Decision Tree Classification on Heart Disease Dataset (July 2023)

Anuj Rayamajhi[1], UG, Aayush Regmi[1], UG. and Editor Arun Subedi[1], UG
[1]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

ABSTRACT Decision tree is a widely used machine learning algorithm that provides a simple and interpretable approach for solving classification problems. This algorithm constructs a tree-like model by recursively partitioning the input space based on the values of the input features. Each partition in the tree represents a decision rule that assigns a class label to the corresponding subset of the data. The algorithm begins by selecting the most informative feature to split the data, aiming to maximize the separation between different classes. It continues to recursively partition the data until a stopping criterion is met, such as reaching a maximum tree depth or a minimum number of samples per leaf node. The resulting decision tree can be interpreted as a set of if-else rules that guide the classification of unseen instances. This interpretability is one of the significant advantages of decision trees, as it allows users to understand the decision-making process and gain insights into the relationships between the input features and the target variable. To evaluate the performance of a decision tree model, various metrics such as accuracy, precision, recall, and F1 score are commonly used. Additionally, techniques such as cross-validation and pruning can help prevent overfitting and improve generalization capabilities.

KEYWORDS Decision Tree, f1-score , precision, recall, pruning

I. INTRODUCTION

Machine learning is a rapidly emerging field having applications in a wide range of businesses today. Machine learning algorithms can learn from data and anticipate the future without being explicitly programmed. As a result, they are ideal for jobs that are difficult or impossible to automate using traditional programming methods.

Many factors have contributed to the rise of machine learning in recent years. The rising availability of data is one factor. The amount of data collected and saved is increasing at an exponential rate, providing machine learning algorithms with the data they require to learn and develop.

The emergence of more powerful computing platforms has also contributed to the advancement of machine learning. These systems enable machine learning algorithms to be trained on big datasets and provide real-time predictions. Machine learning is now being applied in a variety of industries as a result of these characteristics. A decision tree is an example of a machine learning algorithm. Decision Trees are supervised learning algorithms that use previous data/occurrence knowledge to categorize or predict on new data.

In this study, our main focus is on using decision trees for classification problem. We are going to explore the structure of a decision tree and how it works. The heart disease dataset will be used for the classification while also exploring how the nodes are splitted and the criteria for splitting the node. Through this study, we aim to gain insights on working of decision trees and how we can maximize the performance of the decision trees by finding suitable hyperparameters.

The primary focus of this research is on using decision trees to solve classification problems. We'll look at the structure and operation of a decision tree. The heart disease dataset will be utilized for classification while simultaneously investigating how nodes are split and the criteria for node splitting. We hope to learn more about how decision trees work and how to

improve their effectiveness by identifying appropriate hyperparameters through this research aims to provide valuable insights into the practical applications of PCA as a dimensionality reduction technique in various classification tasks, including breast cancer classification.

II. METHODOLOGY

A. THEORY

Decision trees are a sort of supervised learning technique that can be used for classification as well as regression. They operate by recursively separating the data into smaller and smaller subgroups until each subset is homogenous (all data points in the subset belong to the same class) or some halting requirement is fulfilled. It classifies fresh data based on how the prior set of data was classified. A decision rule, which is a mathematical formula that dictates how the data should be split, guides the splitting process.

Typically, the decision rule is set to maximize information gain, which is a measure of how much knowledge is acquired by separating the data. The purpose of decision tree learning is to discover a decision rule that optimizes information gain at each split, resulting in a tree that predicts the output variable accurately for new data points.

Decision trees are a powerful tool for modeling and predicting outcomes in many fields, including business, finance, healthcare, and more. They are relatively easy to understand and interpret, making them a popular choice for business intelligence applications.

Commonly used terminologies in decision tree

- Root Node: The base of the decision tree
- Splitting: Dividing a node into two or more sub-nodes
- Leaf/Terminal Node: Node of the tree that do not have sub-nodes

Steps involved in creating and predicting from decision tree:

- Choose the splitting criteria: The first step is to choose the splitting criteria. This is the measure that will be used to decide how to split the data. The most common splitting criteria are Gini impurity and entropy.
- Split the data: Once the splitting criteria has been chosen, the data is split into two or more subsets. The subset that is split is the one that has the highest value for the splitting criteria.
- Repeat steps 1 and 2: The process of splitting the data is repeated until the desired level of granularity is reached. In other words, the process is repeated until each subset is homogeneous (i.e., all the data points in the subset belong to the same class).
- Assign a label to each leaf node: Once the data has been split into a set of homogeneous subsets, a label is assigned to each leaf node. The label is the most common class of the data points in the leaf node.
- Use the tree to make predictions: Once the tree has been built, it can be used to make predictions. To make a prediction, the tree is traversed from the root node to a leaf node. The label of the leaf node is the predicted class for the data point.

Splitting Criterion

- Information Gain: Information gain is the splitting criterion used in decision trees to determine the best way to split a node. Measures how much information a function provides about its target variable. The function with the highest information gain produces the best splits and better classifies the training data set according to the target variable.

$$IG = E(P) - E(C) \quad (1)$$

Where, IG is the information gain,
E(P) and E(C) is the entropy of parent and child node.

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad (2)$$

Where, $E(S)$ is the entropy of node S and p_i is the probability of i^{th} element in the data.

The impurity of a node is measured by its entropy. Entropy is a measure of how disordered or unpredictable a set of data is. A node with high entropy is a node with a lot of different classes, while a node with low entropy is a node with mostly the same class.

The information gain tells us how much the impurity of the parent node decreases when we split it on a particular feature. A high information gain indicates that the feature is a good predictor of the target variable. The information gain based splitting criteria is a popular choice for decision trees because it is easy to understand and implement. It is also a fairly accurate splitting criteria, and it can be used with both categorical and numerical features.

Entropy based splitting criteria is a popular and effective choice for decision trees. It is easy to understand and implement, and it is fairly accurate. However, it can be computationally expensive for large datasets, and it can lead to overfitting.

- Gini Impurity: Gini Impurity is one of the most commonly used criteria for splitting a decision tree.

$$Gini = 1 - \sum_{i=1}^{c} (p_i)^2 \quad (3)$$

The Gini impurity based splitting criteria chooses the feature that will result in the greatest decrease in Gini impurity. This is done by calculating the Gini impurity of the parent node, and then calculating the Gini impurity of each child node after the split. The feature with the largest difference in Gini impurity is chosen as the best split.

Gini impurity based splitting criteria is a popular and effective choice for decision trees. It is easy to understand and implement, and it is fairly accurate. However, it can be

sensitive to the order of the features, and it can lead to overfitting.

Pruning

Pruning is the technique of removing parts that prevent the full depth of the decision tree from being reached. The parts removed from the tree are those that do not provide the ability to classify instances. Training a decision tree to its full depth increases the likelihood of overfitting the training data. Therefore, pruning is important.

More simply, the goal of decision tree pruning is to build an algorithm that performs poorly on training data but generalizes better on test data. Optimizing the hyperparameters of a decision tree model makes the model fairer and saves significant time and money.

There are two types of pruning:

- Pre pruning:

The decision tree pre-pruning technique tunes the hyperparameters before the training pipeline. This is the so-called "early stopping" heuristic, which halts the growth of the decision tree and prevents it from reaching its full depth.
Stop the tree formation process to avoid small leaves forming. Cross-validation errors are monitored during each phase of tree splitting. Stop growing the decision tree when the error value stops decreasing. Here are the hyperparameters you can tune to stop early and prevent overfitting:
max_Depth, min_samples_leaf, min_samples_split
You can also use the same parameters for optimization to get a robust model. However, it is necessary to be careful because stopping early may lead to lack of fitting.

- Post Pruning:

Post-pruning does the opposite of pre-pruning, allowing the decision tree model to reach that depth. When the model reaches full depth, branches are removed to prevent overfitting the model.
The algorithm continues to split the data into smaller subsets until the resulting subsets have similar outcome variables. The tree has learned up to T data because the last subset of the tree consists of only a few data points. However, it may not predict well when new data points are introduced that differ from the learned data. The hyperparameters that can be adjusted after pruning and prevent overfitting are:
ccp_alpha.
ccp stands for Cost Complexity Pruning and can be used as another option to control the size of the tree. Higher values of ccp_alpha increase the number of nodes cleaned up.

Cost complexity pruning (post-pruning) steps:

- Train the Decision Tree model to its full depth
- Compute the ccp_alphas value using cost_complexity_pruning_path()
- Train the Decision Tree model with different ccp_alphas values and compute train and test performance scores
- Plot the train and test scores for each value of ccp_alphas values.
- This hyperparameter can also be used to tune to get the best fit models.

B. SYSTEM BLOCK DIAGRAM

1) LOAD DATASET:

The initial step of the system is to load the dataset that will be used to train and test the data. For this process we will be loading the Heart Disease dataset. This dataset contains a total of 76 attributes including the target.

2) FEATURE EXTRACTION:

The dataset is separated into the feature matrix and the target vector. The feature matrix contains the input features, while the target vector represents the corresponding target values for classification.

3) SPLITTING DATASET:

The dataset is split into two part: the train set and the test set. The train set contains 80% of the data while the test contains the remaining data. The train set is used to fit/train the dataset and the test set is used to evaluate the performance of the dataset.

4) CREATION OF DECISION TREE

The decision tree classifier class, DecisionTreeClassifier from the library scikit-learn is instantized and the hyper parameters are provided.

5) TRAINING THE DECISION TREE:

The Decision Tree classifier is trained using train set. During training, the nodes and split until a stopping condition is met.

6) TESTING:

The decision tree is used infer on the test set. The accuracy, precision and recall is calculated for the obtained result.

7) HYPERPARAMETER

Tuning: Steps 4, 5 and 6 are repeated until accecptable performance is obtained in the test set.

8) DECISION TREE VISUALIZATION:

After the perfect hyperparameters have been found. Plot the decision tree for visualization

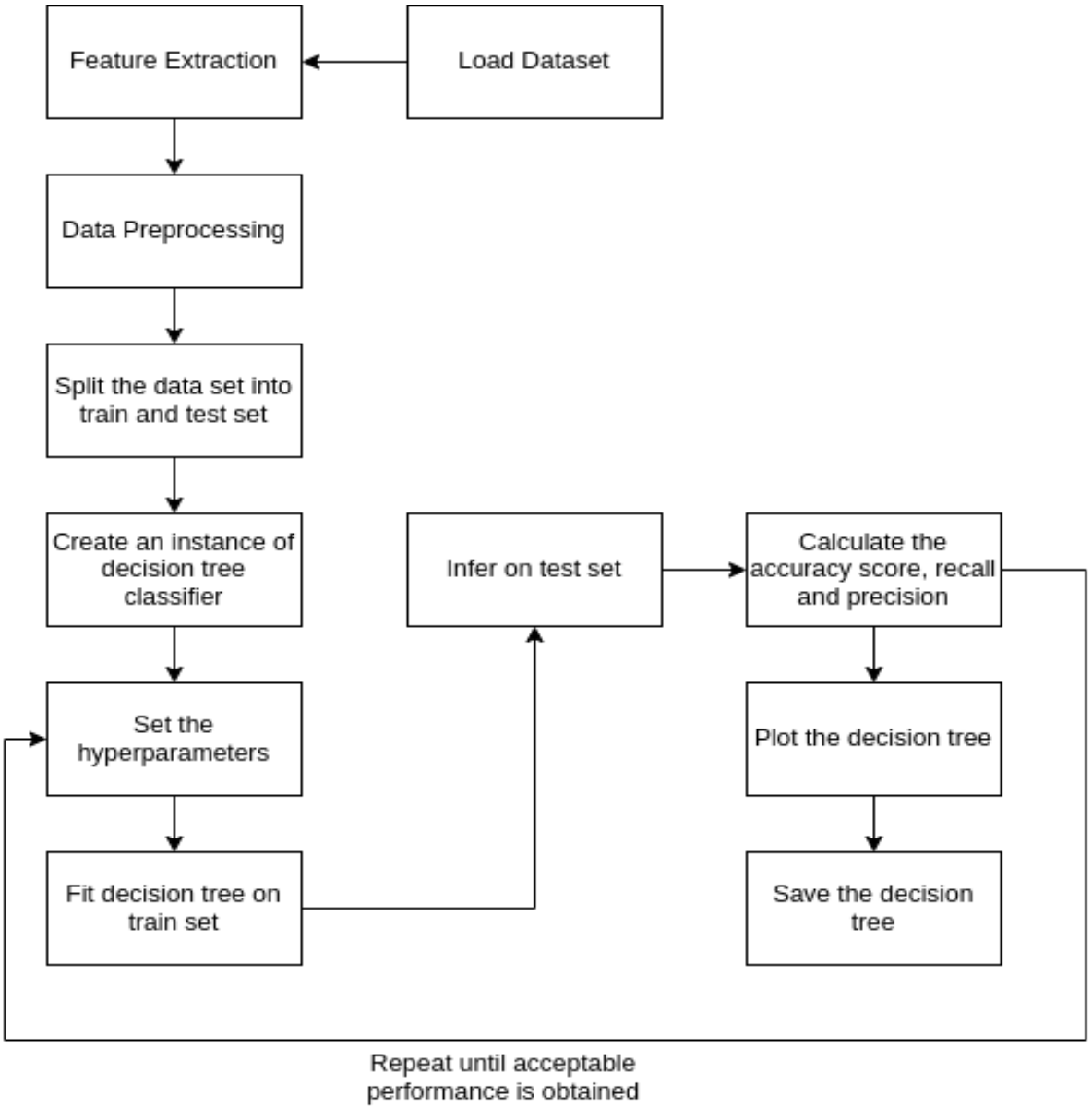

Figure 1 : System Block Diagram

III. Results:

The UCI Heart Disease dataset is a well-known dataset widely used in machine learning and data mining research for studying and predicting heart disease. It was contributed to the UCI Machine Learning Repository by several researchers from the Cleveland Clinic Foundation in Cleveland, Ohio. The dataset contains a collection of clinical and demographic attributes of patients, along with their corresponding diagnosis of heart disease. The following table shows the first five entries of the dataset.

TABLE I
FIRST 5 ENTRIES OF DATASET

Age	Sex	Pain Type	Rest Pb	Chol	fbs	thalach	target
63	1	1	145	233	1	150	0
67	1	4	160	286	0	108	1
67	1	4	120	229	0	129	1
37	1	3	130	250	0	187	0
41	0	2	130	204	0	172	0

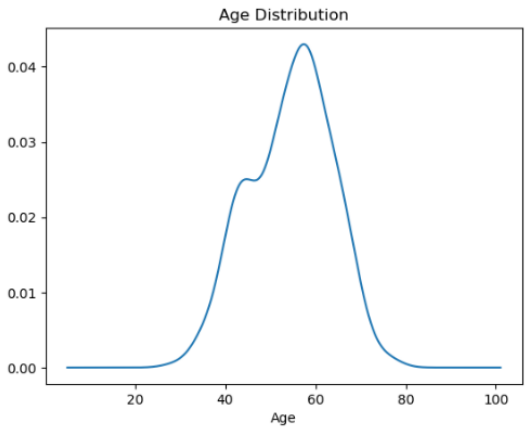

Figure 2 : Age Distribution

The plot above shows the distribution plot for age. It is visible that age follows gaussian distribution. Similarly, from

the plots below we can see that all the numeric attributes follow gaussian distribution.

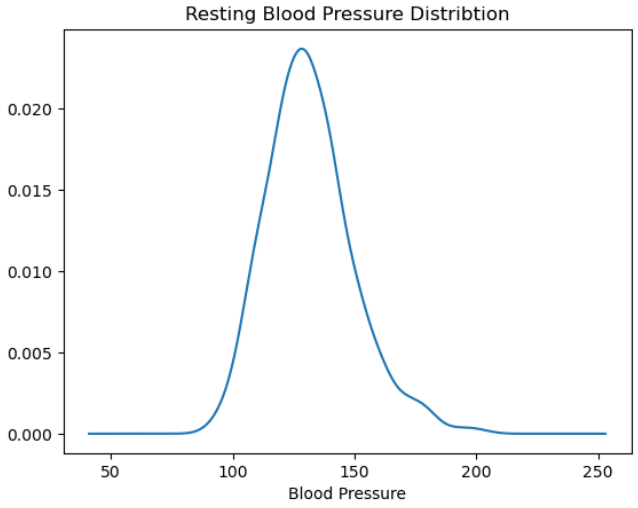

Figure 3 : Resting Blood Pressure Distribution

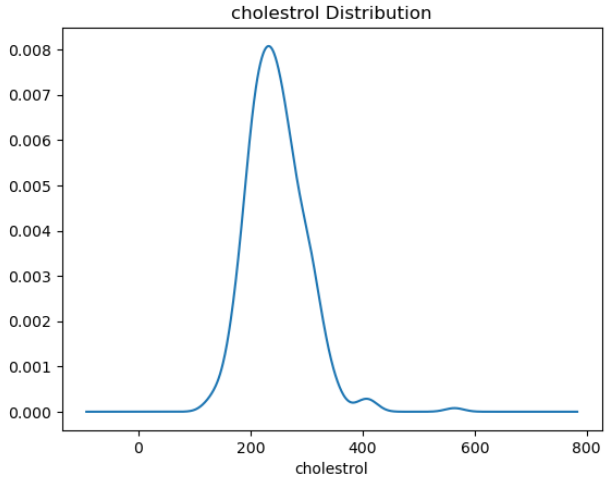

Figure 4 : Cholesterol Distribution

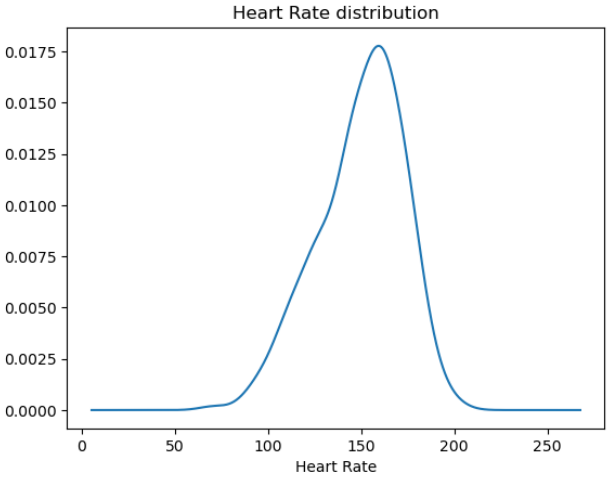

Figure 5 : Heart Rate Distribution

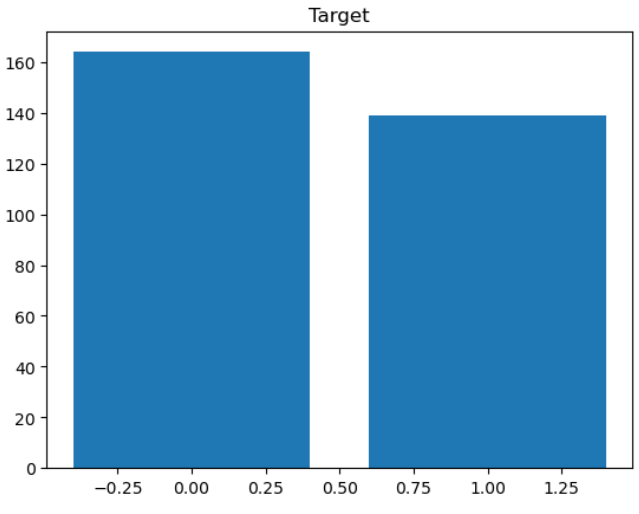

Figure 6 : Target

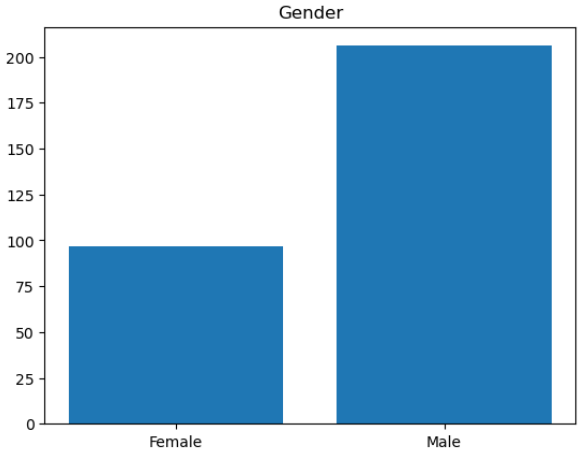

Figure 7 : Gender

From the plot above we can see that age is centered around 50. The mean and standard deviation for age is 53 and 9 respectively. Similarly, for resting blood pressure, cholesterol and heart rate the mean and standard deviations are 131, 17.6, 246, 51 and 149, 22.8 respectively.

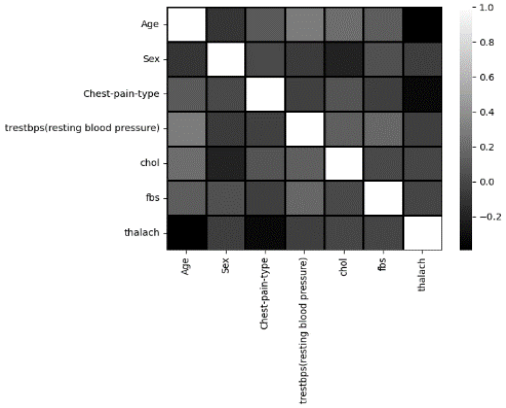

Figure 8 : Correlation Heatmap

The heatmap above represents the correlation among the input features. The darker the shade the closer the correlation coefficient close to -0.2 and the lighter the shade the higher positive correlation between the attributes. From the above heatmap, we can clearly see that there is no high level of correlation between the input attributes.

Initially, a decision tree classifier is instantized without any stopping criteria i.e., the nodes will keep on splitting until all the nodes are homogeneous (until all the elements on a node belong to the same class). The plot above shows the decision tree that we obtained without any stopping criteria. The Decision tree classifier continues to grow and expand until it perfectly fits the training data without any constraints or limitations. In other words, the tree keeps growing and adding nodes until it achieves a pure classification for each training instance, resulting in a model that overfits the data.

When a decision tree is allowed to grow without any stopping criteria, it can potentially create a complex and highly detailed structure. Each node in the tree corresponds to a decision rule based on a feature and its corresponding threshold. As the tree grows deeper, it becomes more specific in capturing intricate patterns and relationships within the training data.

While such an unrestricted decision tree may achieve perfect accuracy on the training data, it is likely to perform poorly on unseen or test data due to overfitting. Overfitting occurs when the model learns noise or irrelevant patterns from the training data, leading to reduced generalization capabilities. The overly complex structure of the decision tree can memorize the training instances instead of capturing meaningful patterns and relationships.

Train Set

	precision	recall	f1-score
False	1.00	1.00	1.00
True	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Test Set

	precision	recall	f1-score
False	0.78	0.66	0.71
True	0.55	0.70	0.62
accuracy			0.67
macro avg	0.67	0.68	0.66
weighted avg	0.69	0.67	0.68

Figure 9 : Classification Report

We found that this tree has a perfect score on the training set i.e., it correctly predicts every instance present in the dataset. While it performs badly on the test set with an accuracy of 67%. The figure below shows the classification report of the prediction on train and test set.

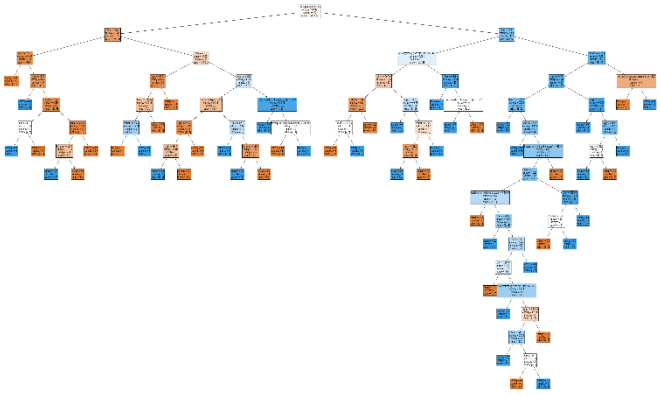

Figure 10 : Decision Tree Without any stopping Criteria

From the figure above, we can clearly see that the decision tree is overfitting. To mitigate overfitting, various stopping criteria can be employed during the construction of decision trees. These criteria include limitations on tree depth, minimum number of samples required in a leaf node, maximum number of leaf nodes, or minimum improvement in impurity measures (e.g., Gini index or information gain) at each split. These constraints prevent the tree from becoming overly complex and promote generalization by finding the most informative and relevant splits. Applying appropriate stopping criteria helps strike a balance between model complexity and generalization performance, avoiding the pitfalls of overfitting. Regularization techniques such as pruning can also be employed to simplify the decision tree after it has been fully grown, removing unnecessary nodes and reducing complexity while preserving important decision rules.

Now, we will be applying various stopping criteria and regularization techniques and will be comparing the performance with this model.

With impurity metrics as gini impurity and max depth of 3 we obtain the tree structure as shown in the figure below.

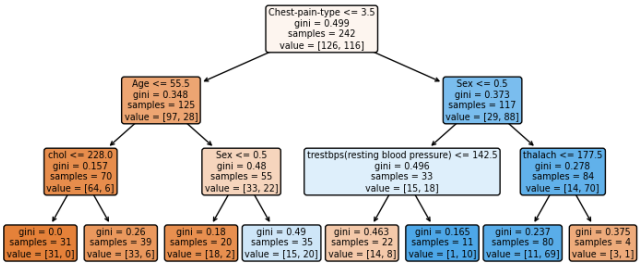

Figure 11 : Decision Tree with max depth 3

The performance on the train and test set is also as show in the figure below.

Train Set

	precision	recall	f1-score	support
False	0.85	0.79	0.82	126
True	0.79	0.85	0.82	116
accuracy			0.82	242
macro avg	0.82	0.82	0.82	242
weighted avg	0.82	0.82	0.82	242

Test Set

	precision	recall	f1-score	support
False	0.82	0.74	0.78	38
True	0.63	0.74	0.68	23
accuracy			0.74	61
macro avg	0.73	0.74	0.73	61
weighted avg	0.75	0.74	0.74	61

Figure 12 : Classification report with max depth 3

We can see that the accuracy on the test set has improved on this tree. The tree is able to classify the false classes more precisely than it can classify true class. Due to its in ability to classify the true cases the performance of the model decreases.

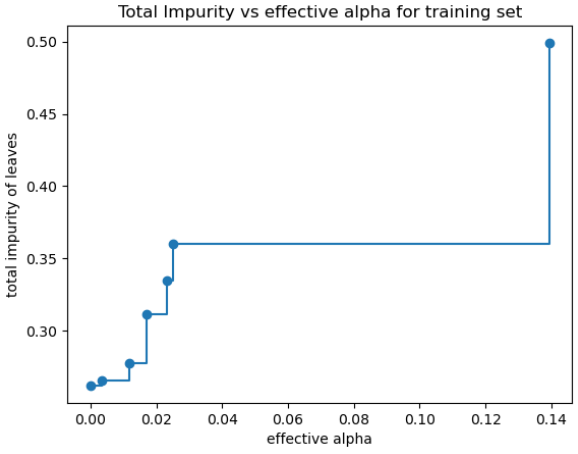

Figure 13 : Total Impurity vs effective alpha for training set

The plot above shows the total impurity in the nodes with respect to the value of alpha for cost-complexity pruning. If the value is alpha is 0 no pruning of nodes occur. Similarly, if the value of alpha is greater than 0.14, every node will be pruned and only the root node will be remaining.

Figure 14 : Decision tree with alpha > 0.14

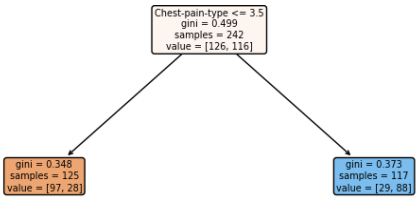

Figure 15 : Decision tree with alpha = 0.026

The decision tree structure as shown in the figure is obtained by allowing the tree to grow until it completely generalizes the training data and then pruning the tree with the value alpha=0.0118.

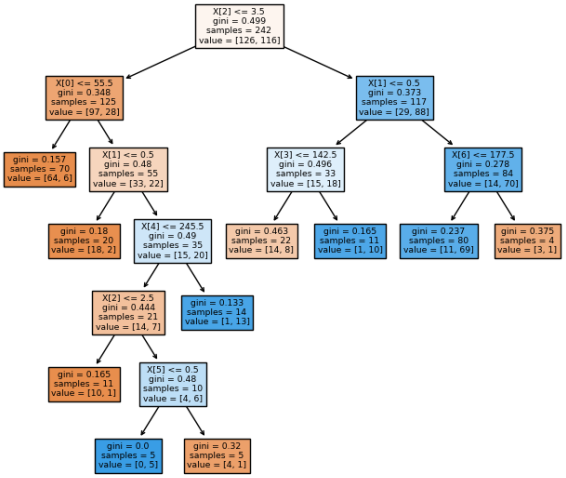

Figure 16 : Decision tree with alpha value 0.0118

Train Set

	precision	recall	f1-score	support
False	0.78	0.77	0.77	126
True	0.75	0.76	0.76	116
accuracy			0.76	242
macro avg	0.76	0.76	0.76	242
weighted avg	0.76	0.76	0.76	242

Test Set

	precision	recall	f1-score	support
False	0.82	0.74	0.78	38
True	0.63	0.74	0.68	23
accuracy			0.74	61
macro avg	0.73	0.74	0.73	61
weighted avg	0.75	0.74	0.74	61

Figure 17 : Classification report

From the figure, we can see that the tree is no longer overfitting, but rather it underfits the data. The train and test accuracy were found to be 76% and 74% percentage.

IV. DISCUSSION AND ANALYSIS

The decision tree algorithm applied to the heart disease dataset provides valuable insights into the relationship between clinical and demographic attributes and the presence of heart disease. Upon analysis, it was observed that the decision tree initially exhibited signs of overfitting, capturing intricate details and noise specific to the training data. This was evident through the excessive depth, numerous decision rules, and highly specific splits within the tree. Consequently, the model achieved near-perfect accuracy on the training set but struggled to generalize well to unseen data.

To address the overfitting, appropriate stopping criteria were introduced to control the growth of the decision tree. By limiting the tree's depth, setting a minimum number of samples per leaf node, and implementing minimum improvement thresholds for splitting, the decision tree's complexity was reduced. This helped prevent the model from memorizing noise and irrelevant patterns, improving its generalization capabilities.

However, in some cases, the introduction of overly restrictive stopping criteria or the complexity of the dataset led to underfitting. The decision tree became too simple, lacking depth, decision rules, and adequate splits to capture important patterns and relationships in the data. As a result, the model exhibited reduced accuracy and poor performance on both the training and test sets.

While decision trees are often effective for classification tasks, there are situations where they may struggle to accurately classify complex datasets, such as the heart disease dataset. In such cases, it becomes evident that a more powerful algorithm is needed to tackle the task at hand. Here, we discuss the limitations of decision trees and the necessity for more advanced techniques.

Decision trees make decisions based on a series of binary splits on feature values, creating hierarchical decision rules. However, these trees have limited expressive power and may struggle to capture intricate relationships and nonlinear interactions among variables. The heart disease dataset, with its diverse set of clinical and demographic attributes, may require a more sophisticated approach to effectively classify instances.

To overcome these limitations, more powerful algorithms can be considered. Ensemble methods like random forests or gradient boosting can improve classification performance by combining multiple decision trees and leveraging their collective decision-making. These algorithms can capture complex relationships and interactions that individual decision trees may overlook, leading to enhanced accuracy and robustness.

Moreover, deep learning algorithms, such as neural networks, have gained popularity in recent years due to their ability to learn complex patterns and extract high-level features from raw data. They are capable of automatically learning intricate representations of the heart disease dataset, potentially outperforming decision trees in terms of accuracy and classification performance.

V. CONCLUSION

In conclusion, the application of decision trees on the heart disease dataset provides valuable insights into the relationships between clinical and demographic attributes and the presence of heart disease. Decision trees offer interpretability, allowing for clear decision rules and an understanding of the factors influencing the classification outcome. However, it is essential to address the limitations of decision trees, such as their tendency to overfit or underfit the data.

By implementing proper stopping criteria, the decision tree's growth can be controlled, mitigating overfitting and improving generalization capabilities. However, in some cases, decision trees may struggle to accurately classify complex datasets, indicating the need for more powerful algorithms. Ensemble methods, like random forests or gradient boosting, and advanced techniques such as deep learning, can be employed to capture intricate relationships and nonlinear interactions, enhancing classification performance.

Choosing the appropriate algorithm depends on various factors, including the trade-off between accuracy and interpretability, computational resources, and availability of training data. While more powerful algorithms offer the potential for improved classification accuracy, they may require additional resources and may sacrifice the interpretability offered by decision trees.

VI. REFERENCES

[1] Janosi,Andras, Steinbrunn,William, Pfisterer,Matthias, and Detrano,Robert. (1988). Heart Disease. UCI Machine Learning Repository. https://doi.org/10.24432/C52P4X.

[2] Kotsiantis, S.B. Decision trees: a recent overview. Artif Intell Rev 39, 261–283 (2013). https://doi.org/10.1007/s10462-011-9272-4

ANUJ RAYAMAJHI is an ambitious individual currently in the final year of his Bachelor's degree program in Computer Engineering at the esteemed Institute of Engineering Thapathali Campus. With a keen interest in various fields such as Artificial Intelligence, Machine Learning, Data Science, Software Development, and Engineering, Anuj possesses a diverse range of skills and a thirst for knowledge. He constantly seeks out opportunities to enhance his understanding of these subjects through research, online courses, and practical experience.

Apart from his academic pursuits, Anuj has a passion for music, sports, and computer games. In his leisure time, he enjoys exploring different genres of music, engaging in sports activities to stay active, and immersing himself in the virtual worlds of computer games.

With a strong foundation in computer engineering and a multifaceted interest in emerging technologies, Anuj is driven to make significant contributions in the fields of AI, machine learning, and data science. He strives to stay up-to-date with the latest advancements in these domains, continuously expanding his knowledge and honing his skills. Anuj's dedication, enthusiasm, and well-rounded interests make him a promising individual poised to excel in the ever-evolving field of technology

AAYUSH REGMI is a dynamic individual currently pursuing his Bachelor's degree in Computer Engineering at the renowned Institute of Engineering Thapathali Campus. With a strong inclination towards technology, Aayush has developed a keen interest in a wide range of fields, including Artificial Intelligence, Machine Learning, Data Science, Web Development, Quantum Computing, and more. His diverse expertise reflects his commitment to exploring various facets of computer science and pushing the boundaries of innovation.

In addition to his academic pursuits, Aayush finds solace and joy in his hobbies. As an avid sports enthusiast, he actively engages in cricket and football, relishing the thrill of competition and teamwork. Aayush also has a creative side and enjoys playing musical instruments, finding harmony in the melodies he creates.

With an ever-curious mind and a passion for learning, Aayush strives to stay ahead in the rapidly evolving world of technology. He is particularly intrigued by the emerging field of Quantum Computing and its potential to revolutionize the computing landscape. Aayush's dedication, coupled with his diverse skill set, positions him to make significant contributions in web development, quantum computing, and other innovative domains.

Driven by a relentless pursuit of knowledge and a desire to create a positive impact, Aayush is determined to shape the future through his expertise in computer engineering. His enthusiasm, adaptability, and holistic interests make him a promising individual poised to excel in the dynamic and ever-expanding field of technology.

APPENDIX A : CODE

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree
import graphviz
from sklearn.preprocessing import StandardScaler,
MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
precision_score, recall_score, classification_report

data =
pd.read_csv('cleveland.csv').drop(['exang','oldpeak','
slope','ca','thal', 'restecg'], axis=1)

X = data.drop(["target"], axis=1)
y = data["target"] != 0

# Dataset Exploration
X[["Age", "trestbps(resting blood pressure)", "chol",
"thalach"]].describe()
X.describe()

y.value_counts().values
plt.bar(y.value_counts().index,
y.value_counts().values)
plt.title("Target")

X["Age"].plot.kde()
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("")

X["trestbps(resting blood pressure)"].plot.kde()
plt.title("Resting Blood Pressure Distribtion")
plt.xlabel("Blood Pressure")
plt.ylabel("")
X["chol"].plot.kde()
plt.title("cholestrol Distribution")
plt.xlabel("cholestrol")
plt.ylabel("")

X["thalach"].plot.kde()
plt.title("Heart Rate distribution")
plt.xlabel("Heart Rate")
plt.ylabel("")
plt.bar(X['fbs'].value_counts().index,

X['fbs'].value_counts().values)
plt.title("Fasting Blood Pressure ( >120)")
plt.xticks([0,1])
plt.bar(X['Sex'].value_counts().index,

X['Sex'].value_counts().values)
plt.title("Gender")
plt.xticks([0,1], ["Female", "Male"])

# Classification without any hyper parameters
x_train, x_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=69)
x_train.shape, x_test.shape, y_train.shape,
y_test.shape

## With Entropy
dt_clf =
tree.DecisionTreeClassifier(criterion='entropy')

dt_clf.fit(x_train, y_train)
plt.figure(figsize=(80,30))
tree.plot_tree(dt_clf, filled=True,
feature_names=X.columns, fontsize=12)

### Classification Report For Train and Test Set
print("Train Set")
print(classification_report(y_train,
dt_clf.predict(x_train)))
print("Test Set")
print(classification_report(y_test,
dt_clf.predict(x_test)))

# Classification With Hyper parameters
## Without Normalization
x_train, x_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=69)
x_train.shape, x_test.shape, y_train.shape,
y_test.shape

### With Entropy
#### max-depth=2
```

```
dt_clf =
tree.DecisionTreeClassifier(criterion='gini',max_depth
=3, ccp_alpha=0.026)

dt_clf.fit(x_train, y_train)
dt_clf.cost_complexity_pruning_path(x_train, y_train)
ccp_alphas, impurities =
dt_clf.cost_complexity_pruning_path(x_train,
y_train).values()
print(ccp_alphas, impurities)

fig, ax = plt.subplots()
ax.plot(ccp_alphas, impurities, marker="o",
drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for
training set")
plt.figure(figsize=(9,4))
tree.plot_tree(dt_clf, filled=True,
feature_names=X.columns, fontsize=7, rounded=True)

##### Classification Report For Train and Test Set
print("Train Set")
print(classification_report(y_train,
dt_clf.predict(x_train)))

print("Test Set")
print(classification_report(y_test,
dt_clf.predict(x_test)))
dt_clf =
tree.DecisionTreeClassifier(criterion='gini',max_depth
=4)

ccp_alphas, impurities =
dt_clf.cost_complexity_pruning_path(x_train,
y_train).values()
ccp_alphas, impurities

fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o",
drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for
training set")
```

```
dt_temp =
tree.DecisionTreeClassifier(ccp_alpha=0.01181129)
dt_temp.fit(x_train, y_train)
plt.figure(figsize=(8,7))
tree.plot_tree(dt_temp, filled=True)

print("Train Set")
print(classification_report(y_train,
dt_clf.predict(x_train)))

print("Test Set")
print(classification_report(y_test,
dt_clf.predict(x_test)))


parameters = {
    "criterion":["gini", "entropy"],
    "max_depth": range(1,8),
    "min_impurity_decrease": np.arange(0.000, 0.005,
0.0001)
}
tree.DecisionTreeClassifier(criterion="log_loss")
best_clf = GridSearchCV(tree.DecisionTreeClassifier(),
param_grid=parameters, n_jobs=-1)
best_clf.fit(x_train, y_train)
```