Supervised method (Decision trees) algorithms

What is a Decision Tree?

A decision tree is one of the supervised machine learning algorithms. This algorithm can be used for regression and classification problems. It is a type of tree structure where each node represents a decision, and each branch represents a possible outcome of that decision. The leaves of the tree represent the final predictions. Trees are powerful algorithms that can handle complex datasets. They are a fundamental building block for more advanced machine learning algorithms in data science and can be effectively used in a wide range of real-world applications.

Here are 7 interesting facts about decision trees:

- ✓ They do not need the numerical input data to be scaled. Whatever the numerical values are, decision trees don't care.
- ✓ Decision trees handle categorical features in the raw text format (Scikit-Learn doesn't support this, TensorFlow's trees implementation does).
- ✓ Different to other complex learning algorithms, the results of decision trees can be interpreted. It's fair to say that decision trees are not blackbox type models.
- ✓ While most models will suffer from missing values, decision trees are okay with them.
- ✓ Trees can handle imbalanced datasets. You will only have to adjust the weights of the classes.
- ✓ Trees can provide the feature importance or how much each feature contributed to the model training results.
- ✓ Trees are the basic building blocks of ensemble methods such as random forests and gradient boosting machines.

- A decision tree follows a set of **if-else** conditions to visualize the data and classify it according to the conditions.

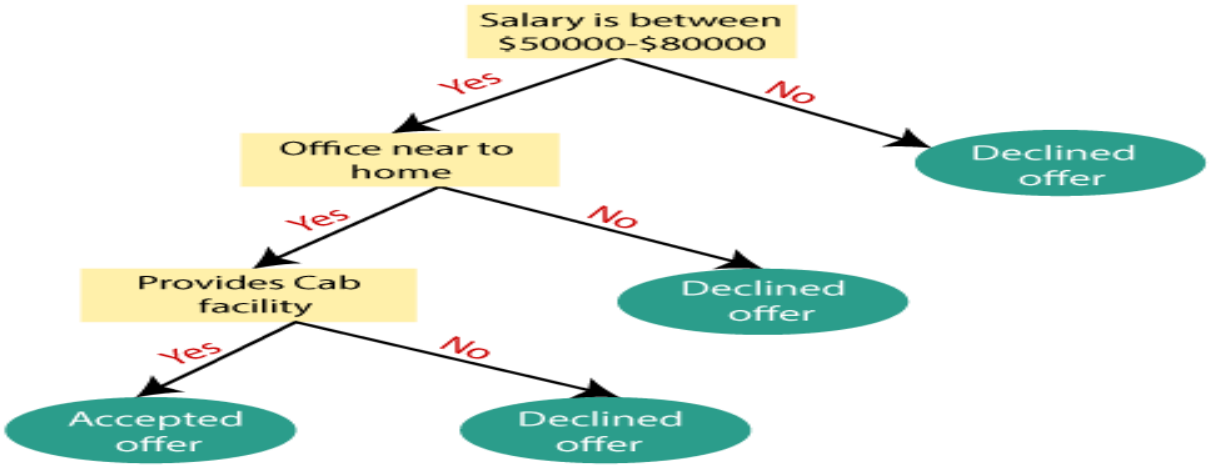
How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute

with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

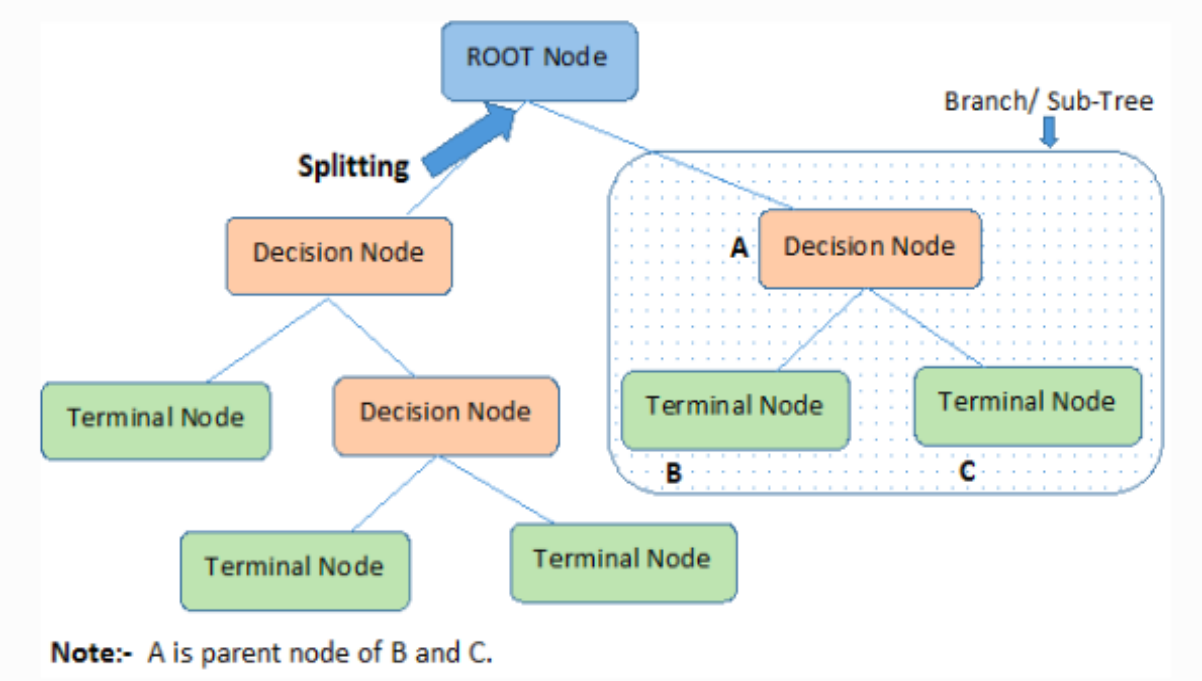
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:
  - ✓ **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
  - ✓ **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).** Attribute selection measures typically refer to methods and metrics used to evaluate the importance or relevance of individual attributes (features) in a dataset when building machine learning models.
  - ✓ **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
  - ✓ **Step-4:** Generate the decision tree node, which contains the best attribute.
  - ✓ **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Important terminology

- **Root Node:** This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
- **Branch or Sub-Tree**: A part of the entire decision tree is called a branch or sub-tree.
- **Splitting:** Dividing a node into two or more sub-nodes based on if-else conditions.
- **Decision Node:** After splitting the sub-nodes into further sub-nodes, then it is called the decision node.
- **Leaf or Terminal Node:** This is the end of the decision tree where it cannot be split into further sub-nodes.
- **Pruning:** Removing a sub-node from the tree is called pruning.



Advantages of decision Trees:

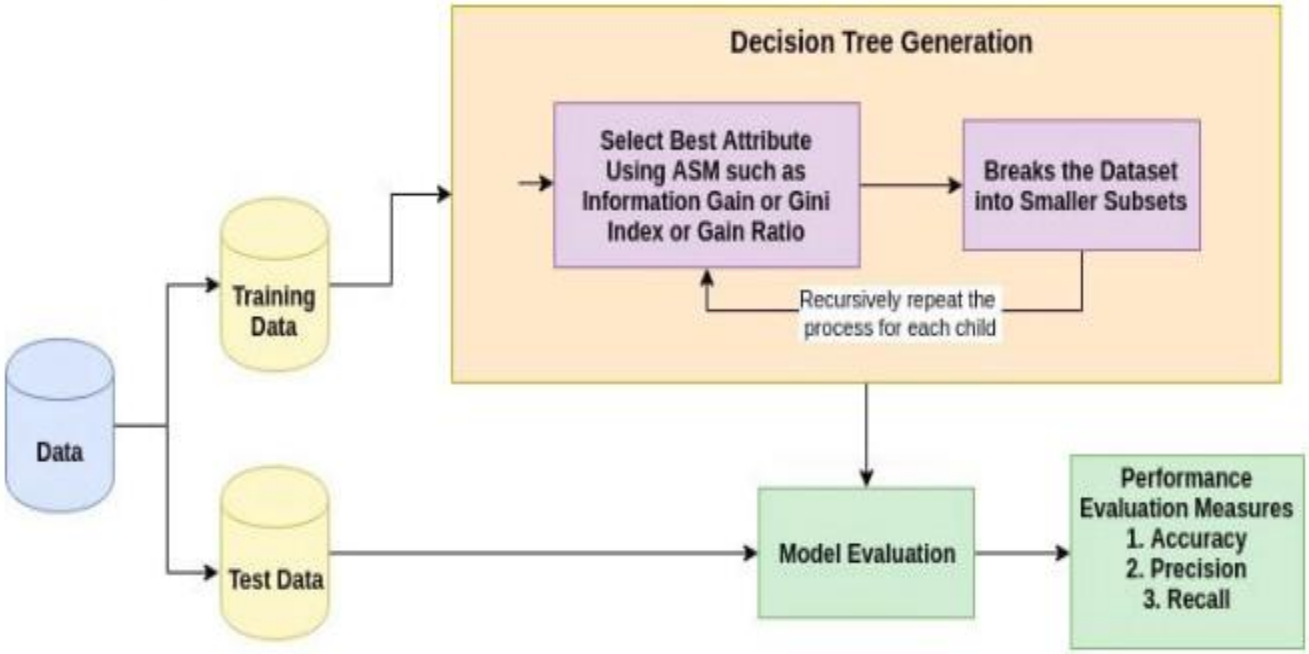
- Works for numerical or categorical data and variables.
- Models problems with multiple outputs.
- Tests the reliability of the tree.
- Requires less data cleaning than other data modeling techniques.
- Easy to explain to those without an analytical background.

Disadvantages

- Affected by noise in the data.
- Not ideal for large datasets.
- Can disproportionately value, or weigh, attributes.
- The decisions at nodes are limited to binary outcomes, reducing the complexity that the tree can handle.
- Trees can become very complex when dealing with uncertainty and numerous linked outcomes.

Working of Decision Tree

- The root node feature is selected based on the results from the Attribute Selection Measure(ASM).
- The ASM is repeated until a leaf node, or a terminal node cannot be split into sub-nodes. These measures are used to assess the contribution of each attribute to the predictive power of a model and to select a subset of the most relevant attributes while excluding less important ones.



<u>What is Attribute Selective Measure (ASM)?</u>

Attribute Subset Selection Measure is a technique used in the data mining process for data reduction. The data reduction is necessary to make better analysis and prediction of the target variable. This technique used in decision tree algorithms to select the best attribute for splitting the data. The goal of ASM is to identify the attribute that will create the most homogeneous subsets of data after the split, thereby maximizing the information gain or reducing the impurity of the data.

The two main ASM techniques are

1. Gini index
2. Information Gain(ID3)

1) Gini index:

The Gini index is a measure of impurity in a set of data. It is calculated by summing the squared probabilities of each class. A lower Gini index indicates a purer set of data. Gini index is typically preferred when the classes are balanced.

The Gini index for a decision tree is calculated by summing the Gini indices of the child nodes. The Gini index of a child node is calculated by summing the squared probabilities of each class in the child node. The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favors mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.

The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

The advantages of Gini index include:

- It is simple to calculate.
- It is interpretable.
- It is robust to overfitting.

The disadvantages of Gini index include:

- It is not as effective as information gain when the classes are imbalanced.
- It can be sensitive to noise.

N/B: To use Gini index to choose the best split in a decision tree, you would start by calculating the Gini index for each of the features. The feature with the lowest Gini index is the best choice for the split.

For example, let's say we have a decision tree that is trying to predict whether a customer will churn. The tree has two features: age and income. The Gini index for age is 0.4 and the Gini index for income is 0.2. Therefore, the best choice for the split is age.

- The measure of the degree of probability of a particular variable being wrongly classified when it is randomly chosen is called the Gini index or Gini impurity. The data is equally distributed based on the Gini index.

When creating a decision tree, the Gini index is used to evaluate the quality of a split (a way to partition the data into subsets) at each node in the tree. The Gini index for a split is calculated by taking the weighted average of the Gini impurity of the subsets created by the split.

Gini Index (GI) is defined as,

$$Gini\ Index\ (GI) = \sum_{i=1}^{K} p_i^2$$

where K is the number of class labels,

p_i is the proportion of ith class label

Pi= probability of an object being classified into a particular class.

When you use the Gini index as the criterion for the algorithm to select the feature for the root node., The feature with the least Gini index is selected.

Gini Impurity: Gini impurity is a measure of the disorder or impurity of a set of data points. In the context of decision trees, it is used to evaluate the impurity of a set of training examples. The Gini impurity for a set is calculated by summing the squared probabilities of each class occurring in that set and subtracting the result from 1.

Gini Impurity

Now, Gini Impurity is just the reverse mathematical term of Gini Index and is defined as,

$$Gini\ Impurity = 1 - \sum_{i=1}^{K} p_i^2$$

$$= 1 - Gini\ Index$$

where K is the number of class labels,

p_i is the proportion of ith class label

So, it is a measure of anti-homogeneity and hence, the feature with the least Gini Impurity is selected to be the best split feature.

CALCULATION: Gini Index and Gini Impunity:

Consider an example of an Exploratory Analyzed Data of people winning or losing a tournament, given their Age and Gender:

Gender	Age (<50)	Age (>=50)
(F)	P = 10 N = 390	P = 0 N = 100
(M)	P = 250 N = 50	P = 50 N = 150

For calculating the Gini Index for Gender, Gini Index of Male (M) and Female (F) categories need to be calculated.

$$Gini\ Index(Gender) = w_1 * Gini\ Index(Male) + w_2 * Gini\ Index(Female)$$

Here, $w_1 = \frac{Total\ number\ of\ Males\ (M)}{Total\ number\ of\ people} = \frac{(250+50)+(50+150)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{1}{2}$

and $w_2 = \frac{Total\ number\ of\ Females\ (F)}{Total\ number\ of\ people} = \frac{(10+390)+(0+100)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{1}{2}$

Now, $Gini\ Index(Male) = \left\{\frac{(250+50)}{(250+50)+(50+150)}\right\}^2 + \left\{\frac{(50+150)}{(250+50)+(50+150)}\right\}^2 = \left(\frac{300}{500}\right)^2 + \left(\frac{200}{500}\right)^2 = \frac{13}{25}$

$Gini\ Index(Female) = \left\{\frac{(10+0)}{(10+0)+(390+100)}\right\}^2 + \left\{\frac{(390+100)}{(10+0)+(390+100)}\right\}^2 = \left(\frac{10}{500}\right)^2 + \left(\frac{490}{500}\right)^2 = \frac{1201}{1250}$

Finally, $Gini\ Index(Gender) = \frac{1}{2} * \frac{13}{25} + \frac{1}{2} * \frac{1201}{1250} = 0.7404$

Similarly, for calculating the Gini Index of Age, Gini Index of labels '<50' i.e., age less than 50 and '>=50' i.e., age greater than or equal to 50 need to be calculated.

$$Gini\ Index(Age) = w_1 * Gini\ Index(< 50) + w_2 * Gini\ Index(>= 50)$$

Here, $w_1 = \frac{Total\ number\ of\ (<50)}{Total\ number\ of\ people} = \frac{(10+390)+(250+50)}{\{(10+390)+(250+50)\}+\{(0+100)+(50+150)\}} = \frac{7}{10}$

and $w_2 = \frac{Total\ number\ of\ (>=50)}{Total\ number\ of\ people} = \frac{(0+100)+(50+150)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{3}{10}$

Now, $Gini\ Index(< 50) = \left\{\frac{(10+250)}{(10+250)+(390+50)}\right\}^2 + \left\{\frac{(390+50)}{(10+250)+(390+50)}\right\}^2 = \left(\frac{260}{700}\right)^2 + \left(\frac{440}{700}\right)^2 = \frac{653}{1225}$

$Gini\ Index(>= 50) = \left\{\frac{(0+50)}{(0+50)+(100+150)}\right\}^2 + \left\{\frac{(100+150)}{(0+50)+(100+150)}\right\}^2 = \left(\frac{50}{300}\right)^2 + \left(\frac{250}{300}\right)^2 = \frac{13}{18}$

Finally, $Gini\ Index(Age) = \frac{7}{10} * \frac{653}{1225} + \frac{3}{10} * \frac{13}{18} = 0.5898 \quad (approx)$

So, as Gini Index (Gender) is greater than Gini Index (Age), hence, Gender is the best split-feature as it produces more homogeneous child nodes.

Calculating Gini Impurity for Gender, Gini Impurity of Male (M) and Female (F) need to be calculated

$$Gini\ Impurity(Gender) = w_1 * Gini\ Impurity(Male) + w_2 * Gini\ Impurity(Female)$$

Here, $w_1 = \frac{Total\ number\ of\ Males\ (M)}{Total\ number\ of\ people} = \frac{(250+50)+(50+150)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{1}{2}$

and $w_2 = \frac{Total\ number\ of\ Females\ (F)}{Total\ number\ of\ people} = \frac{(10+390)+(0+100)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{1}{2}$

Now, $Gini\ Impurity(Male) = 1 - \left[\left\{\frac{(250+50)}{(250+50)+(50+150)}\right\}^2 + \left\{\frac{(50+150)}{(250+50)+(50+150)}\right\}^2\right] = 1 - \left[\left(\frac{300}{500}\right)^2 + \left(\frac{200}{500}\right)^2\right] = 1 - \frac{13}{25} = \frac{12}{25}$

$Gini\ Impurity(Female) = 1 - \left[\left\{\frac{(10+0)}{(10+0)+(390+100)}\right\}^2 + \left\{\frac{(390+100)}{(10+0)+(390+100)}\right\}^2\right] = 1 - \left[\left(\frac{10}{500}\right)^2 + \left(\frac{490}{500}\right)^2\right] = 1 - \frac{1201}{1250} = \frac{49}{1250}$

Finally, $Gini\ Impurity(Gender) = \frac{1}{2} * \frac{12}{25} + \frac{1}{2} * \frac{49}{1250} = 0.2596$

Similarly, for calculating the Gini Impurity of Age, Gini Impurity of labels '<50' i.e., age less than 50 and '>=50' i.e., age greater than or equal to 50 need to be calculated

$$Gini\ Impurity(Age) = w_1 * Gini\ Impurity(< 50) + w_2 * Gini\ Impurity(>= 50)$$

Here, $w_1 = \frac{Total\ number\ of\ (<50)}{Total\ number\ of\ people} = \frac{(10+390)+(250+50)}{\{(10+390)+(250+50)\}+\{(0+100)+(50+150)\}} = \frac{7}{10}$

and $w_2 = \frac{Total\ number\ of\ (>=50)}{Total\ number\ of\ people} = \frac{(0+100)+(50+150)}{\{(250+50)+(50+150)\}+\{(10+390)+(0+100)\}} = \frac{3}{10}$

Now, $Gini\ Impurity(< 50) = 1 - \left[\left\{\frac{(10+250)}{(10+250)+(390+50)}\right\}^2 + \left\{\frac{(390+50)}{(10+250)+(390+50)}\right\}^2\right] = 1 - \left[\left(\frac{260}{700}\right)^2 + \left(\frac{440}{700}\right)^2\right] = \frac{572}{1225}$

$Gini\ Impurity(>= 50) = 1 - \left[\left\{\frac{(0+50)}{(0+50)+(100+150)}\right\}^2 + \left\{\frac{(100+150)}{(0+50)+(100+150)}\right\}^2\right] = 1 - \left[\left(\frac{50}{300}\right)^2 + \left(\frac{250}{300}\right)^2\right] = \frac{5}{18}$

Finally, $Gini\ Impurity(Age) = \frac{7}{10} * \frac{572}{1225} + \frac{3}{10} * \frac{5}{18} = 0.4102 \quad (approx)$

So, as Gini Impurity (Gender) is less than Gini Impurity (Age), hence, Gender is the best split-feature.

2) ***Information Gain (ID3)***

Information gain is a measure of how much information is gained by splitting a set of data on a particular feature. It is calculated by comparing the entropy of the original set of data to the entropy of the two child sets. A higher information gain indicates that the feature is more effective at splitting the data. Information gain is typically preferred when the classes are imbalanced.

The information gain for a decision tree is calculated by comparing the entropy of the original set of data to the entropy *(**Entropy** is a measure of uncertainty or randomness in a system. It is often used in machine learning to measure the impurity of a data set. A high-entropy data set is a data set with a lot of uncertainty, while a low-entropy data set is a data set with a lot of certainty)* of the two child sets. The entropy of a set of data is calculated by summing the probabilities of each class in the set multiplied by the log of the probability of each class.

The advantages of information gain include:

- It is more effective than Gini index when the classes are imbalanced.
- It is less sensitive to noise.

The disadvantages of information gain include:

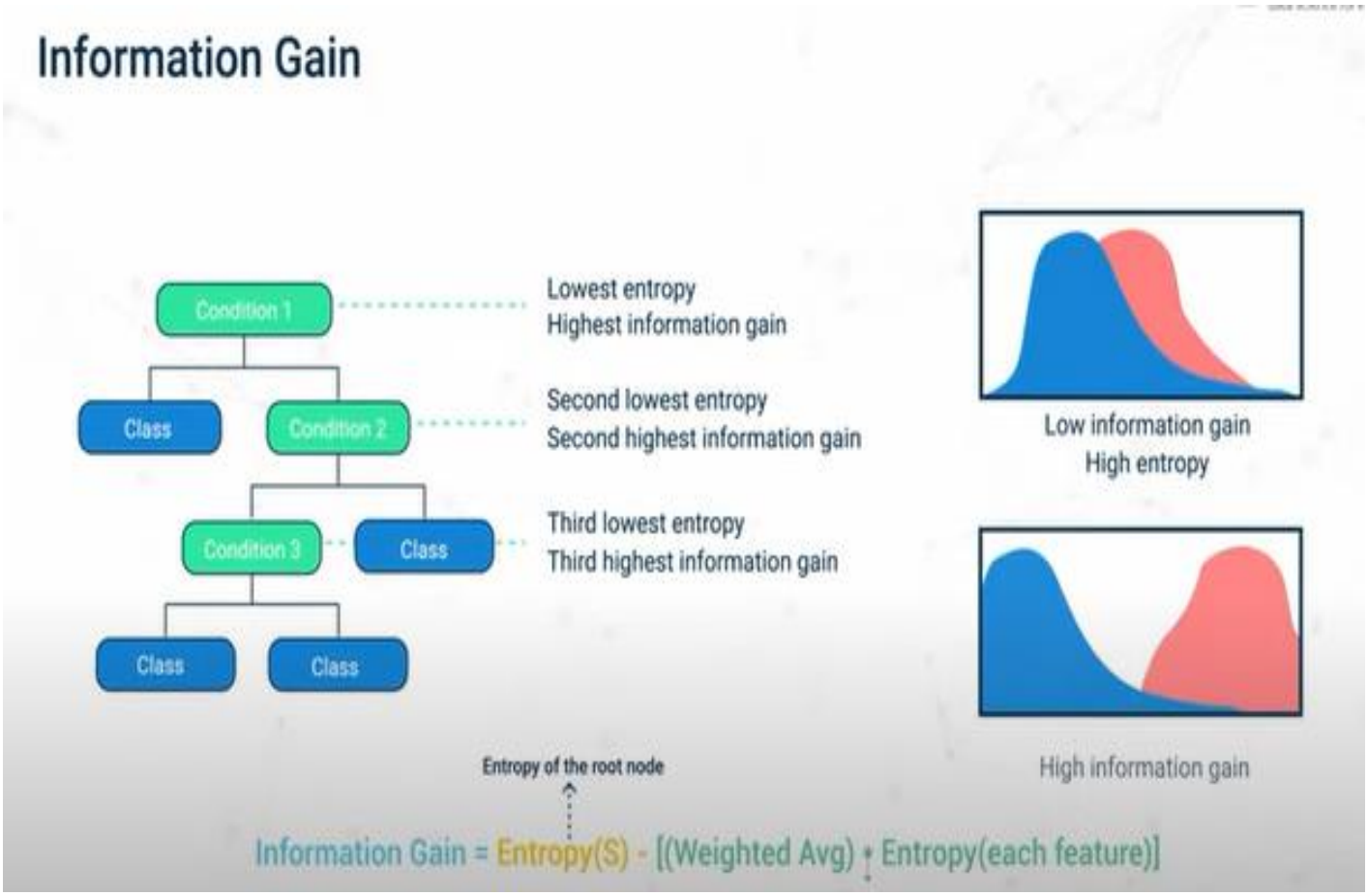
- It is more complex to calculate.
- It is less interpretable.

N/B: To use information gain to choose the best feature to split a set of data, you would start by calculating the information gain for each of the features. The feature with the highest information gain is the best choice for the split.

For example, let's say we have a set of data about customers who have churned. The features in the data set are age, income, and location. The information gain for age is 0.2,

the information gain for income is 0.4, and the information gain for location is 0.1. Therefore, the best choice for the split is income.

- **Entropy is the main concept of this algorithm**
    - which helps determine a feature or attribute that gives maximum information about a class is called Information gain or ID3 algorithm.
    - By using this method, we can reduce the level of entropy from the root node to the leaf node.



CALCULATION: Information Gain:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy (each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. The entropy of a data set can be used to measure how well the data is classified. *A data set with a high entropy is a data set that is not well classified, while a data set with a low entropy is a data set that is well classified.*

Entropy is used in machine learning algorithms such as decision trees and random forests. These algorithms use entropy to decide how to split the data into smaller and smaller subsets. The goal is to create subsets that are as pure as possible, meaning that they contain mostly instances of the same class.
Entropy can be calculated as:
Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)
Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

$$InformationGain(feature) = Entropy(Dataset) - Entropy(feature)$$

The feature with the largest entropy information gain should be the root node to build the decision tree.

This dataset decides if you should buy a car given 3 features: Age, Mileage, and whether or not the car is a road test.

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy
Recent	High	Yes	Buy
Old	Low	No	Don't buy
Recent	High	No	Don't buy

Entropy
Entropy by definition is a lack of order or predictability. It is the measure of impurity in a bunch of examples. The node is the purest if it has the instances of only one class.

$$\text{Entropy} = -\sum_{i}^{n} log_2(P_i)$$

Where n= number of features
i = feature
P = Probability of i

The entropy is calculated for every node. The first node i.e., the root node always has all the examples in the dataset.

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

Root node

✓ 2 instances
✗ 2 instances

Calculating Entropy for the root node

$$E = -\left(P(✓) * \log_2(P(✓)) + P(✗) * \log_2(P(✗))\right)$$

Probability formula:

$$P(✓) = \frac{\text{count of } ✓}{\text{total examples}}$$

$$P(✓) = 2/4 = 0.5$$

$$P(✗) = 2/4 = 0.5$$

Plugging these values in the formula we get:

$$E = -\left(0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)\right)$$

$$E = 1$$

***Entropy for the parent node is 1**. We will use this in the next steps when calculating the information gain.*

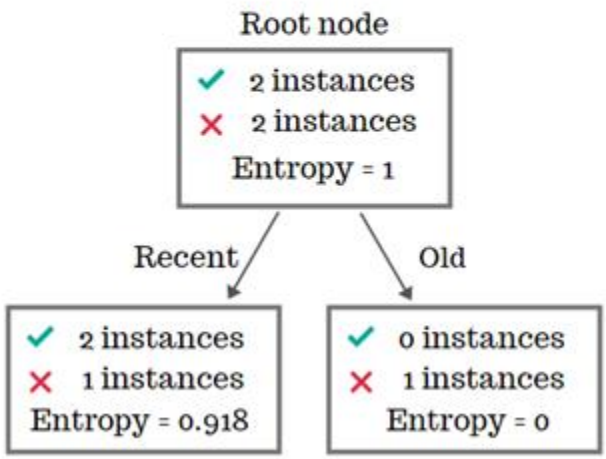
Information Gain (IG)

IG = Entropy (Parent) - weighted_avg * Entropy (Children)

Entropy (Parent) = 1

Information gain for Age

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



#Total instances in child node 1 (Recent)

#Total instances in parent node

#Total instances in child node 2 (Old)

$$\text{Children Entropy} = \frac{3}{4}(0.918) + \frac{1}{4}(0)$$

$$= 0.688$$

Child Entropy

parent Entropy

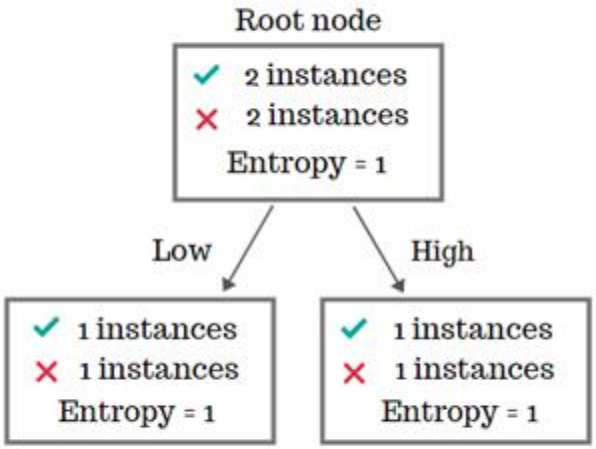
$$I.G = 1 - 0.688$$

$$= 0.3112$$

Information gain if we split at "AGE"

Information gain for Milage

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

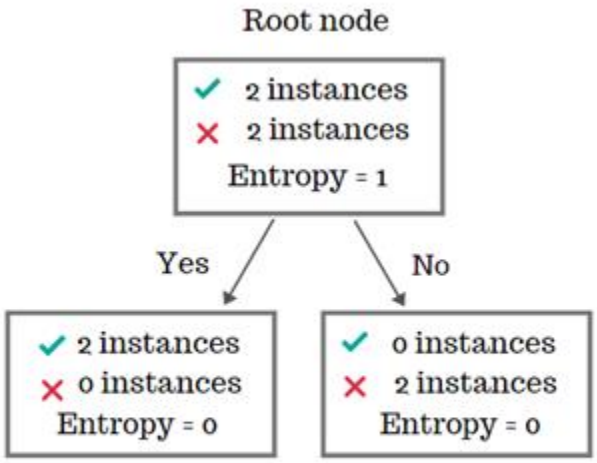


Children Entropy = 2/4 (1) + 2/4 (1) = 1
Information gain = 1 - 1 = 0

In the diagram above, we can tell by looking that the entropy of the left child node is 1 because the ratio of class instances is 1:1. This is the most impure a node can be. The same is true for the child node on the right.

Information gain for Road Tested

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



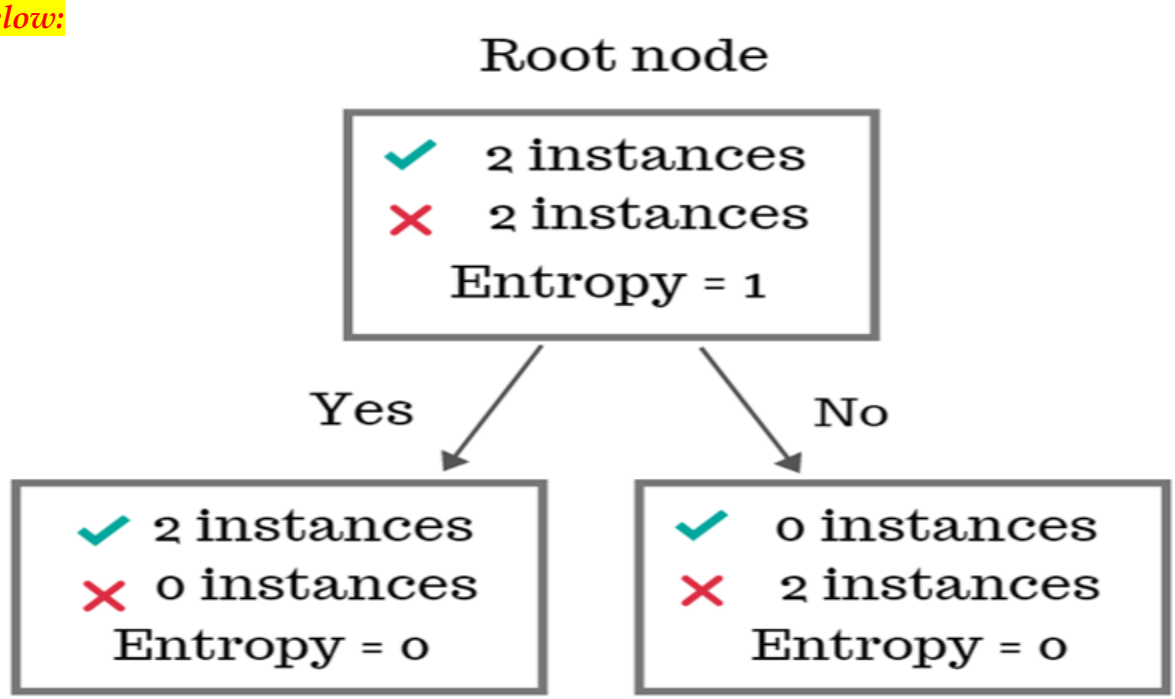
Children Entropy = 2/4 (0) + 2/4 (0) = 0
Information gain = 1 - 0 = 1

Here, the children nodes have the entropy value of 0, because each child node has the instances of only one class: The purest a node can be.
To summarize, the table below shows the **information gain** for every feature:

Information gain for every feature

	Age	Mileage	Road Tested	Buy
	Recent	Low	Yes	Buy
	Recent	High	Yes	Buy
	Old	Low	No	Don't buy
	Recent	High	No	Don't buy
Info. Gain	0.3112	0	1	

The maximum information gain is for the feature "Road Tested," and therefore we'll select this to be our first split feature. Doing so would generate the tree structure given below:



Since the entropy is zero at the leaf nodes, we'll stop slitting. If this wasn't the case, then we would continue to find the information gain for the preceding parent and children nodes, until any one of the stopping criteria is met.

Different decision tree algorithms.

Algorithm	Splitting criterion	Explanation	Pros	Cons
ID3 (Iterative Dichotomiser 3)	Entropy	Entropy is a measure of the uncertainty in a dataset. ID3 selects the attribute that maximizes the information gain, which is the reduction in entropy after the split.	Simple to understand and implement. Good for both categorical and numerical data.	Can be sensitive to noise in the data. Can overfit to the training data.
C4.5 (Classification and Regression Trees, version 4.5.)	Information gain or gain ratio	Information gain is the same measure used by ID3. Gain ratio is a normalized version of information gain that is less sensitive to the number of values in an attribute. C4.5 can use either information gain or gain ratio as its splitting criterion.	More robust to noise in the data than ID3. Less likely to overfit to the training data.	More computationally expensive than ID3.
CART (Classification and Regression Trees)	Gini impurity	Gini impurity is a measure of how often a randomly chosen instance is incorrectly classified if it is assigned to a class based on the majority vote of the other instances in the same set. CART selects the attribute that minimizes the Gini impurity after the split.	Good for both categorical and numerical data. Can handle missing values.	Can be sensitive to the choice of hyperparameters. Can overfit to the training data.

IMPLEMENTATION IN GOOGLE COLAB:

Step 1: Load the diabetes dataset and Have a glance at the shape.

diabetes dataset

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

This data frame contains following columns:

- *Pregnancies:* This column typically represents the number of times a person has been pregnant. It's a numeric (integer) attribute.
- *Glucose:* This column represents the glucose concentration in the blood, often measured in milligrams per deciliter (mg/dL). It's a numeric (continuous) attribute and is an important indicator in diabetes diagnosis.
- *BloodPressure:* This column represents the blood pressure of the individual. It's also a numeric (continuous) attribute and is typically measured in millimeters of mercury (mm Hg).
- *SkinThickness:* This column represents the thickness of the skinfold at a certain location on the body. It's another numeric (continuous) attribute, which might be used in body composition measurements.
- *Insulin:* This column represents the insulin level in the blood, typically measured in microunits per milliliter (µU/mL). It's a numeric (continuous) attribute and is related to glucose metabolism.
- *BMI (Body Mass Index):* BMI is a measure of body fat based on an individual's weight in relation to their height. It's a numeric (continuous) attribute, and it's calculated as weight in kilograms divided by the square of height in meters.
- *DiabetesPedigreeFunction:* This column is a numeric (continuous) attribute that represents a function designed to measure the likelihood of diabetes based on family hitory. It often accounts for the genetic component of diabetes risk.
- *Age:* Age is a numeric (integer) attribute that represents the age of the individual.
- *Outcome:* This column is a binary (categorical) attribute that typically represents the target variable or the label in a diabetes prediction task. It often takes two values, such as 0 for "No Diabetes" and 1 for "Diabetes," indicating the presence or absence of diabetes in the individual.

```
import pandas as pd
wainaina = pd.read_csv('/content/diabetes.csv')
print("Shape of the dataset:", wainaina.shape)
wainaina.head(10)
```

```
Shape of the dataset: (768, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Line	Explanation
import pandas as pd	This line imports the pandas library and assigns it the alias pd, which is a common convention in Python for working with pandas.
wainaina = pd.read_csv('/content/diabetes.csv')	This line reads a CSV file named "diabetes.csv" located in the /content directory and stores the data in a pandas DataFrame named wainaina.
print("Shape of the dataset:", wainaina.shape)	This line prints the shape of the DataFrame wainaina, which is the number of rows and columns in the dataset. It uses the .shape attribute to retrieve this information.
wainaina.head(10)	This line displays the first 10 rows of the DataFrame wainaina. The head() method is used to display a portion of the dataset for a quick overview.

Step 2: Split the dataset into training and testing sets.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load the dataset
wainaina = pd.read_csv('diabetes.csv')
# Split the data into features (X) and labels (y)
X = wainaina.drop('Outcome', axis=1)
y = wainaina['Outcome']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Line	Explanation
import pandas as pd	Import the pandas library as 'pd' for data manipulation.

from sklearn.model_selection import train_test_split	Import the train_test_split function from scikit-learn for splitting data into training and testing sets.
from sklearn.tree import DecisionTreeClassifier	Import the DecisionTreeClassifier class from scikit-learn for creating a decision tree model.
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix	Import various metrics for model evaluation such as accuracy, classification report, and confusion matrix.
wainaina = pd.read_csv('diabetes.csv')	Load a dataset from a CSV file named 'diabetes.csv' into a pandas DataFrame named 'wainaina'.
X = wainaina.drop('Outcome', axis=1)	Create a DataFrame 'X' containing the features (independent variables) by dropping the 'Outcome' column.
y = wainaina['Outcome']	Create a Series 'y' containing the labels (dependent variable), specifically the 'Outcome' column.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)	Split the data into training and testing sets with an 80-20 split ratio. 'X_train' and 'y_train' represent the training data, and 'X_test' and 'y_test' represent the testing data. The 'random_state' parameter ensures reproducibility.

Step 3: ***Build the Decision Tree Model.***

```
[24]
# Initialize the decision tree classifier
geo= DecisionTreeClassifier(criterion='entropy',random_state=42)
#clf = DecisionTreeClassifier(criterion='gini',random_state=42)
# Train the model on the training data
geo.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

Line	Explanation
geo = DecisionTreeClassifier(criterion='entropy', random_state=42)	Initialize a decision tree classifier named 'geo' with the 'entropy' criterion for splitting nodes. The 'random_state' parameter is set to 42 for reproducibility.
geo.fit(X_train, y_train)	Train the 'geo' decision tree model using the training data. 'X_train' contains the features, and 'y_train' contains the corresponding labels. The model is trained to learn patterns and relationships within the data.

Step 4: ***Use the trained model to make predictions on the test data.***

```
[25] # Make predictions on the testing set.
     y_pred = geo.predict(X_test)
     print (y_pred)
```

```
[0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1
 1 0 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0
 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0
 1 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0
 0 1 0 1 1 0]
```

Line	Explanation
y_pred = geo.predict(X_test)	Use the trained decision tree classifier 'geo' to make predictions on the testing set 'X_test'. The resulting predictions are stored in the 'y_pred' variable.
print(y_pred)	Print the predictions to the console. The variable 'y_pred' contains the model's predicted outcomes for the testing data. This allows you to see the model's predictions on the test set.

Step 5: ***Evaluate the model.***

```
[18] # Calculate the accuracy of the decision tree classifier.
     accuracy = accuracy_score(y_test, y_pred)
     print('Accuracy:', accuracy)
     #Alternatively use the code below:
     print(f"Accuracy: {accuracy:.2f}")
     # Evaluate the model's performance
     report = classification_report(y_test, y_pred)
     print("Classification Report:")
     print(report)
     conf_matrix = confusion_matrix(y_test, y_pred)
     print("Confusion Matrix:")
     print(conf_matrix)
```

Line	Explanation
accuracy = accuracy_score(y_test, y_pred)	Calculate the accuracy of the decision tree model by comparing the true labels 'y_test' from the testing data to the predicted labels 'y_pred'. The result is stored in the 'accuracy' variable.

print('Accuracy:', accuracy)	Print the accuracy score to the console. This shows the proportion of correctly predicted outcomes in the testing set.
print(f"Accuracy: {accuracy:.2f}")	Alternatively, this line prints the accuracy score as a formatted string with two decimal places.
report = classification_report(y_test, y_pred)	Generate a classification report that includes precision, recall, F1-score, and support for both classes. The report is stored in the 'report' variable.
print("Classification Report:")	Print the classification report to the console, providing detailed performance metrics for the model's predictions.
conf_matrix = confusion_matrix(y_test, y_pred)	Compute a confusion matrix that displays true positives, true negatives, false positives, and false negatives. The matrix is stored in the 'conf_matrix' variable.
print("Confusion Matrix:")	Print the confusion matrix to the console. It helps assess how well the model is classifying instances in the testing set.

```
+ Code   + Text

2s
Accuracy: 0.7207792207792207
Accuracy: 0.72
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        99
           1       0.60      0.64      0.62        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154

Confusion Matrix:
[[76 23]
 [20 35]]
```

Accuracy:

The first line shows two accuracy scores: 0.7207792207792207 and 0.72 (rounded to two decimal places). These scores represent the proportion of correctly predicted outcomes in the testing dataset. The first score is more precise, while the second score is rounded for simplicity.

Classification Report:

- The classification report provides detailed metrics for each class (0 and 1) as well as summary statistics:

- Precision measures how many of the predicted positive instances were correctly classified. For class 0, it's 0.79, and for class 1, it's 0.60. Precision measures the accuracy of positive predictions made by the model for each class.
  - ✓ Formula: Precision = TP / (TP + FP)
  - ✓ Where TP is the number of true positives, and FP is the number of false positives.
  - ✓ For Class 0 (not diabetic):
    Precision = 76 / (76 + 23) = 76 / 99 ≈ 0.7677 (rounded to two decimal places).
  - ✓ For Class 1 (diabetic):
    Precision = 35 / (35 + 20) = 35 / 55 ≈ 0.6364.
- Recall (also known as true positive rate or sensitivity) measures how many of the actual positive instances were correctly classified. For class 0, it's 0.77, and for class 1, it's 0.64. It measures the model's ability to correctly identify actual positive instances for each class.
  - ✓ Formula: Recall = TP / (TP + FN)
  - ✓ Where TP is the number of true positives, and FN is the number of false negatives.
  - ✓ For Class 0 (not diabetic):
    Recall = 76 / (76 + 23) = 76 / 99 ≈ 0.7677.
  - ✓ For Class 1 (diabetic):
    Recall = 35 / (35 + 20) = 35 / 55 ≈ 0.6364.
- F1-Score is the harmonic mean of precision and recall, providing a balance between the two metrics. For class 0, it's 0.78, and for class 1, it's 0.62. It provides a balanced measure of model performance.
  - ✓ Formula: F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
  - ✓ For Class 0 (not diabetic):
    *F1-Score = 2 * (0.7677 * 0.7677) / (0.7677 + 0.7677) ≈ 0.7677.*
  - ✓ For Class 1 (diabetic):
    *F1-Score = 2 * (0.6364 * 0.6364) / (0.6364 + 0.6364) ≈ 0.6364.*
- Support is the number of occurrences of each class in the testing dataset (e.g., 99 instances of class 0 and 55 instances of class 1). Support indicates the number of instances for each class in the testing dataset.
  - ✓ For Class 0 (not diabetic), the support is 99.
  - ✓ For Class 1 (diabetic), the support is 55.
- Accuracy is the overall accuracy of the model on the testing dataset. It matches the accuracy score shown separately at the beginning. Accuracy measures the overall correctness of the model's predictions.
  - ✓ Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN)
  - ✓ In this case, it's approximately 0.7208 or 72.08%.
  - ✓

Confusion Matrix:

A confusion matrix is a table that summarizes the performance of a classification model. It shows the number of correct and incorrect predictions made by the model on a set of test data. The confusion matrix can be used to calculate various metrics, such as accuracy, precision, recall, and F1 score.

The confusion matrix is typically a 2x2 table for binary classification problems, but it can be extended to larger matrices for multi-class classification problems. The rows of the confusion matrix represent the actual classes, and the columns represent the predicted classes.

- ✓ The top-left cell (76) represents the true negatives (TN), which are instances correctly predicted as class 0.
- ✓ The top-right cell (23) represents false positives (FP), which are instances incorrectly predicted as class 1.
- ✓ The bottom-left cell (20) represents false negatives (FN), which are instances incorrectly predicted as class 0.
- ✓ The bottom-right cell (35) represents true positives (TP), which are instances correctly predicted as class 1.

The confusion matrix helps assess the model's performance in binary classification by showing how well it classifies instances as true positives, true negatives, false positives, and false negatives.

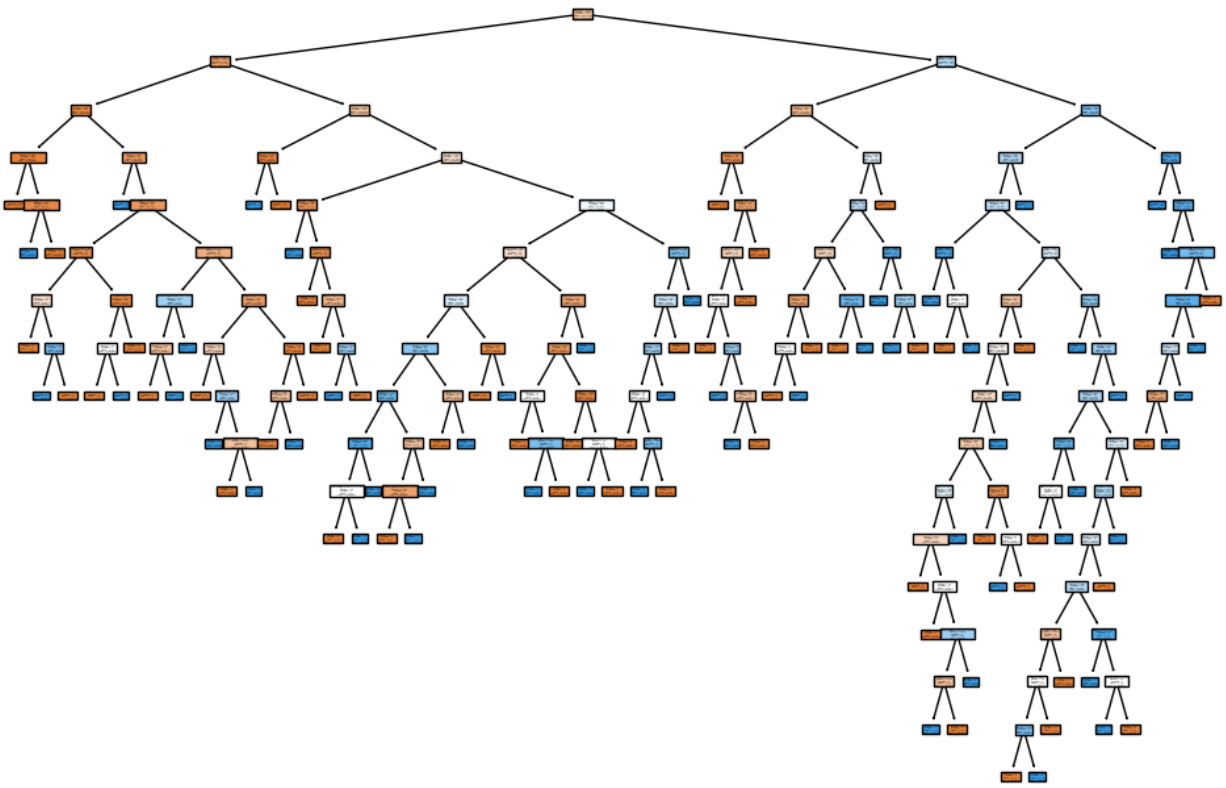
Step 6: Visualize the Decision Tree.

```
+ Code  + Text

[27] from sklearn.tree import plot_tree
     import matplotlib.pyplot as plt
     plt.figure(figsize=(12, 8))
     plot_tree(geo, filled=True, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"])
     plt.show()
```

Line	Explanation
from sklearn.tree import plot_tree	Import the plot_tree function from scikit-learn for visualizing decision trees.
import matplotlib.pyplot as plt	Import the matplotlib library to create and display plots.
plt.figure(figsize=(12, 8))	Create a figure for the plot with a specified size (12 inches in width and 8 inches in height).
plot_tree(geo, filled=True, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"])	Use the plot_tree function to visualize the decision tree model 'geo'. The filled=True argument fills the nodes with color, 'feature_names' is provided to label the features, and 'class_names' labels the target classes.

plt.show()	Display the generated decision tree plot. This will show the structure of the decision tree with nodes, splits, and leaf nodes, each filled with color representing class distribution. The features and class names are also displayed.

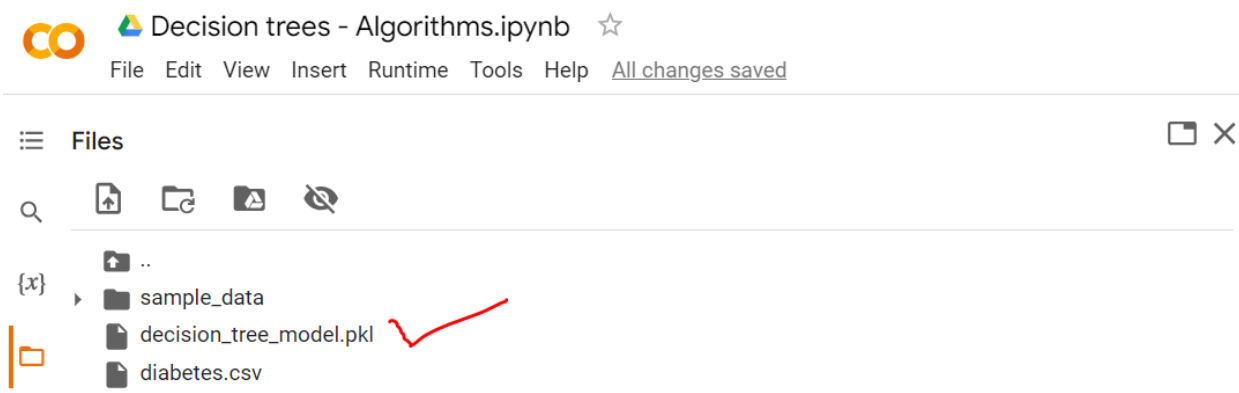


Step 7: Save that model.

```
import joblib
# Assuming 'clf' is your trained DecisionTreeClassifier model
geo = DecisionTreeClassifier(random_state=42)
geo.fit(X_train, y_train)
# Save the trained model to a file
joblib.dump(clf, 'decision_tree_model.pkl')
```

```
['decision_tree_model.pkl']
```

Line	Explanation
import joblib	Import the joblib library for saving and loading Python objects, including machine learning models.
geo = DecisionTreeClassifier(random_state=42)	Initialize a new DecisionTreeClassifier model named 'geo' with the specified random state (for reproducibility).
geo.fit(X_train, y_train)	Train the 'geo' model using the training data 'X_train' (features) and 'y_train' (labels).
joblib.dump(geo, 'decision_tree_model.pkl')	Save the trained 'geo' model to a file named 'decision_tree_model.pkl' using joblib. This allows you to store the model for future use without needing to retrain it.



Step 8: ***Load that model and make a prediction using my data.***

```
import joblib
import pandas as pd
# Load the saved model
loaded_model = joblib.load('decision_tree_model.pkl')
# Prepare your new data with attribute names
new_data = pd.DataFrame({
    'Pregnancies': [2, 4, 1, 5],
    'Glucose': [100, 150, 95, 130],
    'BloodPressure': [70, 80, 60, 90],
    'SkinThickness': [32, 35, 25, 40],
    'Insulin': [45, 50, 40, 60],
    'BMI': [32.0, 35.5, 29.0, 40.2],
    'DiabetesPedigreeFunction': [0.4, 0.5, 0.3, 0.6],
    'Age': [30, 35, 28, 40]
})
```

```python
# Make predictions on the new data
predictions = loaded_model.predict(new_data)
# Create a mapping from 0 to "Not Diabetic" and 1 to "Diabetic"
diabetes_mapping = {0: "0-Not Diabetic", 1: "1-Diabetic"}
# Print the predictions with labels
for i, prediction in enumerate(predictions):
    label = diabetes_mapping[prediction]
    print(f"Data {i + 1}: Predicted Outcome - {label}")
```

```
[31]
Data 1: Predicted Outcome - 0-Not Diabetic
Data 2: Predicted Outcome - 1-Diabetic
Data 3: Predicted Outcome - 0-Not Diabetic
Data 4: Predicted Outcome - 1-Diabetic
```

Line	Explanation
import joblib	Import the joblib library to load the saved model.
import pandas as pd	Import the pandas library to work with data in a tabular format.
loaded_model = joblib.load('decision_tree_model.pkl')	Load the previously saved decision tree model from the file 'decision_tree_model.pkl'.
new_data = pd.DataFrame({ ... })	Prepare new data as a pandas DataFrame with attribute names and values for prediction.
predictions = loaded_model.predict(new_data)	Use the loaded model to make predictions on the new data.
diabetes_mapping = {0: "0-Not Diabetic", 1: "1-Diabetic"}	Create a mapping that associates 0 with "Not Diabetic" and 1 with "Diabetic" for human-readable labels.
for i, prediction in enumerate(predictions): ...	Iterate through the predictions and map numeric predictions to labels, then print the predictions with labels for each data point.