
150204025

Fahim Ahmed
150204039

Md Fazlay Rabbi
150204105

Abrar Jaheen Tahmid
150204112

CERTIFICATION

This thesis report titled, **"Decision Tree Induction for Heart Disease Prediction"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in January 2020.

Group Members:

Narayan Das Nitol	**150204025**
Fahim Ahmed	**150204039**
Md Fazlay Rabbi	**150204105**
Abrar Jaheen Tahmid	**150204112**

Dr. S. M. Abdullah Al-Mamun
Professor & Supervisor
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Dr. Kazi A. Kalpoma
Professor & Head
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

ACKNOWLEDGEMENT

It is an honor for us to thank those who made this research work possible. We owe our deepest gratitude to our supervisor, Dr. S. M. Abdullah Al-Mamun, Professor, Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, whose encouragement, guidance and support from the initial level to the end enabled us to develop an understanding of the subject and helped us a lot to complete our work. We would like to thank Dr. Kazi A Kalpoma, Professor and Head, Department of Computer Science and Engineering, Ahsanullah University of Science and Technology and all our teachers and friends for their constant encouragement and help in different phases including data preparation and testing. We are obviously thankful to the almighty for keeping us in good health and reason which was very much required for successful completion of this work.

Dhaka
January 2020

Narayan Das Nitol

Fahim Ahmed

Md Fazlay Rabbi

Abrar Jaheen Tahmid

ABSTRACT

Data mining is the process of discovering or extracting new patterns from large data sets involving methods from statistics and artificial intelligence. Classification and prediction are the techniques used to find out important data classes and predict probable trends. Decision Tree induction and Random Forests are important classification methods in data mining. They are commonly used in marketing, surveillance, fraud detection, scientific discovery, etc. As the classic algorithms for implementing decision trees, ID3 and CART algorithms and Random Forests have the merits of high classification speed, strong learning ability and simple construction. However, these algorithms have some disadvantages in practical applications. During classification, there arises a problem of inclining to choose attributes which have more value, and overlooking attributes which have less value. This work focuses on characteristics, challenges, advantages and disadvantages of various decision tree and Random Forest algorithms applied on heart disease datasets for prediction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Decision tree is a decision support tool that uses a tree like graph or model of decisions. It is a flowchart like structure in which each internal node represents a "test" on an attribute. Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation.
Random forests are an ensemble method used for classification. In Random Forest, we grow multiple trees as opposed to a single tree in Decision Tree model. But the question arises why to use multiple trees when the same work can be done by a single tree as well. One of the major problems of Decision Tree is overfitting which gives us a very bad predictive model and adding multiple trees in the random forest introduces randomness which in turn gets rid of overfitting and gives us a very good predictive model. To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.
Heart is the most important organ in living beings. If it does not function properly it will affect other parts of the body. As per the estimation of World Health Organization (WHO) 12 million deaths occur due to heart diseases, and nearly 80% of deaths in the world are caused by heart diseases. Now-a-days, at many places clinical test results are often based on doctor's intuition and experiences rather than information available in many large databases. Hence this process leads to unintentional biases, errors and a huge medical cost which affects the quality of service provided to patients.

The Risk Factors of Heart Diseases:

- **Heredity**: Most people know that the heart disease can run in families. If anybody has a family history of heart disease, he/she may be at greater risk for heart diseases.
- **Smoking**: Smoking is a major cause of heart attack. Nearly 40% of all people who smoke tobacco develop fatal heart and blood vessel diseases.
- **Cholesterol**: Cholesterol is a soft, waxy substance found among the lipids in the blood stream and in all the body's cells. Abnormal levels of lipids (fats) in the blood are risk factor of heart diseases.
- **High Blood Pressure**: High blood pressure is a common factor for serious heart problems.
- **Obesity**: The term obesity is used to describe the health condition of anyone significantly above his or her ideal healthy weight, and this is not good for a healthy heart.

1.2 Motivation

A decision tree is an important tool used for machine learning. It is one of the fastest ways to identify most significant variables and find relations between them. It can handle both numerical and categorical variables. It requires relatively little effort from users for data preparation. Output of decision trees is very easy to understand.
A random forest, as its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the model's prediction.
Prediction of heart diseases is a difficult and risky task. Since it is directly dependent on people's health, accuracy is a major factor. If not predicted accurately it can be disastrous. This research therefore focuses on the study of various decision tree techniques to predict heart diseases, and it attempts to highlight the relative performance of different algorithms. Most hospitals today employ some sort of hospital information systems to manage their healthcare or patient data. These systems typically generate huge amounts of data which take the form of numbers, text, charts and images. Unfortunately, these data are rarely used to support clinical decision making. This motivated us to get involved in this field.

1.3 Objectives

The main objective of this study is to be able to understand the process of predicting whether a given patient is affected with heart disease or not using different machine learning algorithms like Decision Trees and Random Forests on a qualified dataset. It targets to pointing out the correlations between different data attributes and to obtaining clear idea of selected data mining techniques. It also aims at analyzing the program output and comparing the results of different techniques. We would like to finally get to the point of inferring, based on our analysis of performance of the techniques, if there is any possibility to bring improvement in results.

1.4 Introduction to Our Report

Our steps of overall work described as follows:

1. Getting a thorough understanding of Decision Tree concepts.
2. Getting a thorough understanding of Random Forests.
3. Getting a thorough understanding of Heart Disease datasets.
4. implementing different algorithms on Heart Disease datasets.
5. Performing comparison and critical analysis of algorithm outputs.
6. Studying different algorithms for Decision Tree induction and Random Forests with a view to understanding their advantages and disadvantages, applicability to specific datasets etc.

This report has seven chapters: **Chapter 1** is the introductory one where we have discussed about general aspects of Decision Trees, Random Forests, Heart Diseases and our motivation for this work. In **Chapter 2**, we have discussed about research papers, books and web pages that we studied to gather knowledge about various aspects of our task. **Chapter 3** provides analysis of different Decision Tree algorithms and Random Forests with examples. In **Chapter 4**, we have discussed about the dataset that we have collected from Kaggle for our work. In **Chapter 5**, we have discussed about our implementation of the chosen algorithms, and in **Chapter 6**, we have shown the results of our implementation of those algorithms, that is, ID3 and CART and Random Forests and compared performance. **Chapter 7** is the final one presenting observations about our efforts, successes, failures and possible future works.

Chapter 2

Literature Review

2.1 Background

In course of searching for interesting machine learning tools for classification and prediction, we came across some publications about decision tree learning ([1]-[4]) and heart disease prediction ([5]-[7]). We also visited some relevant websites ([8]-[10]). We were inspired very much by those works to undertake our project-thesis task. Here is a brief description of the most influential of the works we studied. The survey paper by Sharma and others [1] provided us with a good guidance. From it we found important clues for looking into relevant publications. For basic concepts on machine learning, data mining and decision tree induction we primarily relied on the book by Han and others [2]. In this work we also noted that Data Mining is the core stage of Knowledge Discovery in Databases, which is a nontrivial extraction of implicit, novel, and potentially useful information from data. The other relevant works helped us in many ways. For example, Anbarasi and others [3] mentioned that statistics and machine learning are two main approaches which have been applied to predict the status of heart diseases based on the expression of the clinical data. We found diverse aspects of heart disease prediction in the works by Dangare and others [4] and by Ordonez [5]. For foundational knowledge on data mining, decision trees and classification we also relied on the works of Patel and others [6] and Last and others [7].
In 2004, Franck Le Duff et al [8] build an efficient decision tree for executing medical procedure. Data mining methods, they showed, may aid the clinicians in the prediction of survival of patients, and in the comparison of traditional analysis and data mining analysis which sorted the variables. This provided the importance of data and variables for building a decision tree.
In 2008, Sellappan Palaniappan, et. al. [9] performed a work, “Intelligent Heart Disease Prediction System (IHDPS) Using Data Mining Techniques". In this work they developed a system utilizing data mining methods, i.e. Decision Trees, Naïve Bayes and Neural Network.

2.2 Data Mining

Data Mining is about explaining the past and predicting the future by means of data analysis. Data mining is a multi-disciplinary field that combines statistics, machine learning, artificial intelligence and database technology. The value of data mining applications is often estimated to be very high. Many businesses have stored large amounts of data over years of operation, and data mining is able to extract very valuable knowledge from this data. The businesses are then able to leverage the extracted knowledge into more clients, more sales, and greater profits.This is also true in the engineering and medical fields [6].

The most distinguishing features of data mining are highlighted below:

Statistics:
The science of statistics is about collecting, classifying, summarizing, organizing, analyzing, and interpreting data.

Artificial Intelligence:
The study of computer algorithms is to deal with the simulation of intelligent behavior in order to perform those activities that are normally thought to require intelligence.

Machine Learning:
The study of the computer algorithms aims to learn in order to improve automatically through experience.

Database:
It is about the science and technology of collecting, storing and managing data so that users can retrieve, add, update or remove such data for their specific purposes and recording.

Data Warehousing:
It is about the science and technology of collecting, storing and managing data with advanced multi-dimensional reporting services in support of the decision-making processes.

Explaining the Past:
Data mining explains the past through data exploration.

Predicting the Future:
Data mining predicts the future by means of modeling.

Data Exploration:
Data Exploration is about describing the data by means of statistical and visualization techniques. We explore data in order to bring important aspects of that data into focus for further analysis.

2.3 Medical Data Mining

The knowledge discovered through data mining can be used for different applications, for example, healthcare industry. Nowadays, healthcare industry generates large amount of data about patients, disease diagnosis etc. Data mining provides a set of techniques to discover hidden patterns from medical diagnosis data. A major challenge facing Healthcare industry is quality of service. Quality of service implies diagnosing diseases correctly and providing effective treatment to patients. Poor diagnosis can lead to disastrous consequences that are unacceptable.
According to leading annual surveys worldwide millions of deaths occur due to heart attacks and strokes. The deaths due to heart diseases in many countries happen due to work overload, mental stress and many other problems. The diagnosis is often made based on doctor's experience and knowledge. This leads to unwanted results and excessive medical costs of treatments. Therefore, designing automated medical diagnosis systems that take advantage of huge collected data and decision support techniques deserve special attention, in our opinion.
Medical data mining has great potential for exploring the hidden patterns in the data sets of the medical domain. These patterns can be utilized for clinical diagnosis. However, the available raw medical data are widely distributed, heterogeneous in nature, and voluminous. These data need to be collected in an organized form. This collected data can be then integrated to form a hospital information system. Data mining technology provides a user-oriented approach to novel and hidden patterns in the data [4].

2.4 Heart Diseases

The heart is an important organ of human body. It is nothing more than a pump, which pumps blood through the body. If circulation of blood in body is inefficient, the organs like brain suffer and if heart stops working altogether, death occurs within minutes. Life is completely dependent on efficient working of the heart. The term Heart disease refers to disease of heart and blood vessel system within it [4].

A number of factors have been shown that increases the risk of Heart disease:

- Family history
- Smoking
- Poor diet
- High blood pressure

- High blood cholesterol
- Hyper tension

Factors like these are used to analyze the Heart disease. In many cases, diagnosis is generally based on patient's current test results and doctor's experience. Thus, the diagnosis is a complex task that requires much experience and high skill.
Heart disease is a broad term that includes all types of diseases affecting different components of the heart. Heart means 'cardio.' Therefore, all heart diseases belong to the category of cardiovascular diseases [4].

Some types of Heart diseases are described below:

1. Coronary heart disease, also known as coronary artery disease (CAD), is the most common type of heart diseases across the world. It is a condition in which plaque deposits block the coronary blood vessels leading to a reduced supply of blood and oxygen to the heart.
2. Angina pectoris is a medical term for chest pain that occurs due to insufficient supply of blood to the heart. Also known as angina, it is a warning signal for heart attack. The chest pain is at intervals ranging for few seconds to a few minutes.
3. Congestive heart failure is a condition where the heart cannot pump enough blood to the rest of the body. It is commonly known as heart failure.
4. Cardiomyopathy is the weakening of the heart muscle or a change in the structure of the muscle due to inadequate heart pumping. Some of the common causes of cardiomyopathy are hypertension, alcohol consumption, viral infections, and genetic defects.
5. Congenital heart disease, also known as congenital heart defect, refers to the formation of an abnormal heart due to a defect in the structure of the heart or its functioning. It is also a type of congenital disease that children are born with.
6. Arrhythmias is associated with a disorder in the rhythmic movement of the heartbeat. The heartbeat can be slow, fast, or irregular. These abnormal heartbeats are caused by a short circuit in the heart's electrical system. VII.Myocarditis is an inflammation of the heart muscle usually caused by viral, fungal, and bacterial infections affecting the heart. It is an uncommon disease with few symptoms like joins pain, leg swelling or fever that cannot be directly related to the heart

2.5 Decision Trees

In this section we will discuss about decision trees and its algorithms. How they work and their advantages and disadvantages.

2.5.1 Overview

The decision tree approach is a powerful tool for classification problems. There are two steps in this technique, building a tree and applying the tree to the dataset. There are many popular decision tree algorithms like ID3, CART, C4.5, CHAID and J48.

2.5.2 Types of Decision Trees

Decision trees are classified based on the types of target variables we have. Those can be of two types:

1. **Categorical Variable Decision Tree**: Decision Tree which has categorical target variable then it called as categorical variable decision tree.
2. **Continuous Variable Decision Tree**: Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

2.5.3 Important Terminology Related to Decision Trees

There are different terminology used with Decision trees:

- **Root Node**: It represents the initial entire population or sample, and this further gets divided into two or more homogeneous sets.
- **Splitting**: It is a process of dividing a node into two or more sub-nodes.
- **Decision Node**: When a sub-node splits into further sub-nodes, then it is called a decision node.
- **Leaf/ Terminal Node**: Nodes that do not split are called Leaf or Terminal nodes.
- **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning.
- **Branch / Sub-Tree**: A sub-section of entire tree is called a branch or sub-tree.

2.5.4 Overview of Decision Tree Algorithms

There are a couple of algorithms there to build a decision tree [1]. We consider two of those:

1. **CART (Classification and Regression Trees)**: This uses Gini Index(Classification) as a metric.
2. **ID3 (Iterative Dichotomiser 3)**: It uses Entropy function and Information gain as metrics.

2.5.4.1 CART Algorithm

Classification and Regression Trees or **CART** for short is an acronym introduced by Leo Breiman [1] to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems. A CART tree is a binary decision tree that is constructed by splitting a node into two child nodes repeatedly, beginning with the root node that contains the whole learning sample. CARTs support continuous and nominal attribute data and have average speed of processing.

CART Advantages:

- Simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Can handle both numerical and categorical data. Can also handle multi-output problems.
- Decision trees require relatively little effort from users for data preparation.

CART Disadvantages:

- CART may have unstable decision trees.
- CART splits only by one variable.

2.5.4.2 ID3 Algorithm

Iterative Dichotomiser 3(**ID3**) is a simple decision tree learning algorithm introduced in 1986 by Quinlan Ross [1]. It is serially implemented and based on simple steps. The basic idea of ID3 algorithm is to construct the decision tree by employing a top-down, greedy

search through the given sets to test each attribute at every tree node. In the decision tree method, information gain approach is generally used to determine suitable property for each node of a generated decision tree. Therefore, we can select the attribute with the highest information gain (entropy reduction in the level of maximum) as the test attribute of current node. In this way, the information needed to classify the training sample subset obtained from later on partitioning will be the smallest. So, the use of this property for partitioning the sample set contained in current node will make the mixture degree of different types for all generated sample subsets reduced to a minimum [2].

ID3 Advantages:

- Understandable prediction rules are created from the training data.
- Builds the fastest tree.
- Only need to test enough attributes until all data is classified.
- Finding leaf nodes enables test data to be pruned, reducing number of tests.

ID3 Disadvantages:

- Data may be over-fitted or overclassified, if a small sample is used.
- Only one attribute at a time is tested for making a decision.
- Classifying continuous data may be computationally expensive, as many trees must be generated to see where to break the continuum [1].

2.5.5 Best Attribute Selection

The most important part of decision tree algorithms is deciding the best attribute. But what does 'best' actually mean? In decision tree algorithms the best means the attribute which has most **information gain**.

2.5.5.1 Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous, the entropy is zero and if the sample is an equally divided one it has entropy of one.

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

$$E(S) = -p_{(+)} log_2 p_{(+)} - p_{(-)} log_2 p_{(-)}$$

- Entropy is **0 (Minimum)**, when all the examples belong to same class.
- Entropy is **1 (Maximum)**, when there are equal numbers of positive and negative examples.

2.5.5.2 Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches) [4].

Steps of Information Gain:

1. Entropy of the target is calculated.
2. The dataset is then split on different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy [1].
3. Attribute with the largest information gain is chosen as the decision node.
   (a) A branch with entropy of 0 is a leaf node.
   (b) A branch with entropy more than 0 needs further splitting.
4. The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

2.5.5.3 Avoiding Overfitting

The main problem with decision trees is that it is prone to overfitting. We could create a tree that could classify the data perfectly or we are not left with any attribute to split. This would work well in on the training dataset but will have a bad result on the testing dataset. There are two popular approaches to avoid this in decision trees: stop growing the tree before it becomes too large or prune the tree after it becomes too large. Pruning the tree, on the other hand, involves testing the original tree against pruned versions of it.

Two approaches which we can use to avoid overfitting are as follows:

- Pre-Pruning
- Post-Pruning

Pre-Pruning: In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it's difficult to choose an appropriate stopping point.

Post-Pruning: In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem then pruning is done as a post-pruning step. We use a cross-validation data to check the effect of our pruning. Using cross-validation data, it tests whether expanding a node will make an improvement or not. If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded i.e., the node should be converted to a leaf node [6].

2.6 Random Forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

2.6.1 Overview

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.
It is the simplest and widely used algorithm. Used for both classification and regression. It is an ensemble of randomized decision trees. Each decision tree gives a vote for the prediction of target variable. Random forest chooses the prediction that gets the most vote. An ensemble learning model aggregates multiple machine learning models to give a better performance. [10].

2.6.2 How Random Forests Works

"Random Forest builds multiple decision trees and merges them together to get a more accurate and stable prediction" [10].
One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Given below is a random forest with three trees:

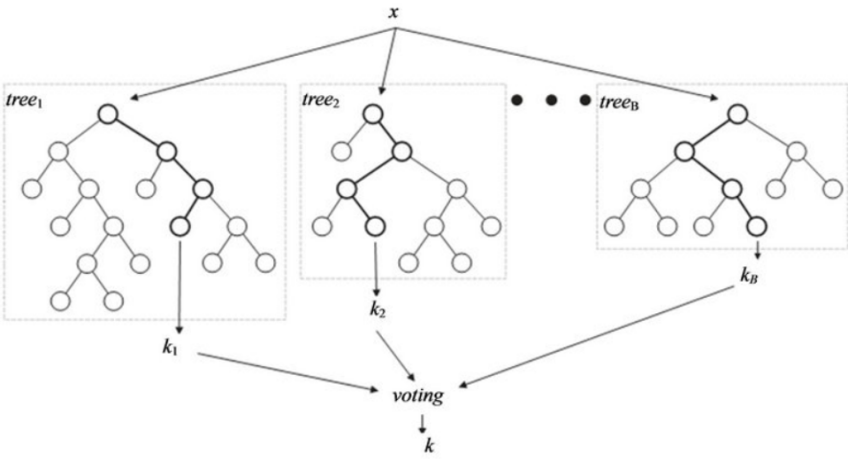


Figure 2.1: Random Forest Example [11]

A random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.
Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.
Therefore, in a random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. One can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

2.6.3 Real Life Analogy

Suppose "XYZ" wants to decide where to go during one-year vacation, so he/she asks the people who know him best for suggestions. The first friend he seeks out asks him about the likes and dislikes of his past travels. Based on the answers, he will give "XYZ" some advice. This is a typical decision tree algorithm approach. "XYZ"'s friend created rules to guide his

decision about what he should recommend, by using “XYZ’s” answers.
Afterwards, “XYZ” starts asking more and more of his friends to advise him and they again ask him different questions they can use to derive some recommendations from. Finally, “XYZ” chooses the places that were recommended by the most, and this is the typical random forest algorithm approach.

2.6.4 Feature Importance

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. sk-learn provides a great tool for this that measures a feature’s importance by looking at how much the tree nodes that use that feature reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results so the sum of all importance is equal to one [10].
By looking at the feature importance we can decide which features to possibly drop because they don’t contribute enough (or sometimes nothing at all) to the prediction process. This is important because a general rule in machine learning is that the more features you have the more likely your model will suffer from overfitting and vice versa.

2.6.5 Difference between Decision Tree and Random Forests

While random forest is a collection of decision trees, there are some differences:

- If we input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions. For example, to predict whether a person will click on an online advertisement, we might collect the ads the person clicked on in the past and some features that describe his/her decision. If we put the features and labels into a decision tree, it will generate some rules that help predict whether the advertisement will be clicked or not.

- In comparison, the random forest algorithm randomly selects observations and features to build several decision trees and then averages the results.

- Another difference is "deep" decision trees might suffer from overfitting. Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the subtrees. It’s important to note this doesn’t work every time and it also makes the computation slower, depending on how many trees the random forest builds [10].

2.6.6 Important Hyper-Parameters

The hyperparameters in random forest are either used to increase the predictive power of the model or to make the model faster.
Let's look at the hyperparameters of sklearn's built-in random forest function [12].

Increasing the predictive power:
Firstly, there is the **n-estimators** hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation. Another important hyperparameter is **max-features**, which is the maximum number of features random forest considers to split a node. sklearn provides several options, all described in the documentation. The last important hyperparameter is **min-sample-leaf**. This determines the minimum number of leafs required to split an internal node.

Increasing the model's speed:
The **n-jobs** hyperparameter tells the engine how many processors it is allowed to use. If it has a value of one, it can only use one processor. A value of "-1" means that there is no limit. The **random-state** hyperparameter makes the model's output replicable. The model will always produce the same results when it has a definite value of random-state and if it has been given the same hyperparameters and the same training data. Lastly, there is the **oob-score** (also called oob sampling), which is a random forest cross-validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out-of-bag samples. It's very similar to the leave-one-out-cross-validation method, but almost no additional computational burden goes along with it [10].

2.6.7 Advantages and Disadvantages

- One of the biggest advantages of random forest is its versatility. It can be used for both regression and classification tasks, and it's also easy to view the relative importance it assigns to the input features.
- Random forest is also a very handy algorithm because the default hyperparameters it uses often produce a good prediction result. Understanding the hyperparameters is pretty straightforward, and there's also not that many of them.
- One of the biggest problems in machine learning is overfitting, but most of the time

this won't happen thanks to the random forest classifier. If there are enough trees in the forest, the classifier won't overfit the model.

- The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained.

2.6.8 Field of Work

Here is some the application where random forest algorithms are widely used:

- Banking
- Medicine
- Stock Market
- E-commerce

Banking:
In the banking sector, random forest algorithm widely used in two main application. These are for finding the loyal customer and finding the fraud customers. The loyal customer means not the customer who pays well, but also the customer whom can take the huge amount as loan and pays the loan interest properly to the bank.

Medicine:
In medicine field, random forest algorithms are used to identify the correct combination of the components to validate the medicine. Random forest algorithms are also helpful for identifying the disease by analyzing the patient's medical records.

Stock Market:
In the stock market, random forest algorithms are used to identify the stock behavior as well as the expected loss or profit by purchasing the particular stock.

E-commerce:
In e-commerce, the random forests are used only in the small segment of the recommendation engine for identifying the likelihood of customers liking the recommended products based on the similar kinds of customers.

Chapter 3

Classification Using Decision Trees and Random Forests

3.1 Approach to Achieve Our Goal

We have chosen Decision Trees for our task. There are other algorithms also. So, why do we have to choose Decision trees? Well, there might be many reasons, but we considered a few, which are shown below.

1. Decision trees often mimic the human level thinking so it's so simple to understand the data and make some good interpretations.
2. Decision trees actually make you see the logic for the data to interpret (not like black box algorithms like SVM, ANN, etc) [13].

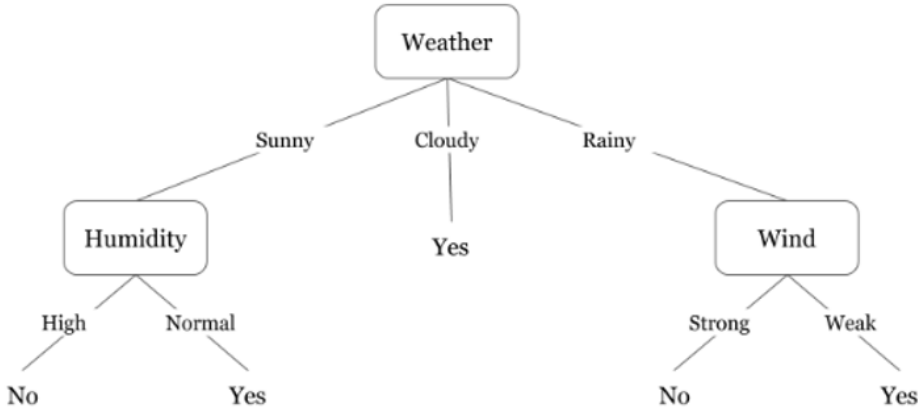


Figure 3.1: Example of a Decision Tree [14]

3.2 Classification with ID3 algorithm

ID3 algorithm, stands for Iterative Dichotomiser 3, is a classification algorithm that follows a greedy approach of building a decision tree by selecting a best attribute that yields maximum Information Gain (IG) or minimum Entropy (H).

3.2.1 Analysis

Let's just take a famous dataset in the machine learning world which is weather dataset (playing game Y or N based on weather condition) [15].

Table 3.1: Simple Weather Dataset for ID3

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	Yes
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

We have four x value (outlook, temp, humidity and windy) being categorical and one y value (play Y or N) also being categorical. So, we need to learn the mapping (what machine learning always does) between x and y.
This is a binary classification problem. Let's build the tree using the ID3 algorithm. To create a tree, we need to have a root node first, and we know that nodes are features/attributes (outlook, temp, humidity and windy).

Choose the Best Attribute:
Use the attribute with the highest **information gain** in ID3. In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called **entropy** that characterizes the (im)purity of an arbitrary collection of examples.

3.2.2 Steps

1. Compute the entropy for data-set
2. For every attribute/feature:
   (a) Calculate entropy for all categorical values
   (b) Take average information entropy for the current attribute
   (c) Calculate gain for the current attribut
3. Pick the highest gain attribute.
4. Repeat until we get the tree we desired [15].

We can summarize the ID3 algorithm as illustrated below:

$$Entropy(S) = \sum -p(I) * log_2(p(I))$$

$$Gain(S,A) = Entropy(S) - \sum [p(S|A) * Entropy(S|A)]$$

Entropy:

We need to calculate the entropy first. Decision column consists of 14 instances and includes two labels: Yes and No. There are **9** decisions labeled Yes, and **5** decisions labeled No.

Entropy (Decision) = - p(Yes) * log2(p(Yes)) - p(No) * log2(p(No))
Entropy (Decision) = - (9/14) * log2(9/14) - (5/14) * log2(5/14) = 0.940

Now, we need to find the most dominant factor for decisions.

Wind factor on decision:

Gain (Decision, Wind) = Entropy (Decision) – [p (Decision | Wind) * Entropy (Decision | Wind)]

Wind attribute has two labels: **weak** and **strong**. We would reflect it to the formula.

Gain (Decision, Wind) = Entropy (Decision) – [p (Decision | Wind=Weak) * Entropy (Decision | Wind=Weak)] – [p (Decision | Wind=Strong) * Entropy (Decision | Wind=Strong)]

Now, we need to calculate **(Decision | Wind=Weak) and (Decision | Wind=Strong)** respectively.

Weak wind factor on decision:

Table 3.2: ID3 Weak Wind Factor

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

There are 8 instances for **weak wind**. Decision of 2 items are no and 6 items are yes as illustrated below:

Entropy (Decision | Wind=Weak) = – p(No) * log2(p(No)) – p(Yes) * log2(p (Yes))
Entropy (Decision | Wind=Weak) = – (2/8) * log2(2/8) – (6/8) * log2(6/8) = 0.811

Strong wind factor on decision:

Table 3.3: ID3 Strong Wind Factor

Day	Outlook	Temp.	Humidity	Wind	Decision
2	sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

Here, there are 6 instances for strong wind. Decision is divided into two equal parts:

Entropy (Decision | Wind=Strong) = – p(No) * log2(p(No)) – p(Yes) * log2(p(Yes))
Entropy (Decision | Wind=Strong) = – (3/6) * log2(3/6) – (3/6) * log2(3/6) = 1

Now, we can turn back to **Gain (Decision, Wind)** equation.

Gain (Decision, Wind) = Entropy (Decision) – [p (Decision | Wind=Weak) * Entropy (Decision | Wind=Weak)] – [p (Decision | Wind=Strong) * Entropy (Decision | Wind=Strong)] = 0.940 – [(8/14) * 0.811] – [(6/14) * 1] = 0.048

Calculations for **wind** column is over. Now, we need to apply same calculations for other columns to find the most dominant factor on decision.

Other factors on decision:

We have applied similar calculation on the other columns.

Gain (Decision, Outlook) = 0.246
Gain (Decision, Temperature) = 0.029
Gain (Decision, Humidity) = 0.151

As seen, **outlook** factor on decision produces the highest score. That's why, **outlook** decision will appear in the root node of the tree.

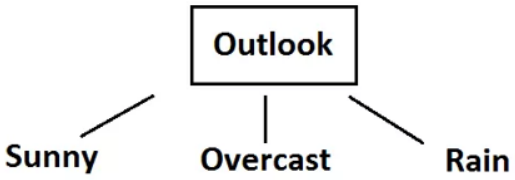


Figure 3.2: Root decision on the tree

Now, we need to test dataset for custom subsets of outlook attribute.

Overcast outlook on decision:

Basically, decision will always be Yes if **outlook** were **overcast**.

Table 3.4: ID3 Overcast Outlook

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Sunny outlook on decision:

Table 3.5: ID3 Sunny Outlook

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Here, there are 5 instances for **sunny outlook**. Decision would be probably 3/5 percent no, 2/5 percent yes.

Gain (Outlook=Sunny | Temperature) = 0.570
Gain (Outlook=Sunny | Humidity) = 0.970
Gain (Outlook=Sunny | Wind) = 0.019

Now, **humidity** is the decision because it produces the highest score if **outlook** were sunny. At this point, decision will always be no if **humidity** were high.

Table 3.6: ID3 Sunny Outlook Prediction

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

On the other hand, decision will always be yes if **humidity** were normal.

Table 3.7: ID3 Sunny Outlook Prediction-2

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Finally, it means that we need to check the **humidity** and decide if **outlook** were sunny.

Rain outlook on decision:

Table 3.8: ID3 Rain Outlook

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Gain (Outlook=Rain | Temperature)
Gain (Outlook=Rain | Humidity)
Gain (Outlook=Rain | Wind)

Here, **wind** produces the highest score if **outlook** were **rain**. That's why, we need to check **wind** attribute in 2nd level if **outlook** were rain. So, it is revealed that decision will always be Yes if **wind** were weak and **outlook** were rain.

Table 3.9: ID3 Rain Outlook Prediction

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes

What's more, decision will be always No if **wind** were strong and **outlook** were **rain**.

Table 3.10: ID3 Rain Outlook Prediction-2

Day	Outlook	Temp.	Humidity	Wind	Decision
6	Rain	Cool	Normal	Strong	No
14	Rain	Mild	High	Strong	No

3.2.3 Results

Our final decision tree takes the following form:

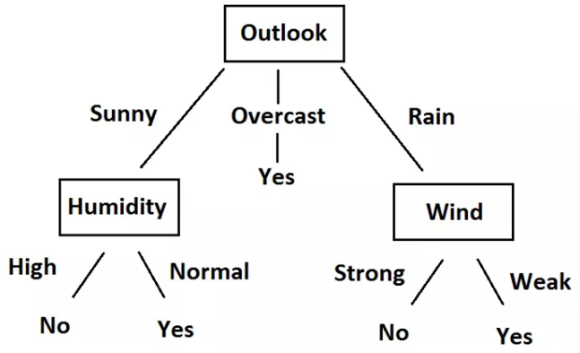


Figure 3.3: ID3 Final Decision Tree

3.3 Classification with CART algorithm

A **Classification And Regression Tree** (CART), is a predictive model, which explains how an outcome variable's values can be predicted based on other values. A CART output is a decision tree where each fork is a split in a predictor variable and each end node contains a prediction for the outcome variable.

3.3.1 Analysis

In CART we use Gini index as a metric. We use the Gini Index as our cost function used to evaluate splits in the dataset.
Our target variable is Binary variable which means it take two values (Yes and No). There can be 4 combinations.

Gini Index for Binary Target variable is **1 - P2(Target=0) - P2(Target=1)**

$$= 1 - \sum_{t=0}^{t=1} P_t^2$$

A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split. A perfect separation results in a Gini score of 0, whereas the worst case split that results in 50/50 classes.
We calculate it for every row and split the data accordingly in our binary tree. We repeat this process recursively.

For Binary Target variable, Max Gini Index value:
= 1 - (1/2)2 - (1/2)2
= 1 – 2*(1/2)2
= 1 – 2*(1/4)
= 1 – 0.5
= 0.5

Similarly, if Target Variable is categorical variable with multiple levels, the Gini Index will be still similar. If Target variable takes k different values, the Gini Index will be

$$= 1 - \sum_{t=0}^{t=k} P_t^2$$

Maximum value of Gini Index could be when all target values are equally distributed.
Minimum value of Gini Index will be 0 when all observations belong to one label.

3.3.2 Steps

1. Compute the Gini index for data-set

2. For every attribute/feature:

(a) Calculate Gini index for all categorical values

(b) Take average information entropy for the current attribute

(c) Calculate the Gini gain

3. Pick the best Gini gain attribute

4. Repeat until we get the tree we desired [16].

This is the same dataset we used before.

Table 3.11: Simple Weather Dataset for CART

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Outlook:

Outlook is a nominal feature. It can be **sunny**, **overcast** or **rain**. I will summarize the final decisions for **outlook** feature.

Table 3.12: CART Outlook Data

Outlook	Yes	No	Number of Instances
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5

Gini (Outlook=Sunny) = 1 – (2/5)2 – (3/5)2 = 1 – 0.16 – 0.36 = 0.48
Gini (Outlook=Overcast) = 1 – (4/4)2 – (0/4)2 = 0
Gini (Outlook=Rain) = 1 – (3/5)2 – (2/5)2 = 1 – 0.36 – 0.16 = 0.48

Then, we will calculate weighted sum of Gini indexes for outlook feature.

Gini (Outlook) = (5/14) * 0.48 + (4/14) * 0 + (5/14) * 0.48 = 0.171 + 0 + 0.171 = 0.342

Temperature:

Similarly, **temperature** is a nominal feature and it could have 3 different values: **Cool**, **Hot** and **Mild**. Let's summarize decisions for **temperature** feature.

Table 3.13: CART Temperature Data

Temperature	Yes	No	Number of Instances
Hot	2	2	4
Cold	3	1	4
Mild	4	2	6

Gini (Temp=Hot) = 1 – (2/4)2 – (2/4)2 = 0.5
Gini (Temp=Cool) = 1 – (3/4)2 – (1/4)2 = 1 – 0.5625 – 0.0625 = 0.375
Gini (Temp=Mild) = 1 – (4/6)2 – (2/6)2 = 1 – 0.444 – 0.111 = 0.445

We'll calculate weighted sum of Gini index for temperature feature.

Gini (Temp) = (4/14) * 0.5 + (4/14) * 0.375 + (6/14) * 0.445 = 0.142 + 0.107 + 0.190 = 0.439

Humidity:

Humidity is a binary class feature. It can be high or normal.

Table 3.14: CART Humidity Data

Humidity	Yes	No	Number of Instances
High	3	4	7
Normal	6	1	7

Gini (Humidity=High) = 1 – (3/7)2 – (4/7)2 = 1 – 0.183 – 0.326 = 0.489
Gini (Humidity=Normal) = 1 – (6/7)2 – (1/7)2 = 1 – 0.734 – 0.02 = 0.244

Weighted sum for humidity feature will be calculated next
Gini (Humidity) = (7/14) * 0.489 + (7/14) * 0.244 = 0.367

Wind:

Wind is a binary class similar to **humidity**. It can be **weak** and **strong**.

Table 3.15: CART Wind Data

Wind	Yes	No	Number of Instances
Weak	6	2	8
Strong	3	3	6

Gini (Wind=Weak) = 1 – (6/8)2 – (2/8)2 = 1 – 0.5625 – 0.062 = 0.375
Gini (Wind=Strong) = 1 – (3/6)2 – (3/6)2 = 1 – 0.25 – 0.25 = 0.5
Gini (Wind) = (8/14) * 0.375 + (6/14) * 0.5 = 0.428

Time to Decide:

We've calculated Gini index values for each feature. The winner will be **outlook** feature because its cost is the lowest.

Table 3.16: CART Time to Decide

Feature	Gini Index
Outlook	0.342
Temperature	0.439
Humidity	0.367
Wind	0.428

We'll put **outlook** decision at the top of the tree.

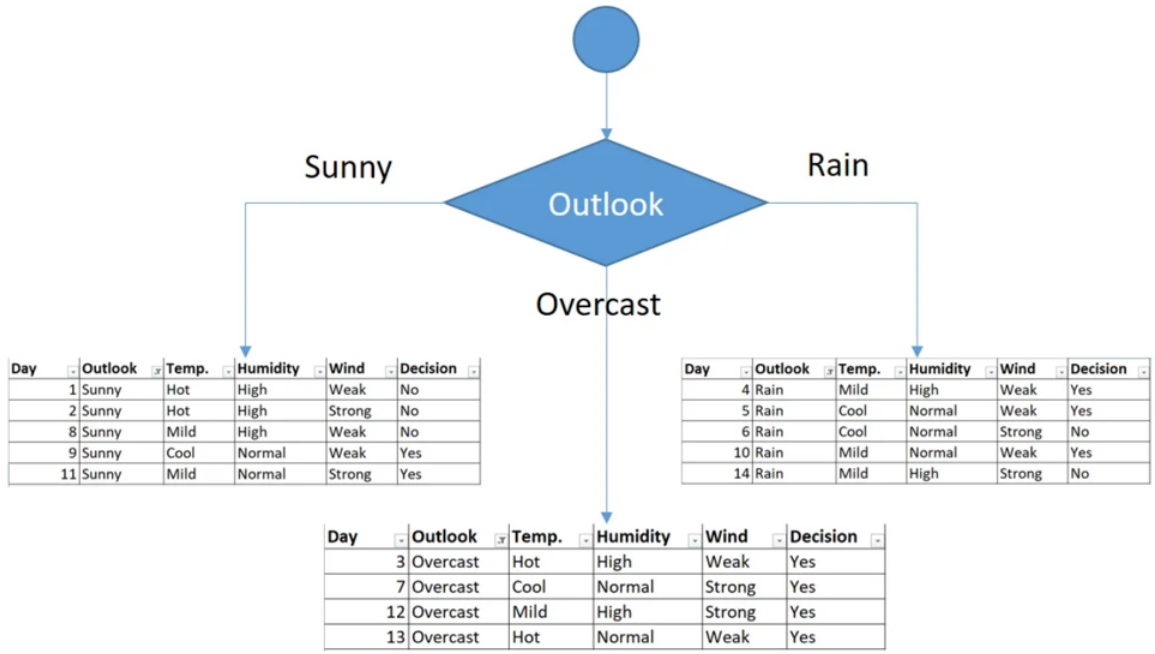
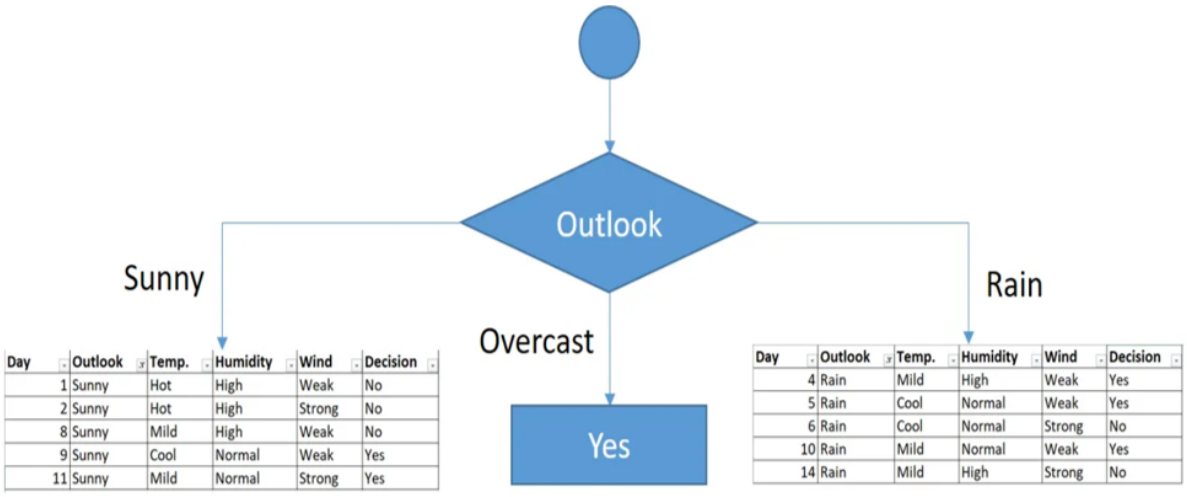


Figure 3.4: First Decision of CART

First Decision would be **Outlook** feature You might realize that sub dataset in the **overcast** leaf has only yes decisions. This means that **overcast** leaf is over.



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure 3.5: Tree for overcast outlook leaf

We will apply same principles to those sub datasets in the following steps.
Focus on the sub dataset for **sunny outlook**. We need to find the **gini index** scores for **temperature**, **humidity** and **wind** features respectively.

Table 3.17: CART Gini Index

Day	Outlook	Temp	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Gini of temperature for sunny outlook:

Table 3.18: CART Gini Temperature

Temperature	Yes	No	Number Of Instances
Hot	0	2	2
Cold	1	0	1
Mild	1	1	2

Gini (Outlook=Sunny and Temp.=Hot) = 1 – (0/2)2 – (2/2)2 = 0
Gini (Outlook=Sunny and Temp.=Cool) = 1 – (1/1)2 – (0/1)2 = 0
Gini (Outlook=Sunny and Temp.=Mild) = 1 – (1/2)2 – (1/2)2 = 1 – 0.25 – 0.25 = 0.5
Gini (Outlook=Sunny and Temp.) = (2/5) * 0 + (1/5) * 0 + (2/5) * 0.5 = 0.2

Gini of humidity for sunny outlook:

Table 3.19: CART Gini Humidity

Humidity	Yes	No	Number of Instances
High	0	3	3
Normal	2	0	2

Gini (Outlook=Sunny and Humidity=High) = 1 – (0/3)2 – (3/3)2 = 0
Gini (Outlook=Sunny and Humidity=Normal) = 1 – (2/2)2 – (0/2)2 = 0
Gini (Outlook=Sunny and Humidity) = (3/5) * 0 + (2/5) * 0 = 0

Gini of wind for sunny outlook:

Table 3.20: CART Gini Wind

Wind	Yes	No	Number Of Instances
Weak	1	2	3
Strong	1	1	2

Gini (Outlook=Sunny and Wind=Weak) = 1 – (1/3)2 – (2/3)2 = 0.266
Gini (Outlook=Sunny and Wind=Strong) = 1- (1/2)2 – (1/2)2 = 0.2
Gini (Outlook=Sunny and Wind) = (3/5) * 0.266 + (2/5) * 0.2 = 0.466

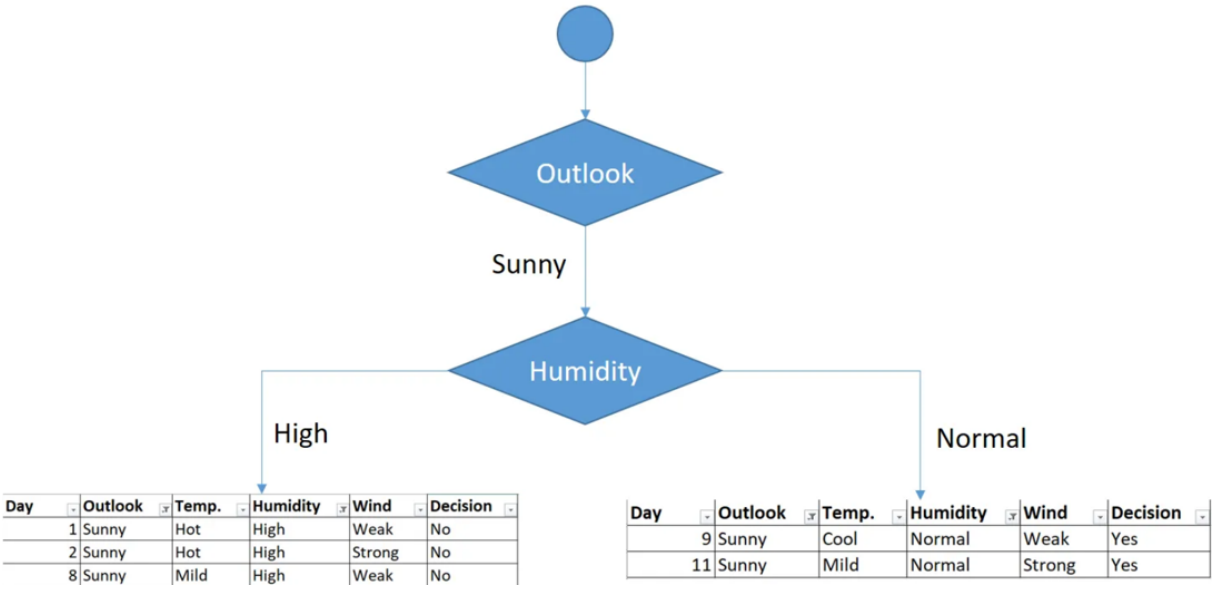
Decision for sunny outlook:

We've calculated **gini index** scores for feature when **outlook** is **sunny**. The winner is **humidity** because it has the lowest value.

Table 3.21: CART Sunny Outlook Decision

Feature	Gini Index
Temperature	0.2
Humidity	0
Wind	0.466

We'll put **humidity** check at the extension of **sunny outlook**.

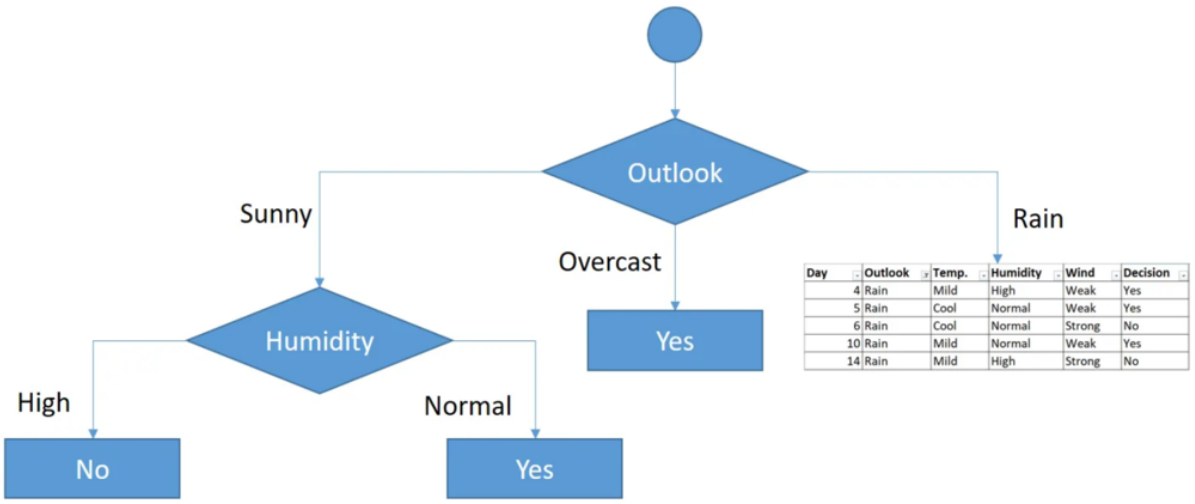


Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Figure 3.6: Sub datasets for high and normal humidity

As seen, decision is always No for high **humidity** and sunny **outlook**. On the other hand, decision will always be Yes for normal **humidity** and **sunny** outlook. This branch is over.



Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure 3.7: Decisions for high and normal humidity

Now, we need to focus on **rain outlook**.

Rain outlook:

Table 3.22: CART Rain Outlook Data

Day	Outlook	Temp	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

We'll calculate **gini index** scores for **temperature**, **humidity** and **wind** features when outlook is rain.

Gini of temperature for rain outlook:

Table 3.23: CART Gini Temperature of Rain Outlook

Temperature	Yes	No	Number of Instances
Cold	1	1	2
Mild	2	1	3

Gini (Outlook=Rain and Temp.=Cool) = 1 – (1/2)2 – (1/2)2 = 0.5
Gini (Outlook=Rain and Temp.=Mild) = 1 – (2/3)2 – (1/3)2 = 0.444
Gini (Outlook=Rain and Temp.) = (2/5) * 0.5 + (3/5) * 0.444 = 0.466

Gini of humidity for rain outlook:

Table 3.24: CART Gini Humidity of Rain Outlook

Humidity	Yes	No	Number of Instances
High	1	1	2
Normal	2	1	3

Gini (Outlook=Rain and Humidity=High) = 1 – (1/2)2 – (1/2)2 = 0.5
Gini (Outlook=Rain and Humidity=Normal) = 1 – (2/3)2 – (1/3)2 = 0.444
Gini (Outlook=Rain and Humidity) = (2/5) * 0.5 + (3/5) * 0.444 = 0.466

Gini of wind for rain outlook:

Table 3.25: CART Gini Wind of Rain Outlook

Wind	Yes	No	Number of Instances
Weak	3	0	3
Strong	0	2	2

Gini (Outlook=Rain and Wind=Weak) = 1 – (3/3)2 – (0/3)2 = 0
Gini (Outlook=Rain and Wind=Strong) = 1 – (0/2)2 – (2/2)2 = 0
Gini (Outlook=Rain and Wind) = (3/5) * 0 + (2/5) * 0 = 0

Decision for rain outlook:

The winner is **wind** feature for **rain outlook** because it has the minimum **gini index** score in features.

Table 3.26: CART Rain Outlook Decision

Feature	Gini Index
Temperature	0.466
Humidity	0.466
Wind	0

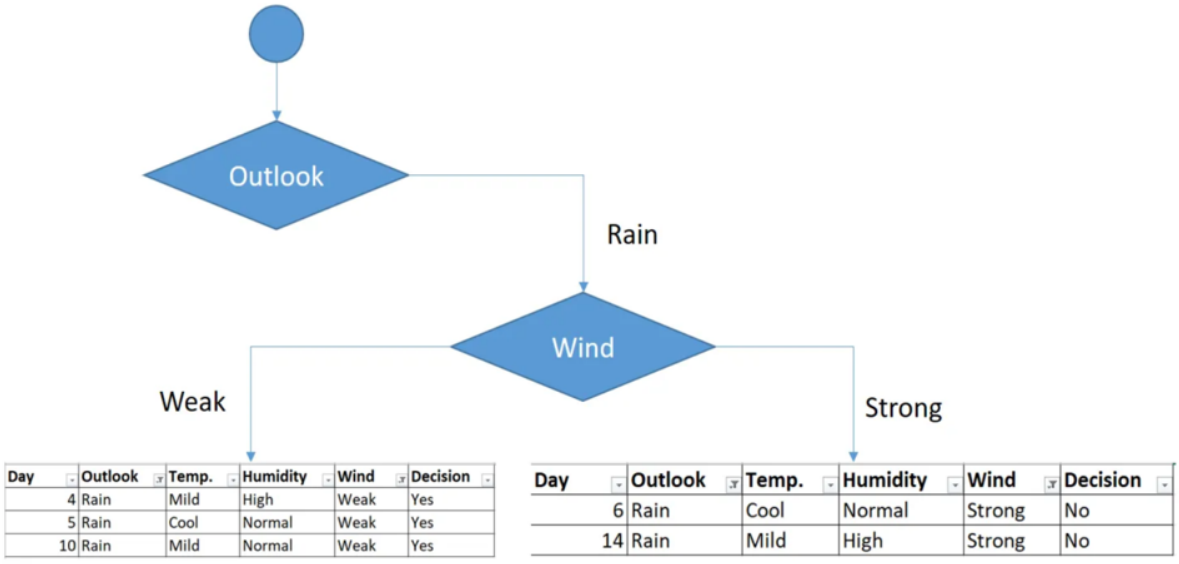


Figure 3.8: Sub data sets for weak and strong wind and rain outlook

3.3.3 Results

Final form of the decision tree by CART algorithm:

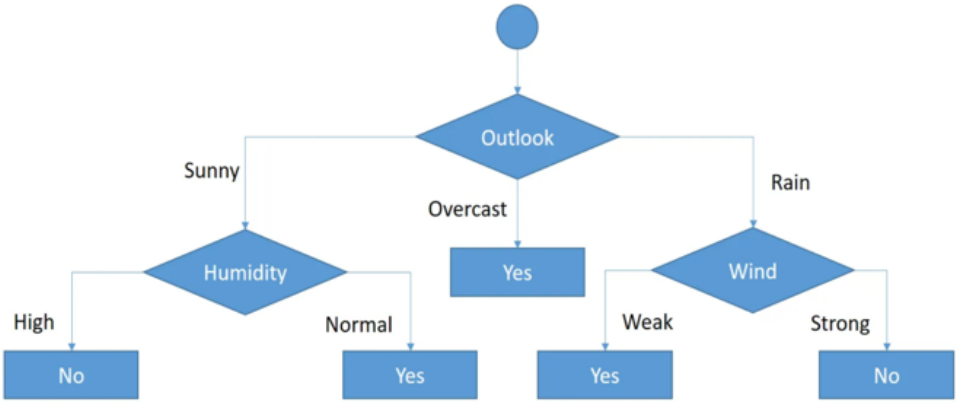


Figure 3.9: CART Final Decision Tree

3.4 Classification with Random Forests algorithm

A Random Forest algorithm is a supervised classification and regression algorithm. As the name suggests, this algorithm randomly creates a forest with several trees. Generally, the more trees in the forest the more robust the forest looks like.
Similarly, in the random forest classifier, the higher the number of trees in the forest, greater is the accuracy of the results [17].

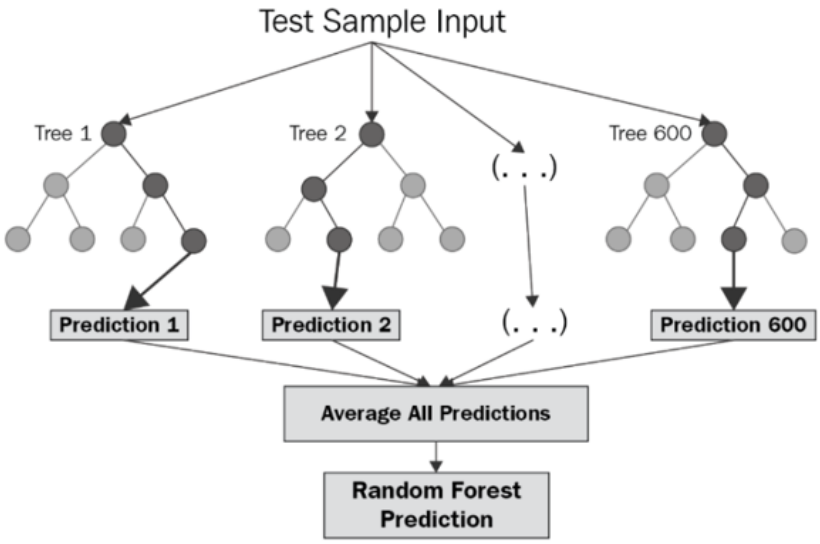


Figure 3.10: Random Forests Example [10]

3.4.1 Analysis

Let's say that you're looking to buy a house, but you're unable to decide which one to buy. So, you consult a few agents and they give you a list of parameters that you should consider before buying a house. The list includes:

- Price of the house
- Locality
- Number of bedrooms
- Parking space
- Available facilities

These parameters are known as predictor variables, which are used to find the response variable. Here's a diagrammatic illustration of how you can represent the above problem statement using a decision tree:

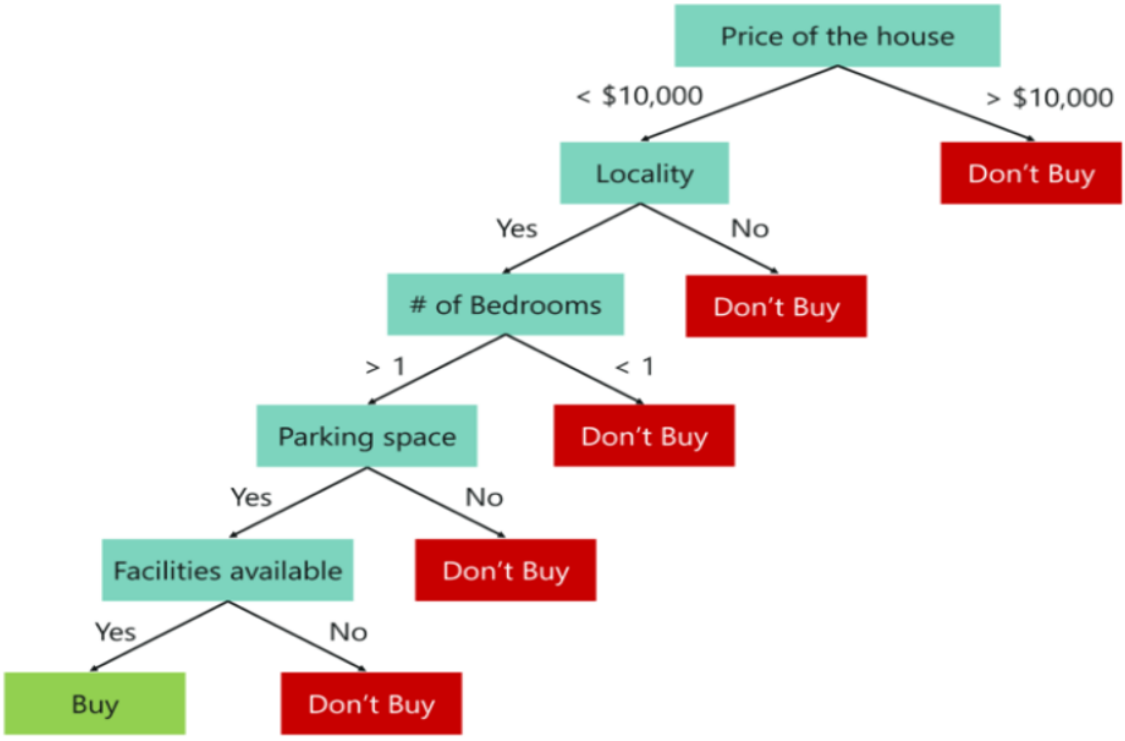


Figure 3.11: Decision Tree Example [18]

Now let's see how Random Forest would solve the same problem.

As mentioned earlier, Random forest is an ensemble of decision trees, it randomly selects a set of parameters and creates a decision tree for each set of chosen parameters [17].

Let's take a look at the figure below:

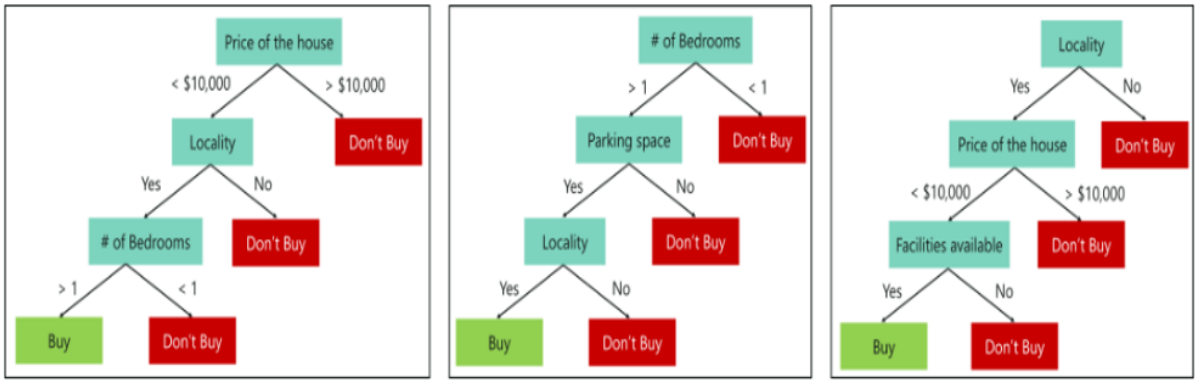


Figure 3.12: Random Forests with 3 Decision Trees [18]

Here are 3 Decision Trees and each Decision Tree is considering only 3 parameters of the data set. Each decision tree predicts the outcome based on the respective predictor variables used in that tree and finally takes the average of the results from all the decision trees in the random forests.

Why Random Forests?

1. Even though Decision trees are convenient and easily implemented, they lack accuracy. Decision trees work very effectively with the training data that was used to build them, but they're not flexible when it comes to classifying the new sample. Which means that the accuracy during testing phase is very low.
2. This happens due to a process called Overfitting.
3. This means that the disturbance in the training data is recorded and learned as concepts by the model. But the problem here is that these concepts do not apply to the testing data and negatively impact the model's ability to classify the new data, hence reducing the accuracy on the testing data.

3.4.2 Steps

To understand Random forests, let's consider the sample data set below [17]. In this data set we have four predictor variables, namely:

- Weight
- Blood flow
- Blocked Arteries
- Chest Pain

Table 3.27: Random Forests Simple Dataset

Blood Flow	Blocked Arteries	Chest Pain	Weight	Heart Disease
Abnormal	No	No	130	No
Normal	Yes	Yes	195	Yes
Normal	No	Yes	218	No
Abnormal	Yes	Yes	180	Yes

These variables are used to predict whether or not a person has heart disease. We're going to use this data set to create a Random Forest that predicts if a person has heart disease or not.

Creating a Random Forests:

Now we will create a random forest and then show every steps how it works and why it works like that

Step 1: Create a Bootstrapped Data Set

Bootstrapping is an estimation method used to make predictions on a data set by re-sampling it. To create a bootstrapped data set, we must randomly select samples from the original data set. A point to note here is that we can select the same sample more than once.

Table 3.28: Random Forests Bootstrapped Data

Blood Flow	Blocked Arteries	Chest Pain	Weight	Heart Disease
Normal	Yes	Yes	195	Yes
Abnormal	No	No	130	No
Abnormal	Yes	Yes	180	Yes
Abnormal	Yes	Yes	180	Yes

We randomly selected samples from the original data set and created a bootstrapped data set. In real-world problems we'll never get such a small data set, thus creating a bootstrapped data set is a little more complex.

Step 2: Creating Decision Trees

Our next task is to build a Decision Tree by using the **bootstrapped data** set created in the

previous step. Since we're making a Random Forest, we will not consider the entire data set that we created, instead we'll only use a random subset of variables at each step.
In this example, we're only going to consider two variables at each step. So, we begin at the root node, here we randomly select two variables as candidates for the root node.
Our next step is to repeat the same process for each of the upcoming branch nodes. Here, we again select two variables at random as candidates for the branch node and then choose a variable that best separates the samples.

In the same way, we build the tree by only considering random subsets of variables at each step. By following the above process, our tree would look something like this:

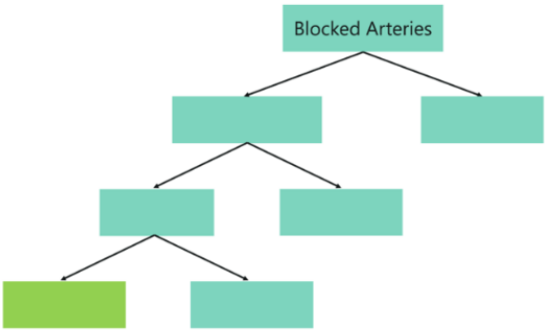


Figure 3.13: Random Forests Initial Tree

Step 3: Go back to Step 1 and Repeats
As mentioned earlier, Random Forest is a collection of Decision Trees. Each Decision Tree predicts the output class based on the respective predictor variables used in that tree. Finally, the outcome of all the Decision Trees in a Random Forest is recorded and the class with the majority votes is computed as the output class.
Thus, we must now create more decision trees by considering a subset of random predictor variables at each step. To do this, we go back to **step 1**, create a new bootstrapped data set and then build a Decision Tree by considering only a subset of variables at each step.
So, by following the above steps, our Random Forest would look something like this:

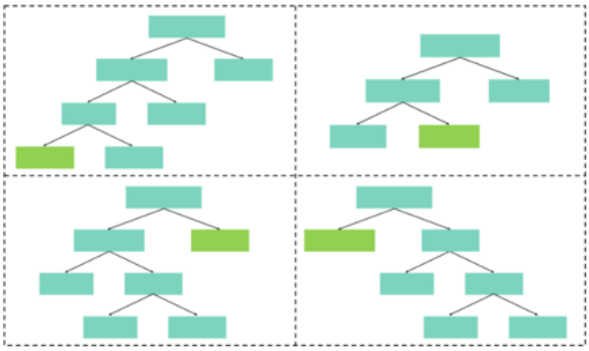


Figure 3.14: Random Forests Iteration

This iteration is performed 100's of times, therefore creating multiple decision trees with each tree computing the output, by using a subset of randomly selected variables at each step.

"Having a variety of Decision Trees in a Random Forest is what makes it more effective than an individual Decision Tree created using all the features and the whole data set" [17].

Step 4: Predicting the outcome of a new data point

Now that we've created a random forest, let's see how it can be used to predict whether a new patient has heart disease or not. The below diagram has the data about the new patient. All we have to do is run this data down the decision trees that we made.

The first tree shows that the patient has heart disease, so we keep a track of that in a table as shown in the figure:

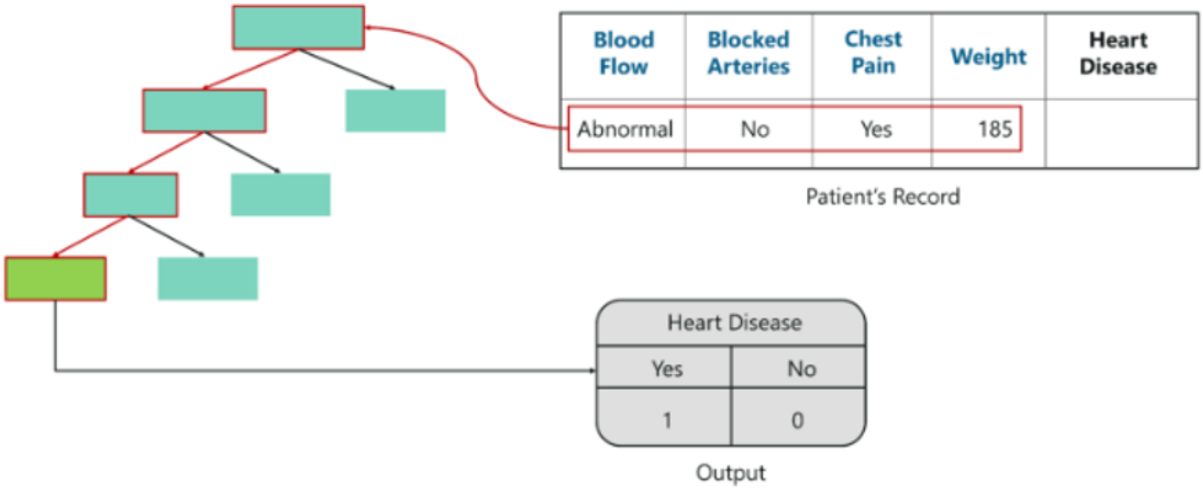


Figure 3.15: Random Forests Output

Similarly, we run this data down the other decision trees and keep a track of the class predicted by each tree. After running the data down all the trees in the Random Forest, we check which class got the majority votes. In our case, the class 'Yes' received the greatest number of votes, hence it's clear that the new patient has heart disease.

3.4.3 Results

Our final step is to evaluate the Random Forest model. Earlier while we created the bootstrapped data set, we left out one entry/sample since we duplicated another sample. In a real-world problem, about 1/3rd of the original data set is not included in the bootstrapped data set.

The following figure shows the entry that didn't end up in the bootstrapped data set:

Table 3.29: Random Forests Out-Of-Bag Example

Blood Flow	Blocked Arteries	Chest Pain	Weight	Heart Disease
Normal	No	Yes	218	No

The data set that is not included in the bootstrapped data set is known as the Out-Of-Bag (**OOB**) data set. "The Out-Of-Bag data set is used to check the accuracy of the model, since the model wasn't created using this **OOB** data it will give us a good understanding of whether the model is effective or not" [17].

In our case, the output class for the **OOB** data is 'No'. So, if we run the **OOB** data down the Decision trees, we must get a majority of 'No' votes. Although, in our case we only have one OOB, this process is carried out for all the samples in the **OOB** set. Therefore, eventually, we can measure the accuracy of a Random Forest by the proportion of **OOB** samples that are correctly classified. The proportion of **OOB** samples that are incorrectly classified is called the Out-Of-Bag Error.

Chapter 4

Dataset Analysis

4.1 Data Source

The data used in this study is taken from a European foundation for Heart diseases. The details are as follows:

Creators:

1. Hungarian Institute of Cardiology. Budapest: AndrasJanosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Donor:
David W. Aha (aha '@' ics.uci.edu) (714) 856-8779

Table 4.1: Dataset Overview [19]

Data Set Characteristics	Multivariate	**Number of Instances**	303	**Area**	Life
Attribute Characteristics	Categorical, Integer, Real	**Number of Attribute**	75	**Date Donated**	32325
Associated Tasks	Classification	**Missing Values**	Yes	**Number of Web Hits**	872608

4.2 Dataset Preprocessing

This dataset contains records with 76 attributes, but all published experiments refer to using a subset containing only 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence [19].
The 14 attributes taken from all 76 are given in table below:

Table 4.2: Dataset Attributes [19]

Clinical Features	**Description**
Age	Instance age in years
Sex	Instance gender
Cp	Chest pain type
Testbps (mmHg)	Resting blood pressure
Chol (mg/dl)	Serum cholesterol
Fbs	Fasting blood sugar
Restecg	Resting electrocardiographic results
Thalach	Maximum heart rate achieved
Exang	Exercise induced angina
Oldpeak	ST depression induced by exercise relative to rest
Slope	The slope of the peak exercise ST segment
Ca	Number of major vessels (0-3) colored by flourosopy
Thal	3 = normal; 6 = fixed defect; 7 = reversible defect
Diagnosis	Diagnosis of heart disease

4.3 Dataset Description

The description of the attributes that we take into consideration are shown in the table below:

Table 4.3: Attribute Documentation [19]

Name	Type	Description
Age	Continuous	Age in Years
Sex	Discrete	1=Male 0=Female
Cp	Discrete	Chest Pain Type: 1= Typical Angina 2= Atypical Angina 3= Non-Anginal Pain 4= Asymptomatic
Trestbps	Continuous	Resting Blood Pressure(In Mm Hg)
Chol	Continuous	Serum Cholesterol in Mg/Dl
Fbs	Discrete	Fasting Blood Sugar>120 Mg/Dl: 1=True 0=False
Rsetecg	Discrete	0=Normal
		1= Having ST-T Wave Abnormality 1= Having ST-T Wave Abnormality 2=Showing Probable or Define Left Ventricular Hypertrophy by Estes Criteria
Thalach	Continuous	Maximum Heart Rate Achieved
Exang	Discrete	Exercise Induced Angina: 1=Yes 0=No
OldPeak	Continuous	Depression Induced by Exercise Relative to Rest
Slope	Discrete	1=UP Sloping 2= Flat 3= Down Sloping
Ca	Discrete	Number of Major Vessels Colored by Fluoroscopy That Ranged Between 0 And 3
Thal	Discrete	3= Normal 6=Fixed Defect 7=Reversible Defect
Diagnosis	Discrete	0=Healthy 1=Patient Who Is Subject to Possible Heart Disease

There are 14 attributes used in this system, including **8 symbolic** and **6 numeric**.

4.4 Dataset Distribution

In this section, we have shown the distribution of our dataset. We have shown all the attribute of our dataset in graphical representation.

4.4.1 Dataset Distribution in Histogram

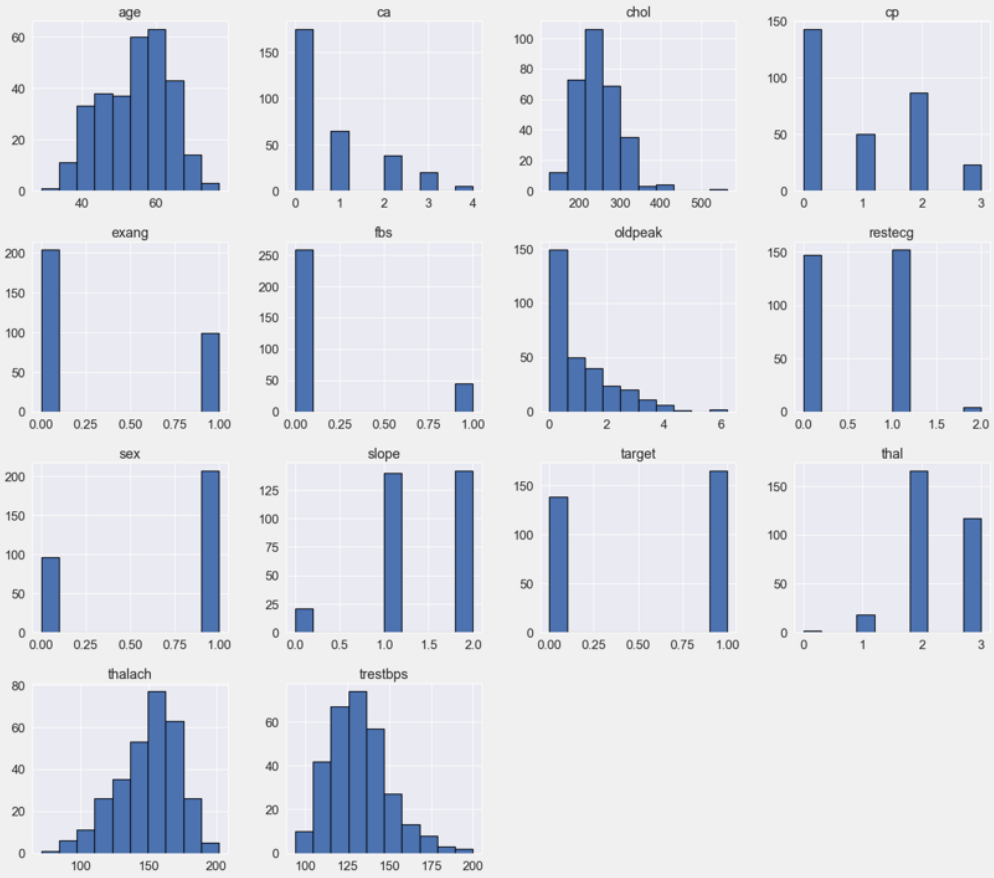


Figure 4.1: Dataset Distribution

4.4.2 Number of Different Attributes

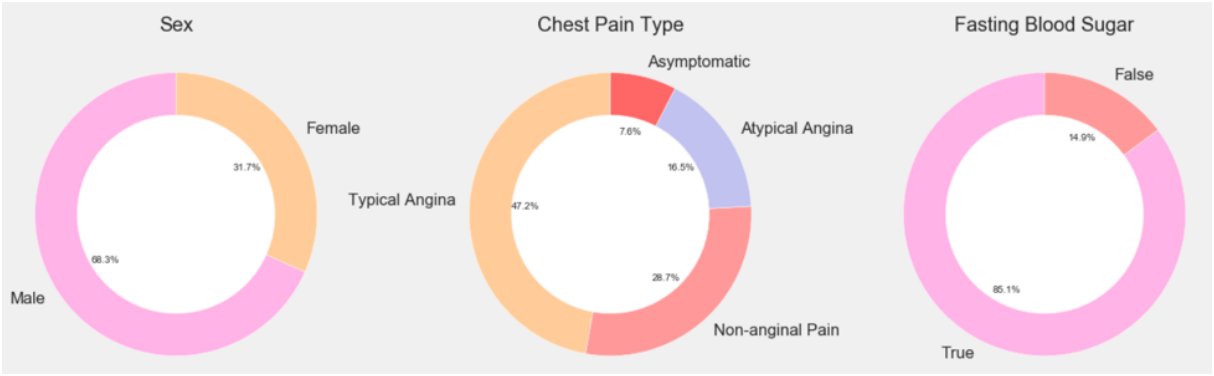


Figure 4.2: Dataset Distribution for Sex, Chest Pain and Fasting Blood Sugar

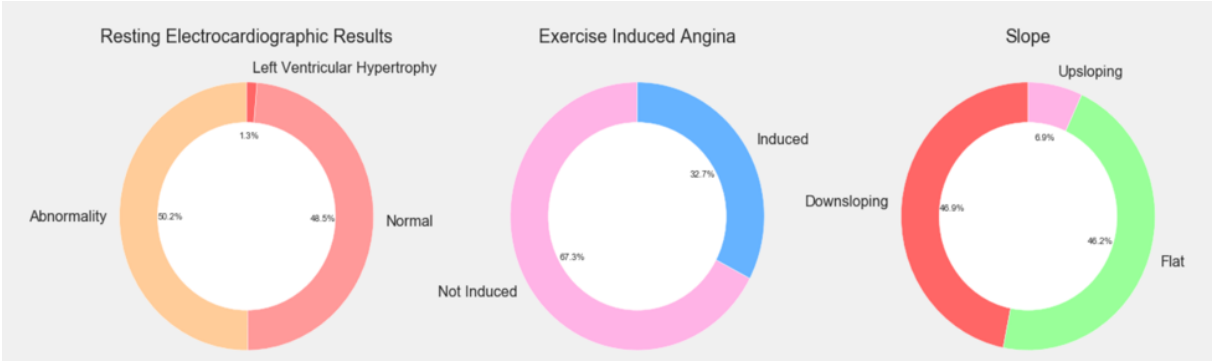


Figure 4.3: Dataset Distribution for ECG, EIA and Slope

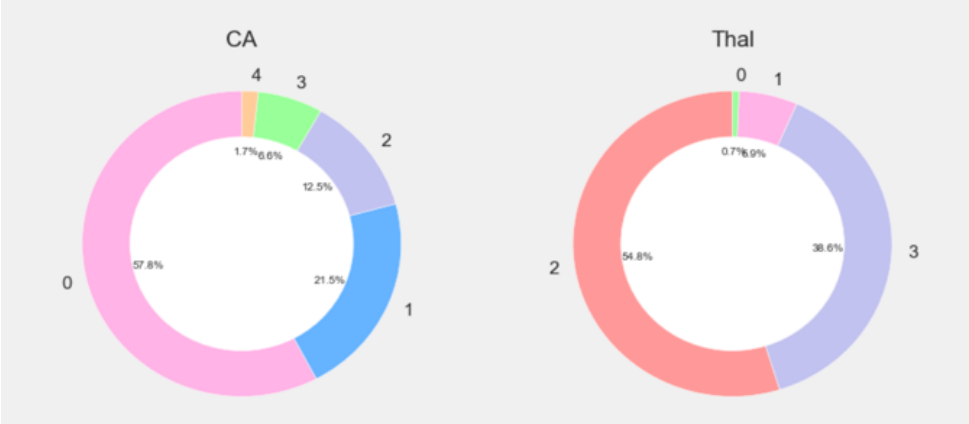


Figure 4.4: Dataset Distribution for CA and Thal

4.4.3 Our Target Attribute:

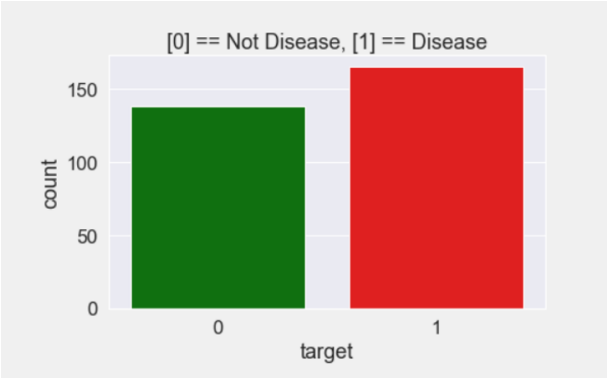


Figure 4.5: Target Attribute

4.4.4 Disease Status

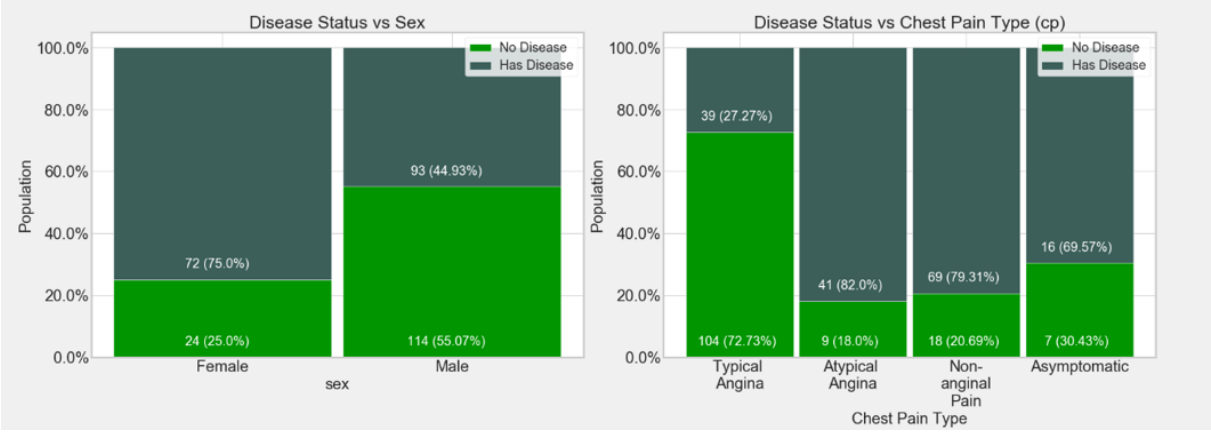


Figure 4.6: Disease Status for Sex and CP

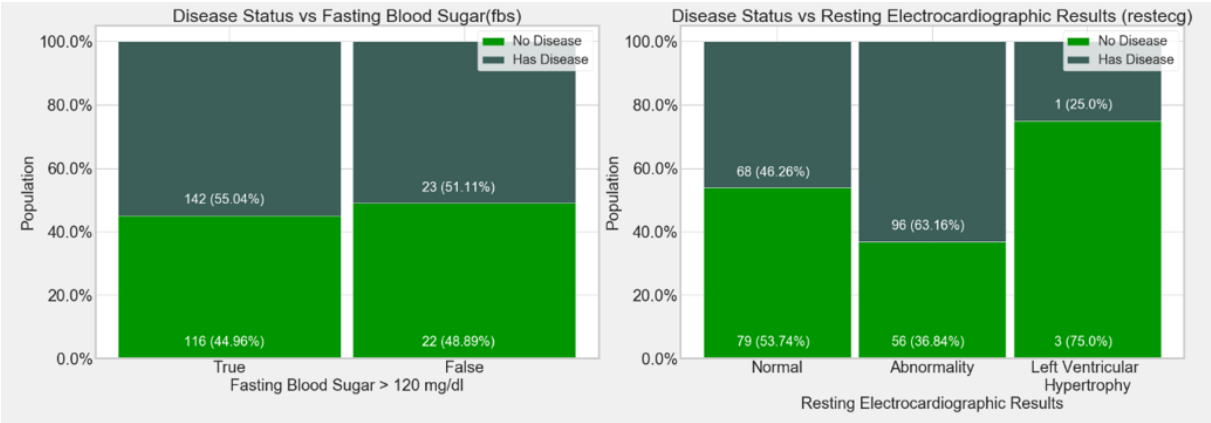


Figure 4.7: Disease Status for FBS and ECG

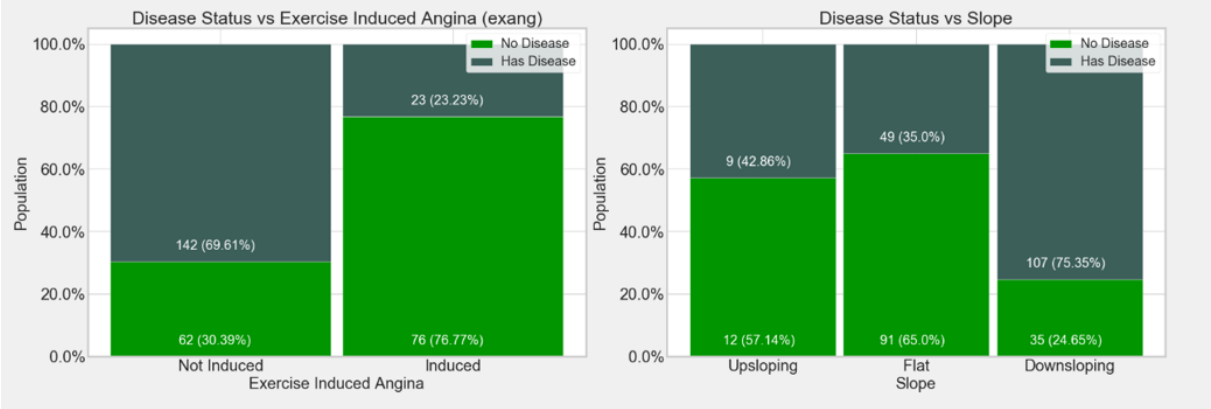


Figure 4.8: Disease Status for Angina and Slope

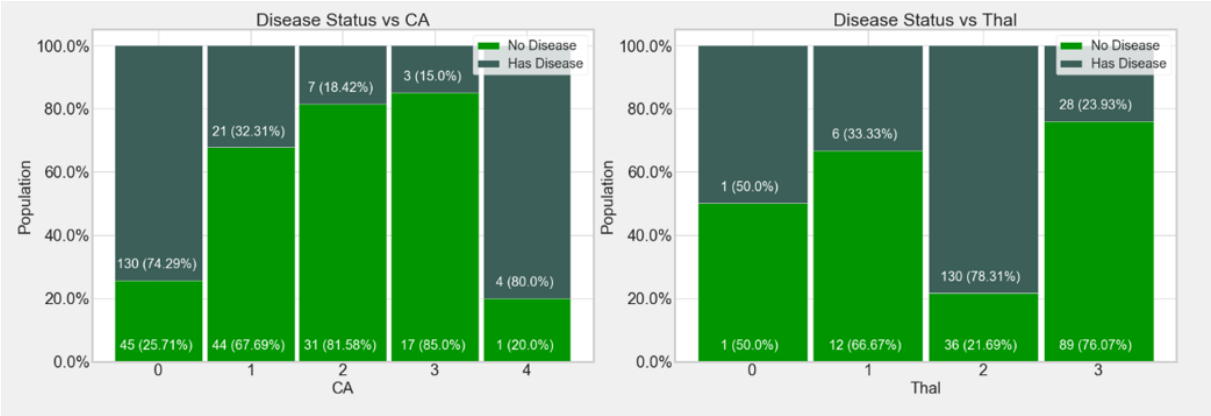


Figure 4.9: Disease Status for CA and Thal

Chapter 5

Implementation of the Algorithms

5.1 Experimental Setup

Table 5.1: Experimental Setup

IDE	Language	Library
Anaconda	Python	NumPy, Matplotlib, Pandas, Seaborn Scikit-learn, Graphviz, PyDotPlus

Python is a high-level general programming language and is very widely used in all types of disciplines such as general programming, web development, software development, data analysis, machine learning etc. Python is used for this project because it is very flexible and easy to use and also documentation and community support is very large [20].

Pandas is a very powerful package which enables us for scientific computing. It comes with sophisticated functions and is able to perform N-dimensional array, algebra, Fourier transform etc. NumPy is used everywhere in data analysis and image processing. Different other libraries are built on NumPy, where NumPy acts as a base stack for those libraries [21].

Pandas is an open source BSD licensed software specially written for python programming language. It provides complete set of data analysis tools for python and is the best competitor for programming languages. Most important feature of Pandas is performing time series analysis [22].

Seaborn is a library used for data visualization and is created by using python programming language. It is a high-level library stacked on top of matplotlib. Seaborn is more attractive and informative than matplotlib and very easy to use and is tightly integrated with NumPy and Pandas [23].

Scikit-learnis a widely used popular library for machine learning, it is a third-party extension to SciPy. For this study, scikit-learn is used because it is based on python and can interoperate to NumPy library. It is also very easy to use [24].

Graphviz layout programs take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser [25].

PyDotPlus is an improved version of the old pydot project that provides a Python Interface to Graphviz's Dot language [25].

Experimental Issues:

- If there is a large number of same types of labelled data continuously then we need to shuffle the main dataset with other labelled data. Otherwise the tree is going to be biased.
- We have to remove the less important attributes from the dataset so it gives us the right prediction.
- We have to take the perfect (not more not less) number of trees to make the prediction right.

5.2 ID3

The Process of ID3 algorithm implemented by us from scratch is given below:

1. At first, we imported our required libraries like "pandas", "numpy", "seaborn", "pprint", "matplotlib".
2. Then we loaded our dataset (heart.csv) and prepared it.
3. In this section, we apply ID3 algorithm.
   (a) First, we check the impurity.
   (b) Then we classify with its corresponding label.
   (c) We then find all potential splits for each attribute.
   (d) Next, by calculating overall entropy, we find the best splitting value and attribute.
4. Finally, we find the accuracy of test data [16].

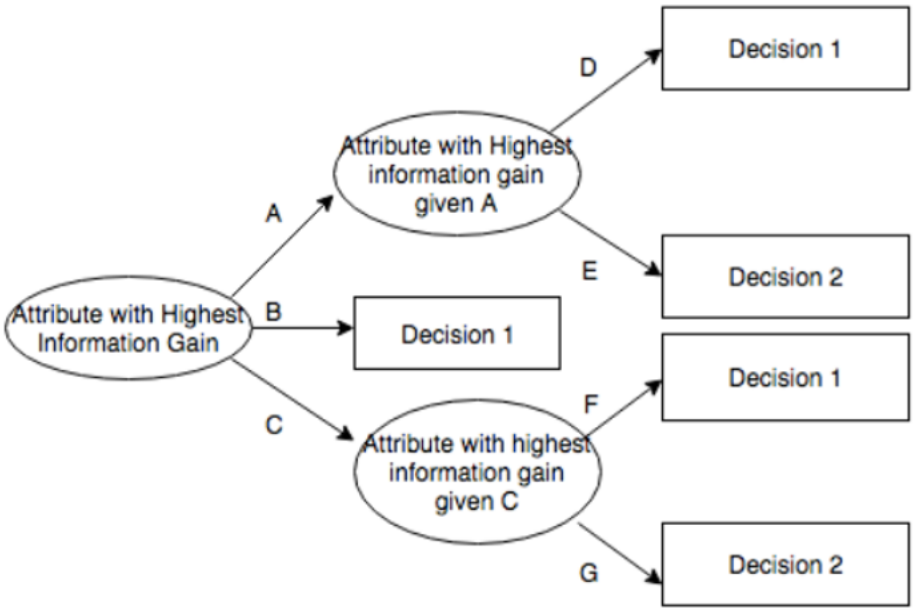


Figure 5.1: Flowchart of ID3 Algorithm [14]

Python code implementing the algorithm, written from scratch and also using built-in functions for induction and testing of the decision tree, has been given in Appendix A.

5.3 CART

The process of CART algorithm using Decision Tree Classifier (Criterion = 'gini') from sklearn is given below:

1. At first, we imported our required libraries like "pandas", "sklearn", "matplotlib".
2. Then we loaded our dataset (heart.csv) and prepared it.
3. We then split the data into test dataset and train dataset.
4. Next, we applied Decision Tree Classifier to train our train dataset.
5. Finally, we found the accuracy of test data [16].

Python code implementing the algorithm, written using built-in functions for induction and testing of the decision tree, has been given in Appendix B.

5.4 Random Forests

The process of Random Forest algorithm is given below:

1. Randomly select “k” features from total “m” features, where k < m.
2. From among the “k” features, calculate the node “d” using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat steps 1 to 3 until “l” number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for “n” number of times to create “n” number of trees [17].

The random forest algorithm starts with randomly selecting “k” features out of a total of “m” features.
In the next stage, we are using the randomly selected “k” features to find the root node by using the best split approach.
In the next stage, we are calculating the daughter nodes using the same best split approach.
The first 3 stages are repeated until we form the tree with a root node and having the target as the leaf node.
Finally, we repeat 1 to 4 stages to create “n” randomly created trees. This randomly created trees forms the random forest.

Random Forest prediction pseudocode:

To perform prediction using the trained random forest algorithm the steps executed are as follows:

1. Take the test features and use the rules of each randomly created decision tree to predict the outcome and store the predicted outcome (target).
2. Calculate the votes for each predicted target.
3. Consider the target with highest vote as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created tree. Suppose, we formed 100 random decision trees to from the random forest. Each tree will predict its own target (outcome) for the same test feature. If 100 random decision trees produce3 unique targets x, y and z, then

the votes of x are nothing but number of x prediction by trees out of those 100. Likewise, for other 2 targets y and z. Let's say, out of 100 random decision trees 60 trees are predicting the target x. Then the final random forest returns x as the predicted target. This concept of voting is known as majority voting. Python code implementing the algorithm, written from scratch and also using built-in functions for induction and testing of the random forest has been given in Appendix C.

Figure 5.2: Workflow of Random Forests Algorithm [10]

Chapter 6

Result Analysis

6.1 ID3

Final output of **ID3 – Built-in** algorithm:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	label	classification	classification_correct
204	62	0	0	160	164	0	0	145	0	6.2	0	3	3	0	0	True
159	56	1	1	130	121	0	0	163	0	0.0	2	0	3	1	1	True
219	48	1	0	130	256	1	0	150	1	0.0	2	2	3	0	0	True
174	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0	0	True
184	50	1	0	150	143	0	0	128	0	2.6	1	0	3	0	0	True
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0	0	True
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3	0	0	True
119	46	0	0	138	243	0	0	152	1	0.0	1	0	2	1	1	True
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3	0	0	True
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1	1	True
51	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1	1	True
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3	0	0	True
278	58	0	1	136	319	1	0	152	0	0.0	2	2	2	0	1	False
229	64	1	2	125	309	0	1	131	1	1.8	1	0	3	0	0	True
208	49	1	2	120	188	0	1	139	0	2.0	1	3	3	0	1	False
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0	1	False
58	34	1	3	118	182	0	0	174	0	0.0	2	0	2	1	1	True
220	63	0	0	150	407	0	0	154	0	4.0	1	3	3	0	0	True
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1	1	True
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3	0	0	True
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1	1	True
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0	0	True
70	54	1	2	120	258	0	0	147	0	0.4	1	0	3	1	1	True
138	57	1	0	110	201	0	1	126	1	1.5	1	0	1	1	1	True
122	41	0	2	112	268	0	0	172	1	0.0	2	0	2	1	1	True
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1	1	True
191	58	1	0	128	216	0	0	131	1	2.2	1	3	3	0	0	True
80	41	1	2	112	250	0	1	179	0	0.0	2	0	2	1	1	True
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1	1	True
123	54	0	2	108	267	0	0	167	0	0.0	2	0	0	1	1	True
157	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1	1	True

Figure 6.1: ID3 - Built-in Output Matrix

ID3 Complete Tree:

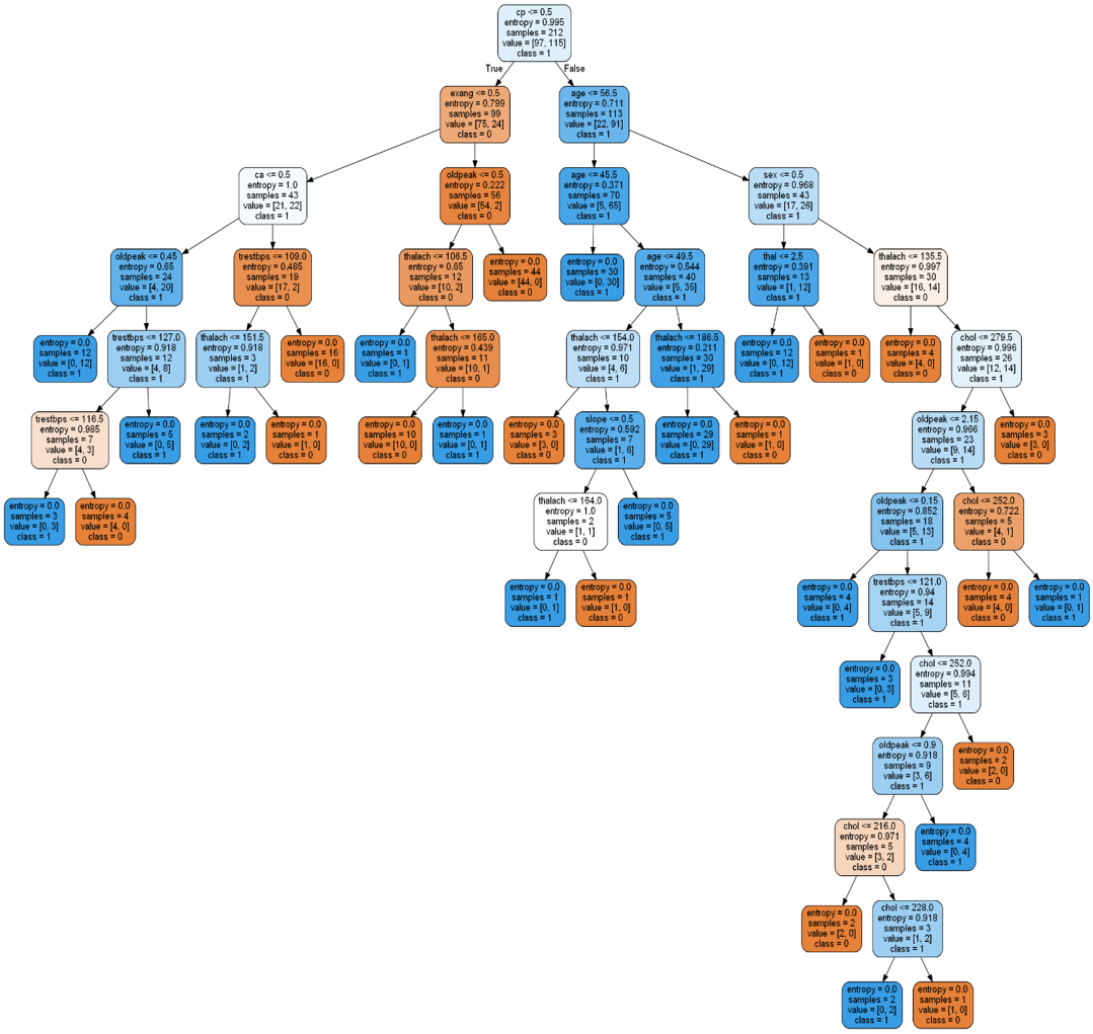


Figure 6.2: ID3 Tree at Max Level 12

ID3 Tree: Level 0-1:

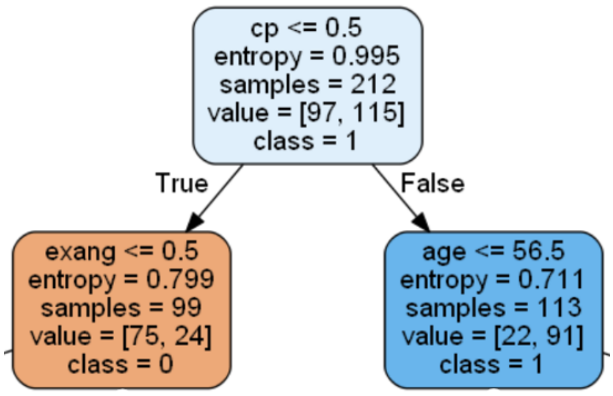


Figure 6.3: ID3 Tree: Level 0-1

ID3 Tree: Level 2-3:

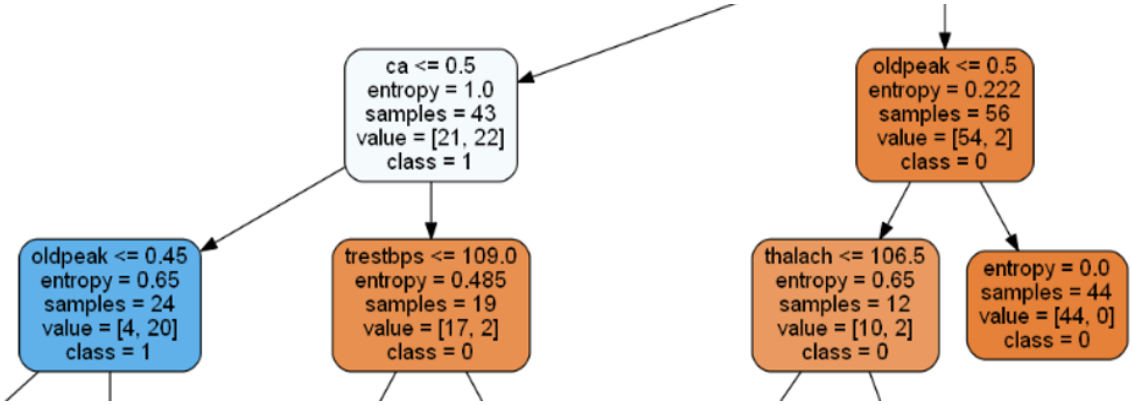


Figure 6.4: ID3 Tree: Level 2-3 Left Child

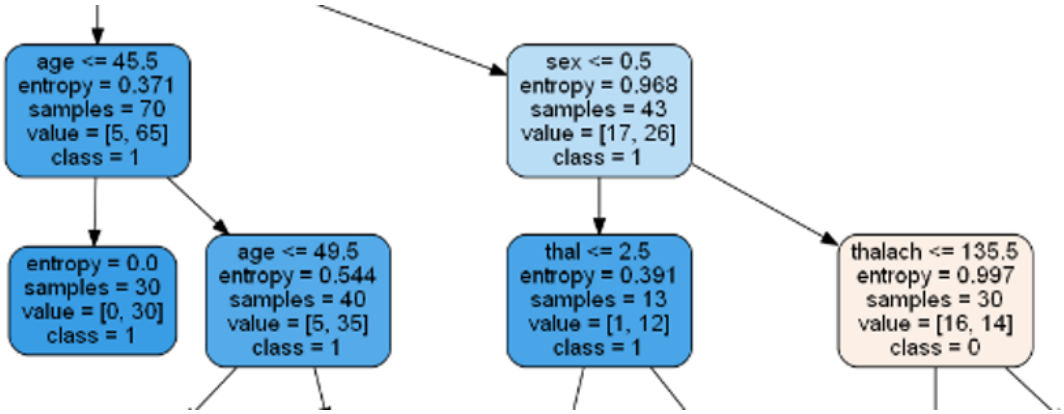


Figure 6.5: ID3 Tree: Level 2-3 Right Child

ID3 Tree: Level 4-5:

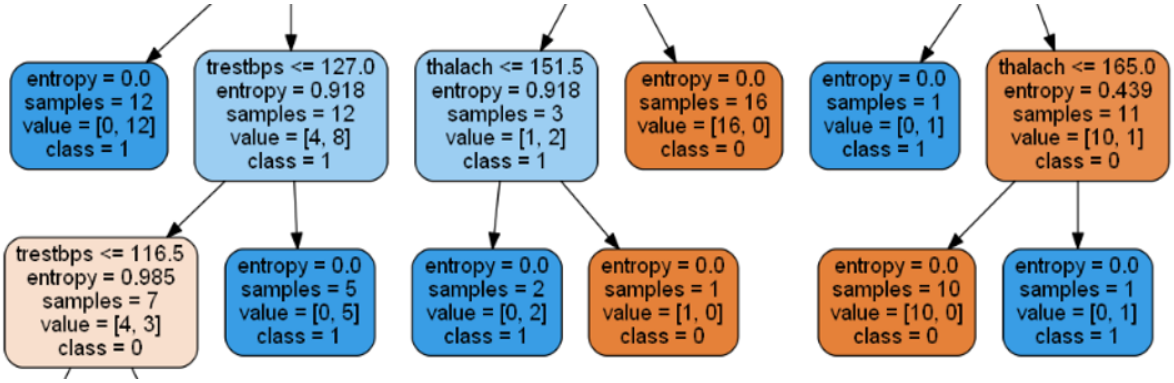


Figure 6.6: ID3 Tree: Level 4-5 Left Child

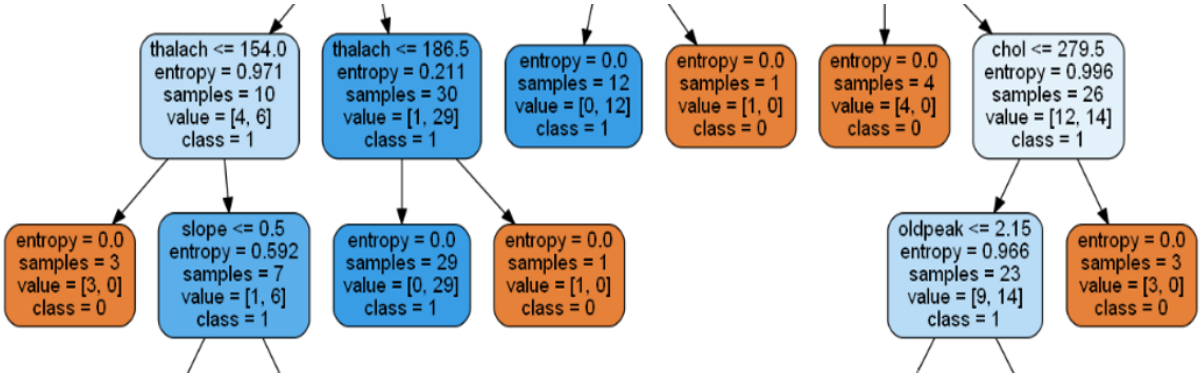


Figure 6.7: ID3 Tree: Level 4-5 Right Child

ID3 Tree: Level 6 Left Child:

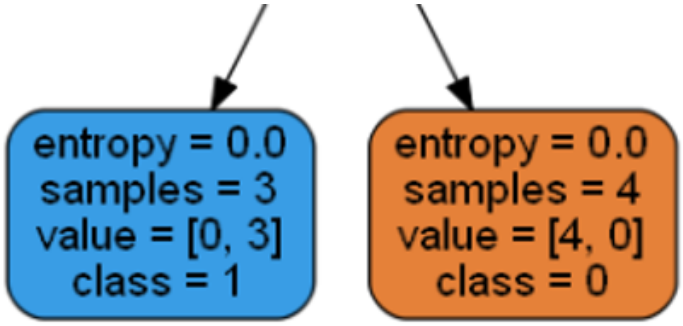


Figure 6.8: ID3 Tree: Level 6 Left Child

ID3 Tree: Level 6-8 Right Child:

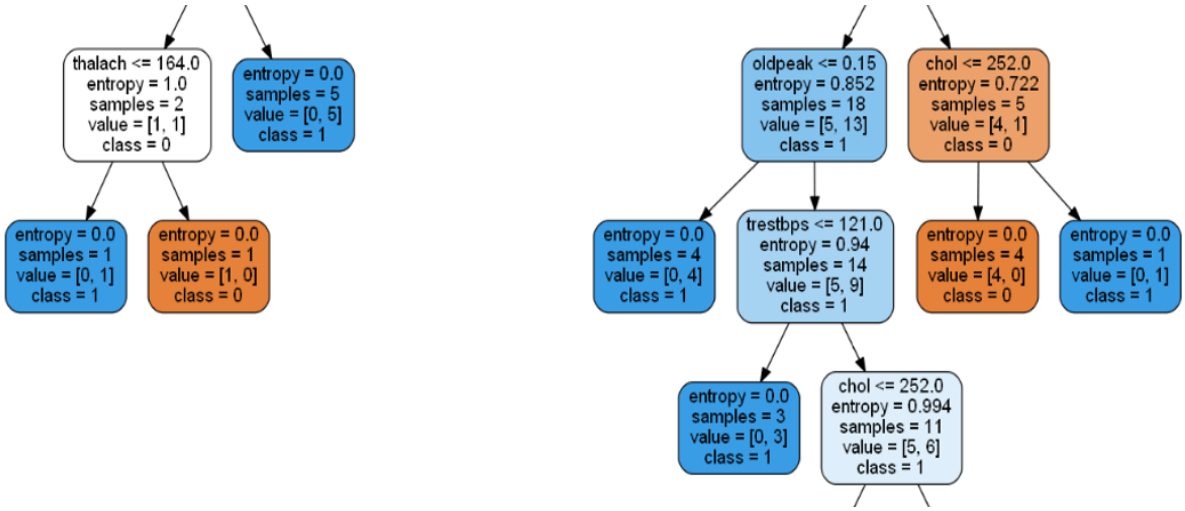


Figure 6.9: ID3 Tree: Level 6-8 Right Child

ID3 Tree: Level 9-12 Right Child:

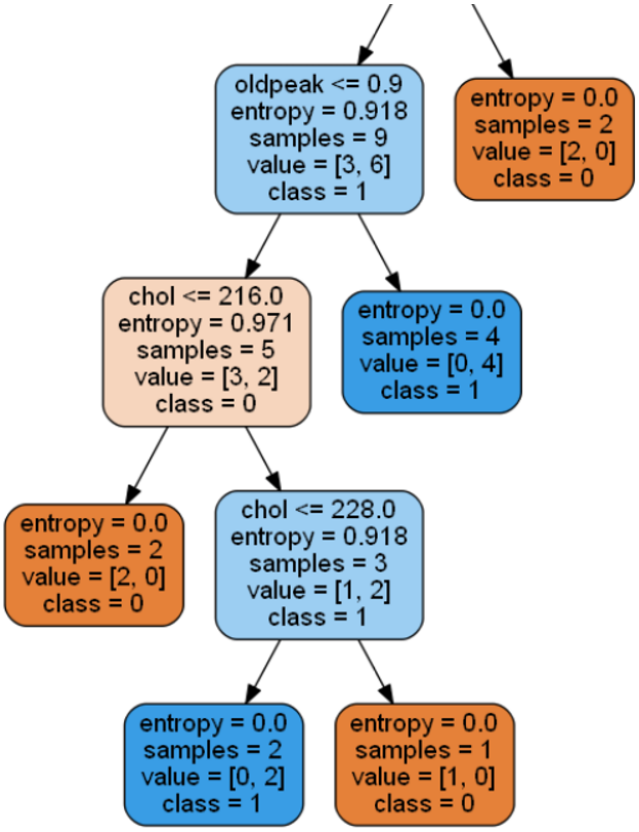


Figure 6.10: ID3 Tree: Level 9-12 Right Child

Accuracy:

The accuracy of ID3 - Built-in is **90.32%.**

Final output of **ID3 - Scratch** algorithm:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	label	classification	classification_correct
204	62	0	0	160	164	0	0	145	0	6.2	0	3	3	0	0.0	True
159	56	1	1	130	221	0	0	163	0	0.0	2	0	3	1	1.0	True
219	48	1	0	130	256	1	0	150	1	0.0	2	2	3	0	0.0	True
174	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0	0.0	True
184	50	1	0	150	243	0	0	128	0	2.6	1	0	3	0	0.0	True
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0	0.0	True
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3	0	0.0	True
119	46	0	0	138	243	0	0	152	1	0.0	1	0	2	1	1.0	True
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3	0	0.0	True
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1	1.0	True
51	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1	0.0	False
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3	0	0.0	True
278	58	0	1	136	319	1	0	152	0	0.0	2	2	2	0	1.0	False
229	64	1	2	125	309	0	1	131	1	1.8	1	0	3	0	0.0	True
208	49	1	2	120	188	0	1	139	0	2.0	1	3	3	0	0.0	True
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0	1.0	False
58	34	1	3	118	182	0	0	174	0	0.0	2	0	2	1	1.0	True
220	63	0	0	150	407	0	0	154	0	4.0	1	3	3	0	0.0	True
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1	1.0	True
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3	0	0.0	True
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1	1.0	True
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0	0.0	True
70	54	1	2	120	258	0	0	147	0	0.4	1	0	3	1	1.0	True
138	57	1	0	110	201	0	1	126	1	1.5	1	0	1	1	0.0	False
122	41	0	2	112	268	0	0	172	1	0.0	2	0	2	1	1.0	True
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1	1.0	True
191	58	1	0	128	216	0	0	131	1	2.2	1	3	3	0	0.0	True
80	41	1	2	112	250	0	1	179	0	0.0	2	0	2	1	1.0	True
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1	1.0	True
123	54	0	2	108	267	0	0	167	0	0.0	2	0	2	1	1.0	True
157	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1	1.0	True

Figure 6.11: ID3 - Scratch Output Matrix

Accuracy:

The accuracy of ID3 - Scratch is **87.09%**.

6.2 CART

Final output of **CART – Built-in** algorithm:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	label	classification	classification_correct
204	62	0	0	160	164	0	0	145	0	6.2	0	3	3	0	0	True
159	56	1	1	130	121	0	0	163	0	0.0	2	0	3	1	1	True
219	48	1	0	130	256	1	0	150	1	0.0	2	2	3	0	0	True
174	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0	0	True
184	50	1	0	150	143	0	0	128	0	2.6	1	0	3	0	0	True
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0	0	True
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3	0	0	True
119	46	0	0	138	243	0	0	152	1	0.0	1	0	2	1	1	True
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3	0	0	True
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1	1	True
51	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1	1	True
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3	0	1	False
278	58	0	1	136	319	1	0	152	0	0.0	2	2	2	0	1	False
229	64	1	2	125	309	0	1	131	1	1.8	1	0	3	0	1	False
208	49	1	2	120	188	0	1	139	0	2.0	1	3	3	0	1	False
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0	1	False
58	34	1	3	118	182	0	0	174	0	0.0	2	0	2	1	1	True
220	63	0	0	150	407	0	0	154	0	4.0	1	3	3	0	0	True
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1	1	True
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3	0	1	False
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1	1	True
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0	0	True
70	54	1	2	120	258	0	0	147	0	0.4	1	0	3	1	1	True
138	57	1	0	110	201	0	1	126	1	1.5	1	0	1	1	1	True
122	41	0	2	112	268	0	0	172	1	0.0	2	0	2	1	1	True
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1	1	True
191	58	1	0	128	216	0	0	131	1	2.2	1	3	3	0	0	True
80	41	1	2	112	250	0	1	179	0	0.0	2	0	2	1	1	True
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1	1	True
123	54	0	2	108	267	0	0	167	0	0.0	2	0	0	1	1	True
157	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1	1	True

Figure 6.12: CART - Built-in Output Matrix

CART Complete Tree:

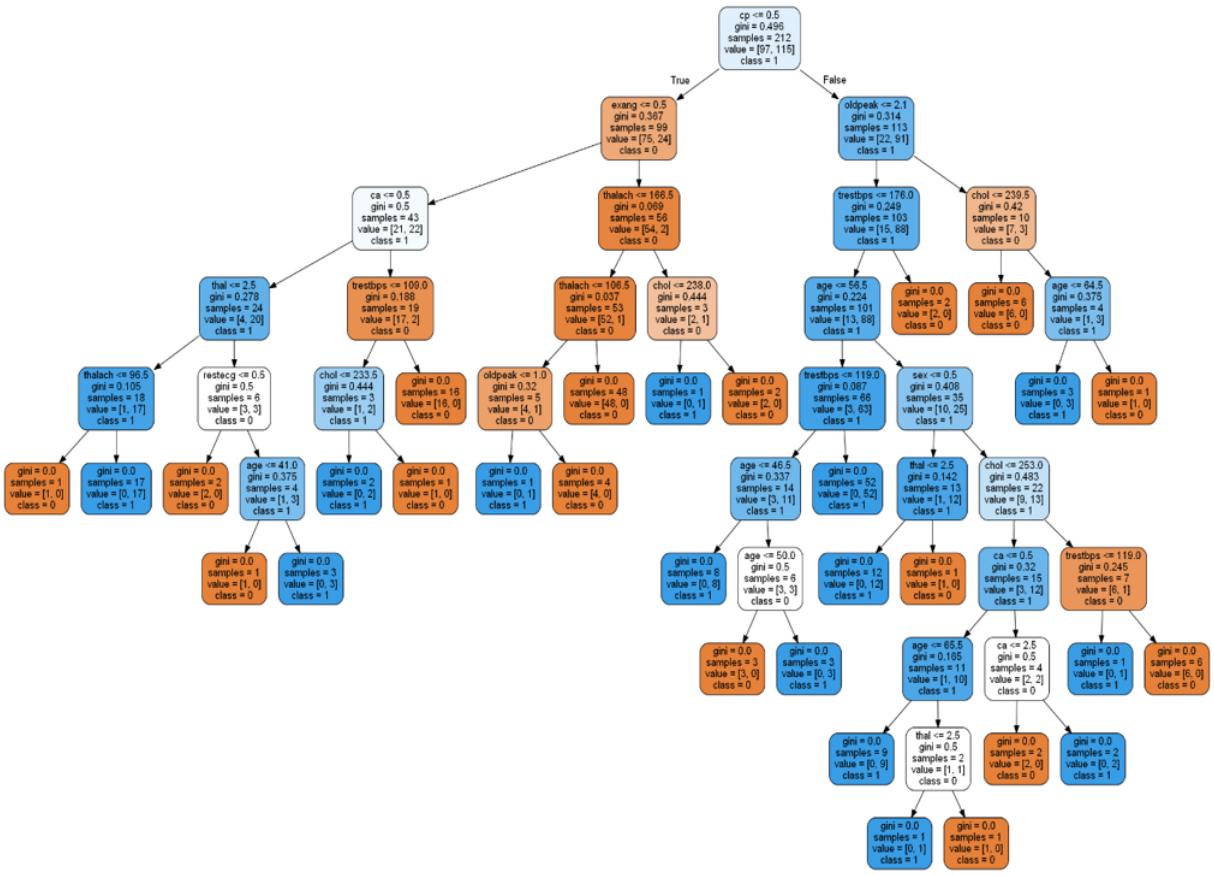


Figure 6.13: CART Tree at Max Level 9

CART Tree: Level 0-1:

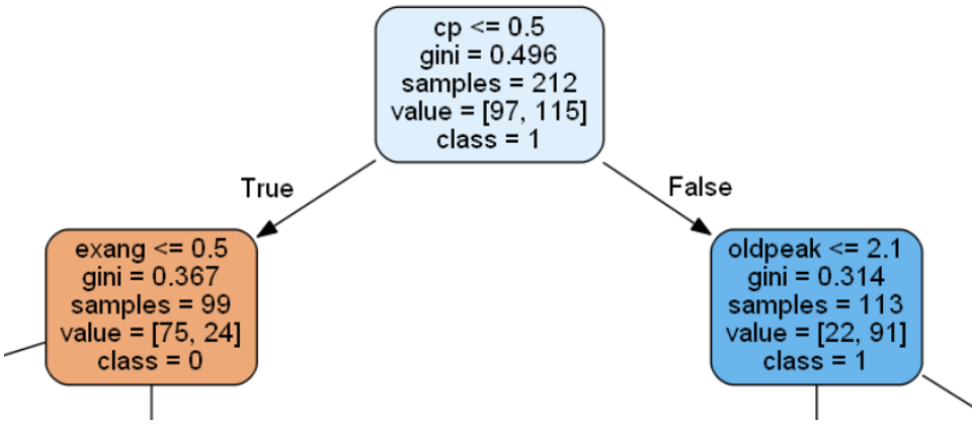


Figure 6.14: CART Tree: Level 0-1

CART Tree: Level 2-3:

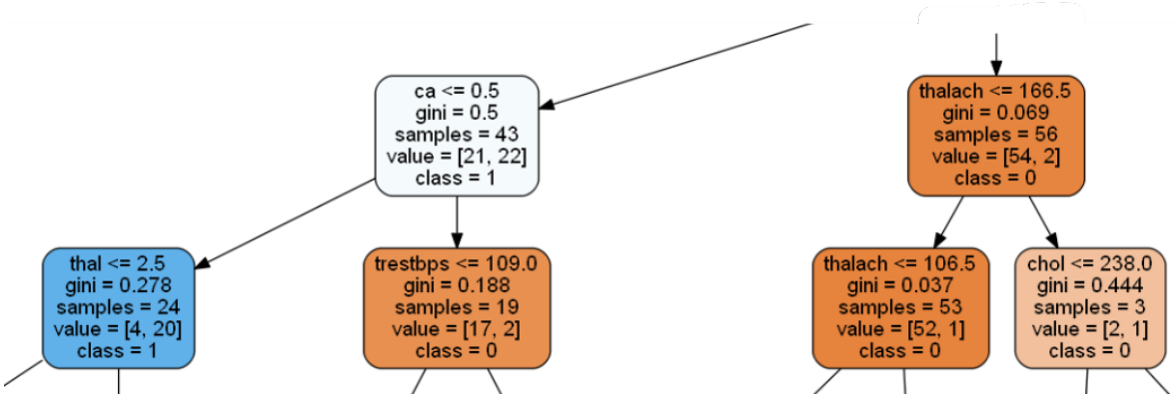


Figure 6.15: CART Tree: Level 2-3 Left Child

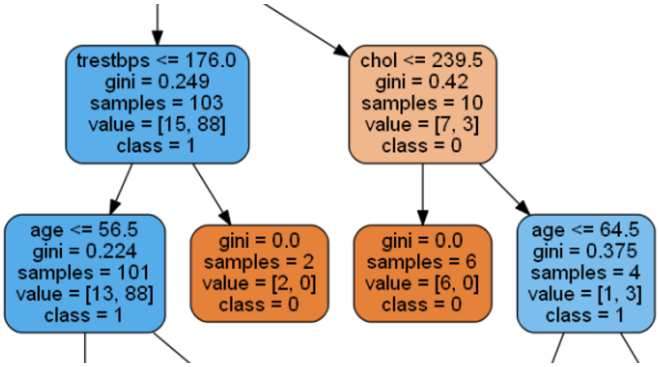


Figure 6.16: CART Tree: Level 2-3 Right Child

CART Tree: Level 4-5:

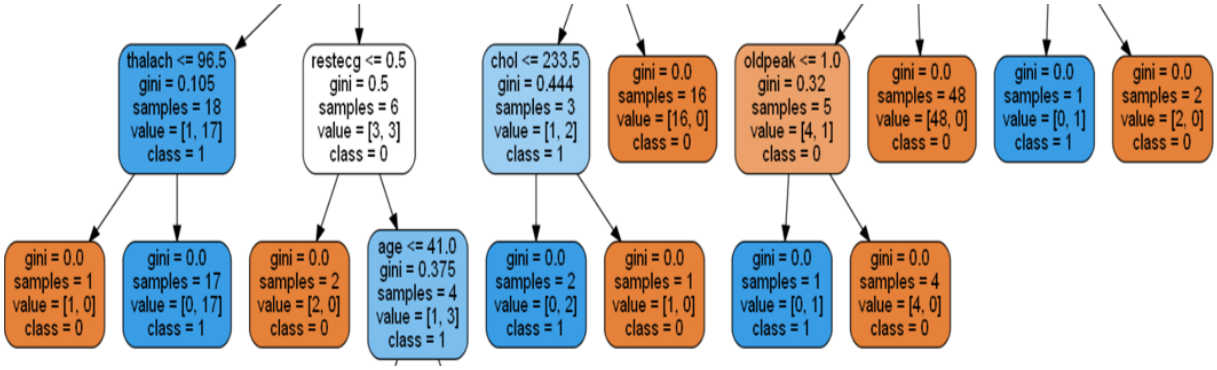


Figure 6.17: CART Tree: Level 4-5 Left Child

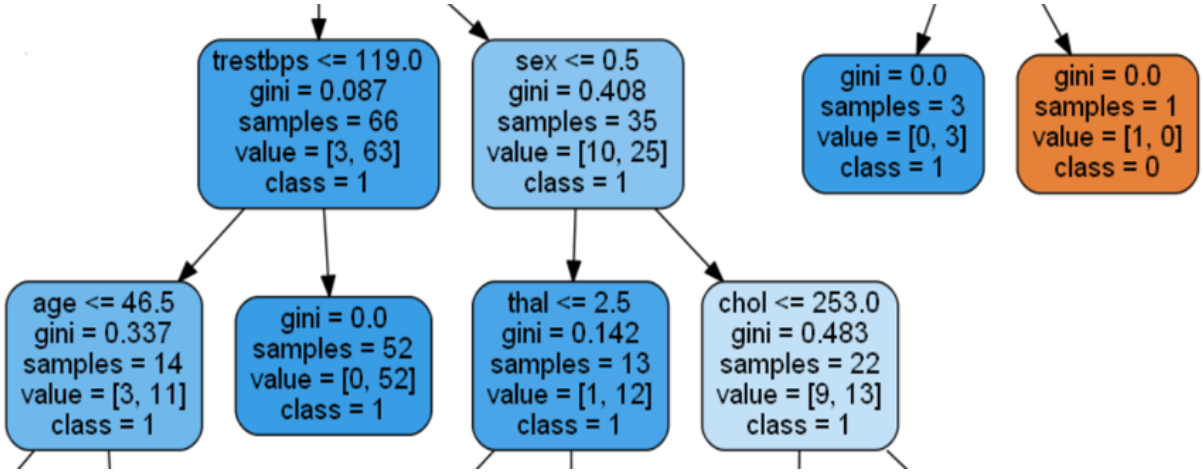


Figure 6.18: CART Tree: Level 4-5 Right Child

CART Tree: Level 6 Left Child:

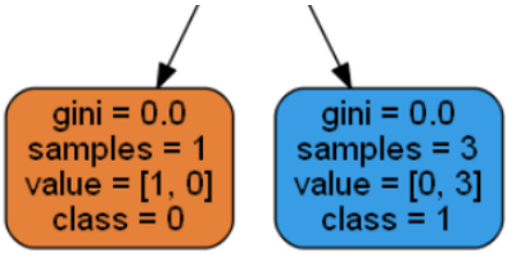


Figure 6.19: CART Tree: Level 6 Left Child

CART Tree: Level 6-7 Right Child:

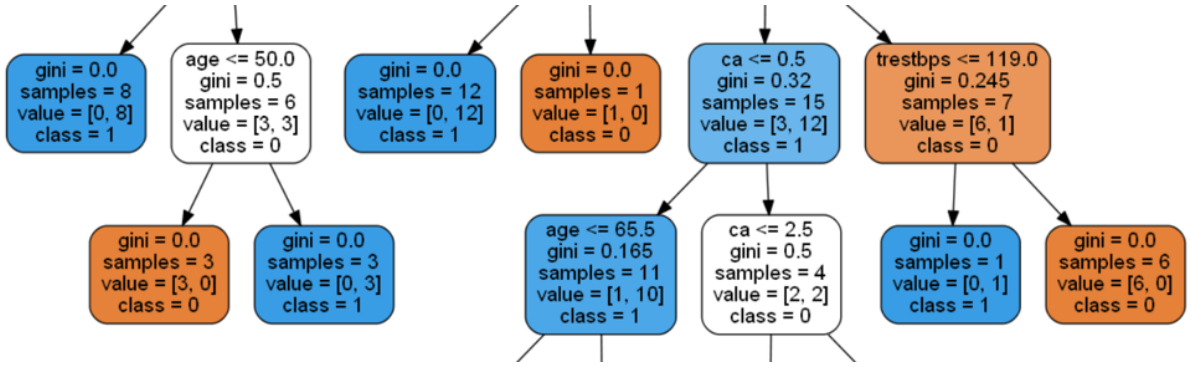


Figure 6.20: CART Tree: Level 6-7 Right Child

CART Tree: Level 8-9 Right Child:

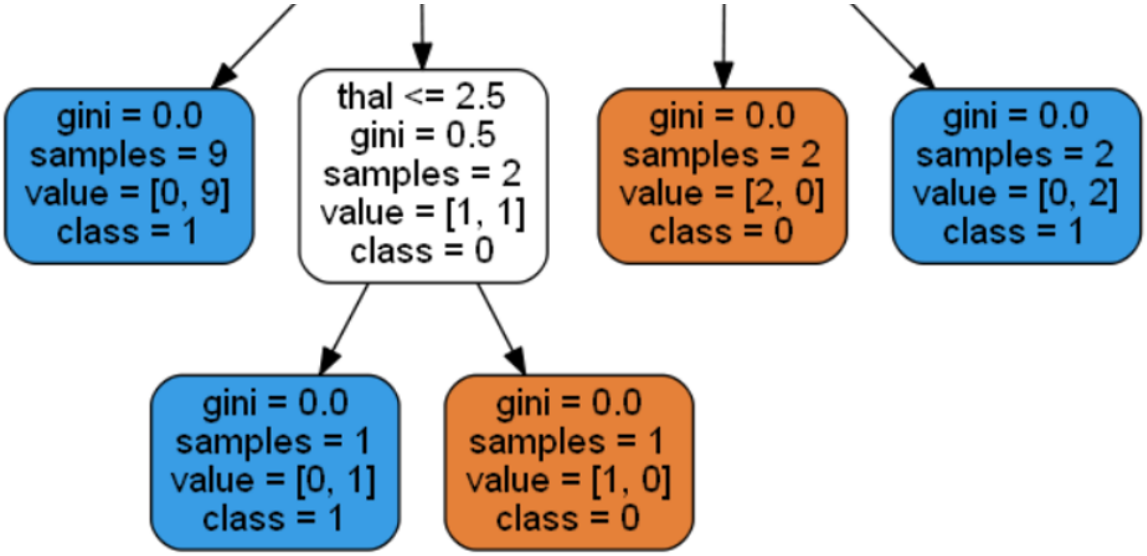


Figure 6.21: CART Tree: Level 8-9 Right Child

Accuracy:

The accuracy of CART - Built-in is **80.64%.**.

Final output of **CART - Scratch** algorithm:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	label	classification	classification_correct
204	62	0	0	160	164	0	0	145	0	6.2	0	3	3	0	0	True
159	56	1	1	130	121	0	0	163	0	0.0	2	0	3	1	1	True
219	48	1	0	130	256	1	0	150	1	0.0	2	2	3	0	1	False
174	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0	1	False
184	50	1	0	150	143	0	0	128	0	2.6	1	0	3	0	0	True
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0	0	True
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3	0	0	True
119	46	0	0	138	243	0	0	152	1	0.0	1	0	2	1	1	True
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3	0	1	False
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1	0	False
51	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1	0	False
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3	0	0	True
278	58	0	1	136	319	1	0	152	0	0.0	2	2	2	0	0	True
229	64	1	2	125	309	0	1	131	1	1.8	1	0	3	0	0	True
208	49	1	2	120	188	0	1	139	0	2.0	1	3	3	0	0	True
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0	1	False
58	34	1	3	118	182	0	0	174	0	0.0	2	0	2	1	1	True
220	63	0	0	150	407	0	0	154	0	4.0	1	3	3	0	0	True
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1	1	True
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3	0	0	True
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1	0	False
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0	0	True
70	54	1	2	120	258	0	0	147	0	0.4	1	0	3	1	0	False
138	57	1	0	110	201	0	1	126	1	1.5	1	0	1	1	1	True
122	41	0	2	112	268	0	0	172	1	0.0	2	0	2	1	1	True
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1	1	True
191	58	1	0	128	216	0	0	131	1	2.2	1	3	3	0	0	True
80	41	1	2	112	250	0	1	179	0	0.0	2	0	2	1	1	True
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1	1	True
123	54	0	2	108	267	0	0	167	0	0.0	2	0	0	1	1	True
157	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1	1	True

Figure 6.22: CART - Scratch Output Matrix

Accuracy:

The accuracy of CART - Scratch is **74.19%.**.

6.3 Random Forests

Final output of **Random Forest – Built-in** algorithm:

	tree_0	tree_1	tree_2	tree_3	tree_4	tree_5	tree_6	Label	Classification	Classification_correct
204	1	1	1	1	1	1	1	0	1	False
159	1	0	0	0	1	1	1	1	1	True
219	0	0	0	1	1	1	1	0	1	False
174	0	0	0	0	0	0	1	0	0	True
184	1	1	1	0	0	0	0	0	0	True
295	1	1	1	0	0	0	0	0	0	True
269	0	0	0	0	1	0	0	0	0	True
119	1	1	1	1	1	1	0	1	1	True
193	0	0	0	0	1	1	1	0	0	True
154	1	1	0	1	0	1	0	1	1	True
51	1	0	1	1	1	0	1	1	1	True
249	0	1	0	1	0	1	0	0	0	True
278	0	0	0	1	1	0	0	0	0	True
229	1	0	0	0	0	0	0	0	0	True
208	1	1	1	0	0	0	0	0	0	True
302	0	1	1	1	0	0	0	0	0	True
58	0	1	1	1	1	1	1	1	1	True
220	0	1	0	0	0	0	0	0	0	True
18	0	1	1	1	1	1	1	1	1	True
228	0	0	0	0	0	0	0	0	0	True
11	1	1	1	1	1	1	1	1	1	True
300	1	1	0	0	0	1	0	0	0	True
70	1	1	1	1	1	1	0	1	1	True
138	0	0	0	1	0	0	0	1	1	True
122	1	1	1	1	1	1	1	1	1	True
164	0	1	0	0	1	1	1	1	1	True
191	1	0	0	0	0	0	0	0	0	True
80	1	0	1	1	1	0	1	1	1	True
27	1	1	0	1	1	1	1	1	1	True
123	1	1	0	1	1	1	1	1	1	True
157	0	1	0	1	1	1	1	1	1	True

Figure 6.23: Random Forest - Built-in Output Matrix

Accuracy:

The accuracy of Random Forest built-in is **93.54%**.

Final output of **Random Forest – Scratch** algorithm:

	tree_0	tree_1	tree_2	tree_3	tree_4	tree_5	tree_6	Label	Classification	Classification_correct
204	0	0	0	0	0	0	0	0	0	True
159	1	1	1	1	0	0	1	1	1	True
219	0	1	1	0	0	0	0	0	0	True
174	0	1	0	0	0	0	0	0	0	True
184	1	1	1	1	1	1	1	0	1	False
295	0	0	0	0	0	1	1	0	0	True
269	1	0	0	0	0	1	0	0	0	True
119	0	0	0	0	0	1	1	1	0	False
193	1	1	1	1	0	1	0	0	1	False
154	1	1	1	1	0	1	1	1	1	True
51	0	1	1	1	1	1	1	1	1	True
249	0	0	0	0	0	0	1	0	0	True
278	1	1	1	1	0	1	0	0	1	False
229	0	0	0	1	1	0	0	0	0	True
208	1	1	1	1	0	1	1	0	1	False
302	1	0	0	0	1	0	0	0	0	True
58	1	1	1	1	1	1	1	1	1	True
220	1	0	0	0	0	0	0	0	0	True
18	1	0	1	1	1	1	1	1	1	True
228	1	1	0	0	0	0	0	0	0	True
11	1	0	1	1	1	1	0	1	1	True
300	1	1	1	1	1	1	1	0	1	False
70	1	1	1	1	1	0	0	1	1	True
138	0	1	0	1	1	1	1	1	1	True
122	0	1	1	1	1	1	1	1	1	True
164	1	1	1	0	1	1	1	1	1	True
191	1	0	1	0	0	0	0	0	0	True
80	1	1	1	1	1	1	1	1	1	True
27	1	0	1	1	0	0	1	1	0	False
123	1	1	1	1	1	0	1	1	1	True
157	1	1	0	1	1	1	1	1	1	True

Figure 6.24: Random Forest - Scratch Output Matrix

Accuracy:

The accuracy of Random Forest Scratch is **77.41%**.

Accuracy Comparison:

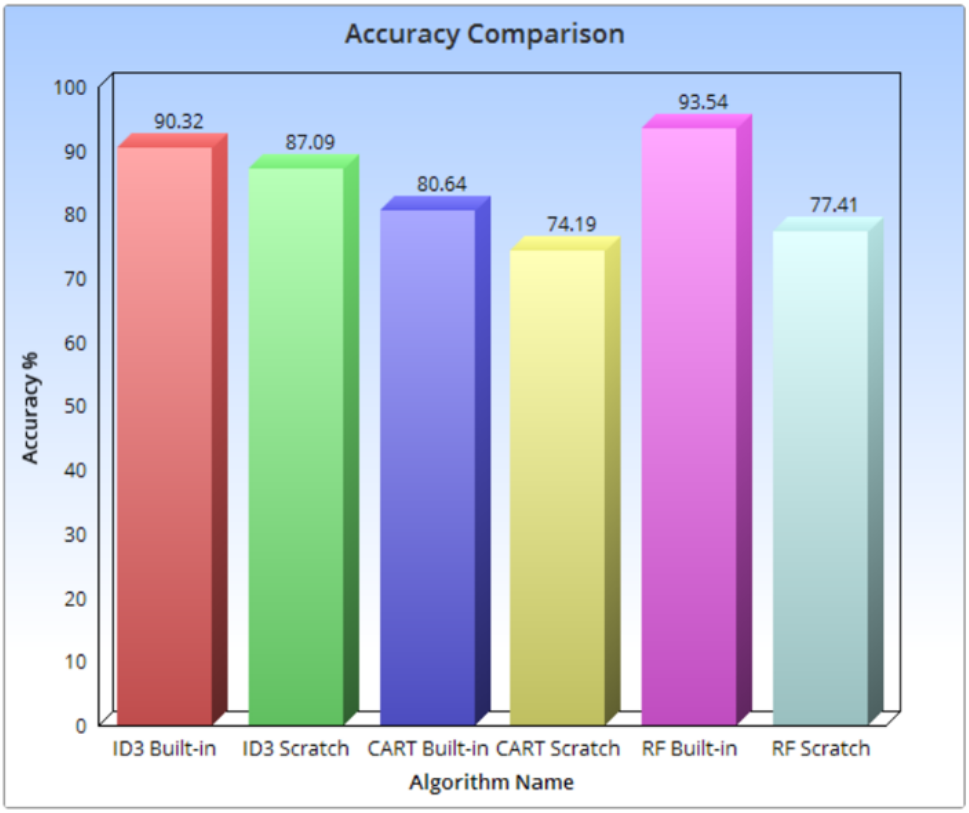


Figure 6.25: Accuracy Comparison Graph

Table 6.1: Performance of Different Algorithms

Algorithm	**ID3 (Built-in)**	**ID3 (Scratch)**	**CART (Built-in)**	**CART (Scratch)**	**Random Forest (Built-in)**	**Random Forest (Scratch)**
Accuracy	90.32%	87.09%	80.64%	74.19%	93.54%	77.41%
Data Selected	Training Data = 272, Testing Data = 31					

From the tables above we gather that algorithms performed depending upon the situation whether cross validation and feature selection were used or not. Every algorithm has its intrinsic capacity to out-perform other algorithms in certain situation. For example, **Random Forest – Built-in** performed much better (**93.54%** accuracy) in our chosen environment, and the performance of Decision Tree, in particular, **ID3 – Built-in** was comparable to it (90.32% accuracy). Overall, performance of Random Forests, as was expected, is comparatively better.

Chapter 7

Discussion and Conclusion

7.1 The Problem and Our Efforts

Thorough study of a few machine learning algorithms - there mechanism and how they perform on a typical set of data was the problem set forth for our thesis work. The algorithms chosen were commonly known as decision tree induction algorithms, and the data domain was heart diseases. We have used Cleveland dataset for heart diseases which contains 303 instances and divided the data into two sections, which were training and testing datasets. We have considered 13 attributes and implemented different algorithms to analyze their accuracy. The well-known Decision Tree algorithms, namely, ID3 and CART were taken for study. Besides, a Random Forest algorithm was also chosen. To get a perfect understanding, we went for implementation of ID3 and Random Forest algorithms from the scratch also, that is, without using available built-in functions. Ultimately, we have come out with five algorithms implemented. We compared the performances and have found that the Random Forests have performed comparatively better, as was expected. The performance of the algorithms implemented from scratch was also comparable with that of the implementations with built-in functions. In our work we have tried to compare different machine learning algorithms that might be used to predict whether a certain person with particular personal characteristics and symptoms has developed heart diseases or not. The main motive of our report was to demonstrate the working of the chosen algorithms and compare the accuracy of them while working on a common dataset. We also tried to analyze the reasons behind the variation in performance of different algorithms. Probably, for other datasets or other instances the algorithms will give a different performance. Moreover, if we would have increased the number of attributes, we would have got more accurate result, although it would take more time to process.

7.2 Issues and Challenges

Medical diagnosis is considered to be a significant yet intricate task that needs to be carried out precisely and efficiently. The automation of the same would be highly beneficial. Clinical decisions are often made based on doctor's intuition and experience rather than on the knowledge rich data hidden in the databases. This practice leads to unwanted biases, errors and excessive medical costs which affects the quality of service provided to patients. Data mining have the potential to generate a knowledge-rich environment which can help to significantly improve the quality of clinical decisions.

7.3 Future Work

For some very practical reasons related to academic load we were not able to work with big datasets till now, and we were confined to implementing and getting a good understanding of the chosen algorithms only. But it is a necessity to check the performance of the algorithms with those. So, we have plan to work with larger volumes of data from different medical databases in future. And further, we plan to find a better algorithmic solution with some other extended implementations of the data mining, like SVM, KNN, etc.

References

[1] H Sharma and S Kumar, "A survey on decision tree algorithms of classification in data mining," International Journal of Science and Research (IJSR), 5(4): 2094-2097, April 2016. [Online; accessed:25 January,2019].

[2] M. J Han and J Pei, "Data mining: Concepts and techniques," 3rd Edition, Morgan Kaufmann, 2011.

[3] A. Anbarasi M and Iyengar N, "Enhanced prediction of heart disease with feature subset selection using genetic algorithm," International Journal of Engineering Science and Technology, 2(10):5370-76, 2010. [Online; accessed:20 February,2019].

[4] C S Dangare and S S Apte, "Improved study of heart disease prediction system using data mining classification techniques," International Journal of Computer Applications, 47(10): 46-48, June 2012. [Online; accessed:25 February,2019].

[5] C Ordonez, "Association rule discovery with the train and test approach for heart disease prediction," IEEE Transactions on Information Technology in Biomedicine, 10(2), April 2006. [Online; accessed:22 February,2019].

[6] S. B N Patel and K I Lakhtaria, "Efficient classification of data using decision tree," Bonfring International Journal of Data Mining, 2(1): 6-12, March 2012. [Online; accessed:29 January,2019].

[7] M Last and O. Maimon, "A compact and accurate model for classification," IEEE Transactions on Knowledge and Data Engineering,16(2): 203-215, February 2004. [Online; accessed:19 February,2019].

[8] M. Franck Le Duff, Cristian Munteanb and Philippe Mabob, "Predicting survival causes after out of hospital cardiac arrest using data mining method," Studies in health technology and informatics, Vol. 107, No. Pt 2, 2004. [Online; accessed:29 February,2019].

[9] Sellappan Palaniappan and Rafiah Awang, "Intelligent heart disease prediction system using data mining techniques," Proceedings of the 2008 IEEE/ACS International Con-

ference on Computer Systems and Applications, March 2008 Pages 108–115. [Online; accessed:29 February,2019].

[10] "Random forests algorithms and hyperparameter." https://towardsdatascience.com/hyperparameter/randomforest_algorithm. [Online; accessed:31 December,2019].

[11] "Decision tree learning in machine learning." https://s3-ap-southeast-1.amazonaws.com/83475897894335. [Online; accessed:30 November,2019].

[12] "Deep machine learning with random forests." https://medium.com/deep-math-machine-learning_random_forest. [Online; accessed:31 December,2019].

[13] "What is a decision tree? how it works?." https://hackernoon.com/what_is_a_decision-tree_example. [Online; accessed:12 December,2019].

[14] "Id3 algorithm details understanding." https://upload.wikimedia.org/wikipedia/id3_decision_algorithm. [Online; accessed:30 November,2019].

[15] "Random forests algorithm simple example with solutions." https://towardsdatascience.com/random_forest_steps_math/example. [Online; accessed:5 November,2019].

[16] "Id3 algorithm simple example with solutions." https://sefiks.com/2017/id3_algorithm_steps_math/example. [Online; accessed:5 November,2019].

[17] "Heart disease dataset with 76 attributes." https://archive.ics.uci.edu/datasets/heart_disease/heart.csv. [Online; accessed:12 August,2019].

[18] "Decision tree algorithms example explained with steps." https://towardsdatascience.com/decision_tree_algorithms_example. [Online; accessed:31 December,2019].

[19] "Basic python with example and sudo code." https://www.python.org/8343598579847433333/about/basic_python. [Online; accessed:12 August,2019].

[20] "Machine learning numpy example." http://www.numpy.org/9473895733545643/basic/introduction.html. [Online; accessed:9 August,2019].

[21] "Pandas pydata working process in machine learning." http://pandas.pydata.org/002344846330/introduction.html. [Online; accessed:9 November,2019].

[22] "Seaborn pydata working process in machine learning." http://seaborn.pydata.org/t049670496674/introduction.html. [Online; accessed:9 November,2019].

[23] "Scikit-learn example in machine learning." http://jmlr.csail.mit.edu/papers/work/pedregosa11a.html. [Online; accessed:9 November,2019].

[24] "Graphviz working process in machine learning visualization." https://www.graphviz.org/. [Online; accessed:20 November,2019].

[25] "Random forests basic steps and images." https://www.scirp.org/87843303858956644398 4. [Online; accessed:20 November,2019].

Appendix A

Code snippets of ID3

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import random
from pprint import pprint

df=pd.read_csv("heart.csv")
df.head()
df=df.rename(columns={"target":"label"})
df.info()

def train_test_split(df,test_size):

    if isinstance(test_size,float):
        test_size=round(test_size * len(df))

    indices=df.index.tolist()
    test_indices=random.sample(population=indices,k=test_size)
    print(type(test_indices))

    test_df=df.loc[test_indices]
    train_df=df.drop(test_indices)

    return train_df,test_df

random.seed(0)
train_df,test_df=train_test_split(df,test_size=23)

```

```
def check_purity(data):

    label_columns=data[:,-1]
    unique_classes=np.unique(label_columns)
    if len(unique_classes) == 1:
        return True
    else:
        return False

def classify_data(data):

    label_columns=data[:,-1]
    unique_classes,counts_unique_classes=np.unique(label_columns,
   return_counts=True)
    index=counts_unique_classes.argmax()
    classification=unique_classes[index]

    return classification

def get_potential_splits(data):

    potential_splits={}
    _,n_columns=data.shape
    for column_index in range(n_columns-1):
        potential_splits[column_index]=[]
        values=data[:,column_index]
        unique_values=np.unique(values)

        for index in range(len(unique_values)):
            if index != 0:
                current_value=unique_values[index]
                previous_value=unique_values[index-1]
                potential_split=(current_value+previous_value)/2

                potential_splits[column_index].append(potential_split
   )

    return potential_splits

def split_data(data,split_column,split_value):

    split_column_values=data[:,split_column]

    data_below=data[split_column_values <= split_value]
    data_above=data[split_column_values > split_value]

    return data_below,data_above
```

```
78
79  def calculate_entropy(data):
80
81      label_column = data[:, -1]
82      _, counts = np.unique(label_column, return_counts=True)
83
84      probabilities = counts / counts.sum()
85      entropy = sum(probabilities * -np.log2(probabilities))
86
87      return entropy
88
89  def calculate_overall_entropy(data_below, data_above):
90
91      n = len(data_below) + len(data_above)
92      p_data_below = len(data_below) / n
93      p_data_above = len(data_above) / n
94
95      overall_entropy =  (p_data_below * calculate_entropy(data_below)
96                        + p_data_above * calculate_entropy(data_above))
97
98      return overall_entropy
99
100 def determine_best_split(data, potential_splits):
101
102     overall_entropy = 9999
103     for column_index in potential_splits:
104         for value in potential_splits[column_index]:
105             data_below, data_above = split_data(data, split_column=
    column_index, split_value=value)
106             current_overall_entropy = calculate_overall_entropy(
    data_below, data_above)
107
108             if current_overall_entropy <= overall_entropy:
109                 overall_entropy = current_overall_entropy
110                 best_split_column = column_index
111                 best_split_value = value
112
113     return best_split_column, best_split_value
114
115 def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth
    =7):
116
117     if counter == 0:
118         global COLUMN_HEADERS #, FEATURE_TYPES
119         COLUMN_HEADERS = df.columns
120         #FEATURE_TYPES = determine_type_of_feature(df)
121         data = df.values
```

```
    else:
        data = df

    if (check_purity(data)) or (len(data) < min_samples) or (counter
== max_depth):
        classification = classify_data(data)

        return classification

    else:
        counter += 1
        potential_splits = get_potential_splits(data)
        split_column, split_value = determine_best_split(data,
potential_splits)
        data_below, data_above = split_data(data, split_column,
split_value)

        feature_name = COLUMN_HEADERS[split_column]
        question = "{} <= {}".format(feature_name, split_value)
        sub_tree = {question: []}

        yes_answer = decision_tree_algorithm(data_below, counter,
min_samples, max_depth)
        no_answer = decision_tree_algorithm(data_above, counter,
min_samples, max_depth)

        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)

        return sub_tree

tree=decision_tree_algorithm(train_df, max_depth=9)
pprint(tree)

def classify_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")

    if example[feature_name] <= float(value):
        answer = tree[question][0]
    else:
        answer = tree[question][1]

    if not isinstance(answer, dict):
```

```
        return answer

    else:
        residual_tree = answer
        return classify_example(example, residual_tree)

classify_example(example, tree)

def calculate_accuracy(df, tree):

    df["classification"] = df.apply(classify_example, args=(tree,), axis=1)
    df["classification_correct"] = df["classification"] == df["label"]

    accuracy = df["classification_correct"].mean()

    return accuracy

accuracy = calculate_accuracy(test_df, tree)
accuracy
test_df
```

Appendix B

Code snippets of CART

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import pydotplus

df = pd.read_csv("heart.csv")
df.head()
df.describe()

feature_cols=['age','sex','cp','trestbps','chol','fbs','restecg','
    thalach','exang','oldpeak','slope','ca','thal']
X=df[feature_cols]
Y=df.target

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,
    random_state=1)
clf = DecisionTreeClassifier(criterion="gini")
clf = clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
dtree_cm_id3=confusion_matrix(Y_test,Y_pred)

print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('ID3_Built-In.png')
Image(graph.create_png())
```

Appendix C

Code snippets of Random Forests

```
import numpy as np
import pandas as pd
import random

from helper_functions import determine_type_of_feature

def check_purity(data):

    label_column = data[:, -1]
    unique_classes = np.unique(label_column)

    if len(unique_classes) == 1:
        return True
    else:
        return False


def classify_data(data):

    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column,
   return_counts=True)

    index = counts_unique_classes.argmax()
    classification = unique_classes[index]

    return classification


def get_potential_splits(data, random_subspace):

    potential_splits = {}
```

```
    _, n_columns = data.shape
    column_indices = list(range(n_columns - 1))

    if random_subspace and random_subspace <= len(column_indices):
        column_indices = random.sample(population=column_indices, k=
random_subspace)

    for column_index in column_indices:
        values = data[:, column_index]
        unique_values = np.unique(values)

        potential_splits[column_index] = unique_values

    return potential_splits


def calculate_entropy(data):

    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy


def calculate_overall_entropy(data_below, data_above):

    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_entropy =  (p_data_below * calculate_entropy(data_below)
                      + p_data_above * calculate_entropy(data_above))

    return overall_entropy

def determine_best_split(data, potential_splits):

    overall_entropy = 9999
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=
column_index, split_value=value)
            current_overall_entropy = calculate_overall_entropy(
data_below, data_above)
```

```
76
77              if current_overall_entropy <= overall_entropy:
78                  overall_entropy = current_overall_entropy
79                  best_split_column = column_index
80                  best_split_value = value
81
82      return best_split_column, best_split_value
83
84
85  def split_data(data, split_column, split_value):
86
87      split_column_values = data[:, split_column]
88
89      type_of_feature = FEATURE_TYPES[split_column]
90      if type_of_feature == "continuous":
91          data_below = data[split_column_values <= split_value]
92          data_above = data[split_column_values >  split_value]
93
94      else:
95          data_below = data[split_column_values == split_value]
96          data_above = data[split_column_values != split_value]
97
98      return data_below, data_above
99
100 def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth
    =5, random_subspace=None):
101
102     # data preparations
103     if counter == 0:
104         global COLUMN_HEADERS, FEATURE_TYPES
105         COLUMN_HEADERS = df.columns
106         FEATURE_TYPES = determine_type_of_feature(df)
107         data = df.values
108     else:
109         data = df
110
111     if (check_purity(data)) or (len(data) < min_samples) or (counter
    == max_depth):
112         classification = classify_data(data)
113
114         return classification
115
116     else:
117         counter += 1
118
119         potential_splits = get_potential_splits(data, random_subspace
    )
```

```
120         split_column, split_value = determine_best_split(data,
    potential_splits)
121         data_below, data_above = split_data(data, split_column,
    split_value)
122
123
124         if len(data_below) == 0 or len(data_above) == 0:
125             classification = classify_data(data)
126             return classification
127
128         feature_name = COLUMN_HEADERS[split_column]
129         type_of_feature = FEATURE_TYPES[split_column]
130         if type_of_feature == "continuous":
131             question = "{} <= {}".format(feature_name, split_value)
132
133
134         else:
135             question = "{} = {}".format(feature_name, split_value)
136
137
138         sub_tree = {question: []}
139
140         yes_answer = decision_tree_algorithm(data_below, counter,
    min_samples, max_depth, random_subspace)
141         no_answer = decision_tree_algorithm(data_above, counter,
    min_samples, max_depth, random_subspace)
142
143
144         if yes_answer == no_answer:
145             sub_tree = yes_answer
146         else:
147             sub_tree[question].append(yes_answer)
148             sub_tree[question].append(no_answer)
149
150         return sub_tree
151
152 def predict_example(example, tree):
153     question = list(tree.keys())[0]
154     feature_name, comparison_operator, value = question.split(" ")
155
156
157     if comparison_operator == "<=":
158         if example[feature_name] <= float(value):
159             answer = tree[question][0]
160         else:
161             answer = tree[question][1]
162     else:
```

```
163         if str(example[feature_name]) == value:
164             answer = tree[question][0]
165         else:
166             answer = tree[question][1]
167 
168 
169     if not isinstance(answer, dict):
170         return answer
171 
172     else:
173         residual_tree = answer
174         return predict_example(example, residual_tree)
175 
176 def decision_tree_predictions(test_df, tree):
177     predictions = test_df.apply(predict_example, args=(tree,), axis
    =1)
178     return predictions
```

Generated using Undegraduate Thesis LaTeX Template, Version 2.0. Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh.

This thesis was generated on Friday 24th January, 2020 at 5:30am.