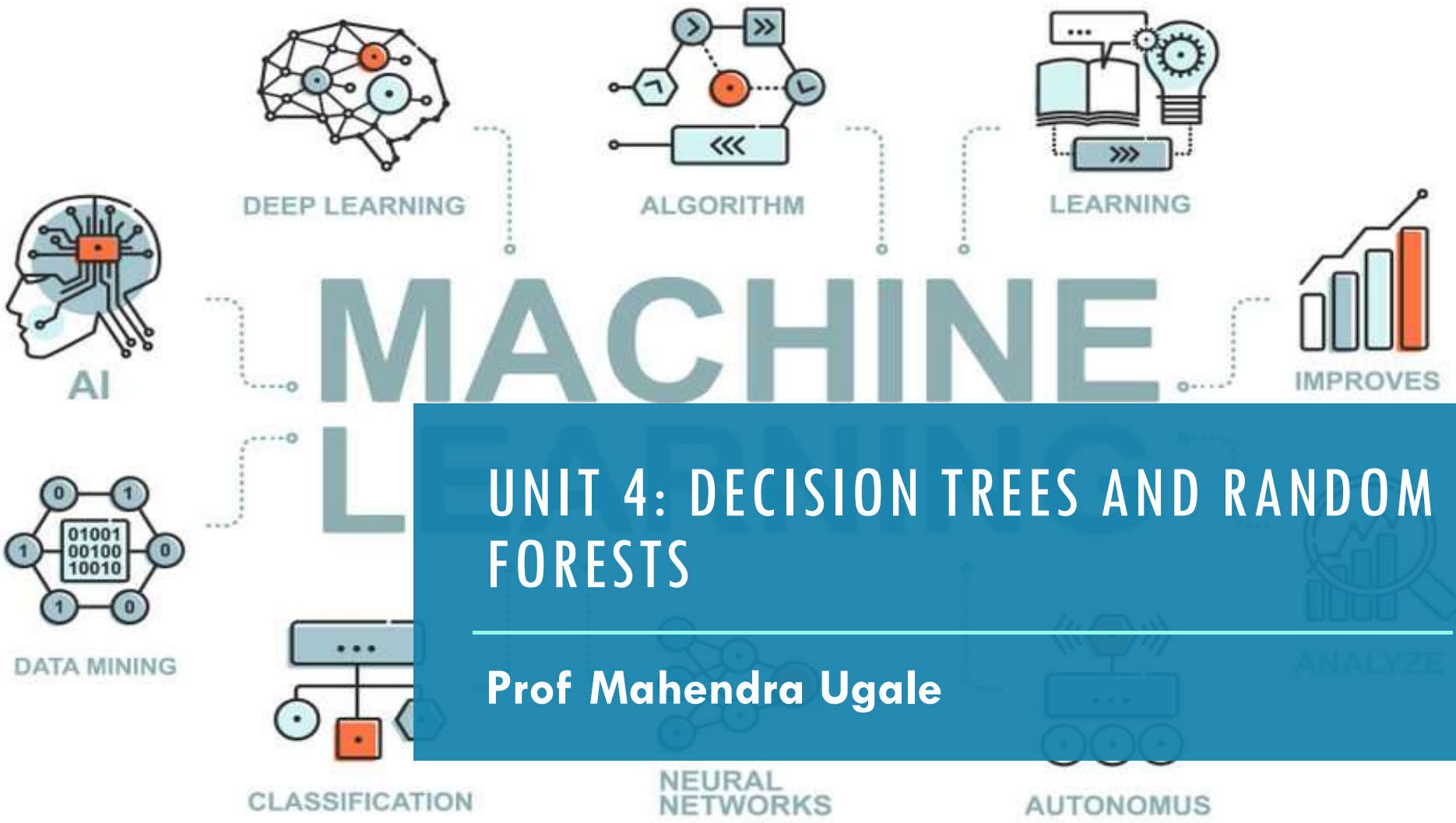
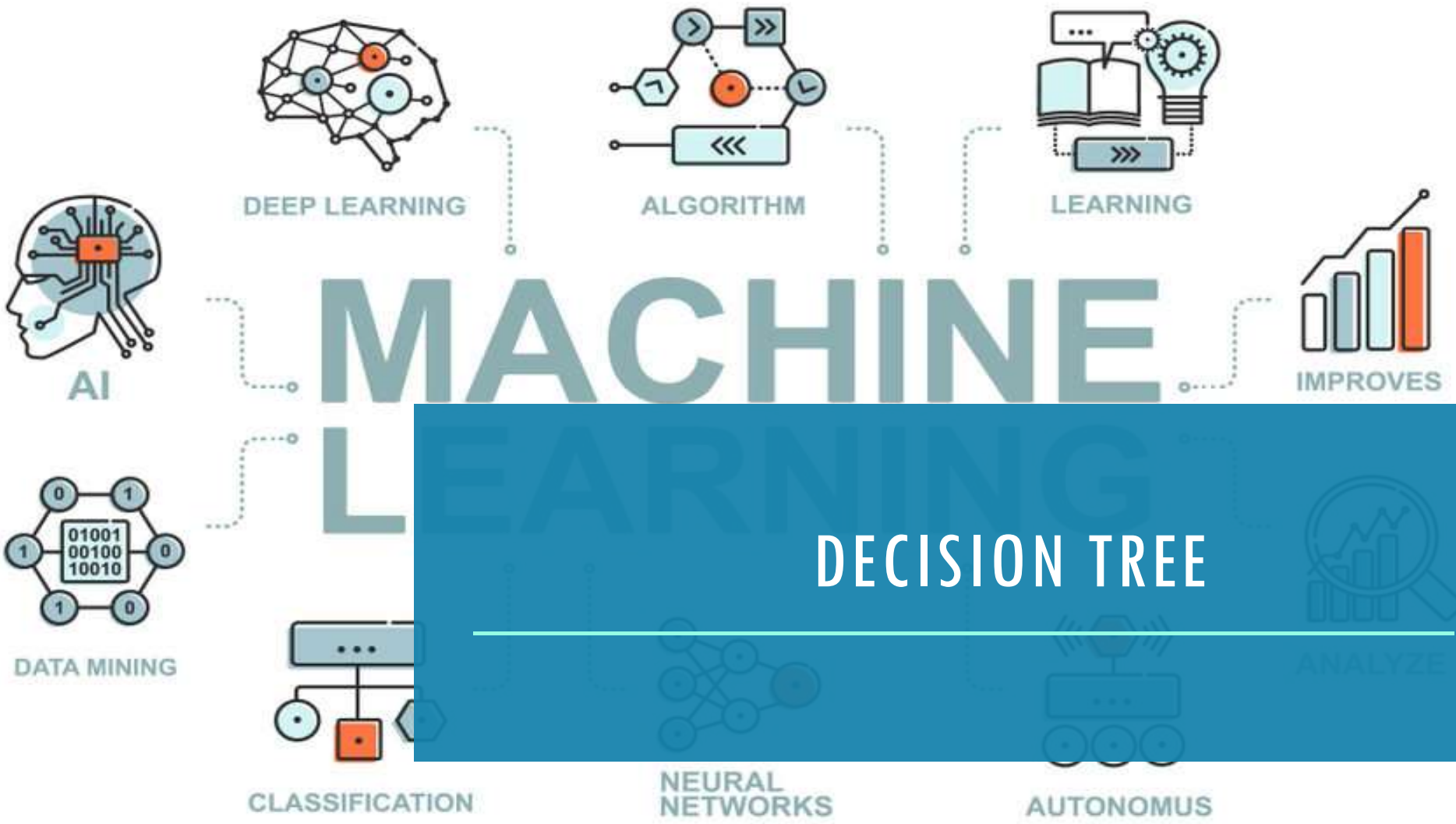


UNIT 4: DECISION TREES AND RANDOM FORESTS

Prof Mahendra Ugale



DECISION TREE

Decision Tree (Classification Use Case)

16 November 2022 10:09 AM

Binary classifier

o/p

Decision Tree:

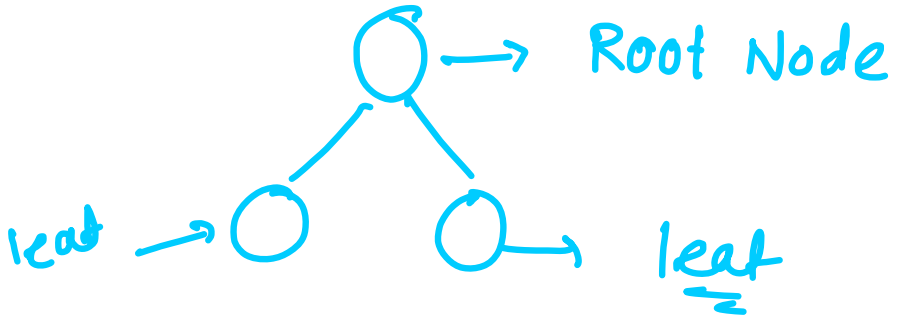
- Classification
- Regression

E.g:

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

→ Used to solve many usecase

→ If-then-else statement



Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No

(1) purity → a] Entropy

b] Gini impurity

D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

b] Gini impurity

② Root Node Selection → Information Gain

* Entropy: → two classes

$H(S) = -P_+ \log_2(P_+) - P_- \log_2(P_-)$

∴ is Binary classifier

① Checking purity of Node

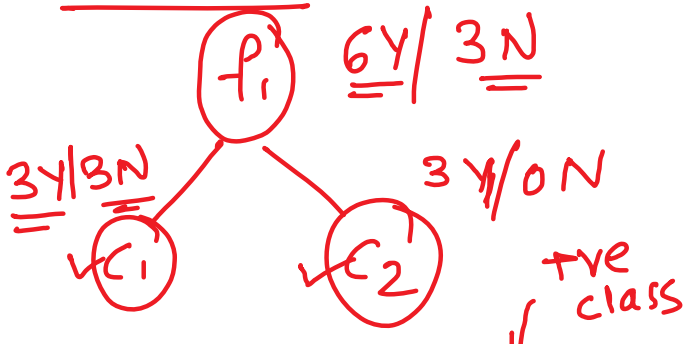
② How to select the feature/ Selection of Root Node

* Gini impurity :

$G.I. = 1 - \sum_{i=1}^{n} (P)^2$

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

Let say



D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$H(S) = -P_+ \log_2(P_+) - P_- \log_2(P_-)$

C_1 C_2 +ve class

$H(C_1) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right)$

+ve class -ve class

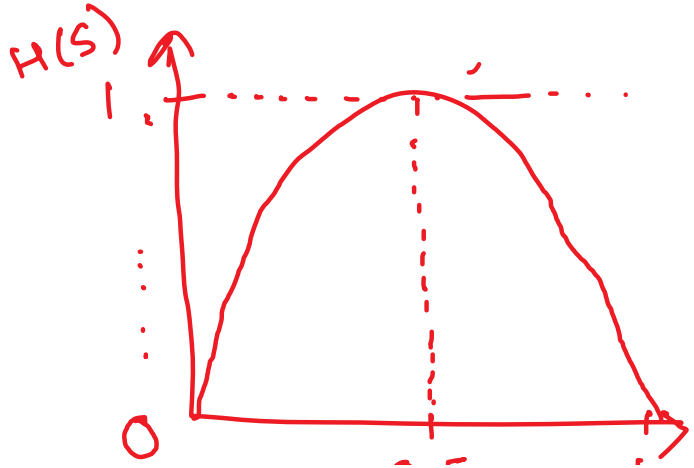
* $\{H(C_1) = 1\} \rightarrow$ (Impure Split)

* $H(C_2) = -\frac{3}{3}\log_2\left(\frac{3}{3}\right) - \frac{0}{3}\log_2\left(\frac{0}{3}\right)$

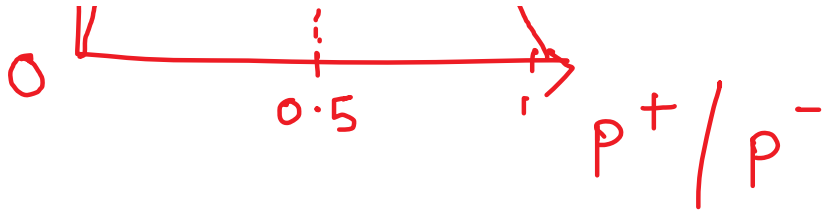
34 | ON

$\{H(C_2) = 0\} \Rightarrow$ pure split

* Entropy is lies in betn 0 & 1.

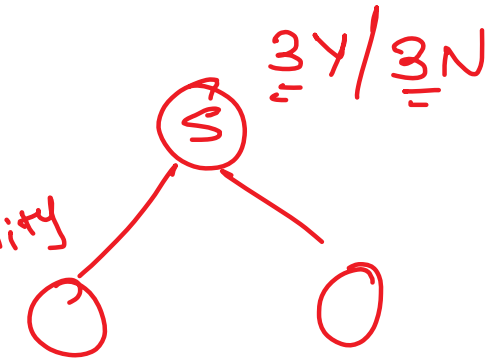


→ Entropy is lies in betn 0 & 1



② Gini impurity / Gini Index:

$$G.I = 1 - \sum_{i=1}^{n} (p)^2$$



$$G.I = 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{3}{6}\right)^2\right)$$

↳ -ve class probability

$$= 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right)$$

$$= 1 - 1/2 = 0.5$$

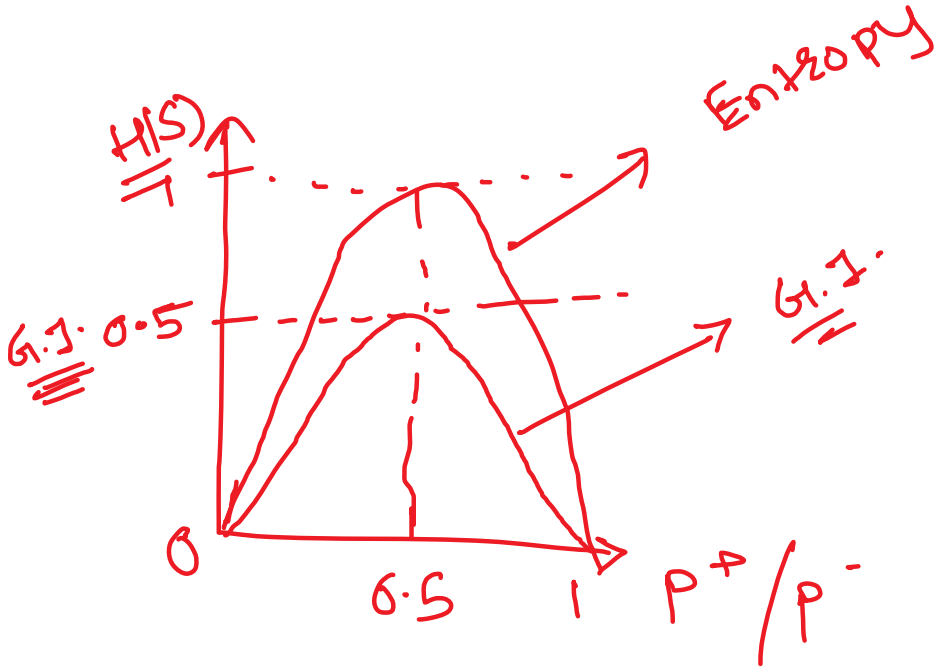
Entropy - 3Y/3N

(:) 1

G.I : 3Y/3N

(:) 0.5

G.I = 0.5 → (Impure Split)



Entropy	G I
① Entropy lies in betn 0 & 1	① G.I. lies in betn 0 & 0.5
② Entropy is 0 then pure split	② if G.I is 0 then pure split
③ Entropy is 1 then impure split	③ if G.I is 0.5 then impure split

Split

* Information Gain :→ To find Root Node

$$\{Gain(S, f_1) = H(S) - \sum_{v \in Val} \frac{|S_v|}{|S|} H(S_v)\}$$

Root → f_1 → 9Y|5N

6Y|2N → C_1

3Y|3N → C_2

$H(S) = -P_+ \log_2(P_+) - P_- \log_2(P_-)$

Root Node

$$* H(S) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right)$$

$\{H(S) \simeq 0.94\}$ $\{H(S) : 0.94\}$

$$H(C_1) = -\frac{6}{8}\log_2\left(\frac{6}{8}\right) - \frac{2}{8}\log_2\left(\frac{2}{8}\right)$$

$H(S_{V_1})$

$H(C_1) = 0.81$

3Y|3N

$*H(C_2) = 1$

f1 9Y|5N

6Y|2N → C_1 → 3Y|3N → C_2

$H(S) = 0.94$

$\{Gain(S, f_1) = H(S) - \sum_{v \in Val} \frac{-|S_v|}{|S|} * H(S_v)\}$

$= 0.94 - \left[\frac{8}{14} * 0.81 + \frac{6}{14} * 1\right]$

$\frac{8}{14}$ → C_1 observation / total observation

0.81 → $H(C_1)$

$\frac{6}{14}$ → C_2 observation / total observation

1 → $H(C_2)$

$Gain(S, f_1) = 0.049$

f_1 f_2 f_3

Result Interpretation: →

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

$Gain(S, f_1) = 0.049$

$Gain(S, f_2) = 0.051$

$Gain(S, f_3) = 0.061$

$Gain(S, f_3) \gg Gain(S, f_2) \gg Gain(S, f_1)$

As Gain of feature f_3 is Maxm so Root Node for Decision Tree is "feature f_3."

---

* When to use Gini ✓ when to use entropy (

* When to use Gini & when to use

→ {Entropy = $-P_+ \log_2(P_+) - P_- \log_2(P_-)$}

{G.I = $1 - \sum_{i=1}^{n} (P)^2$}

→ Max^m Time is taken by entropy to check purity

→ If No. of features are max^m → then use Gini Index.

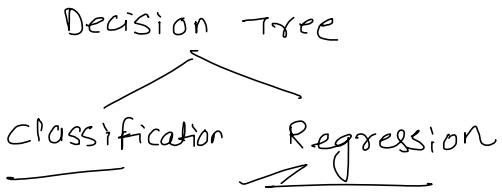
→ If No. of feature are less → then use entropy

→ By Default, DT uses Gini index to check purity.

Decision Tree Day 2

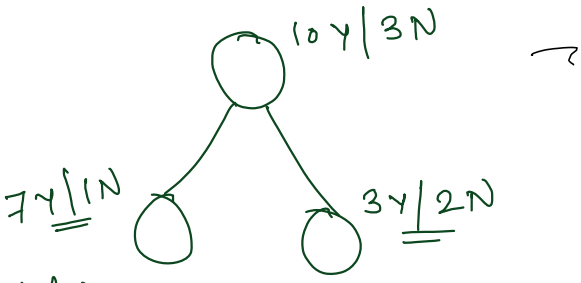
21 November 2022 10:24 AM

Decision Tree:



Classification

(Binary classifier)



Binary classifier:

① Purity

② from which Node (Root Node)

* Purity of Node:

↳ Entropy - 1 → Impure split, 0 → pure split

↳ Gini index - 0.5 → Impure split, 0 → pure split

* Selection of Root Node: →

Information Gain

↑ (Highest info. Gain) ⇒ Root Node

2] Decision Tree Regressor: → (DTR)

Regression
↳ o/p column
is continuous

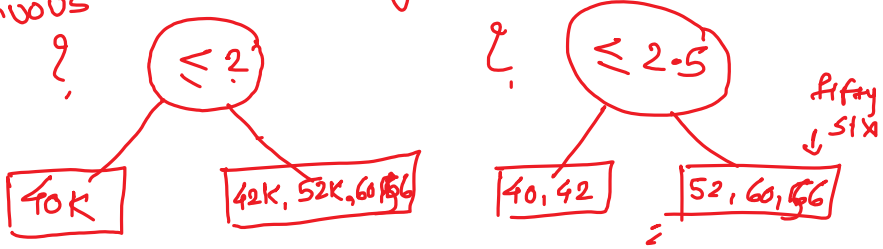
Dataset:

Exp	Gap	Salary → o/p
2	yes	40K
2.5	Yes	42K
3	NO	52K
4	NO	60K
4.5	Yes	56K

o/p is continuous

⇒ Sort columns (Exp)

(Already sorted)

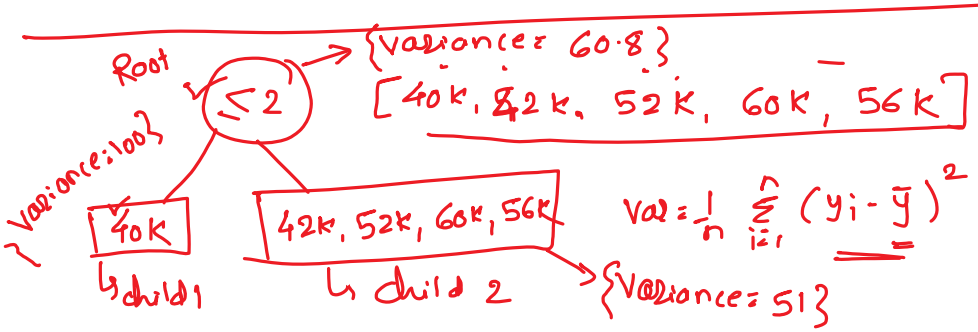


which to split?

Concept: → {Variance Reduction}

{ Variance - $\frac{1}{n}\sum (y_i - \bar{y})^2$ $\bar{y}$ = Average o/p }

{ Variance (MSE) $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$ $\bar{y}$ = Average o/p }



Var $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$

Exp	Gap	Salary
2	Yes	40k
2.5	Yes	42k
3	No	52k
4	No	60k
4.5	Yes	56k

{Avg = 50k}

* Variance (Root) $= \frac{1}{5}[(40-50)^2 + (42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2]$

$= \frac{1}{5} * 304 \approx 60.8$

{V (Root Node) = 60.8}

V (child) 1 $= \frac{1}{1}[(40-50)^2]$

{V (child 1) = 100}

V (child 2) $= \frac{1}{4}[64 + 4 + 100 + 36]$

$= \frac{204}{4} = 51$

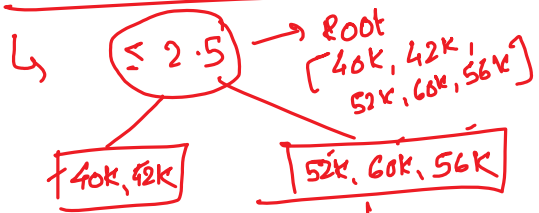
finally {V (Root) = 60.8, V (C1) = 100, V (C2) = 51}

* Variance Reduction: Var (Root) $- \sum w_i *$ Var (child)

$= 60.8 - [\frac{1}{5} * 100 + \frac{4}{5} * 51]$

$= 60.8 - 20 + 40.8 \approx 0$

(≤2) → {Variance Reduction = 0}



Var (Root) = 60.8

Var (C1) $= \frac{1}{2}[100 + 64] = 82$

Var (C2) $= \frac{1}{3}[4 + 100 + 36] = 46.66$

[40K, 42K] → C1 → Var (C1) = 82

[52K, 60K, 36K] → C2 → Var (C2) = 46.66

$Var(C_2) = \frac{1}{3}[4 + 100 + 36] = 46.66$

Variance Reduction = Var(Root) − Σ Wi * Var(child)

$= 60 - 8 - \left[\frac{2}{5}(82) + \frac{3}{5}(46.66)\right]$

Variance Reduction = 0.304

≤ 2.5

Final conclusion:

{ Split (1) ≤ 2 } = Variance Reduction = 0

{ Split (2) ≤ 2.5 } ≃ Variance Reduction = 0.304

Note: → Split (2) Variance Reduction is greater than Split (1) Variance Reduction

---

Additional points for Decision Tree: →

Disadvantage of DT is "overfitting"

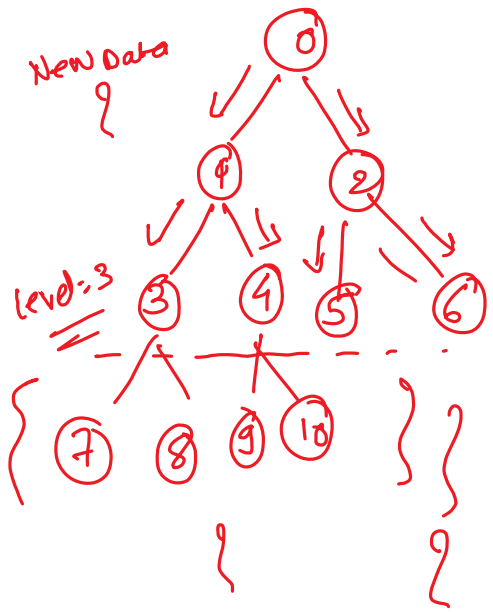
{ Training Data ↑ Testing Data ↓ } ⇒ overfitting

Hyperparameter: →

↳ {

To avoid overfitting;

(1) Post pruning → create DT &

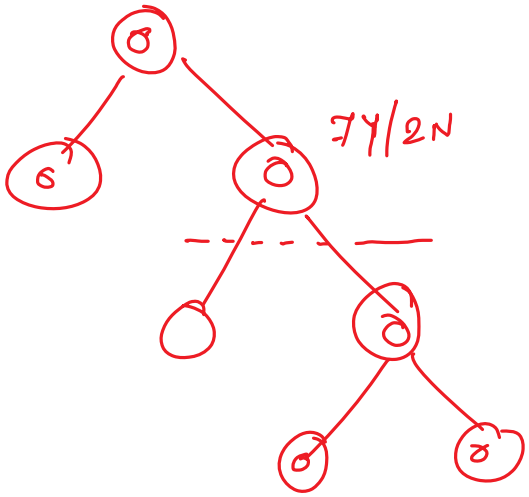


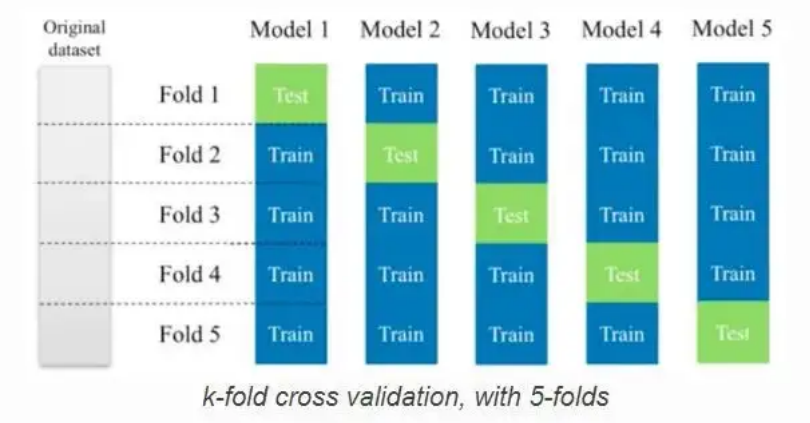
① post pruning → create DT & Delete / prune at specific condn

② pre pruning

↓

#hyperparameter





In [21]:

```
# import k-folder
from sklearn.model_selection import cross_val_score
```

In [22]:

```
# use the same model as before
knn = KNeighborsClassifier(n_neighbors = 5)
# X,y will automatically devided by 5 folder, the scoring I will still use the accuracy
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
# print all 5 times scores
print(scores)
# [ 0.96666667  1.          0.93333333  0.96666667  1.        ]
# then I will do the average about these five scores to get more accuracy score.
print(scores.mean())
# 0.973333333333
```

```
[0.96666667 1.         0.93333333 0.96666667 1.        ]
0.9733333333333334
```

Curse of Dimensionality

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Decision Tree

Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

Decision Tree Classification Algorithm / Decision Tree Classifier

It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node.

Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

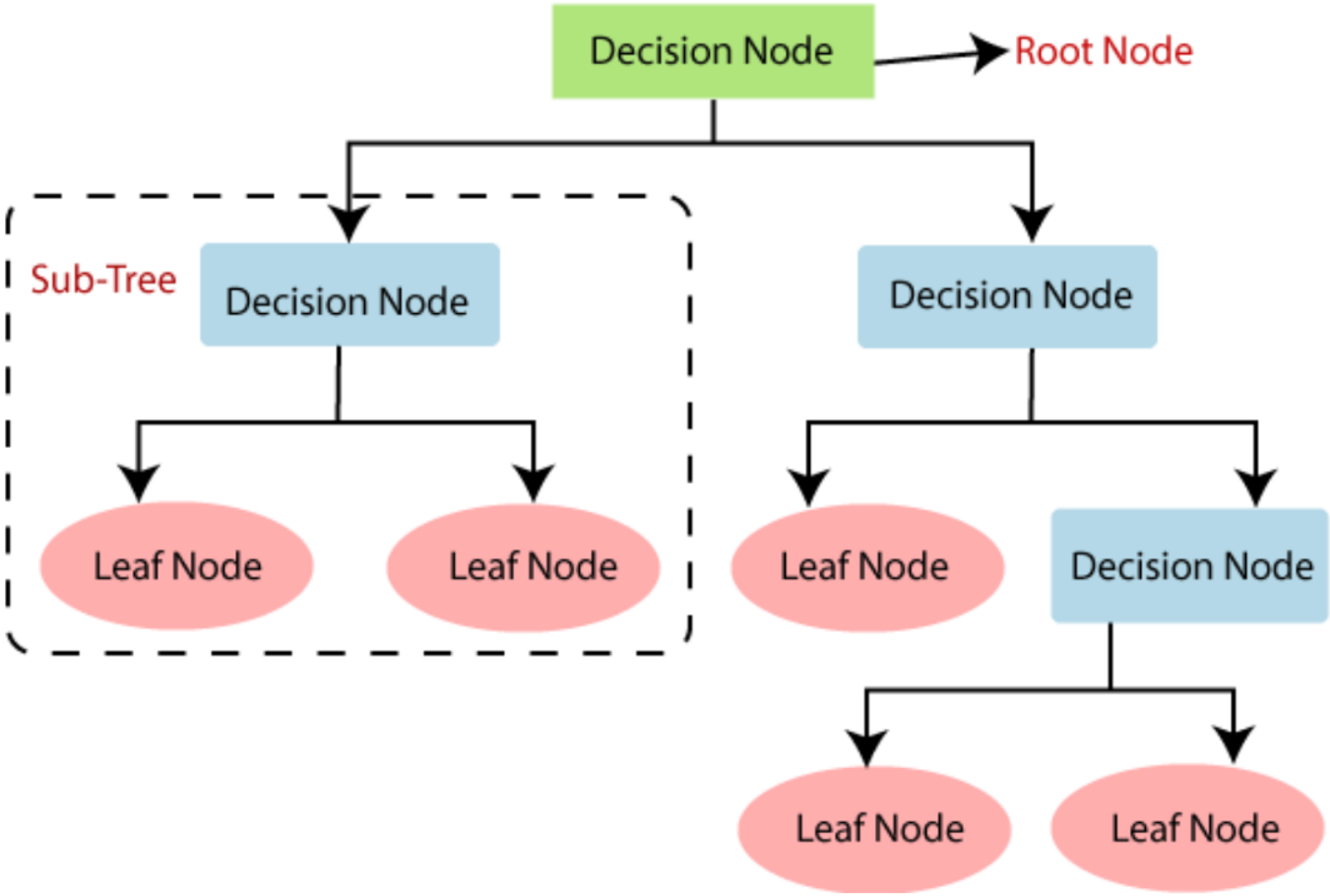
In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Below diagram explains the general structure of a decision tree:

Note:

A decision tree can contain categorical data (YES/NO) as well as numeric data.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

Root Node:

```
Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more h
omogeneous sets.
```

Leaf Node:

```
Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
```

Splitting:

```
Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
```

Branch/Sub Tree:

```
A tree formed by splitting the tree.
```

Pruning:

```
Pruning is the process of removing the unwanted branches from the tree.
```

Parent/Child node:

```
The root node of the tree is called the parent node, and other nodes are called the child nodes.
```

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).
```

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 r
eached where you cannot further classify the nodes and called the final node as a leaf node.
```

Example:

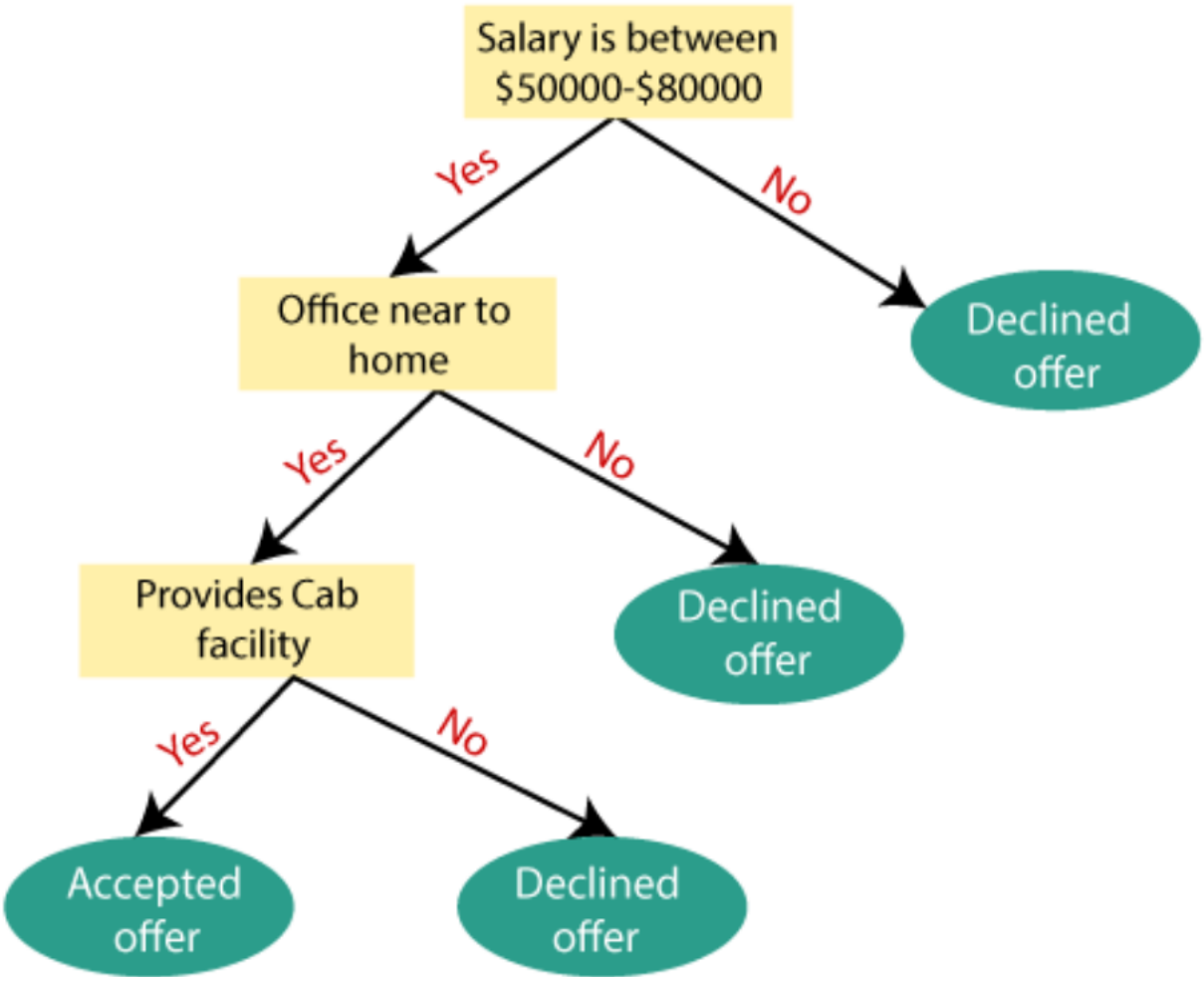
Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM).

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:



Decision Trees

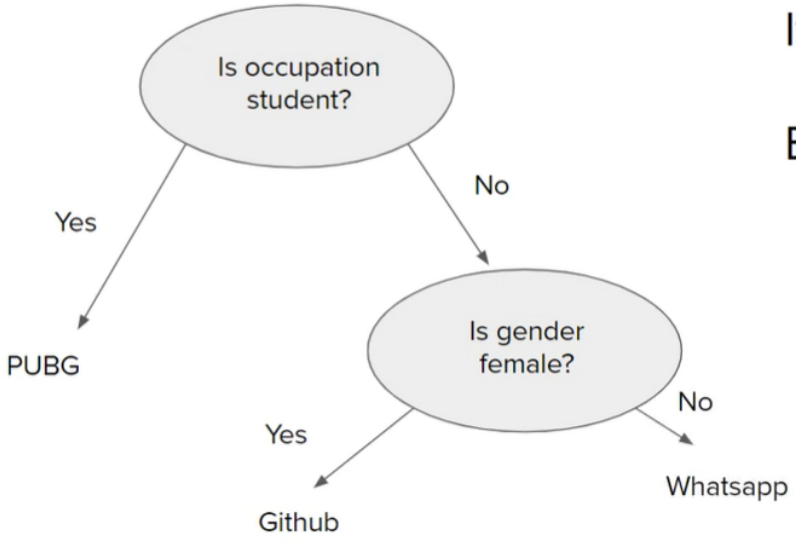
Introduction and Geometric Intuition

Example 1

Gender	Occupation	Suggestion
F	Student	PUBG
F	Programmer	Github
M	Programmer	Whatsapp
F	Programmer	Github
M	Student	PUBG
M	Student	PUBG

```
If occupation==student
    print(PUBG)
Else
    If gender==female
        print(Github)
    Else
        print(Whatsapp)
```

Where is the Tree?

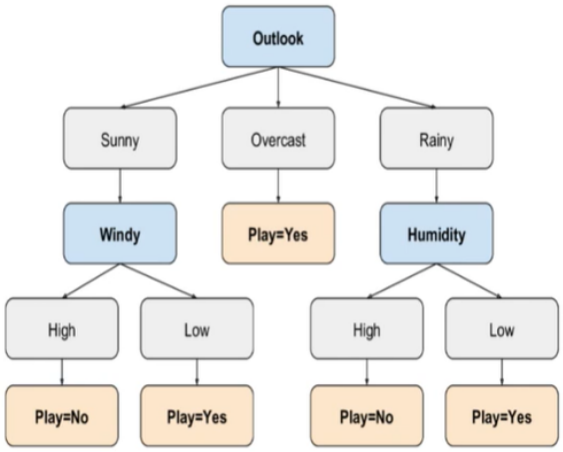


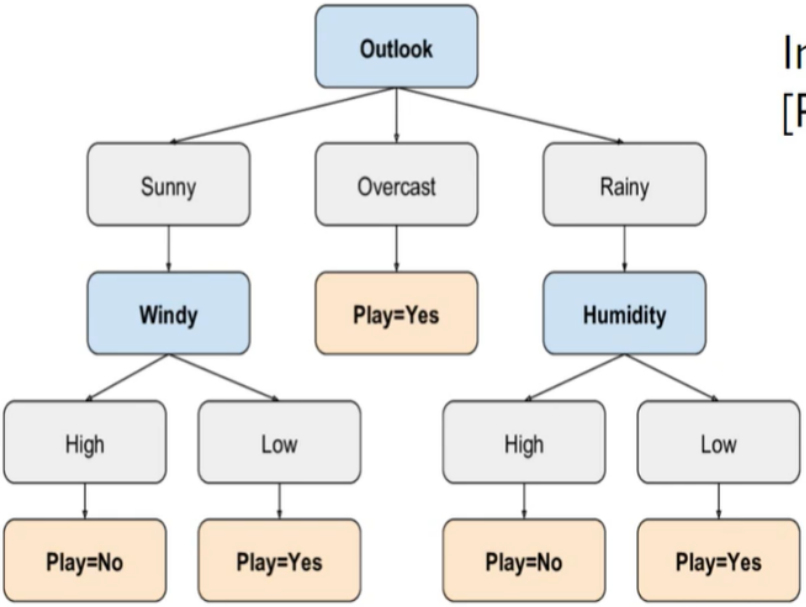

```
If occupation==student
    print(PUBG)
Else
    If gender==female
        print(Github)
    Else
        print(Whatsapp)
```

#

Example 2

Day	Outlook	Temp	Humid	Wind	Play?
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

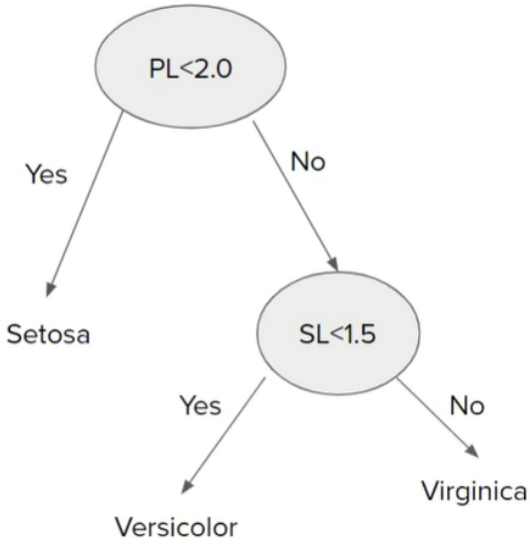




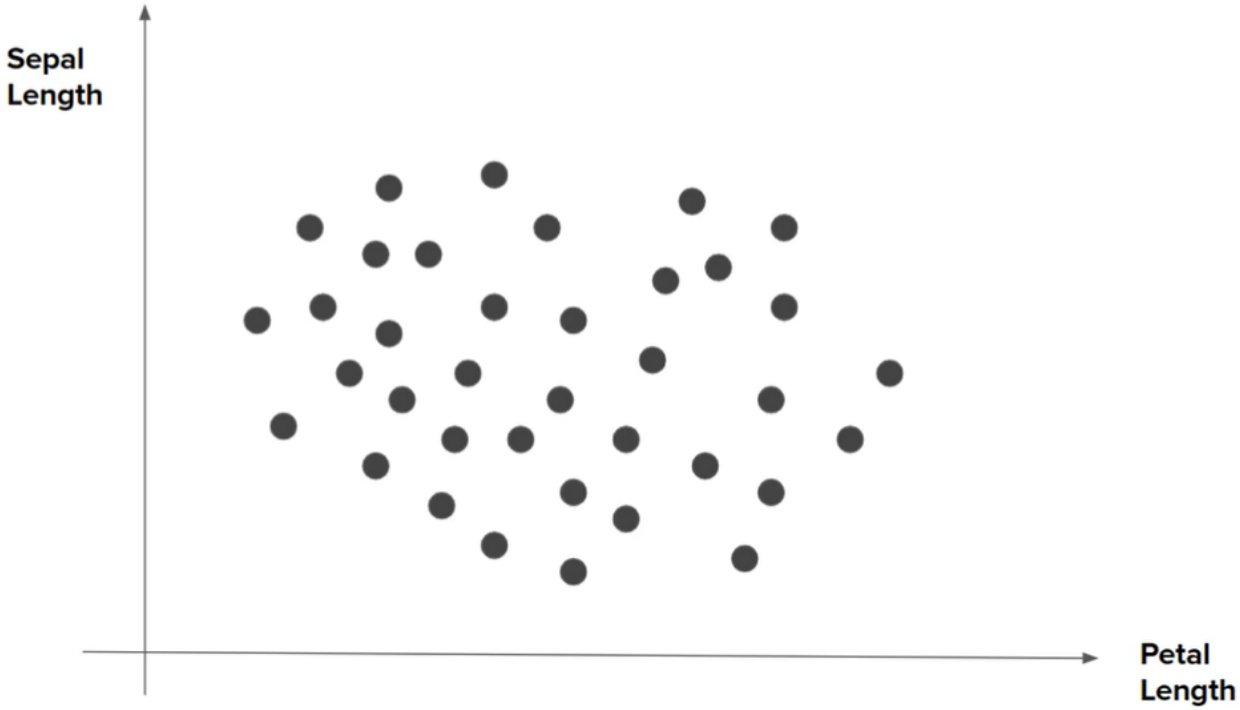
Input query point:
[Rainy, Mild, High, Strong]

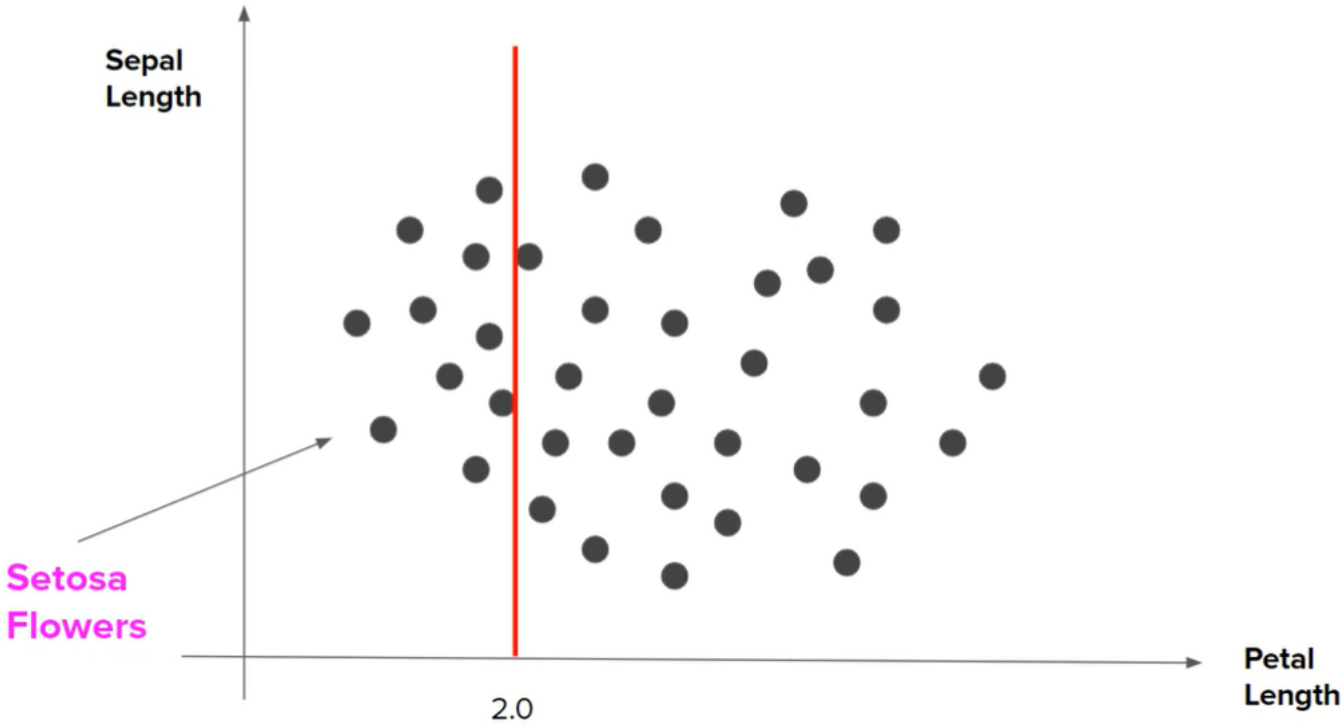
What if we have numerical data?

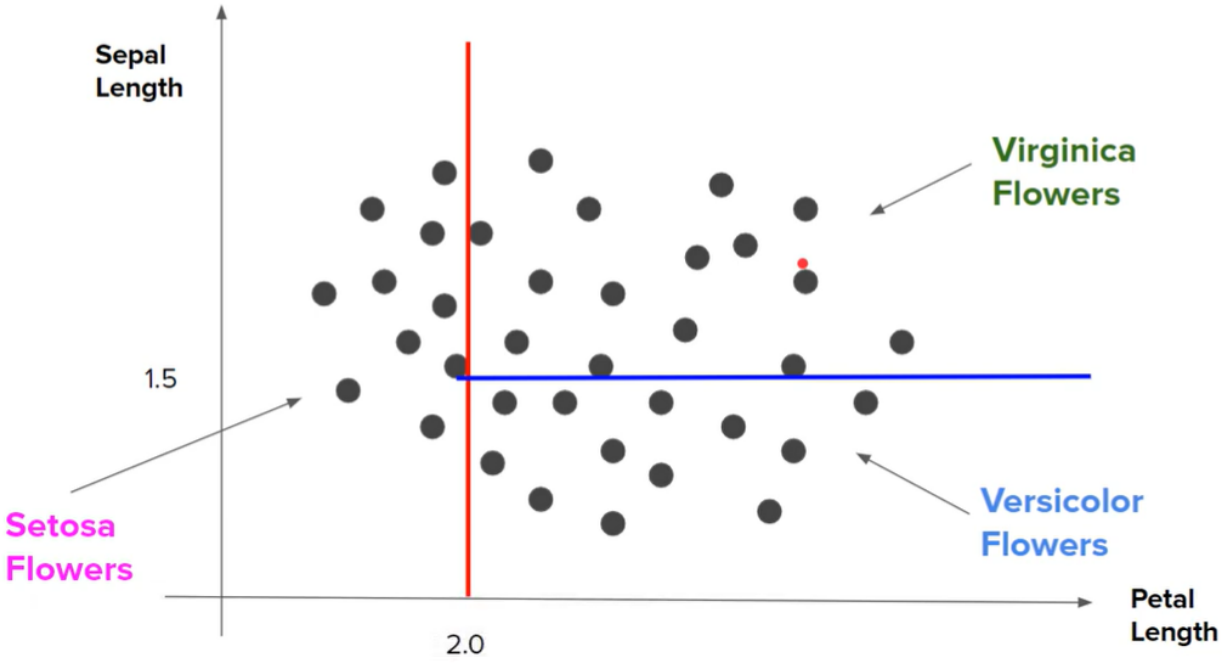
Petal Length	Sepal Length	Type
1.34	0.34	Setosa
3.45	1.45	Versicolor
1.69	0.98	Setosa
2.56	1.79	Virginica
3.00	1.13	Versicolor
1.3	0.88	Setosa



Geometric Intuition




Sepal
Length
Setosa
Flowers
Petal
Length
2.0


Sepal
Length
Virginica
Flowers
1.5
Setosa
Flowers
Versicolor
Flowers
Petal
Length
2.0

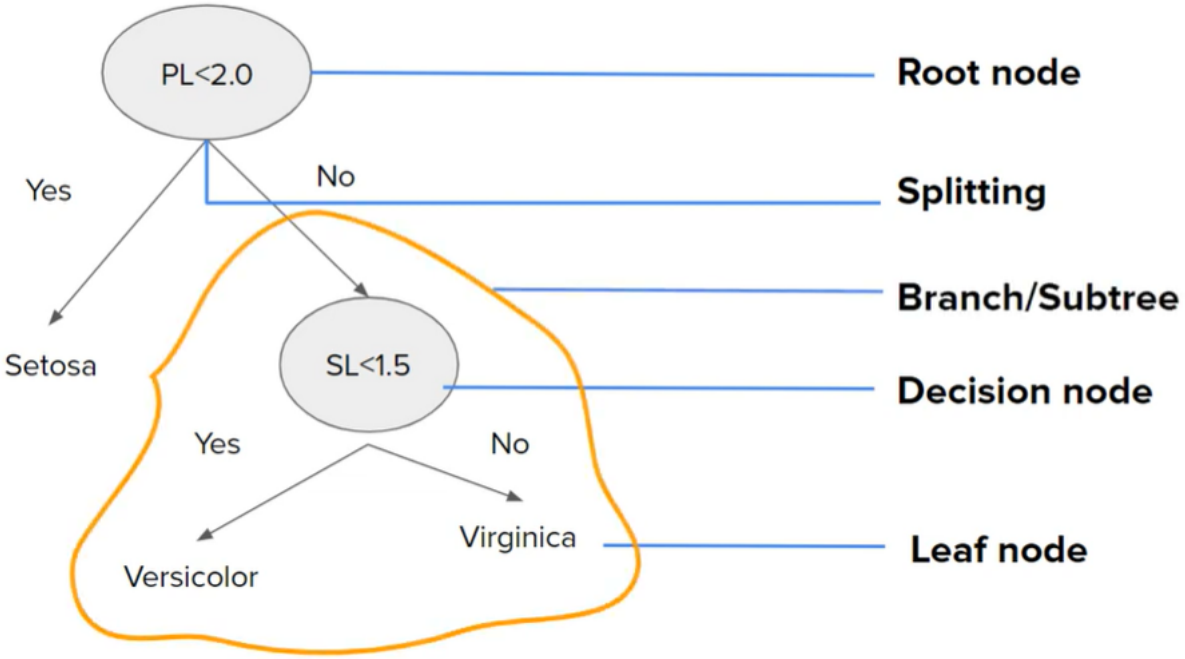
Pseudo code

- Begin with your training dataset, which should have some feature variables and classification or regression output.
- Determine the “best feature” in the dataset to split the data on; more on how we define “best feature” later
- Split the data into subsets that contain the correct values for this best feature. This splitting basically defines a node on the tree i.e each node is a splitting point based on a certain feature from our data.
- Recursively generate new tree nodes by using the subset of data created from step 3.

Programatically speaking, Decision trees are nothing but a giant structure of nested if-else condition

Mathematically speaking, Decision trees use hyperplanes which run parallel to any one of the axes to cut your coordinate system into hyper cuboids

Terminology



Some unanswered questions

How to decide which column should be considered as root node?

How to select subsequent decision nodes?

How to decide splitting criteria in case of numerical columns?

Advantages	Disadvantages
Intuitive and easy to understand	Overfitting
Minimal data preparation is required	Prone to errors for imbalanced datasets
The cost of using the tree for inference is **logarithmic** in the number of data points used to train the tree	

CART - Classification and Regression Trees

The logic of decision trees can also be applied to regression problems, hence the name CART

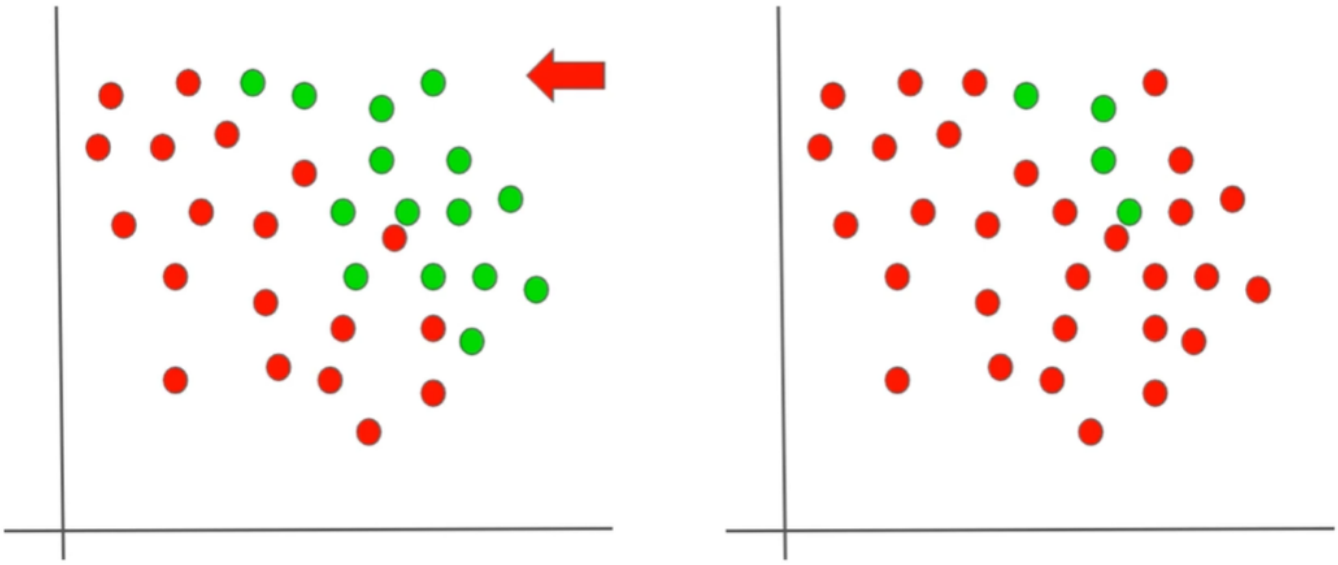
Entropy in Decision Trees

Decision Trees

Entropy

What is Entropy?

In the most layman terms, Entropy is nothing but the measure of disorder. Or you can also call it the measure of purity/impurity. Let's see an example...



More knowledge less entropy

How to calculate Entropy?

The mathematical formula for entropy is:

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

Where 'Pi' is simply the frequentist probability of an element/class 'i' in our data.

For e.g if our data has only 2 class labels **Yes** and **No.**

$$E(D) = -p_{yes}\log_2(p_{yes}) - p_{no}\log_2(p_{no})$$

Example - Dataset

Salary	Age	Purchase
20000	21	Yes
10000	45	No
60000	27	Yes
15000	31	No
12000	18	No

Salary	Age	Purchase
34000	31	No
15000	25	No
69000	57	Yes
25000	21	No
32000	28	No

$H(d) = -P_y\log_2(P_y) - P_n\log_2(P_n)$

$H(d) = -2/5\log_2(2/5) - 3/5\log_2(3/5)$

$H(d) = 0.97$

$H(d) = -P_y\log_2(P_y) - P_n\log_2(P_n)$

$H(d) = -1/5\log_2(1/5) - 4/5\log_2(4/5)$

$H(d) = 0.72$

Calculating entropy for a 3 class problem

Salary	Age	Purchase
20000	21	Yes
10000	45	No
60000	27	Yes
15000	31	No
30000	30	Maybe
12000	18	No
40000	40	Maybe
20000	20	Maybe

$$H(d) = - P_y \log_2(P_y) - P_n \log_2(P_n) - P_m \log_2(P_m)$$

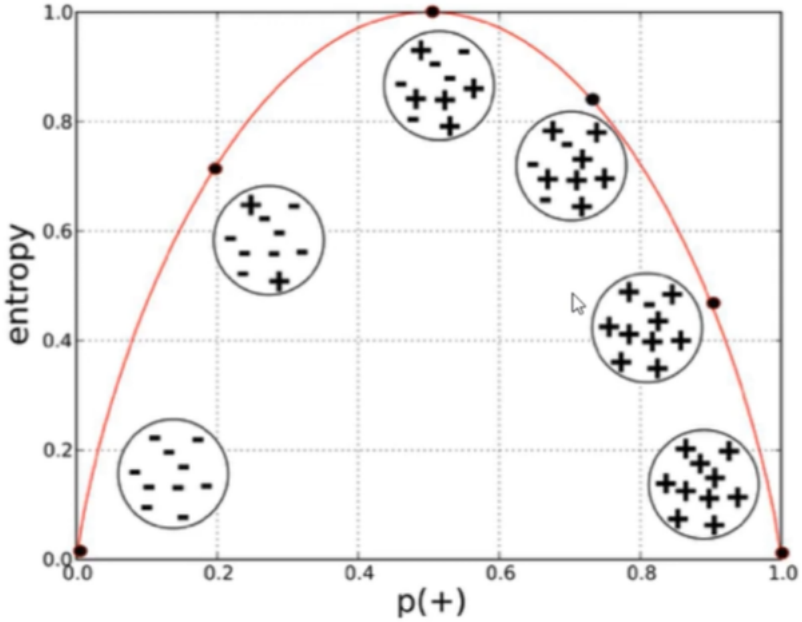
$$H(d) = -2/8 \log_2(2/8) - 3/8 \log_2(3/8) - 3/8 \log_2(3/8)$$

$$H(d) = 1.56$$

Observation

- More the uncertainty more is entropy
- For a 2 class problem the min entropy is 0 and the max is 1
- For more than 2 classes the min entropy is 0 but the max can be greater than 1
- Both $\log_2$ or $\log_e$ can be used to calculate entropy

Entropy Vs Probability



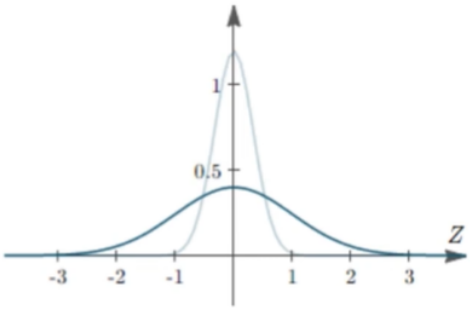
Entropy for continuous variables

Area	Built in	Price
1200	1999	3.5
1800	2011	5.6
1400	2000	7.3
...	...	...

Dataset 1

Area	Built in	Price
2200	1989	4.6
800	2018	6.5
1100	2005	12.8
...	...	...

Dataset 2



Quiz: Which of the above datasets have higher entropy?

Ans: Whichever is less peaked

Information Gain

Information Gain, is a metric used to train Decision Trees. Specifically, this metric measures the quality of a split.

The information gain is based on the decrease in entropy after a data-set is split on an attribute.Constructing a decision tree is all about finding attribute that returns the highest information gain

Information Gain = E(Parent) - {Weighted Average} * E{Children}

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Step 1:

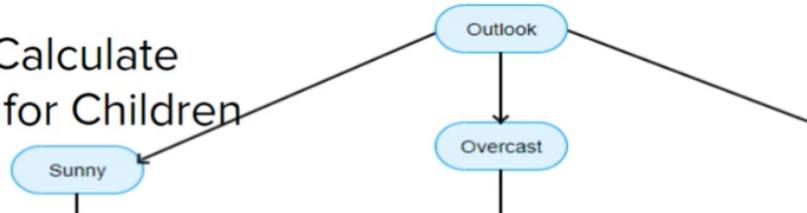
Entropy of Parent

$E(P) = -p_y \log_2(p_y) - p_n \log_2(p_n)$

$= 9/14 \log_2(9/14) - 5/14 / \log_2(5/14)$

$E(P) = \mathbf{0.94}$

Step 2: Calculate Entropy for Children



Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes

Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

Day	Outlook	Humidity	Wind	Play
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D10	Rain	Normal	Weak	Yes
D14	Rain	High	Strong	No

E(S) =	E(O) =	E(R) =
-2/5log(2/5) - 3/5log(3/5)	-5/5log(5/5) - 0/5log(0/5)	-3/5log(3/5) - 2/5log(2/5)
E(S)= 0.97	E(O)= 0	E(S)= 0.97

Step 3 : Calculate weighted Entropy of Children

Weighted Entropy = 5/14 * 0.97 + 4/14 * 0 + 5/14 * 0.97

W.E(Children) = **0.69**

P(Overcast) is a leaf node as it's entropy is 0

Step 4 : Calculate Information Gain

Information Gain = E(Parent) - {Weighted Average} * E(Children)

IG = **0.97 - 0.69 = 0.28**

So the information gain(or the decrease in entropy/impurity) when you split this data on the basis of **Outlook** condition/column is **0.28**

Step 5 : Calculate Information Gain for all the columns

Whichever column has the highest Information Gain(maximum decrease in entropy) the algorithm will select that column to split the data.

Step 6 : Find Information Gain recursively

Decision tree then applies a recursive greedy search algorithm in top bottom fashion to find Information Gain at every level of the tree.

Once a leaf node is reached (Entropy = 0), no more splitting is done.

Gini Impurity

Decision Trees

Gini Impurity

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

Gini Index= 1- ∑jPj2

Example - Dataset

Salary	Age	Purchase
20000	21	Yes
10000	45	No
60000	27	Yes
15000	31	No
12000	18	No

Salary	Age	Purchase
34000	31	No
15000	25	No
69000	57	Yes
25000	21	No
32000	28	No

$G = 1 - (Py^2 + Pn^2)$

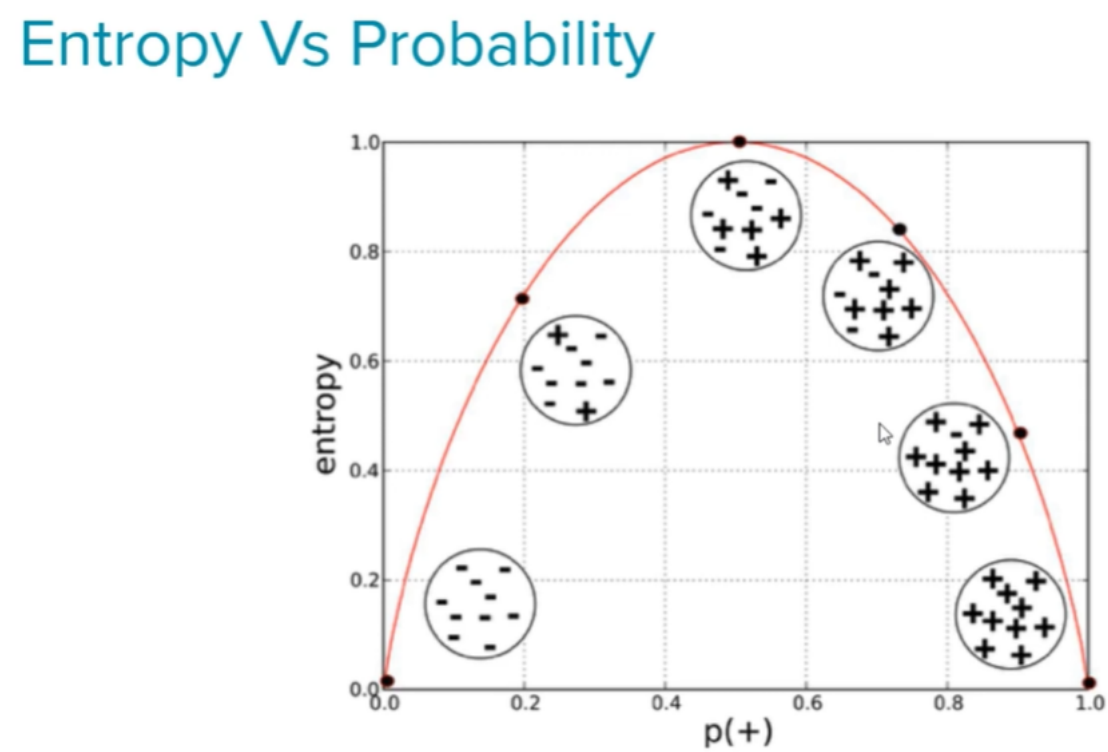
$G = 1 - (4/25 + 9/25)$

$G = 0.48$

$G = 1 - (Py^2 + Pn^2)$

$G = 1 - (1/25 + 16/25)$

$G = 0.32$



Handling Numerical Values in Decison Trees

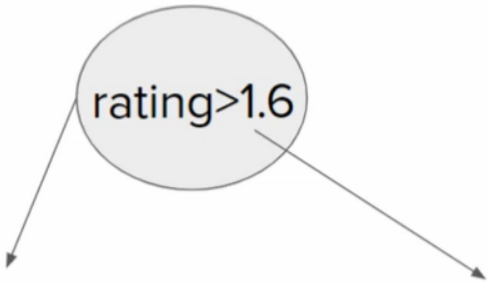
S No	User Rating	Downloaded
1	3.5	Yes
2	4.6	Yes
3	2.2	No
4	1.6	Yes
5	4.1	No
6	3.9	No
7	3.2	No
9	2.9	Yes
10	4.8	Yes
11	3.3	No
12	2.5	Yes
13	1.9	Yes

Step 1:
Sort the data on the basis of numerical column

S No	User Rating	Downloaded
1	1.6	Yes
2	1.9	Yes
3	2.2	No
4	2.5	Yes
5	2.9	Yes
6	3.2	No
7	3.3	No
9	3.5	Yes
10	3.9	No
11	4.1	No
12	4.6	Yes
13	4.8	Yes

Step 2:

Split the entire data on the basis of every value of user_rating

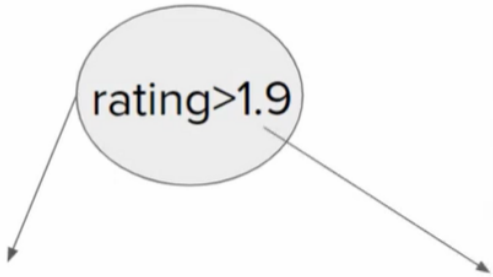


S No	User Rating	Downloaded
1	1.6	Yes

S No	User Rating	Downloaded
2	1.9	Yes
3	2.2	No
4	2.5	Yes
5	2.9	Yes
6	3.2	No
7	3.3	No
9	3.5	Yes
10	3.9	No
11	4.1	No
12	4.6	Yes
13	4.8	Yes

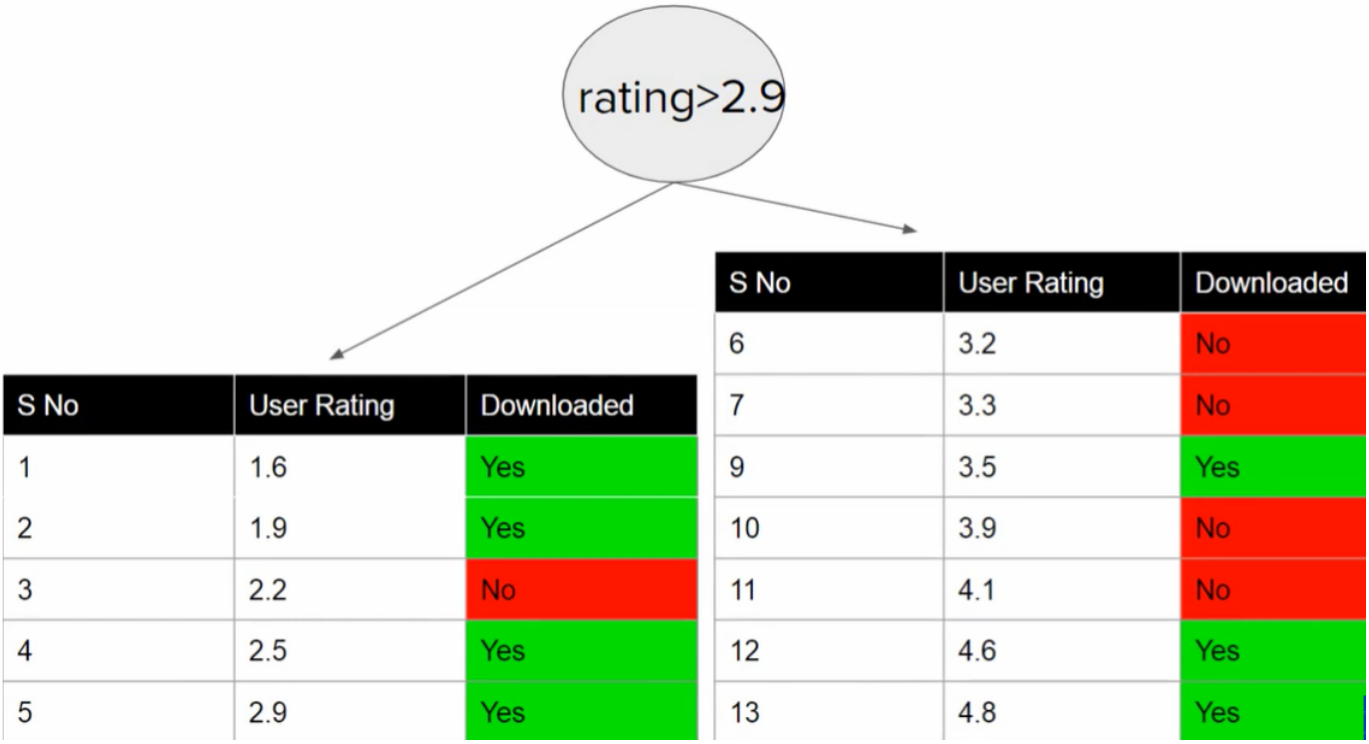
Step 2:

Split the entire data on the basis of every value of user_rating



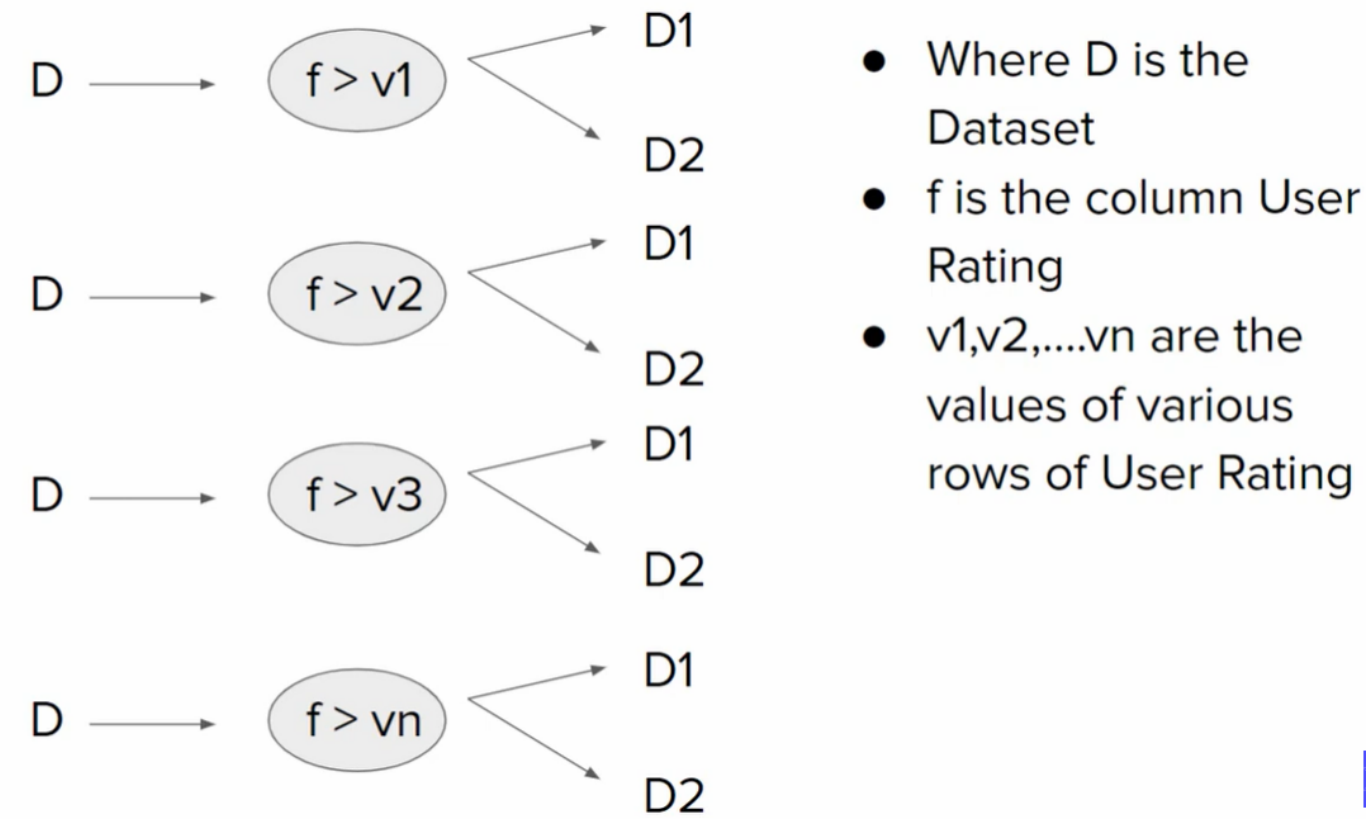
S No	User Rating	Downloaded
1	1.6	Yes
2	1.9	Yes

S No	User Rating	Downloaded
3	2.2	No
4	2.5	Yes
5	2.9	Yes
6	3.2	No
7	3.3	No
9	3.5	Yes
10	3.9	No
11	4.1	No
12	4.6	Yes
13	4.8	Yes



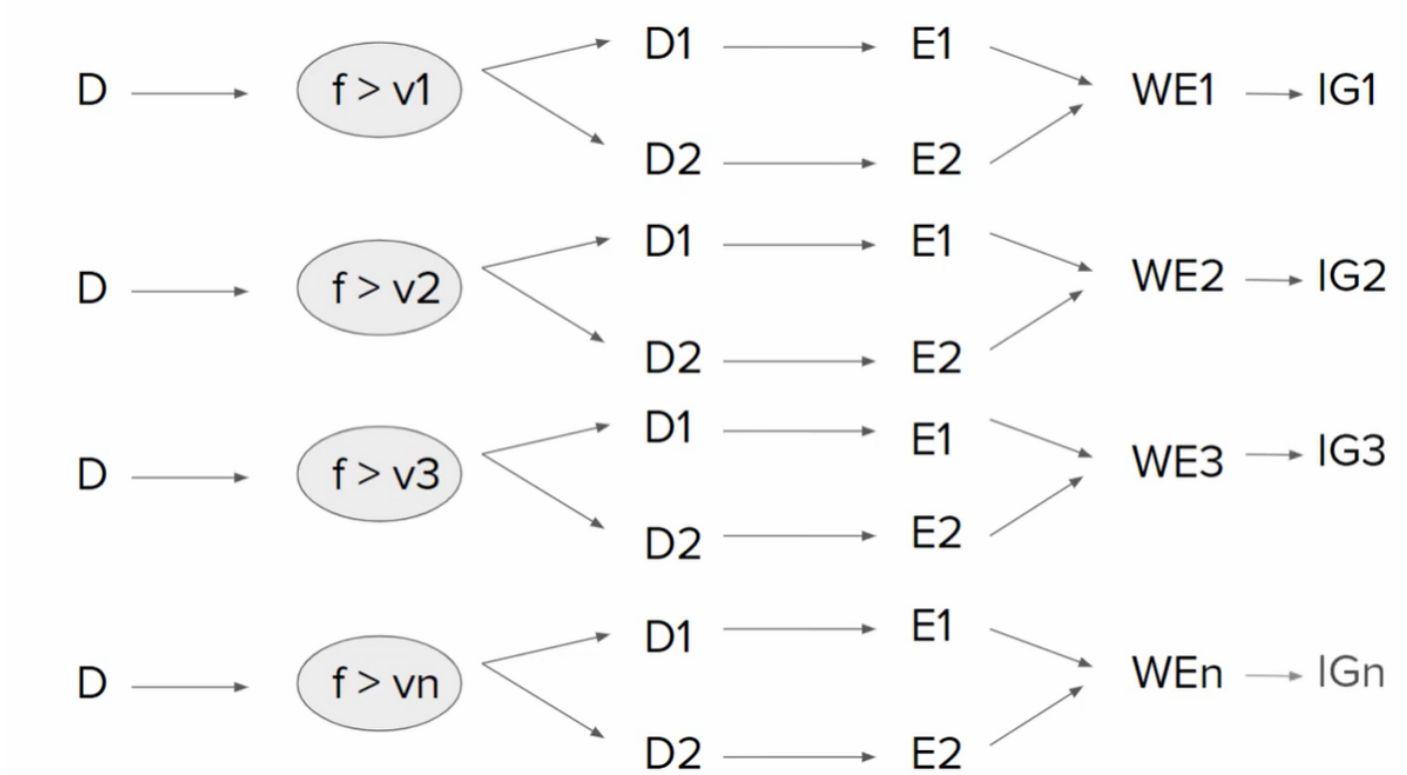
S No	User Rating	Downloaded
1	1.6	Yes
2	1.9	Yes
3	2.2	No
4	2.5	Yes
5	2.9	Yes

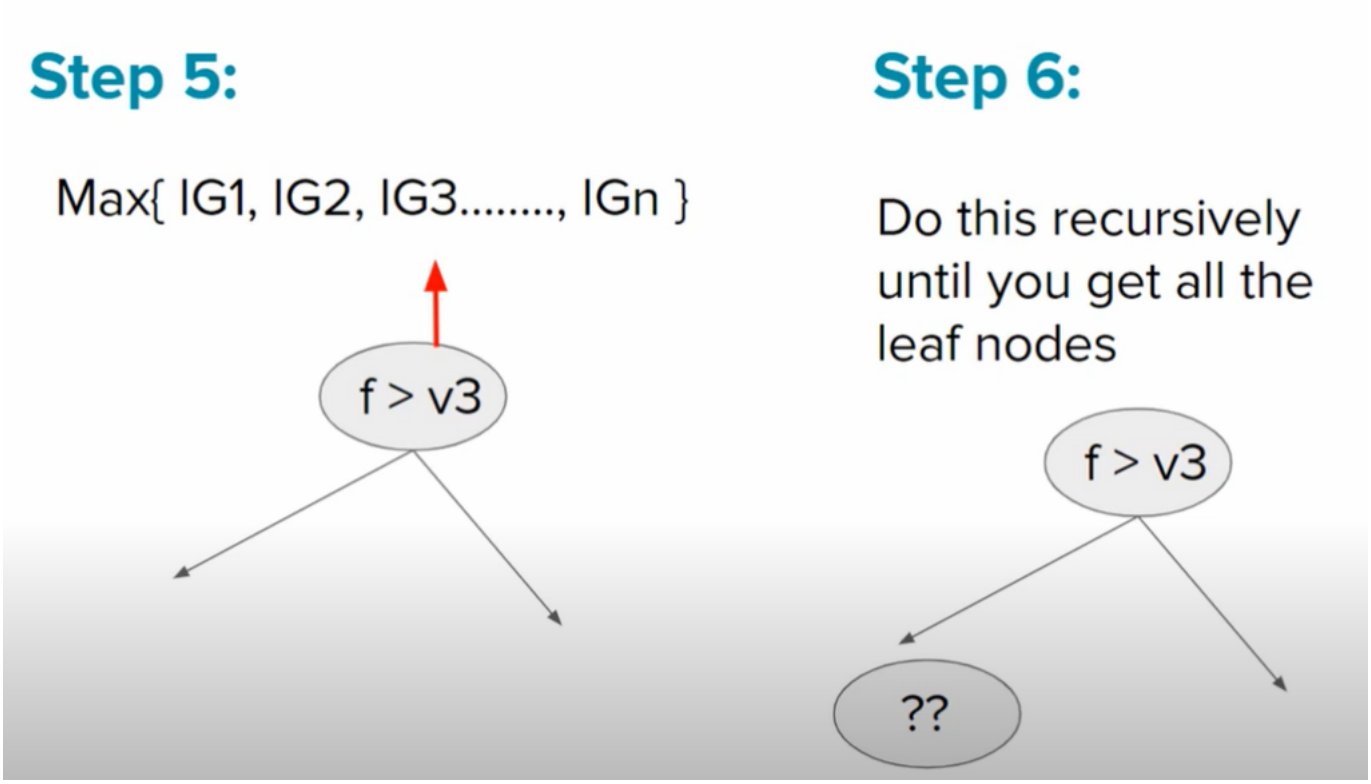
S No	User Rating	Downloaded
6	3.2	No
7	3.3	No
9	3.5	Yes
10	3.9	No
11	4.1	No
12	4.6	Yes
13	4.8	Yes



- Where D is the Dataset
- f is the column User Rating
- v1,v2,....vn are the values of various rows of User Rating

In [ ]:





Visualizing a Decision Tree- Decision Tree Classification Python Code

In [23]:

```
from sklearn.datasets import load_iris
```

In [25]:

```
iris=load_iris()
```

In [28]:

```
X=iris.data
X
```

Out[28]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
```

In [27]:

```
y=iris.target
y
```

Out[27]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [29]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2,random_state=0)
```

In [30]:

```
X_train.shape
```

Out[30]:

```
(120, 4)
```

In [31]:

```
X_test.shape
```

Out[31]:

```
(30, 4)
```

In [32]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [33]:

```
clf=DecisionTreeClassifier()
```

In [34]:

```
clf.fit(X_train,y_train)
```

Out[34]:

```
DecisionTreeClassifier()
```

In [35]:

```
y_pred=clf.predict(X_test)
```

In [37]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[37]:

```
1.0
```

In [39]:

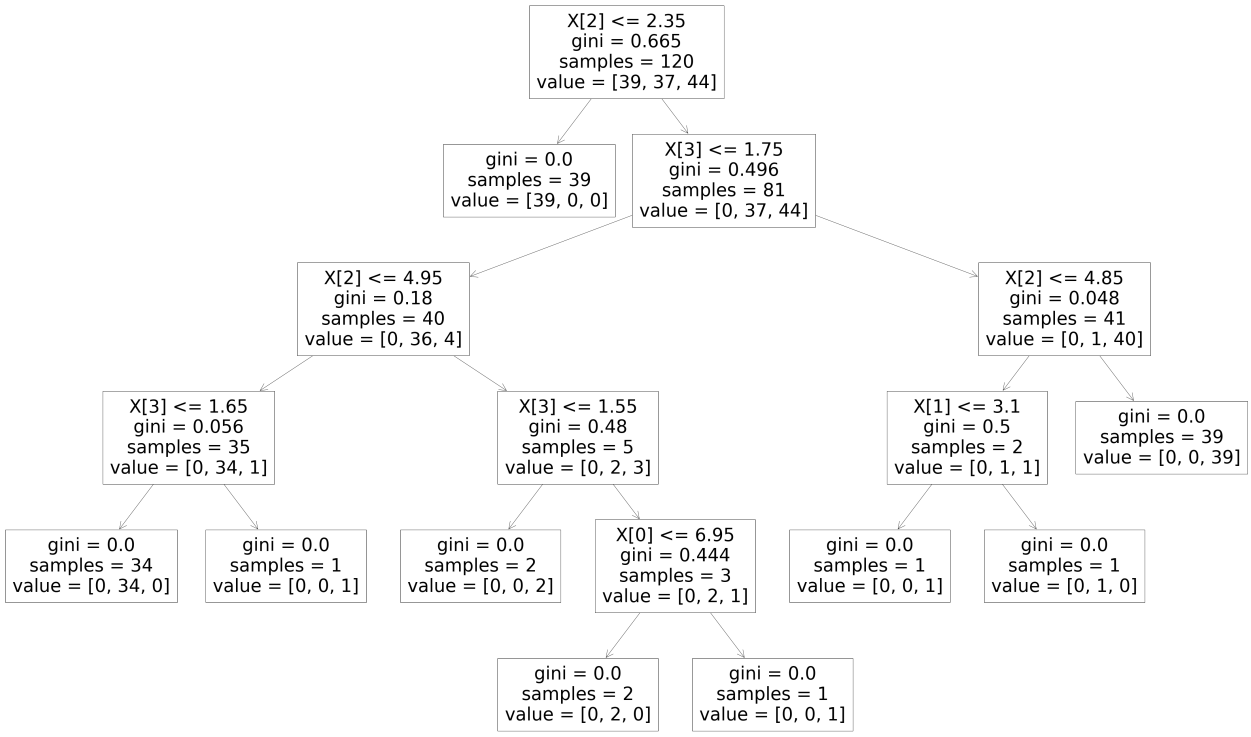
```
from sklearn.tree import plot_tree
```

In [41]:

```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 80,50
plot_tree(clf)
```

Out[41]:

```
[Text(0.5, 0.9166666666666666, 'X[2] <= 2.35\ngini = 0.665\nsamples = 120\nvalue = [39, 37, 44]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'X[3] <= 1.75\ngini = 0.496\nsamples = 81\nvalue = [0, 37, 44]'),
 Text(0.3076923076923077, 0.5833333333333334, 'X[2] <= 4.95\ngini = 0.18\nsamples = 40\nvalue = [0, 36, 4]'),
 Text(0.15384615384615385, 0.4166666666666667, 'X[3] <= 1.65\ngini = 0.056\nsamples = 35\nvalue = [0, 34, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 34\nvalue = [0, 34, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'X[3] <= 1.55\ngini = 0.48\nsamples = 5\nvalue = [0, 2, 3]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.5384615384615384, 0.25, 'X[0] <= 6.95\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'X[2] <= 4.85\ngini = 0.048\nsamples = 41\nvalue = [0, 1, 40]'),
 Text(0.7692307692307693, 0.4166666666666667, 'X[1] <= 3.1\ngini = 0.5\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 39\nvalue = [0, 0, 39]')]
```



In [20]:

```
#Program 2
```

In [42]:

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

In [43]:

```
data=pd.read_csv('Social_Network_Ads.csv')
data.head()
```

Out[43]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [46]:

```
data['Gender'].replace({'Male':0,'Female':1},inplace=True)
data.head()
```

Out[46]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	0	19	19000	0
1	15810944	0	35	20000	0
2	15668575	1	26	43000	0
3	15603246	1	27	57000	0
4	15804002	0	19	76000	0

In [47]:

```
X=data.iloc[:, 1:4].values
y=data.iloc[:,-1].values
```

In [48]:

```
X.shape
```

Out[48]:

```
(400, 3)
```

In [49]:

```
y.shape
```

Out[49]:

```
(400,)
```

In [50]:

```
#clf1=DecisionTreeClassifier(max_depth=3)
clf1=DecisionTreeClassifier()
```

In [52]:

```
clf1.fit(X,y)
```

Out[52]:

```
DecisionTreeClassifier()
```

In [53]:

```
rcParams['figure.figsize'] = 80,50
plot_tree(clf1)
```

Out[53]:

```
[Text(0.44502314814814814, 0.9666666666666667, 'X[1] <= 42.5\ngini = 0.459\nsamples = 400\nvalue = [257, 143]'),
 Text(0.24189814814814814, 0.9, 'X[2] <= 90500.0\ngini = 0.271\nsamples = 285\nvalue = [239, 46]'),
 Text(0.11342592592592593, 0.8333333333333334, 'X[1] <= 36.5\ngini = 0.072\nsamples = 241\nvalue = [232, 9]'),
 Text(0.09490740740740741, 0.7666666666666667, 'gini = 0.0\nsamples = 162\nvalue = [162, 0]'),
 Text(0.13194444444444445, 0.7666666666666667, 'X[2] <= 83500.0\ngini = 0.202\nsamples = 79\nvalue = [70, 9]'),
 Text(0.11342592592592593, 0.7, 'X[2] <= 67500.0\ngini = 0.165\nsamples = 77\nvalue = [70, 7]'),
 Text(0.09490740740740741, 0.6333333333333333, 'gini = 0.0\nsamples = 40\nvalue = [40, 0]'),
 Text(0.13194444444444445, 0.6333333333333333, 'X[2] <= 70500.0\ngini = 0.307\nsamples = 37\nvalue = [30, 7]'),
 Text(0.08333333333333333, 0.5666666666666667, 'X[0] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.06481481481481481, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.10185185185185185, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.18055555555555555, 0.5666666666666667, 'X[1] <= 41.5\ngini = 0.284\nsamples = 35\nvalue = [29, 6]'),
 Text(0.1388888888888889, 0.5, 'X[1] <= 40.5\ngini = 0.231\nsamples = 30\nvalue = [26, 4]'),
 Text(0.12037037037037036, 0.43333333333333335, 'X[2] <= 77500.0\ngini = 0.287\nsamples = 23\nvalue = [19, 4]'),
 Text(0.05555555555555555, 0.36666666666666664, 'X[1] <= 38.5\ngini = 0.219\nsamples = 16\nvalue = [14, 2]'),
 Text(0.037037037037037035, 0.3, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.07407407407407407, 0.3, 'X[2] <= 71500.0\ngini = 0.346\nsamples = 9\nvalue = [7, 2]'),
 Text(0.037037037037037035, 0.23333333333333334, 'X[1] <= 39.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
```

In [54]:

```
clf1=DecisionTreeClassifier(max_depth=3)
```

In [55]:

```
clf1.fit(X,y)
```

Out[55]:

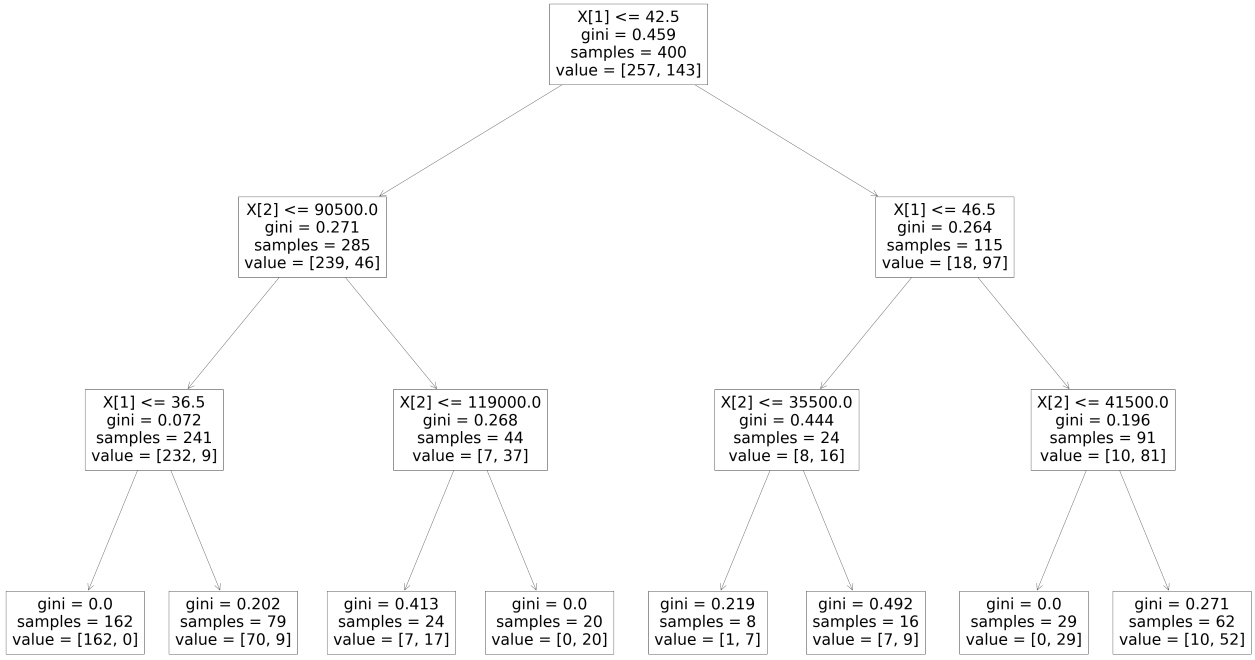
```
DecisionTreeClassifier(max_depth=3)
```

In [56]:

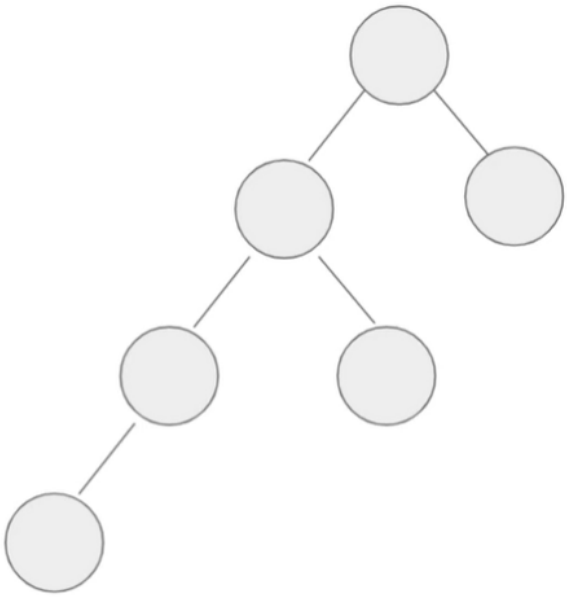
```
rcParams['figure.figsize'] = 80,50

plot_tree(clf1)
```

Out[56]:

```
[Text(0.5, 0.875, 'X[1] <= 42.5\ngini = 0.459\nsamples = 400\nvalue = [257, 143]'),
 Text(0.25, 0.625, 'X[2] <= 90500.0\ngini = 0.271\nsamples = 285\nvalue = [239, 46]'),
 Text(0.125, 0.375, 'X[1] <= 36.5\ngini = 0.072\nsamples = 241\nvalue = [232, 9]'),
 Text(0.0625, 0.125, 'gini = 0.0\nsamples = 162\nvalue = [162, 0]'),
 Text(0.1875, 0.125, 'gini = 0.202\nsamples = 79\nvalue = [70, 9]'),
 Text(0.375, 0.375, 'X[2] <= 119000.0\ngini = 0.268\nsamples = 44\nvalue = [7, 37]'),
 Text(0.3125, 0.125, 'gini = 0.413\nsamples = 24\nvalue = [7, 17]'),
 Text(0.4375, 0.125, 'gini = 0.0\nsamples = 20\nvalue = [0, 20]'),
 Text(0.75, 0.625, 'X[1] <= 46.5\ngini = 0.264\nsamples = 115\nvalue = [18, 97]'),
 Text(0.625, 0.375, 'X[2] <= 35500.0\ngini = 0.444\nsamples = 24\nvalue = [8, 16]'),
 Text(0.5625, 0.125, 'gini = 0.219\nsamples = 8\nvalue = [1, 7]'),
 Text(0.6875, 0.125, 'gini = 0.492\nsamples = 16\nvalue = [7, 9]'),
 Text(0.875, 0.375, 'X[2] <= 41500.0\ngini = 0.196\nsamples = 91\nvalue = [10, 81]'),
 Text(0.8125, 0.125, 'gini = 0.0\nsamples = 29\nvalue = [0, 29]'),
 Text(0.9375, 0.125, 'gini = 0.271\nsamples = 62\nvalue = [10, 52]')]
```

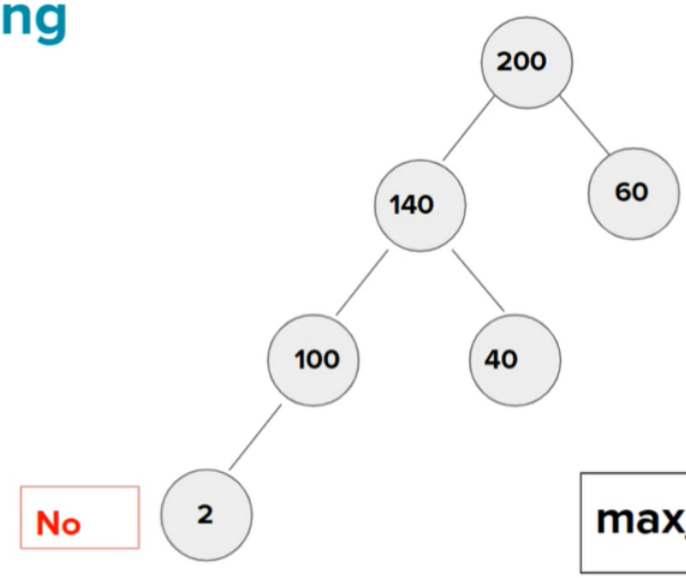


Overfitting in Decision Tree



Depth of Tree

Overfitting



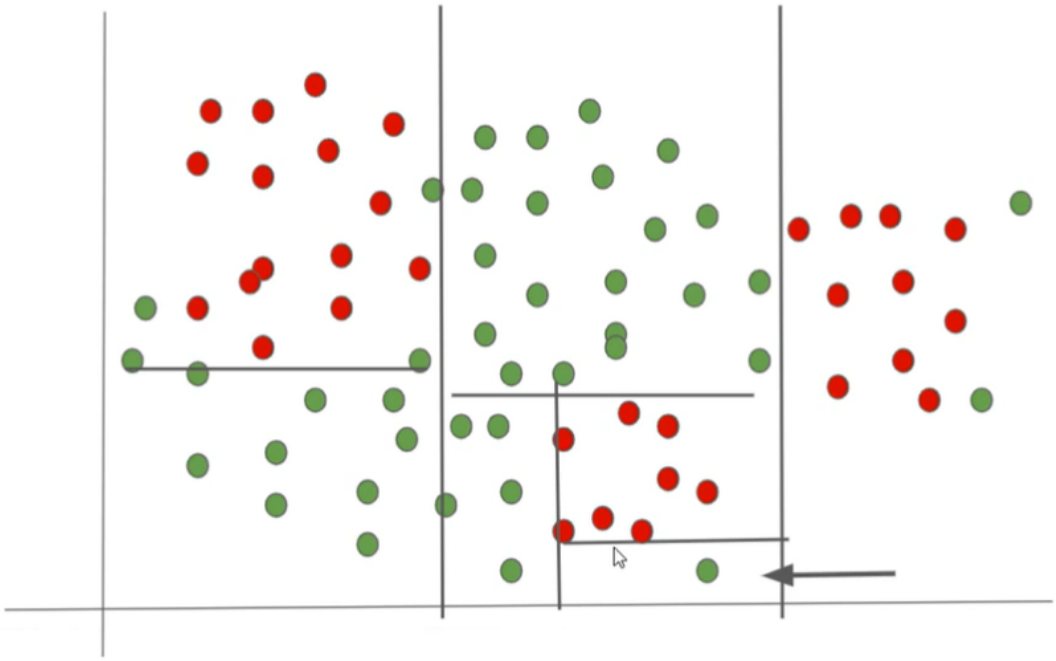
Performs well - Training
Performs badly - Test

No

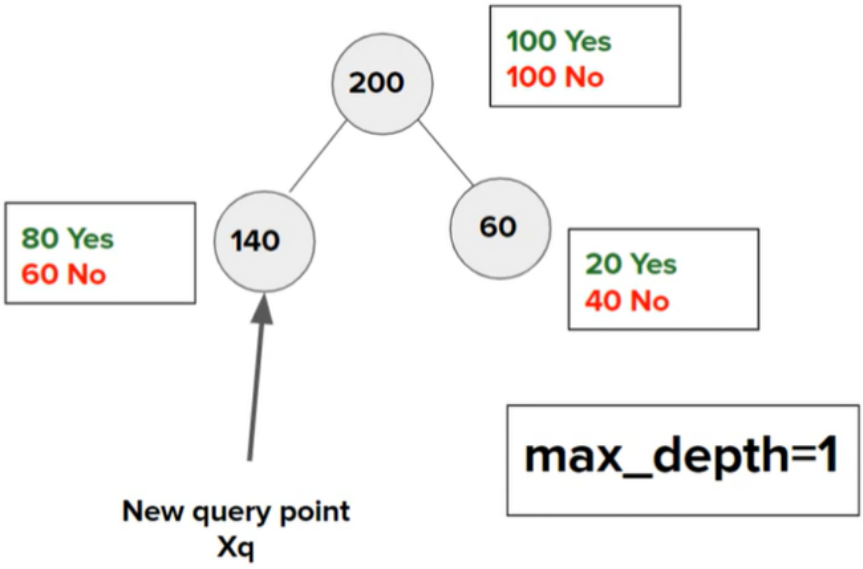
max_depth=None

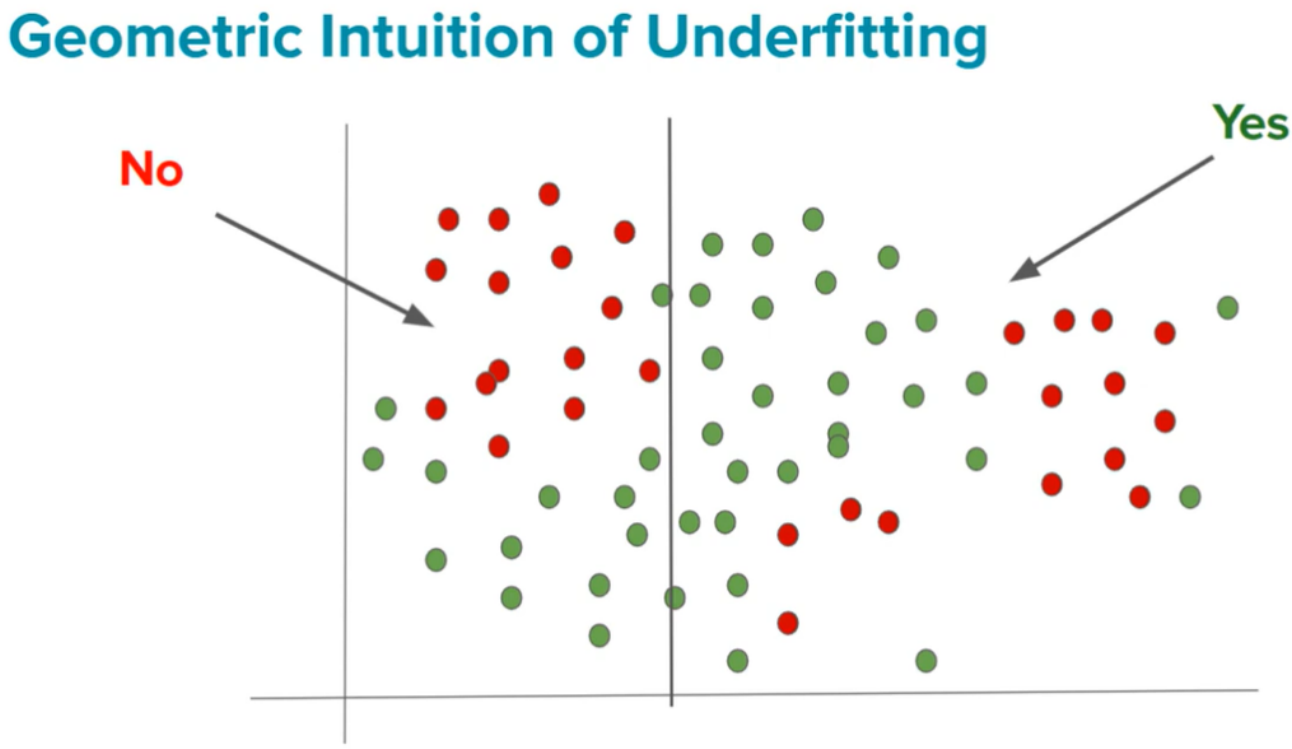
Noisy/Outlier/Erroneous

Geometric Intuition of Overfitting



Underfitting





Decision Tree Hyperparameters In-depth Intuition

https://dt-visualise.herokuapp.com/ (https://dt-visualise.herokuapp.com/)

In [ ]:

```
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=decision+tree
```

Hyper-parameter Tuning using GridSearchCV

In [1]:

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

In [2]:

```
data=pd.read_csv('Social_Network_Ads.csv')
data.head()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:

```
data['Gender'].replace({'Male':0,'Female':1},inplace=True)
data.head()
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	0	19	19000	0
1	15810944	0	35	20000	0
2	15668575	1	26	43000	0
3	15603246	1	27	57000	0
4	15804002	0	19	76000	0

In [4]:

```
X=data.iloc[:, 1:4].values
y=data.iloc[:,-1].values
```

In [5]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [6]:

```
X=scaler.fit_transform(X)
X
```

Out[6]:

```
array([[-1.02020406, -1.78179743, -1.49004624],
       [-1.02020406, -0.25358736, -1.46068138],
       [ 0.98019606, -1.11320552, -0.78528968],
       ...,
       [ 0.98019606,  1.17910958, -1.46068138],
       [-1.02020406, -0.15807423, -1.07893824],
       [ 0.98019606,  1.08359645, -0.99084367]])
```

In [13]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
```

In [14]:

```
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
```

In [15]:

```
clf.fit(X_train,y_train)
```

Out[15]:

```
DecisionTreeClassifier()
```

In [16]:

```
y_pred=clf.predict(X_test)
```

In [17]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[17]:

```
0.775
```

In [20]:

```
param_dist={
    "criterion":["gini","entropy"],
    "max_depth":[1,2,3,4,5,6,7,None]
}
```

In [21]:

```
from sklearn.model_selection import GridSearchCV
grid=GridSearchCV(clf,param_grid=param_dist,cv=10,n_jobs=-1)
```

In [22]:

```
grid.fit(X_train,y_train)
```

Out[22]:

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, None]})
```

In [37]:

```
grid.best_estimator_
```

Out[37]:

```
DecisionTreeClassifier(max_depth=2)
```

In [38]:

```
grid.best_score_
```

Out[38]:

```
0.91875
```

In [39]:

```
grid.best_params_
```

Out[39]:

```
{'criterion': 'gini', 'max_depth': 2}
```

Decision Tree Regressor

Python Code-

In [44]:

```
import pandas as pd
#from pandas_datareader import data
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.datasets import load_boston
from sklearn.model_selection import GridSearchCV
```

In [47]:

```
boston = load_boston()
df = pd.DataFrame(boston.data)
df
```

Out[47]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [48]:

```
df.columns = boston.feature_names
df['MEDV'] = boston.target
```

In [49]:

```
df.head()
```

Out[49]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [50]:

```
X = df.iloc[:,0:13]
y = df.iloc[:,13]
```

In [51]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

In [37]:

```
rt = DecisionTreeRegressor(criterion = 'mse', max_depth=5)
```

In [52]:

```
rt.fit(X_train,y_train)
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated in
v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
```

Out[52]:

```
DecisionTreeRegressor(criterion='mse', max_depth=5)
```

In [53]:

```
y_pred = rt.predict(X_test)
```

In [54]:

```
r2_score(y_test,y_pred)
```

Out[54]:

```
0.8833565347917997
```

Hyperparameter Tuning

In [56]:

```
param_grid = {
    'max_depth':[2,4,8,10,None],
    'criterion':['mse','mae'],
    'max_features':[0.25,0.5,1.0],
    'min_samples_split':[0.25,0.5,1.0]
}
```

In [42]:

```
reg = GridSearchCV(DecisionTreeRegressor(),param_grid=param_grid)
```

In [57]:

```
reg.fit(X_train,y_train)
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning: Criterion 'mse' was deprecated
```

In [58]:

```
reg.best_score_
```

Out[58]:

```
0.6591758393991596
```

In [59]:

```
reg.best_params_
```

Out[59]:

```
{'criterion': 'mse',
 'max_depth': 8,
 'max_features': 0.5,
 'min_samples_split': 0.25}
```

Feature Importance

In [23]:

```
for importance, name in sorted(zip(rt.feature_importances_, X_train.columns),reverse=True):
  print (name, importance)
```

```
RM 0.6344993240692439
LSTAT 0.20562720153418443
DIS 0.06744514557703217
CRIM 0.03550388785641197
NOX 0.02531597355560238
PTRATIO 0.01669708628286102
INDUS 0.007018566233811912
AGE 0.006176126174365166
CHAS 0.00117395935157392
B 0.0005427293649131195
ZN 0.0
TAX 0.0
RAD 0.0
```

In [ ]:

In [ ]:

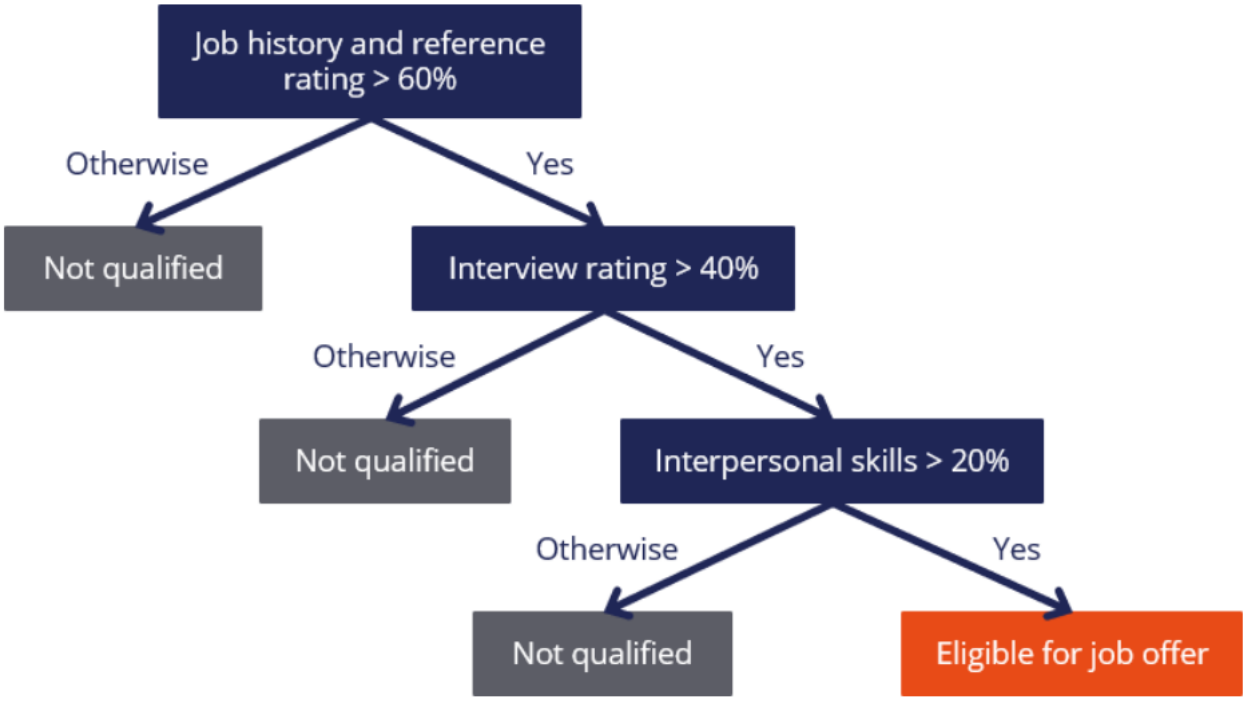
In [ ]:

2. Decision Trees for Interview call

For example,

- a human resources application contains a process for assessing a job candidate. The candidate receives a set of ratings during the interviews. These ratings are evaluated to determine whether to extend a job offer to the candidate.

- A decision tree is configured to automatically use the ratings as test conditions to decide whether the candidate is qualified.
- The decision starts at the top of the tree and proceeds downward. Each yes advances the evaluation.
- The result is either Not qualified or Eligible for job offer.



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes.

So, to solve such problems there is a technique which is called as Attribute selection measure or ASM.

By this measurement, we can easily select the best attribute for the nodes of the tree.

There are two popular techniques for ASM, which are:

- Information Gain

- Gini Index

1. Information Gain:

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.

It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree.

A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

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

1. **Entropy:**

Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. 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

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

Gini Index= 1- ∑jPj2

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python.

For this, we will use the dataset "user_data.csv," which we have used in previous classification models.

By using the same dataset, we can compare the Decision tree classifier with other classification models such as

- KNN
- SVM,
- LogisticRegression, etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

In [15]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [16]:

```
#importing datasets
data_set= pd.read_csv('user_data.csv')
data_set
```

Out[16]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [17]:

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

In [18]:

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

In [19]:

```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set.

For this, we will import the DecisionTreeClassifier class from sklearn.tree library.

Below is the code for it:

In [20]:

```
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

Out[20]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In the above code, we have created a classifier object, in which we have passed two main parameters;

"criterion='entropy':

Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.

random_state=0":

For generating the random states.

3. Predicting the test result

Now we will predict the test set result.

We will create a new prediction vector y_pred. Below is the code for it:

In [21]:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix.

Below is the code for it:

In [22]:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[22]:

```
array([[62,  6],
       [ 3, 29]], dtype=int64)
```

In the above output image, we can see the confusion matrix, which has 6+3= 9 incorrect predictions and62+29=91 correct predictions.

Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

Decision Tree Regression in Python

We will now go through a step-wise Python implementation of the Decision Tree Regression algorithm that we just discussed.

1. Importing necessary libraries

First, let us import some essential Python libraries.

In [12]:

```
# Importing the libraries
import numpy as np # for array operations
import pandas as pd # for working with DataFrames

# scikit-learn modules
from sklearn.model_selection import train_test_split # for splitting the data
from sklearn.metrics import mean_squared_error,r2_score # for calculating the cost function
from sklearn.tree import DecisionTreeRegressor # for building the model
```

2. Importing the data set

The dataset consists of data related to petrol consumptions (in millions of gallons) for 48 US states.

This value is based upon several features such as the petrol tax (in cents), Average income (dollars), paved highways (in miles), and the proportion of the population with a driver's license.

We will be loading the data set using the read_csv() function from the pandas module and store it as a pandas DataFrame object.

In [25]:

```
# Reading the data
dataset = pd.read_csv('petrol_consumption.csv')
dataset.head()
```

Out[25]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

3. Separating the features and the target variable

After loading the dataset, the independent variable (x) and the dependent variable (y) need to be separated.

Our concern is to model the relationships between the features (Petrol_tax, Average_income, etc.) and the target variable (Petrol_consumption) in the dataset.

In [26]:

```
x = dataset.drop('Petrol_Consumption', axis = 1) # Features
y = dataset['Petrol_Consumption']  # Target
```

4. Splitting the data into a train set and a test set

We use the train_test_split() module of scikit-learn for splitting the data into a train set and a test set.

We will be using 20% of the available data as the testing set and the remaining data as the training set.

In [27]:

```
# Splitting the dataset into training and testing set (80/20)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 28)
```

5. Fitting the model to the training dataset

After splitting the data, let us initialize a Decision Tree Regressor model and fit it to the training data.

This is done with the help of DecisionTreeRegressor() module of scikit-learn.

In [28]:

```
# Initializing the Decision Tree Regression model
model = DecisionTreeRegressor(random_state = 0)
# Fitting the Decision Tree Regression model to the data
model.fit(x_train, y_train)
```

Out[28]:

```
DecisionTreeRegressor(random_state=0)
```

6. Calculating the loss after training

Let us now calculate the loss between the actual target values in the testing set and the values predicted by the model with the use of a cost function called the Root Mean Square Error (RMSE).

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where,
y_i is the actual target value,
$\hat{y}_i$ is the predicted target value, and
n is the total number of data points.

The RMSE of a model determines the absolute fit of the model to the data.

In other words, it indicates how close the actual data points are to the model's predicted values.

A low value of RMSE indicates a better fit and is a good measure for determining the accuracy of the model's predictions.

In [13]:

```
# Predicting the target values of the test set
y_pred = model.predict(x_test)
# RMSE (Root Mean Square Error)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("\nRMSE: ", rmse)
```

```
RMSE:  103.70669457657976
```

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.

It can be used for both Classification and Regression problems in ML.

It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

Random Forest Classifier

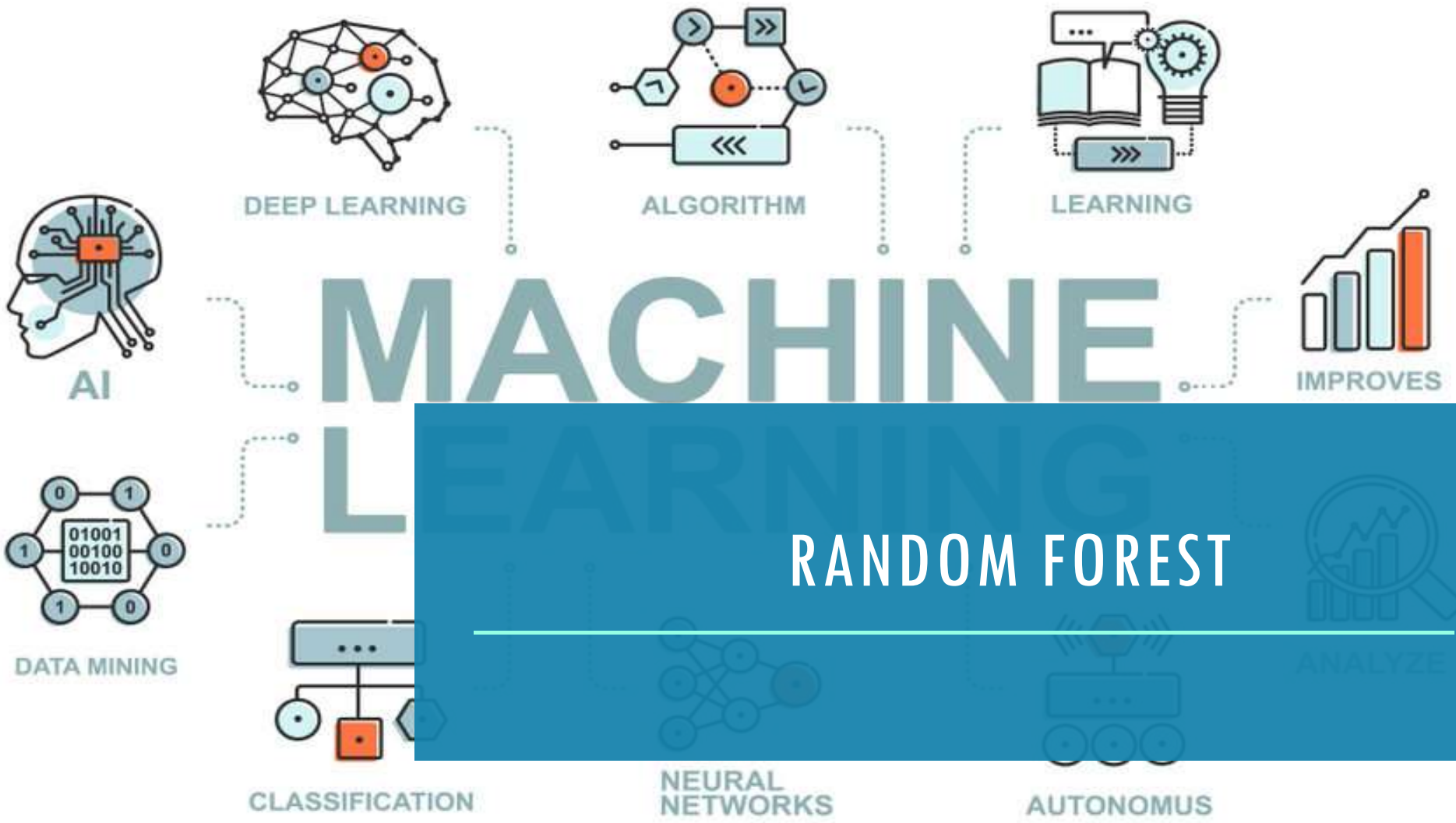
As the name suggests,

"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."

Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



RANDOM FOREST

Random Forest

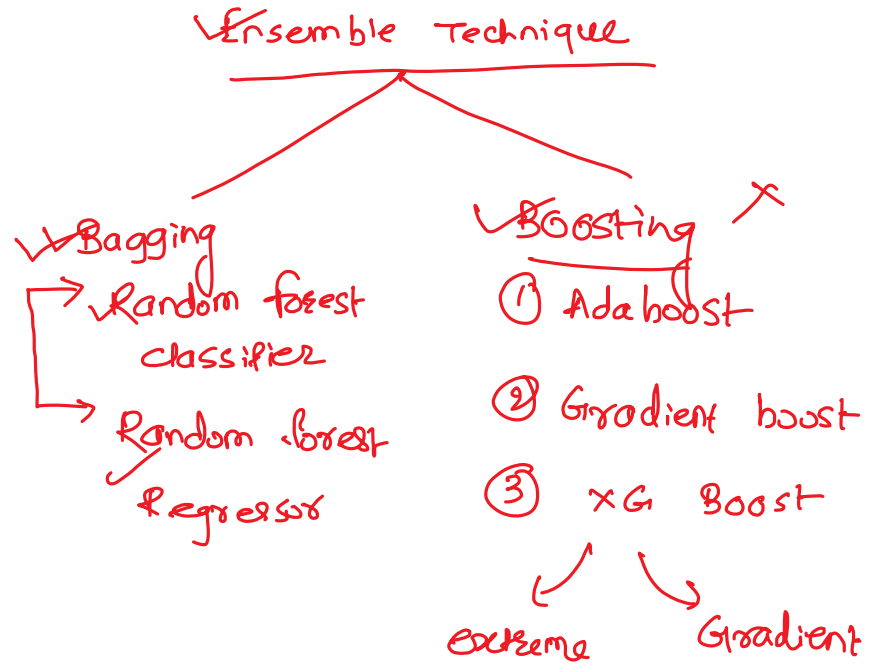
28 November 2022 11:33 AM

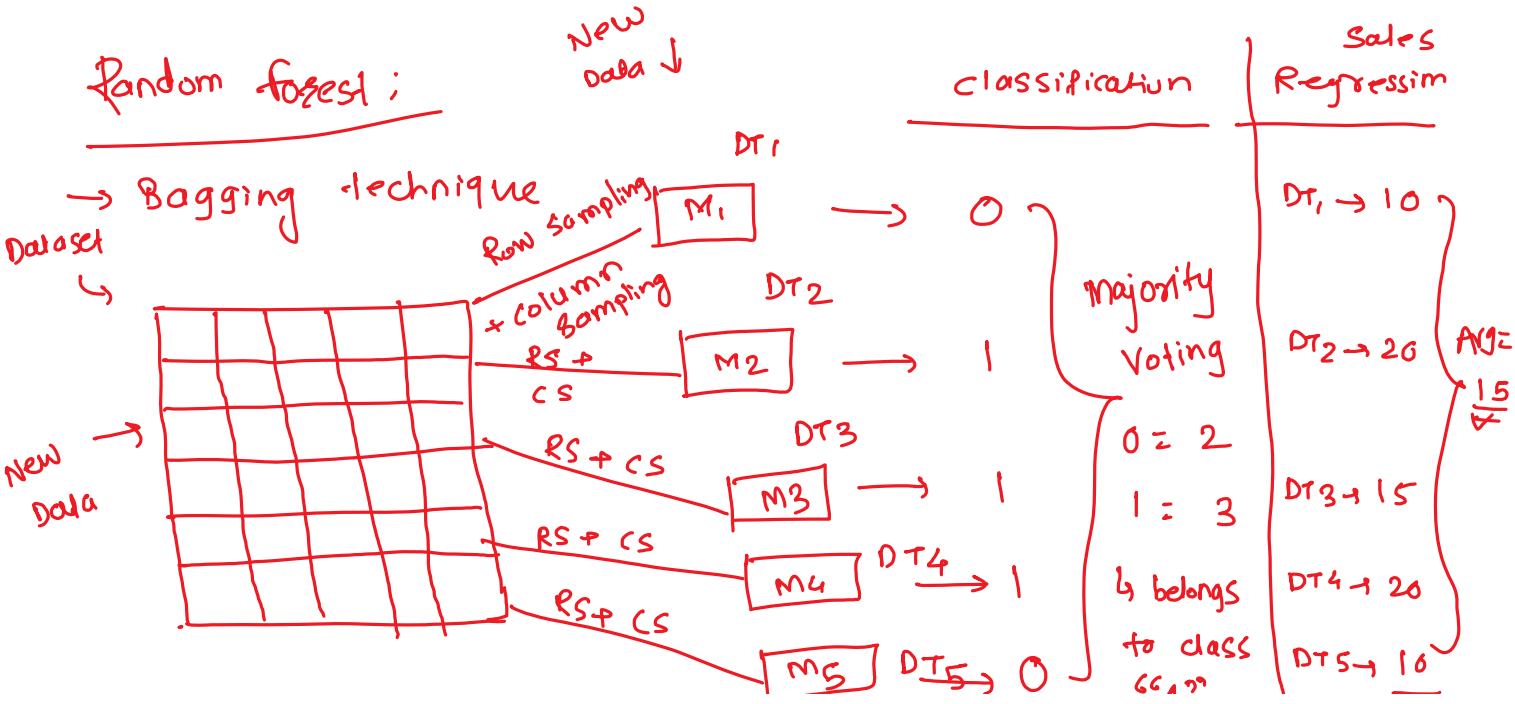
→ It is an extension of Decision Tree.

→ It over come issue / drawback of decision tree
↳ i.e. overfitting.

→ It comes under "ensemble Technique"

→ It is used to solve both problems i.e
① Regression
② classification





→ Also known as "Bootstrap Aggregator".

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where,
y_i is the actual target value,
$\hat{y}_i$ is the predicted target value, and
n is the total number of data points.

The RMSE of a model determines the absolute fit of the model to the data.

In other words, it indicates how close the actual data points are to the model's predicted values.

A low value of RMSE indicates a better fit and is a good measure for determining the accuracy of the model's predictions.

In [13]:

```
# Predicting the target values of the test set
y_pred = model.predict(x_test)
# RMSE (Root Mean Square Error)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("\nRMSE: ", rmse)
```

```
RMSE:  103.70669457657976
```

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.

It can be used for both Classification and Regression problems in ML.

It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

Random Forest Classifier

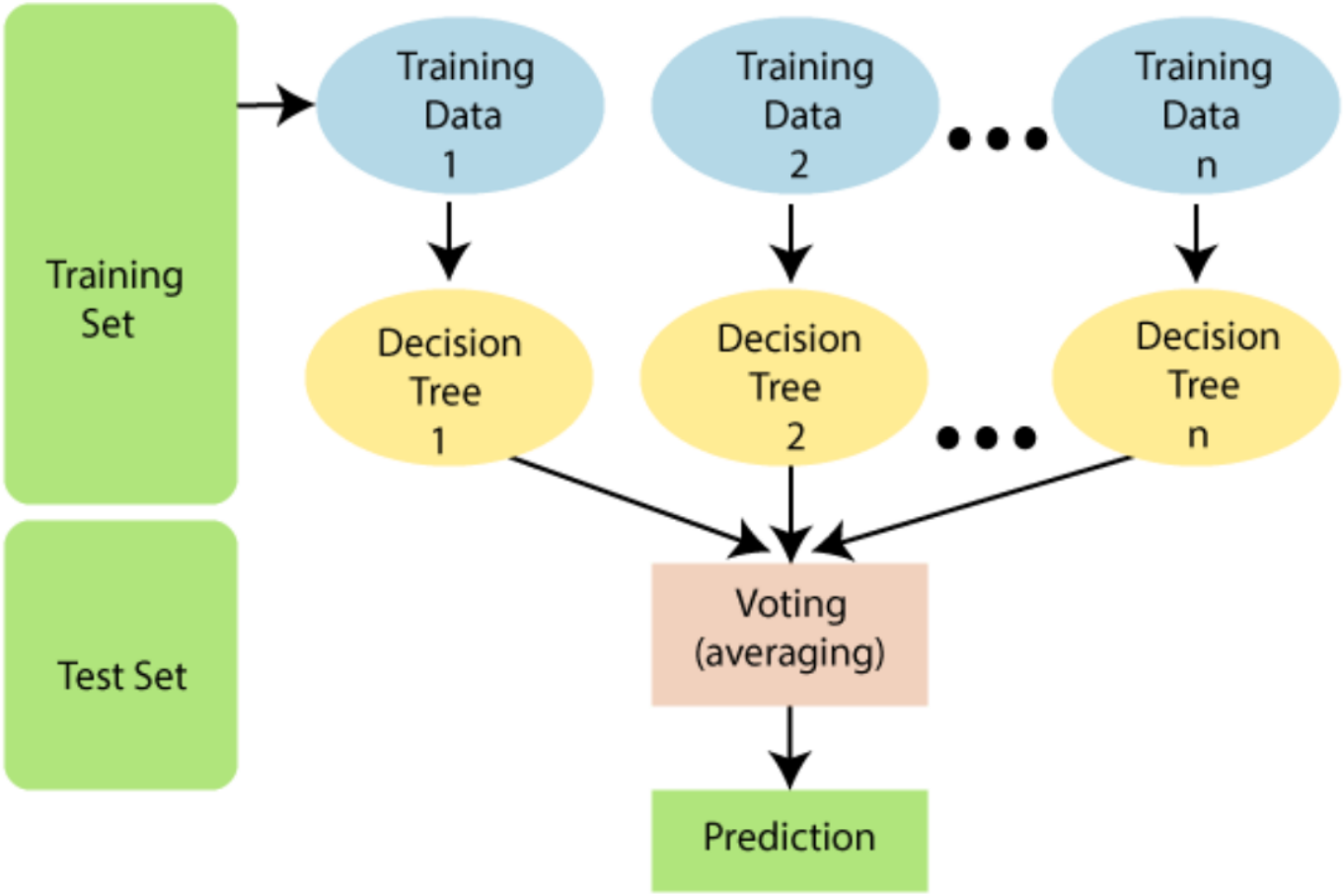
As the name suggests,

"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."

Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1:

Select random K data points from the training set.

Step-2:

Build the decision trees associated with the selected data points (Subsets).

Step-3:

Choose the number N for decision trees that you want to build.

Step-4:

Repeat Step 1 & 2.

Step-5:

For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Example:

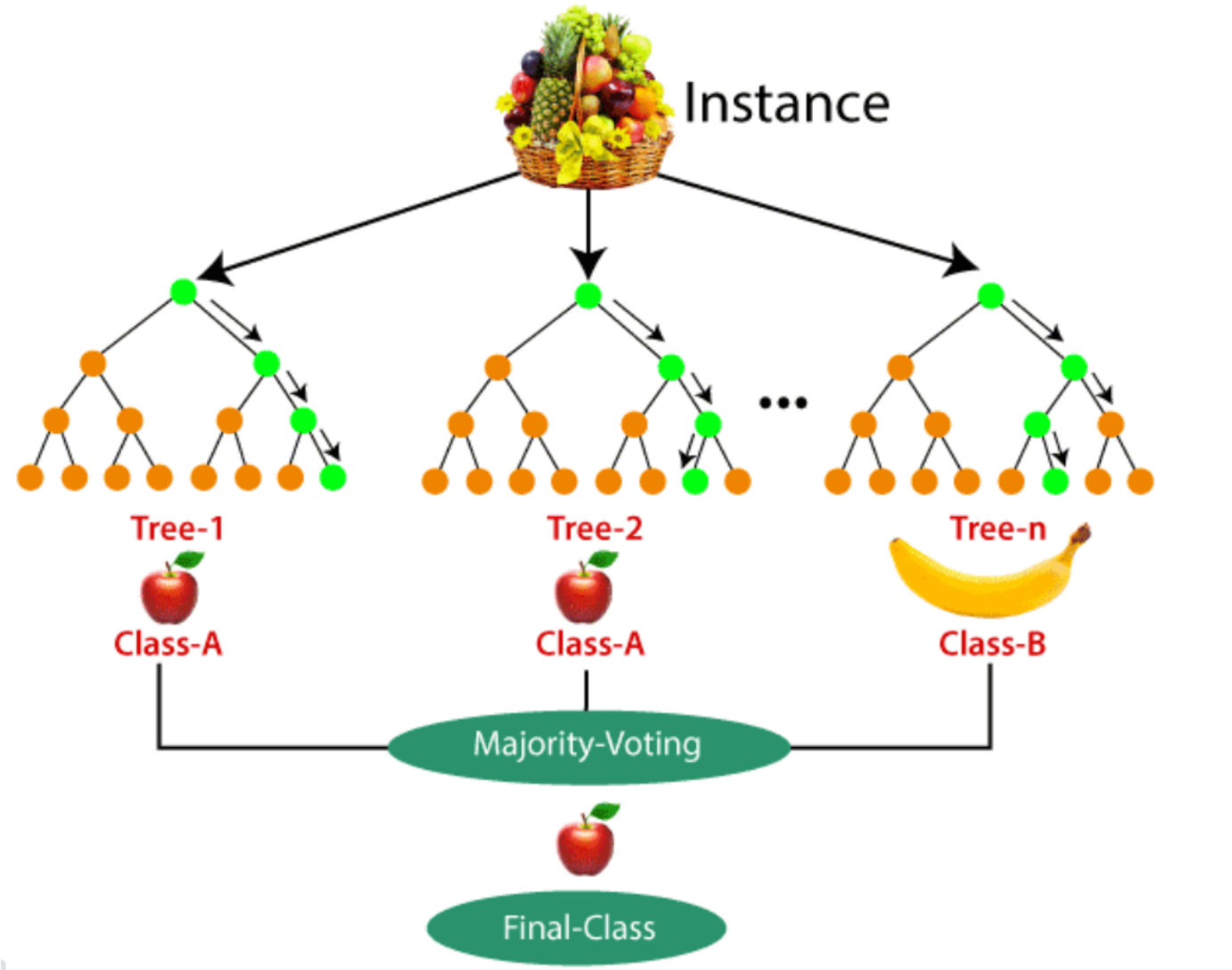
Suppose there is a dataset that contains multiple fruit images.

So, this dataset is given to the Random forest classifier.

The dataset is divided into subsets and given to each decision tree.

During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision.

Consider the below image:



Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user_data.csv", which we have used in previous classification models.

By using the same dataset, we can compare the Random Forest classifier with other classification models such as

- Decision tree Classifier,
- KNN,
- SVM,
- Logistic Regression,

etc.

Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

In [35]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [36]:

```
#importing datasets
data_set= pd.read_csv('user_data.csv')
data_set
```

Out[36]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [37]:

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

In [40]:

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

In [41]:

```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set.

To fit it, we will import the RandomForestClassifier class from the sklearn.ensemble library.

The code is given below:

In [43]:

```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```

Out[43]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In the above code, the classifier object takes below parameters:

n_estimators=

The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.

criterion=

It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result.

For prediction, we will create a new prediction vector y_pred. Below is the code for it:

In [44]:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions.

Below is the code for it:

In [45]:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[45]:

```
array([[63,  5],
       [ 4, 28]], dtype=int64)
```

As we can see in the above matrix, there are 4+4= 8 incorrect predictions and 62+29= 91 correct predictions.

Random Forest Regressor

1. Importing necessary libraries

First, let us import some essential Python libraries.

In [1]:

```
# Importing the libraries
import numpy as np # for array operations
import pandas as pd # for working with DataFrames
```

In [14]:

```
# scikit-learn modules
from sklearn.model_selection import train_test_split # for splitting the data
from sklearn.metrics import mean_squared_error,r2_score # for calculating the cost function
from sklearn.ensemble import RandomForestRegressor # for building the model
```

2. Importing the dataset

In [3]:

```
# Reading the data
dataset = pd.read_csv("petrol_consumption.csv")
dataset.head()
```

Out[3]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

3. Separating the features and the target variable

After loading the dataset, the independent variable and the dependent variable need to be separated.

Our concern is to model the relationships between the features (Petrol_tax, Average_income, etc.) and the target variable (Petrol_consumption) in the dataset.

In [4]:

```
dataset.head()
```

Out[4]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

In [5]:

```
x = dataset.drop('Petrol_Consumption', axis = 1) # Features
y = dataset['Petrol_Consumption']  # Target
```

4. Splitting the data into a train set and a test set

We use the train_test_split() module of scikit-learn for splitting the data into a train set and a test set.

We will be using 20% of the available data as the testing set and the remaining data as the training set.

In [6]:

```
# Splitting the dataset into training and testing set (80/20)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 28)
```

5. Fitting the model to the training dataset

After splitting the data, let us initialize a Random Forest Regression model and fit it to the training data.

This is done with the help of RandomForestRegressor() module of scikit-learn.

In [16]:

```
# Initializing the Random Forest Regression model with 10 decision trees
model = RandomForestRegressor(n_estimators = 10, random_state = 0)

# Fitting the Random Forest Regression model to the data
model.fit(x_train, y_train)
```

Out[16]:

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

6. Calculating the loss after training

Let us now calculate the loss between the actual target values in the testing set and the values predicted by the model with the use of a cost function called the Root Mean Square Error (RMSE).

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

The RMSE of a model determines the absolute fit of the model to the data.

In other words, it indicates how close the actual data points are to the model's predicted values.

A low value of RMSE indicates a better fit and is a good measure for determining the accuracy of the model's predictions.

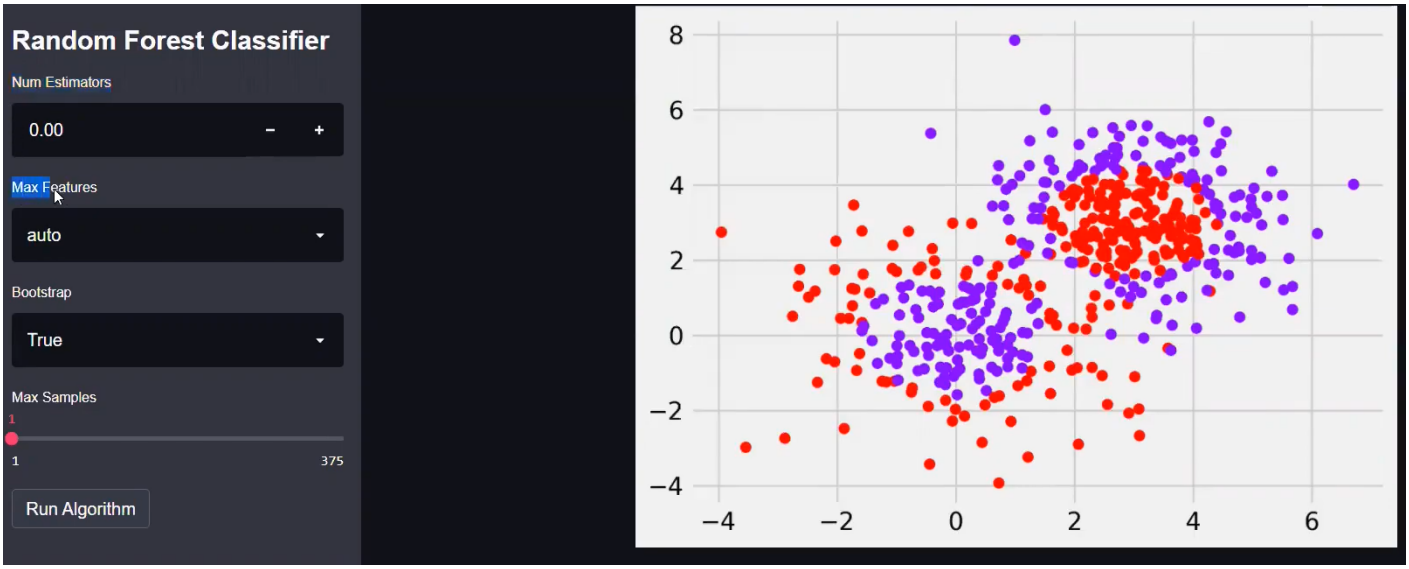
In [17]:

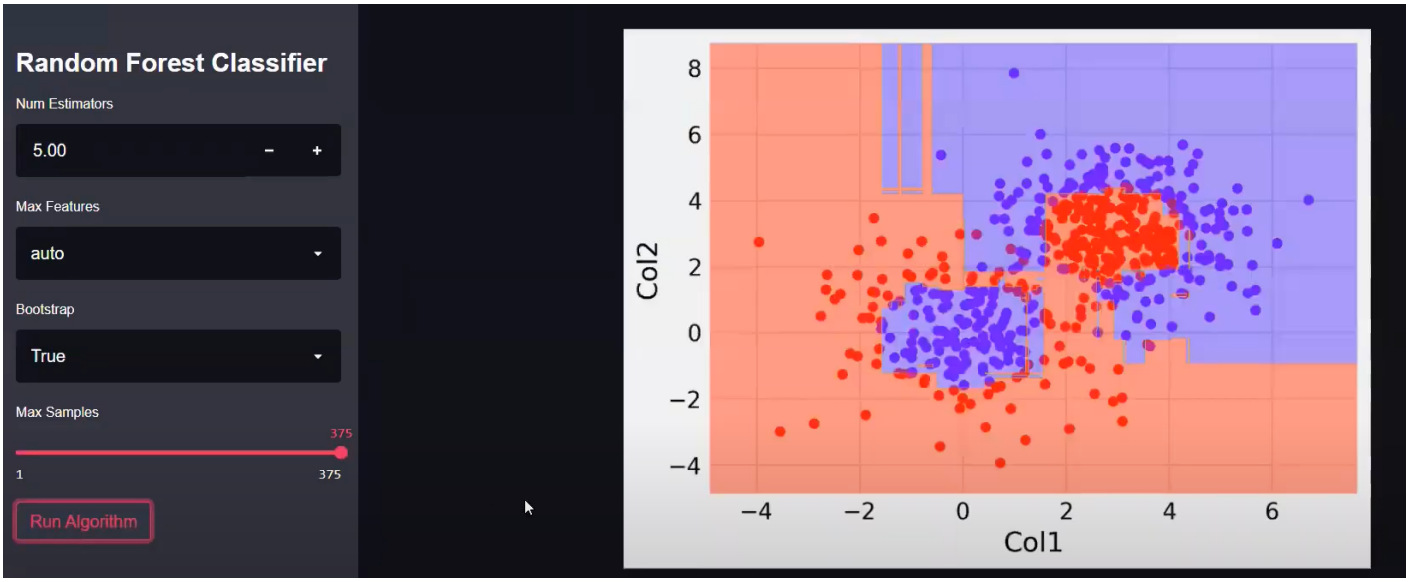
```
# Predicting the target values of the test set
y_pred = model.predict(x_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)

rscore=r2_score(y_test, y_pred)
print(rscore)
```

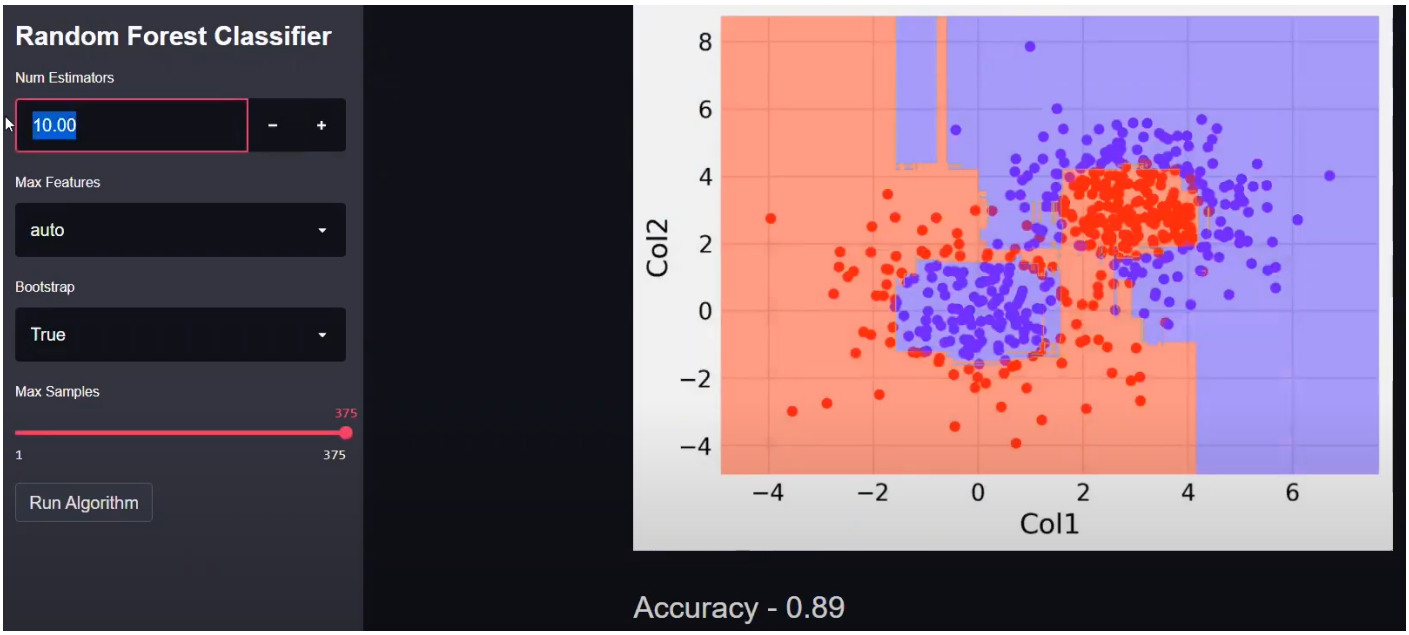
```
RMSE:  96.389
0.48405279841131643
```

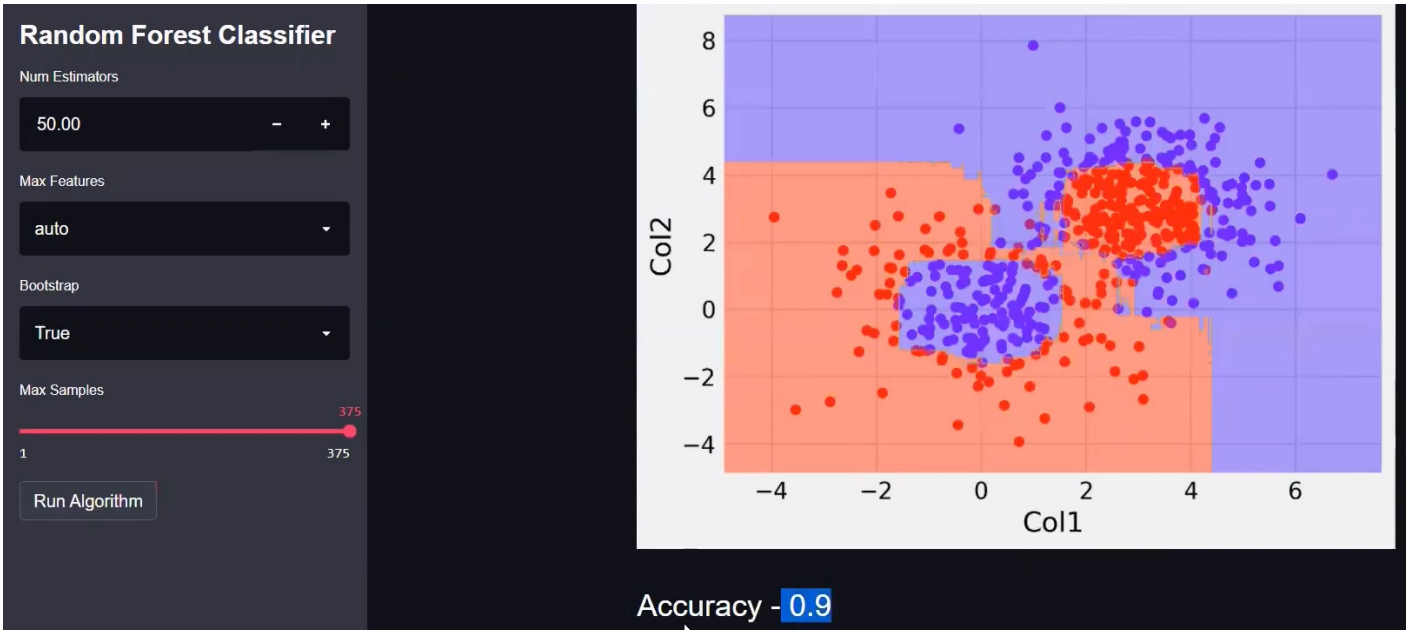
As we can see, the value of error metric has decreased significantly and this model performed quite well than the single decision tree regression model that we studied in the previous lesson.

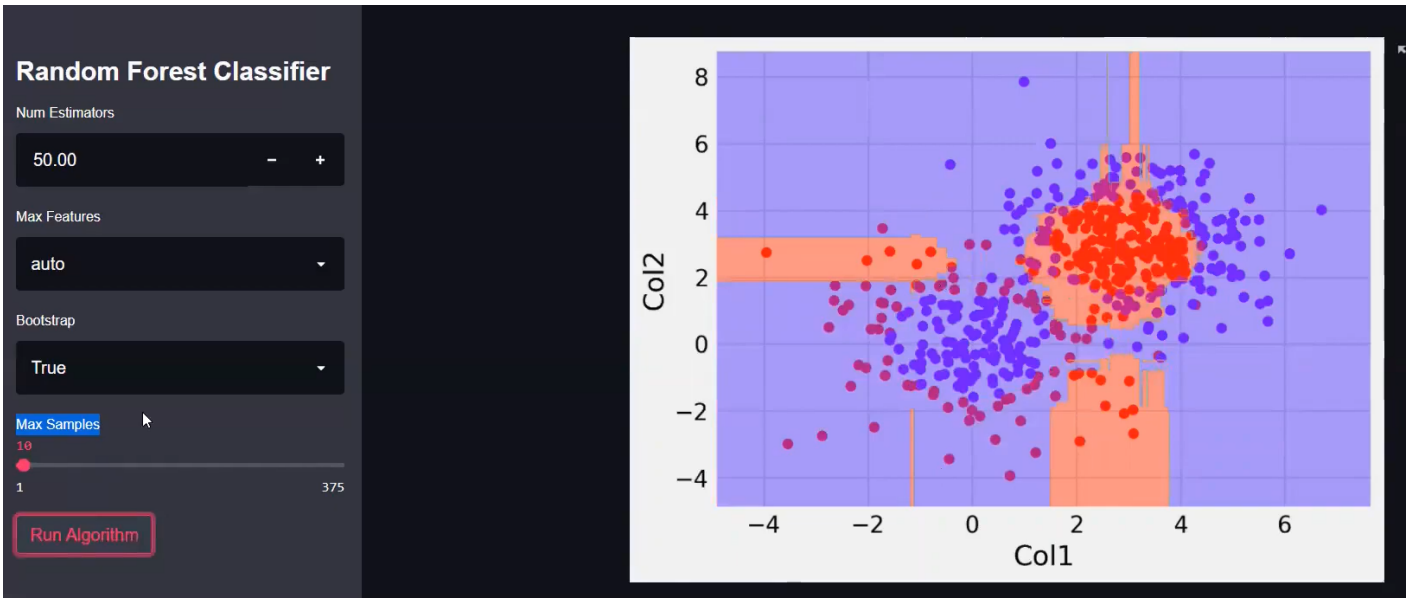
Random Forest Hyperparametrs

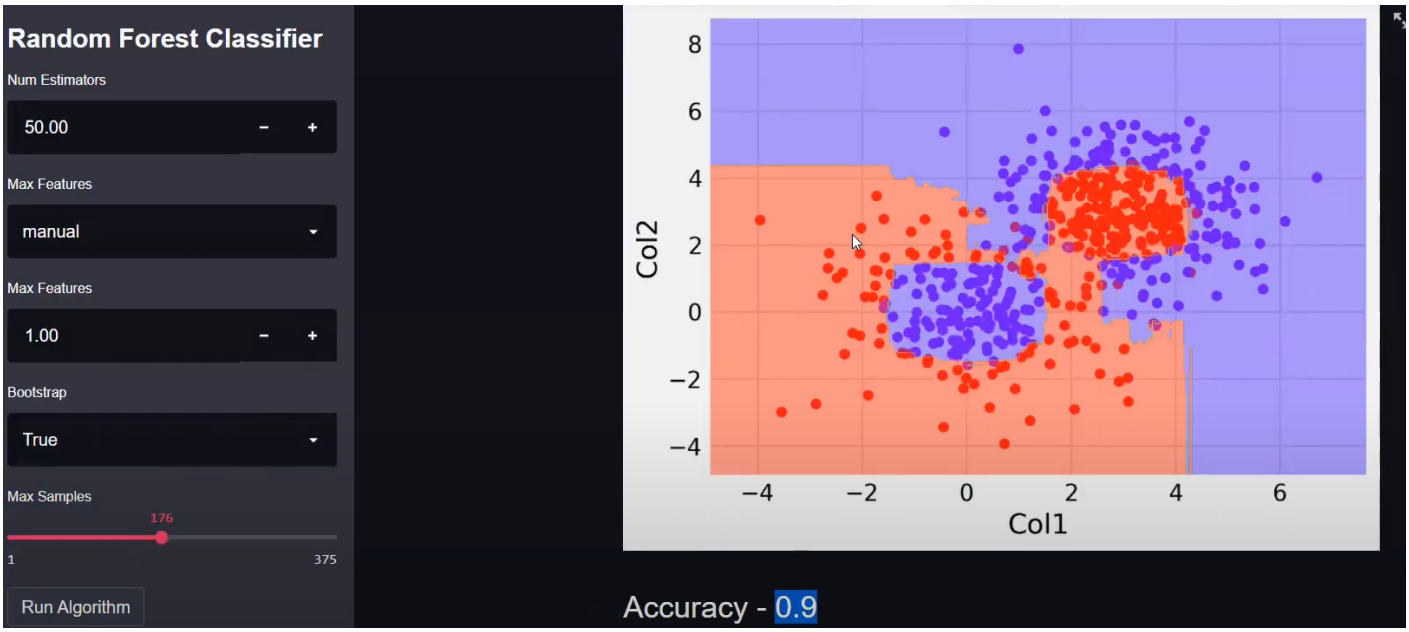





Random Forest Classifier
Num Estimators
10.00
Max Features
auto
Bootstrap
True
Max Samples
375
1
375
Run Algorithm
Col2
Col1
Accuracy - 0.89


Random Forest Classifier
Num Estimators
50.00
Max Features
auto
Bootstrap
True
Max Samples
375
1
375
Run Algorithm
Col2
Col1
Accuracy - 0.9


Random Forest Classifier
Num Estimators
50.00
Max Features
auto
Bootstrap
True
Max Samples
10
1
375
Run Algorithm
Col2
Col1



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Imortant Note About Random State-

In [12]:

```
import pandas as pd
df = pd.read_csv('http://bit.ly/kaggletrain', nrows=6)
df
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q

In [14]:

```
cols = ['Fare', 'Embarked', 'Sex']
X = df[cols]
y = df['Survived']
```

In [15]:

```
X
```

Out[15]:

	Fare	Embarked	Sex
0	7.2500	S	male
1	71.2833	C	female
2	7.9250	S	female
3	53.1000	S	female
4	8.0500	S	male
5	8.4583	Q	male

In [16]:

```
y
```

Out[16]:

```
0    0
1    1
2    1
3    1
4    0
5    0
Name: Survived, dtype: int64
```

In [28]:

```
# any positive integer can be used for the random_state value

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

# With random_state=0, We get the same train and test sets across different executions.

X_train
```

Out[28]:

	Fare	Embarked	Sex
3	53.10	S	female
0	7.25	S	male
4	8.05	S	male

In [27]:

```
X_train
```

Out[27]:

	Fare	Embarked	Sex
3	53.10	S	female
0	7.25	S	male
4	8.05	S	male

Naive Bayes Classifier

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve:

It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.

Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple.

Random Forest-Bias_Variance Tradeoff

30 November 2022 10:07 AM

Sampling

Row Sampling

Column / feature sampling

Random forest: →

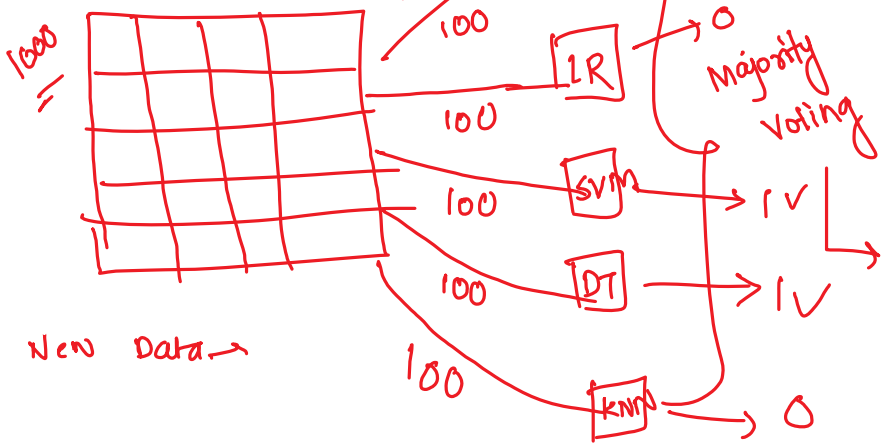
Ensemble Techniques

Bagging

Boosting

Bagging: ✓

Dataset



Random forest

↳ RFC

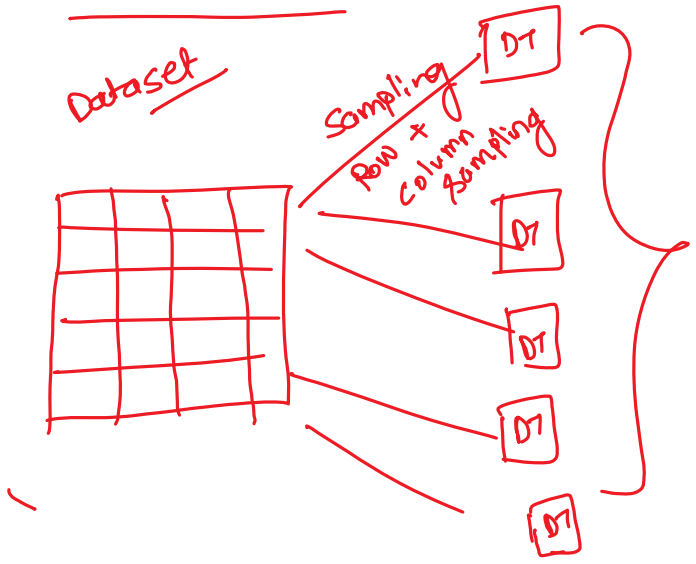
↳ RFR

Majority Voting = 1

≈

Random forest: →

Dataset



Classification

Regression

Majority Voting

Average / mean

* Bias Variance Tradeoff in Random forest: →

↳ Expection after building any ML model → ✓Low Bias

✓Low variance

↳ Overfitting → Low Bias

↳ Overfitting: → Low Bias
→ High Variance ↓

↳ Underfitting → High Bias
→ High Variance

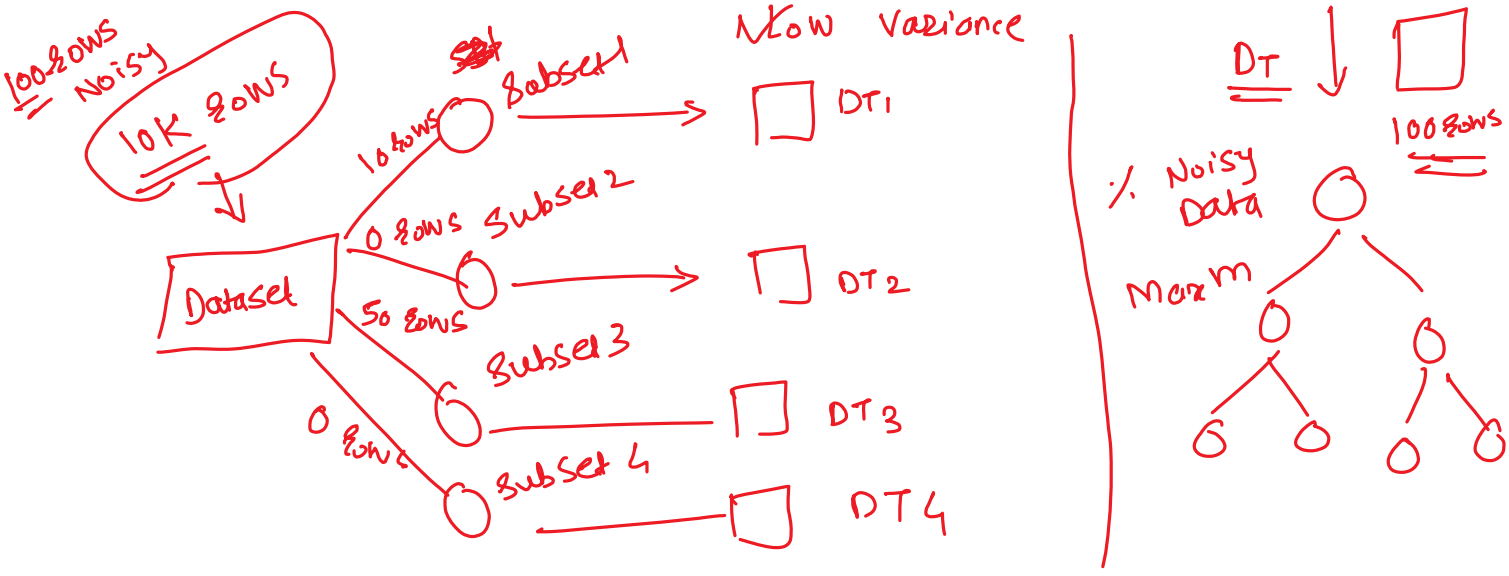
IMP Note: Most of ML Algorithms Struggle with Bias-Variance tradeoff.

Decision Tree:

↳ issue → overfitting → low Bias
→ High Variance ↓

↳ Issue exist because we have fully grown DT.
(Max-Depth = 0)

Low Bias
High Variance → RF → Low Bias
Low Variance



Random_Forest_Classifier

In [1]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [2]:

```
#importing datasets
data_set= pd.read_csv('user_data.csv')
data_set
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [3]:

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

In [4]:

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

In [5]:

```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [6]:

```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```

Out[6]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [7]:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In [8]:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[8]:

```
array([[64,  4],
       [ 4, 28]], dtype=int64)
```

Random_Forest_Regressor

In [9]:

```
# Importing the libraries
import numpy as np # for array operations
import pandas as pd # for working with DataFrames
```

In [10]:

```
# scikit-learn modules
from sklearn.model_selection import train_test_split # for splitting the data
from sklearn.metrics import mean_squared_error # for calculating the cost function
from sklearn.ensemble import RandomForestRegressor # for building the model
```

In [11]:

```
dataset = pd.read_csv("petrol_consumption.csv")
dataset.head()
```

Out[11]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumpt
0	9.0	3571	1976	0.525	
1	9.0	4092	1250	0.572	
2	9.0	3865	1586	0.580	
3	7.5	4870	2351	0.529	
4	8.0	4399	431	0.544	

In [12]:

```
dataset
```

Out[12]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Cons
0	9.00	3571	1976	0.525	
1	9.00	4092	1250	0.572	
2	9.00	3865	1586	0.580	
3	7.50	4870	2351	0.529	
4	8.00	4399	431	0.544	
5	10.00	5342	1333	0.571	
6	8.00	5319	11868	0.451	
7	8.00	5126	2138	0.553	
8	8.00	4447	8577	0.529	
9	7.00	4512	8507	0.552	
10	8.00	4391	5939	0.530	
11	7.50	5126	14186	0.525	
12	7.00	4817	6930	0.574	
13	7.00	4207	6580	0.545	
14	7.00	4332	8159	0.608	
15	7.00	4318	10340	0.586	
16	7.00	4206	8508	0.572	
17	7.00	3718	4725	0.540	
18	7.00	4716	5915	0.724	
19	8.50	4341	6010	0.677	
20	7.00	4593	7834	0.663	
21	8.00	4983	602	0.602	
22	9.00	4897	2449	0.511	
23	9.00	4258	4686	0.517	
24	8.50	4574	2619	0.551	
25	9.00	3721	4746	0.544	
26	8.00	3448	5399	0.548	
27	7.50	3846	9061	0.579	
28	8.00	4188	5975	0.563	
29	9.00	3601	4650	0.493	
30	7.00	3640	6905	0.518	
31	7.00	3333	6594	0.513	
32	8.00	3063	6524	0.578	
33	7.50	3357	4121	0.547	

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Cons
34	8.00	3528	3495	0.487	
35	6.58	3802	7834	0.629	
36	5.00	4045	17782	0.566	
37	7.00	3897	6385	0.586	
38	8.50	3635	3274	0.663	
39	7.00	4345	3905	0.672	
40	7.00	4449	4639	0.626	
41	7.00	3656	3985	0.563	
42	7.00	4300	3635	0.603	
43	7.00	3745	2611	0.508	
44	6.00	5215	2302	0.672	
45	9.00	4476	3942	0.571	
46	7.00	4296	4083	0.623	
47	7.00	5002	9794	0.593	

In [13]:

```
x = dataset.drop('Petrol_Consumption', axis = 1) # Features
y = dataset['Petrol_Consumption']  # Target
```

In [14]:

```
# Splitting the dataset into training and testing set (80/20)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 2
```

In [15]:

```
# Initializing the Random Forest Regression model with 10 decision trees
model = RandomForestRegressor(n_estimators = 10, random_state = 0)

# Fitting the Random Forest Regression model to the data
model.fit(x_train, y_train)
```

Out[15]:

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

In [16]:

```
# Predicting the target values of the test set
y_pred = model.predict(x_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  96.389
```

In [18]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
```

In [19]:

```
np.random.seed(42)
X, y = make_circles(n_samples=500, factor=0.1, noise=0.35, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [20]:

```
X.shape
```

Out[20]:

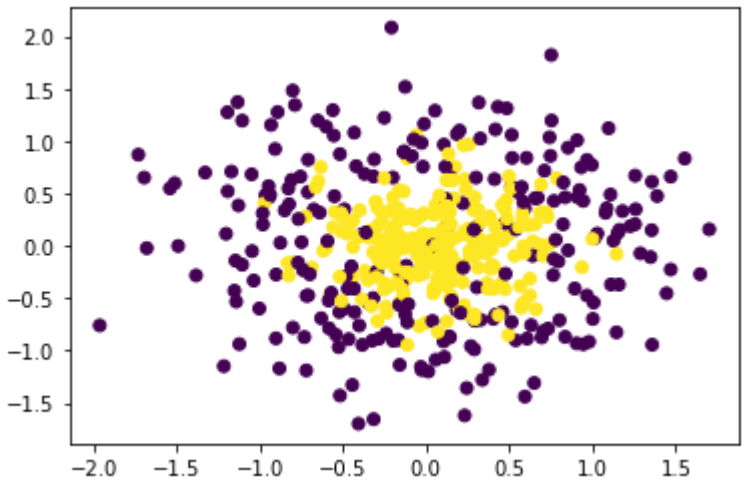
```
(500, 2)
```

In [21]:

```
plt.scatter(X[:,0],X[:,1],c=y)
```

Out[21]:

```
<matplotlib.collections.PathCollection at 0x15e42a32e80>
```

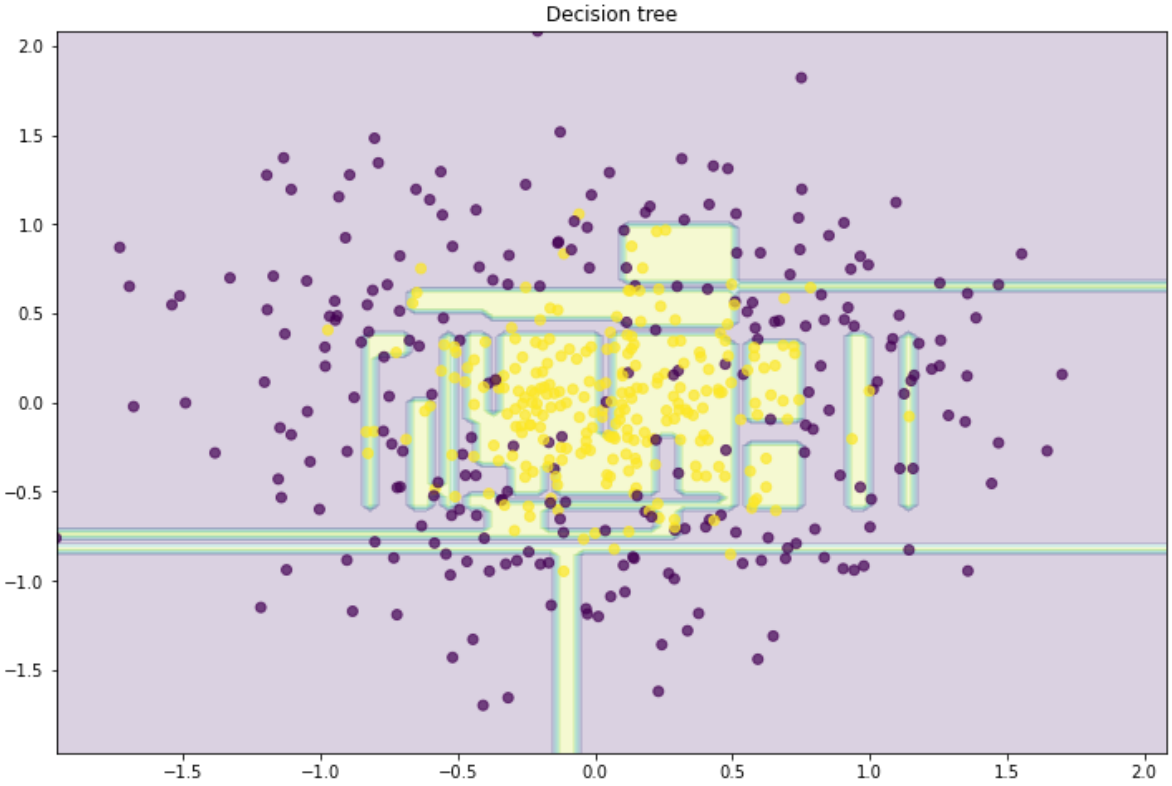


In [22]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [23]:

```
dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train, y_train)

plt.figure(figsize=(12, 8))
x_range = np.linspace(X.min(), X.max(), 100)
xx1, xx2 = np.meshgrid(x_range, x_range)
y_hat = dtree.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat = y_hat.reshape(xx1.shape)
plt.contourf(xx1, xx2, y_hat, alpha=0.2)
plt.scatter(X[:,0], X[:,1], c=y, cmap='viridis', alpha=.7)
plt.title("Decision tree")
plt.show()
```

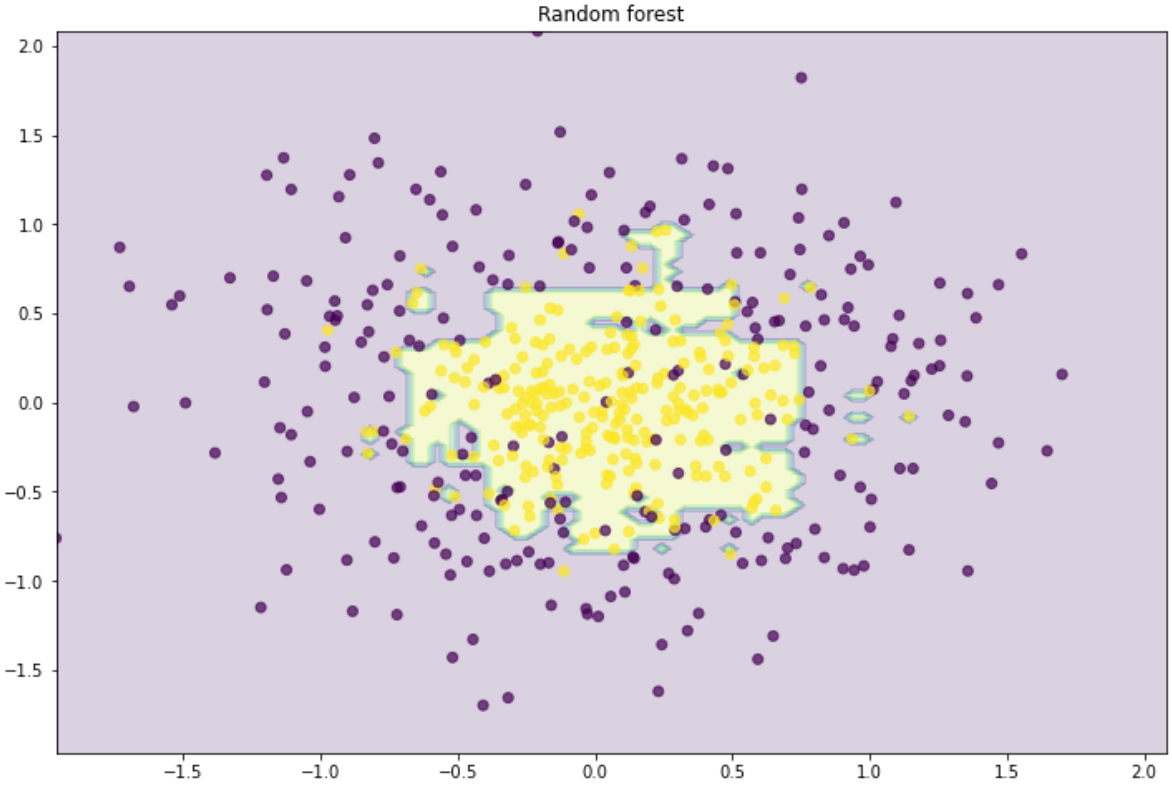


In [24]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [25]:

```
rf = RandomForestClassifier(n_estimators=500, random_state=42)
rf.fit(X_train, y_train)

plt.figure(figsize=(12, 8))
x_range = np.linspace(X.min(), X.max(), 100)
xx1, xx2 = np.meshgrid(x_range, x_range)
y_hat = rf.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat = y_hat.reshape(xx1.shape)
plt.contourf(xx1, xx2, y_hat, alpha=0.2)
plt.scatter(X[:,0], X[:,1], c=y, cmap='viridis', alpha=.7)
plt.title("Random forest")
plt.show()
```



Random_Forest_Hyperparamter

In [26]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

In [27]:

```
df=pd.read_csv("heart.csv")
```

In [28]:

```
df.head()
```

Out[28]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [29]:

```
df.shape
```

Out[29]:

```
(303, 14)
```

In [30]:

```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
```

In [31]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 42)
```

In [32]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(242, 13)
(61, 13)
```

In [33]:

```
rf = RandomForestClassifier()
gb = GradientBoostingClassifier()
svc = SVC()
lr = LogisticRegression()
```

In [34]:

```
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[34]:

```
0.8688524590163934
```

In [35]:

```
gb.fit(X_train,y_train)
y_pred = gb.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[35]:

```
0.7704918032786885
```

In [36]:

```
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[36]:

```
0.7049180327868853
```

In [37]:

```
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
C:\Users\katka\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[37]:

```
0.8852459016393442
```

In [38]:

```
rf = RandomForestClassifier(max_samples=0.75,random_state=42)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[38]:

```
0.9016393442622951
```

In [39]:

```
from sklearn.model_selection import cross_val_score
np.mean(cross_val_score(RandomForestClassifier(max_samples=0.75),X,y,cv=10,scoring='accurac
```

Out[39]:

```
0.8213978494623657
```

Grid_Search_CV

In [40]:

```
# Number of trees in random forest
n_estimators = [20,60,100,120]

# Number of features to consider at every split
max_features = [0.2,0.6,1.0]

# Maximum number of levels in tree
max_depth = [2,8,None]

# Number of samples
max_samples = [0.5,0.75,1.0]

# 108 diff random forest train
```

In [41]:

```
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
              'max_samples':max_samples
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_d
epth': [2, 8, None], 'max_samples': [0.5, 0.75, 1.0]}
```

In [42]:

```
rf = RandomForestClassifier()
```

In [43]:

```
from sklearn.model_selection import GridSearchCV

rf_grid = GridSearchCV(estimator = rf,
                       param_grid = param_grid,
                       cv = 5,
                       verbose=2,
                       n_jobs = -1)
```

In [44]:

```
rf_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

C:\Users\katka\anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:922: UserWarning: One or more of the test scores are non-finite: [0.818367
35 0.81802721 0.81403061 0.81403061 0.82653061 0.80578231
 0.83443878 0.81802721        nan        nan        nan        nan
 0.80178571 0.80161565 0.80986395 0.82227891 0.79319728 0.81811224
 0.80569728 0.81386054        nan        nan        nan        nan
 0.78103741 0.80178571 0.80586735 0.80986395 0.78103741 0.81802721
 0.80569728 0.79311224        nan        nan        nan        nan
 0.79362245 0.82636054 0.81802721 0.81794218 0.80569728 0.81386054
 0.80544218 0.81386054        nan        nan        nan        nan
 0.80977891 0.79362245 0.80161565 0.80161565 0.81411565 0.79736395
 0.80187075 0.80161565        nan        nan        nan        nan
 0.77661565 0.83061224 0.79736395 0.78503401 0.80561224 0.78112245
 0.78920068 0.80153061        nan        nan        nan        nan
 0.82210884 0.81819728 0.83044218 0.81811224 0.77287415 0.85935374
 0.81411565 0.80960884        nan        nan        nan        nan
 0.79778912 0.81403061 0.80586735 0.80153061 0.77687075 0.79353741
 0.79753401 0.79336735        nan        nan        nan        nan
 0.79744898 0.80153061 0.79744898 0.80586735 0.78528912 0.79345238
 0.80161565 0.81394558        nan        nan        nan        nan]
  warnings.warn(
```

Out[44]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [2, 8, None],
                         'max_features': [0.2, 0.6, 1.0],
                         'max_samples': [0.5, 0.75, 1.0],
                         'n_estimators': [20, 60, 100, 120]},
             verbose=2)
```

In [45]:

```
rf_grid.best_params_
```

Out[45]:

```
{'max_depth': None,
 'max_features': 0.2,
 'max_samples': 0.75,
 'n_estimators': 60}
```

In [46]:

```
rf_grid.best_score_
```

Out[46]:

```
0.8593537414965986
```

Random_SearcHCV

In [47]:

```
# Number of trees in random forest
n_estimators = [20,60,100,120]

# Number of features to consider at every split
max_features = [0.2,0.6,1.0]

# Maximum number of levels in tree
max_depth = [2,8,None]

# Number of samples
max_samples = [0.5,0.75,1.0]

# Bootstrap samples
bootstrap = [True,False]

# Minimum number of samples required to split a node
min_samples_split = [2, 5]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]
```

In [48]:

```
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
              'max_samples':max_samples,
              'bootstrap':bootstrap,
              'min_samples_split':min_samples_split,
              'min_samples_leaf':min_samples_leaf
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_d
epth': [2, 8, None], 'max_samples': [0.5, 0.75, 1.0], 'bootstrap': [True, Fa
lse], 'min_samples_split': [2, 5], 'min_samples_leaf': [1, 2]}
```

In [49]:

```
from sklearn.model_selection import RandomizedSearchCV

rf1_grid = RandomizedSearchCV(estimator = rf,
                        param_distributions = param_grid,
                        cv = 5,
                        verbose=2,
                        n_jobs = -1)
```

In [50]:

```
rf1_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

C:\Users\katka\anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:922: UserWarning: One or more of the test scores are non-finite: [       n
an 0.76845238 0.82210884 0.78095238 0.78520408 0.81411565
 0.79744898        nan        nan        nan]
  warnings.warn(
```

Out[50]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [2, 8, None],
                                        'max_features': [0.2, 0.6, 1.0],
                                        'max_samples': [0.5, 0.75, 1.0],
                                        'min_samples_leaf': [1, 2],
                                        'min_samples_split': [2, 5],
                                        'n_estimators': [20, 60, 100, 120]},
                   verbose=2)
```

In [51]:

```
rf1_grid.best_params_
```

Out[51]:

```
{'n_estimators': 100,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_samples': 0.75,
 'max_features': 0.2,
 'max_depth': 8,
 'bootstrap': True}
```

In [52]:

```
rf1_grid.best_score_
```

Out[52]:

```
0.8221088435374149
```

OOB_Score

In [53]:

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
```

In [54]:

```
df = pd.read_csv('heart.csv')
df.head()
```

Out[54]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [55]:

```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
```

In [56]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [57]:

```
rf = RandomForestClassifier(oob_score=True)
```

In [58]:

```
rf.fit(X_train,y_train)
```

Out[58]:

```
RandomForestClassifier(oob_score=True)
```

In [59]:

```
rf.oob_score_
```

Out[59]:

```
0.8140495867768595
```

In [60]:

```
y_pred = rf.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[60]:

```
0.8852459016393442
```

Random_Forest VS Bagging

In [61]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.ensemble import BaggingClassifier,RandomForestClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

In [62]:

```
X,y = make_classification(n_features=5, n_redundant=0, n_informative=5,n_clusters_per_class
```

In [63]:

```
df = pd.DataFrame(X,columns=['col1','col2','col3','col4','col5'])
df['target'] = y
print(df.shape)
df.head()
```

```
(100, 6)
```

Out[63]:

	col1	col2	col3	col4	col5	target
0	-1.765952	-0.791515	-0.099873	2.437252	-0.927235	1
1	-1.215499	-2.406256	1.331668	2.457582	1.125860	0
2	1.370530	0.399268	0.874582	2.641590	-1.832184	1
3	0.213046	-1.013744	-0.371020	2.251846	-0.427333	1
4	1.871673	0.916576	-0.282287	0.819674	-1.079200	1

In [64]:

```
bag = BaggingClassifier(max_features=2)
```

In [65]:

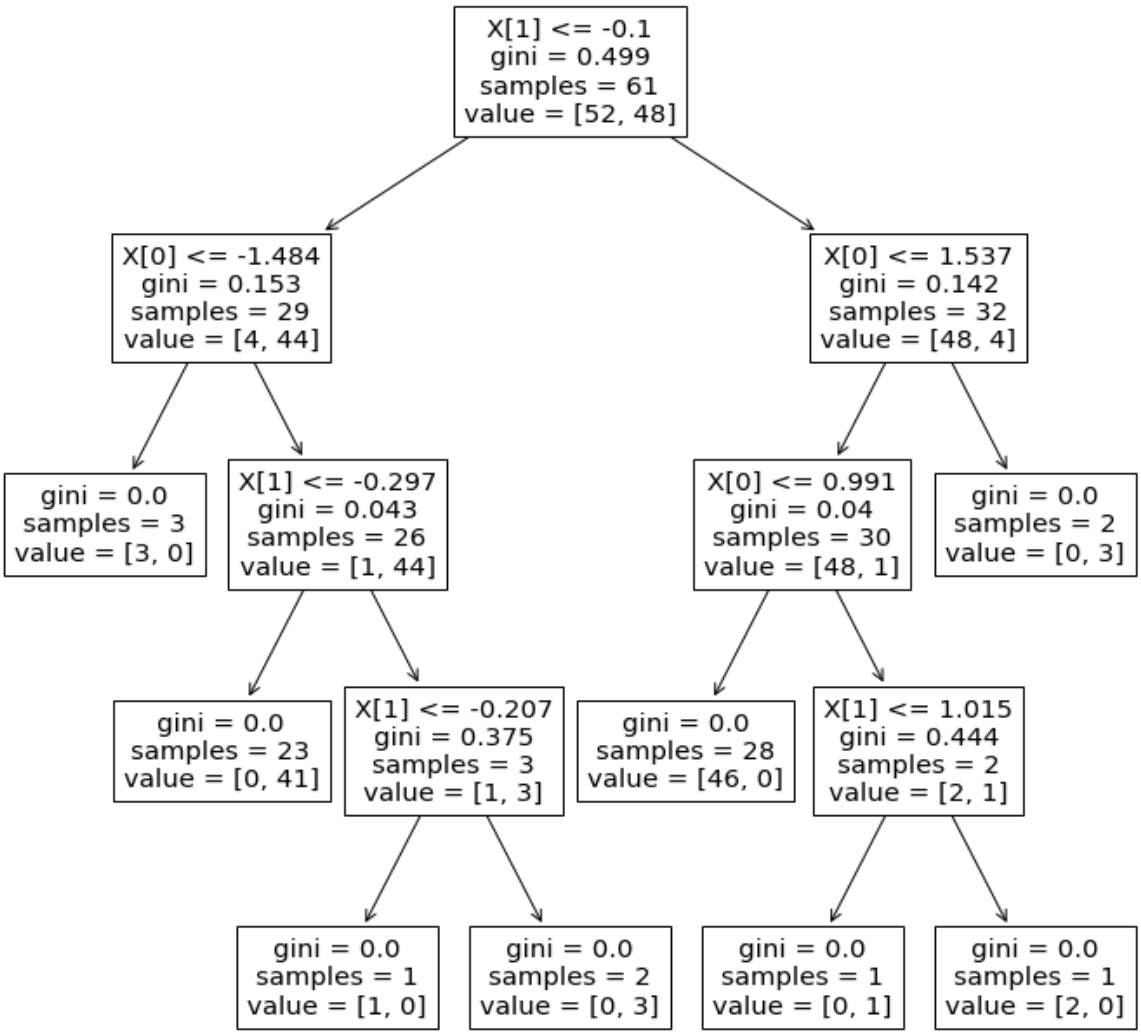
```
bag.fit(df.iloc[:,:5],df.iloc[:,-1])
```

Out[65]:

```
BaggingClassifier(max_features=2)
```

In [66]:

```
plt.figure(figsize=(12,12))
plot_tree(bag.estimators_[0])
plt.show()
```



In [67]:

```
rf = RandomForestClassifier(max_features=2)
```

In [68]:

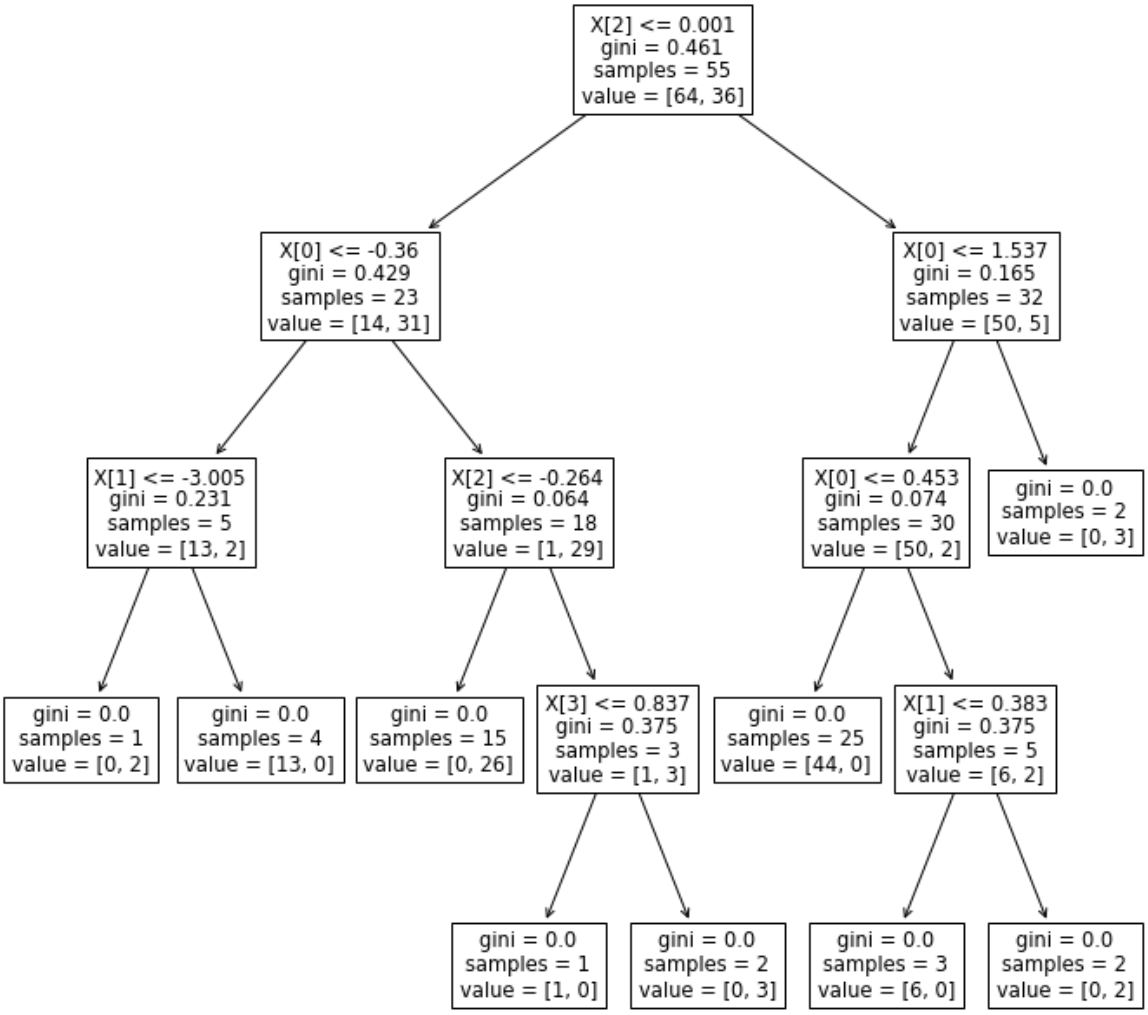
```
rf.fit(df.iloc[:,:5],df.iloc[:,-1])
```

Out[68]:

```
RandomForestClassifier(max_features=2)
```

In [69]:

```
plt.figure(figsize=(12,12))
plot_tree(rf.estimators_[4])
plt.show()
```



In [ ]:

mahendra_ugale@csmssengg.org

DISCLAIMER

The material for the presentation has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this lecture/presentation is for general information and education purpose only. While we endeavor to keep the information up to date and correct, we make no representation of any kind about the completeness and accuracy of the material. The information shared through this presentation material should be used for educational purpose only.