# Chapter -1 Classification Algorithms

## Classification Algorithms

Classification algorithms are used when the **output variable is categorical**, which means there are two classes such as Yes-No, Male-Female, True-false,

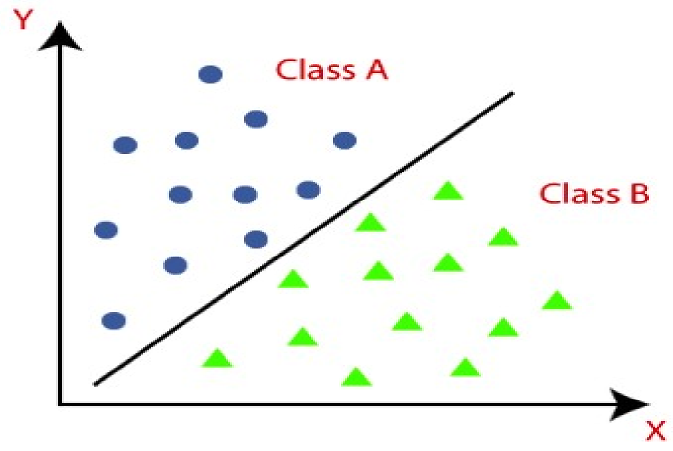
**Popular classification algorithms**

- Logistic Regression
- Decision Trees
- Naïve Bayes
- Support vector Machines
- KNN
- Random Forest

## Classification Algorithms

- In classification algorithm, a discrete output function(y) is mapped to input variable(x).

  y=f(x), where y = categorical output



## Classification Algorithms

- The algorithm which implements the classification on a dataset is known **as a classifier.**

There are two types of Classifications:

- **Binary Classifier:** classification problem has only two possible outcomes, then it is called as Binary Classifier.

Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.

- **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.

Example: Classifications of types of crops, Classification of types of music.

## Types of ML Classification Algorithms

Classification Algorithms can be further divided into the Mainly two category:

Linear Models

- ✓ Logistic Regression
- ✓ Support Vector Machines

Non-linear Models

- ✓ K-Nearest Neighbours
- ✓ Decision Tree
- ✓ Naïve Bayes
- ✓ Random Forest

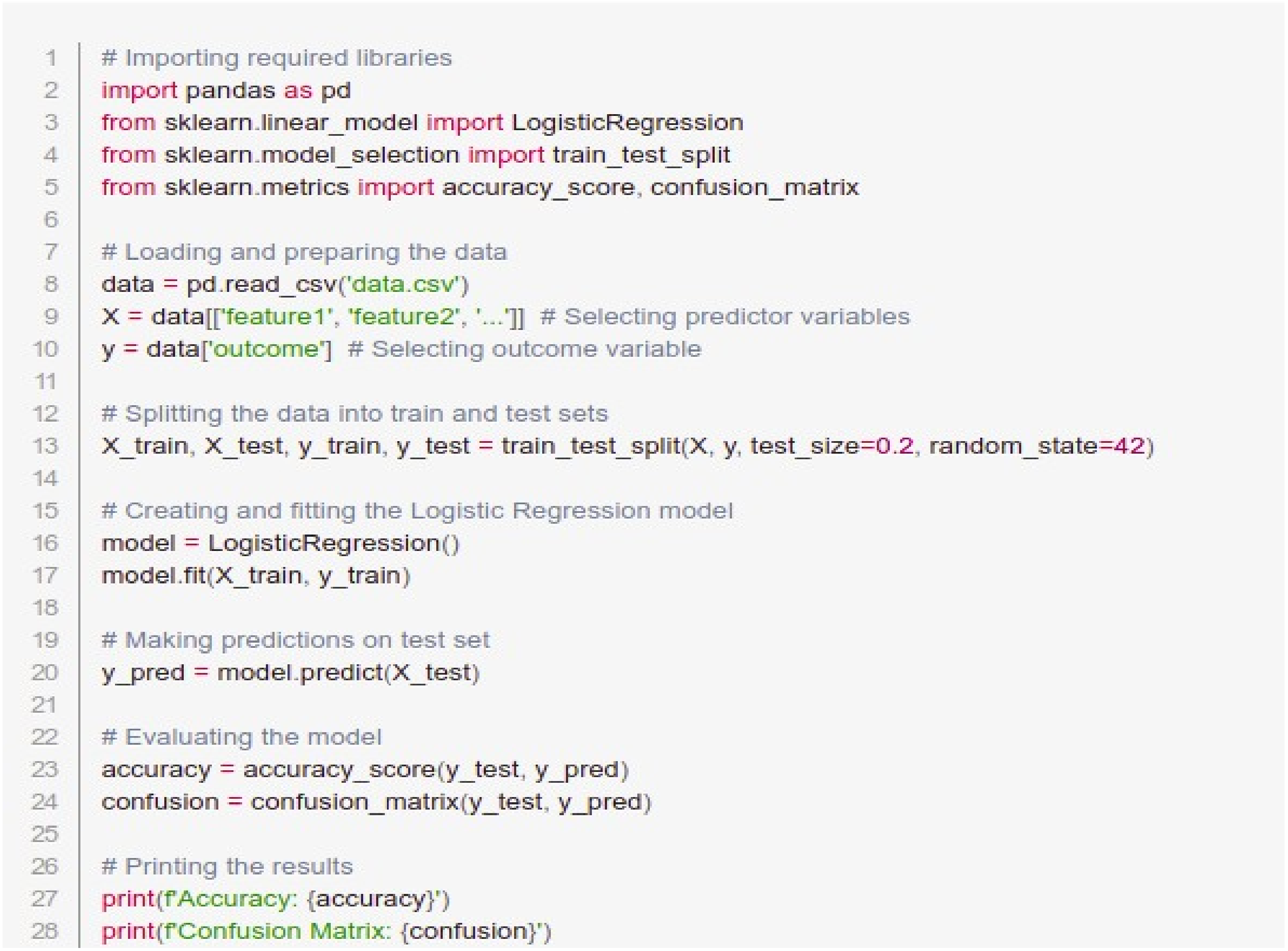
## Types of Logistic Regression

- On the **basis of the categories**, Logistic Regression can be classified into three types:
- **Binomial**: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial**: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal**: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

## ML Implementation of Logistic Regression

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result

## Code samples



```python
# Importing required libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Loading and preparing the data
data = pd.read_csv('data.csv')
X = data[['feature1', 'feature2', '...']]  # Selecting predictor variables
y = data['outcome']  # Selecting outcome variable

# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and fitting the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Making predictions on test set
y_pred = model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

# Printing the results
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix: {confusion}')
```

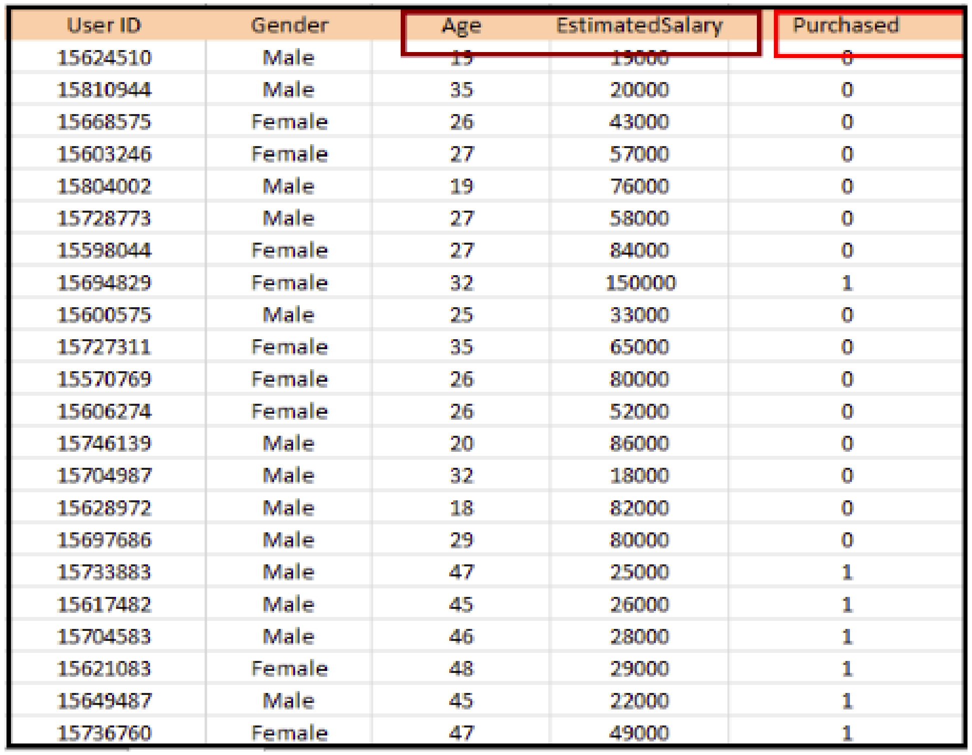
## Example -

There is a car making company that has recently launched a new car

So the company **wanted to check predict whether a user will purchase the product or not, one needs to find out the relationship between Age and Estimated Salary.**

**Source of data**

https://www.kaggle.com/code/sandragracenelson/logistic-regression-on-user-data-csv/input



| User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| 15624510 | Male | 19 | 19000 | 0 |
| 15810944 | Male | 35 | 20000 | 0 |
| 15668575 | Female | 26 | 43000 | 0 |
| 15603246 | Female | 27 | 57000 | 0 |
| 15804002 | Male | 19 | 76000 | 0 |
| 15728773 | Male | 27 | 58000 | 0 |
| 15598044 | Female | 27 | 84000 | 0 |
| 15694829 | Female | 32 | 150000 | 1 |
| 15600575 | Male | 25 | 33000 | 0 |
| 15727311 | Female | 35 | 65000 | 0 |
| 15570769 | Female | 26 | 80000 | 0 |
| 15606274 | Female | 26 | 52000 | 0 |
| 15746139 | Male | 20 | 86000 | 0 |
| 15704987 | Male | 32 | 18000 | 0 |
| 15628972 | Male | 18 | 82000 | 0 |
| 15697686 | Male | 29 | 80000 | 0 |
| 15733883 | Male | 47 | 25000 | 1 |
| 15617482 | Male | 45 | 26000 | 1 |
| 15704583 | Male | 46 | 28000 | 1 |
| 15621083 | Female | 48 | 29000 | 1 |
| 15649487 | Male | 45 | 22000 | 1 |
| 15736760 | Female | 47 | 49000 | 1 |

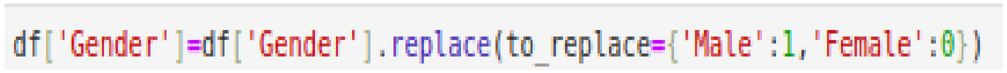
## Data Processing -Related to our data set

**Data Preprocessing Techniques Techniques you should apply:**

1.Learn about your data using pandas **df.shape ,df.describe() ,df.isnull().sum()**

How about if we want to include the **age as independent** variable

Replace male and female with discrete values b/n 0 and 1



```
df['Gender']=df['Gender'].replace(to_replace={'Male':1,'Female':0})
```

Select appropriate Data

X=df.iloc[:,:] or X= df [[, , , ,]]

2. as we see there is a variation b/n **age and salary** value which may create bias

So, need to apply **Feature scaling /normalization using StandardScalaer or MinMaxScalar**

**3.split ,train and test your algorithm**

## K-Nearest Neighbors(KNN)

**K-Nearest Neighbors (KNN)** is a **simple and versatile** machine learning algorithm used for both **classification and regression** tasks.

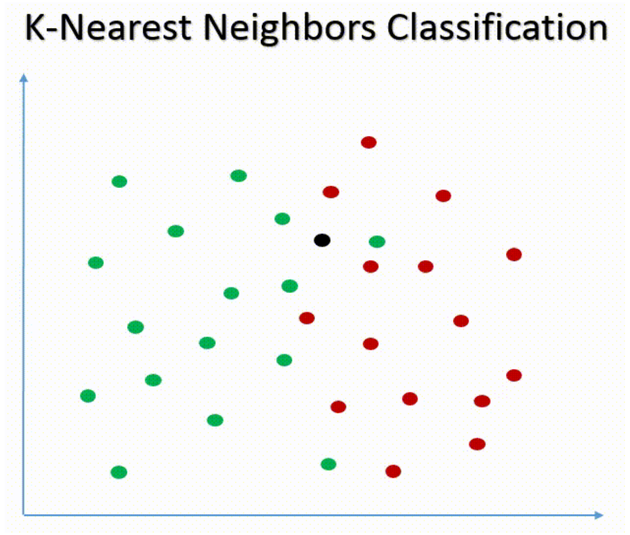
The fundamental idea behind KNN is to predict the label of a data point by **looking at its k nearest neighbors** in the feature space.

### Technique to classify

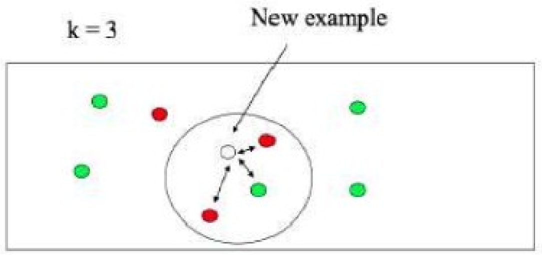
Given a new, unseen data point, find the **k-nearest neighbors** in the training set based on **some distance metric (Euclidean distance).**

**For classification:** Assign the **majority class label among** the k-nearest neighbors to the **new data point.**

# K-Nearest Neighbors(KNN)



When unknown tuple is given – searches the pattern space for the k training tuples (k nearest nieghbors) that are closest to the unknown tuple.



- The most used distance function is the Euclidian distance: $d(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$

## K-Nearest Neighbors(KNN)

**Advantages**

– Conceptually simple, easy to understand and explain

– Very flexible decision boundaries

– Not much learning at all

**Disadvantages**

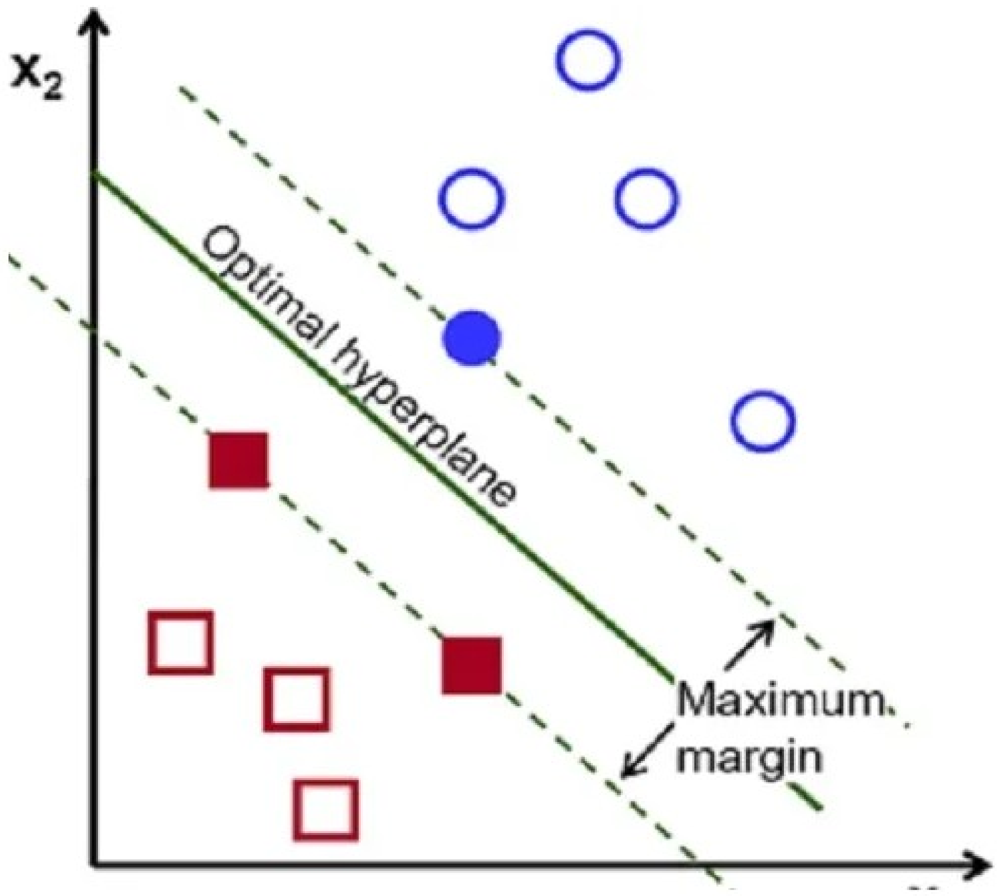
– It can be hard to find a good distance measure

– Irrelevant features and noise can be very detrimental

– Typically can not handle more than a few dozen attributes

– Computational cost: requires a lot computation and memory

## SVM Machine Learning algorithm



## SVM Machine Learning algorithm

**Support Vector Machine (SVM)** is one of the **most useful supervised ML** algorithms.

It can be used for both classification and regression tasks.

**Basic idea of support vector machines:**

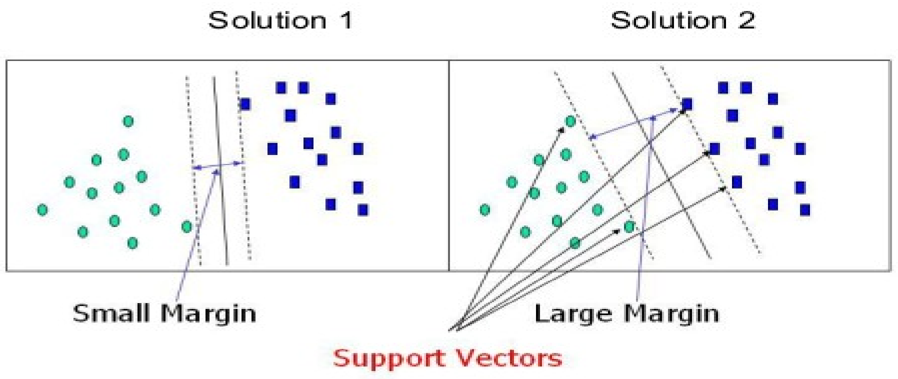
SVM is a geometric model that views the input data as two **sets of vectors** in an n-dimensional space.

- It constructs **a separating hyperplane in that space**, one which **maximizes the margin between the two data sets.**

# SVM Machine Learning algorithm

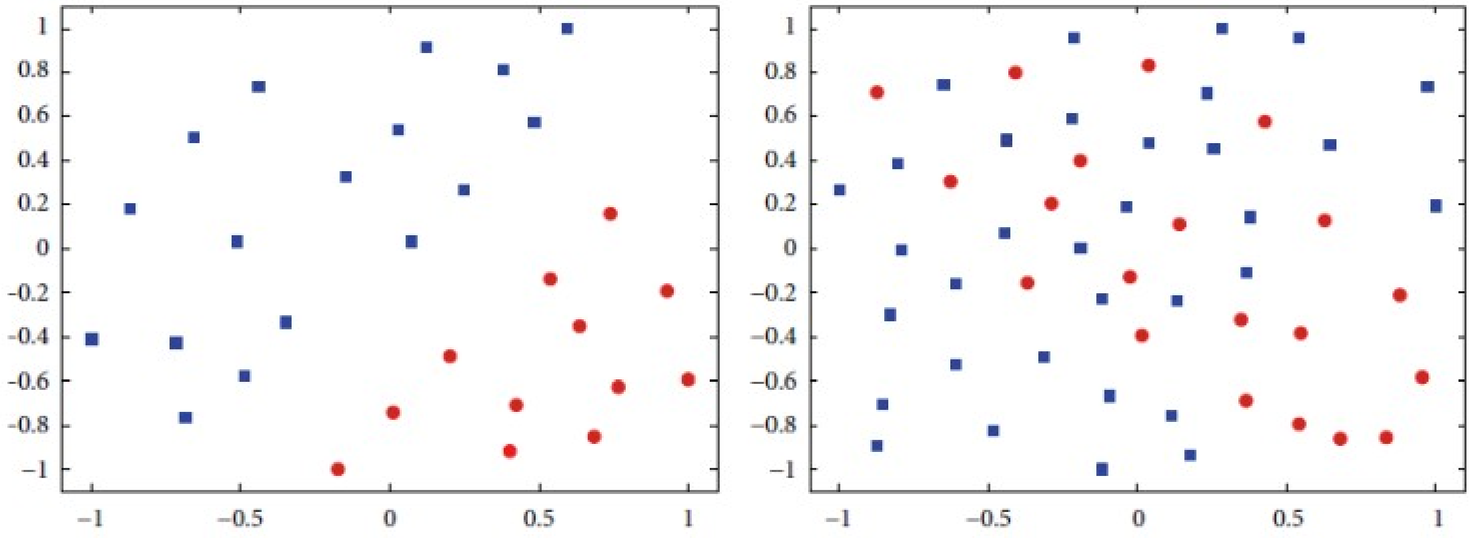
A good separation is achieved by the **hyperplane** that has the **largest distance** to the neighbouring data points of both classes.

- **The vectors (points)** that **constrain the width of the margin** are the support vectors.
- **Support vectors** are the data points that **lie closest to the decision surface**



An SVM analysis finds the line (or, in general, hyperplane) that is oriented so that the **margin between the support vectors is maximized**.
In the figure above, Solution 2 is superior to Solution 1 because it **has a larger margin.**

## SVM Machine Learning algorithm

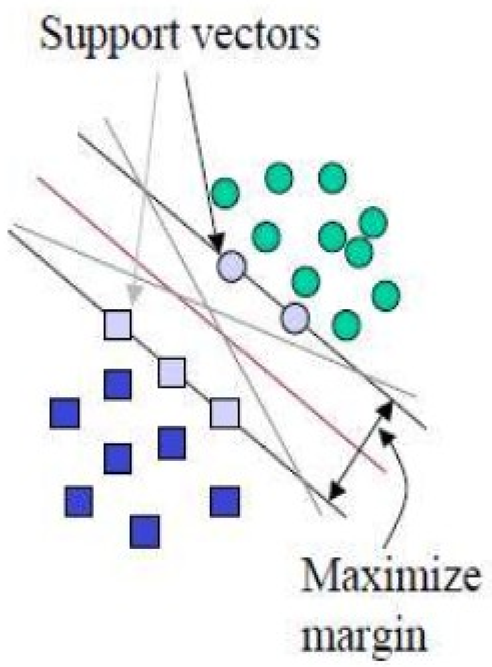


Which one is easy to separate?

## SVM Machine Learning algorithm

SVMs maximize the **margin around the separating hyperplane.**

- The decision function is fully specified by a subset of training samples, the support vectors.
- 2-Ds, it's a **line.**
- 3-Ds, it's a **plane.**
- In more dimensions, call it a hyperplane.



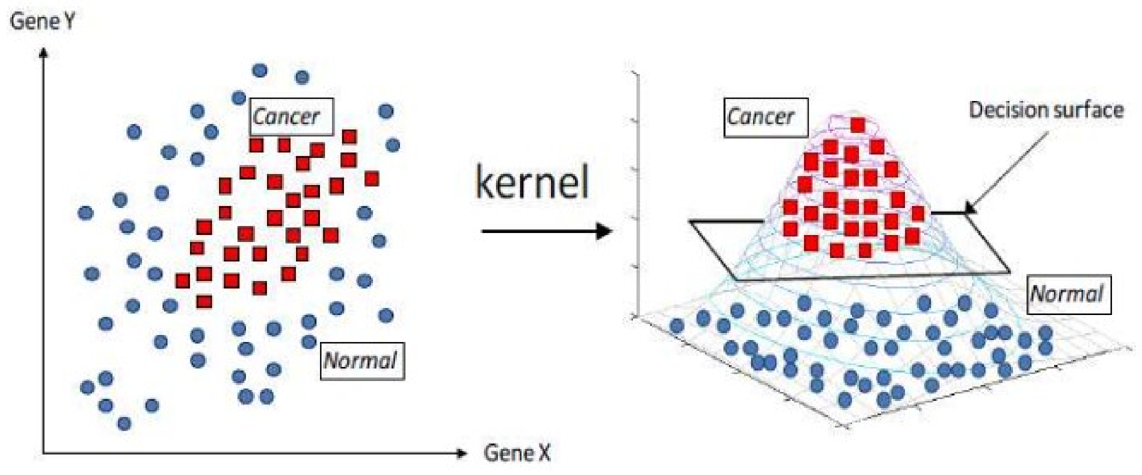
# SVM Machine Learning algorithm

Basic idea of support vector machines:

– **hyperplane** for linearly separable patterns

-A hyperplane is a **linear decision surface** that splits the space into two parts

– For non-linearly separable data-- t**ransformations of original data to map** into new space – the Kernel function



## SVM Machine Learning algorithm

Important because of:

– Robust to very large number of **variables and small samples**

– Can learn both simple and highly complex classification models

– Employ sophisticated mathematical principles to **avoid overfitting**

– Can be used for **both classification and regression** tasks

-Effective in cases of limited data.

## SVM Implementation Python

Scenario

**Worldwide, breast cancer is the most common type of cancer** in women and the second highest in terms of mortality rates. Diagnosis of breast cancer is performed when an abnormal lump is found (from self-examination or x-ray) or a tiny speck of calcium is seen (on an x-ray).

After a suspicious lump is found, the doctor will **conduct a diagnosis to determine** whether it is cancerous or not

## Naïve Bayes ML Algorithm

- Naïve Bayes Classifier is one of the **simplest and most effective** Classification algorithms which helps in building the **fast machine learning models** that can make **quick predictions**.
- It is mainly used in **text classification** that includes a high-dimensional training dataset.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

# Naïve Bayes ML Algorithm

## Bayes Theorem

- Let X be a data tuple.
- Let H be some hypothesis such as that the data tuple X belongs to a specified class C.
- We want to determine P(H|X)– posterior probability of H conditioned on X
- We are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

# Naïve Bayes ML Algorithm

Bayes theorem provides a way of computing posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$\underset{\text{Posterior Probability}}{P(c \mid x)} = \frac{\overset{\text{Likelihood}}{P(x \mid c)}\,\overset{\text{Class Prior Probability}}{P(c)}}{\underset{\text{Predictor Prior Probability}}{P(x)}}$$

$$P(\text{Class} \mid \text{Features}) = \frac{P(\text{Features} \mid \text{Class}) \times P(\text{Class})}{P(\text{Features})}$$

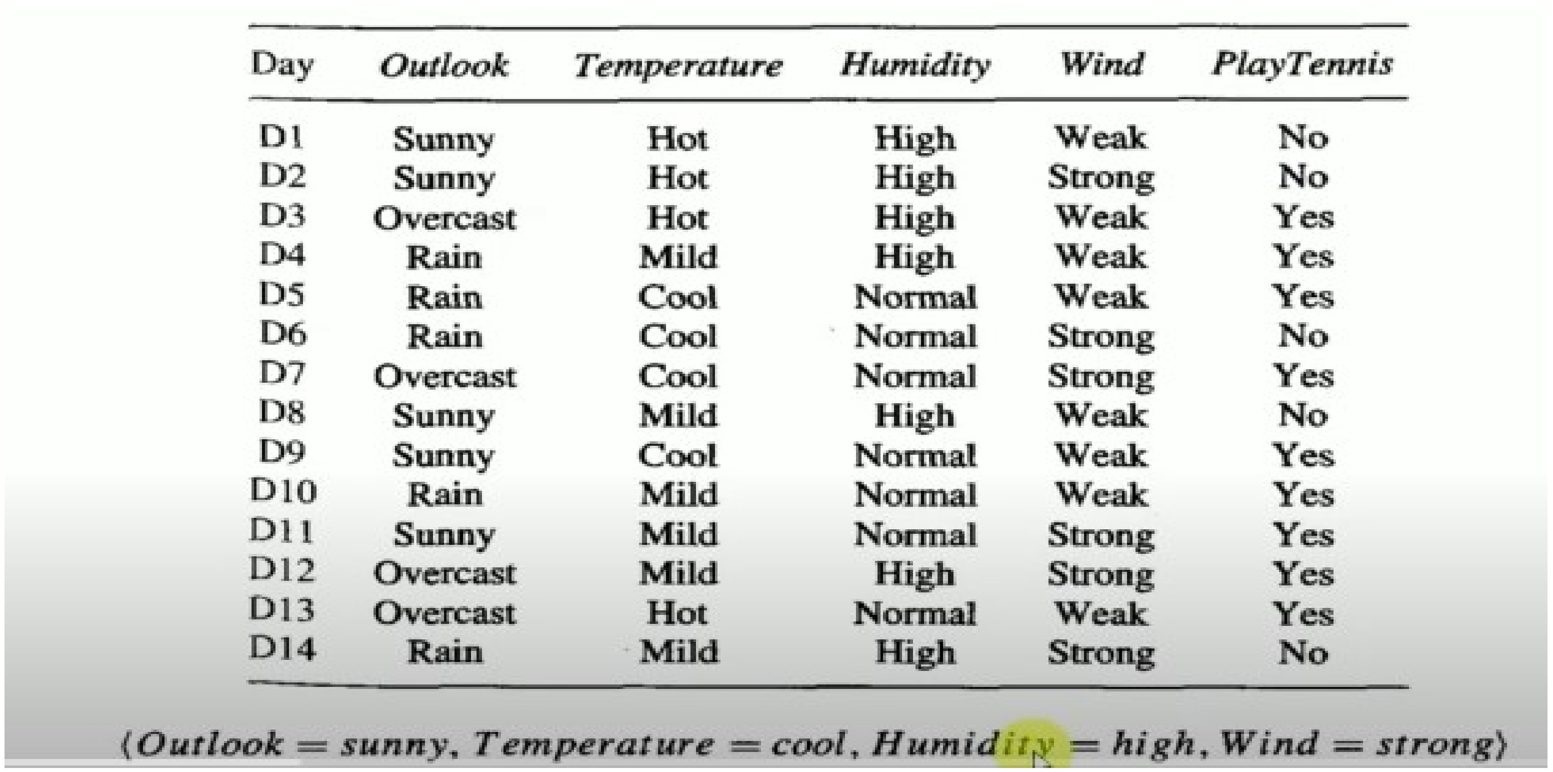
$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- *P*(*c*/*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*/*c*) is the likelihood which is the probability of the *predictor* given *class*.
- *P*(*x*) is the prior probability of the *predictor*.

## Example : Naïve Bayes ML

**Problem:** using the given data set , classify or predict weather a person with the given condition will play tennis or not?

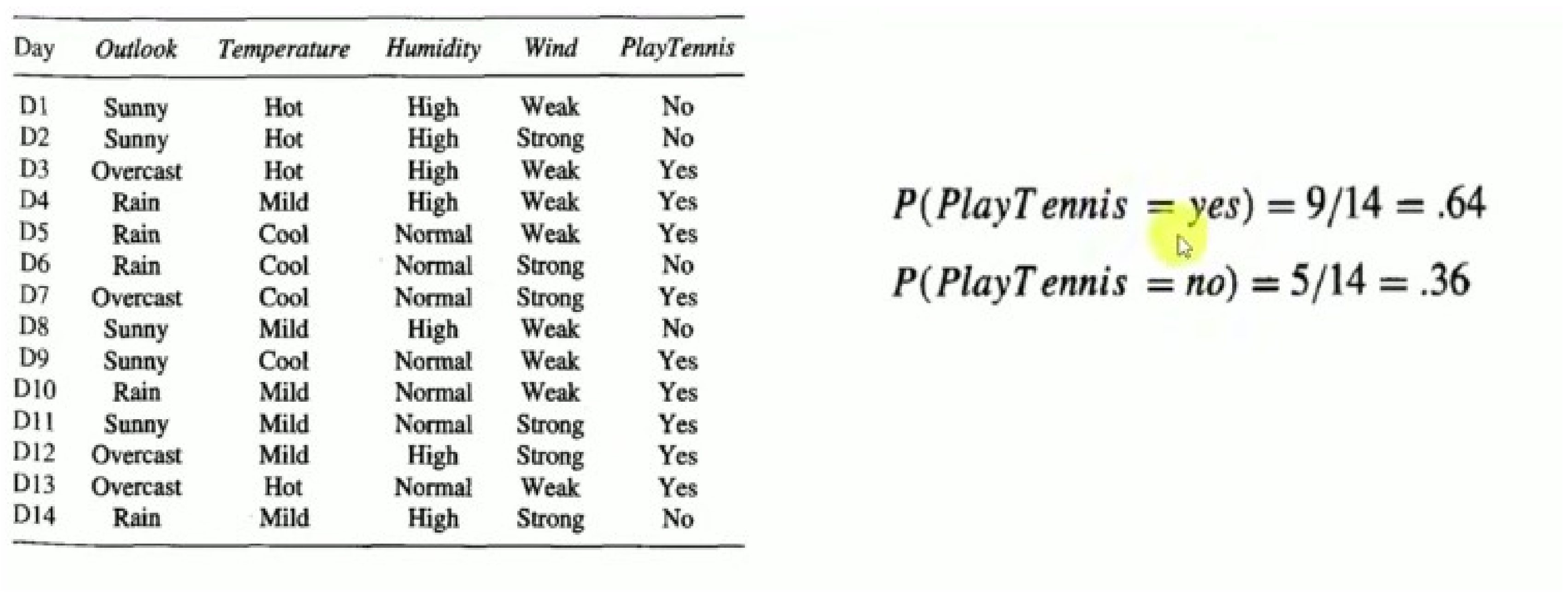


| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* | *PlayTennis* |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | 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 |

$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong\rangle$

# Example : Naïve Bayes ML

Step-1 calculate the prior/class label probability for Yes / No conditions Yes appeared 9 , and no appeared 5 out of 14 probability



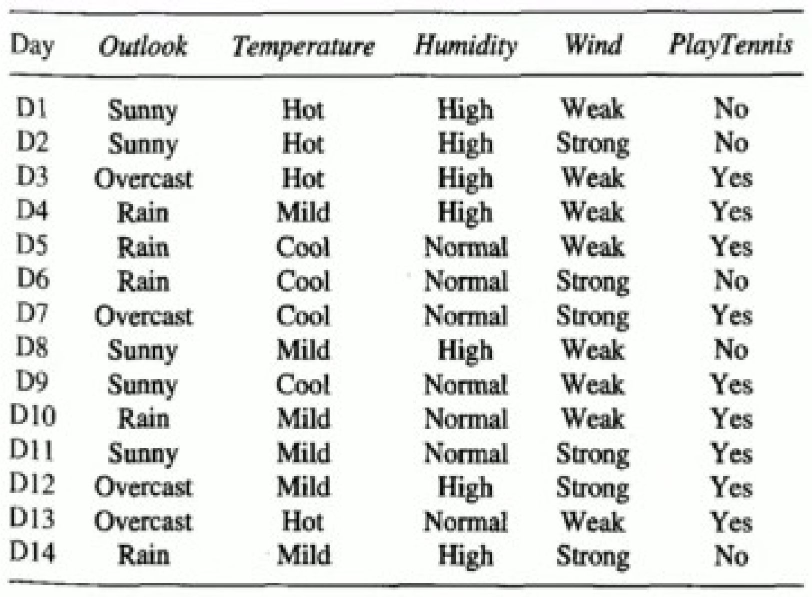
| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* | *PlayTennis* |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | 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 |

$$P(PlayTennis = yes) = 9/14 = .64$$

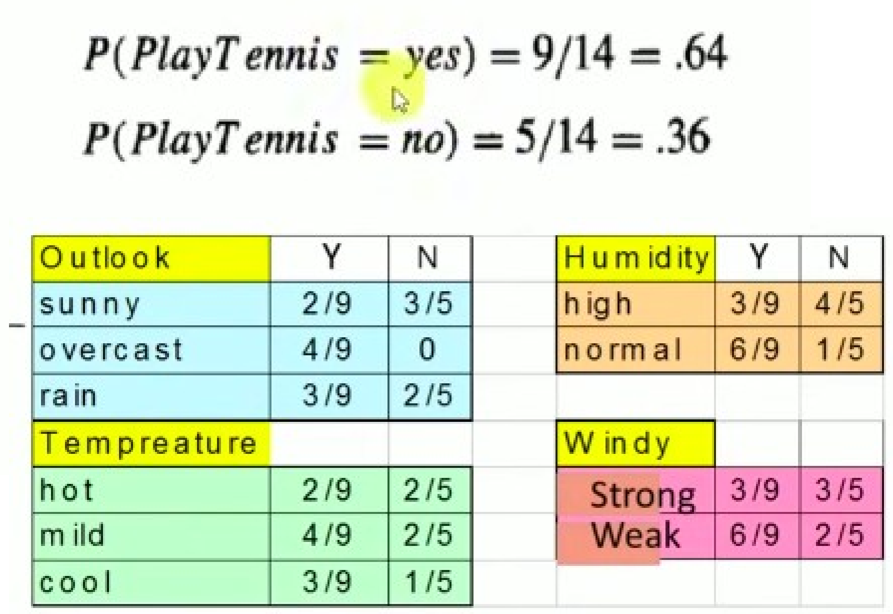
$$P(PlayTennis = no) = 5/14 = .36$$

# Example : Naïve Bayes ML

**Step-2 calculate** the conditional probability of individual attributes/predictors(outlook , temperature,Humidity,Windy)



| 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 |



$$P(PlayTennis = yes) = 9/14 = .64$$

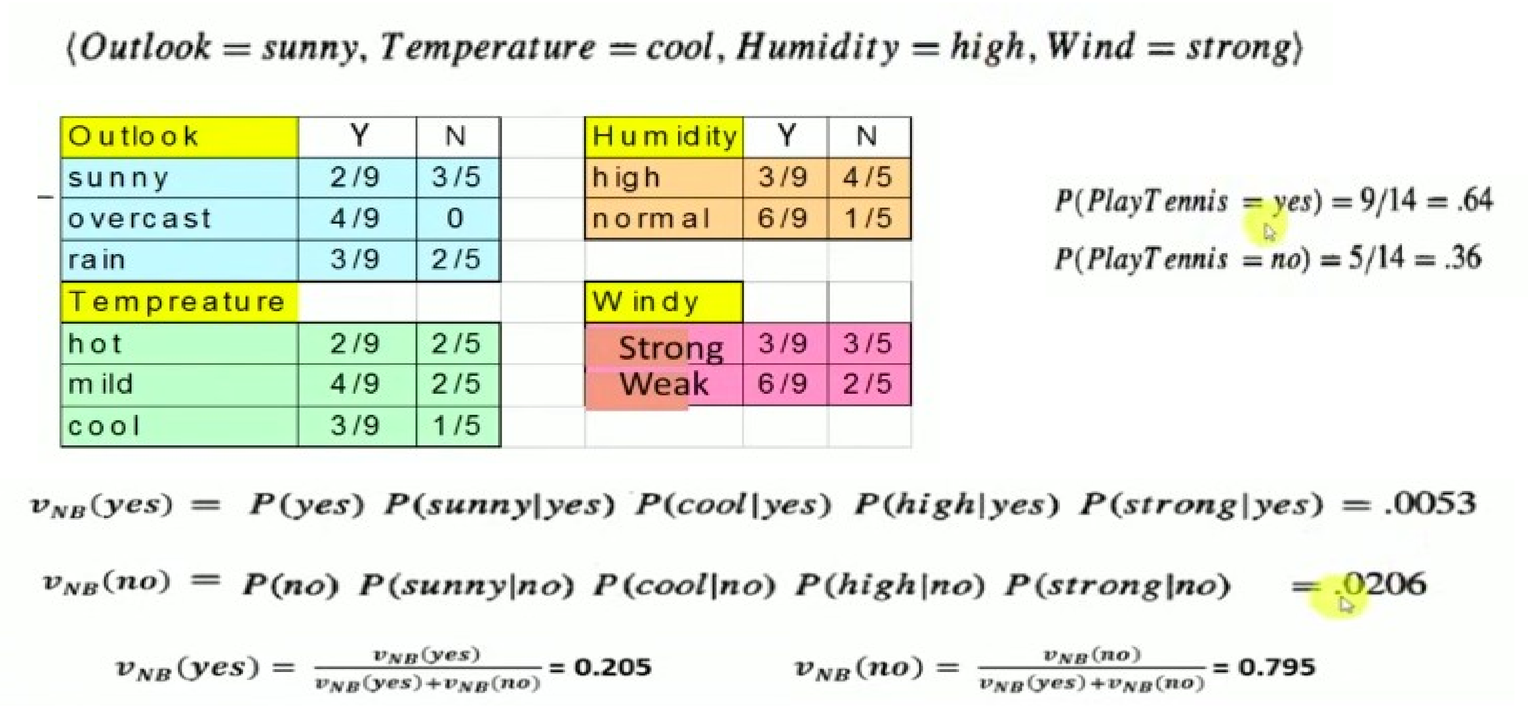
$$P(PlayTennis = no) = 5/14 = .36$$

| Outlook | Y | N |
|---|---|---|
| sunny | 2/9 | 3/5 |
| overcast | 4/9 | 0 |
| rain | 3/9 | 2/5 |
| **Tempreature** | | |
| hot | 2/9 | 2/5 |
| mild | 4/9 | 2/5 |
| cool | 3/9 | 1/5 |

| Humidity | Y | N |
|---|---|---|
| high | 3/9 | 4/5 |
| normal | 6/9 | 1/5 |
| **Windy** | | |
| Strong | 3/9 | 3/5 |
| Weak | 6/9 | 2/5 |

# Example : Naïve Bayes ML

Step-3 apply naive bayes formula to find new instance classification: sum up yes_ conditional probabilities of all feature and no probabilities , then compare the value lastly normalize it



$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong \rangle$

| Outlook | Y | N |
|---|---|---|
| sunny | 2/9 | 3/5 |
| overcast | 4/9 | 0 |
| rain | 3/9 | 2/5 |
| Tempreature | | |
| hot | 2/9 | 2/5 |
| mild | 4/9 | 2/5 |
| cool | 3/9 | 1/5 |

| Humidity | Y | N |
|---|---|---|
| high | 3/9 | 4/5 |
| normal | 6/9 | 1/5 |
| | | |
| Windy | | |
| Strong | 3/9 | 3/5 |
| Weak | 6/9 | 2/5 |

$P(PlayTennis = yes) = 9/14 = .64$

$P(PlayTennis = no) = 5/14 = .36$

$$v_{NB}(yes) = P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$

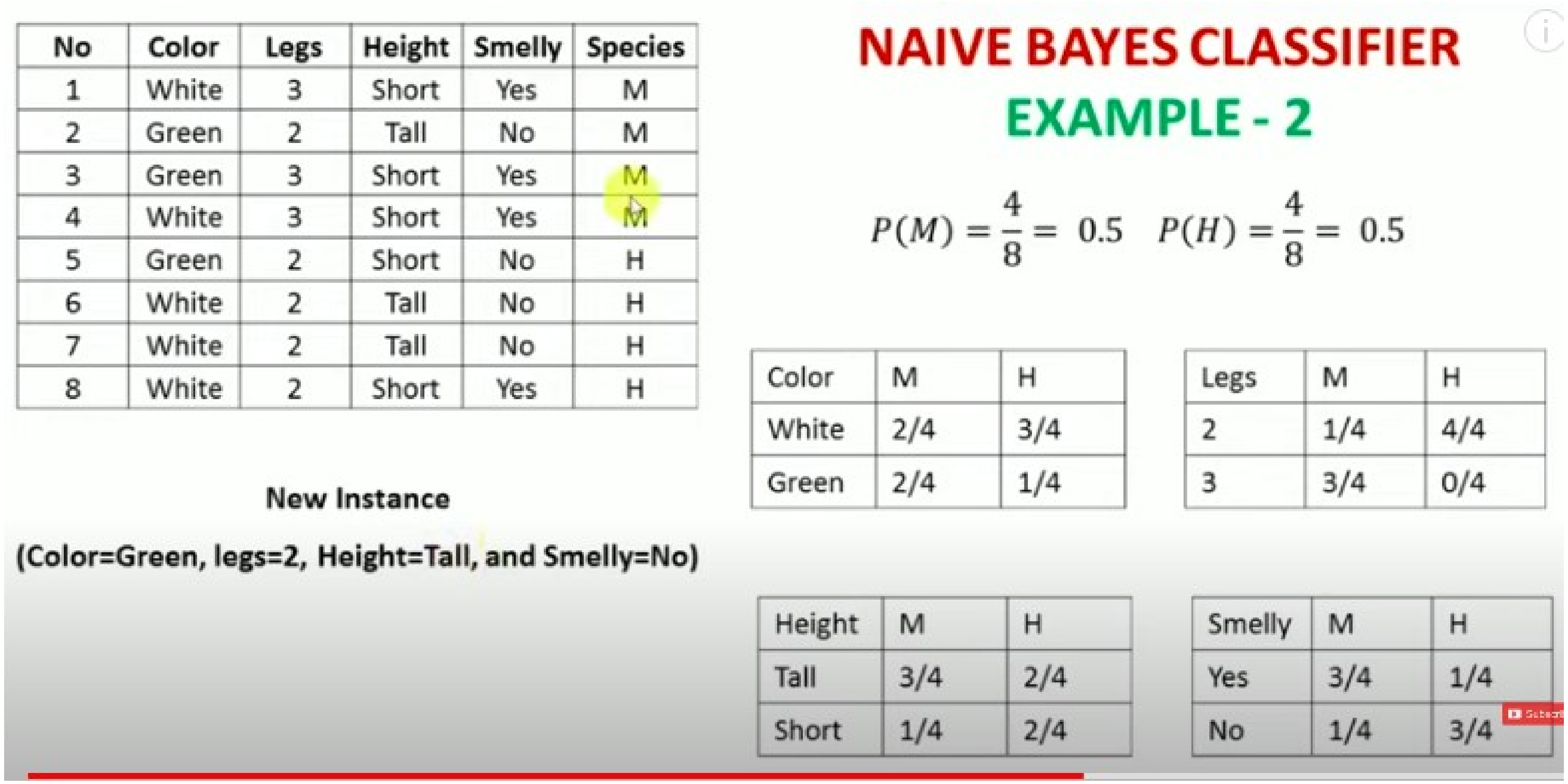
$$v_{NB}(no) = P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

$$v_{NB}(yes) = \frac{v_{NB}(yes)}{v_{NB}(yes)+v_{NB}(no)} = 0.205 \qquad v_{NB}(no) = \frac{v_{NB}(no)}{v_{NB}(yes)+v_{NB}(no)} = 0.795$$

Finaly , we can conclude that with the given features person **will not play tennis**

# Example : Naïve Bayes ML

Step-3 based on the following classify the new species ?



| No | Color | Legs | Height | Smelly | Species |
|---|---|---|---|---|---|
| 1 | White | 3 | Short | Yes | M |
| 2 | Green | 2 | Tall | No | M |
| 3 | Green | 3 | Short | Yes | M |
| 4 | White | 3 | Short | Yes | M |
| 5 | Green | 2 | Short | No | H |
| 6 | White | 2 | Tall | No | H |
| 7 | White | 2 | Tall | No | H |
| 8 | White | 2 | Short | Yes | H |

**New Instance**

**(Color=Green, legs=2, Height=Tall, and Smelly=No)**

**NAIVE BAYES CLASSIFIER**

**EXAMPLE - 2**

$$P(M) = \frac{4}{8} = 0.5 \quad P(H) = \frac{4}{8} = 0.5$$

| Color | M | H |
|---|---|---|
| White | 2/4 | 3/4 |
| Green | 2/4 | 1/4 |

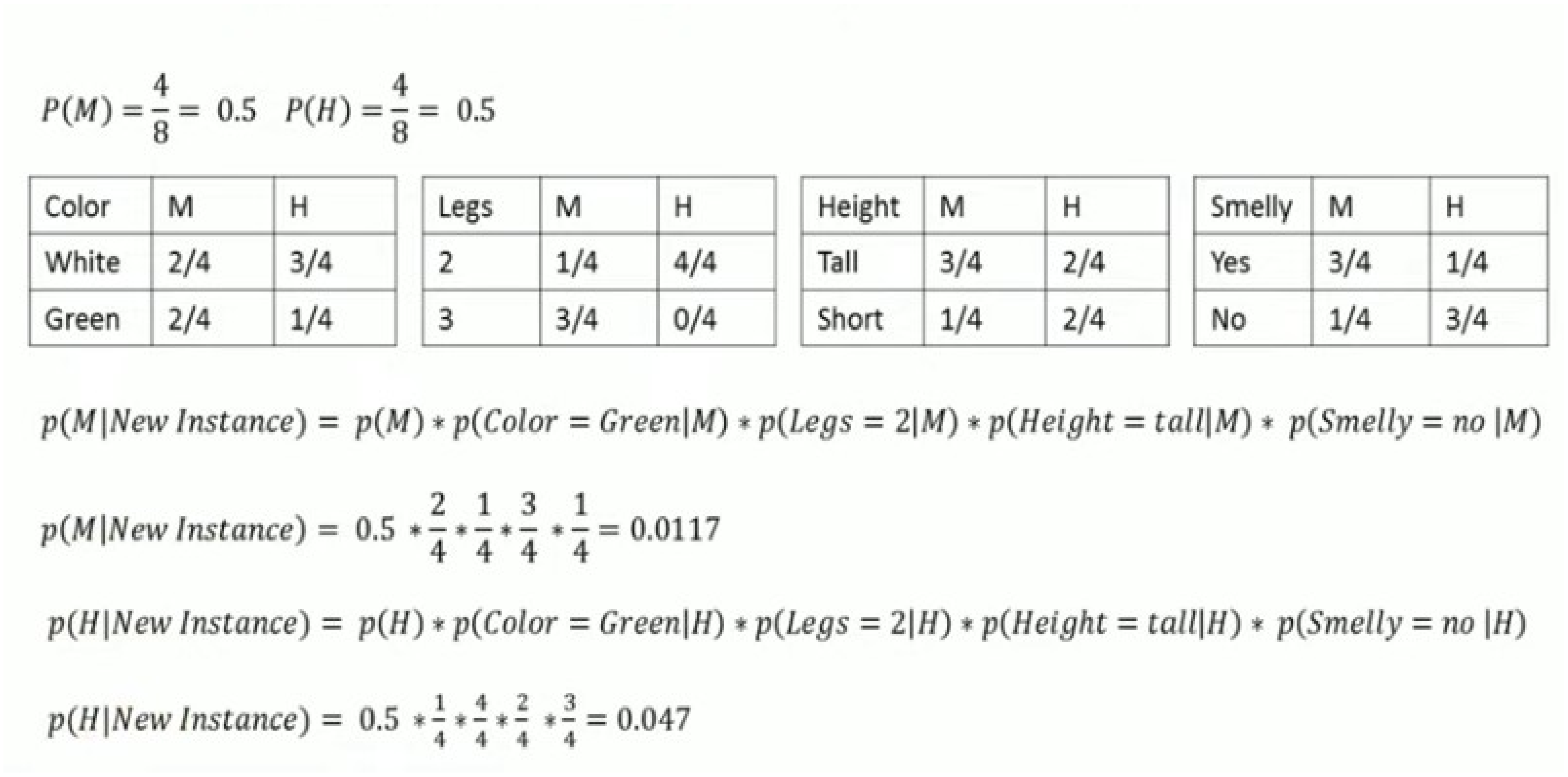
| Legs | M | H |
|---|---|---|
| 2 | 1/4 | 4/4 |
| 3 | 3/4 | 0/4 |

| Height | M | H |
|---|---|---|
| Tall | 3/4 | 2/4 |
| Short | 1/4 | 2/4 |

| Smelly | M | H |
|---|---|---|
| Yes | 3/4 | 1/4 |
| No | 1/4 | 3/4 |

# Example : Naïve Bayes ML

From the below we understand that the new instance to classified as H is higher than M , so the new instance is H ,



$P(M) = \frac{4}{8} = 0.5 \quad P(H) = \frac{4}{8} = 0.5$

| Color | M | H |
|---|---|---|
| White | 2/4 | 3/4 |
| Green | 2/4 | 1/4 |

| Legs | M | H |
|---|---|---|
| 2 | 1/4 | 4/4 |
| 3 | 3/4 | 0/4 |

| Height | M | H |
|---|---|---|
| Tall | 3/4 | 2/4 |
| Short | 1/4 | 2/4 |

| Smelly | M | H |
|---|---|---|
| Yes | 3/4 | 1/4 |
| No | 1/4 | 3/4 |

$$p(M|New\ Instance) = p(M) * p(Color = Green|M) * p(Legs = 2|M) * p(Height = tall|M) * p(Smelly = no\ |M)$$

$$p(M|New\ Instance) = 0.5 * \frac{2}{4} * \frac{1}{4} * \frac{3}{4} * \frac{1}{4} = 0.0117$$

$$p(H|New\ Instance) = p(H) * p(Color = Green|H) * p(Legs = 2|H) * p(Height = tall|H) * p(Smelly = no\ |H)$$

$$p(H|New\ Instance) = 0.5 * \frac{1}{4} * \frac{4}{4} * \frac{2}{4} * \frac{3}{4} = 0.047$$

## Example : Naïve Bayes ML

**Advantage**

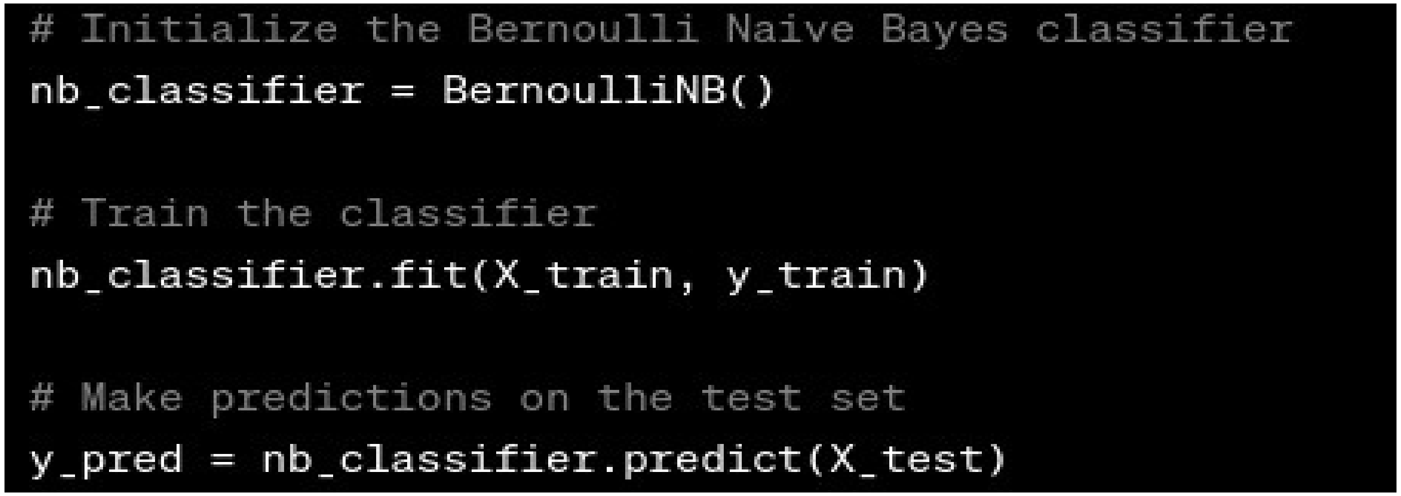
- Simple
- Incremental learning
- Naturally a probability estimator
- Easily handles missing values

**Disadvantage / Weakness**

- Independence assumption
- Categorical/discrete attributes
- Sensitive to missing values

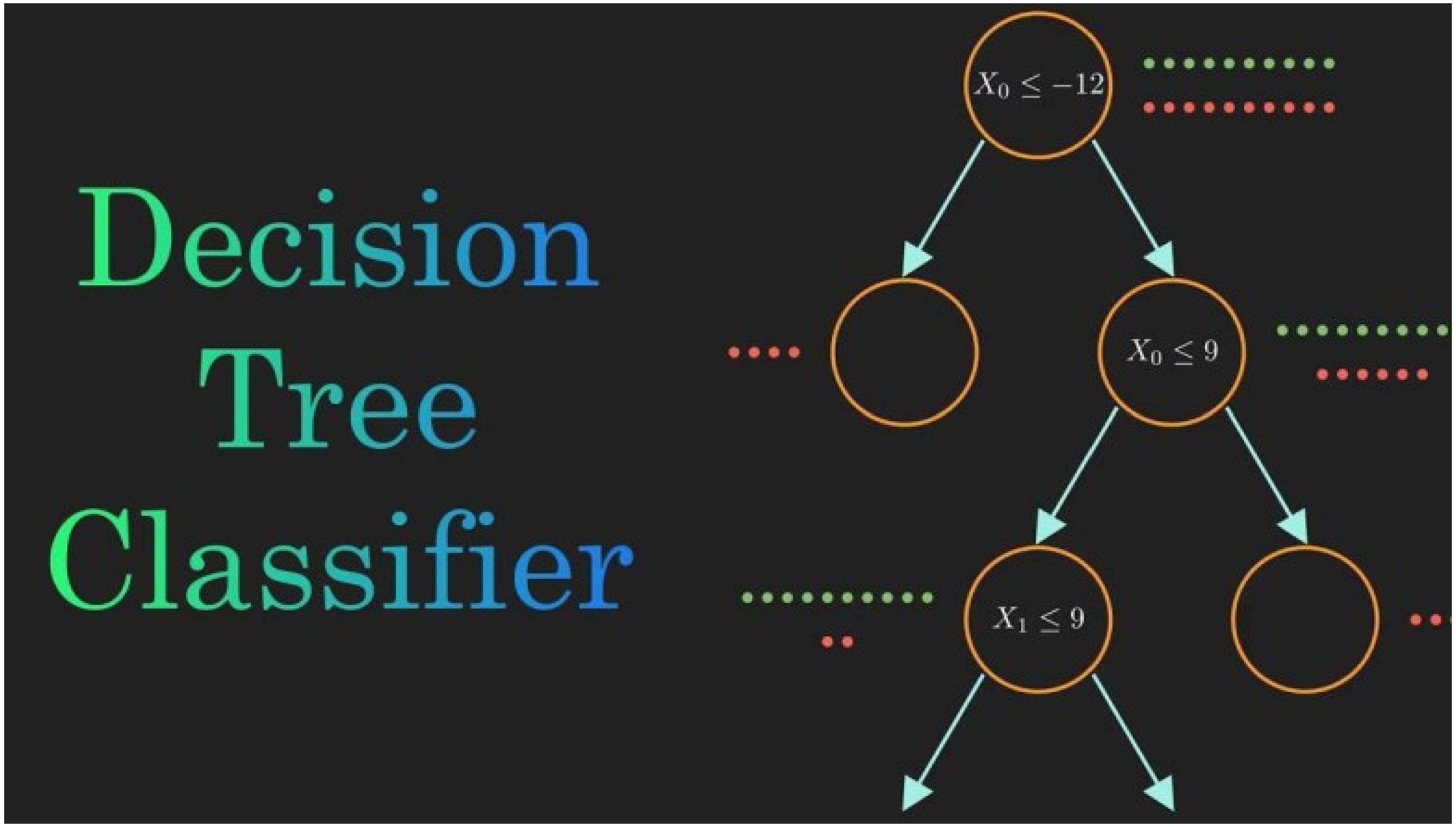
## Example : Naïve Bayes ML Python Implementation

from sklearn.naive_bayes import BernoulliNB



```
# Initialize the Bernoulli Naive Bayes classifier
nb_classifier = BernoulliNB()

# Train the classifier
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
```

## Decision Tree



## Definition

**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.**

- It is a tree-structured classifier, where

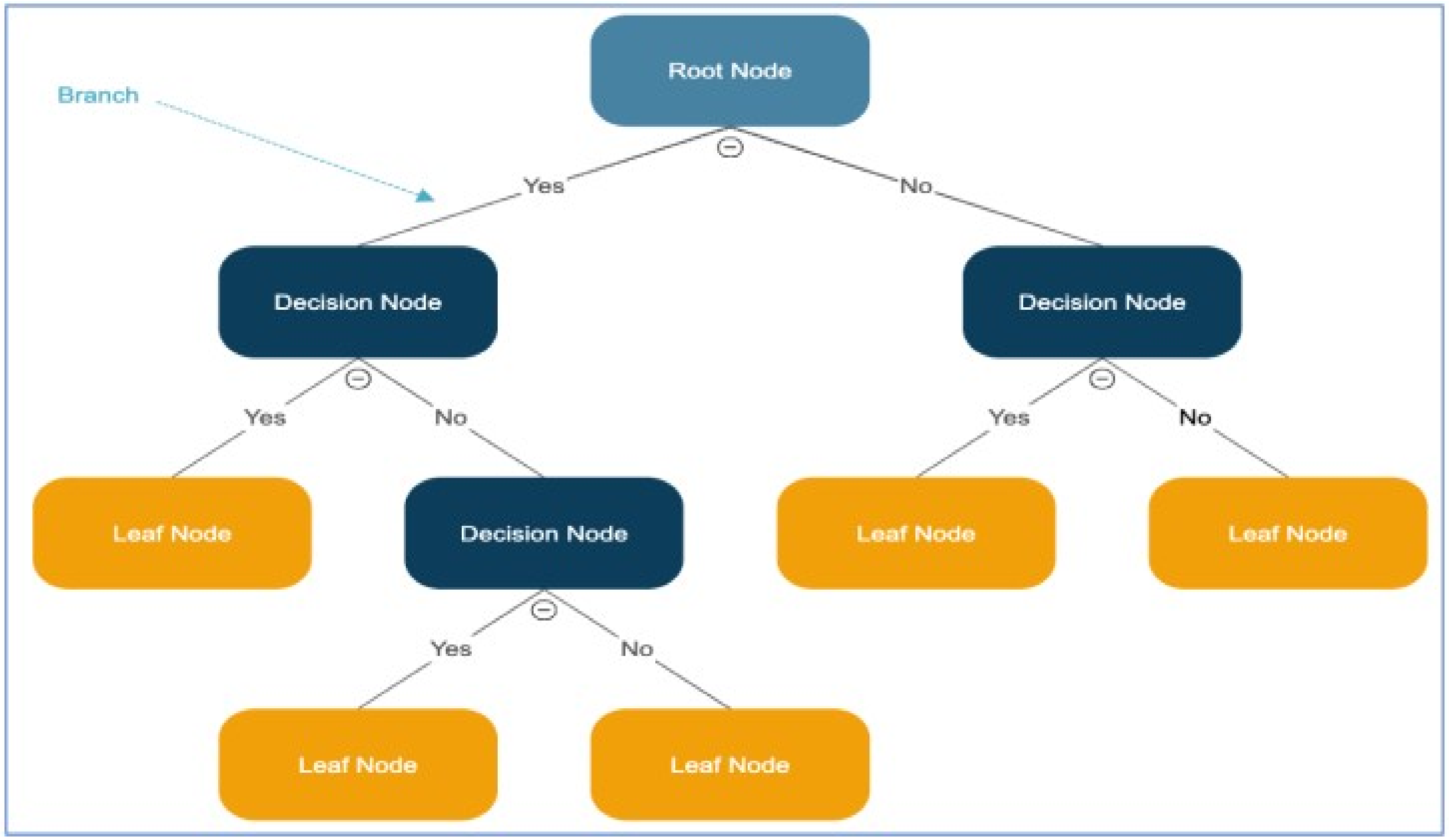
  **internal nodes** represent the features of a dataset,

  **branches** represent the decision rules and each

  **leaf node** represents the outcome.

## Decision Tree Cont..

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



## Why Decision Tree?

• 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 Terminology

- **Root Node:** Root node is from where the **decision tree starts.** It represents the **entire dataset,** which further gets divided into two or more **homogeneous 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.

## How Does the Decision Tree Algorithm Work?

• **Splitting**: The decision tree starts with the entire dataset at the root node. At each step, **it selects the best feature** to split the data into two or more subsets.

• The goal is to **create partitions that are as homogeneous** as possible with respect to the target variable for classification /regression.

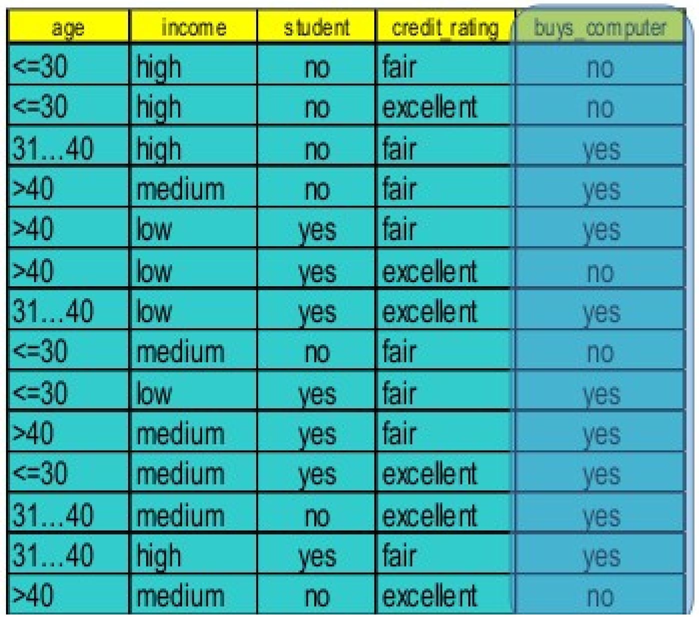
**Decision Making:** After splitting the data, the algorithm **makes decisions based on** the **values of the selected feature.**

This process continues recursively for each subset of data until a stopping criterion is met - **a maximum depth**

# Decision Tree -Example

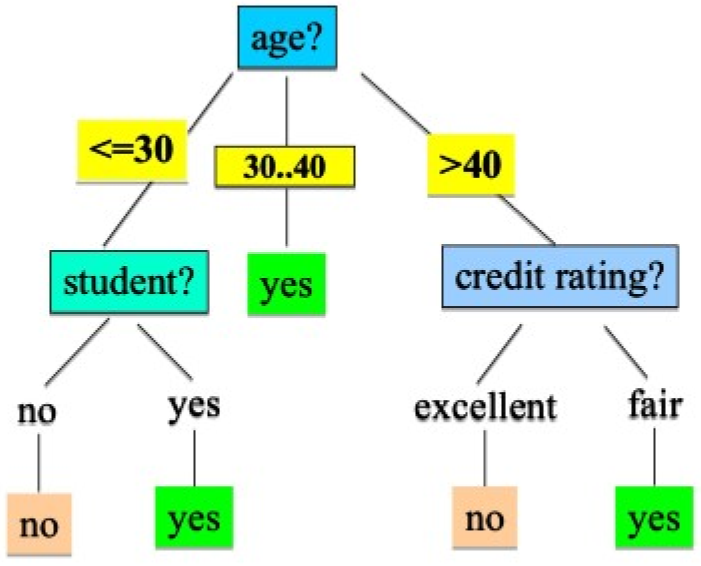
Output: A Decision Tree for "*buys_computer*"

**Training Dataset**



| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

**Decision Tree**



# Decision Tree-Algorithm

## Classification Rule Extraction

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

  IF *age* = "<=30" AND *student* = "*no*" THEN *buys_computer* = "*no*"
  IF *age* = "<=30" AND *student* = "*yes*" THEN *buys_computer* = "*yes*"
  IF *age* = "31…40" THEN *buys_computer* = "*yes*"
  IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN *buys_computer* = "*yes*"
  IF *age* = ">40" AND *credit_rating* = "*fair*" THEN *buys_computer* = "*no*"

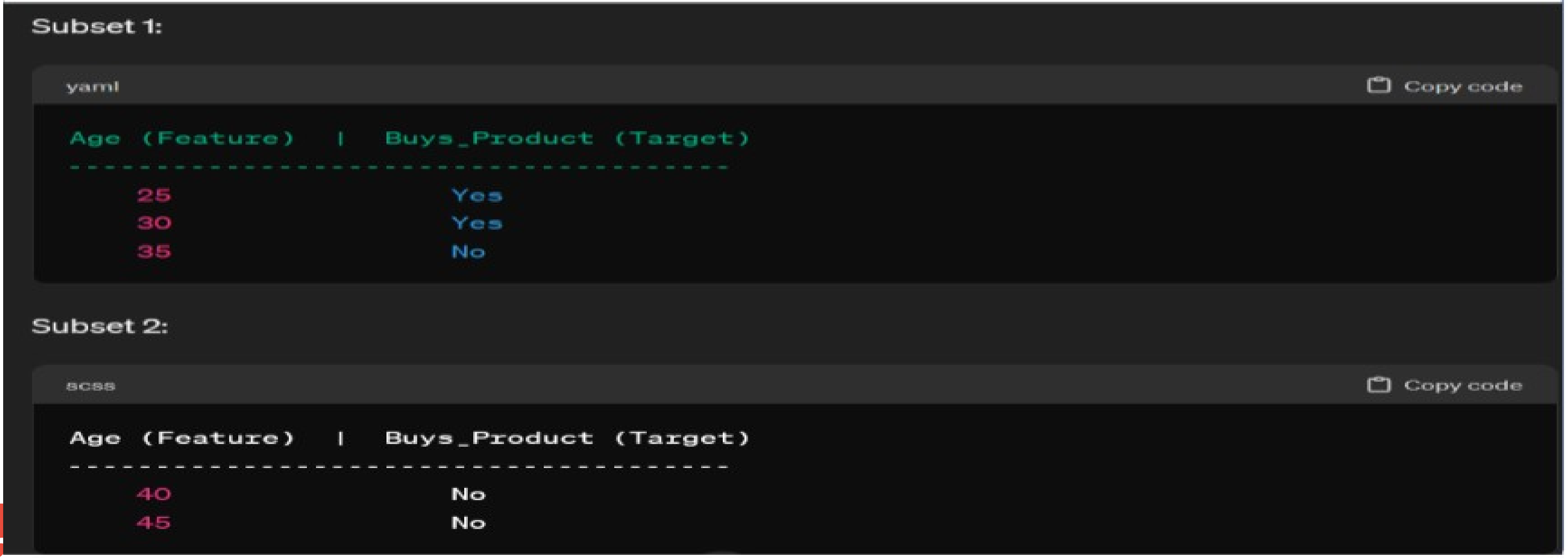
## Pure Vs Impure Split

- **A pure split** is achieved when **all the samples in a node belong** to the same class (for classification tasks) or have the **same value** (for regression tasks). In other words, after splitting the data based on a particular feature, all samples in the resulting subsets **are homogeneous with respect to the target variable**.
- **impure split** occurs when the resulting subsets after splitting the data based on a feature **contain samples from different classes**  In other words, the **subsets are not homogeneous** with respect to the target variable.
- **Example**: Consider a binary classification problem where we want to predict whether a person will buy a product based on their age. Suppose we have the following dataset

## Pure Vs Impure Split

| Age (Feature) | Buy Product (Target) |
|---|---|
| 25 | Yes |
| 30 | Yes |
| 35 | No |
| 40 | No |
| 45 | No |

- If we split the data based on the **age <= 35**, we'll have two subsets:



Subset 1:

```yaml
Age (Feature)   |   Buys_Product (Target)
------------------------------------
     25                   Yes
     30                   Yes
     35                   No
```

Subset 2:

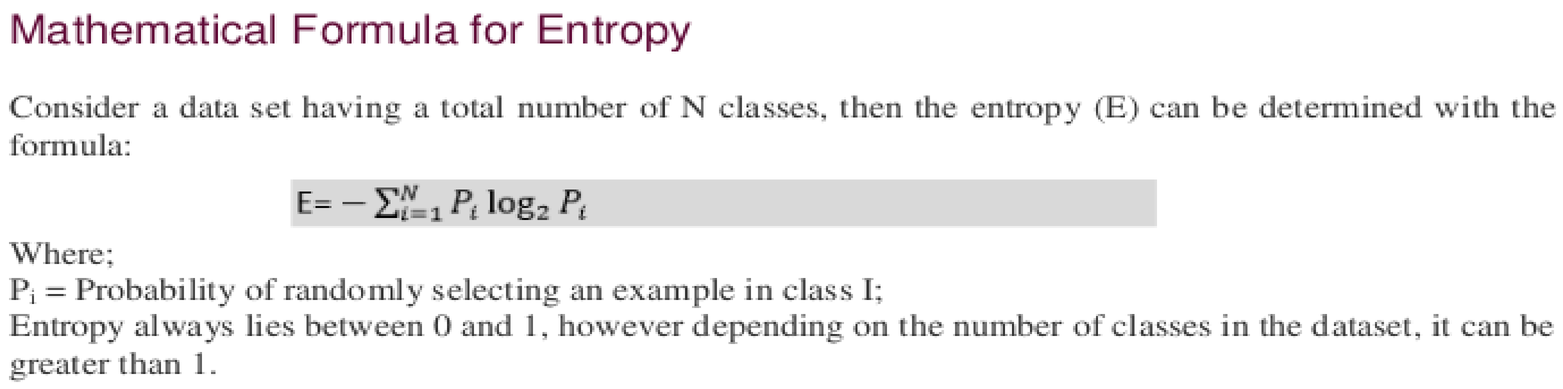
```scss
Age (Feature)   |   Buys_Product (Target)
------------------------------------
     40                   No
     45                   No
```

## Entropy Vs Gini Impurity

- **Entropy and Gini** Impurity are **two commonly used measures of impurity** in decision tree algorithms.

- Both measures are used to **determine the best split** at each node of the decision tree by **quantifying the uncertainty or randomness** of the data.

- Entropy is a measure of **randomness or impurity** in a dataset. It is calculated using the formula:



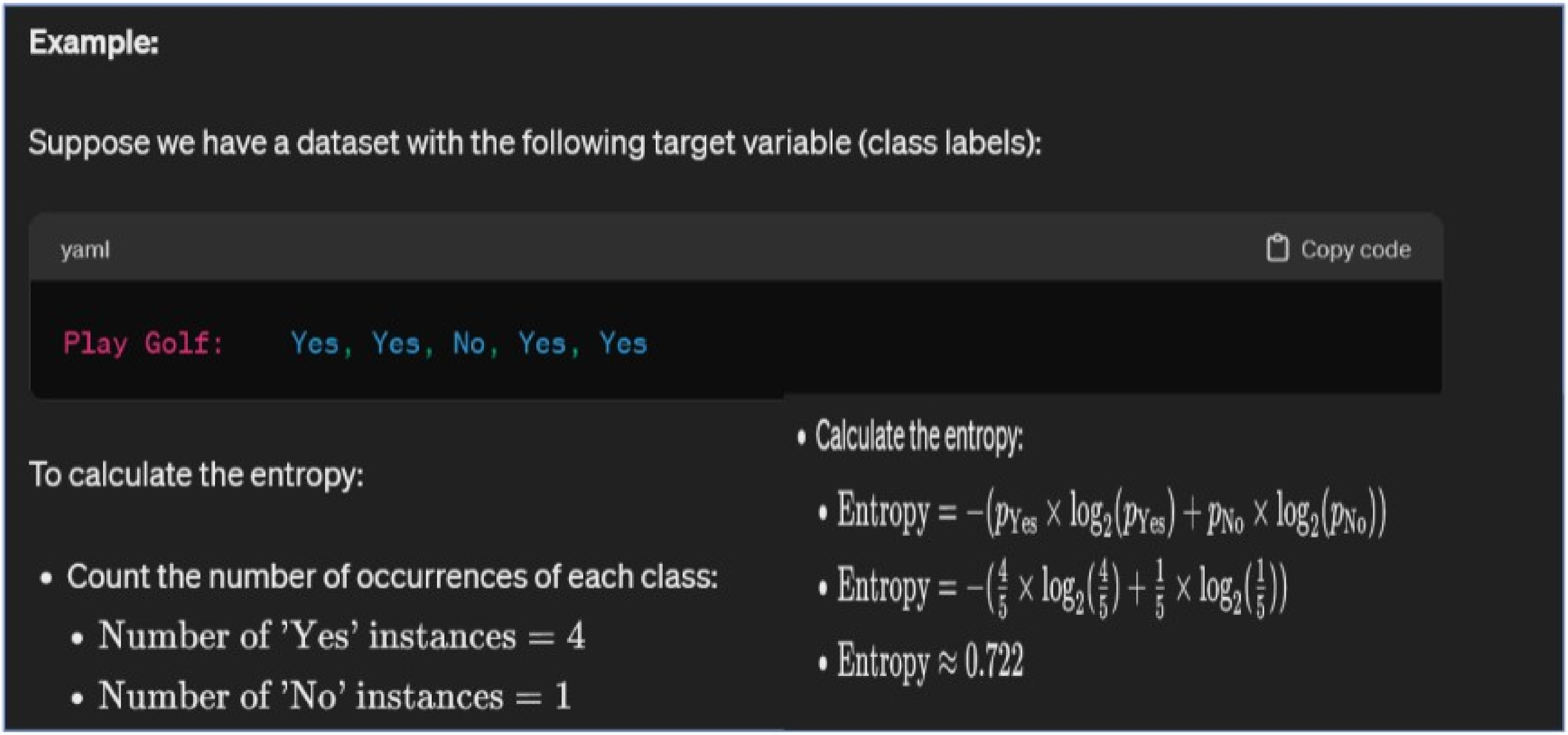
### Mathematical Formula for Entropy

Consider a data set having a total number of N classes, then the entropy (E) can be determined with the formula:

$$E= -\sum_{i=1}^{N} P_i \log_2 P_i$$

Where;
$P_i$ = Probability of randomly selecting an example in class I;
Entropy always lies between 0 and 1, however depending on the number of classes in the dataset, it can be greater than 1.

# Entropy example



**Example:**

Suppose we have a dataset with the following target variable (class labels):

```yaml
Play Golf:    Yes, Yes, No, Yes, Yes
```

To calculate the entropy:

- Count the number of occurrences of each class:
  - Number of 'Yes' instances $= 4$
  - Number of 'No' instances $= 1$
- Calculate the entropy:
  - $\text{Entropy} = -(p_{\text{Yes}} \times \log_2(p_{\text{Yes}}) + p_{\text{No}} \times \log_2(p_{\text{No}}))$
  - $\text{Entropy} = -(\frac{4}{5} \times \log_2(\frac{4}{5}) + \frac{1}{5} \times \log_2(\frac{1}{5}))$
  - $\text{Entropy} \approx 0.722$

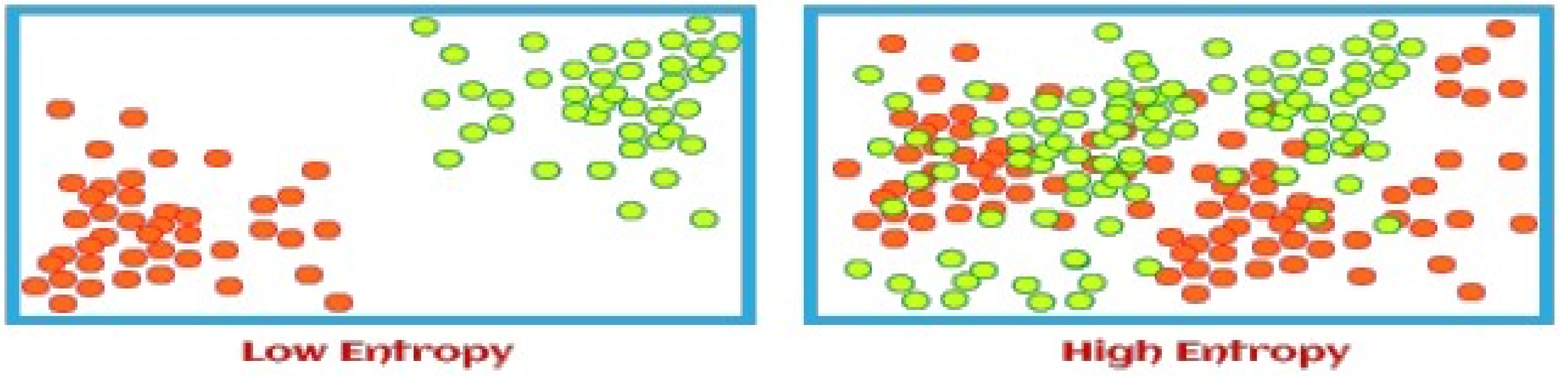
# Entropy cont..

- In order to build a tree, we use the **CART algorithm,** which stands for Classification and

Regression Tree algorithm.

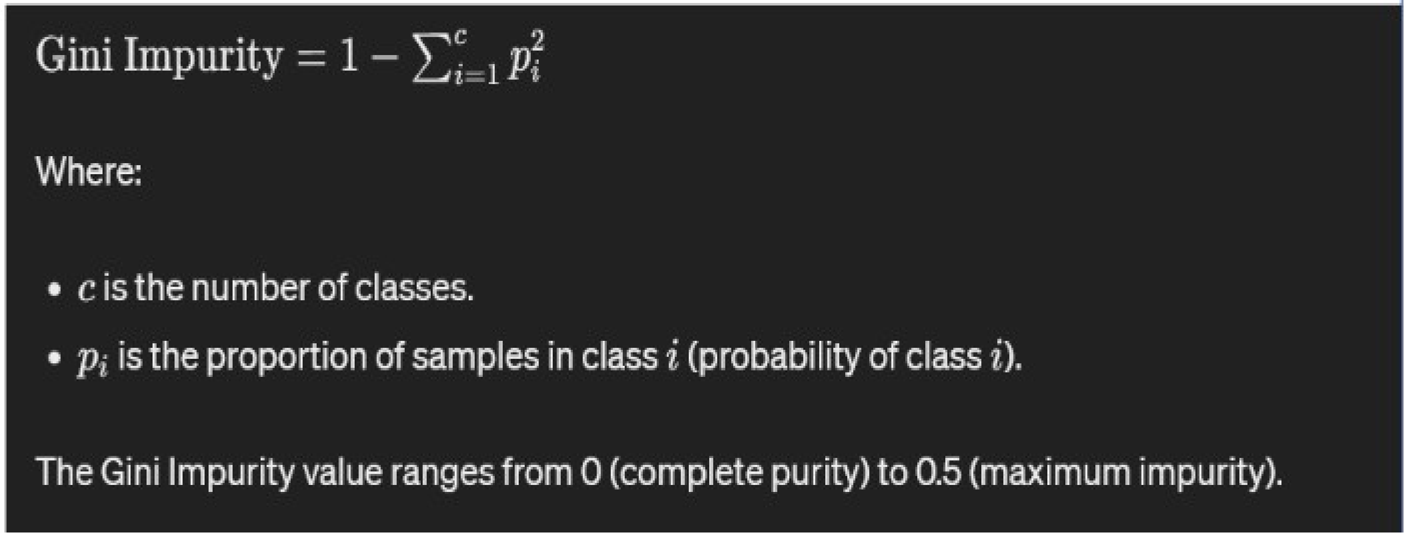
- every piece of information has a specific value to make and can be used **to draw conclusions from it.**

- Entropy is higher=> **difficult to draw any conclusion from that piece of information**.



## Gini Impurity

- **Gini Impurity** measures the probability of **incorrectly classifying a randomly** chosen sample's label if it were randomly labeled according to the class distribution in the dataset. It is calculated using the formula:

- 

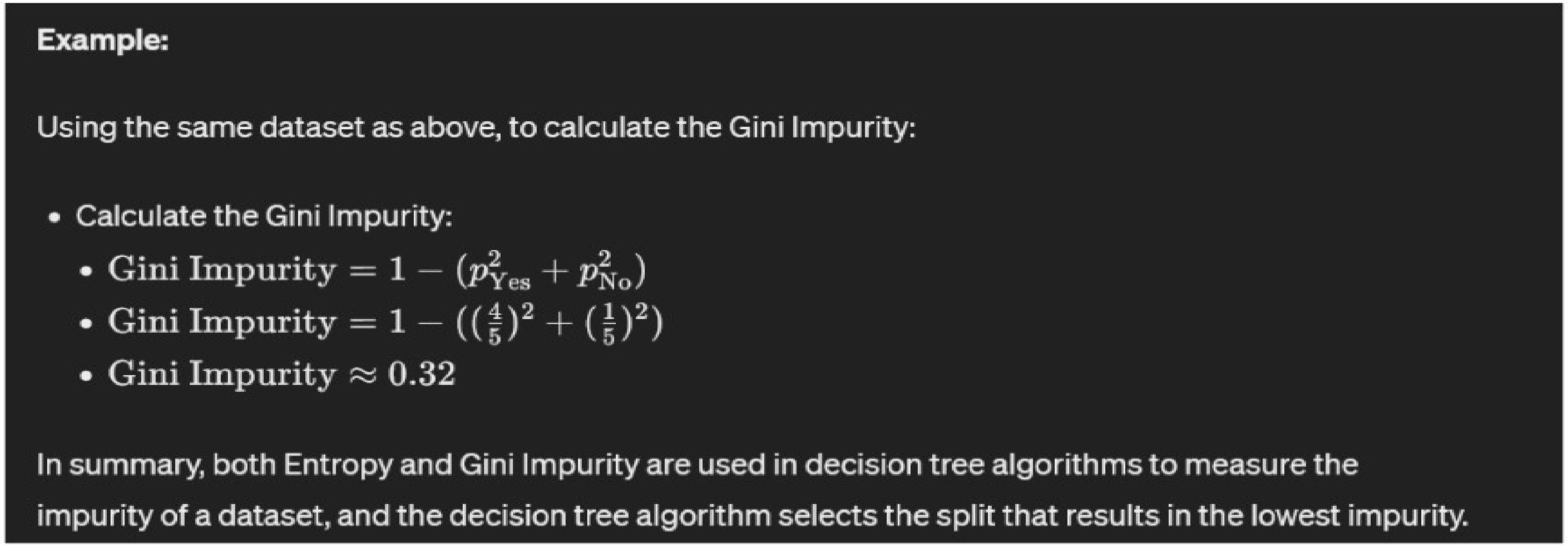
$$\text{Gini Impurity} = 1 - \sum_{i=1}^{c} p_i^2$$

Where:

- $c$ is the number of classes.
- $p_i$ is the proportion of samples in class $i$ (probability of class $i$).

The Gini Impurity value ranges from 0 (complete purity) to 0.5 (maximum impurity).

## Gini impurity example



**Example:**

Using the same dataset as above, to calculate the Gini Impurity:

- Calculate the Gini Impurity:
  - $\text{Gini Impurity} = 1 - (p_{\text{Yes}}^2 + p_{\text{No}}^2)$
  - $\text{Gini Impurity} = 1 - ((\frac{4}{5})^2 + (\frac{1}{5})^2)$
  - $\text{Gini Impurity} \approx 0.32$

In summary, both Entropy and Gini Impurity are used in decision tree algorithms to measure the impurity of a dataset, and the decision tree algorithm selects the split that results in the lowest impurity.

- Let's consider a case when all observations belong to **the same class**; then **entropy will always be 0.**
- When **entropy becomes 0**, then the **dataset has no impurity**.
- Datasets with **0 impurities** are not useful **for learning.** Further, if the entropy is 1, **then this kind of dataset is good for learning.**

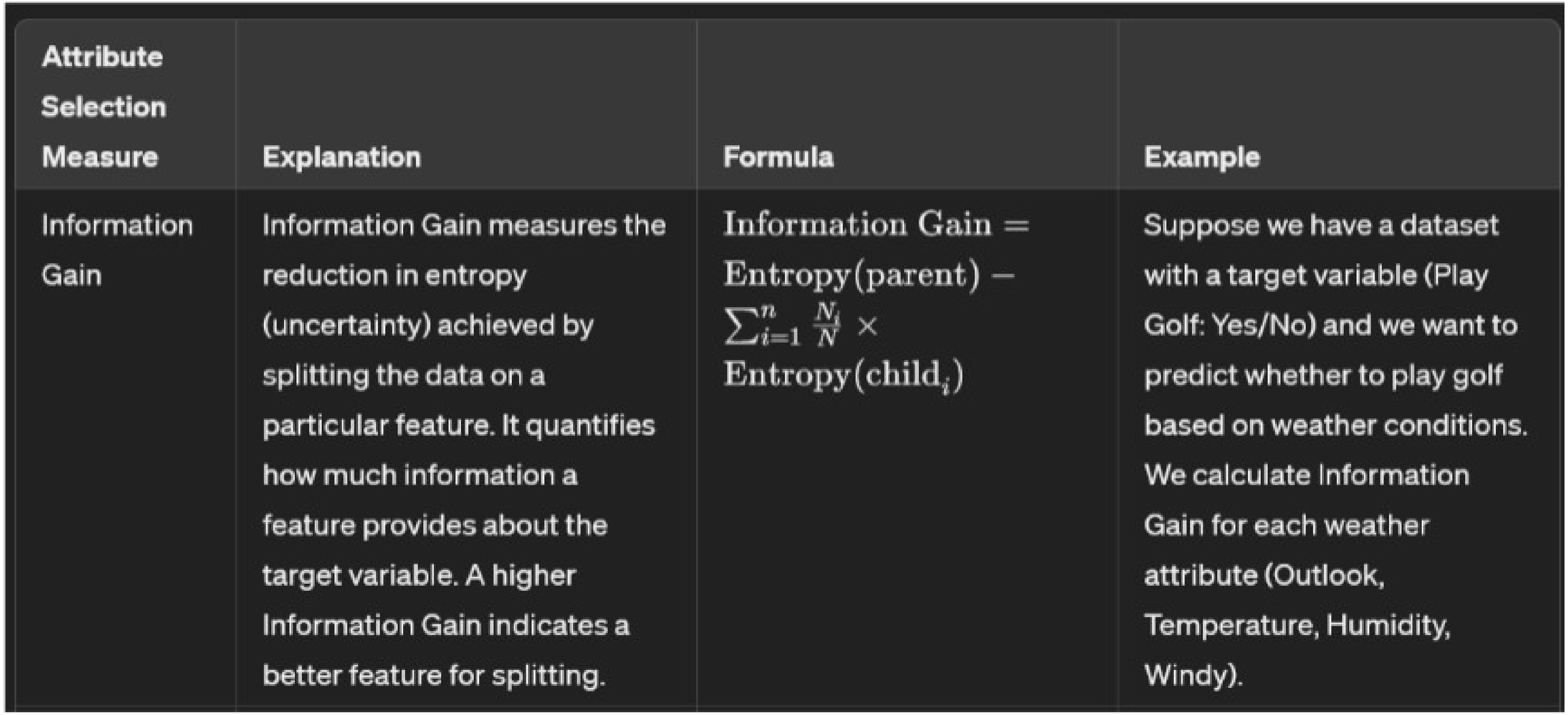
## Attribute Selection Measures (ASM)

In DT, the main issue arises that **how to select the best attribute** for the root node and for sub-nodes.

to solve such problems there is a technique which is called as Attribute Selection Measure (ASM). The are two popular methods

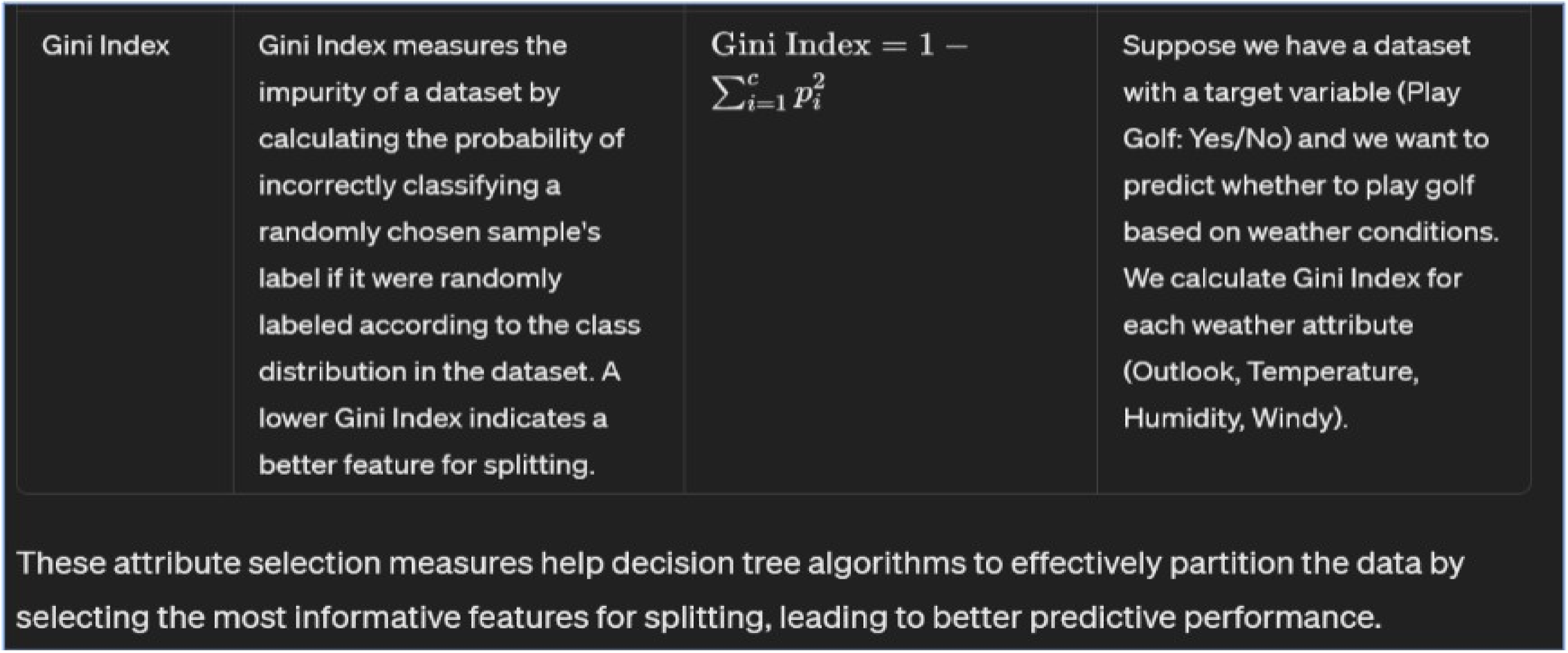
- **Information Gini**
  - ✓ is the **measurement of changes in entropy** after the segmentation dataset based on an attribute
  - ✓ According to the value of information gain, we **split the node and build the decision** tree.
  - ✓ DT algorithm always tries to maximize the value of **information gain,** and a
  - ✓ Node / attribute having the **highest information gain** is split first.
- **Gain Index:**
  - ✓ is a measure of **impurity** used while creating a decision tree in the CART(uses gain index for splitting )
  - ✓ An attribute with the **low Gini index** should be preferred as compared to the high Gini index.

# Information gain vs Gini Index



| Attribute Selection Measure | Explanation | Formula | Example |
|---|---|---|---|
| Information Gain | Information Gain measures the reduction in entropy (uncertainty) achieved by splitting the data on a particular feature. It quantifies how much information a feature provides about the target variable. A higher Information Gain indicates a better feature for splitting. | $\text{Information Gain} = \text{Entropy}(\text{parent}) - \sum_{i=1}^{n} \frac{N_i}{N} \times \text{Entropy}(\text{child}_i)$ | Suppose we have a dataset with a target variable (Play Golf: Yes/No) and we want to predict whether to play golf based on weather conditions. We calculate Information Gain for each weather attribute (Outlook, Temperature, Humidity, Windy). |

# Information gain vs Gini Index

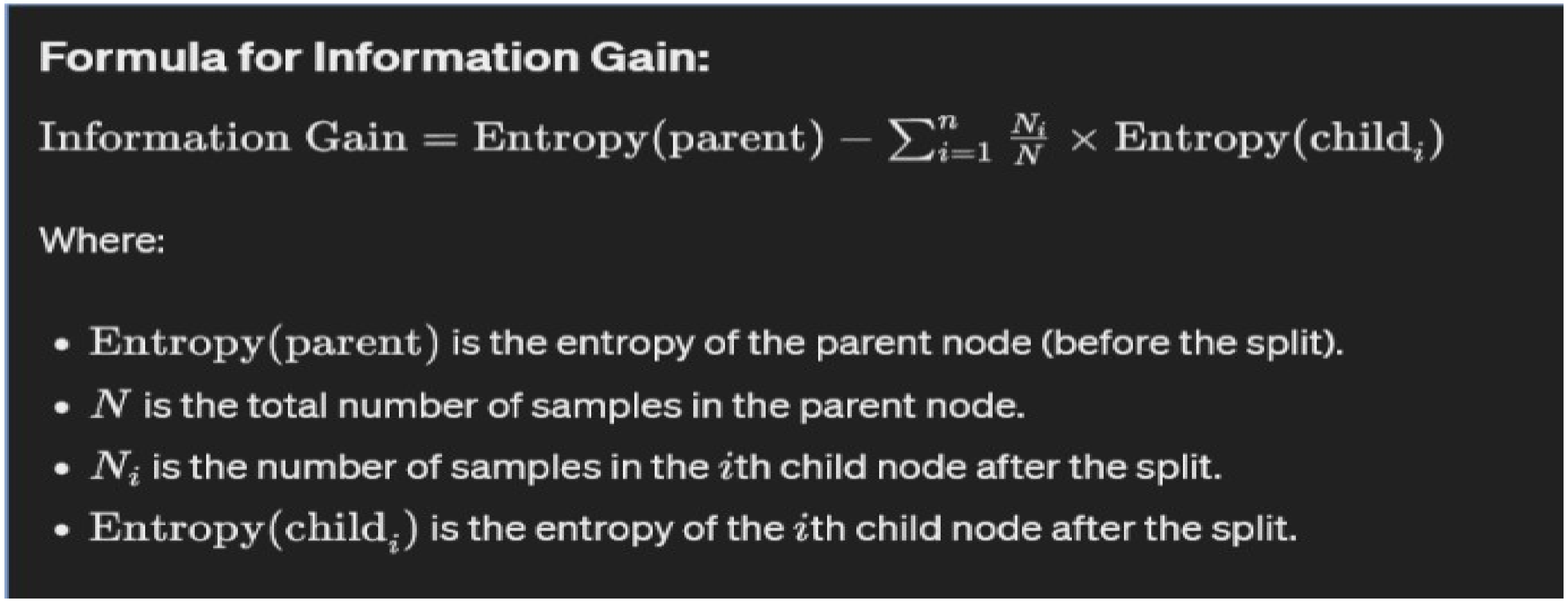


| | | | |
|---|---|---|---|
| Gini Index | Gini Index measures the impurity of a dataset by calculating the probability of incorrectly classifying a randomly chosen sample's label if it were randomly labeled according to the class distribution in the dataset. A lower Gini Index indicates a better feature for splitting. | $\text{Gini Index} = 1 - \sum_{i=1}^{c} p_i^2$ | Suppose we have a dataset with a target variable (Play Golf: Yes/No) and we want to predict whether to play golf based on weather conditions. We calculate Gini Index for each weather attribute (Outlook, Temperature, Humidity, Windy). |

These attribute selection measures help decision tree algorithms to effectively partition the data by selecting the most informative features for splitting, leading to better predictive performance.

# Information gain example

- A **higher Information Gain** indicates that splitting the data on that feature results in more homogenous subsets with respect to the target variable.



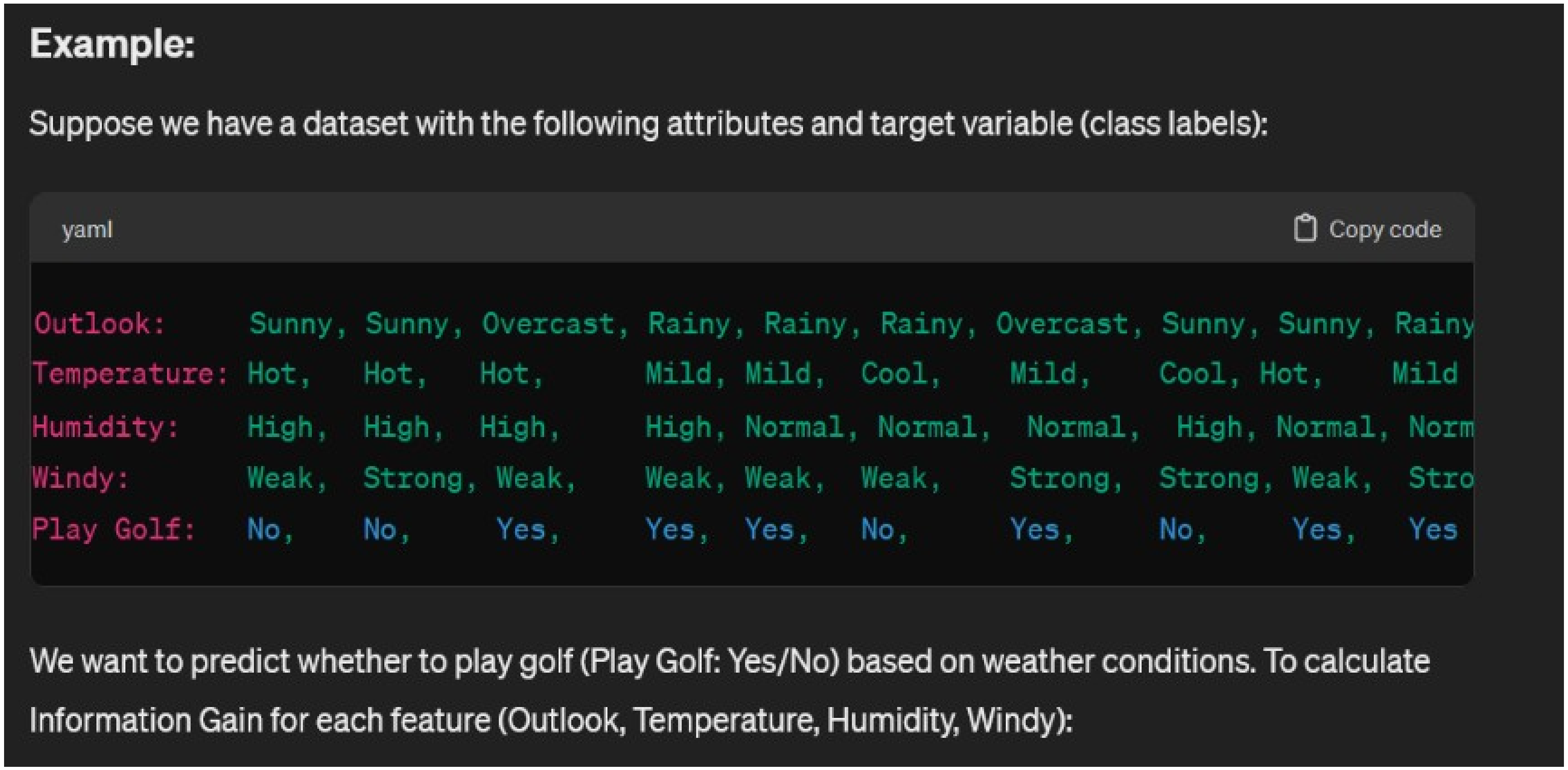
**Formula for Information Gain:**

$$\text{Information Gain} = \text{Entropy}(\text{parent}) - \sum_{i=1}^{n} \frac{N_i}{N} \times \text{Entropy}(\text{child}_i)$$

Where:

- $\text{Entropy}(\text{parent})$ is the entropy of the parent node (before the split).
- $N$ is the total number of samples in the parent node.
- $N_i$ is the number of samples in the $i$th child node after the split.
- $\text{Entropy}(\text{child}_i)$ is the entropy of the $i$th child node after the split.

## Home Exercise



**Example:**

Suppose we have a dataset with the following attributes and target variable (class labels):

yaml Copy code

```yaml
Outlook:     Sunny, Sunny, Overcast, Rainy, Rainy, Rainy, Overcast, Sunny, Sunny, Rainy
Temperature: Hot,   Hot,   Hot,      Mild, Mild,  Cool,   Mild,     Cool, Hot,     Mild
Humidity:    High,  High,  High,     High, Normal, Normal,  Normal,  High, Normal, Norm
Windy:       Weak,  Strong, Weak,    Weak, Weak,  Weak,    Strong,  Strong, Weak,  Stro
Play Golf:   No,    No,     Yes,     Yes,  Yes,   No,      Yes,     No,     Yes,   Yes
```

We want to predict whether to play golf (Play Golf: Yes/No) based on weather conditions. To calculate Information Gain for each feature (Outlook, Temperature, Humidity, Windy):

You can watch this for how to calculate : https://www.youtube.com/watch?v=wefc_36d5mU&ab_channel=MaheshHuddar

## Advantages of the Decision Tree

✓It is simple to **understand** as it follows the same process which a human follow while making any decision in real-life.

✓It can be **very useful for solving decision-related** problems.

✓It helps to think about all the possible outcomes for a problem.

✓There is **less requirement of data cleaning** compared to other algorithms.

## Dis-advantages of the Decision Tree

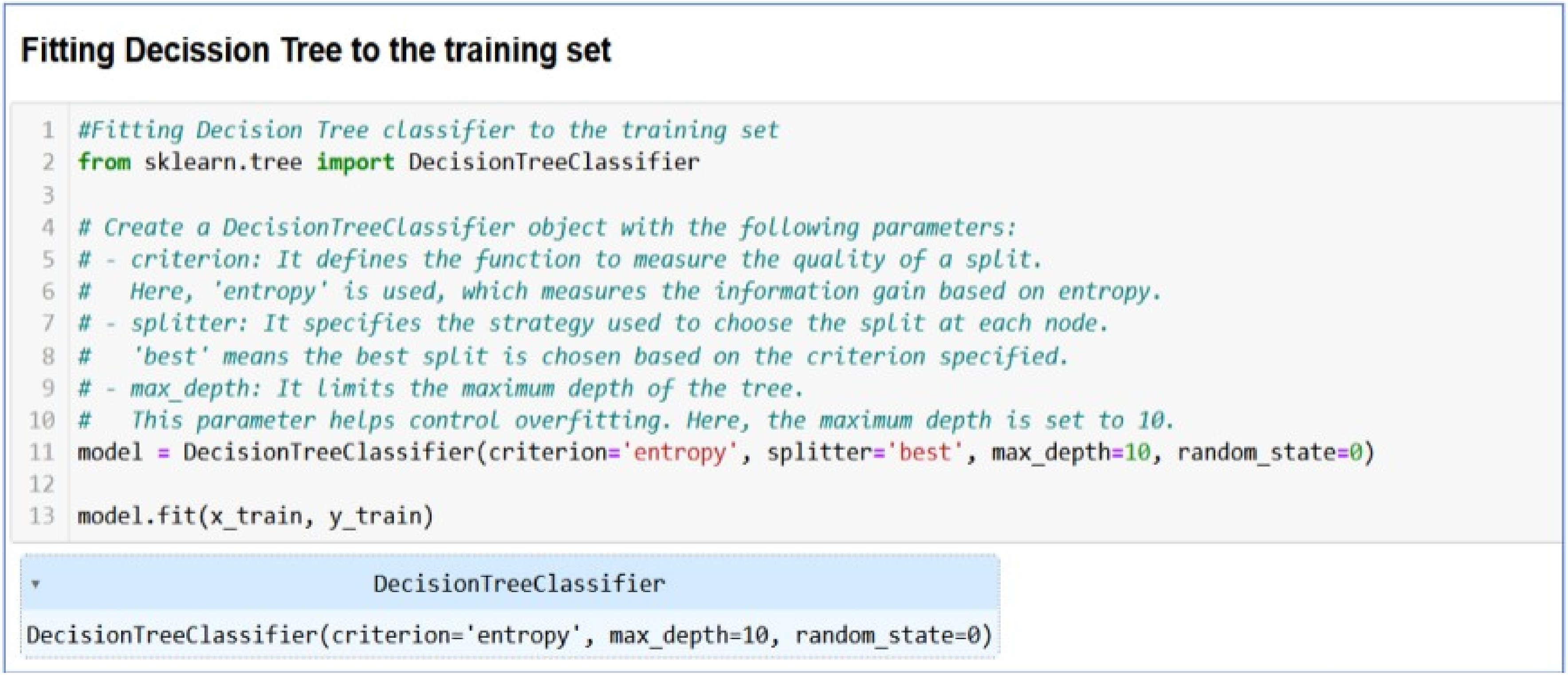
✓The decision tree **contains lots of layers,** which makes it complex.

✓It may have **an overfitting issue**

✓For more class labels, the **computational complexity of the decision tree may** increase.

# Decision Tree-Python Implementation

Assume all the data preprocessing and splitting has been done as we did so far ,

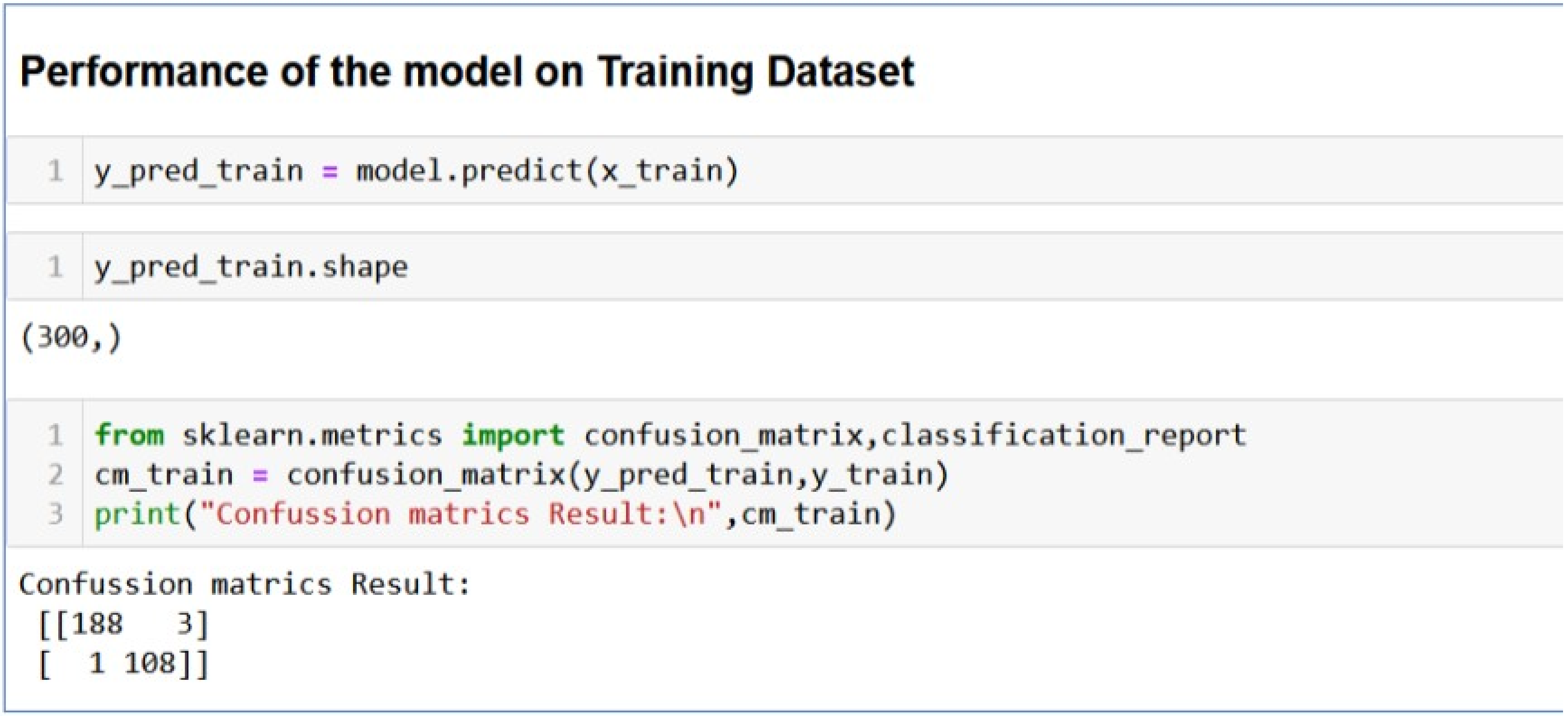


**Fitting Decission Tree to the training set**

```python
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier

# Create a DecisionTreeClassifier object with the following parameters:
# - criterion: It defines the function to measure the quality of a split.
#   Here, 'entropy' is used, which measures the information gain based on entropy.
# - splitter: It specifies the strategy used to choose the split at each node.
#   'best' means the best split is chosen based on the criterion specified.
# - max_depth: It limits the maximum depth of the tree.
#   This parameter helps control overfitting. Here, the maximum depth is set to 10.
model = DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=10, random_state=0)

model.fit(x_train, y_train)
```

```
▾                    DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)
```

# Decision Tree-Python Implementation



## Performance of the model on Training Dataset

```
y_pred_train = model.predict(x_train)
```

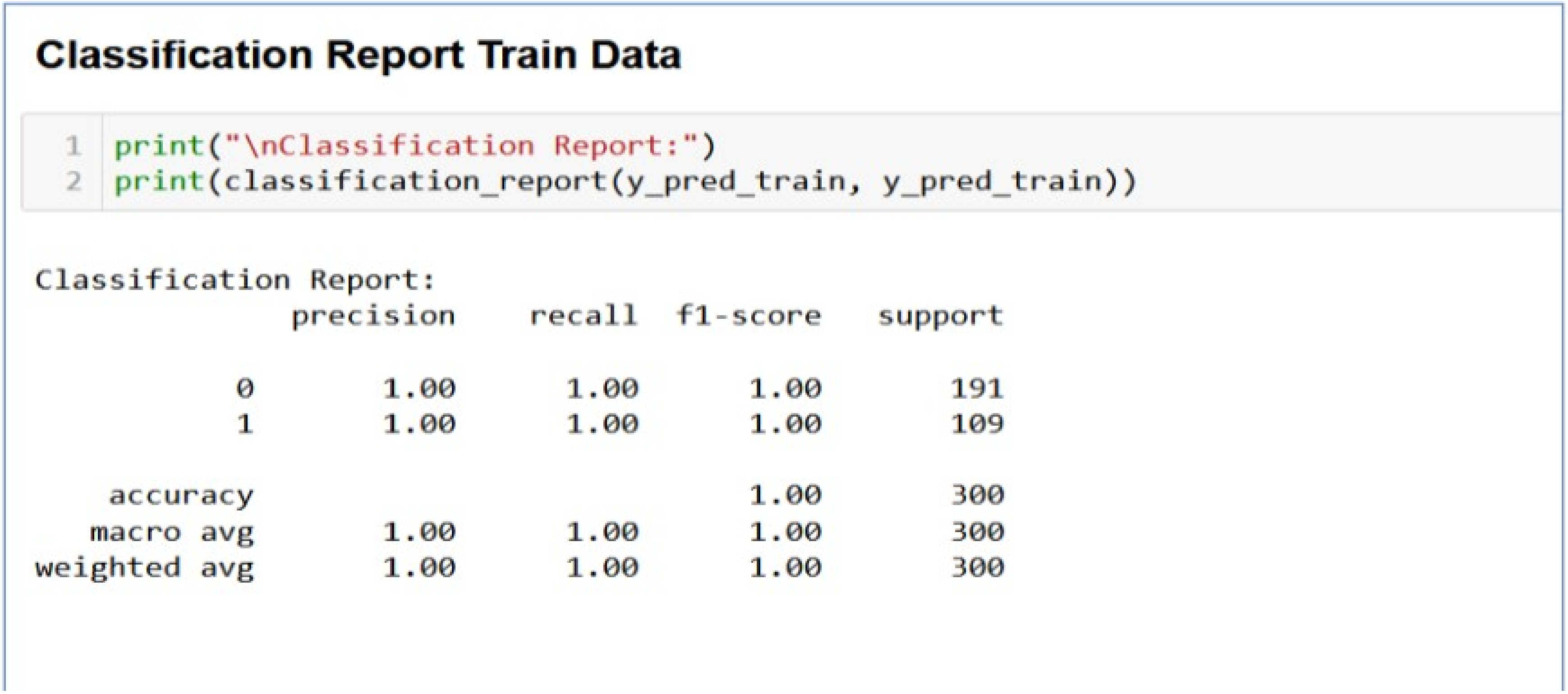
```
y_pred_train.shape
```

```
(300,)
```

```
from sklearn.metrics import confusion_matrix,classification_report
cm_train = confusion_matrix(y_pred_train,y_train)
print("Confussion matrics Result:\n",cm_train)
```

```
Confussion matrics Result:
 [[188   3]
 [  1 108]]
```

## Decision Tree-Python Implementation

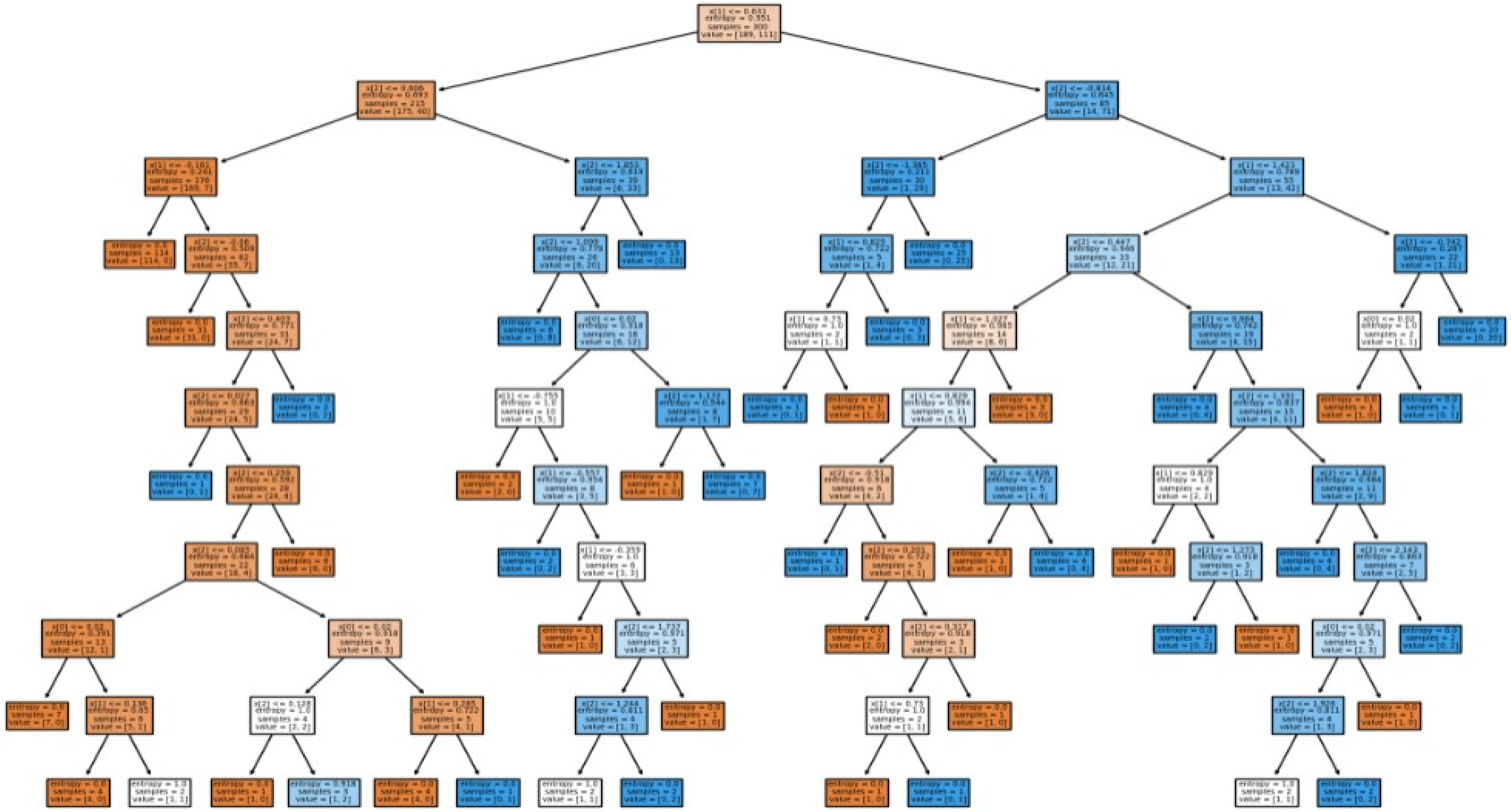


### Classification Report Train Data

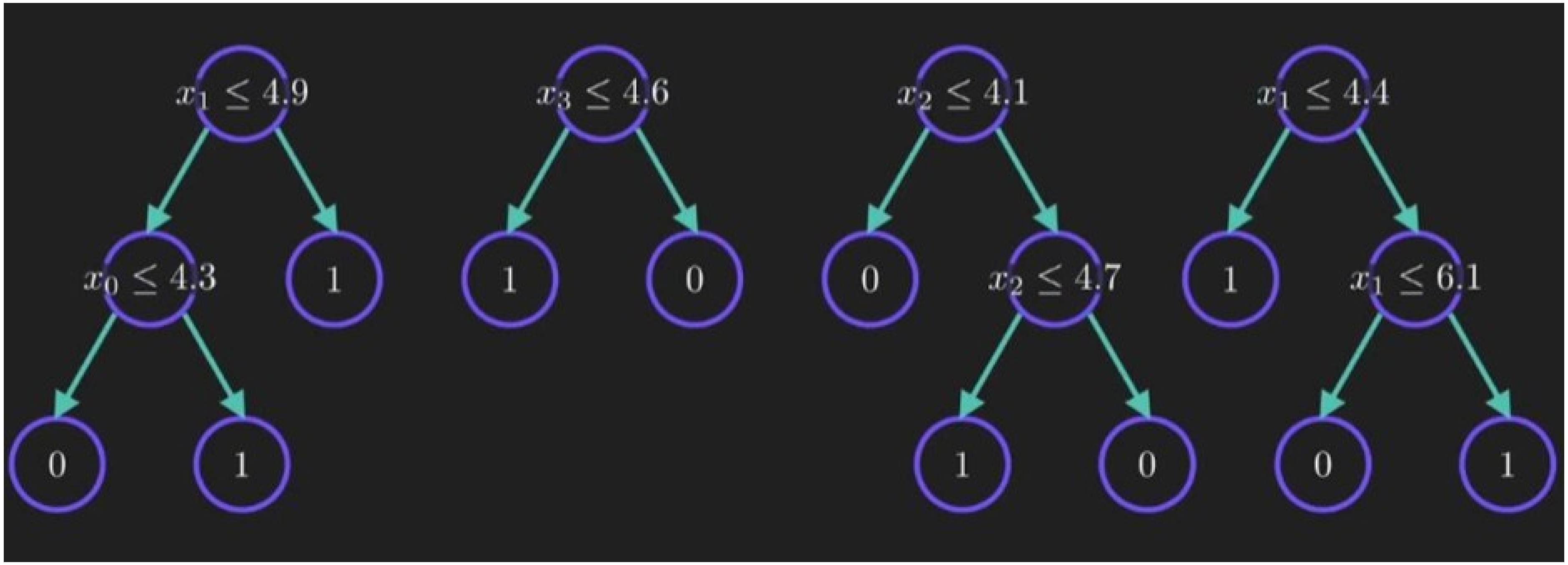
```
print("\nClassification Report:")
print(classification_report(y_pred_train, y_pred_train))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       191
           1       1.00      1.00      1.00       109

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

## Descion Tree Implementation



## Random Forest



## Evaluating a Classification model:

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

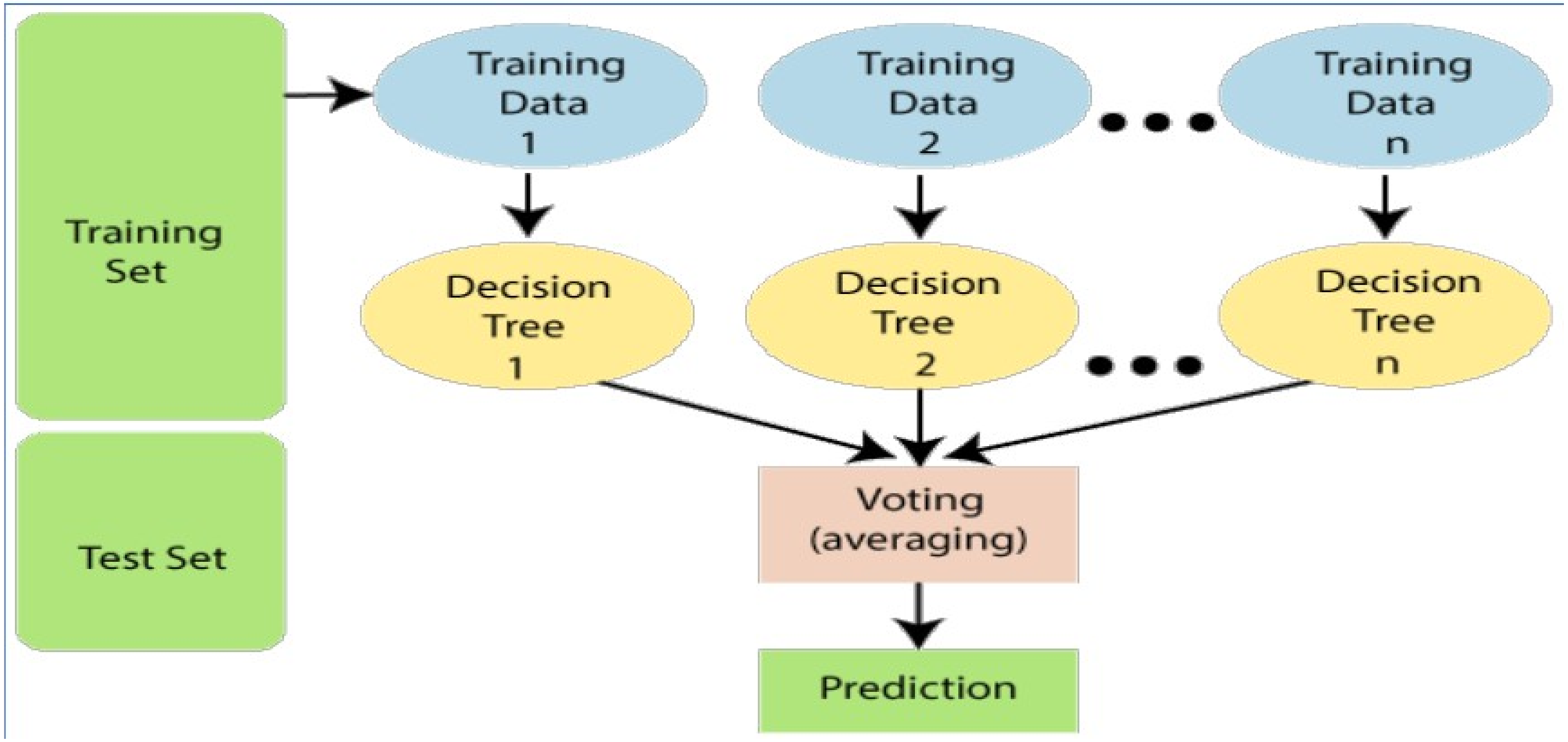
- It can be used for **both Classification and Regression** problems in ML.

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.

- 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."

## Evaluating a Regression models

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



## Why Random Forest

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

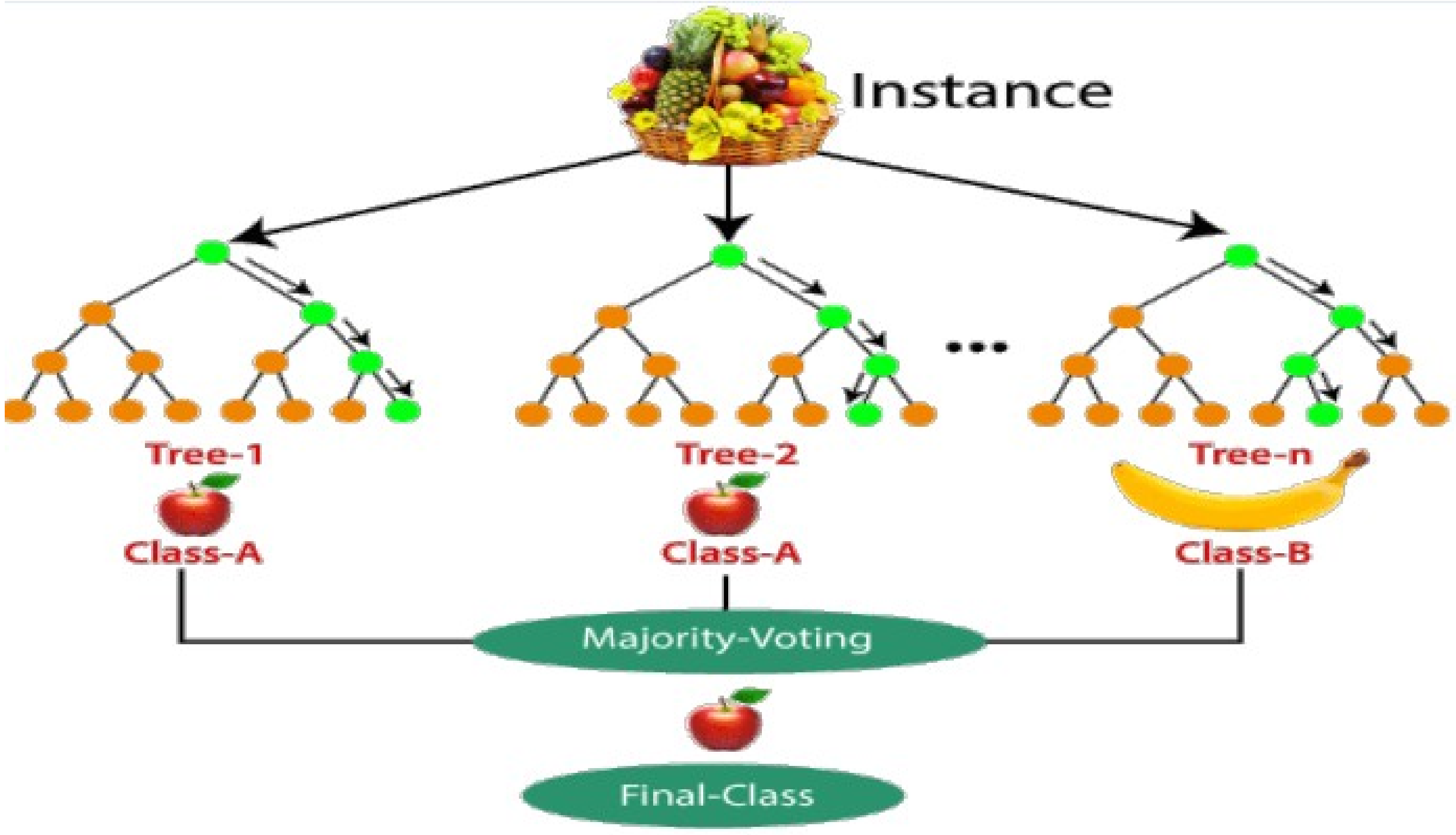
## How Random Forest works

- 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 Random Forest



## Advantage and Dis Advantage

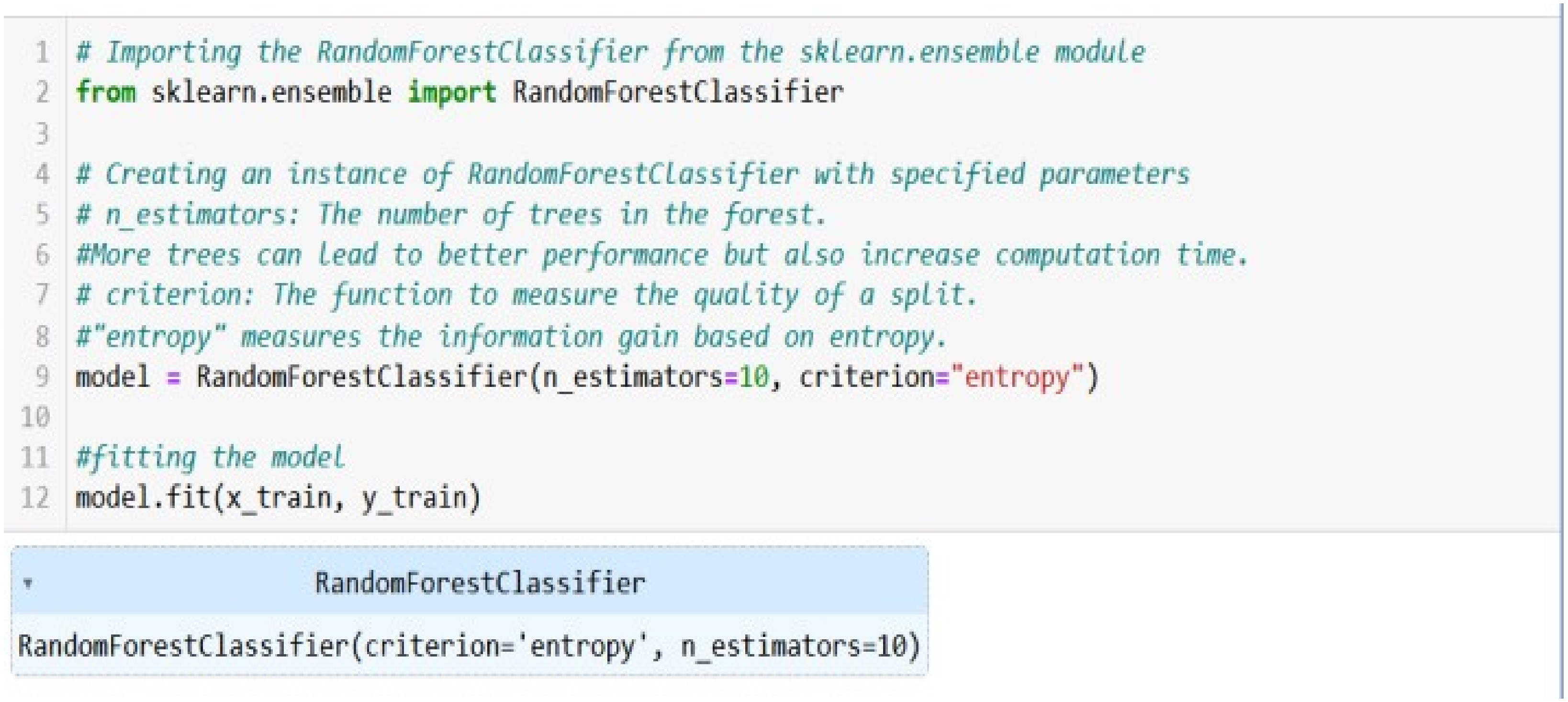
**Advantages of Random Forest**

- Random Forest is capable of **performing both Classification and Regression** tasks.
- It is capable of **handling large datasets with high dimensionality.**
- It enhances the accuracy of the model and **prevents the overfitting issue.**

**Disadvantages of Random Forest**

- Although random forest can be used for both classification and regression tasks, **it is not more suitable for Regression tasks.**

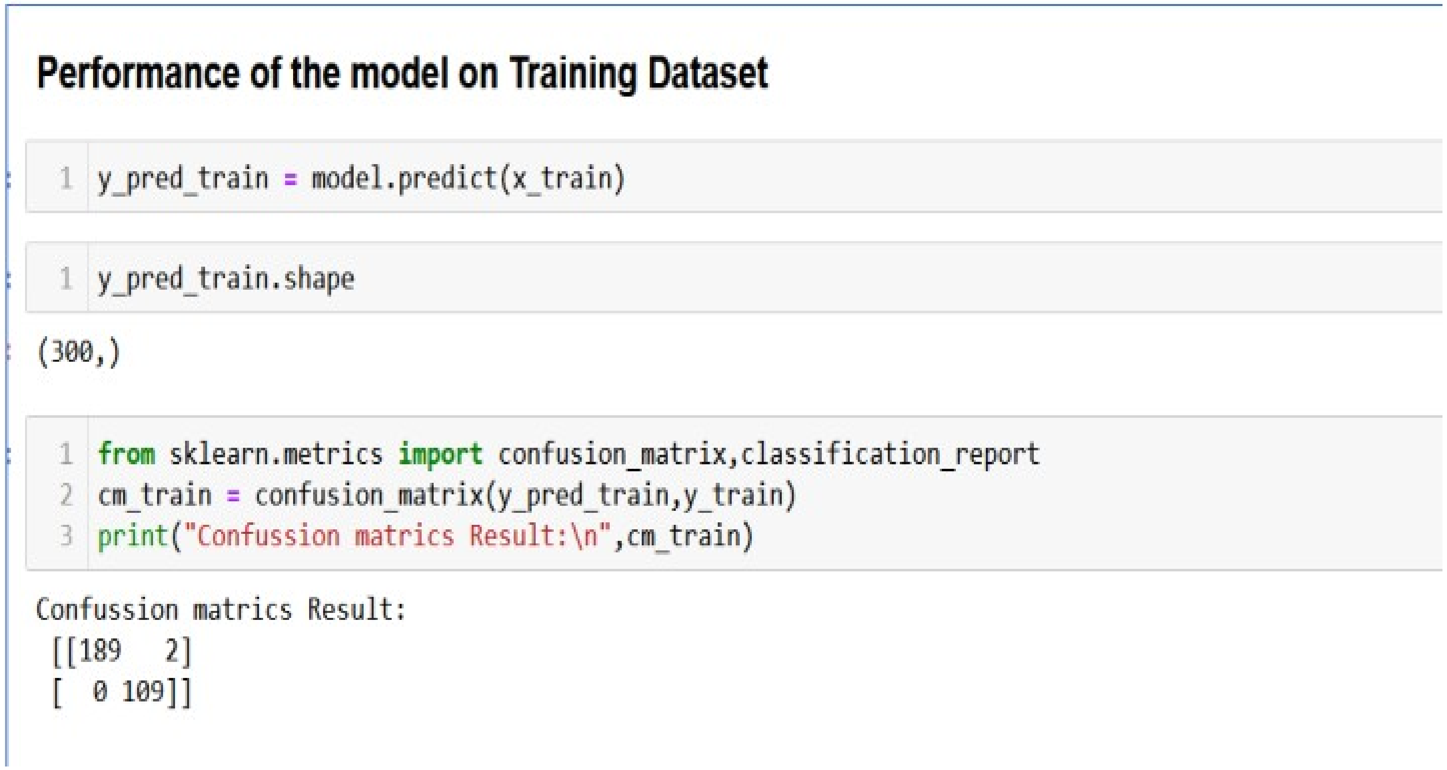
## Implementation of Random Forest Algorithm



```
# Importing the RandomForestClassifier from the sklearn.ensemble module
from sklearn.ensemble import RandomForestClassifier

# Creating an instance of RandomForestClassifier with specified parameters
# n_estimators: The number of trees in the forest.
#More trees can lead to better performance but also increase computation time.
# criterion: The function to measure the quality of a split.
#"entropy" measures the information gain based on entropy.
model = RandomForestClassifier(n_estimators=10, criterion="entropy")

#fitting the model
model.fit(x_train, y_train)
```

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

# Implementation of Random Forest Algorithm



## Performance of the model on Training Dataset

```
y_pred_train = model.predict(x_train)
```

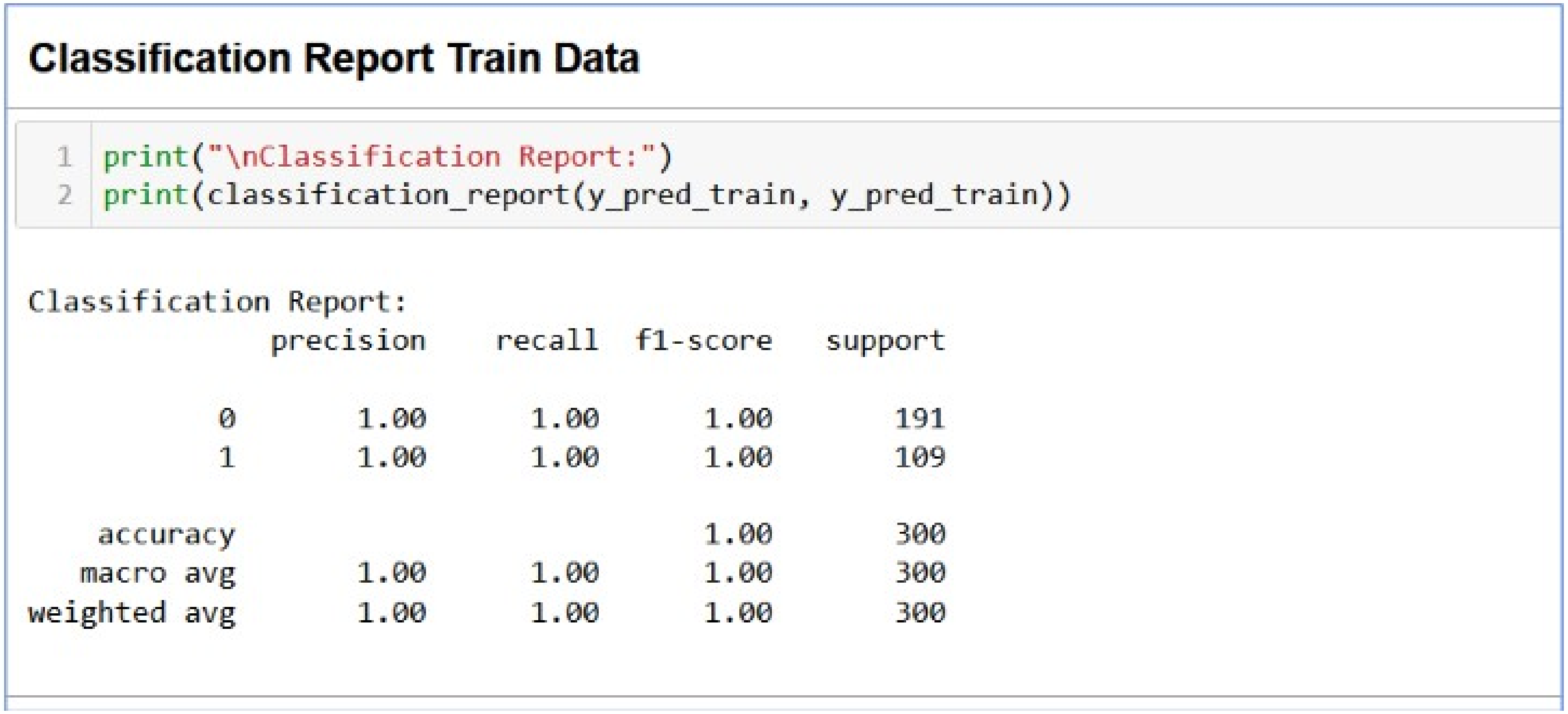
```
y_pred_train.shape
```

```
(300,)
```

```
from sklearn.metrics import confusion_matrix,classification_report
cm_train = confusion_matrix(y_pred_train,y_train)
print("Confussion matrics Result:\n",cm_train)
```

```
Confussion matrics Result:
 [[189   2]
 [  0 109]]
```

## Implementation of Random Forest Algorithm



### Classification Report Train Data

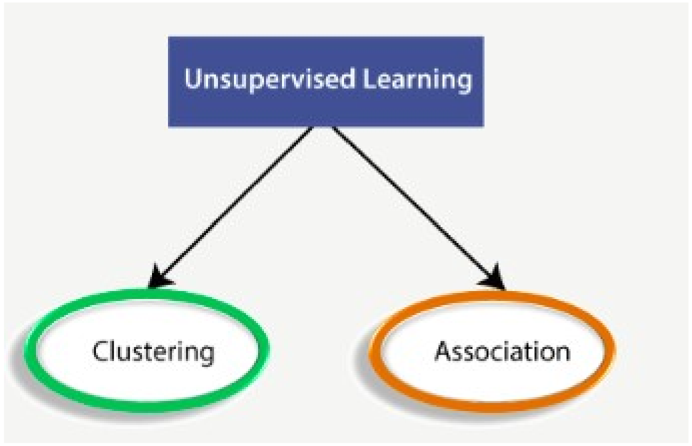
```
print("\nClassification Report:")
print(classification_report(y_pred_train, y_pred_train))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       191
           1       1.00      1.00      1.00       109

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

## Unsupervised Learning

- Unsupervised learning aims to find the **underlying structure or the distribution** of data. We want to explore the data to find some intrinsic structures in them.
- models itself find the **hidden patterns and insights** from the given data.
- Unsupervised learning **cannot be directly applied** to a regression or classification problem
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on **unlabeled and uncategorized** data which make unsupervised learning more important.

## Basic Steps In ML



**Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group.

**Association:** An association rule is an unsupervised learning method which is used for **finding the relationships between variables** in the large database. It determines the set of **items that occurs together** in the dataset.

## Unsupervised Learning algorithms

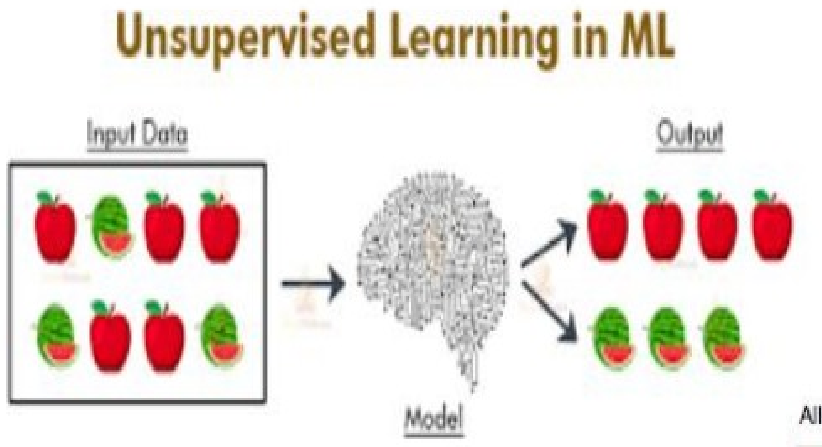
Below is the list of some popular unsupervised learning algorithms:

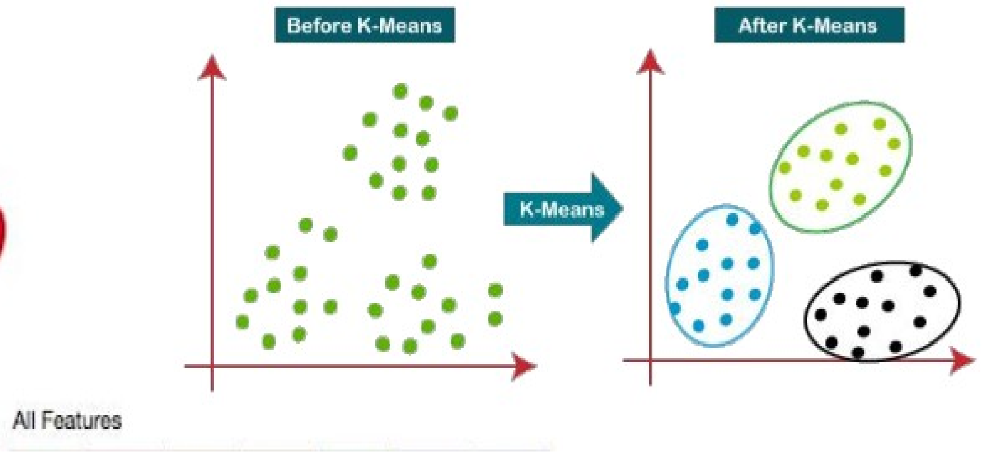
➔ K-means clustering

➔ KNN (k-nearest Neighbors)

➔ Principal component analysis

## K-means clustering





## K-means Algorithm

- **Definition:** K-Means is a **partitioning clustering algorithm** that separates a dataset into K distinct, non-overlapping subsets (clusters).

- **Working Principle**

- **Initialization:** Randomly select K data points as initial cluster centers.

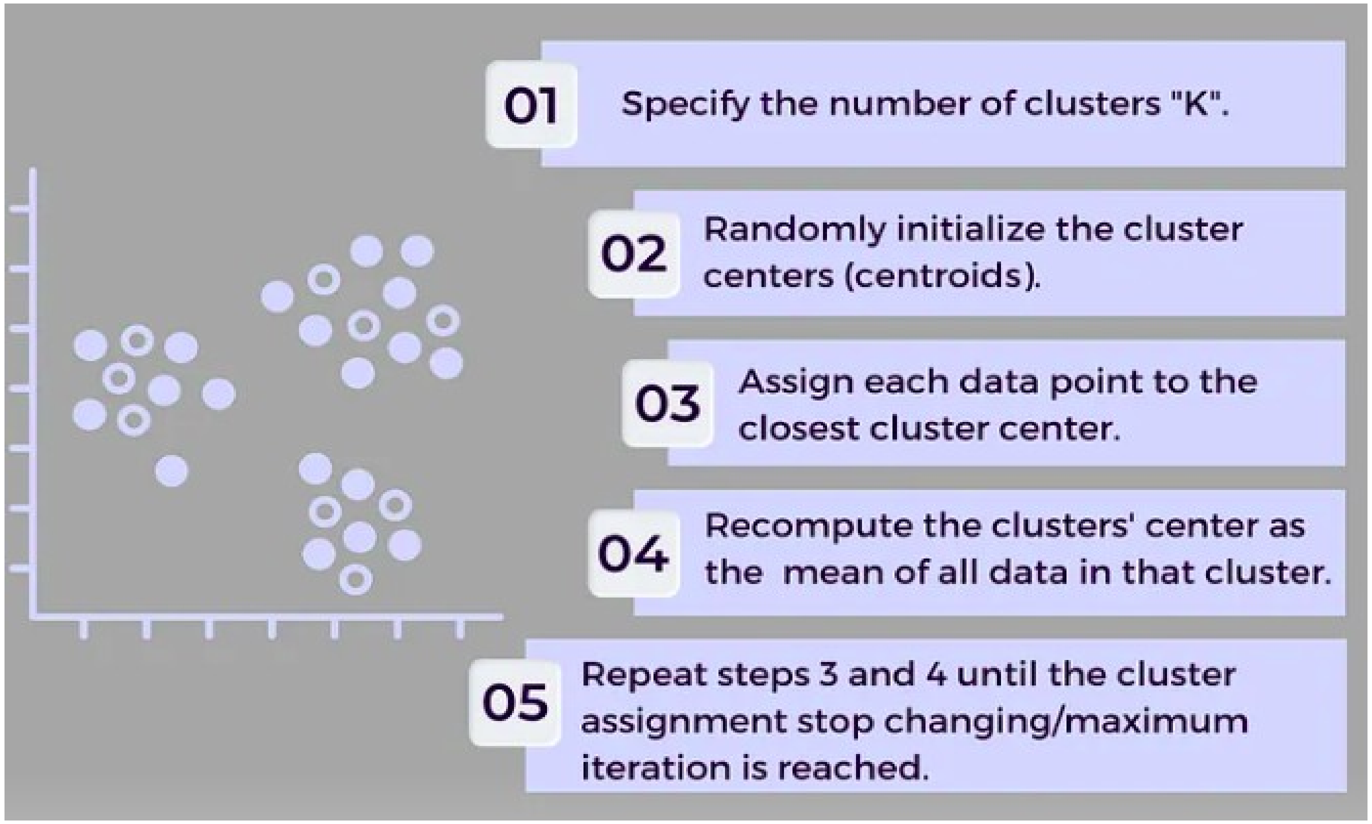
- **Assignment:** Assign each data point to the cluster whose center is nearest.

- **Update Centers:** Recalculate the cluster centers as the mean, variance, Euclidian distance of the data points in each cluster.

- **Repeat:** Iterate steps 2 and 3 until convergence (when cluster assignments stabilize).

## Common Terms

- Given k, the k-means algorithm is **implemented in 5 steps:**



01 Specify the number of clusters "K".

02 Randomly initialize the cluster centers (centroids).

03 Assign each data point to the closest cluster center.

04 Recompute the clusters' center as the mean of all data in that cluster.

05 Repeat steps 3 and 4 until the cluster assignment stop changing/maximum iteration is reached.

https://domino.ai/blog/getting-started-with-k-means-clustering-in-python

## Python Implementation

- **EX: Problem Statement:**

A retail store wants to get **insights about its customers.** And then build a system that can cluster customers into different groups.

- **https://github.com/NelakurthiSudheer/Mall-Customers-Segmentation/blob/main/Python%20Code/Implemenation%20in%20Python.ipynb**