

PYTHON PROGRAMMING & DATA SCIENCE

Patterns and Features



- In statistical machine learning, pattern recognition and data mining, **data** is represented as a *pattern matrix* or *data matrix*.
- The **rows** of the data matrix correspond to patterns in the collection.
- A pattern is also called **point**, **vector**, and **sample** in pattern recognition.
- **columns** of the data matrix correspond to **features**.
- A feature is a **property** or **characteristic** of a pattern.



Patterns and Features

Example:

Pattern number	Weight (in kgs)	Height (in feet)	Class label
1	10	3.5	Chair
2	63	5.4	Human
3	10.4	3.45	Chair
4	10.3	3.3	Chair
5	73.5	5.8	Human
6	81	6.1	Human
7	10.4	3.35	Chair
8	71	6.4	Human

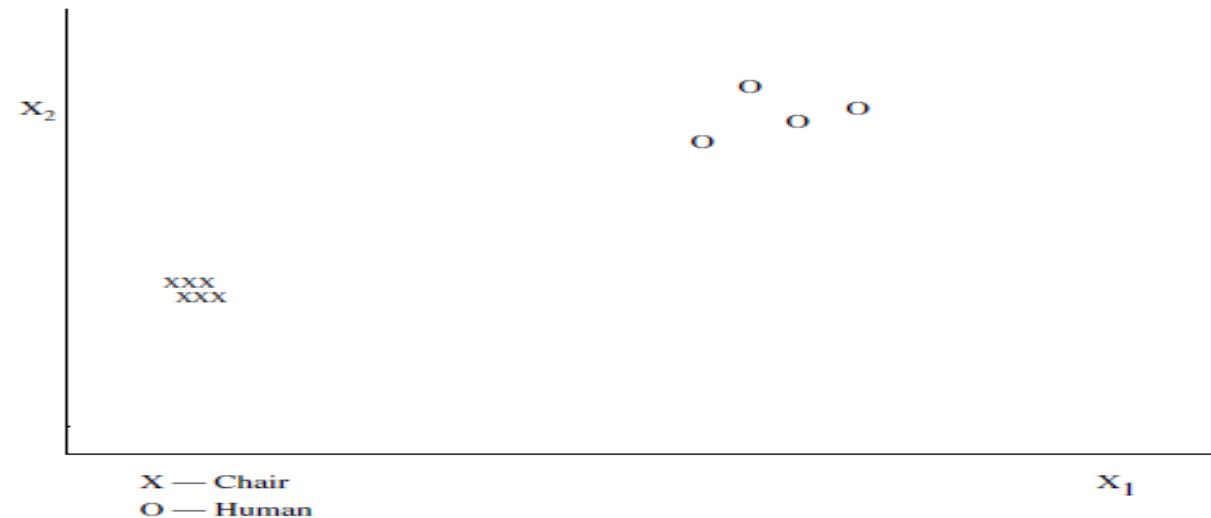
|

- There are **eight patterns** which are represented using **height** in feet and **weight** in Kilograms.
- There are **two classes** labeled **chair** and **human** corresponding to a possible collection of chairs and humans.



Patterns and Features

- Weight and height are the **two different features** characterizing the chairs and humans in the **collection** and class label is the dependent feature that **provides the semantic labels** of the objects considered in table.
- Each pattern is represented as a point in the **two-dimensional space** as follows.



- In the graph where weight is the x_1 feature and height is the second x_2 feature.



Patterns and Features

- A pattern is described by a collection of feature values.
- Each feature is assigned a number or a symbol(Strings) as its value.

Types of Features

- There are different types of features or variables. We may categorize them as follows:
 1. Nominal variable
 2. Ordinal variable
 3. Interval variable
 4. Ratio variable
 5. Temporal data



Patterns and Features

1. Nominal variable:

The simplest variable where the domain has **distinct values**.

2. Ordinal variable:

The domain of this variable is an **ordered set**; so, the values are ordered.

3. Interval variable:

The domain is an ordered set where the **differences between values have a meaningful interpretation**.

4. Ratio variable:

The domain is similar to that of the **interval variable where not only differences, but ratios are also meaningful**.

5. Temporal data :

data which varies with **time and spatial data**.



Patterns and Features

1. Nominal data

➤ A nominal feature fN assumes values from a set, that is the domain of fN denoted by $D(fN)$. So, distinct objects can have different values that are drawn from the set $D(fN)$. Here, different elements of $D(fN)$ are distinct and they are not ordered as $D(fN)$ is a set.

➤ Some examples of nominal features are:

1. *Type of Curve*: A possible domain of this feature is
 {line, parabola, circle, ellipse}.
2. *Type of Publication*: The domain of this variable will include
 technical report, journal paper, conference paper, and book.
3. *TV Manufacturer*: The domain of this attribute could be
 {Sony, Philips, Samsung, LG, Videocon, Onida}.



Patterns and Features

1. Nominal data

- It is possible that a nominal variable is either **binary** or non-binary.
- A **binary feature** has a domain with **two elements**.
- Some examples of **binary nominal variables** are:
 1. *Gender* : domain = {male, female}.
 2. *Beverage available*: domain = {tea, coffee}.
- The most popular application area where nominal data is routinely encountered is **information retrieval**.
- A document is typically viewed as a bag of words; so, it is a **multiset** without any ordering on the values or elements of the set.



Patterns and Features

1. Nominal data

Example 1.

- Let us consider the following two documents.
- D1: The good old teacher teaches several courses
- D2: In the big old college
- We represent the **documents as multisets** given by:
R1 = {The, good, old, teacher, teaches, several, courses} and
R2 = {In, the, big, old, college},
➤ where R1 and R2 are representations of the documents D1 and D2 respectively.



Patterns and Features

1. Nominal data

➤ several simple operations are performed to reduce the total number of terms.

➤ For example,

1. converting uppercase characters to lowercase gives us the following representations:

R1 = {the, good, old, teacher, teaches, several, courses} and

R2 = {in, the, big, old, college}.

2. By stemming, that is by transforming the words to their stemmed forms, we can replace “teacher” and “teaches” by “teach” and “courses” by “course” to get

R1 = {the, good, old, teach, teach, several, course} and

R2 = {in, the, big, old, college}.



Patterns and Features

1. Nominal data

For example,

3. It is possible to represent a collection of documents by the **union of their multisets.**

$R = \{(the, 2), (good, 1), (old, 2), (teach, 2), (several, 1), (course, 1), (in, 2), (big, 1), (college, 1)\}$

- It is possible **to view each document also as a histogram** of term frequency values.
- For example, the histograms corresponding to documents D1 and D2, after case folding and stemming are given by



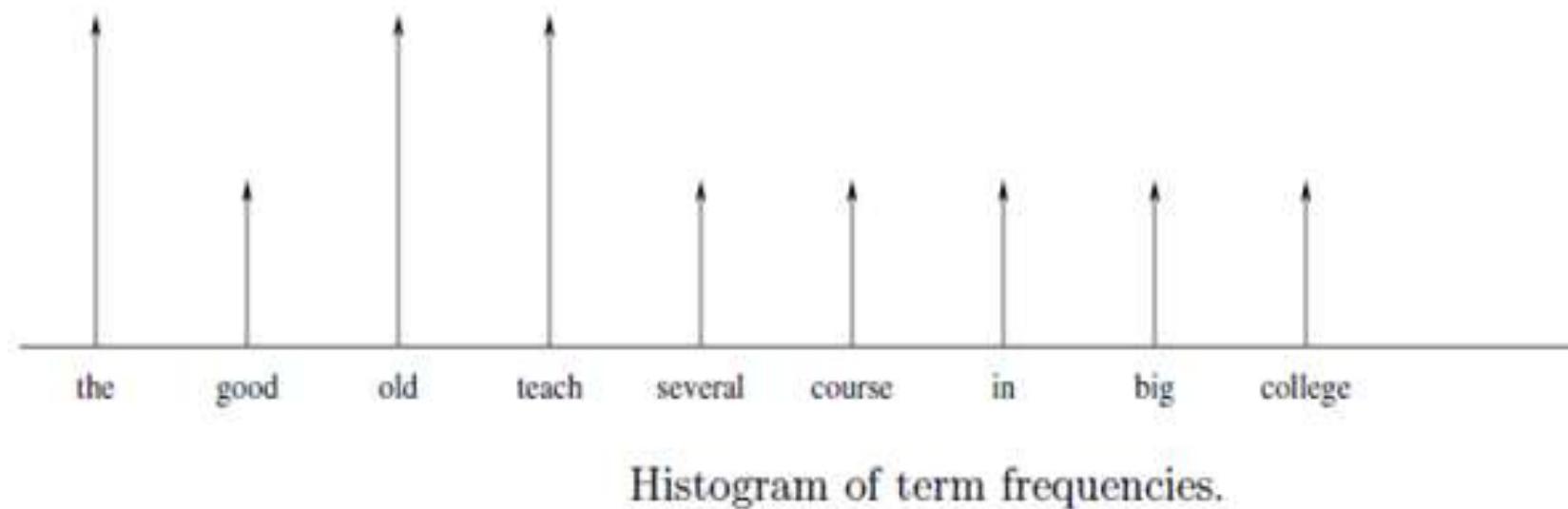
Patterns and Features

1. Nominal data

For example,

$\text{Histogram}(D1): \{(the, 1), (\text{good}, 1), (\text{old}, 1), (\text{teach}, 2), (\text{several}, 1), (\text{course}, 1)\}$.

$\text{Histogram}(D2): \{(\text{in}, 1), (\text{the}, 1), (\text{big}, 1), (\text{old}, 1), (\text{college}, 1)\}$.





Patterns and Features

1. Nominal data

Operations on nominal variables

➤ Following **3 operations** are applied on nominal variables. Those are:

1. comparison,
2. mode and
3. entropy

Example

➤ Consider a dataset of 10 objects which are characterized by only one nominal variable. let the **nominal variable be color**.

➤ The objects and their colors are given in the following set, where obj*i* stands for object *i*.

{(obj1, blue), (obj2, blue), (obj3, red), (obj4, green), (obj5, blue),
(obj6, green), (obj7, blue), (obj8, red), (obj9, blue), (obj10, green)}



Patterns and Features

1. Nominal data

- we can say that the domain of the nominal variable color is
 $D_{color} = \{\text{blue, red, green}\};$

1. comparison:

example:

obj1 and obj2 are identical and are different from obj3.

- In a similar manner we can compare any pair of objects in the set based on the value assumed.
- there are 5 blue, 3 green and 2 red objects in the collection, which means that the set can be represented by a histogram given by $\{(blue, 5), (red, 2), (green, 3)\}$.
- Once we have the histogram, we can obtain the mode and entropy.



Patterns and Features

2. Mode:

Mode is the **most frequent value** of the variable.

- In the example, we can observe from the histogram that blue has the highest frequency and so the mode is blue.

3. Entropy :

It is a **function of the frequencies of values**.

- It characterizes in some sense impurity of the dataset;
- if the variable assumes only one value in the whole dataset, then the dataset is pure and the entropy is zero.



Patterns and Features

- Shannon's entropy is the most popular characterization of entropy. It is given, for the dataset D by

$$\text{Entropy}(D) = - \sum_{i=1}^d p_i \log p_i,$$

- where p_i is the probability of value i and d is the size of the domain of the variable under consideration.
- the values of probabilities obtained based on their frequencies of occurrence are:

$$P(\text{blue}) = \frac{5}{10} = 0.5; \quad P(\text{green}) = \frac{3}{10} = 0.3; \quad P(\text{red}) = \frac{2}{10} = 0.2.$$

- For these probability values, the entropy is 0.4472.



Patterns and Features

2. Ordinal data

- In the case of ordinal data, the **elements of the domain of the variable are ordered, in addition to being distinct.**
- Some examples of ordinal features are:
 1. Height of an object:
domain = {very tall, tall, medium, short, very short}.
 2. Ranking of documents.
quality of a document based on a scale from 1–9; reviewers are asked to rank a paper submitted for possible publication.
 3. Sentiment mined from a collection of documents (perhaps tweets) on a product:
domain = {very negative, negative, neutral, positive, very positive}.



Patterns and Features

2. Ordinal data

Operations possible on ordinal variables

- As an ordinal variable has domain whose elements are distinct, all the operations on nominal variables are possible on ordinal variables also. So, **comparison, mode, and entropy are possible.**
- In addition, ordering among the values permits operations **like median and percentile.**
- Median is the most **centrally located value in the domain.**

For example,

medium value of variable Height; neutral for variable sentiment; and value 5 for the variable ranking based on a scale from 1 to 9 may be viewed as the median values in each case.



Patterns and Features

2. Ordinal data

Operations possible on ordinal variables

- Percentile makes sense when the values of the variable are ordered. Top ten percentile indicates the value below which 90% of the values are located.
- It is possible to convert a **nominal variable** into an **ordinal variable** by imposing some meaningful ordering.

Binary variables

- In this method variable values are represented using **binary digits**. i.e 0 or 1.
For example,
consider D1 and D2.

There are nine distinct terms in these two documents. They are: big, college, course, good, in, old,several, teach, the in the lexicographically sorted order.



Patterns and Features

Using these nine terms, the two documents can be represented as binary strings as shown below

Representation of two documents in a binary form.

	Big	College	Course	Good	In	Old	Several	Teach	The
D_1	0	0	1	1	0	1	1	1	1
D_2	1	1	0	0	1	1	0	0	1

3. Interval-valued variables

- For interval-valued variables, the **differences between values are meaningful**.
- An example is: Temperature in Celcius and Fahrenheit:
 10°C is five degrees more than 5°C



Patterns and Features

Interval-valued variables

Operations possible on interval-valued variables

➤ Mean and Standard Deviation are two possible operations on these variables.

$$\text{Mean} = \frac{1}{n} \sum_{i=1}^n x_i,$$

$$\text{Standard Deviation} = \frac{1}{n-1} \left(\sum_{i=1}^n (x_i - \text{Mean})^2 \right)^{\frac{1}{2}}.$$

Example :

consider temperature over 5 consecutive days in summer in Bengaluru to be 35°C, 36°C, 36°C, 37°C, 36°C.

Then the mean value is 36°C and the variance is 0.4.

➤ Another example of the interval-valued type is calendar dates.



Patterns and Features

4. Ratio variables

- A ratio variable, in addition to properties like distinctness, order, addition and subtraction, permits **usage of multiplication and division**.
- Ratio variables are the most popular in Data Mining, Pattern Recognition, and Machine Learning.
- Examples of ratio variables include weight and height of physical objects

5. Spatio-temporal data

- There are several applications where the data is not static or fixed; it is dynamic.

Dynamic data is routinely understood as **time varying**.



Patterns and Features

➤ In dynamic datasets, we have the data to be one of the following.

1. Spatial Data:
2. Temporal Data:
3. Spatio-temporal Data:

Spatial Data:

In some applications, learning the predictive models is influenced by the **spatial information of the data**.

Example :

➤ in predicting earthquakes, it is possible that for all other conditions being equal, an area in some geographical location has a higher probability of being earthquake prone whereas some other area in a different geographical location may have a lower probability.



Patterns and Features

Temporal Data:

- Data that **varies with time** is called temporal data.
- Time series data is popular and here the successive time intervals are equally spaced or regular.

Examples:

1. Speech signal
2. web clicks;

Spatio-temporal Data:

- In some applications, **data varies both with space and time**.
- For example a **search engine** provides ranked results to a query. These results might change with time and also based on geographical location from which the user queried the search engine.



Patterns

Pattern:

A pattern is some phenomenon that repeats regularly based on a set rule or condition.

(or)

A pattern represents a physical object or an abstract notion.

- In generally , the pattern may represent physical objects like balls, animals or furniture.
- Abstract notions could be like whether a person will play tennis or not(depending on features like weather etc.)
- It gives the description of the object or the notion.
- The description is given in the form of attributes of the object.
- These are also called the features of the object.



Patterns

Example:

Imagine that we want **to draw a series of cats**.

- All cats share common characteristics such as
 1. Have eyes, tails and fur,
 2. eating fish and make meowing sounds.
- we can make a good attempt at drawing a cat, simply by including these common characteristics.
- In computational thinking, these characteristics are known as patterns.
- Once we know how to describe one cat we can describe others, simply by following this pattern.

The only things that are different are the specifics:

one cat may have green **eyes**, a long **tail** and black **fur**

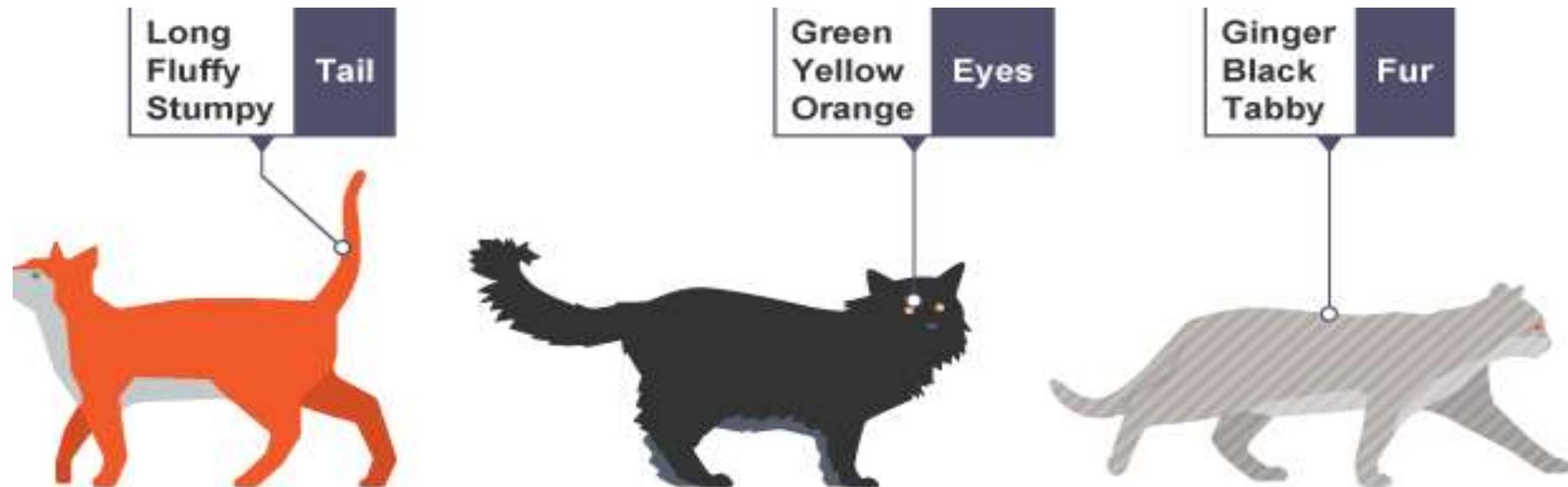
another cat may have yellow **eyes**, a short **tail** and striped **fur**



Patterns

Example:

Imagine that we want to draw a series of cats.





Patterns

Other examples are:

The colours on the clothes, speech pattern , ...etc.

Pattern Recognition :

Pattern recognition is the process of recognizing patterns by using machine learning algorithm.

- Pattern recognition can be defined as **the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation.**
- One of the important aspects of the pattern recognition is its application potential.
- Pattern recognition is the scientific discipline that allows us to classify objects into several categories or classes that can be further used to perform analysis and improve certain things.



Patterns

➤ The three best-known approaches for pattern recognition are:

- 1) *Template matching*
- 2) *Statistical classification*
- 3) *Syntactic or structural matching*

1) *Template matching-*

- Template Matching is used to determine the similarity between two entities (points, curves, or shapes) of the same type.
- The pattern to be recognized is matched with a stored template along with geometrical transformations.
- This approach has some obvious disadvantages of being too rigid and having the need for lots of templates.



Patterns

2) *Statistical classification*–

- In this method, each pattern is represented in terms of some features or measurements.
- The main objective of this approach is to establish decision boundaries in the feature space. This separates patterns belonging to different classes creating some rules for an inter-class boundary.

3) *Syntactic or structural matching*–

- This method works on a hierarchy framework where a pattern is said to be composed of simple sub-patterns that are themselves built from yet simpler sub-patterns.



Patterns

STEPS INVOLVED IN PATTERN RECOGNITION

- The major steps involved in a typical pattern recognition process are-
1. Collection of relevant data from various sources
 2. Pre-processing of data – It involves removing noise from data and making data in a format suitable for applying algorithms.
 3. Examining data and Dividing into classes
 4. Analyzing of various classes and its boundaries
 5. Applying these analyses according to the needs.



Patterns

- In a typical pattern recognition application, the raw data is processed and converted into a form that is amenable for a machine to use.
- Pattern recognition involves classification and cluster of patterns.
- In classification, an appropriate class label is assigned to a pattern based on an abstraction that is generated using a set of training patterns or domain knowledge.
- Classification is used in supervised learning.
- Clustering generated a partition of the data which helps decision making, the specific decision making activity of interest to us.
- Clustering is used in an unsupervised learning.



Patterns

➤ **Features** may be represented as **continuous, discrete or discrete binary variables**. A feature is a function of one or more measurements, computed so that it quantifies some significant characteristics of the object.

Example:

➤ consider our face then eyes, ears, nose etc are features of the face.

Pattern recognition possesses the following features:

1. Pattern recognition system should recognise familiar pattern quickly and accurate
2. Recognize and classify unfamiliar objects
3. Accurately recognize shapes and objects from different angles
4. Identify patterns and objects even when partly hidden
5. Recognise patterns quickly with ease, and with automaticity.



Patterns

Patterns in Machine Learning

- Machine learning uses mathematics, statistics, and domain-specific knowledge and data to solve complex problems.

Machine Learning:

Machine learning is turning things(data) into numbers and **finding patterns** in those numbers.

- For finding patterns, algorithms are used. An algorithm is a specific set of steps to perform a task.

An “*algorithm*” in machine learning is a procedure that is run on data to create a machine learning “*model*.”

- A machine learning *algorithm* is written to derive the *model*.
- The *model* identifies the *patterns* in data that **fit** the *dataset*. **Fit** is a synonym to “**find patterns in data**”.



Patterns

Patterns in Machine Learning

- A “model” in machine learning is **the output of a machine learning algorithm run on data.**
- A model represents what was **learned** by a machine learning algorithm. It is basically a mathematical function that can adapt to new data by tweaking its parameters.





Patterns

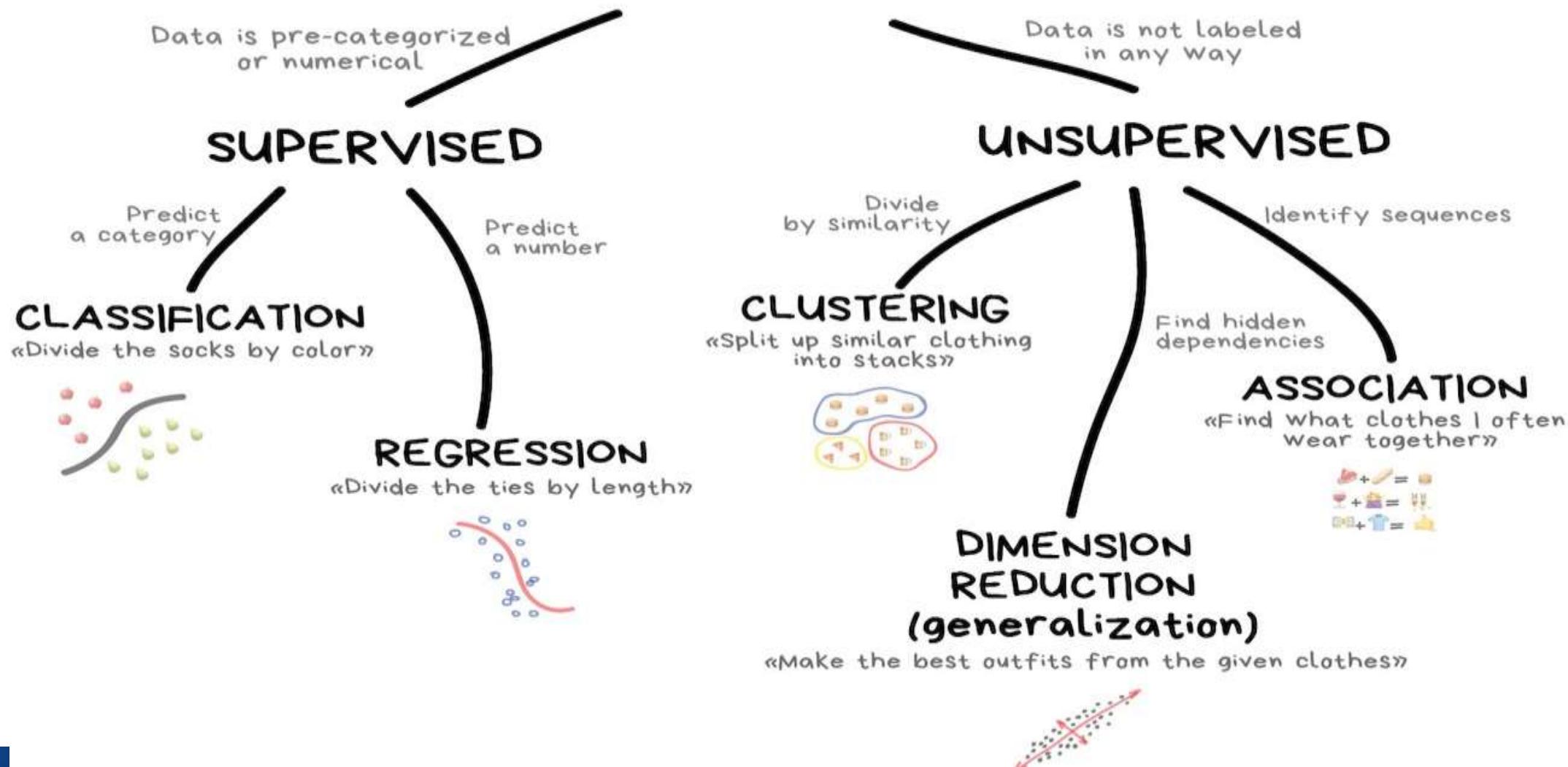
Patterns in Machine Learning

- **Models** are like the general equation **of** a line $y = a + bx$, while **patterns** are like a specific equation, e.g. $y = 5 + 2x$.
- Machine learning is about generalizing correctly to brand-new situations.
- The basic task of machine learning is to create a model that can predict or classify different patterns from data.
- One of the applications of this is the classification of spam or non-spam data.
- The algorithms adaptively improve their performance as the number of samples available for learning increases.
- Two main types of machine learning are supervised and unsupervised machine learning.



Patterns

CLASSICAL MACHINE LEARNING





Patterns

Patterns in Machine Learning

- Pattern recognition in **a supervised approach** is called **classification**.
- These algorithms use a **two-stage methodology** for identifying the **patterns**.
- The first stage is the development/construction of the model and the second stage involves the prediction for new or unseen objects.
- **Unsupervised learning** is a type of machine learning that **looks for previously undetected patterns** in a data set with no pre-existing labels and with minimum human supervision.
- Classification **is** supervised learning, while **clustering** **is** unsupervised learning.



Patterns

Representation of patterns

➤ Patterns can be represented mainly in **4 different ways**. Those are:

1. Representing patterns as vectors
2. Representing patterns as strings
3. Representing patterns by using logical operators
4. Representing patterns using fuzzy and rough sets



Patterns

1. Representing patterns as vectors

- The most popular method of representing patterns is as **vectors**.
- Here, the training dataset may be **represented as a matrix of size (nxd)**.
- where each **row corresponds to a pattern and each column represents a feature**.
- Each attribute/feature/variable is associated with a **domain**.
- A domain is a set of numbers, each number pertains to a value of an **attribute for that particular pattern**.
- The class label is a dependent attribute which depends on the ‘d’ independent attributes.



Patterns

Example:

The dataset could be as follows :

	f_1	f_2	f_3	f_4	f_5	f_6	Class label
Pattern 1:	1	4	3	6	4	7	1
Pattern 2:	4	7	5	7	4	2	2
Pattern 3:	6	9	7	5	3	1	3
Pattern 4:	7	4	6	2	8	6	1
Pattern 5:	4	7	5	8	2	6	2
Pattern 6:	5	3	7	9	5	3	3
Pattern 7:	8	1	9	4	2	8	3

- In this case, $n=7$ and $d=6$.
- Each pattern has six attributes(or features).
- Each attribute in this case is a number between 1 and 9.
- The last number in each line gives the class of the pattern.
- In this case, the class of the patterns is either 1, 2 or 3.



Patterns

- If the patterns are **two- or three-dimensional**, they can be **plotted**.

Example:

- Consider the dataset

Pattern 1 : (1,1.25,1)

Pattern 3 : (1.5,0.75,1)

Pattern 5 : (1,3,2)

Pattern 7 : (1.5,3.5,2)

Pattern 9 : (4,2,3)

Pattern 11 : (5,1,3)

Pattern 2 : (1,1,1)

Pattern 4 : (2,1,1)

Pattern 6 : (1,4,2)

Pattern 8 : (2,3,2)

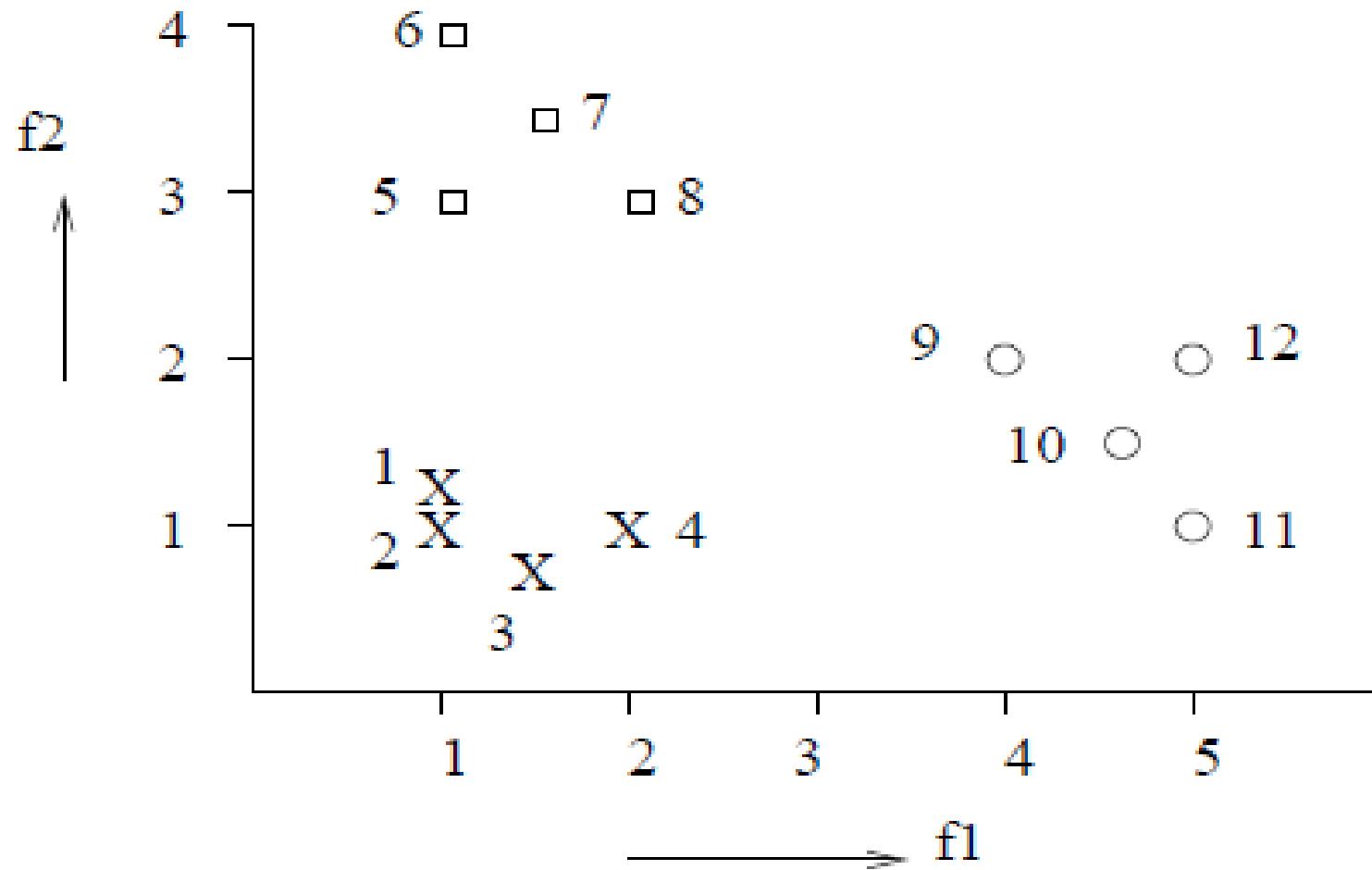
Pattern 10 : (4.5,1.5,3)

Pattern 12 : (5,2,3)

- Each **triplet** consists of feature 1, feature 2 and the class label. This is shown in Figure as follows



Patterns





Patterns

2. Representing patterns as strings

- Here each pattern is a string of characters from an alphabet.
- This is generally used to represent gene expressions.
- For example, DNA can be represented as
GTGCATCTGACTCCT...
- RNA is expressed as
GUGCAUCUGACUCCU....

- This can be translated into protein which would be of the form
VHLTPEEK
- Each string of characters represents a pattern.
- Operations like pattern matching or finding the similarity between strings are carried out with these patterns.



Patterns

3. Representing patterns by using logical operators

➤ Here each pattern is represented by a sentence(well formed formula) in a logic.

Example1:

if (beak(x) = red) and (colour(x) = green) then parrot(x)

➤ This is a rule where the antecedent is a conjunction of primitives and the consequent is the class label.

Example2:

if (has-trunk(x)) and (colour(x) = black) and (size(x) = large) then elephant(x)



Patterns

4. Representing patterns using fuzzy and rough sets

- The features in a fuzzy pattern may consist of linguistic values, fuzzy numbers and intervals.
- For example, linguistic values can be like tall, medium, short for height which is very subjective and can be modelled by fuzzy membership values.
- A feature in the pattern maybe represented by an interval instead of a single number. This would give a range in which that feature falls.

Example:

(3, small, 6.5, [1, 10])

- The above example gives a pattern with 4 features.
- The first feature is an integer and second feature as a linguistic value.
- The third feature is a real value.
- The 4th feature is in the form of an interval. In this case the feature falls within the range 1 to 10. This is also used when there are missing values.



Patterns

4. Representing patterns using fuzzy and rough sets

- Rough sets are used to represent classes.
- A class description will consist of an upper approximate set and a lower approximate set.
- An element 'y' belongs to the lower approximation if the equivalence class to which 'y' belongs is included in the set.
- On the other hand 'y' belongs to the upper approximation of the set if its equivalence class has a non- empty intersection with the set.
- The lower approximation consists of objects which are members of the set with full certainty.
- The upper approximation consists of objects which may possibly belong to the set.



Patterns

4. Representing patterns using fuzzy and rough sets

- For example, consider the Figure 3.

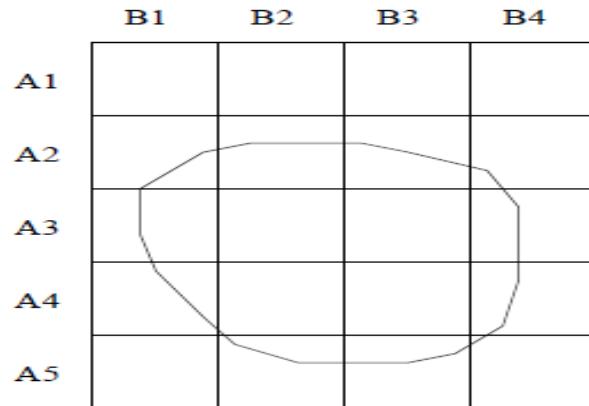


Figure 3: Representation of an object

- This represents an object whose location can be found by the grid shown.
- The object shown completely covers (A3,B2), (A3,B3), (A4,B2) and (A4,B3).
- The object falls partially in (A2,B1),(A2,B2),(A2,B3),(A2,B4),(A3,B1),(A3,B4),(A4,B1),(A4,B4),(A5,B2), and (A5,B3).



Patterns

4. Representing patterns using fuzzy and rough sets

- The pattern can be represented as a rough set where the **first four values** of the grid gives the **lower approximation** and the **rest of the values** of the grid listed above form the upper approximation.

- The classes can also be fuzzy. One example of this would be to have linguistic values for classes. The classes for a set of patterns can be small and big. These classes are fuzzy in nature as the perception of small and big is different for different people.

The curse of dimensionality



The curse of dimensionality

- The Curse of Dimensionality is termed by mathematician **R. Bellman** in his book “**Dynamic Programming**” in 1957.
- According to him, the curse of dimensionality is the **problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space**.
- The curse of dimensionality basically means that the **error increases with the increase in the number of features**. It refers to the fact that algorithms are harder to design in high dimensions and often have a running time exponential in the dimensions.
- A higher number of dimensions theoretically **allow more information to be stored**, but **practically it rarely helps** due to the **higher possibility of noise and redundancy** in the real-world data.

The curse of dimensionality



The curse of dimensionality

- Curse of Dimensionality refers to a **set of problems** that arise when working with **high-dimensional data**.
- The **dimension** of a dataset corresponds to the number of attributes or features that exist in a dataset.
- A dataset with a **large number of attributes**. generally , the order of a **hundred or more**, is referred to as **high dimensional data**.
- Some of the **difficulties** that come with **high dimensional data** manifest during analyzing or visualizing the data to identify patterns, and some manifest while training machine learning models.
- The **difficulties** related to **training machine learning models** due to **high dimensional data** is referred to as '**Curse of Dimensionality**'.

The curse of dimensionality



The curse of dimensionality

- The **2** popular aspects of the curse of dimensionality are:
 1. data sparsity and
 2. distance concentration .

Data Sparsity

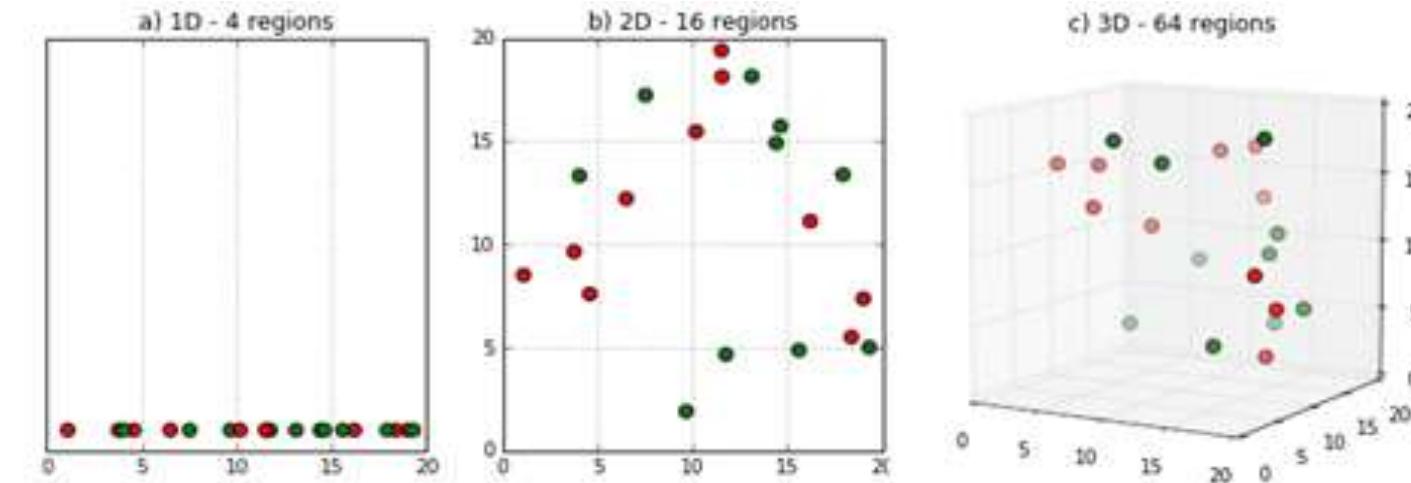
- Sparsity of data occurs when moving to higher dimensions.
- The **volume of the space represented grows so quickly that the data cannot keep up and thus becomes sparse.**
- The sparsity issue is a major one for anyone whose goal has some statistical significance.

The curse of dimensionality



The curse of dimensionality

Example 1:



- As the **data space** seen above moves from one dimension to two dimensions and finally to three dimensions, the given data fills less and less of the data space.
- In order to maintain an accurate representation of the space, the **data for analysis grows exponentially**.

The curse of dimensionality



The curse of dimensionality

Example 2:

- if we are trying to predict a target, that is dependent on **two attributes: gender and age group.**
- we should ideally capture the targets for all possible combinations of values for the two attributes as shown in figure 1.

Age group levels

- Children (0-14yrs)
- Youth (15-24yrs)
- Adult (25-60yrs)
- Senior (61 and over)

Gender levels

- Male
- Female

Age group	Gender	Target
Children	Male	T1
Youth	Male	T2
Adult	Male	T3
Senior	Male	T4
Children	Female	T5
Youth	Female	T6
Adult	Female	T7
Senior	Female	T8

Figure 1. Combination of values of 2 attributes for generalizing a model

The curse of dimensionality



The curse of dimensionality

- If this data is used to train a model that is capable of learning the mapping between the attribute values and the target, its performance could be generalized.
- As long as the future unseen data comes from this distribution (a combination of values), the model would predict the target accurately.
- In the example, we assume that the target value depends on gender and age group only. If the target depends on a third attribute, let's say body type, the number of training samples required to cover all the combinations increases phenomenally.
- The combinations are shown in figure 2. For two variables, we needed eight training samples. For three variables, we need 24 samples.

The curse of dimensionality



The curse of dimensionality

Age group	Gender	Body type	Target
Children	Male	Normal	T1
Children	Male	Over weight	T2
Children	Male	Obese	T3
Youth	Male	Normal	T4
Youth	Male	Over weight	T5
Youth	Male	Obese	T6
Adult	Male	Normal	T7
Adult	Male	Over weight	T8
Adult	Male	Obese	T9
Senior	Male	Normal	T10

Senior	Male	Over weight	T11
Senior	Male	Obese	T12
Children	Female	Normal	T13
Children	Female	Over weight	T14
Children	Female	Obese	T15
Youth	Female	Normal	T16
Youth	Female	Over weight	T17
Youth	Female	Obese	T18
Adult	Female	Normal	T19
Adult	Female	Over weight	T20
Adult	Male	Obese	T21
Senior	Male	Normal	T22
Senior	Male	Over weight	T23
Senior	Male	Obese	T24

Figure 2. Combination of values of 3 attributes for generalizing a model

The curse of dimensionality



The curse of dimensionality

- The above examples show that, as the **number of attributes** or the **dimensions increases**, the **number of training samples required to generalize a model also increase phenomenally**.
- In reality, the available training samples may not have observed targets for all combinations of the attributes. This is because some combination occurs more often than others. Due to this, the training samples available for building the model may not capture all possible combinations. This aspect, where the **training samples do not capture all combinations**, is referred to as '**Data sparsity**' or simply '**sparsity**' in high dimensional data.
- Training a model with sparse data could lead to **high-variance** or **overfitting condition**.

The curse of dimensionality

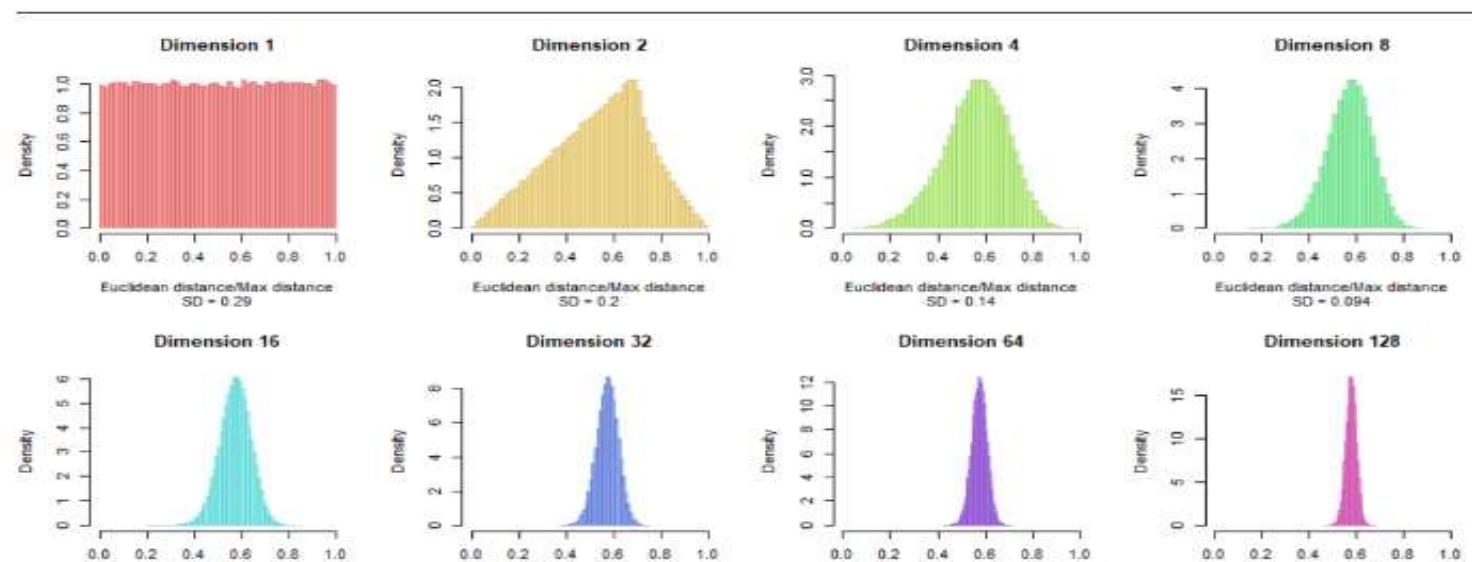


The curse of dimensionality

Distance Concentration

- Distance concentration refers to the problem of all the pairwise distances between different samples/points in the space converging to the same value as the dimensionality of the data increases.
- Due to distance concentration, the concept of proximity or similarity of the samples may not be qualitatively relevant in higher dimensions.

Example:



The curse of dimensionality



The curse of dimensionality

Distance Concentration

- As the number of dimensions increases, we see that the **spread of the frequency plot decreases** indicating that distances between different samples or points tend towards a single value as the dimension increases.

The curse of dimensionality



The curse of dimensionality

- Although the curse of dimensionality certainly raises **important issues for pattern recognition applications**, it does not prevent us from finding effective techniques applicable to high-dimensional spaces.
- The reasons for this are **twofold**:
 1. real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the target variables occur may be so confined.
 2. real data will typically exhibit some smoothness properties . so that for the most part small changes in the input variables will produce small changes in the target variables, and so we can exploit local interpolation-like techniques to allow us to make predictions of the target variables for new values of the input variables.

Dimensionality Reduction



Dimensionality Reduction

- The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.
- Dimensionality reduction technique can be defined as, "*It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information.*"
- These techniques are widely used in machine learning for obtaining a better fit predictive model while solving the classification and regression problems.
- It is commonly used in the fields that deal with high-dimensional data, such as **speech recognition, signal processing, bioinformatics, etc.** It can also be used for **data visualization, noise reduction, cluster analysis, etc.**

Dimensionality Reduction



Approaches of Dimension Reduction

➤ There are **two ways** to apply the dimension reduction techniques. Those are

1. Feature selection
2. Feature extraction

Feature Selection

➤ Feature selection is the **process of selecting the subset of the relevant features and leaving out the irrelevant features present in a dataset to build a model of high accuracy.** In other words, it is a way of selecting the optimal features from the input dataset.

➤ **Three methods** are used for the feature selection:

1. Filters Methods
2. Wrappers Methods
3. Embedded Methods

Dimensionality Reduction



Approaches of Dimension Reduction

1. Filters Methods

- In this method, the **dataset is filtered**, and a **subset that contains only the relevant features** is taken.
- Some common techniques of filters method are:
 1. Correlation
 2. Chi-Square Test
 3. ANOVA
 4. Information Gain, etc.

Dimensionality Reduction



Approaches of Dimension Reduction

2. Wrappers Methods

- The wrapper method has the **same goal as the filter method**, but it **takes a machine learning model for its evaluation**. In this method, some features are fed to the ML model, and **evaluate the performance**.
- The **performance decides whether to add those features or remove to increase the accuracy of the model**.
- This method is more accurate than the filtering method but complex to work.
- Some common techniques of wrapper methods are:
 1. Forward Selection
 2. Backward Selection
 3. Bi-directional Elimination

Dimensionality Reduction



Approaches of Dimension Reduction

3. Embedded Methods:

- Embedded methods check the different training iterations of the machine learning model and evaluate the importance of each feature.
- Some common techniques of Embedded methods are:
 1. LASSO
 2. Elastic Net
 3. Ridge Regression, etc.

Dimensionality Reduction



Approaches of Dimension Reduction

Feature Extraction:

- Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions.
- This approach is useful when we want to keep the whole information but use fewer resources while processing the information.
- Some common feature extraction techniques are:
 1. Principal Component Analysis
 2. Linear Discriminant Analysis
 3. Kernel PCA
 4. Quadratic Discriminant Analysis

Dimensionality Reduction



Common techniques of Dimensionality Reduction

➤ **Ten** important techniques used to reduce dimensionality are:

1. Principal Component Analysis
2. Backward Elimination
3. Forward Selection
4. Score comparison
5. Missing Value Ratio
6. Low Variance Filter
7. High Correlation Filter
8. Random Forest
9. Factor Analysis
10. Auto-Encoder

Dimensionality Reduction



Techniques of Dimensionality Reduction

1. Principal Component Analysis (PCA)

- Principal Component Analysis, or PCA, is a dimensionality-reduction technique in which high dimensional correlated data is transformed to a lower dimensional set of uncorrelated components, referred to as principal components.
- The lower dimensional principle components capture most of the information in the high dimensional dataset

Dimensionality Reduction



Techniques of Dimensionality Reduction

1. *Principal Component Analysis (PCA)*

➤ Figure shows a simple example in which a 10-dimensional data is transformed to 10-principle components. To capture 90% of the variance in the data only 3 principle components are needed. Hence, we have reduced a 10-dimensional data to 3-dimensions.

Component	Initial Eigenvalues		
	Total	% of Variance	Cumulative %
1	5.994	59.938	59.938
2	1.654	16.545	76.482
3	1.123	11.227	87.709
4	.339	3.389	91.098
5	.254	2.541	93.640
6	.199	1.994	95.633
7	.155	1.547	97.181
8	.130	1.299	98.480
9	.091	.905	99.385
10	.061	.615	100.000

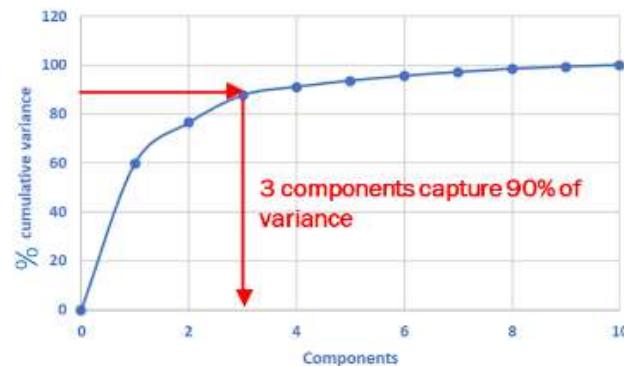


Figure Example of converting 10-dimensional data to 3-dimensional data through PCA

Dimensionality Reduction



Techniques of Dimensionality Reduction

2. Factor Analysis (FA)

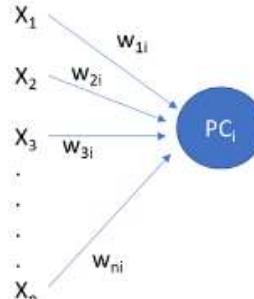
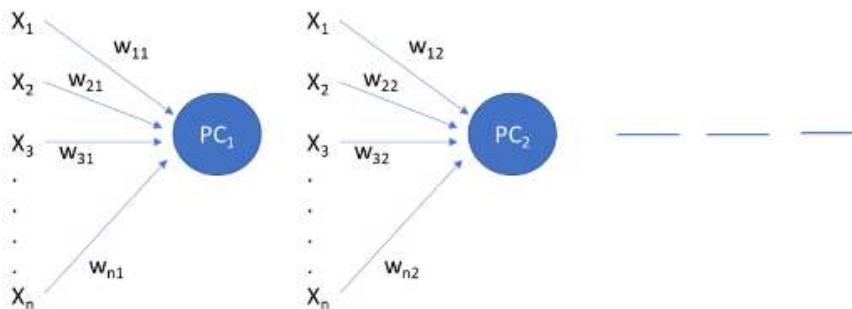
- Factor analysis is based on the assumption that all the observed attributes in a dataset can be represented as a weighted linear combination of latent factors.
- The intuition in this technique is that an ‘n’ dimensional data can be represented by ‘m’ factors ($m < n$).
- The main difference between PCA and FA is in the fact that While PCA synthesizes components from the base attributes, FA decomposes the attributes into latent factors as shown in figure

Dimensionality Reduction

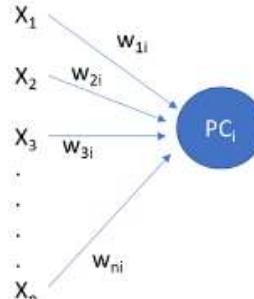


Techniques of Dimensionality Reduction

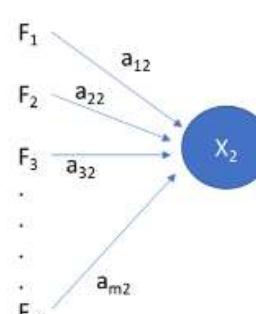
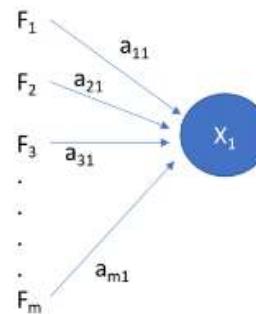
2. Factor Analysis (FA)



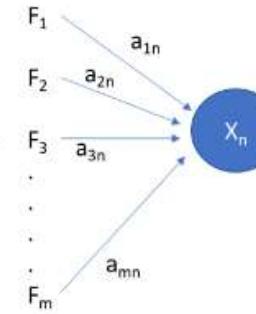
— — —



Principle component analysis ($i \leq n$)



— — —



Factor analysis ($m \leq n$)

Dimensionality Reduction



Techniques of Dimensionality Reduction

3. Independent Component Analysis (ICA)

- ICA assumes that all the attributes are essentially a mixture of independent components and resolves the variables into a combination of these independent components.
- ICA is perceived to be more robust than PCA is generally used when PCA and FA fail.

Dimensionality Reduction



Techniques of Dimensionality Reduction

4. Backward Feature Elimination

- The backward feature elimination technique is mainly used while developing Linear Regression or Logistic Regression model.
- Below steps are performed in this technique to reduce the dimensionality or in feature selection:
 1. firstly, all the n variables of the given dataset are taken to train the model.
 2. The performance of the model is checked.
 3. Now we will remove one feature each time and train the model on n-1 features for n times, and will compute the performance of the model.
 4. We will check the variable that has made the smallest or no change in the performance of the model, and then we will drop that variable or features; after that, we will be left with n-1 features.
 5. Repeat the complete process until no feature can be dropped.

Dimensionality Reduction



Techniques of Dimensionality Reduction

5. Backward Feature Elimination

➤ In this technique, by selecting the optimum performance of the model and maximum tolerable error rate, we can define the optimal number of features required for the machine learning algorithms.

5. Forward Feature Selection

➤ Forward feature selection follows **the inverse process of the backward elimination process**. It means, in this technique, **we don't eliminate the feature; instead, we will find the best features that can produce the highest increase in the performance of the model**.

Dimensionality Reduction



Techniques of Dimensionality Reduction

5. Forward Feature Selection

➤ Below steps are performed in this technique:

1. We start with a single feature only, and progressively we will add each feature at a time.
2. Here we will train the model on each feature separately.
3. The feature with the best performance is selected.
4. The process will be repeated until we get a significant increase in the performance of the model.

Dimensionality Reduction



Techniques of Dimensionality Reduction

6. Missing Value Ratio

➤ If a dataset has too many missing values, then we drop those variables as they do not carry much useful information. To perform this, we can **set a threshold level, and if a variable has missing values more than that threshold, we will drop that variable**. The higher the threshold value, the more efficient the reduction.

7. Low Variance Filter

➤ As same as missing value ratio technique, data columns with some changes in the data have less information. Therefore, we need **to calculate the variance of each variable, and all data columns with variance lower than a given threshold are dropped** because low variance features will not affect the target variable.

Dimensionality Reduction



Techniques of Dimensionality Reduction

8. High Correlation Filter

- High Correlation refers to the case when **two variables carry approximately similar information**. Due to this factor, the performance of the model can be degraded.
- This correlation between the independent numerical variable gives the calculated value of the correlation coefficient.
- If this value is higher than the threshold value, we can remove one of the variables from the dataset.
- We can consider those variables or features that show a high correlation with the target variable.

Dimensionality Reduction



Techniques of Dimensionality Reduction

9. Random Forest

- Random Forest is a popular and very useful feature selection algorithm in machine learning.
- This algorithm contains an **in-built feature importance package**, so we do not need to program it separately.
- In this technique, we need to generate a large set of trees against the target variable, and with the help of usage statistics of each attribute, we need to find the subset of features.
- Random forest algorithm **takes only numerical variables**, so we need to convert the input data into numeric data using **hot encoding**.

Dimensionality Reduction



Techniques of Dimensionality Reduction

10. Auto-encoders

➤ One of the popular methods of dimensionality reduction is auto-encoder, which is a type of ANN or artificial neural network, and its **main aim is to copy the inputs to their outputs**.

➤ In this, the input is compressed into latent-space representation, and output is occurred using this representation. It has mainly two parts:

1. Encoder:

The function of the encoder is to compress the input to form the latent-space representation.

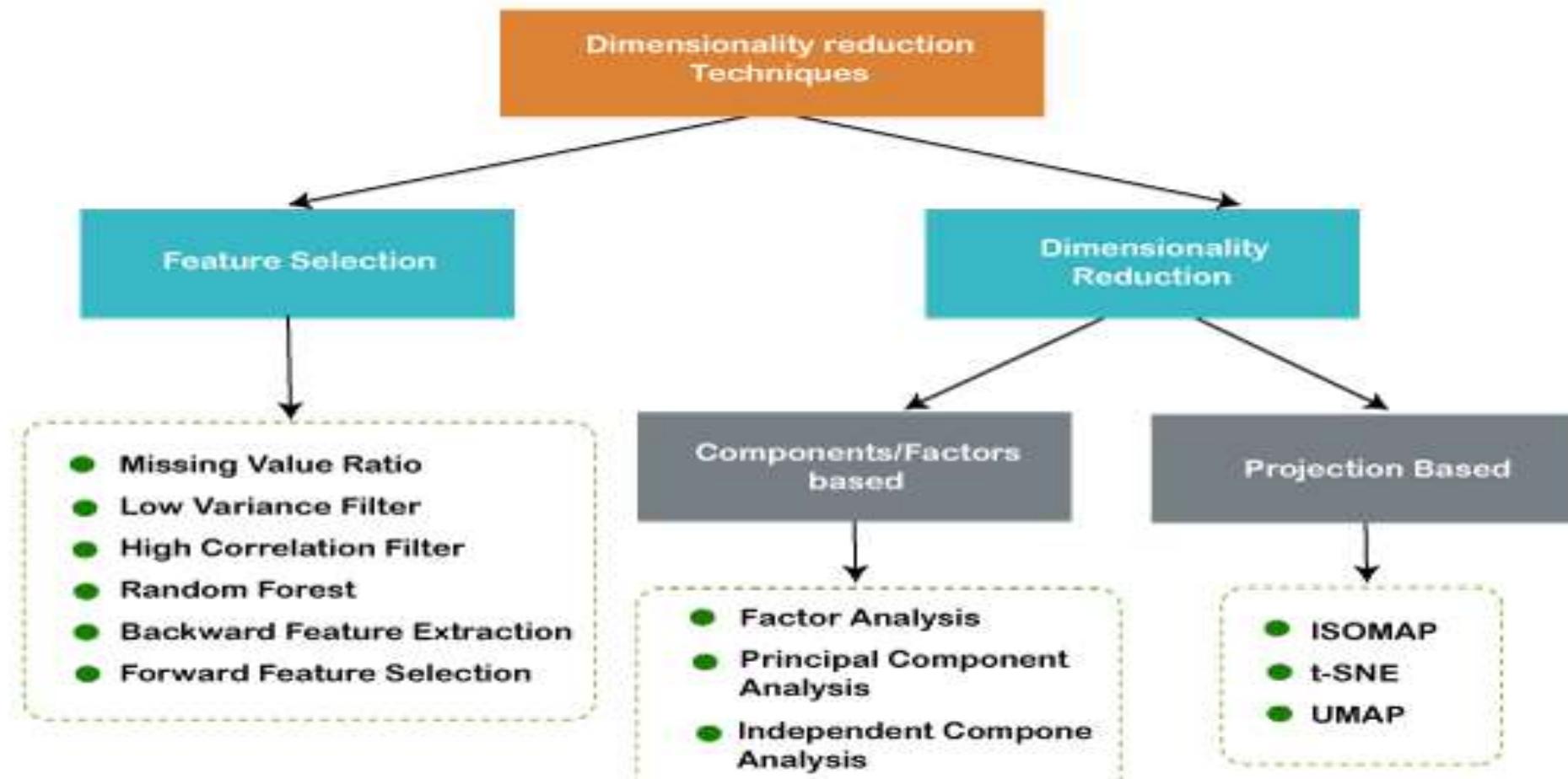
2. Decoder:

The function of the decoder is to recreate the output from the latent-space representation.

Dimensionality Reduction



Techniques of Dimensionality Reduction



Dimensionality Reduction



Benefits of applying Dimensionality Reduction

- Some **benefits of applying dimensionality reduction** technique to the given dataset are given below:

 1. By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
 2. Less Computation training time is required for reduced dimensions of features.
 3. Reduced dimensions of features of the dataset help in visualizing the data quickly.
 4. It removes the redundant features (if present) by taking care of multicollinearity.

Dimensionality Reduction



Disadvantages of dimensionality Reduction

➤ There are also some **disadvantages of applying the dimensionality reduction**, which are given below:

1. Some data may be lost due to dimensionality reduction.
2. In the PCA dimensionality reduction technique, sometimes the principal components required to consider are unknown.



K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification.But mostly it is used for the Classification problems.



K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-NN is a **non-parametric algorithm**, which means it does **not make any assumption** on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.



K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

Example:

Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



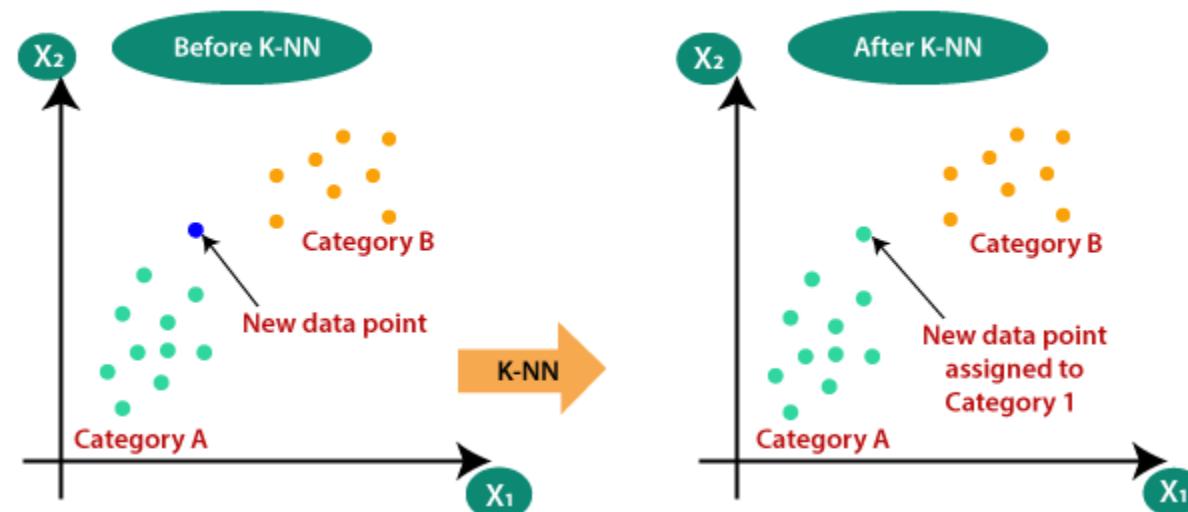


K-Nearest Neighbor(KNN) Algorithm

Why do we need a K-NN Algorithm?

- Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x_1 , so this data point will lie in which of these categories.
- To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily **identify the category or class of a particular dataset**.

Consider the below diagram:





K-Nearest Neighbor(KNN) Algorithm

How does K-NN work?

➤The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of **K number of neighbors**

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.



K-Nearest Neighbor(KNN) Algorithm

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them.
- The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.



K-Nearest Neighbor(KNN) Algorithm

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.



K-Nearest Neighbor(KNN) Algorithm

Python implementation of the KNN algorithm

➤ Python's **Scikit-Learn library** can be used to implement the KNN algorithm

Example:

➤ There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network.

➤ The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:



K-Nearest Neighbor(KNN) Algorithm

Python implementation of the KNN algorithm

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



K-Nearest Neighbor(KNN) Algorithm

Steps to implement the K-NN algorithm:

1. Data Pre-processing step
2. Fitting the K-NN algorithm to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



K-Nearest Neighbor(KNN) Algorithm

1. Data Pre-Processing Step:

➤ In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv(  
    'user_data.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0



K-Nearest Neighbor(KNN) Algorithm

Data Pre-Processing Step:

➤ The next step is to **split our dataset into its attributes and labels**. i.e, extract the **dependent and independent variables** from the given dataset.

#Extracting Independent and dependent Variable

```
x= data_set.iloc[:, [2,3]].values
```

```
y= data_set.iloc[:, 4].values
```

➤ In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4.

➤ Print(x)

➤ Print(y)

x - NumPy array	
0	19
1	35
2	26
3	27
4	19
5	27
6	27
7	32

y - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1



K-Nearest Neighbor(KNN) Algorithm

Data Pre-Processing Step:

Train Test Split

➤ To avoid **over-fitting**, we will **divide our dataset into training and test splits**, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application.

➤ To create training and test splits, execute the following script:

```
# 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_st  
ate=0)
```

➤ The above script **splits the dataset into 75% train data and 25% test data**.



K-Nearest Neighbor(KNN) Algorithm

Data Pre-Processing Step:

Train Test Split

For **test** set:

x_test - NumPy array		y_test - NumPy array	
0	0	1	
0	-0.804802	0.504964	0
1	-0.0125441	-0.567782	0
2	-0.309641	0.157046	0
3	-0.804802	0.273019	0
4	-0.309641	-0.567782	0
5	-1.1019	-1.43758	0
6	-0.70577	-1.58254	0
7	-0.210609	2.15757	1
8	-1.99319	-0.0459058	0
9	0.878746	-0.770734	0
10	-0.804802	-0.596776	0
11	-1.00287	-0.422817	0



K-Nearest Neighbor(KNN) Algorithm

Data Pre-Processing Step:

Train Test Split

For training set:

	x_test - NumPy array			y_test - NumPy array	
	0	1			
0	-0.804802	0.504964		0	0
1	-0.0125441	-0.567782		1	0
2	-0.309641	0.157046		2	0
3	-0.804802	0.273019		3	0
4	-0.309641	-0.567782		4	0
5	-1.1019	-1.43758		5	0
6	-0.70577	-1.58254		6	0
7	-0.210609	2.15757		7	1
8	-1.99319	-0.0459058		8	0
9	0.878746	-0.770734		9	0
10	-0.804802	-0.596776		10	0
11	-1.00287	-0.422817		11	0

K-Nearest Neighbor(KNN) Algorithm



Data Pre-Processing Step:

Feature Scaling

➤ Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

#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)
```

➤ From the output image, we can see that our data is successfully scaled.

x_test - NumPy array		y_test - NumPy array	
0	1	0	0
-0.804802	0.504964	1	0
-0.0125441	-0.567782	0	0
-0.309641	0.157046	0	0
-0.804802	0.273019	0	0
-0.309641	-0.567782	0	0
-1.1019	-1.43758	0	0
-0.70577	-1.58254	0	0
-0.210609	2.15757	1	0
-1.99319	-0.0459058	0	0
0.878746	-0.770734	0	0
-0.804802	-0.596776	0	0
-1.00287	-0.422817	0	0
-0.111576	-0.422817	0	0



K-Nearest Neighbor(KNN) Algorithm

2. Fitting K-NN classifier to the Training data:

- Now we will fit the K-NN classifier to the training data.
- To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library.
- After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be

n_neighbors: To define the required neighbors of the algorithm. Usually, it takes 5.

metric='minkowski': This is the default parameter and it decides the distance between the points.

p=2: It is equivalent to the standard Euclidean metric.

- And then we will fit the classifier to the training data.



K-Nearest Neighbor(KNN) Algorithm

Fitting K-NN classifier to the Training data:

```
#Fitting K-NN classifier to the training set  
from sklearn.neighbors import KNeighborsClassifier  
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )  
classifier.fit(x_train, y_train)
```

Output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
metric_params=None, n_jobs=None, n_neighbors=5, p=2,  
weights='uniform')
```



K-Nearest Neighbor(KNN) Algorithm

3. Predicting the Test Result:

To predict the test set result, we will create a `y_pred` vector .

```
#Predicting the test set result
```

```
y_pred= classifier.predict(x_test)
```

Output:

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0
12	0



K-Nearest Neighbor(KNN) Algorithm

4. Test Accuracy:

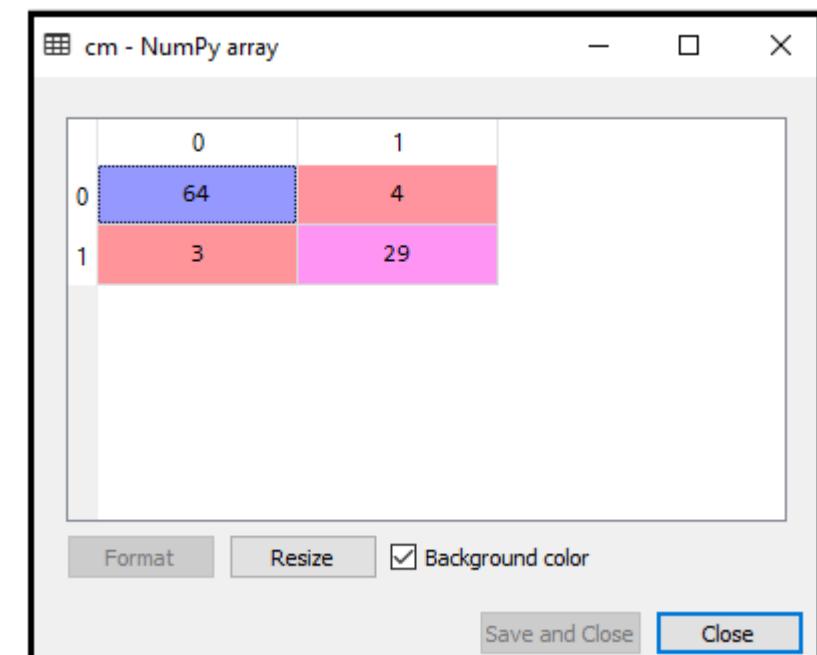
- Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier.

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix(y_test, y_pred)
```

Output:

see there are $64+29= 93$ correct predictions and $3+4= 7$ incorrect predictions.





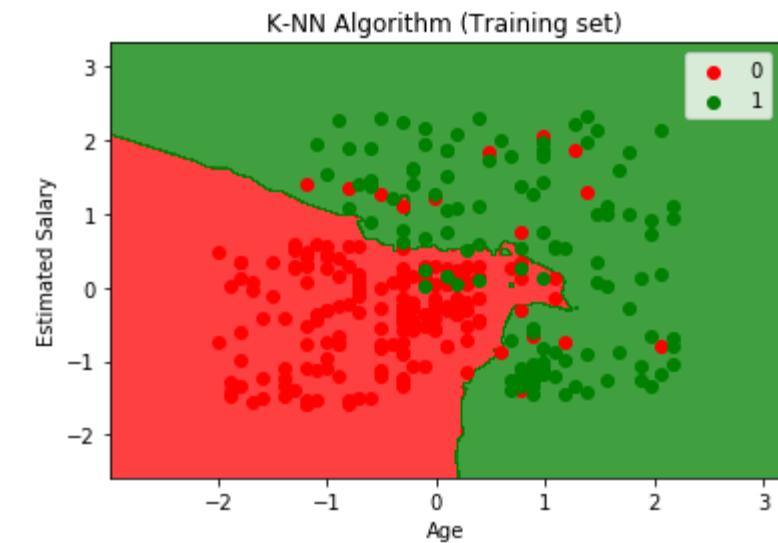
K-Nearest Neighbor(KNN) Algorithm

5. Visualizing the Training set result:

- Now, we will visualize the training set result for K-NN model

#Visulaizing the trianing set result

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```





K-Nearest Neighbor(KNN) Algorithm

Visualizing the Training set result:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.



K-Nearest Neighbor(KNN) Algorithm

Visualizing the Test set result:

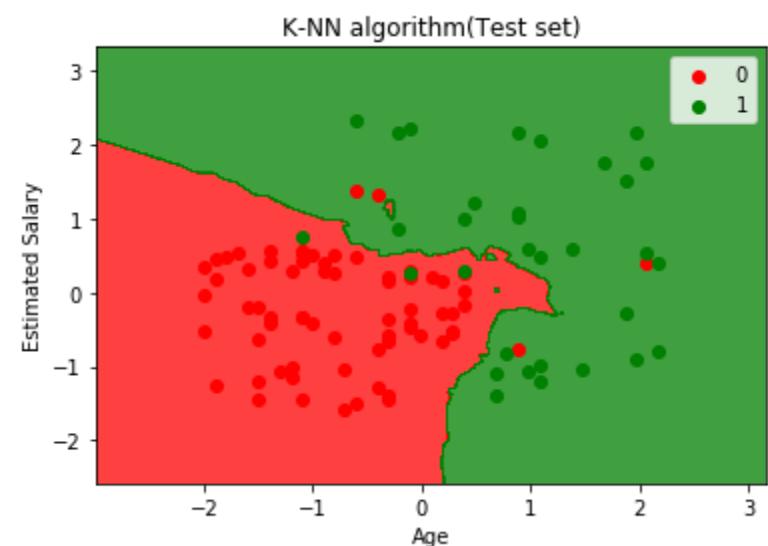
➤ After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x_train** and **y_train** will be replaced by **x_test** and **y_test**.



K-Nearest Neighbor(KNN) Algorithm

Visualizing the Test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```





K-Nearest Neighbor(KNN) Algorithm

- As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.
- However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).



Machine Learning

Machine Learning:

Machine learning is turning things(data) into numbers and **finding patterns** in those numbers.

- For finding patterns, algorithms are used.
- An algorithm is a specific set of steps to perform a task.

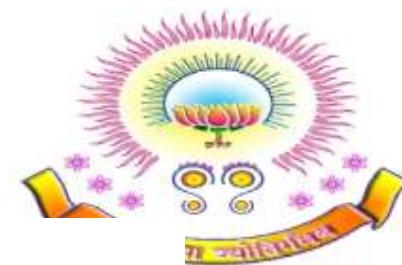
An “*algorithm*” in machine learning is a procedure that is run on data to create a machine learning “*model*.”

- A machine learning *algorithm* is written to derive the *model*.
- The *model* identifies the *patterns* in data that **fit** the *dataset*. **Fit** is a synonym to “**find patterns in data**”.
- A “*model*” in machine learning is **the output of a machine learning algorithm run on data**



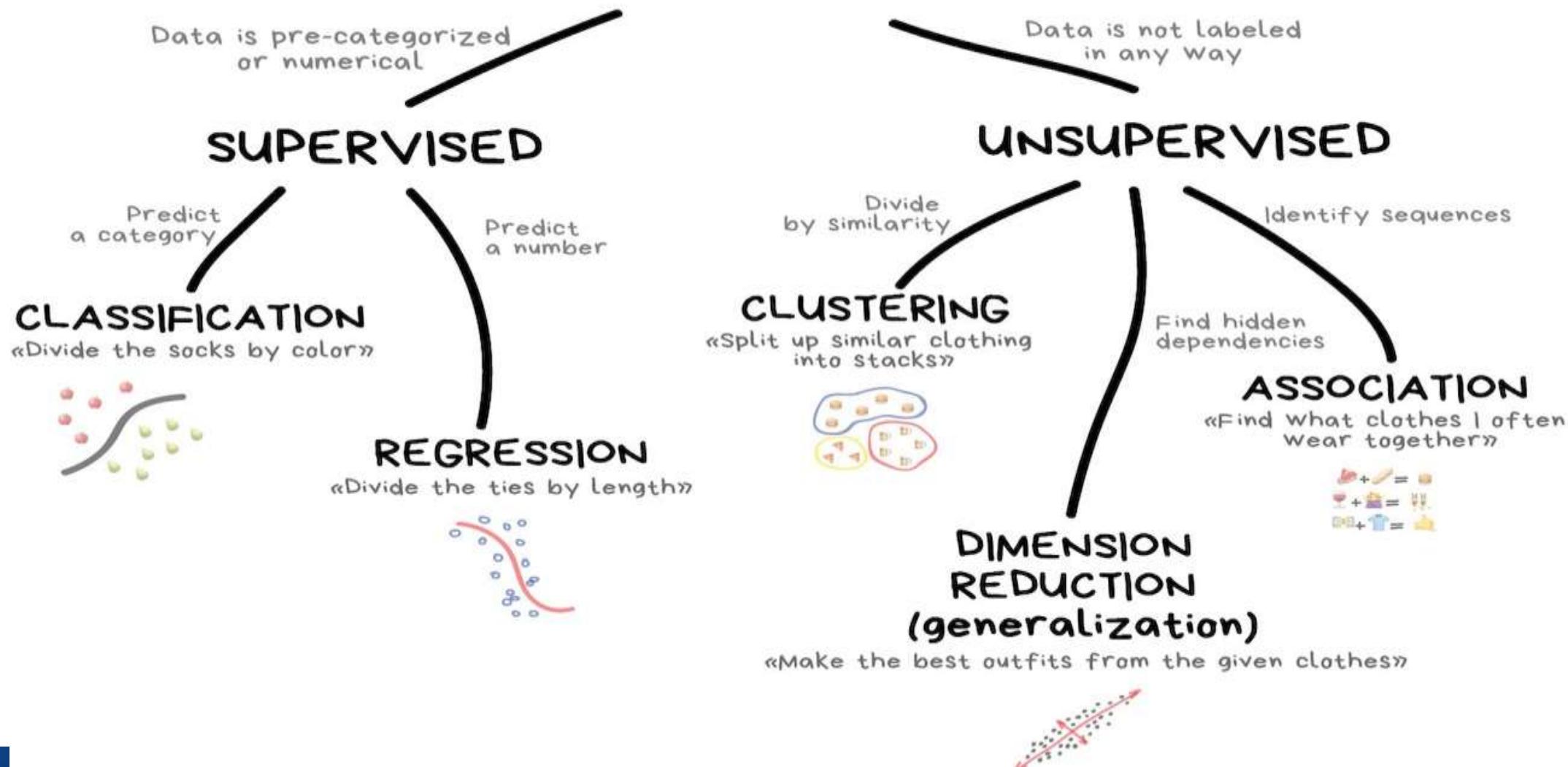
Machine Learning

- **Models** are like the general equation **of** a line $y = a + bx$, while **patterns** are like a specific equation, e.g. $y = 5 + 2x$.
- Machine learning is about generalizing correctly to brand-new situations.
- The basic task of machine learning is to create a model that can predict or classify different patterns from data.
- One of the applications of this is the classification of spam or non-spam data.
- The algorithms adaptively improve their performance as the number of samples available for learning increases.
- Two main types of machine learning are supervised and unsupervised machine learning.



Patterns

CLASSICAL MACHINE LEARNING





Classification

Classification:

Classification is the process of recognizing, understanding, and grouping ideas and objects into present categories or “sub-populations.” Using pre-categorized training datasets, machine learning programs use a variety of algorithms to classify future datasets into categories.

or

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

Or

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.



Classification

Classification Algorithms

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.
- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
- the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc.
- In classification algorithm, a discrete output function(y) is mapped to input variable(x).

$$y=f(x), \text{ where } y = \text{categorical output}$$

- The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.



Classification

Classification Algorithms

- The algorithm which implements the classification on a dataset is known as a **classifier**.
- There are two types of Classifications:
 1. **Binary Classifier**
 2. **Multi-class Classifier**

Binary Classifier:

If the classification problem has only **two possible outcomes**, then it is called as BinaryClassifier.

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-classClassifier.

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

Classification



Types of ML Classification Algorithms:

➤ Classification Algorithms can be further divided into the Mainly two category: 1. **Linear Models** 2. **Non-linear Models**

Linear Models

1. Linear Regression
2. Logistic Regression
3. Support Vector Machines

Non-linear Models

1. K-Nearest Neighbors
2. Kernel SVM
3. Naïve Bayes
4. Decision Tree Classification
5. Random Forest Classification

1. Linear Regression

Regression

The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.

- This line can be used to predict future values.

- Let's consider a dataset in which we have a number of responses y per feature x :

X	10	11	12	13	14	15	16	17	18	19
Y	11	13	12	15	17	18	18	19	20	22

- We need to fit the linear equation $y=mx+b$
- Where **m** is called slope or coefficient and **b** is called intercept.



Classification

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
- It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used.
- Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**



Classification

Logistic Regression in Machine Learning

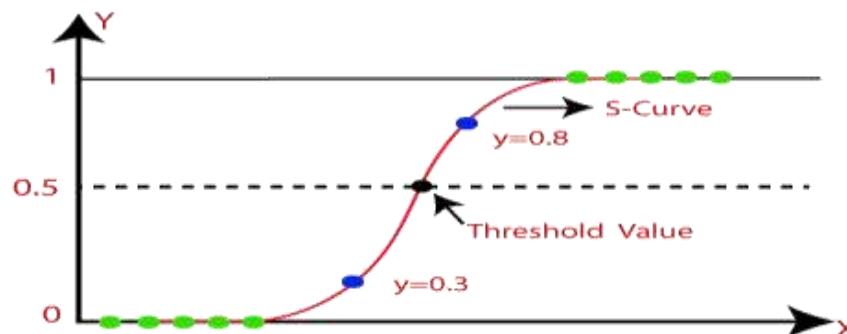
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped **logistic function**, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the **likelihood** of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the **ability to provide probabilities and classify new data using continuous and discrete datasets**.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.



Classification

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- The below image is showing the logistic function:



- Linear Regression Equation: $y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n$

Where, y is a dependent variable and x₁, x₂ ... and X_n are explanatory variables.

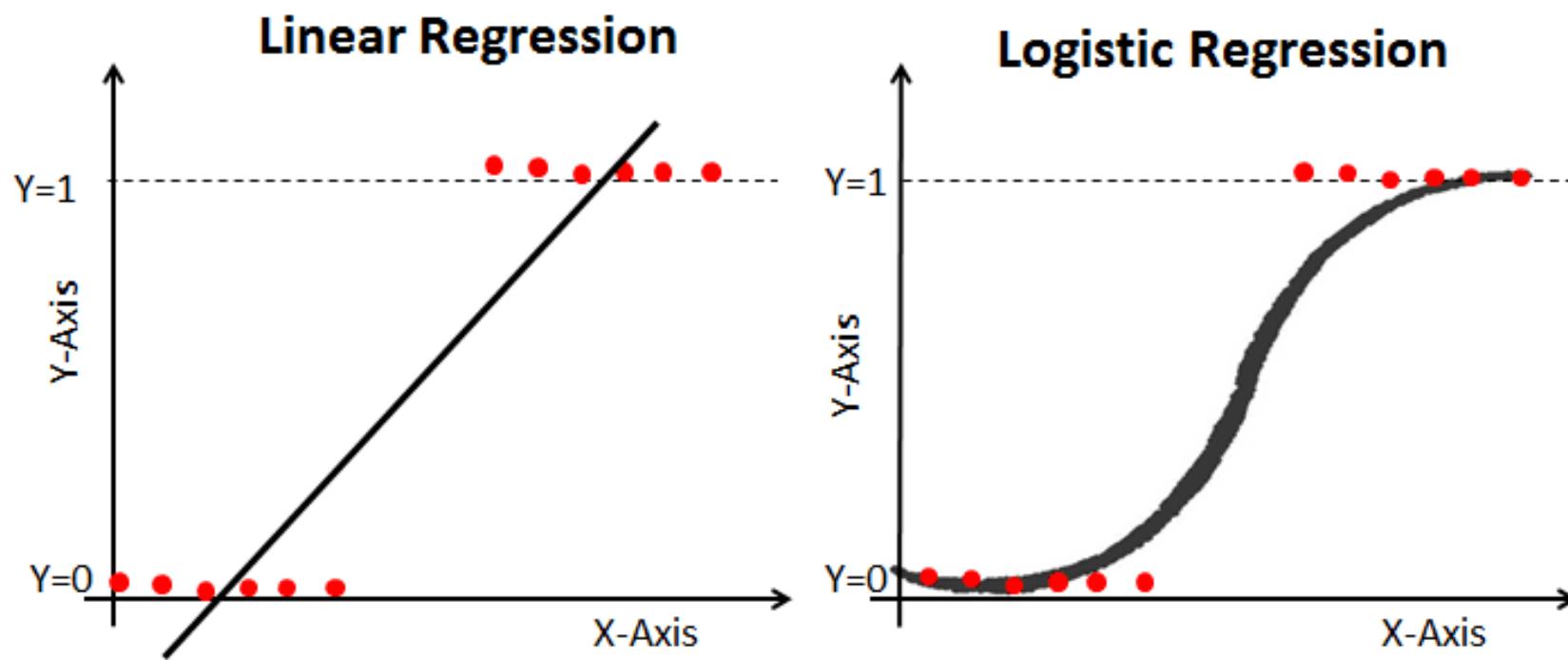
Sigmoid Function:

$$p = \frac{1}{1 + e^{-y}}$$

- The dependent variable in logistic regression follows Bernoulli Distribution.

Linear Regression Vs. Logistic Regression

- **Linear regression** gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.



Age	Lic_bought
21	0
35	1
24	0
18	0
55	1
45	1
32	1

- $Y = mx + b$

$$m = (n \sum xy - \sum x \sum y) / (n \sum x^2 - (\sum x)^2)$$

$$b = (\sum y \sum x^2 - \sum x \sum xy) / (n \sum x^2 - (\sum x)^2)$$



Classification

Logistic Function (Sigmoid Function):

➤ In logistic regression, we use the concept of the **threshold value**, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

1. The **dependent variable must be categorical** in nature.
2. The **independent variable should not have multi-collinearity**.



Classification

Logistic Regression Equation:

- The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:
- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by $(1-y)$:

$$\frac{y}{1-y}; 0 \text{ for } y=0, \text{ and infinity for } y=1$$



Classification

Logistic Regression Equation:

➤ But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

➤ The above equation is the **final equation for Logistic Regression.**



Classification

Type of Logistic Regression:

➤ On the basis of the **categories**, Logistic Regression can be classified into **three** types:

1. Binomial
2. Multinomial
3. Ordinal

Binomial:

In binomial Logistic regression, there can be only two possible types of the dependent variables.

Examples:

0 or 1,

Pass or Fail, etc.



Classification

Type of Logistic Regression:

Multinomial:

In multinomial Logistic regression, there can be **3 or more possible unordered types of the dependent variable.**

Examples:

"cat", "dogs", or "sheep"

Ordinal:

In ordinal Logistic regression, there can be **3 or more possible ordered types of dependent variables.**

Examples:

"low", "Medium", or "High".



Classification

Steps in Logistic Regression:

- To implement the Logistic Regression using Python the following steps are used:
 1. Data Pre-processing step
 2. Fitting Logistic Regression to the Training set
 3. Predicting the test result
 4. Test accuracy of the result(Creation of Confusion matrix)
 5. Visualizing the test set result.



Classification

Example:

- There is a **dataset** given which **contains the information of various users obtained from the social networking sites**. There is a **car making company that has recently launched a new SUV car**. So the company wanted to check how many users from the dataset, wants to purchase the car.
- For this problem, we will build a **Machine Learning model using the Logistic regression algorithm**. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.



Classification

Example:

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



Classification

Data Pre-processing step:

- In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv('user_data.csv')
```

By executing the above lines of code, we will get the dataset as the output.



Classification

data_set - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Format Resize Background color Column min/max Save and Close Close



Classification

Data Pre-processing step:

- we will extract the dependent and independent variables from the given dataset.

#Extracting Independent

```
x= data_set.iloc[:, [2,3]].values
```

#Extracting dependent Variable

```
y= data_set.iloc[:, 4].values
```

➤ In the above code, we have **taken [2, 3] for x** because our **independent variables are age and salary**, which are at index 2, 3. And we have **taken 4 for y** variable because our **dependent variable** is at index 4.

➤ The output will be:



Classification

x - NumPy array

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000
10	26	80000
11	26	52000
12	20	86000

Format Resize Background color

Save and Close Close

y - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Format Resize Background color

Save and Close Close



Classification

- we will split the dataset into a training set 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)
```

The output for this is given below

The image shows two side-by-side data visualization windows. The left window is titled "x_test - NumPy array" and displays a 13x2 grid of numerical values. The right window is titled "y_test - NumPy array" and displays a 13x1 column of binary values (0 or 1). Both windows have "Format", "Resize", and "Background color" buttons at the bottom.

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0



Classification

For training set:

	0	1	
0	-0.804802	0.504964	
1	-0.0125441	-0.567782	
2	-0.309641	0.157046	
3	-0.804802	0.273019	
4	-0.309641	-0.567782	
5	-1.1019	-1.43758	
6	-0.70577	-1.58254	
7	-0.210609	2.15757	
8	-1.99319	-0.0459058	
9	0.878746	-0.770734	
10	-0.804802	-0.596776	
11	-1.00287	-0.422817	
12	-0.111576	-0.422817	

Format Resize Background color
[Save and Close](#) [Close](#)

	0	
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	1	
8	0	
9	0	
10	0	
11	0	
12	0	

Format Resize Background color
[Save and Close](#) [Close](#)



Classification

- In logistic regression, we will do **feature scaling because** we want **accurate result of predictions**. Here we will only scale the independent variable because dependent variable have only 0 and 1 values.

```
from sklearn.preprocessing import StandardScaler
```

```
st_x= StandardScaler()
```

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

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

The scaled output is given below:



Classification

x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

Format Resize Background color

Save and Close Close

x_train - NumPy array

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.210609	-0.190872
10	-0.309641	-1.29261
11	-0.309641	-0.567782
12	0.383585	0.0990599

Format Resize Background color

Save and Close Close



Classification

2. Fitting Logistic Regression to the Training set:

- We have well prepared our dataset, and now we will **train the dataset using the training set**. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.
- After importing the class, we will create a classifier object and use it to fit the model to the logistic regression.

```
#Fitting Logistic Regression to the training set  
from sklearn.linear_model import LogisticRegression  
classifier= LogisticRegression(random_state=0)  
classifier.fit(x_train, y_train)
```



Classification

2. Fitting Logistic Regression to the Training set:

Output:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, l1_ratio=None, max_iter=100,  
                    multi_class='warn', n_jobs=None, penalty='l2',  
                    random_state=0, solver='warn', tol=0.0001, verbose=0,  
                    warm_start=False)
```

Hence our model is well fitted to the training set.

3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data.

```
#Predicting the test set result
```

```
y_pred= classifier.predict(x_test)
```



Classification

3. Predicting the Test Result

Output:

- a new vector (`y_pred`) will be created under the variable explorer option. It can be seen as:

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0

The output image shows the corresponding predicted users who want to purchase or not purchase the car



Classification

4. Test Accuracy of the result

- Now we will create the **confusion matrix** here **to check the accuracy** of the classification.
- To create it, we need to import the **confusion_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true**(the actual values) and **y_pred** (the targeted value return by the classifier).

```
#Creating the Confusion matrix
```

```
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix()
```



Classification

4. Test Accuracy of the result

Output:

- a new confusion matrix will be created.

We can find the accuracy of the predicted result by interpreting the confusion matrix.

By above output, we can interpret that $65+24= 89$ (Correct Output) and $8+3= 11$ (Incorrect Output).

		0	1
0	65	3	
1	8	24	

Format Resize Background color Save and Close Close



Classification

5. Visualizing the training set result

➤ Finally, we will **visualize the training set result**. To visualize the result, we will use **ListedColormap** class of matplotlib library.

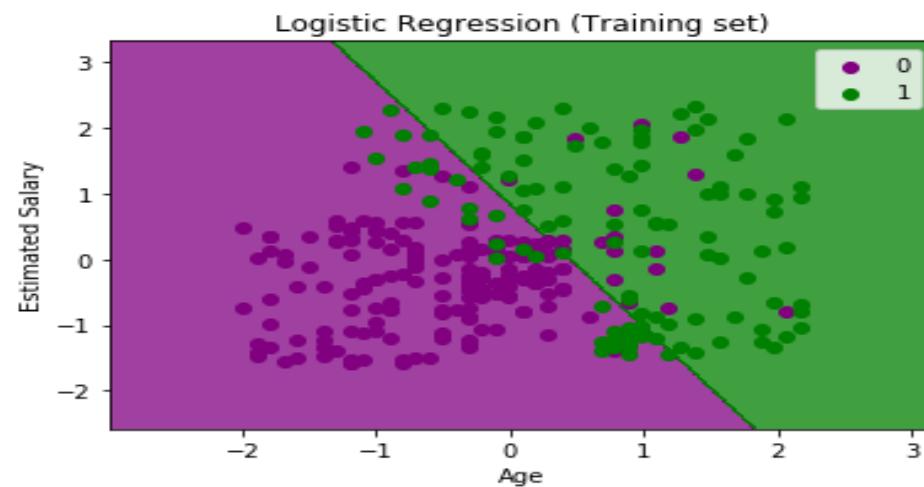
```
from matplotlib.colors import ListedColormap  
x_set, y_set = x_train, y_train  
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),  
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))  
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),  
alpha = 0.75, cmap = ListedColormap(['purple','green' )))  
mtp.xlim(x1.min(), x1.max())  
mtp.ylim(x2.min(), x2.max())  
for i, j in enumerate(nm.unique(y_set)):  
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],  
    c = ListedColormap(['purple', 'green'])(i), label = j)  
mtp.title('Logistic Regression (Training set)')  
mtp.xlabel('Age')  
mtp.ylabel('Estimated Salary')  
mtp.legend()  
mtp.show()
```



Classification

5. Visualizing the training set result

Output:





Classification

5. Visualizing the training set result

The graph can be explained in the below points:

- In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
- All these data points are the observation points from the training set, which shows the result for purchased variables.
- This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
- The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.
- The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.



Classification

5. Visualizing the training set result

- We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
- But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.



Classification

5. Visualizing the test set result:

- Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set).
- The code for the test set will remain same as above except that here we will use **x_test** and **y_test** instead of **x_train** and **y_train**.



Classification

5. Visualizing the test set result:

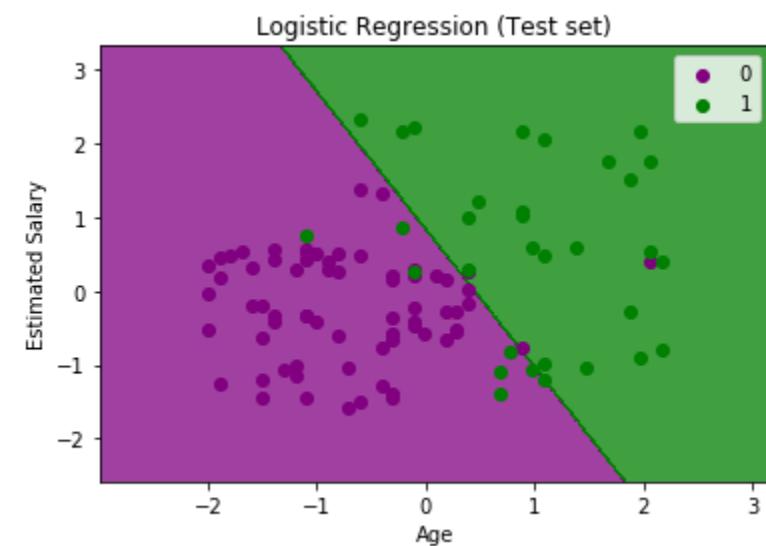
```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



Classification

5. Visualizing the test set result:

- The graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).
- Hence our model is pretty good and ready to make new predictions for this classification problem.





Classification

Advantages:

- Logistic regression is specifically meant for classification, it is useful in understanding how a set of independent variables affect the outcome of the dependent variable.

Disadvantages:

- The main disadvantage of the logistic regression algorithm is that it only works when the predicted variable is binary, it assumes that the data is free of missing values and assumes that the predictors are independent of each other.

Use Cases:

Identifying risk factors for diseases

Word classification

Weather Prediction

Voting Applications



Naïve Bayes Classifier Algorithm

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.**



Naïve Bayes Classifier Algorithm

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
 - **Bayes:**
It is called Bayes because it depends on the principle of [Bayes' Theorem](#).

Naïve Bayes Classifier Algorithm



Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the **probability of a hypothesis with prior knowledge**. It depends on **the conditional probability**.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

P(A | B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B | A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.



Naïve Bayes Classifier Algorithm

Working of Naïve Bayes' Classifier:

Example:

- Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions.
- **Problem:** If the weather is sunny, then the Player should play or not?
- So to solve this problem, we need to follow the below steps:
 1. Convert the given dataset into frequency tables.
 2. Generate Likelihood table by finding the probabilities of given features.
 3. Now, use Bayes theorem to calculate the posterior probability.



Naïve Bayes Classifier Algorithm

Working of Naïve Bayes' Classifier:

Solution:

- To solve this, first consider the below dataset:
- **Frequency table for the Weather Conditions:**

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes



Naïve Bayes Classifier Algorithm

Working of Naïve Bayes' Classifier:

➤ Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes' theorem:

$$P(\text{Yes} | \text{Sunny}) = P(\text{Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny} | \text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes} | \text{Sunny}) = 0.3 * 0.71 / 0.35 = 0.60$$



Naïve Bayes Classifier Algorithm

Working of Naïve Bayes' Classifier:

$$P(\text{No} \mid \text{Sunny}) = P(\text{Sunny} \mid \text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny} \mid \text{NO}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No} \mid \text{Sunny}) = 0.5 * 0.29 / 0.35 = 0.41$$

- So as we can see from the above calculation that $P(\text{Yes} \mid \text{Sunny}) > P(\text{No} \mid \text{Sunny})$
- **Hence on a Sunny day, Player can play the game.**



Naïve Bayes Classifier Algorithm

Advantages of Naïve Bayes Classifier:

1. Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
2. It can be used for Binary as well as Multi-class Classifications.
3. It performs well in Multi-class predictions as compared to the other Algorithms.
4. It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

1. Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.



Naïve Bayes Classifier Algorithm

Applications of Naïve Bayes Classifier:

1. It is used for **Credit Scoring**.
2. It is used in **medical data classification**.
3. It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
4. It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Naïve Bayes Classifier Algorithm



Types of Naïve Bayes Model:

➤ There are **three types** of Naive Bayes Model. Those are:

- 1. Gaussian**
- 2. Multinomial**
- 3. Bernoulli**

Gaussian:

The Gaussian model assumes that features follow a normal distribution. This means if **predictors take continuous values instead of discrete**, then the model assumes that these values are sampled from the Gaussian distribution.

Multinomial:

The Multinomial Naïve Bayes classifier is used **when the data is multinomial distributed**. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.



Naïve Bayes Classifier Algorithm

Types of Naïve Bayes Model:

Bernoulli:

The Bernoulli classifier works similar to the Multinomial classifier, but the **predictor variables are the independent Booleans variables**. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.



Naïve Bayes Classifier Algorithm

Python Implementation of the Naïve Bayes algorithm:

Steps to implement:

1. Data Pre-processing step
2. Fitting Naive Bayes to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



Naïve Bayes Classifier Algorithm

1) Data Pre-processing step:

➤ In this step, we will pre-process/prepare the data so that we can use it efficiently in our code.

```
#importing the libraries
```

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
# Importing the dataset
```

```
dataset = pd.read_csv('user_data.csv')
```

```
x = dataset.iloc[:, [2, 3]].values
```

```
y = dataset.iloc[:, 4].values
```



Naïve Bayes Classifier Algorithm

1) Data Pre-processing step:

```
# Splitting the dataset into the Training set 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)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_train = sc.fit_transform(x_train)
```

```
x_test = sc.transform(x_test)
```



Naïve Bayes Classifier Algorithm

1) Data Pre-processing step:

➤ The output for the dataset is given as

Index	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
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1



Naïve Bayes Classifier Algorithm

2) Fitting Naive Bayes to the Training Set:

➤ After the pre-processing step, now we will fit the Naive Bayes model to the Training set.

```
# Fitting Naive Bayes to the Training set  
from sklearn.naive_bayes import GaussianNB  
classifier = GaussianNB()  
classifier.fit(x_train, y_train)
```

Output:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

➤ In the above code, we have used the **GaussianNB classifier** to fit it to the training dataset. We can also use other classifiers as per our requirement.



Naïve Bayes Classifier Algorithm

3) Prediction of the test set result:

- Now we will predict the test set result. For this, we will create a new predictor variable **y_pred**, and will use the predict function to make the predictions.

```
# Predicting the Test set results  
y_pred = classifier.predict(x_test)
```

Output:

- The output shows the result for prediction vector **y_pred** and real vector **y_test**.
- We can see that some predictions are different from the real values, which are the incorrect predictions.

y_pred - NumPy array		y_test - NumPy array	
0	0	0	0
1	0	1	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	1	1	0
8	0	0	0
9	1	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0



Naïve Bayes Classifier Algorithm

4) Creating Confusion Matrix:

- Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix.

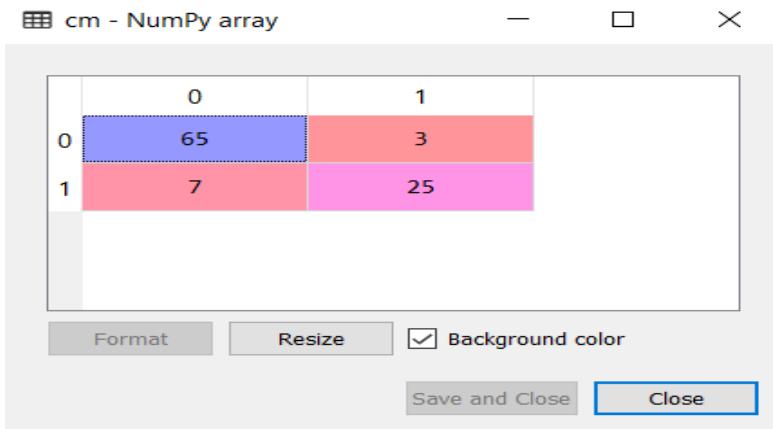
```
# Making the Confusion Matrix
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred)
```

Output:

- in the output, there are $7+3= 10$ incorrect predictions, and $65+25=90$ correct predictions.





Naïve Bayes Classifier Algorithm

5) Visualizing the training set result:

➤ we will visualize the training set result using Naïve Bayes Classifier.

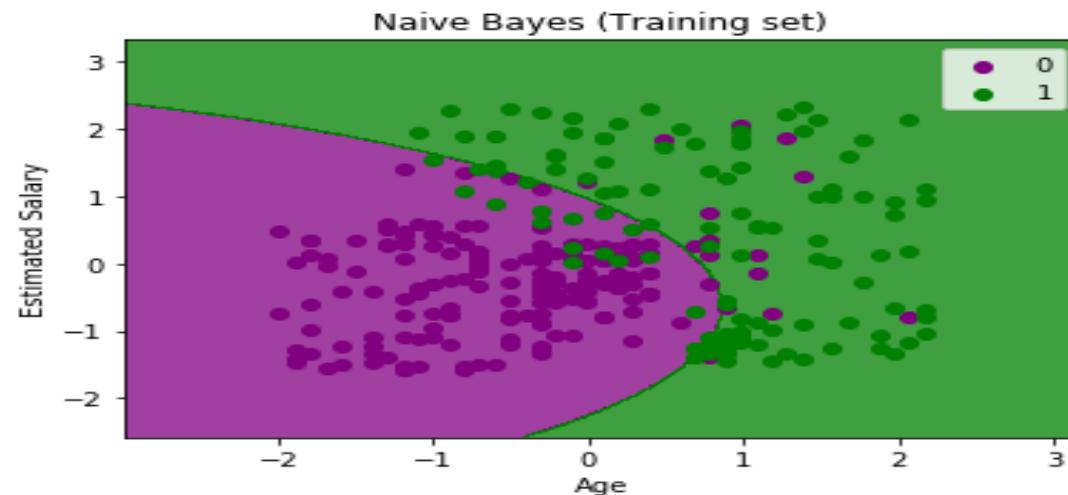
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                      nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
              alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



Naïve Bayes Classifier Algorithm

5) Visualizing the training set result:

Output:



- In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used **GaussianNB** classifier in our code.



Naïve Bayes Classifier Algorithm

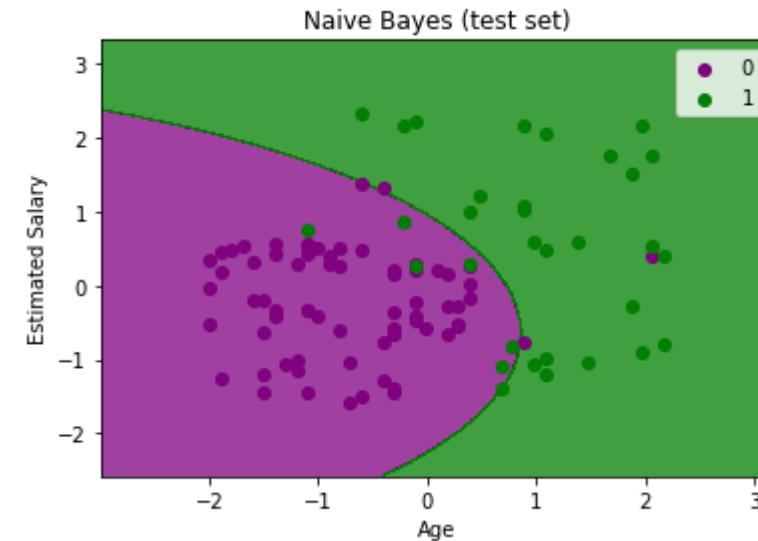
6) Visualizing the Test set result:

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                      nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
              alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Naïve Bayes Classifier Algorithm



6) Visualizing the Test set result: Output:



- The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables.
- There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.



Decision Tree Classification Algorithm

Decision Tree Classification Algorithm

➤ 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 is the **hierarchical tree structured algorithm** that is used for derived a meaningful output from a variety of inputs.

The output fetched from this kind of hierarchical arrangement is considered a valuable contribution for producing analytical results for essential business decision-making.

➤ 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 Classification Algorithm

Decision Tree Classification Algorithm

- 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 a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***

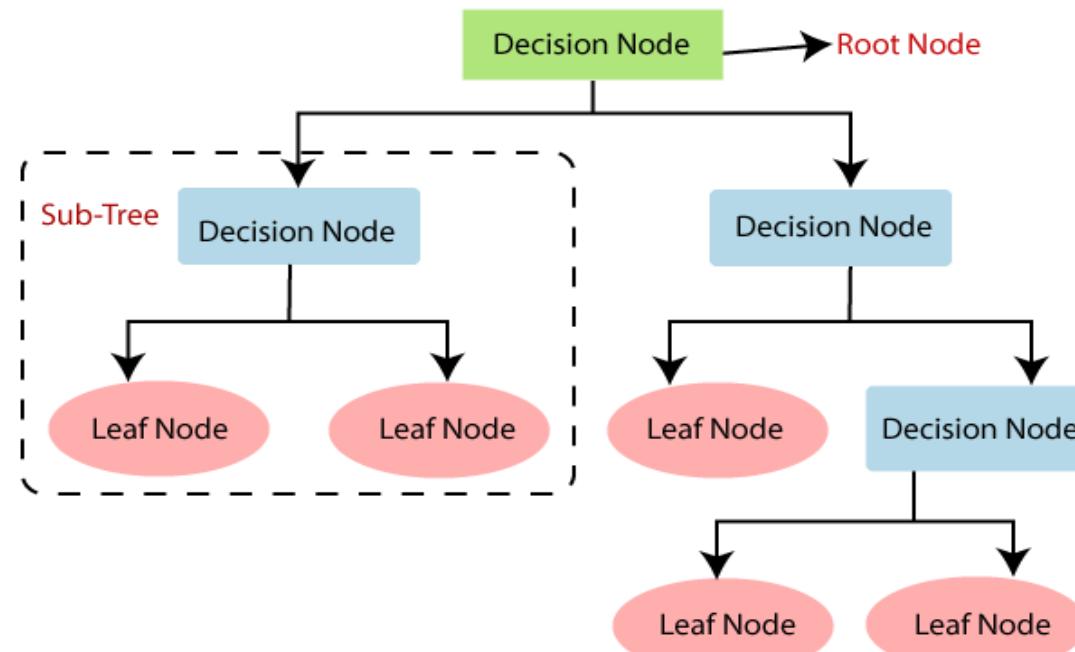
It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.



Decision Tree Classification Algorithm

Decision Tree Classification 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:



Decision Tree Classification Algorithm



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 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.

Decision Tree Classification Algorithm



Decision Tree Terminologies

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.

Decision Tree Classification Algorithm



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.

Decision Tree Classification Algorithm



How does the Decision Tree algorithm Work?

➤ 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 reached where you cannot further classify the nodes and called the final node as a leaf node.

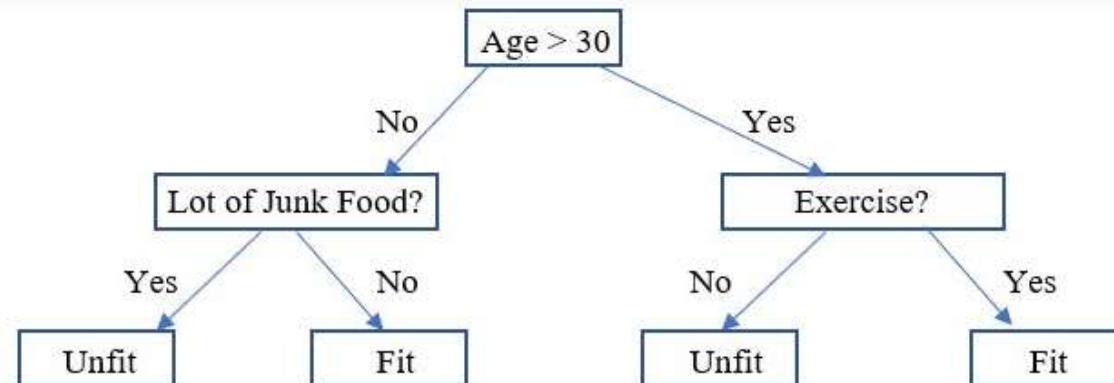


Decision Tree Classification Algorithm

Example:

Age	A lot of Junk food	Exercise	Fit/Unfit
45	No	No	Unfit
40	No	Yes	Fit
25	Yes	No	Unfit
29	No	Yes	Fit
23	Yes	Yes	Unfit

➤ decision tree for the table mentioned with data



Decision Tree Classification Algorithm



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:

1. **Information Gain**
2. **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.

Decision Tree Classification Algorithm



Attribute Selection Measures

1. Information Gain:

- 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)]

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)= $-P(\text{yes})\log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$

Where, S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no

Decision Tree Classification Algorithm



Attribute Selection Measures

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:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Decision Tree Classification Algorithm



Pruning: Getting an Optimal Decision tree

- *Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning.
- There are mainly **two** types of tree **pruning** technology used:
 1. **Cost Complexity Pruning**
 2. **Reduced Error Pruning.**

Decision Tree Classification Algorithm



Advantages of the Decision Tree

1. It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
2. It can be very useful for solving decision-related problems.
3. It helps to think about all the possible outcomes for a problem.
4. There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

1. The decision tree contains lots of layers, which makes it complex.
2. It may have an overfitting issue, which can be resolved using the **Random Forest algorithm**.
3. For more class labels, the computational complexity of the decision tree may increase.



Decision Tree Classification Algorithm

- Python implementation of the Decision Tree Classification Algorithm
- Python's **Scikit-Learn library** can be used to implement the **Decision Tree Classification Algorithm**

Example:

- we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**" . By using the same dataset, we can compare the Decision tree classifier with other classification models such as [**KNN**](#) [**SVM**](#), [**LogisticRegression**](#), etc.

Decision Tree Classification Algorithm



Python implementation of the Decision Tree Classification Algorithm

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

Decision Tree Classification Algorithm



Steps to implement the Decision Tree Classification Algorithm

1. Data Pre-processing step
2. Fitting the Decision Tree Classification Algorithm to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



Decision Tree Classification Algorithm

1. Data Pre-Processing Step:

➤ In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv(  
    'user_data.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Decision Tree Classification Algorithm



Data Pre-Processing Step:

➤ The next step is to **split our dataset into its attributes and labels**. i.e, extract the **dependent and independent variables** from the given dataset.

#Extracting Independent and dependent Variable

```
x= data_set.iloc[:, [2,3]].values
```

```
y= data_set.iloc[:, 4].values
```

➤ In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4.

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1

Decision Tree Classification Algorithm



Data Pre-Processing Step:

Train Test Split

➤ To avoid **over-fitting**, we will **divide our dataset into training and test splits**, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application.

➤ To create training and test splits, execute the following script:

```
# 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_st  
ate=0)
```

➤ The above script **splits the dataset into 75% train data and 25% test data**.



Decision Tree Classification Algorithm

Data Pre-Processing Step:

Train Test Split

For **test** set:

x_test - NumPy array		y_test - NumPy array	
0	0	1	
0	-0.804802	0.504964	0
1	-0.0125441	-0.567782	0
2	-0.309641	0.157046	0
3	-0.804802	0.273019	0
4	-0.309641	-0.567782	0
5	-1.1019	-1.43758	0
6	-0.70577	-1.58254	0
7	-0.210609	2.15757	1
8	-1.99319	-0.0459058	0
9	0.878746	-0.770734	0
10	-0.804802	-0.596776	0
11	-1.00287	-0.422817	0



Decision Tree Classification Algorithm

Data Pre-Processing Step:

Train Test Split

For training set:

	x_test - NumPy array			y_test - NumPy array	
	0	1			
0	-0.804802	0.504964		0	0
1	-0.0125441	-0.567782		1	0
2	-0.309641	0.157046		2	0
3	-0.804802	0.273019		3	0
4	-0.309641	-0.567782		4	0
5	-1.1019	-1.43758		5	0
6	-0.70577	-1.58254		6	0
7	-0.210609	2.15757		7	1
8	-1.99319	-0.0459058		8	0
9	0.878746	-0.770734		9	0
10	-0.804802	-0.596776		10	0
11	-1.00287	-0.422817		11	0

Decision Tree Classification Algorithm



Data Pre-Processing Step:

Feature Scaling

➤ Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

#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)
```

➤ From the output image, we can see that our data is successfully scaled.

x_test - NumPy array		y_test - NumPy array	
0	1	0	0
-0.804802	0.504964	1	0
-0.0125441	-0.567782	0	0
-0.309641	0.157046	0	0
-0.804802	0.273019	0	0
-0.309641	-0.567782	0	0
-1.1019	-1.43758	0	0
-0.70577	-1.58254	0	0
-0.210609	2.15757	1	0
-1.99319	-0.0459058	0	0
0.878746	-0.770734	0	0
-0.804802	-0.596776	0	0
-1.00287	-0.422817	0	0
-0.111576	-0.422817	0	0



Decision Tree Classification Algorithm

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.

#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)
```

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.



Decision Tree Classification Algorithm

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

Output:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy',
max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
random_state=0, splitter='best')
```

Decision Tree Classification Algorithm



3. Predicting the Test Result:

To predict the test set result, we will create a `y_pred` vector .

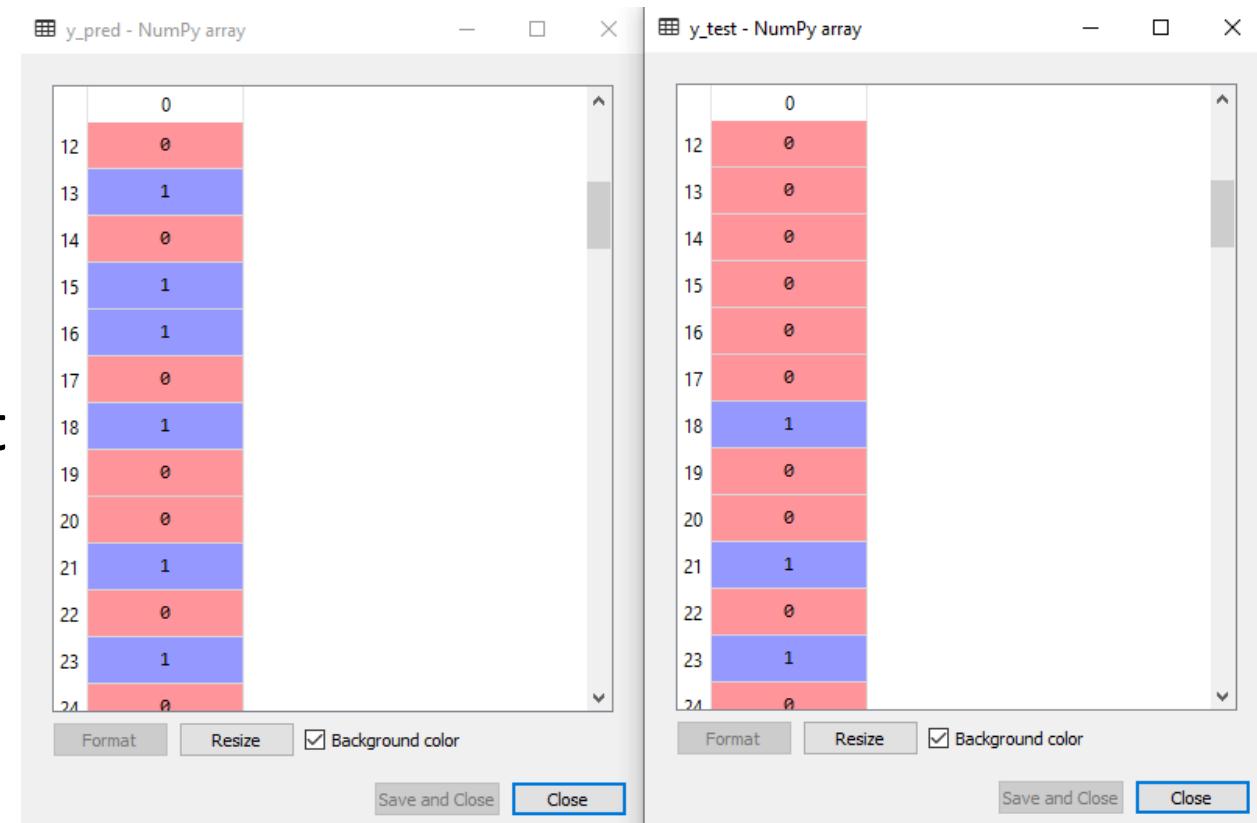
```
#Predicting the test set result
```

```
y_pred= classifier.predict(x_test)
```

Output:

there are some values in the prediction vector, which are different from the real vector values.

These are prediction errors.



12	0
13	1
14	0
15	1
16	1
17	0
18	1
19	0
20	0
21	1
22	0
23	1
24	0

12	0
13	0
14	0
15	0
16	0
17	0
18	1
19	0
20	0
21	1
22	0
23	1
24	0

Decision Tree Classification Algorithm



4. Test Accuracy:

- to know the number of correct and incorrect predictions, we need to use the confusion matrix

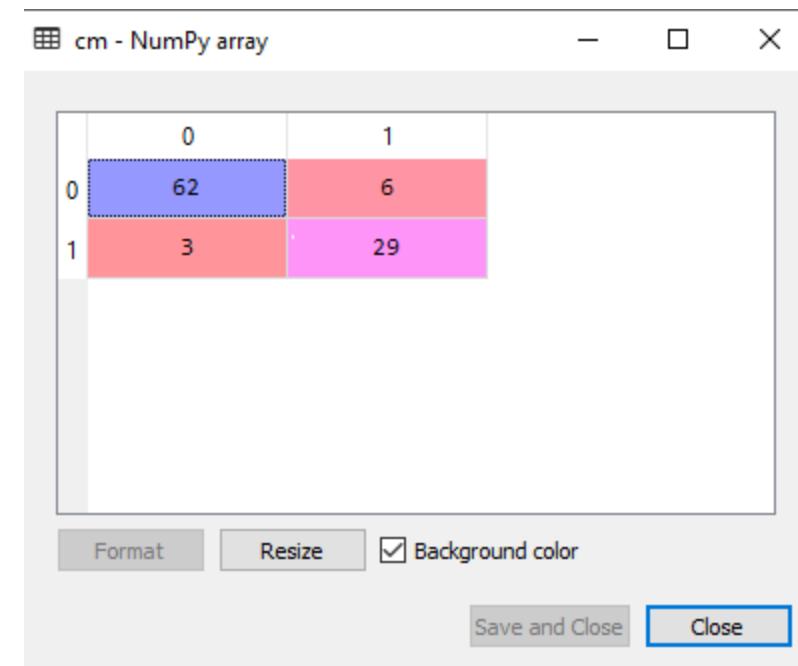
#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix  
cm=confusion_matrix(y_test, y_pred)
```

Output:

we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and **62+29=91 correct predictions.**

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





Decision Tree Classification Algorithm

5. Visualizing the Training set result:

- we will visualize the training set result.
- To visualize the training set result we will plot a graph for the decision tree classifier.
- The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car



Decision Tree Classification Algorithm

5. Visualizing the Training set result:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(['purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(['purple', 'green'])(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



Decision Tree Classification Algorithm

5. Visualizing the Training set result:

Output:

- The output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.
- we can see, the tree is trying to capture each dataset, which is the case of overfitting.



Decision Tree Classification Algorithm



Visualizing the Test set result:

➤ After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x_train** and **y_train** will be replaced by **x_test** and **y_test**.

Decision Tree Classification Algorithm



Visualizing the Test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
Output:
```



Decision Tree Classification Algorithm



- we can see in the image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.



Bias and Variance

Bias and Variance

- For the same set of data, different algorithms behave differently. For example, if we want to predict the price of houses given for some dataset, some of the algorithms that can be used are Linear Regression and Decision Tree Regressor.
- Both of these algorithms will interpret the dataset in different ways and thus make different predictions. One of the key distinctions is how much bias and variance they produce.
- There are 3 types of prediction error:
 1. Bias
 2. variance and
 3. irreducible error.
- Irreducible error, also known as “noise,” can’t be reduced by the choice of algorithm.



Bias and Variance

Bias

bias is an **error from erroneous assumptions in the learning algorithm**. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).

- Models with **high bias are less flexible** and are not fully able to learn from the training data.
- Generally, models with **low bias are preferred**.

Variance

variance is an **error from sensitivity to small fluctuations in the training set**. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

- Variance defines the **deviation in prediction when switching from one dataset to another**
- We want a model with **low variance**



Bagging and Boosting

Bagging and Boosting

➤ Bagging and Boosting are both **ensemble methods** in Machine Learning.

What is an Ensemble Method?

➤ The ensemble is a method used in the machine learning algorithm.

Ensemble methods combine different decision trees to deliver better predictive results, afterward utilizing a single decision tree. The primary principle behind the ensemble model is that a group of weak learners come together to form an active learner.

Or

Ensemble learning is a machine learning paradigm where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.



Bagging and Boosting

Bagging and Boosting

➤ two techniques are used to perform ensemble decision tree.

1. Bagging
2. Boosting

Bagging

- Bagging is used when our objective is to reduce the variance of a decision tree.
- Here the concept is to create a few subsets of data from the training sample, which is chosen randomly with replacement.
- Now each collection of subset data is used to prepare their decision trees thus, we end up with an ensemble of various models. The average of all the assumptions from numerous trees is used, which is more powerful than a single decision tree.



Bagging and Boosting

Bagging and Boosting

- **Random Forest** is an **expansion over bagging**. It takes one additional step to predict a random subset of data. It also makes the random selection of features rather than using all features to develop trees. When we have numerous random trees, it is called the Random Forest.
- These are the following steps which are taken to implement a Random forest:
 1. Let us consider **X** observations **Y** features in the training data set. First, a model from the training data set is taken randomly with substitution.
 2. The tree is developed to the largest.
 3. The given steps are repeated, and prediction is given, which is based on the collection of predictions from n number of trees.



Bagging and Boosting

Bagging and Boosting

Advantages of using Random Forest technique:

- It manages a higher dimension data set very well.
- It manages missing quantities and keeps accuracy for missing data.

Disadvantages of using Random Forest technique:

- Since the last prediction depends on the mean predictions from subset trees, it won't give precise value for the regression model.



Bagging and Boosting

Boosting:

- Boosting is another ensemble procedure to make a collection of predictors. In other words, we fit consecutive trees, usually random samples, and at each step, the objective is to solve net error from the prior trees.
- If a given input is misclassified by theory, then its weight is increased so that the upcoming hypothesis is more likely to classify it correctly by consolidating the entire set at last converts weak learners into better performing models.
- **Gradient Boosting** is an expansion of the boosting procedure.

$$\text{Gradient Boosting} = \text{Gradient Descent} + \text{Boosting}$$

- It utilizes a gradient descent algorithm that can optimize any differentiable loss function. An ensemble of trees is constructed individually, and individual trees are summed successively. The next tree tries to restore the loss (It is the difference between actual and predicted values).



Bagging and Boosting

Boosting:

Advantages of using Gradient Boosting methods:

- It supports different loss functions.
- It works well with interactions.

Disadvantages of using a Gradient Boosting methods:

- It requires cautious tuning of different hyper-parameters.



Bagging and Boosting

Difference between Bagging and Boosting:

Bagging	Boosting
Various training data subsets are randomly drawn with replacement from the whole training dataset.	Each new subset contains the components that were misclassified by previous models.
Bagging attempts to tackle the over-fitting issue.	Boosting tries to reduce bias.
If the classifier is unstable (high variance), then we need to apply bagging.	If the classifier is steady and straightforward (high bias), then we need to apply boosting.
Every model receives an equal weight.	Models are weighted by their performance.
Objective to decrease variance, not bias.	Objective to decrease bias, not variance.
It is the easiest way of connecting predictions that belong to the same type.	It is a way of connecting predictions that belong to the different types.
Every model is constructed independently.	New models are affected by the performance of the previously developed model.

Random Forest Algorithm



Random Forest Algorithm

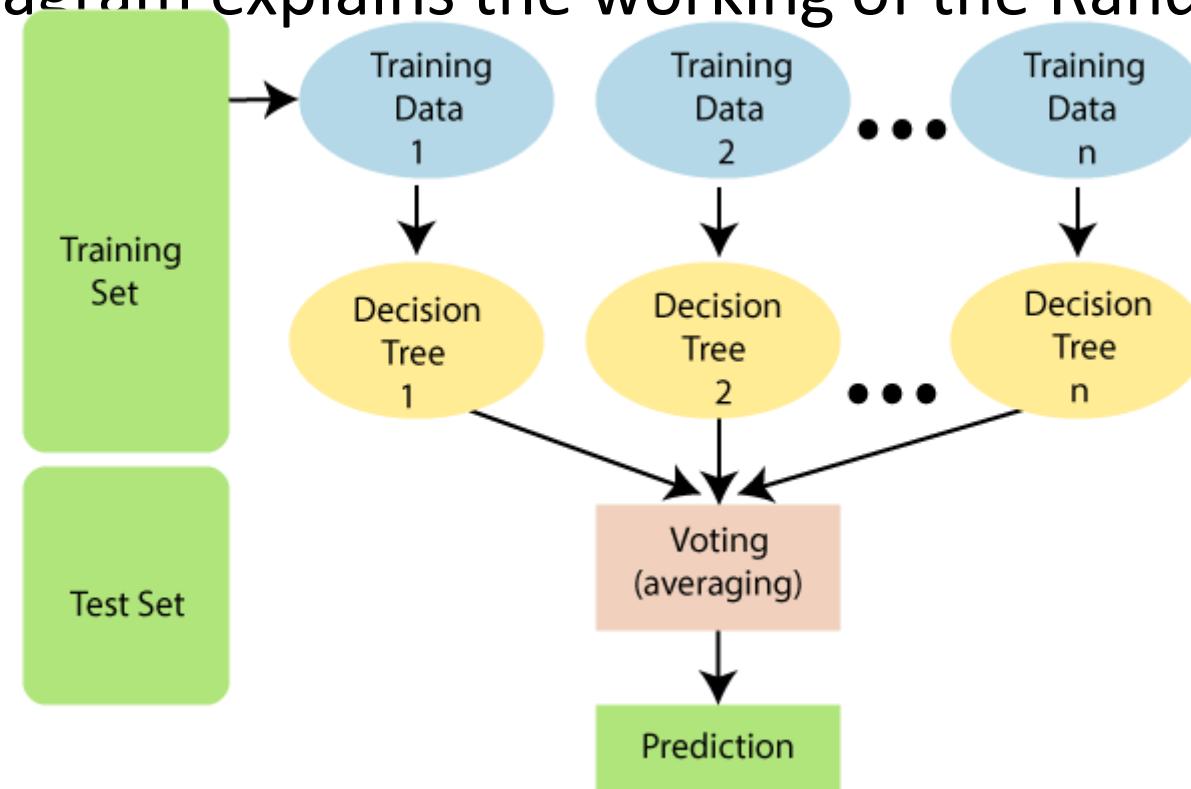
- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.
- It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a *process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- 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.



Random Forest Algorithm

Random Forest Algorithm

- 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 Algorithm

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output.

- Therefore, below are **two assumptions** for a better Random forest classifier:
- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Random Forest Algorithm



Why use Random Forest?

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

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



Random Forest Algorithm

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.

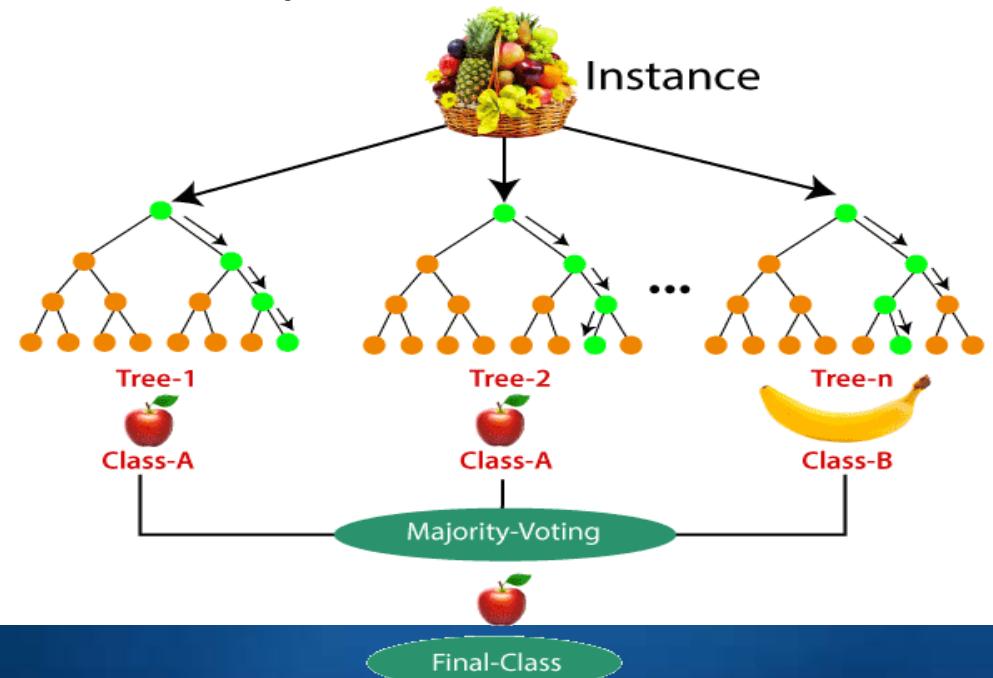
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.



Random Forest Algorithm

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:





Random Forest Algorithm

Applications of Random Forest

➤ There are mainly four sectors where Random forest mostly used:

Banking: Banking sector mostly uses this algorithm for the identification of loan risk.

Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.

Land Use: We can identify the areas of similar land use by this algorithm.

Marketing: Marketing trends can be identified using this algorithm.

Advantages of Random Forest

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



Random Forest Algorithm

Disadvantages of Random Forest

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



Random Forest Algorithm

Python implementation of the KNN algorithm

➤ Python's **Scikit-Learn library** can be used to implement the **Random Forest Algorithm**

Example:

➤ we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**" . By using the same dataset, we can compare the Random Forest Algorithm with other classification models such as [KNN](#) [SVM](#), [LogisticRegression](#), etc.



Random Forest Algorithm

Python implementation of the KNN algorithm

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



Random Forest Algorithm

Steps to implement the Random Forest Algorithm

1. Data Pre-processing step
2. Fitting the Random Forest Algorithm to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



Random Forest Algorithm

1. Data Pre-Processing Step:

➤ In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv(  
    'user_data.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Random Forest Algorithm



Data Pre-Processing Step:

➤ The next step is to **split our dataset into its attributes and labels**. i.e, extract the **dependent and independent variables** from the given dataset.

#Extracting Independent and dependent Variable

```
x= data_set.iloc[:, [2,3]].values
```

```
y= data_set.iloc[:, 4].values
```

➤ In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4.

x - NumPy array		
	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000

y - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1

Random Forest Algorithm



Data Pre-Processing Step:

Train Test Split

➤ To avoid **over-fitting**, we will **divide our dataset into training and test splits**, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application.

➤ To create training and test splits, execute the following script:

```
# 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_st  
ate=0)
```

➤ The above script **splits the dataset into 75% train data and 25% test data**.



Random Forest Algorithm

Data Pre-Processing Step:

Train Test Split

For **test** set:

	0	1	
0	-0.804802	0.504964	
1	-0.0125441	-0.567782	
2	-0.309641	0.157046	
3	-0.804802	0.273019	
4	-0.309641	-0.567782	
5	-1.1019	-1.43758	
6	-0.70577	-1.58254	
7	-0.210609	2.15757	
8	-1.99319	-0.0459058	
9	0.878746	-0.770734	
10	-0.804802	-0.596776	
11	-1.00287	-0.422817	

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0



Random Forest Algorithm

Data Pre-Processing Step:

Train Test Split

For training set:

	x_test - NumPy array			y_test - NumPy array	
	0	1			
0	-0.804802	0.504964		0	0
1	-0.0125441	-0.567782		1	0
2	-0.309641	0.157046		2	0
3	-0.804802	0.273019		3	0
4	-0.309641	-0.567782		4	0
5	-1.1019	-1.43758		5	0
6	-0.70577	-1.58254		6	0
7	-0.210609	2.15757		7	1
8	-1.99319	-0.0459058		8	0
9	0.878746	-0.770734		9	0
10	-0.804802	-0.596776		10	0
11	-1.00287	-0.422817		11	0

Random Forest Algorithm



Data Pre-Processing Step:

Feature Scaling

➤ Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

#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)
```

➤ From the output image, we can see that our data is successfully scaled.

x_test - NumPy array		y_test - NumPy array	
0	1	0	0
-0.804802	0.504964	1	0
-0.0125441	-0.567782	0	0
-0.309641	0.157046	0	0
-0.804802	0.273019	0	0
-0.309641	-0.567782	0	0
-1.1019	-1.43758	0	0
-0.70577	-1.58254	0	0
-0.210609	2.15757	1	0
-1.99319	-0.0459058	0	0
0.878746	-0.770734	0	0
-0.804802	-0.596776	0	0
-1.00287	-0.422817	0	0
-0.111576	-0.422817	0	0



Random Forest Algorithm

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

```
#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)
```

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.



Random Forest Algorithm

2. Fitting a RandomForest algorithm to the Training set:

Output:

```
RandomForestClassifier(bootstrap=True, class_weight=None,  
criterion='entropy', max_depth=None, max_features='auto',  
max_leaf_nodes=None, min_impurity_decrease=0.0,  
min_impurity_split=None, min_samples_leaf=1,  
min_samples_split=2, min_weight_fraction_leaf=0.0,  
n_estimators=10, n_jobs=None, oob_score=False,  
random_state=None, verbose=0, warm_start=False)
```

Random Forest Algorithm



3. Predicting the Test Result:

To predict the test set result, we will create a `y_pred` vector .

```
#Predicting the test set result
```

```
y_pred= classifier.predict(x_test)
```

Output:

By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0



Random Forest Algorithm

4. Test Accuracy:

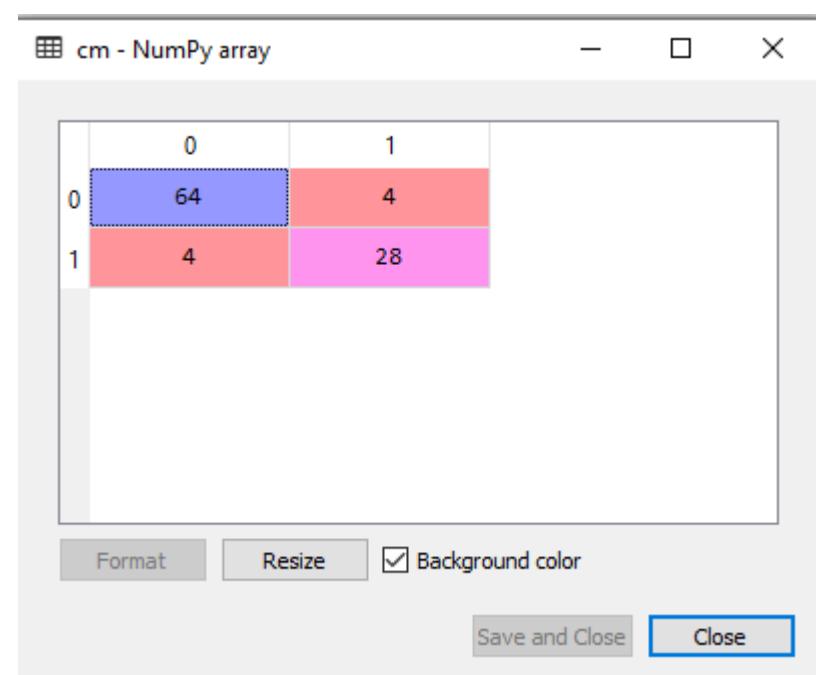
- to know the number of correct and incorrect predictions, we need to use the confusion matrix

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix(y_test, y_pred)
```

Output:

As we can see in the above matrix, there are
4+4= 8 incorrect predictions and
64+28= 92 correct predictions.





Random Forest Algorithm

5. Visualizing the Training set result:

- we will visualize the training set result.
- To visualize the training set result we will plot a graph for the decision tree classifier.
- The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car



Random Forest Algorithm

5. Visualizing the Training set result:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(['purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(['purple', 'green'])(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

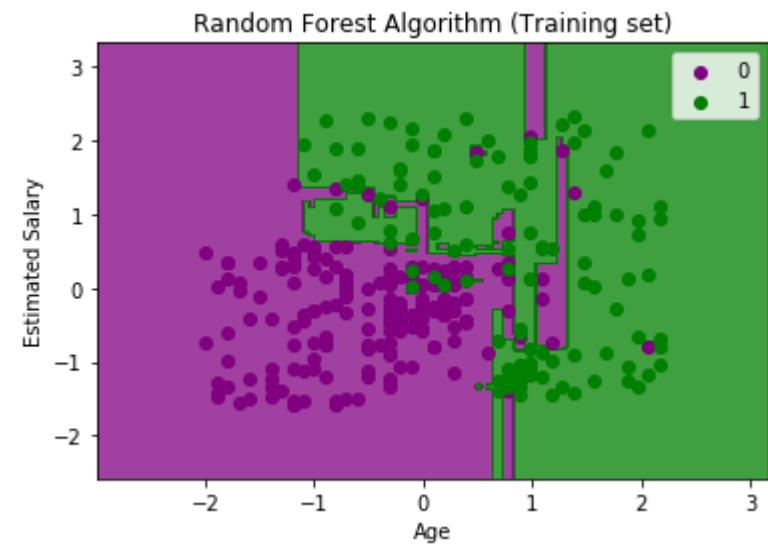


Random Forest Algorithm

5. Visualizing the Training set result:

Output:

Each data point corresponds to each user of the user_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV. So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.





Random Forest Algorithm

Visualizing the Test set result:

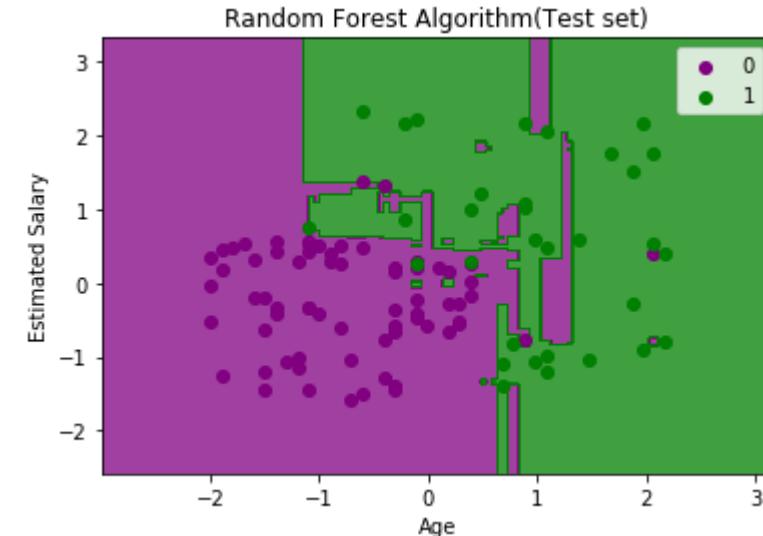
➤ After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x_train** and **y_train** will be replaced by **x_test** and **y_test**.



Random Forest Algorithm

Visualizing the Test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(['purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(['purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



Random Forest Algorithm



- The image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

Regression Analysis



Regression Analysis in Machine learning

- Regression analysis is a **statistical method** to model the relationship between a **dependent (target)** and **independent (predictor)** variables with one or more independent variables.
- Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed.
- It predicts continuous/real values such as **temperature, age, salary, price**, etc.

Regression Analysis



Regression Analysis in Machine learning

Example:

- Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:
- Now, the company wants to do the advertisement of **\$200 in the year 2019 and wants to know the prediction about the sales for this year.**
So to solve such type of prediction problems in machine learning, we need regression analysis

Advertisement	Sales
\$90	\$1000
\$120	\$1300
\$150	\$1800
\$100	\$1200
\$130	\$1380
\$200	??

Regression Analysis



Regression Analysis in Machine learning

- Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables.
- It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.**
- In Regression, we plot a graph between the variables which best fits the given data points.
- using this plot, the machine learning model can make predictions about the data.

Regression Analysis



Regression Analysis in Machine learning

- In simple words, "*Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum.*"
- The distance between data points and line tells whether a model has captured a strong relationship or not.
- Some examples of regression can be as:
 1. Prediction of rain using temperature and other factors
 2. Determining Market trends
 3. Prediction of road accidents due to rash driving.

Regression Analysis



Terminologies Related to the Regression Analysis:

1. Dependent Variable
2. Independent Variable
3. Outliers
4. Multicollinearity
5. Underfitting and Overfitting

Dependent Variable:

The **main factor** in Regression analysis which we **want to predict or understand** is called the **dependent variable**. It is also called **target variable**.

Independent Variable:

The factors which affect the dependent variables or which are **used to predict the values of the dependent variables** are called independent variable, also called as a **predictor**.

Regression Analysis



Terminologies Related to the Regression Analysis:

Outliers:

Outlier is an observation which contains either **very low value or very high value in comparison to other observed values**. An outlier may hamper the result, so it should be avoided.

Multicollinearity:

If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.

Underfitting and Overfitting:

If algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

Regression Analysis



Why do we use Regression Analysis?

- Regression analysis helps in the **prediction of a continuous variable**.
- There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science.
- Below are some other reasons for using Regression analysis:
 1. Regression estimates the relationship between the target and the independent variable.
 2. It is used to find the trends in data.
 3. It helps to predict real/continuous values.
 4. By performing the regression, we can confidently determine the **most important factor, the least important factor, and how each factor is affecting the other factors**.

Regression Analysis



Types of Regression

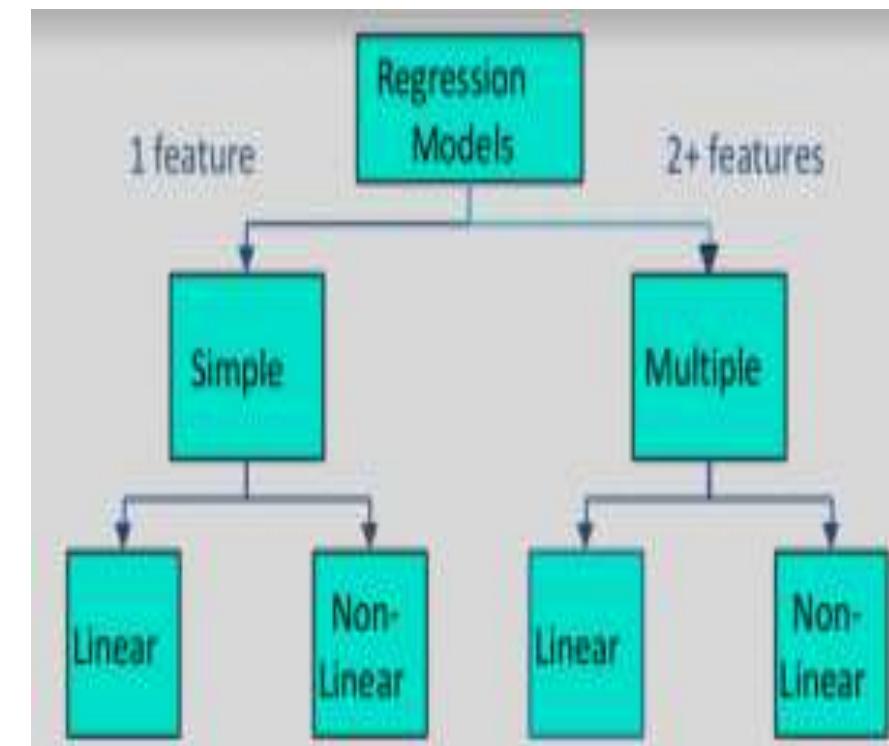
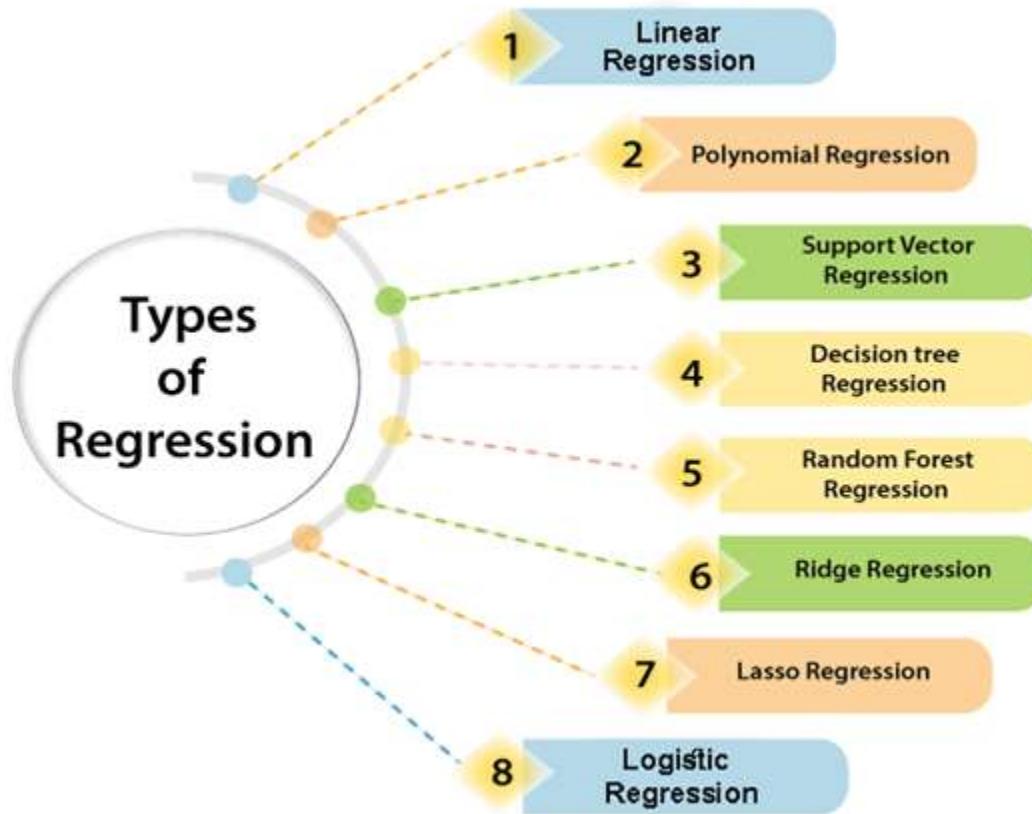
- There are various types of regressions which are used in data science and machine learning.
- Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables.
- some important types of regression are :

1. **Linear Regression**
2. **Logistic Regression**
3. **Polynomial Regression**
4. **Support Vector Regression**
5. **Decision Tree Regression**
6. **Random Forest Regression**
7. **Ridge Regression**
8. **Lasso Regression**

Regression Analysis



Types of Regression



Regression Analysis



Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.

Regression Analysis



Linear Regression:

- The relationship between variables in the linear regression model can be explained using the below image.
- Here we are predicting the salary of an employee on the basis of **the year of experience**.

➤ the mathematical equation for Linear regression

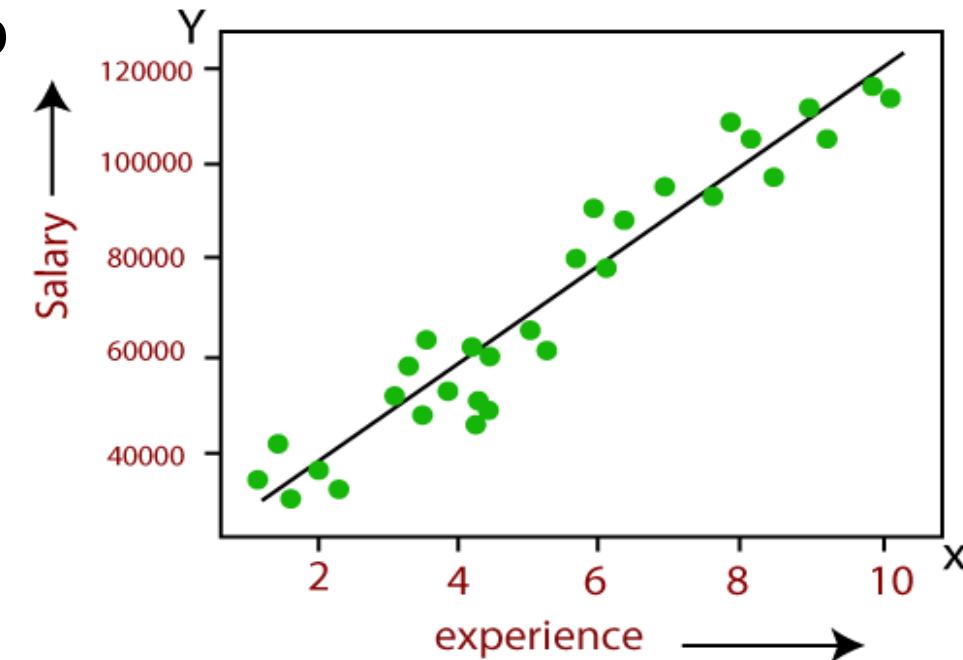
$$Y = aX + b$$

Here,

Y = dependent variables (target variables),

X= Independent variables (predictor variables),

a and b are the linear coefficients



Regression Analysis



- Some popular applications of linear regression are:
 1. **Analyzing trends and sales estimates**
 2. **Salary forecasting**
 3. **Real estate prediction**

Regression Analysis



Logistic Regression:

- Logistic regression is another supervised learning algorithm **which is used to solve the classification problems**. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the **concept of probability**.
- Logistic regression uses **sigmoid function** or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression.

Regression Analysis



Logistic Regression:

➤ The function can be represented as:

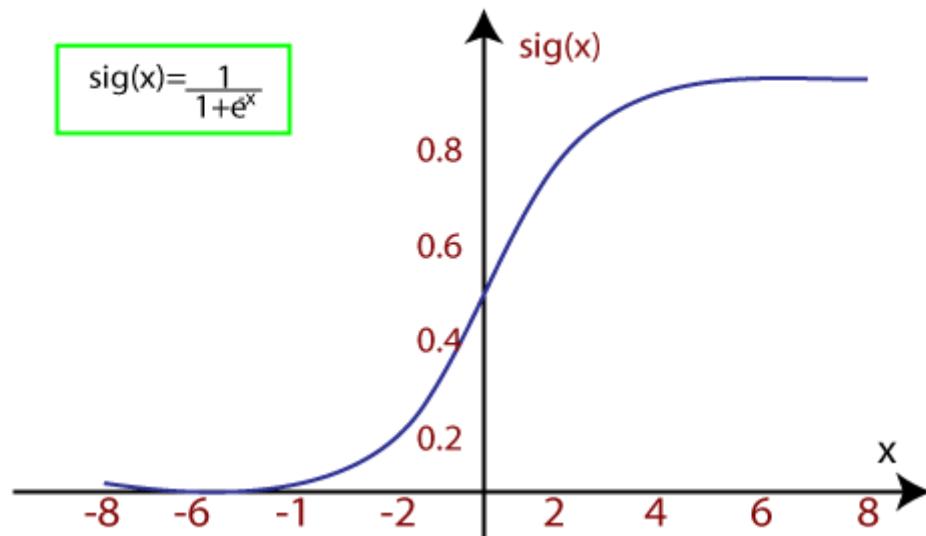
$$f(x) = \frac{1}{1+e^{-x}}$$

$f(x)$ = Output between the 0 and 1 value.

x = input to the function

e = base of natural logarithm.

When we provide the input values (data) to the function, it gives the S-curve as follows:



Regression Analysis



Logistic Regression:

- It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.
- There are three types of logistic regression:
 1. **Binary(0/1, pass/fail)**
 2. **Multi(cats, dogs, lions)**
 3. **Ordinal(low, medium, high)**

Regression Analysis



Polynomial Regression:

- Polynomial Regression is a type of regression which **models the non-linear dataset using a linear model.**
- It is similar to multiple linear regression, but it fits a non-linear curve between the value of x and corresponding conditional values of y.
- Suppose there is a dataset which consists of datapoints which are present in a non-linear fashion, so for such case, linear regression will not best fit to those datapoints. To cover such datapoints, we need Polynomial regression.
- **In Polynomial regression, the original features are transformed into polynomial features of given degree and then modeled using a linear model.** Which means the datapoints are best fitted using a polynomial line.

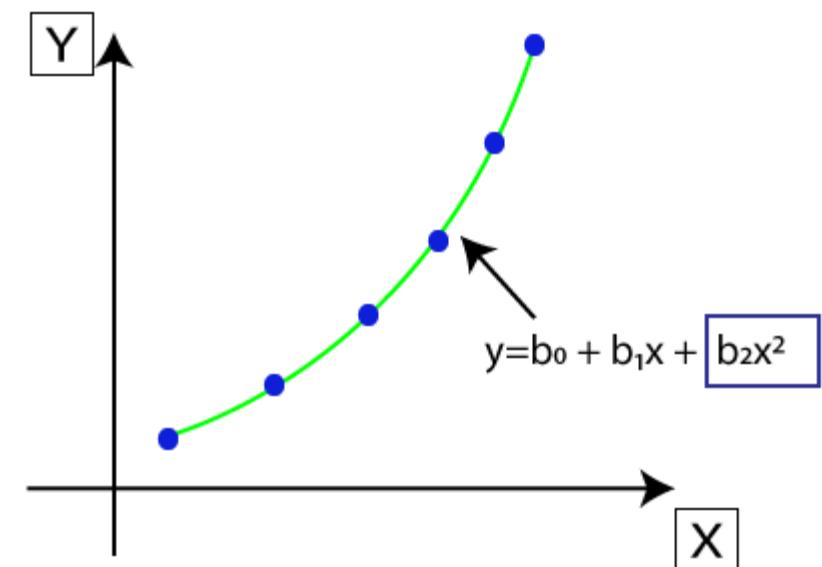
Regression Analysis



Polynomial Regression:

- The equation for polynomial regression also derived from linear regression equation that means Linear regression equation $Y = b_0 + b_1x$, is transformed into Polynomial regression equation $Y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$.
- Here Y is the **predicted/target output**, b_0, b_1, \dots, b_n are the **regression coefficients**. x is our **independent/input variable**.
- The model is still linear as the coefficients are still linear with quadratic.

Note: This is different from Multiple Linear regression in such a way that in Polynomial regression, a single element has different degrees instead of multiple variables with the same degree.



Regression Analysis



Support Vector Regression:

- Support Vector Machine is a supervised learning algorithm which can be used for regression as well as classification problems. So if we use it for regression problems, then it is termed as Support Vector Regression.
- Support Vector Regression is a regression algorithm which **works for continuous variables**.
- Below are some keywords which are used in **Support Vector Regression**:

Kernel:

It is a function used to map a lower-dimensional data into higher dimensional data.

Hyperplane:

In general SVM, it is a separation line between two classes, but in SVR, it is a line which helps to predict the continuous variables and cover most of the datapoints.

Regression Analysis



Support Vector Regression:

Boundary line:

Boundary lines are the two lines apart from hyperplane, which creates a margin for datapoints.

Support vectors:

Support vectors are the datapoints which are nearest to the hyperplane and opposite class.

Regression Analysis

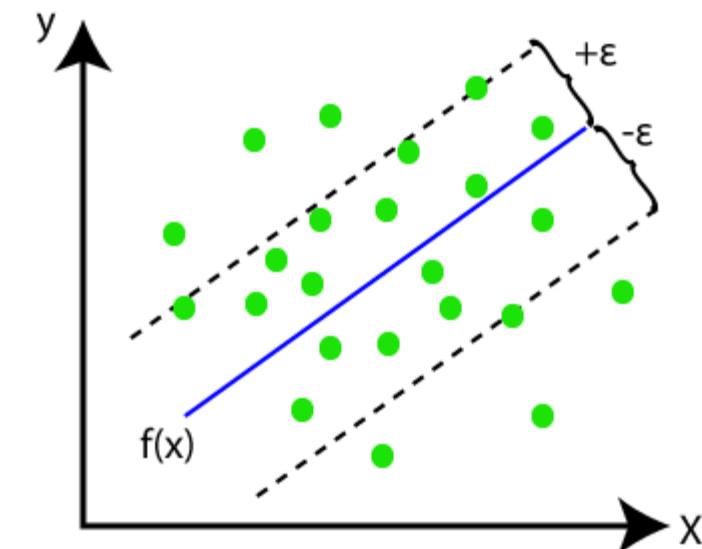


Support Vector Regression:

- In SVR, we always try to determine a hyperplane with a maximum margin, so that maximum number of datapoints are covered in that margin.
- ***The main goal of SVR is to consider the maximum datapoints within the boundary lines and the hyperplane (best-fit line) must contain a maximum number of datapoints.***

Consider the below image:

the blue line is called hyperplane, and the other two lines are known as boundary lines.



Regression Analysis



Decision Tree Regression:

- Decision Tree is a supervised learning algorithm which can be used for solving both classification and regression problems.
- It can solve problems for both categorical and numerical data
- Decision Tree regression builds a tree-like structure in which each internal node represents the "test" for an attribute, each branch represent the result of the test, and each leaf node represents the final decision or result.
- A decision tree is constructed starting from the root node/parent node (dataset), which splits into left and right child nodes (subsets of dataset). These child nodes are further divided into their children node, and themselves become the parent node of those nodes.

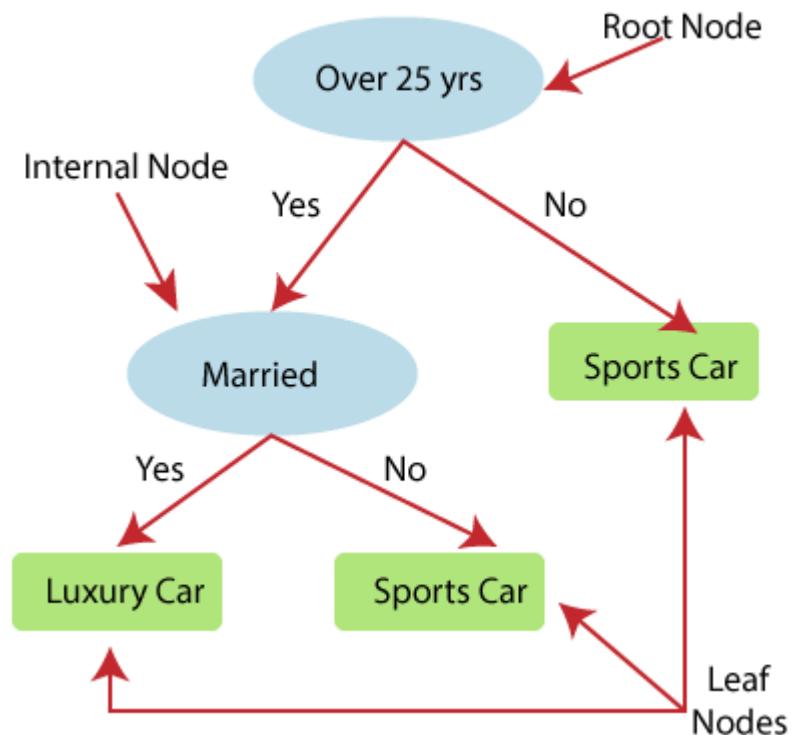
Regression Analysis



Decision Tree Regression:

➤ Consider the below image: image showing the example of Decision Tree regression, here, the model is trying to predict the choice of a person between Sports cars or Luxury car.

➤ Random forest is one of the most powerful supervised learning algorithms which is capable of performing regression as well as classification tasks.



Regression Analysis



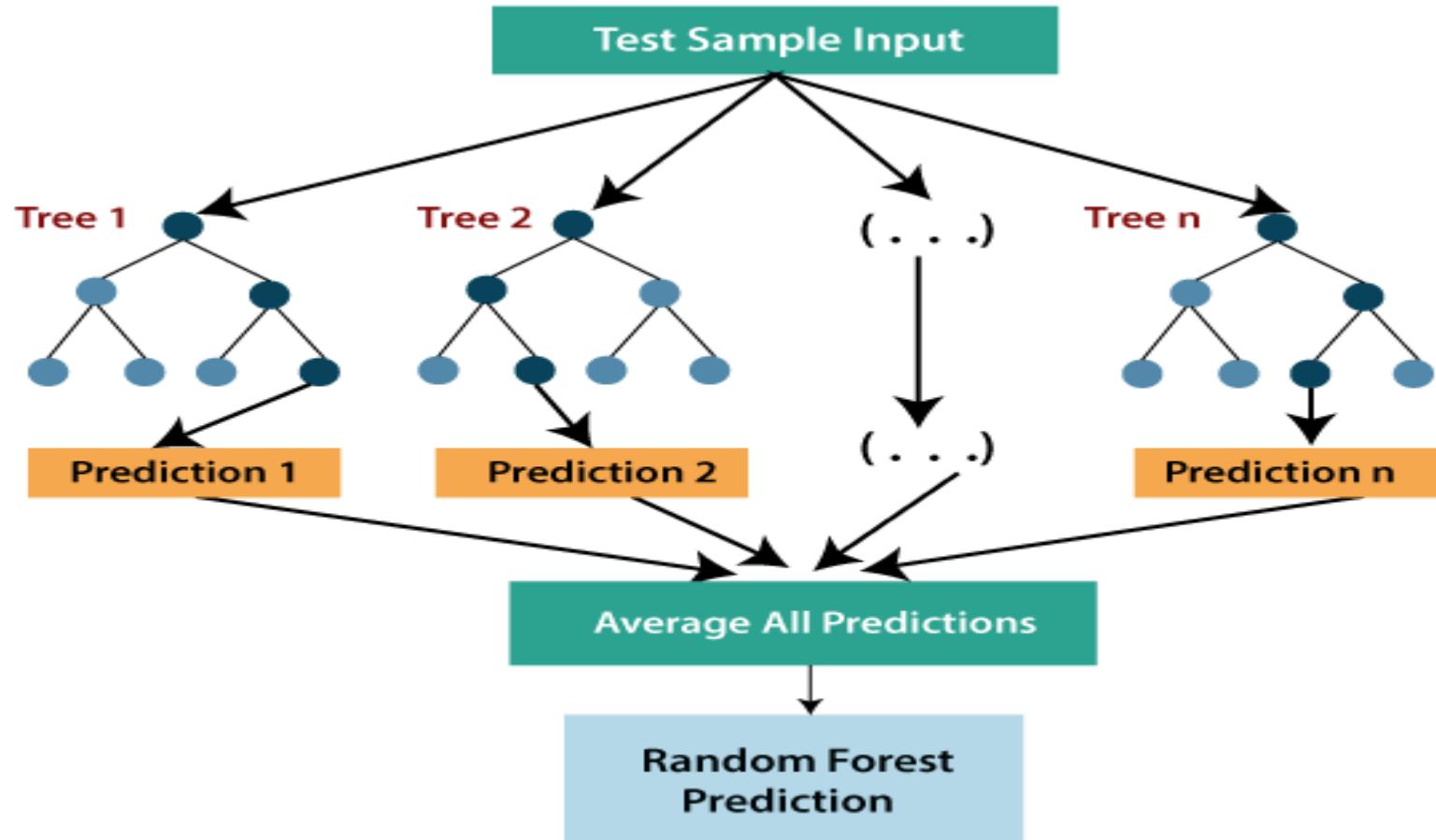
Decision Tree Regression:

- The Random Forest regression is an ensemble learning method which combines multiple decision trees and predicts the final output based on the average of each tree output. The combined decision trees are called as base models, and it can be represented more formally as:
- $g(x) = f_0(x) + f_1(x) + f_2(x) + \dots$ Random forest uses **Bagging or Bootstrap Aggregation** technique of ensemble learning in which aggregated decision tree runs in parallel and do not interact with each other.
- With the help of Random Forest regression, **we can prevent Overfitting in the model by creating random subsets of the dataset**

Regression Analysis



Decision Tree Regression:



Regression Analysis



Ridge Regression:

- Ridge regression is one of the most robust versions of linear regression in which a small amount of bias is introduced so that we can get better long term predictions.
- The amount of bias added to the model is known as **Ridge Regression penalty**.
- We can compute this penalty term by multiplying with the lambda to the squared weight of each individual features.
- The equation for ridge regression will be:

$$L(x, y) = \text{Min}(\sum_{i=1}^n (y_i - w_i x_i)^2 + \lambda \sum_{i=1}^n (w_i)^2)$$

- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.

Regression Analysis



Ridge Regression:

- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- It helps to solve the problems if we have more parameters than samples.

Lasso Regression:

- Lasso regression is another regularization technique to reduce the complexity of the model.
- It is similar to the Ridge Regression except that penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization**. The equation for Lasso regression will be:

$$L(x, y) = \text{Min}(\sum_{i=1}^n (y_i - w_i x_i)^2 + \lambda \sum_{i=1}^n |w_i|)$$

Hierarchical Clustering in Machine Learning



Hierarchical Clustering in Machine Learning

- Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.
- In this algorithm, we develop the hierarchy of clusters in the form of **a tree**, and this **tree-shaped structure is known as the dendrogram**.
- The hierarchical clustering technique has two approaches:
 1. **Agglomerative**
 2. **Divisive**

Hierarchical Clustering in Machine Learning



Agglomerative Hierarchical clustering

- The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm **considers each dataset as a single cluster at the beginning**, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.
- This hierarchy of clusters is represented in the form of the **dendrogram**.

Divisive:

- Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach**.

Hierarchical Clustering in Machine Learning

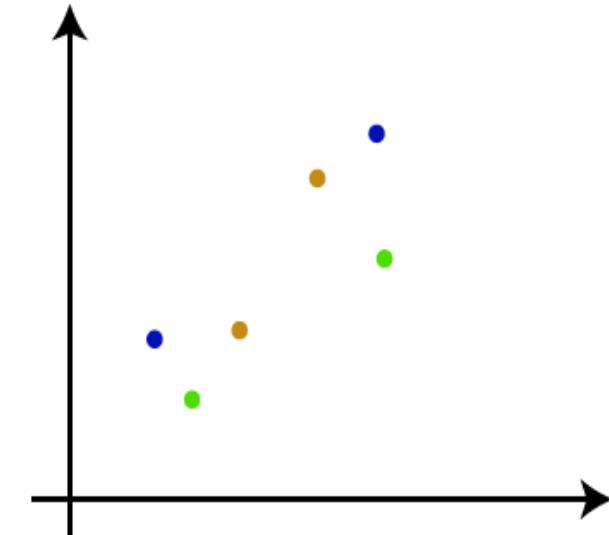
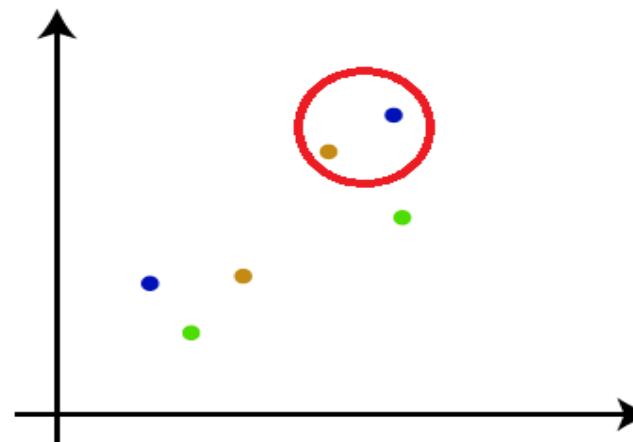


How the Agglomerative Hierarchical clustering Work?

➤ The working of the AHC algorithm can be explained using the below steps:

Step-1: Create **each data point as a single cluster.** Let's say there are N data points, so the number of clusters will also be N.

Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

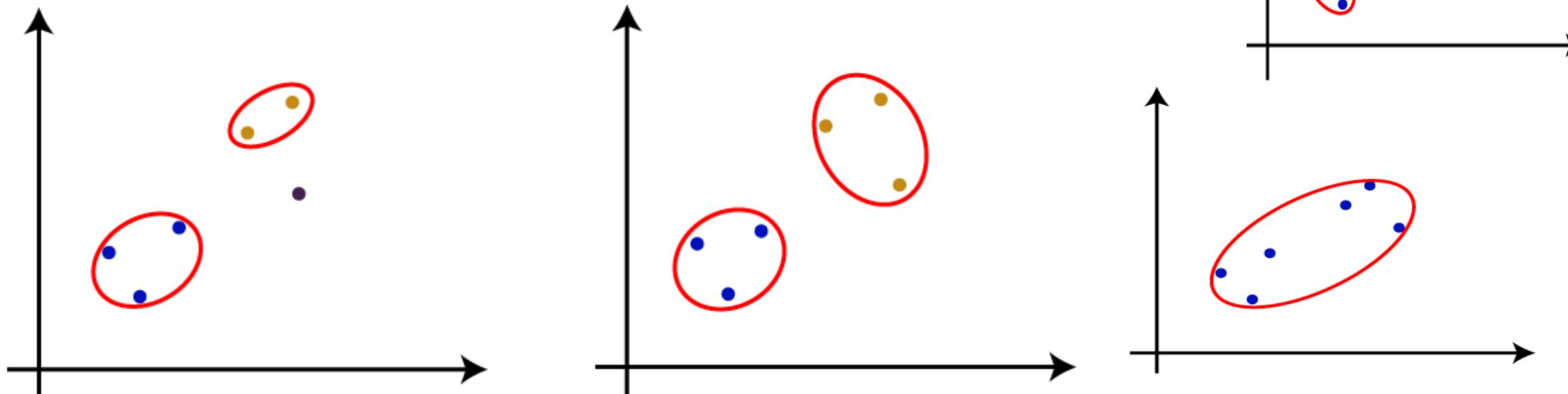


Hierarchical Clustering in Machine Learning



How the Agglomerative Hierarchical clustering Work?

- **Step-3:** Again, take the two closest clusters and merge them together to form one cluster. There will be $N-2$ clusters.
- **Step-4:** Repeat Step 3 until only one cluster left.
So, we will get the following clusters.



Hierarchical Clustering in Machine Learning



How the Agglomerative Hierarchical clustering Work?

Step-5: Once all the clusters are combined into one big cluster, **develop the dendrogram to divide the clusters as per the problem.**

Measure for the distance between two clusters

➤ The **closest distance** between the two clusters is crucial for the hierarchical clustering. There are **various ways to calculate** the **distance between two clusters**, and these ways decide the rule for clustering.

➤ These measures are called **Linkage methods**.

➤ Some of the popular linkage methods are given below:

1. Single Linkage
2. Complete Linkage
3. Average Linkage
4. Centroid Linkage

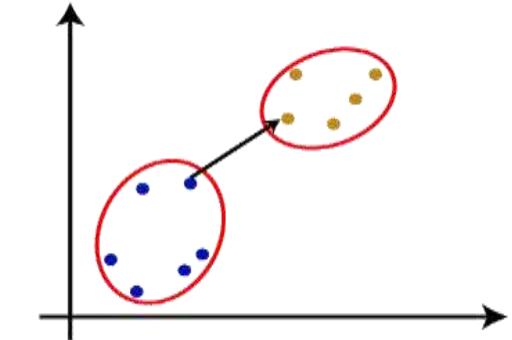
Hierarchical Clustering in Machine Learning



Measure for the distance between two clusters

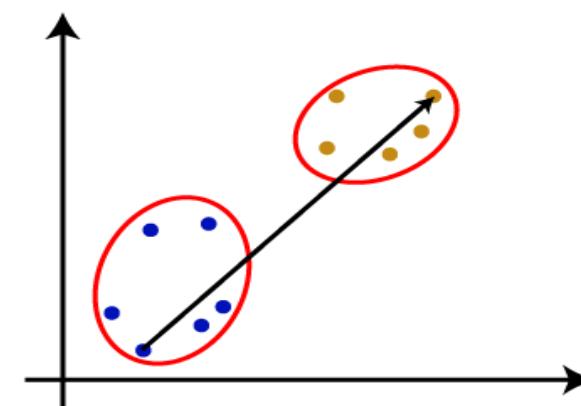
Single Linkage:

- It is the **Shortest Distance** between the closest points of the clusters.



Complete Linkage:

- It is the **farthest distance** between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



Hierarchical Clustering in Machine Learning



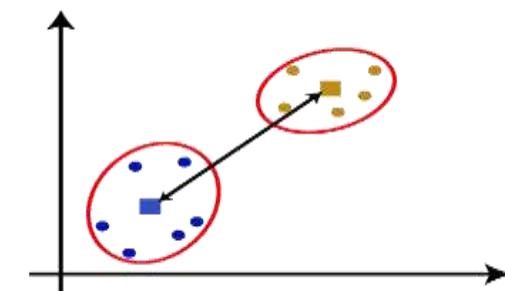
Measure for the distance between two clusters

Average Linkage:

➤ It is the linkage method in which the **distance between each pair of datasets** is added up and then divided by the total number of datasets to calculate the **average distance between two clusters**. It is also one of the most popular linkage methods

Centroid Linkage:

➤ It is the linkage method in which the distance between the centroid of the clusters is calculated.
➤ we can apply any of them according to the type of problem or business requirement.

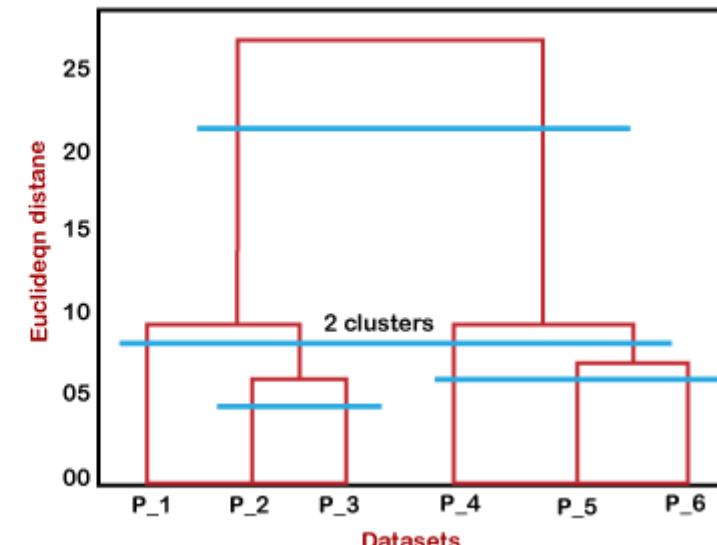
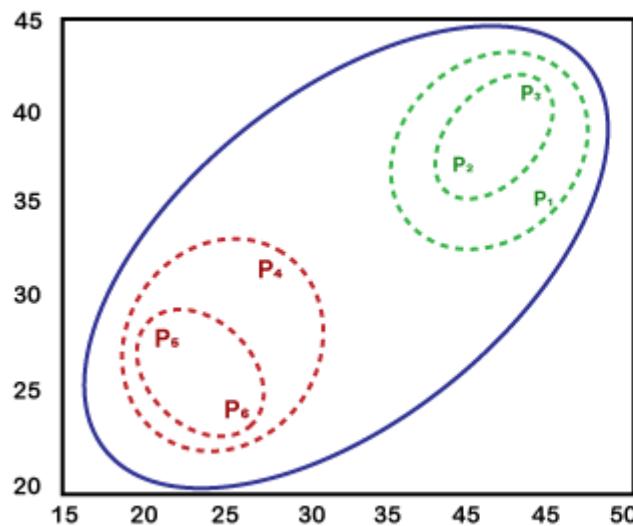


Hierarchical Clustering in Machine Learning



Working of Dendrogram in Hierarchical clustering

- The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.
- The working of the dendrogram can be explained using the below diagram:



Hierarchical Clustering in Machine Learning



Working of Dendrogram in Hierarchical clustering

- In the diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.
- firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The height is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.

Hierarchical Clustering in Machine Learning



Working of Dendrogram in Hierarchical clustering

- Again, two new dendograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.
- We can cut the dendrogram tree structure at any level as per our requirement.

Hierarchical Clustering in Machine Learning



Python Implementation of Agglomerative Hierarchical Clustering

Steps for implementation of AHC using Python:

➤ The steps for implementation are:

1. Data Pre-processing
2. Finding the optimal number of clusters using the Dendrogram
3. Training the hierarchical clustering model
4. Visualizing the clusters

Example:

➤ we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there amounts.

Hierarchical Clustering in Machine Learning



Step-1: Data pre-processing Step

- import the libraries for our model, which is part of data pre-processing.

```
# importing libraries
```

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

- In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

Hierarchical Clustering in Machine Learning



Step-1: Data pre-processing Step

Importing the Dataset:

➤ import the dataset Mall_Customer_data.csv dataset.

```
dataset = pd.read_csv('Mall_Customers_data.csv')
```

➤ The dataset looks like the below image:

➤ From the above dataset,
we need to find some patterns in it.

Index	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	27	20	12

Hierarchical Clustering in Machine Learning



Step-1: Data pre-processing Step

Extracting the matrix of features

➤ Here we will **extract only the matrix of features** as we don't have any further information about the dependent variable.

`x = dataset.iloc[:, [3, 4]].values`

➤ Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters.

➤ So, we are considering the Annual income and spending score as the matrix of features.

➤ `Print(x)`

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]]
```

Hierarchical Clustering in Machine Learning



Step-2: Finding the optimal number of clusters using the Dendrogram

- Now we will **find the optimal number of clusters using the Dendrogram** for our model.
- For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram for our code.

#Finding the optimal number of clusters using the dendrogram

```
import scipy.cluster.hierarchy as shc  
dendro = shc.dendrogram(shc.linkage(x, method="ward"))  
plt.title("Dendrogram Plot")  
plt.ylabel("Euclidean Distances")  
plt.xlabel("Customers")  
plt.show()
```

Hierarchical Clustering in Machine Learning



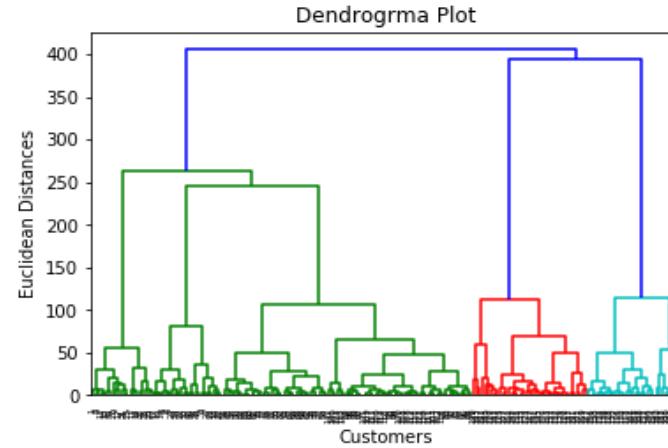
Step-2: Finding the optimal number of clusters using the Dendrogram

- In the code, we have imported the **hierarchy** module of scipy library. This module provides us a method **shc.dendrogram()**, which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the **x**(matrix of features), and method "**ward**," the popular method of linkage in hierarchical clustering.
- The remaining lines of code are to describe the labels for the dendrogram plot.

Hierarchical Clustering in Machine Learning



Step-2: Finding the optimal number of clusters using the Dendrogram Output:



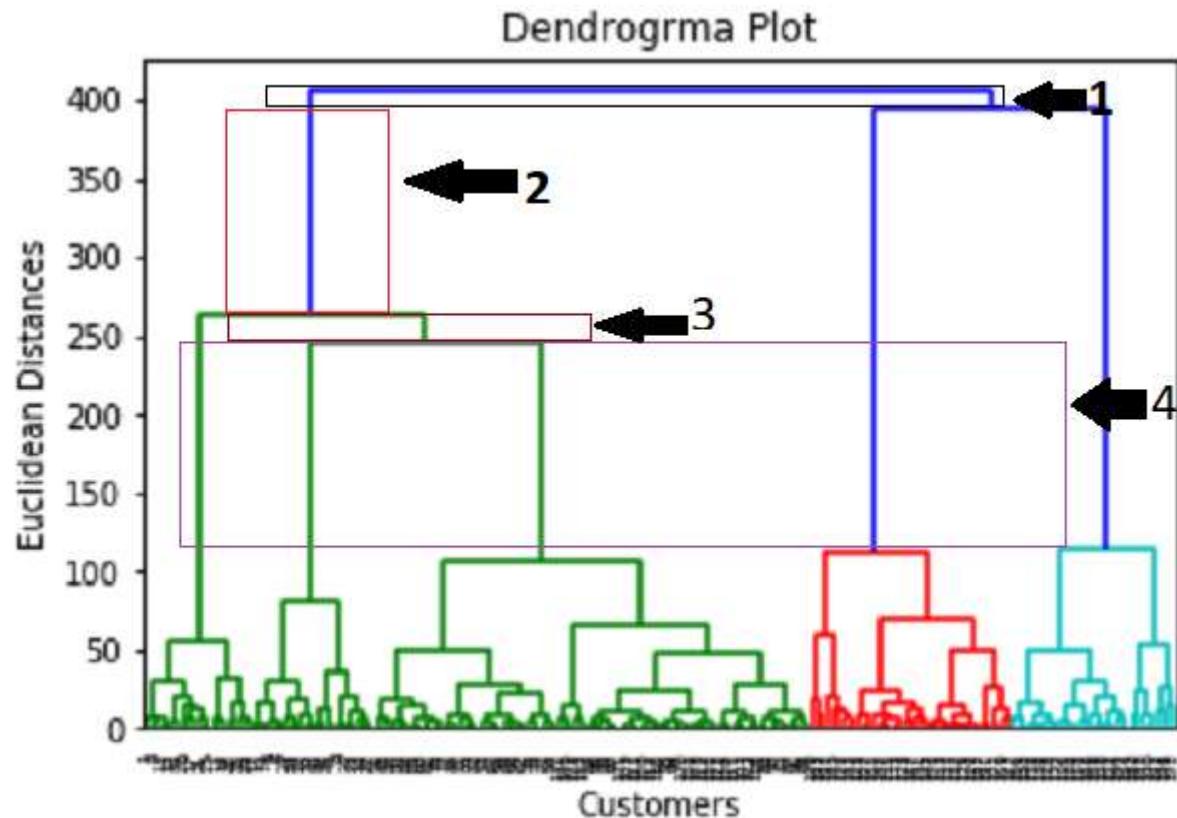
- Using this Dendrogram, we will now determine the optimal number of clusters for our model.
- For this, we will find the **maximum vertical distance** that does not cut any horizontal bar.

Hierarchical Clustering in Machine Learning



Step-2: Finding the optimal number of clusters using the Dendrogram

➤ Consider the below diagram:



Hierarchical Clustering in Machine Learning



Step-2: Finding the optimal number of clusters using the Dendrogram

- In the diagram, we have shown the vertical distances that are not cutting their horizontal bars. As we can visualize, the 4th distance is looking the maximum, so according to this, **the number of clusters will be 5**(the vertical lines in this range). We can also take the 2nd number as it approximately equals the 4th distance, but we will consider the 5 clusters because the same we calculated in the K-means algorithm.
- **So, the optimal number of clusters will be 5**, and we will train the model in the next step, using the same.

Hierarchical Clustering in Machine Learning



Step-3: Training the hierarchical clustering model

- As we know the required optimal number of clusters, we can now **train our model**.

```
#training the hierarchical model on dataset  
from sklearn.cluster import AgglomerativeClustering  
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')  
y_pred= hc.fit_predict(x)
```

- In the code, we have imported the **AgglomerativeClustering** class of **cluster module** of scikit learn library.
- Then we have created the **object of this class named as hc**. The AgglomerativeClustering class takes the following parameters:
 - **n_clusters=5**: It defines the **number of clusters**, and we have taken here 5 because it is the optimal number of clusters.
 - **affinity='euclidean'**: It is a metric used to compute the linkage.
 - **linkage='ward'**: It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

Hierarchical Clustering in Machine Learning



Step-3: Training the hierarchical clustering model

- In the last line, we have **created the dependent variable y_pred to fit or train the model**. It does train not only the model but also returns the clusters to which each data point belongs.
- We can compare the original dataset with the y_pred variable.
- in the image, the **y_pred** shows the clusters value, which means the customer id 1 belongs to the 5th cluster (as indexing starts from 0, so 4 means 5th cluster), the customer id 2 belongs to 4th cluster, and so on.

Index	CustomerID	Gender
0	1	Male
1	2	Male
2	3	Fema
3	4	Fema
4	5	Fema
5	6	Fema
6	7	Fema
7	8	Fema
8	9	Male
9	10	Fema

0
4
3
4
3
4
3
4
3
4

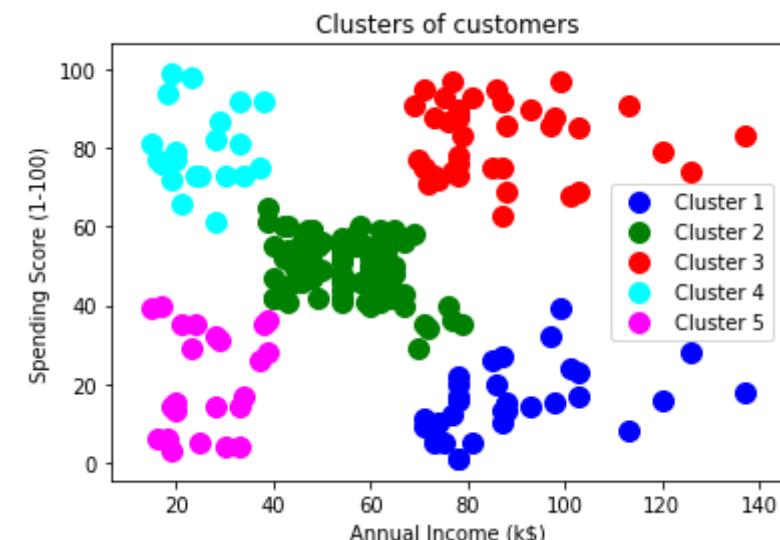
Hierarchical Clustering in Machine Learning



Step-4: Visualizing the clusters

#visualizing the clusters

```
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```



Cost Functions



Cost Function

➤ It is a function that **measures the performance of a Machine Learning model** for given data.

Cost Function quantifies the error between predicted values and expected values and **presents it in the form of a single real number**.

(Or)

The cost function returns the error between predicted outcomes compared with the actual outcomes.

Cost Functions



Cost Function

➤ Depending on the problem **Cost Function** can be formed in many different ways.

➤ The purpose of Cost Function is to be either:
Minimized or Maximized

Minimized -

 returned value is usually called **cost, loss or error**. The goal is to find the values of model parameters for which Cost Function return as small number as possible.

Maximized -

 the value it yields is named a **reward**. The goal is to find values of model parameters for which returned number is as large as possible.

Cost Functions



Cost Function

- The **aim** of supervised machine learning is to minimize the overall cost, thus optimizing the correlation of the model to the system that it is attempting to represent.

Types of the cost function

- There are many cost functions in machine learning and **each has its use cases depending on whether it is a regression problem or classification problem.**
- Some of the important cost functions are:
 1. Regression cost Function
 2. Binary Classification cost Functions
 3. Multi-class Classification cost Functions

Cost Functions



1. Regression cost Function:

- Regression models deal with predicting a continuous value.
- Examples: salary of an employee, price of a car, loan prediction, etc.
- A cost function used in the regression problem is called “Regression Cost Function”.
- They are calculated on the distance-based error as follows:

$$\text{Error} = y - y'$$

Y – Actual Input

Y' – Predicted output

- The most used Regression cost functions are:

1.1 Mean Error (ME)

1.2 Mean Squared Error (MSE)

1.3 Mean Absolute Error (MAE)

Cost Functions



1.1 Mean Error (ME)

- In this cost function, the error for each training data is calculated and then the mean value of all these errors is derived.
- Calculating the mean of the errors is the simplest and most intuitive way possible.
- The errors can be both negative and positive. So they can cancel each other out during summation giving zero mean error for the model.
- Thus this is **not a recommended cost function** but it does lay the foundation for other cost functions of regression models.

Cost Functions



1.2 Mean Squared Error (MSE)

- This improves the drawback we encountered in Mean Error . Here a square of the difference between the actual and predicted value is calculated to avoid any possibility of negative error.
- It is measured as the average of the sum of squared differences between predictions and actual observations.

$$\text{MSE} = (\text{sum of squared errors})/n$$

- It is also known as L2 loss.
- In MSE, since each error is squared, it helps to penalize even small deviations in prediction when compared to MAE.
- But if our dataset has outliers that contribute to larger prediction errors, then squaring this error further will magnify the error many times more and also lead to higher MSE error.
- Hence we can say that it is less robust to outliers

Cost Functions



1.3 Mean Absolute Error (MAE)

- This cost function also addresses the shortcoming of mean error differently. Here an **absolute difference between the actual and predicted value is calculated to avoid any possibility of negative error.**
- So in this cost function, MAE is measured as the average of the sum of absolute differences between predictions and actual observations.

$$\text{MAE} = \frac{\sum_{i=0}^n |y - y'|}{n}$$

- $\text{MAE} = (\text{sum of absolute errors})/n$
- It is also known as **L1 Loss**.
- It is **robust to outliers** thus it will give better results even when **our dataset has noise or outliers**.

Cost Functions



Cost Function For Linear Regression

➤ A Linear Regression model uses a straight line to fit the model. This is done using the equation for a straight line as shown :

$$\text{Output} = a * \text{Input} + b$$

➤ For the Linear regression model, the cost function will be the minimum of the Root Mean Squared Error of the model, obtained by subtracting the predicted values from actual values. The cost function will be the minimum of these error values.

$$\text{Cost Function } (J) = \frac{1}{n} \sum_{i=0}^n (h_{\theta}(x^i) - y^i)^2$$

Cost Functions



2. Cost functions for Classification problems

- Cost functions used in classification problems are different than what we use in the regression problem.
- A commonly used loss function for classification is the **cross-entropy loss**.

Example:

Consider that we have a classification problem of 3 classes as follows.

Class(Orange,Apple,Tomato)

- The machine learning model will give a probability distribution of these 3 classes as output for a given input data.
- The class with the highest probability is considered as a winner class for prediction.

Output = [P(Orange),P(Apple),P(Tomato)]

Cost Functions



2. Cost functions for Classification problems

➤ The actual probability distribution for each class is shown below.

Orange = [1,0,0]

Apple = [0,1,0]

Tomato = [0,0,1]

➤ If during the training phase, the input class is Tomato, the predicted probability distribution should tend towards the actual probability distribution of Tomato.

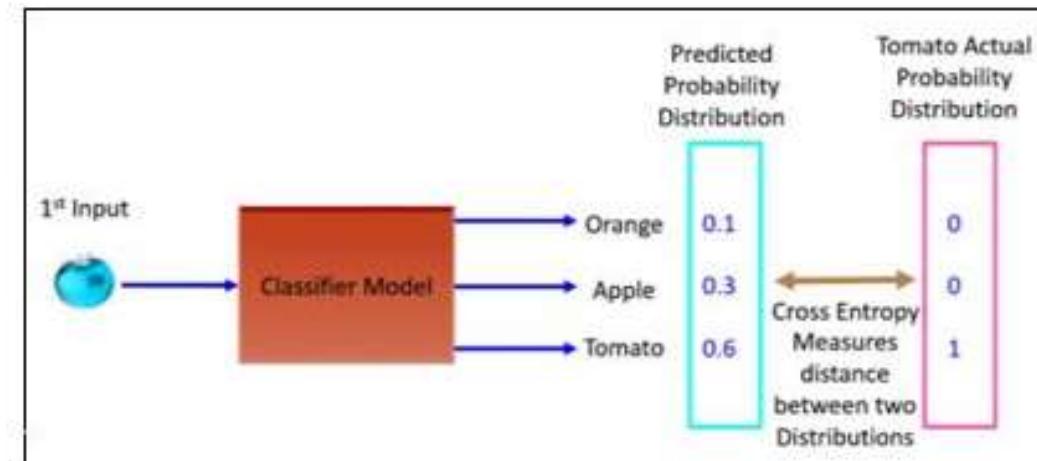
➤ If the predicted probability distribution is not closer to the actual one, the model has to adjust its weight. This is where cross-entropy becomes a tool to calculate how much far the predicted probability distribution from the actual one is.

Cost Functions



2. Cost functions for Classification problems

- In other words, Cross-entropy can be considered as **a way to measure the distance between two probability distributions.**
- The following image illustrates the intuition behind cross-entropy:



Cost Functions



2.1 Multi-class Classification cost Functions

- This cost function is used in the classification problems where there are multiple classes and input data belongs to only one class.
- Let us now understand how cross-entropy is calculated. Let us assume that the model gives the probability distribution as below for 'n' classes & for a particular input data D.

$$P(D) = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}$$

- And the actual or target probability distribution of the data D is

$$Y(D) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Cost Functions



2.1 Multi-class Classification cost Functions

➤ Then cross-entropy for that particular data D is calculated as

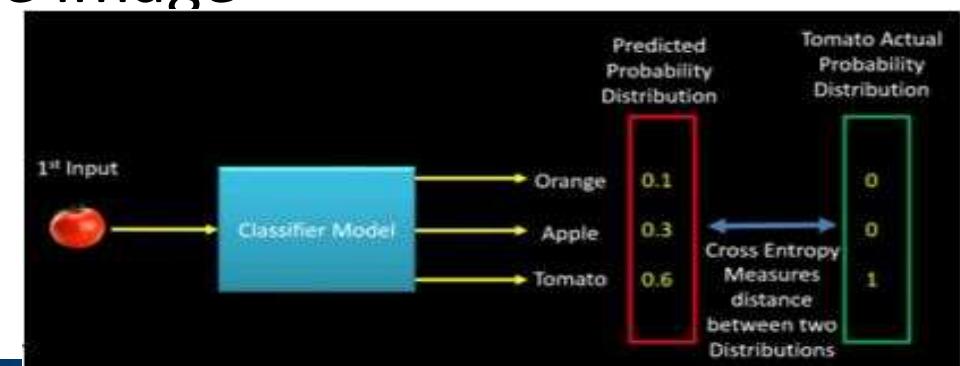
$$\text{Cross-entropy loss}(y, p) = - y^T \log(p)$$

$$= -(y_1 \log(p_1) + y_2 \log(p_2) + \dots + y_n \log(p_n))$$

$$\therefore \text{Cross-entropy loss}(y, p) = - [y_1 \ y_2 \ \dots \ y_n] \begin{bmatrix} \log(p_1) \\ \log(p_2) \\ \vdots \\ \log(p_n) \end{bmatrix}$$

➤ Example:

➤ Let us now define the cost function for the image



Cost Functions



2.1 Multi-class Classification cost Functions

$$p(\text{Tomato}) = [0.1, 0.3, 0.6]$$

$$y(\text{Tomato}) = [0, 0, 1]$$

$$\text{Cross-Entropy}(y, P) = - (0 * \text{Log}(0.1) + 0 * \text{Log}(0.3) + 1 * \text{Log}(0.6)) = 0.51$$

- The above formula just measures the cross-entropy for a single observation or input data.
- The error in classification for the complete model is given by categorical cross-entropy which is nothing but the mean of cross-entropy for all N training data.

Categorical Cross-Entropy = (Sum of Cross-Entropy for N data)/N

Cost Functions



2.2 Binary Cross Entropy Cost Function

➤ Binary cross-entropy is a special case of categorical cross-entropy when there is only one output that just assumes a **binary value of 0 or 1 to denote negative and positive class respectively.**

➤ For example-classification between cat & dog.

➤ Let us assume that actual output is denoted by a single variable y , then cross-entropy for a particular data D is can be simplified as follows –

$$\text{Cross-entropy}(D) = -y * \log(p) \text{ when } y = 1$$

$$\text{Cross-entropy}(D) = -(1-y) * \log(1-p) \text{ when } y = 0$$

The error in binary classification for the complete model is given by binary cross-entropy which is nothing but the mean of cross-entropy for all N training data.

$$\text{Binary Cross-Entropy} = (\text{Sum of Cross-Entropy for } N \text{ data})/N$$

Cost Functions



- The objective of a ML model, therefore, is to find parameters, weights or a structure that **minimises the cost function**.

Minimizing the cost function: Gradient descent

- Gradient descent is an efficient optimization algorithm that **attempts to find a local or global minima of a function**.
- Gradient descent enables a model to learn the **gradient or direction** that the model **should take in order to reduce errors** (differences between actual y and predicted y)

Cost Functions



Gradient Descent For Linear Regression

- By the definition of gradient descent, we have to **find the direction in which the error decreases constantly.**
- This can be **done by finding the difference between errors.** The small difference between errors can be obtained by differentiating the cost function and subtracting it from the previous gradient descent to move down the slope.

$$\text{Gradient Descent } \theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta}$$

- After substituting the value of the cost function (J) in the above equation, we get Linear regression gradient descent function simplified.

Cost Functions



Implementing Cost Functions in Python

Example:

- take a numpy array of random numbers as our data.
- importing important modules.

```
import numpy  
import matplotlib.pyplot as plt
```

- The numpy array is a 2-D array with random points.
- Each element of the array corresponds to an x and y coordinate.
- Here, x is the input and y is the output required.
- Let's separate these points and plot them.

```
points = numpy.loadtxt('data.txt', delimiter=',')  
points  
  
array([[ 6.1101 , 17.592 ],  
       [ 5.5277 ,  9.1302 ],  
       [ 8.5186 , 13.662 ],  
       [ 7.0032 , 11.854 ],  
       [ 5.8598 ,  6.8233 ],  
       [ 8.3829 , 11.886 ],  
       [ 7.4764 ,  4.3483 ],  
       [ 8.5781 , 12. ],  
       [ 6.4862 ,  6.5987 ],  
       [ 5.0546 ,  3.8166 ],  
       [ 5.7107 ,  3.2522 ],  
       [ 14.164 , 15.505 ],  
       [ 5.734 ,  3.1551 ],  
       [ 8.4084 , 7.2258 ],  
       [ 5.6407 ,  0.71618],  
       [ 5.3794 ,  3.5129 ],  
       [ 6.3654 ,  5.3048 ],  
       [ 5.1301 ,  0.56077],
```

Cost Functions

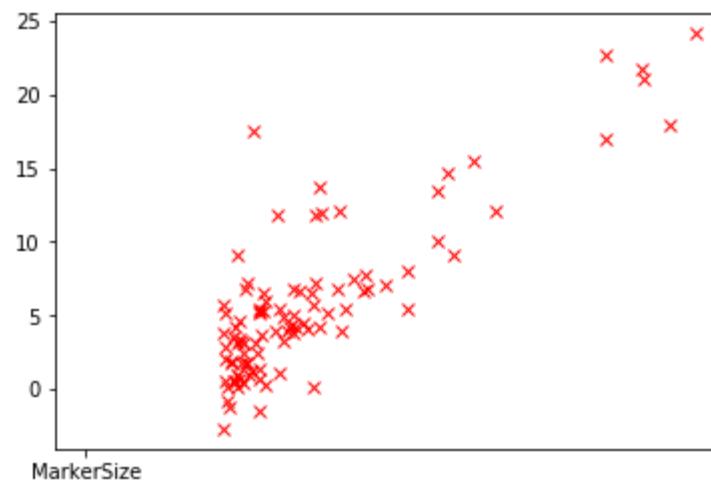


Implementing Cost Functions in Python

```
X = points[:, 0]
Y = points[:, 1]

plt.plot(X, Y, 'rx', 'MarkerSize', 10)

[<matplotlib.lines.Line2D at 0x7f691c32f550>,
 <matplotlib.lines.Line2D at 0x7f691c32f7b8>]
```



- Now, let's set our theta value and store the y values in a different array so we can predict the x values.

Cost Functions



Implementing Cost Functions in Python

```
theta = numpy.zeros((2,1))

new_x = numpy.delete(points, numpy.s_[1:], 1)
X = numpy.insert(new_x, 0, 1, axis=1)
```

➤ Let's initialize the 'm' and 'b' values along with the learning rate.

```
learning_rate = 0.01
initial_m=0
initial_b=0
iteration=1500
```

➤ Using mathematical operations, find the cost function value for our inputs.

```
# cost function
m = len(points)
y = numpy.delete(points, numpy.s_[1:], 1)
next = 1/(2*m)*numpy.sum( numpy.power(numpy.subtract( numpy.dot(X, theta), y), 2))
next
#J = 1/(2*m)*sum(((X*theta)- y)**2)
#J
```

Cost Functions



Implementing Cost Functions in Python

- Using the cost function, you can update the theta value.

```
# cost function
thetasecond = numpy.array([[-1],[2]])
m = len(points)
y = numpy.delete(points, numpy.s_[:1], 1)
nextout = 1/(2*m)*numpy.sum( numpy.power(numpy.subtract( numpy.dot(X, thetasecond), y), 2))
nextout

54.24245508201238
```

- Now, find the gradient descent and print the updated value of theta at every iteration.

```
# Gradient descent
cost_history = numpy.zeros(iteration)
theta_history = numpy.zeros((iteration, 2))
theta_second = numpy.zeros((2,1))
for iter in range(iteration):
    prediction = numpy.dot(X, theta_second)
    theta_second = theta_second -(1/m)*learning_rate*(X.T.dot((prediction - y)))
    theta_history[iter, :] = theta_second.T
    tempcost = 1/(2*m)*numpy.sum( numpy.power(numpy.subtract( numpy.dot(X, theta_second), y), 2))
    cost_history[iter] = tempcost

print(theta_history)

[[ 0.05839135  0.6532885 ]
 [ 0.06289175  0.77000978]
 [ 0.05782293  0.79134812]
 ...
 [-3.6293317   1.16626593]
 [-3.62981201  1.16631419]]
```

Cost Functions

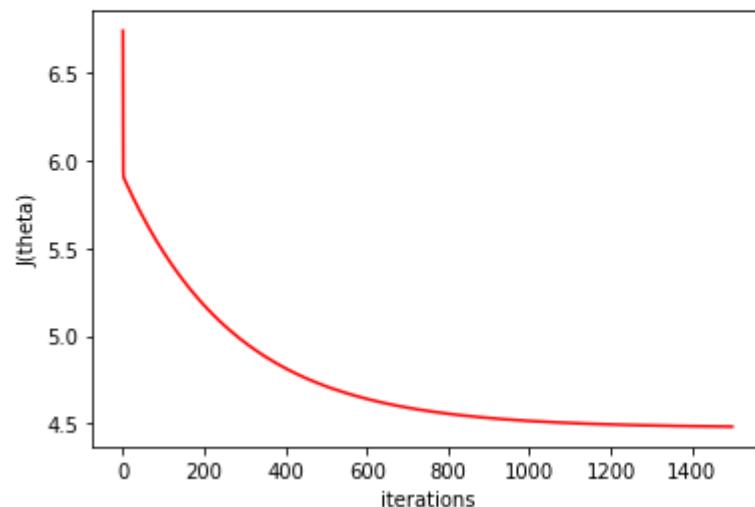


Implementing Cost Functions in Python

- On plotting the gradient descent, you can see the decrease in the loss at each iteration.

```
plt.plot(numpy.arange(iteration), cost_history, 'r', 5)
plt.xlabel('iterations')
plt.ylabel('J(theta)')

Text(0, 0.5, 'J(theta)')
```



Plotting gradient descent

Cross_Validation



- If our algorithm **works well with the training dataset** but **not well with test dataset**, then such problem is **called Overfitting**.



- To overcome over-fitting problems, we **use a technique called Cross-Validation**.

Cross_Validation



What is Cross Validation?

- Cross-validation is a statistical method used to estimate the performance (or accuracy) of machine learning models.
- It is used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited.
- In cross-validation, we make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.

(or)

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

Cross_Validation



Basic Steps of Cross Validation

- the basic steps of cross-validations are:
 1. Reserve a subset of the dataset as a validation set.
 2. Provide the training to the model using the training dataset.
 3. Now, evaluate model performance using the validation set. If the model performs well with the validation set, perform the further step, else check for the issues.

Cross_Validation



Types of Cross Validation

There are **2 categories** of cross validation methods. Those are:

1. Non-exhaustive and
2. Exhaustive Methods

Non-exhaustive Methods:

➤ Non-exhaustive cross validation methods **do not compute all ways of splitting the original data.**

➤ Some of the important Non-exhaustive Methods are:

1. Holdout Method
2. K-Fold Cross-Validation
3. Stratified K-Fold Cross-Validation

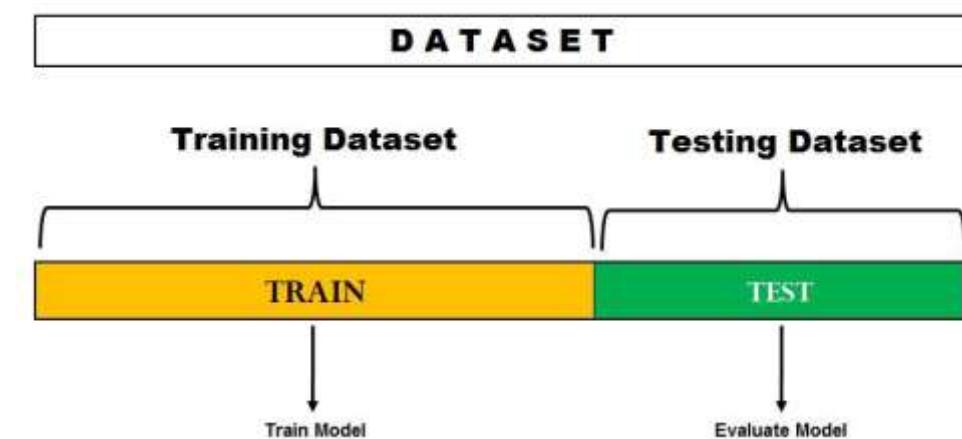
Cross_Validation



1. Hold Out method

- This is the simplest evaluation method and is widely used in Machine Learning projects.
- Here the entire dataset(population) is **divided into 2 sets – train set and test set**. The data can be divided into 70-30 or 60-40, 75-25 or 80-20, or even 50-50 depending on the use case.
- As a **rule, the proportion of training data has to be larger than the test data.**

- The data split happens randomly, and we can't be sure which data ends up in the train and test bucket during the split unless we specify random_state.
- This can lead to extremely high variance and every time, the split changes, the accuracy will also change.



Cross_Validation



1. Hold Out method

➤ One of the major **advantages** of this method is that it is **computationally inexpensive compared to other cross-validation techniques**.

There are some **drawbacks** to this method:

1. In the Hold out method, the test error rates are highly variable (**high variance**) and it totally depends on which observations end up in the training set and test set
2. Only a part of the data is used to train the model (**high bias**) which is not a very good idea when data is not huge and this will lead to overestimation of test error.

Cross_Validation



implementation of Hold Out method in Python

```
from sklearn.model_selection import train_test_split  
X = [10,20,30,40,50,60,70,80,90,100]  
train, test= train_test_split(X,test_size=0.3, random_state=1)  
print("Train:",X_train,"Test:",X_test)
```

Output

Train: [50, 10, 40, 20, 80, 90, 60]

Test: [30, 100, 70]

➤ Here, *random_state* is the seed used for reproducibility.

Cross_Validation



2. K-Fold Cross-Validation

- In this resampling technique, the **whole data** is divided into **k** sets of almost **equal sizes**.
- The first set is selected as the test set and the model is trained on the remaining $k-1$ sets. The test error rate is then calculated after fitting the model to the test data.
- In the second iteration, the 2nd set is selected as a test set and the remaining $k-1$ sets are used to train the data and the error is calculated. This process continues for all the k sets.
- The mean of errors from all the iterations is calculated as the CV test error estimate.



$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k MSE_i$$

Cross_Validation



2. K-Fold Cross-Validation

- Typically, K-fold Cross Validation is performed using $k=5$ or $k=10$ as these values have been empirically shown to yield test error estimates that neither have high bias nor high variance.
- The major **disadvantage** of this method is that the **model has to be run from scratch k-times and is computationally expensive than the Hold Out method** but better than the Leave One Out method.

Cross_Validation



implementation of K-Fold Cross-Validation in Python

```
from sklearn.model_selection import Kfold  
X = ["a",'b','c','d','e','f']  
kf = KFold(n_splits=3, shuffle=False, random_state=None)  
for train, test in kf.split(X):  
    print("Train data",train,"Test data",test)
```

Output

Train: [2 3 4 5]

Test: [0 1]

Train: [0 1 4 5]

Test: [2 3]

Train: [0 1 2 3]

Test: [4 5]

Cross_Validation



3. Stratified K-Fold Cross-Validation

- This is a slight variation from K-Fold Cross Validation, which uses '**stratified sampling**' instead of 'random sampling.'
- Suppose our data contains **reviews for a cosmetic product used by both the male and female population.**
- When we perform random sampling to split the data into train and test sets, there is a possibility that most of the data representing males is not represented in training data but might end up in test data. When we train the model on sample training data that is not a correct representation of the actual population, the model will not predict the test data with good accuracy.
- This is where **Stratified Sampling** comes to the rescue. Here the **data is split in such a way that it represents all the classes from the population.**

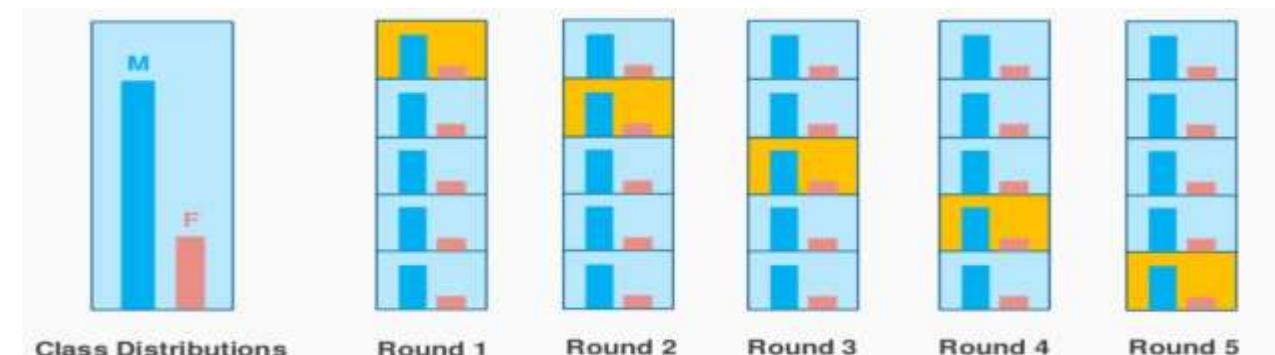
Cross_Validation



3. Stratified K-Fold Cross-Validation

➤ Let's consider the above example which has a cosmetic product review of 1000 customers out of which 60% is female and 40% is male. I want to split the data into train and test data in proportion (80:20). 80% of 1000 customers will be 800 which will be chosen in such a way that there are 480 reviews associated with the female population and 320 representing the male population. In a similar fashion, 20% of 1000 customers will be chosen for the test data (with the same female and male representation).

➤ This is exactly what stratified K-Fold CV does and it will create K-Folds by preserving the percentage of sample for each class. This solves the problem of random sampling associated with Hold out and K-Fold methods.



Cross_Validation



Implementation of Stratified K-Fold Cross-Validation in Python

```
from sklearn.model_selection import StratifiedKFold  
X = np.array([[1,2],[3,4],[5,6],[7,8],[9,10],[11,12]])  
y= np.array([0,0,1,0,1,1]) skf =  
StratifiedKFold(n_splits=3,random_state=None,shuffle=False)
```

```
for train_index,test_index in skf.split(X,y):  
    print("Train:",train_index,'Test:',test_index)
```

```
X_train,X_test = X[train_index], X[test_index]
```

```
y_train,y_test = y[train_index], y[test_index]
```

Output

Train: [1 3 4 5] Test: [0 2]

Train: [0 2 3 5] Test: [1 4]

Train: [0 1 2 4] Test: [3 5]

➤The output clearly shows the stratified split done based on the classes '0' and '1' in 'y'.

Cross_Validation



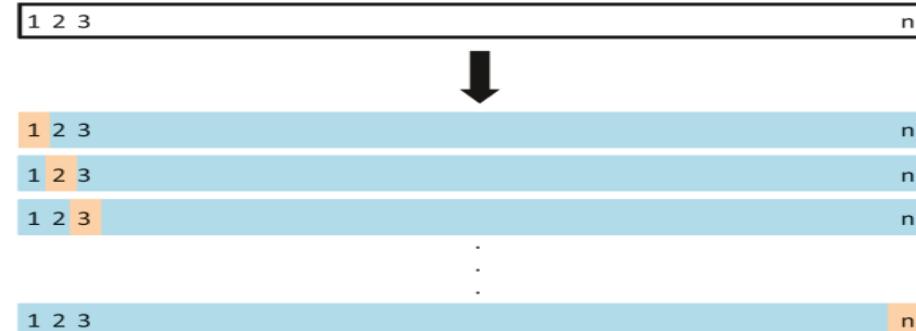
Exhaustive Methods

➤ Exhaustive cross validation methods and test on all possible ways to divide the original sample into a training and a validation set.

1. Leave-P-Out cross validation

1. Leave One Out Cross-Validation

➤ In this method, we divide the data into train and test sets – but with a twist. Instead of dividing the data into 2 subsets, we select a single observation as test data, and everything else is labeled as training data and the model is trained. Now the 2nd observation is selected as test data and the model is trained on the remaining data.



Cross_Validation



implementation of Leave One Out Cross-Validation in Python

```
from sklearn.model_selection import LeaveOneOut  
X = [10,20,30,40,50,60,70,80,90,100]  
l = LeaveOneOut()  
for train, test in l.split(X):  
    print("%s %s"% (train,test))
```

Output

[1 2 3 4 5 6 7 8 9] [0]	[0 2 3 4 5 6 7 8 9] [1]
[0 1 3 4 5 6 7 8 9] [2]	[0 1 2 4 5 6 7 8 9] [3]
[0 1 2 3 5 6 7 8 9] [4]	[0 1 2 3 4 6 7 8 9] [5]
[0 1 2 3 4 5 7 8 9] [6]	[0 1 2 3 4 5 6 8 9] [7]
[0 1 2 3 4 5 6 7 9] [8]	[0 1 2 3 4 5 6 7 8] [9]

➤ This output clearly shows how LOOCV keeps one observation aside as test data and all the other observations go to train data.



Confusion Matrix

Confusion Matrix

- The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data.
- It can only be determined if the true values for test data are known.
- The matrix itself can be easily understood, but the related terminologies may be confusing.
- It shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**.
- Some features of Confusion matrix are :
 1. For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
 2. The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.



Confusion Matrix

3. Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.

➤ It looks like the table:

True Negative: Model has given prediction No, and the real or actual value was also No.

True Positive: The model has predicted yes, and the actual value was also true.

False Negative: The model has predicted yes, but the actual value was no, it is also called as **Type-II error**.

False Positive: The model has predicted no, but the actual value was yes. It is also called a **Type-I error**.

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive



Confusion Matrix

Need for Confusion Matrix in Machine learning

1. It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
2. It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
3. With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.



Confusion Matrix

Example:

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
73	27		

➤ From the above example, we can conclude that:

1. The table is given for the two-class classifier, which has two predictions "Yes" and "NO." Here, Yes defines that patient has the disease, and No defines that patient does not have that disease.



Confusion Matrix

Example:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

2. The classifier has made a **total of 100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
3. The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the **actual "Yes"** was 27, and **actual "No"** was 73 times.



Confusion Matrix

Calculations using Confusion Matrix:

➤ We can perform **various calculations for the model**, such as the **model's accuracy**, using this matrix. These calculations are given below:

- 1. Classification Accuracy**
- 2. Misclassification rate**
- 3. Precision**
- 4. Recall**
- 5. F-measure**



Confusion Matrix

Classification Accuracy:

- It is one of the important parameters to determine the **accuracy** of the **classification problems**.
- It defines how often the model predicts the correct output.
- It can be **calculated** as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers.
- The formula is given below:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$



Confusion Matrix

Misclassification rate:

- It is also termed as **Error rate**, and it defines **how often the model gives the wrong predictions**.
- The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier.
- The formula is given below:

$$\text{Error rate} = \frac{FP + FN}{TP + FP + FN + TN}$$



Confusion Matrix

Precision:

- It can be defined as the **number of correct outputs provided by the model** or out of all positive classes that have predicted correctly by the model, how many of them were actually true.
- It can be calculated using the below formula:

$$\text{Precision} = \frac{TP}{TP+FP}$$



Confusion Matrix

Recall:

- It is defined as the **out of total positive classes**, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

F-measure:

- If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score.
- This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F-measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall}+\text{Precision}}$$



Confusion Matrix

Confusion Matrix using scikit-learn in Python

```
# confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]
# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0])
print('Confusion matrix : \n',matrix)
# outcome values order in sklear
ntp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n',matrix)
```

```
Confusion matrix :
[[2 2]
 [1 5]]
Outcome values :
2 2 1 5

Classification report :
precision    recall    f1-score   support
          1       0.67     0.50      0.57       4
          0       0.71     0.83      0.77       6

           micro avg       0.70     0.70      0.70       10
           macro avg       0.69     0.67      0.67       10
           weighted avg    0.70     0.70      0.69       10
```

K-Means Clustering

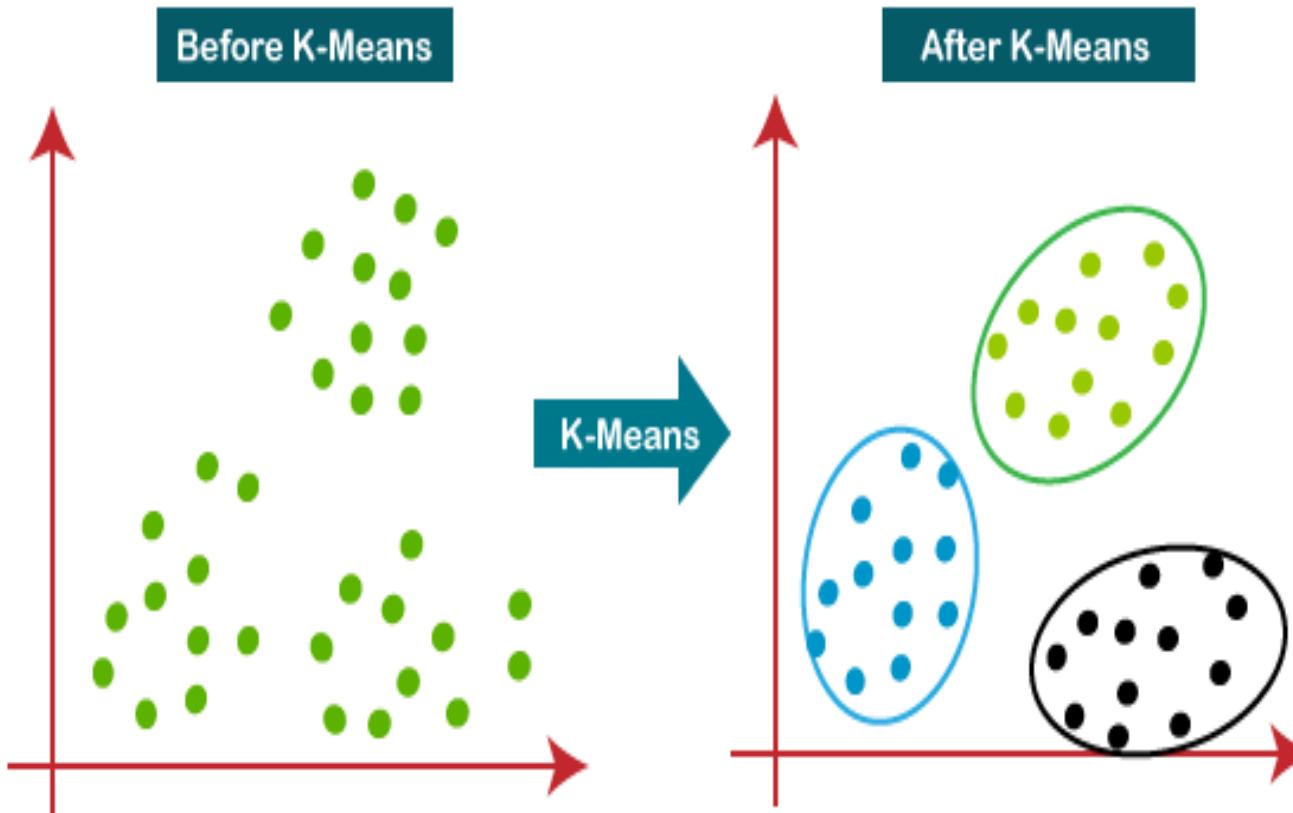


K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

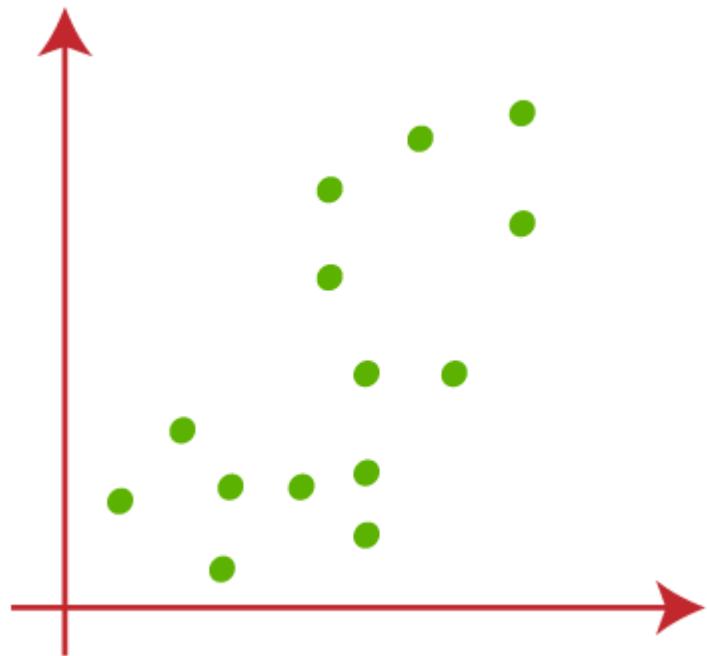
the below diagram explains the working of the K-means Clustering Algorithm:



K-Means algorithm working method

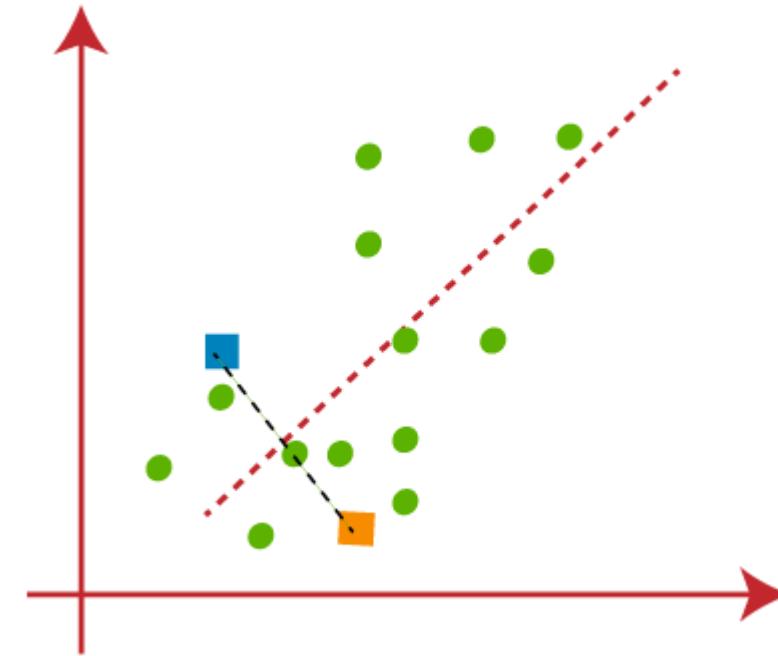
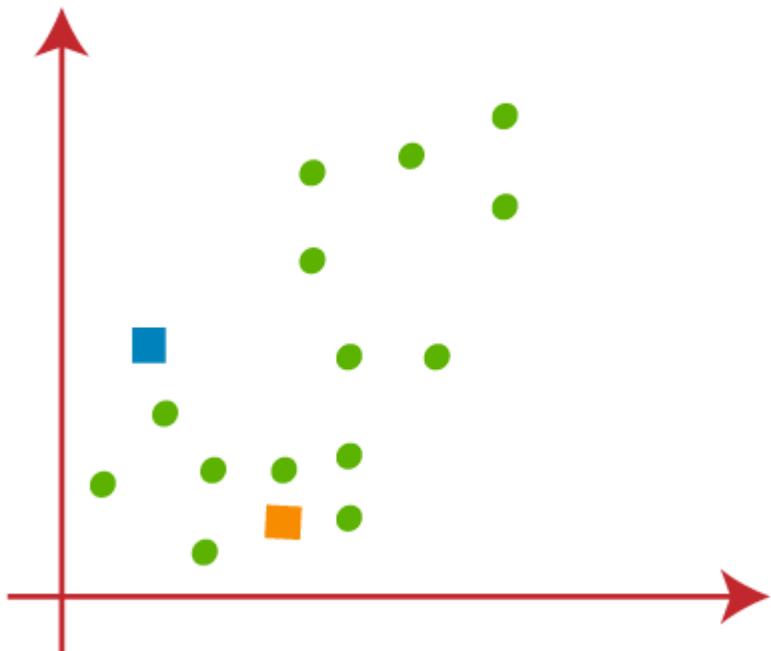
- The working of the K-Means algorithm is explained in the below steps:
- **Step-1:** Select the number K to decide the number of clusters.
- **Step-2:** Select random K points or centroids. (It can be other from the input dataset).
- **Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- **Step-4:** Calculate the variance and place a new centroid of each cluster.
- **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step-7:** The model is ready.

- Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



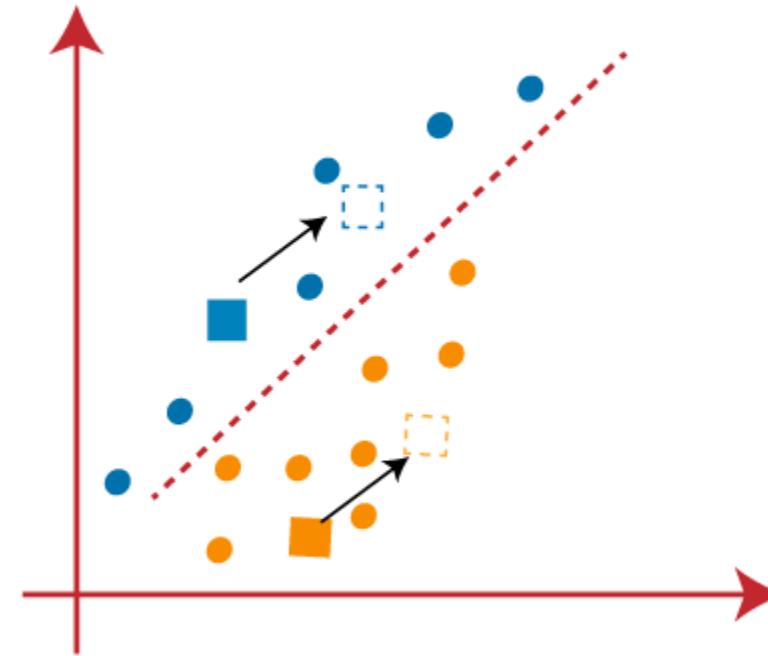
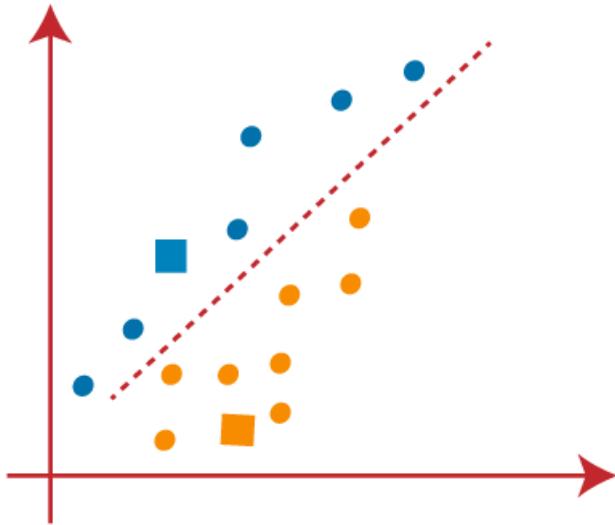
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters.

- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point.

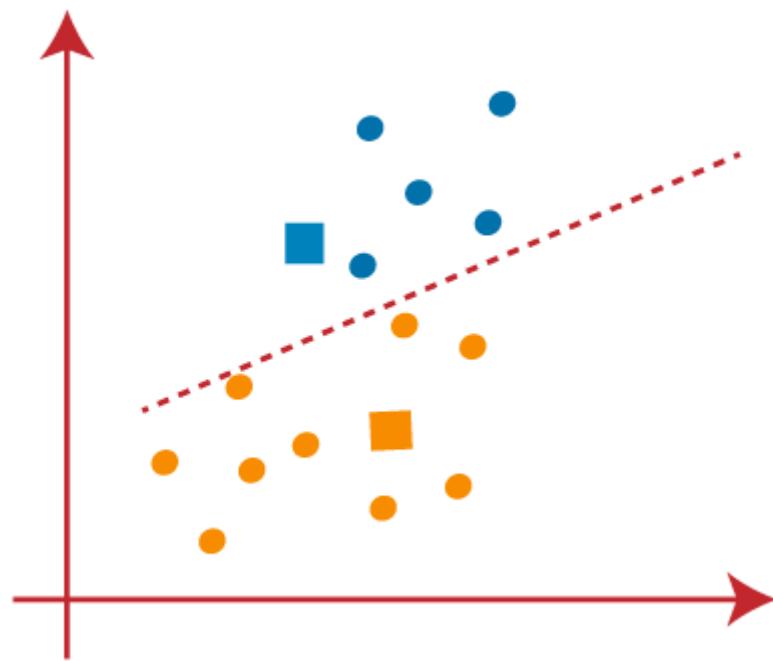
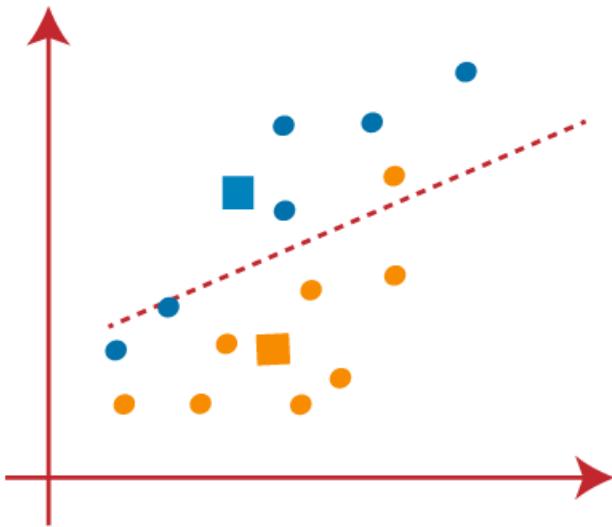


- From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.

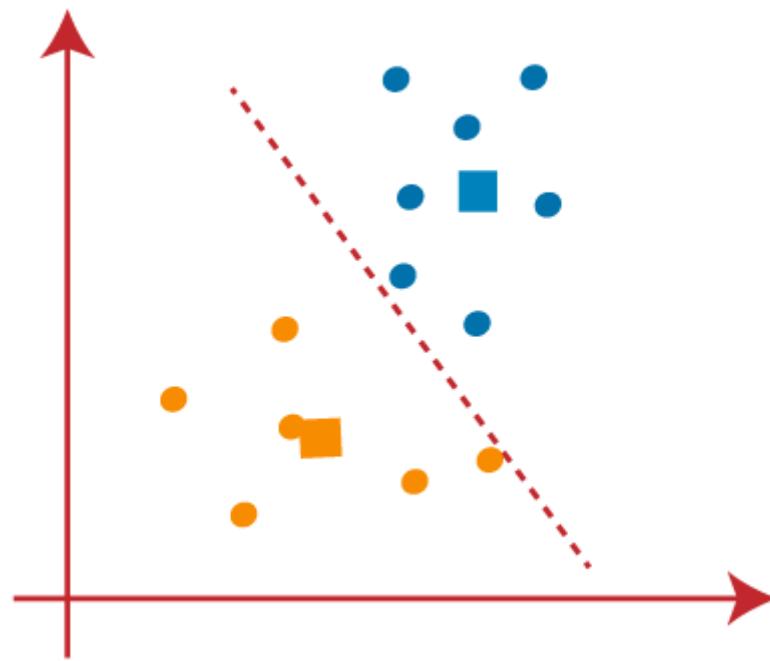
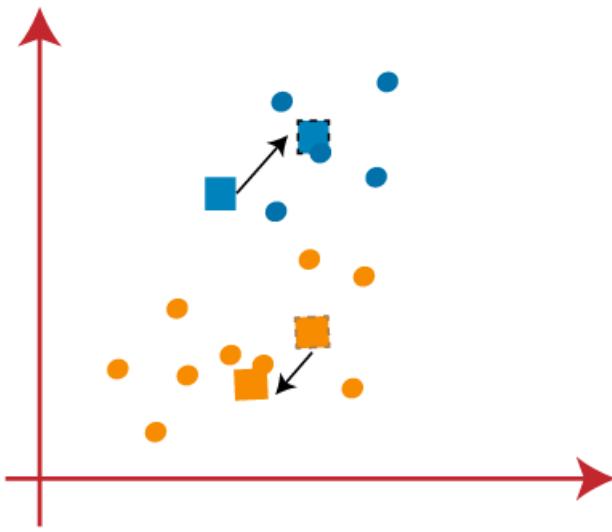
As we need to find the closest cluster, so we will repeat the process by choosing a **new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



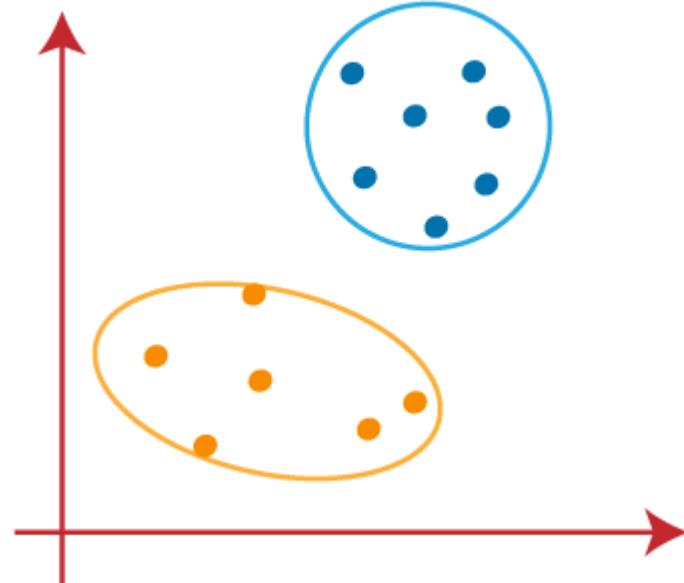
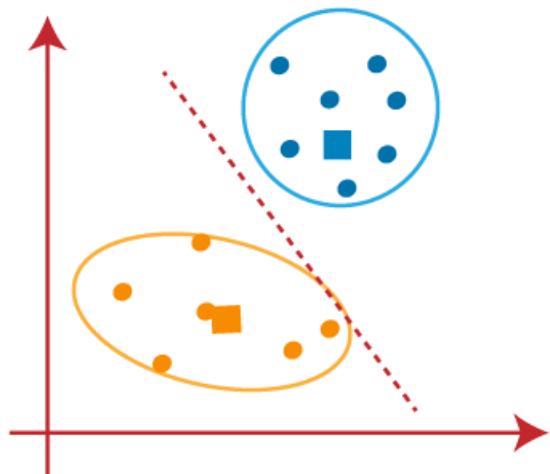
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



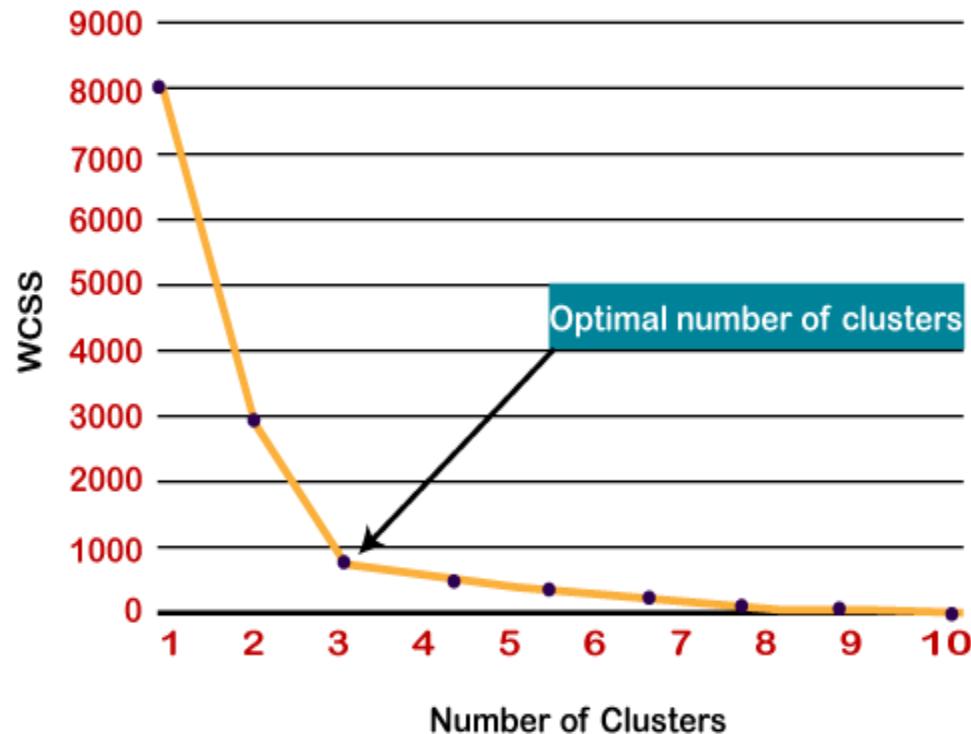
As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



Elbow Method

- The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:
- $\text{WCSS} = \sum_{P_i \text{ in Cluster}_1} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster}_2} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster}_3} \text{distance}(P_i, C_3)^2$
- In the above formula of WCSS,
- $\sum_{P_i \text{ in Cluster}_1} \text{distance}(P_i, C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

- To find the optimal value of clusters, the elbow method follows the below steps:
- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.



Python Implementation of K-means Clustering Algorithm

The steps to be followed for the implementation are given below:

- 1. Data Pre-processing**
- 2. Finding the optimal number of clusters using the elbow method**
- 3. Training the K-means algorithm on the training dataset**
- 4. Visualizing the clusters**



Evaluation Metrics

Evaluation Metrics

- Evaluation metrics are used to measure the quality of the statistical or [machine learning](#) model.
- There are many different types of evaluation metrics available to test a model.
- Some Examples are:
 1. classification accuracy
 2. logarithmic loss
 3. [confusion matrix](#), etc
- Classification accuracy is the **ratio of the number of correct predictions to the total number of input samples**.
- Logarithmic loss, also called **log loss**, works by **penalizing the false classifications**.



Evaluation Metrics

Evaluation Metrics

- A confusion matrix gives us a **matrix as output** and describes the **complete performance of the model**.
- Evaluation metrics involves using a **combination of these individual evaluation metrics** to test a model or algorithm.



Evaluation Metrics

Evaluation metrics for classification, Regression & Clustering

➤ For Classification some of the important evaluation matrices are:

1. **Confusion Matrix**
2. **Recall, Sensitivity & Specificity**
3. **F1 Score**
4. **AUC-ROC(Area under ROC curve)**

AUC-ROC(Area under ROC curve)

A [ROC curve \(receiver operating characteristic curve\)](#) is a graph showing the performance of a classification model at all classification thresholds.



Evaluation Metrics

Evaluation metrics for classification

AUC-ROC(Area under ROC curve)

➤ This curve plots two parameters:

1. True Positive Rate- *Number of True positive divided by the sum of the number of True positive and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.*

2. False Positive Rate- *The number of false positives divided by the sum of the number of false positives and the number of true negatives.*

➤ The ROC curves plot the graph between True positive rate and false-positive rate. These plots are generated at different classification thresholds.

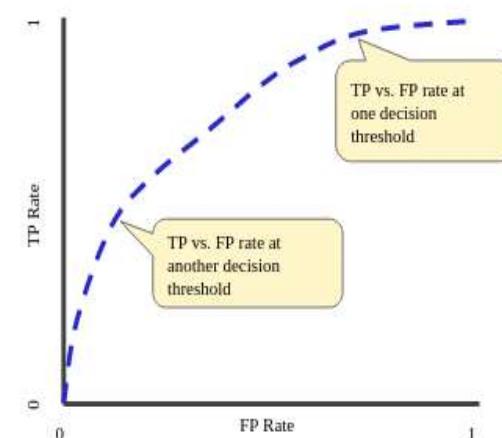
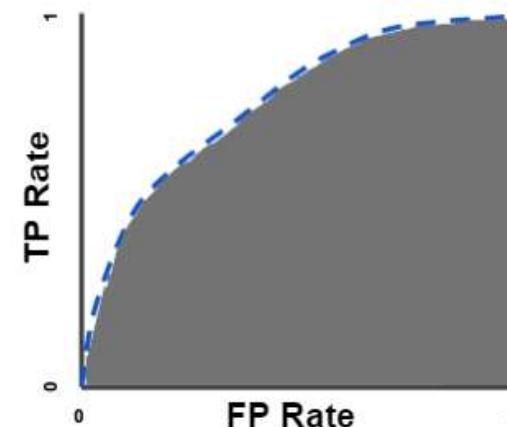


Evaluation Metrics

Evaluation metrics for classification

AUC-ROC(Area under ROC curve)

- The following figure shows a typical ROC curve.
- AUC is an acronym for Area under the curve.
It computes the entire 2D area under the ROC curve.



- it is a plot of FPR(False positive rate) on the x-axis and TPR(True positive rate) on the y-axis for different threshold ranging from 0.0 to 1.



Evaluation Metrics

Evaluation metrics for classification

AUC-ROC(Area under ROC curve)

- The AUC ROC Plot is one of the most popular metrics used for determining machine learning model predictive capabilities.
- some reasons for using AUC ROC plot-
 1. The Different Machine learning models curves can be checked with different thresholds.
 2. Model predictive capability is summarized by the area under the curve(AUC).
 3. AUC is considered to be scaled variant, it measures the rank of predictions rather than its absolute values
 4. AUC always focuses on the quality of the Model's skills on prediction irrespective of what threshold has been chosen.



Evaluation Metrics

Evaluation metrics for classification

Logarithmic Loss

- Logarithmic Loss or Log loss **works by penalizing the false classification.**
- It is nothing but a negative average of the log of corrected predicted probabilities for each instance.
- Log loss mostly suits the multi-class classification problem.
- Log loss takes the **probabilities for all classes present in the sample.**
- If we minimize the log loss for a particular classifier we get better performance metrics.



Evaluation Metrics

Evaluation metrics for classification

Logarithmic Loss

- The math formula is given below

$$\text{LogarithmicLoss} = \frac{-1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} * \log(p_{ij})$$

y_{ij} , indicates whether sample i belongs to class j or not

p_{ij} , indicates the **probability of sample i belonging to class j**

- Log loss has a range $[0, \infty)$. Moreover, it has no upper bound limit. We can interpret log loss as it is nearer to 0 have higher accuracy whereas if log loss moves away from 0 indicates lower accuracy.



Evaluation Metrics

Evaluation metrics for Regression

➤ For **Regression** some of the important evaluation matrices are:

1. **Mean Absolute Error**
2. **Mean Squared Error**
3. **R Squared**

R Squared

➤ R squared is a statistical measure of how close the data point is fitted to the regression line.

- It is also known as the coefficient of determination.
- R-Squared is defined by the explained variation divided by total variation that is explained by the linear model.



Evaluation Metrics

Evaluation metrics for Regression

R Squared

- R squared value always lies **between 0% to 100 %**
- 0% indicates none of the variability of the response data around its mean and 100 % shows model explains all the variability of the response data around its mean.
- This clearly means a higher R square value model perfect your model is.

R-squared = Explained variation / Total variation



Evaluation Metrics

Evaluation metrics for Clustering

Clustering

- it is difficult to figure out the quality of results from clustering. Evaluation metric cannot depend on the labels but only on the goodness of split.
- *For Clustering some of the important evaluation matrices are:*

1. ***Adjusted Rand Score***
2. ***Adjusted mutual information***
3. ***Silhouette***



Evaluation Metrics

Evaluation metrics for Clustering

Adjusted Rand Score

- Adjusted Rand score does not depend on the label values but on the cluster split.
- The formula for Adjusted Rand score is given by
- Where ‘n’ be the number of observations in a sample,
- a to be the number of observation pairs with the same labels and located in the same cluster, and
- b to be the number of observations with different labels and located in different clusters.

$$RI = \frac{2(a + b)}{n(n - 1)}.$$



Evaluation Metrics

Evaluation metrics for Clustering

Adjusted mutual information

- Adjusted mutual information is defined by entropy function and interpreted by sample split which is the likelihood of assigning a cluster.

$$h = 1 - \frac{H(C | K)}{H(C)}, c = 1 - \frac{H(K | C)}{H(K)},$$

- Here K is a clustering result and C is the initial split.
- h evaluates whether each cluster is composed of same class objects
- c measures how well the same class fits the clusters.



Evaluation Metrics

Evaluation metrics for Clustering

Silhouette

- The coefficient of silhouette score is calculated by mean intra-cluster distance and the mean nearest-cluster distance for each sample.

$$s = \frac{b - a}{\max(a, b)}.$$

- The Silhouette distance shows up to which extent the distance between the objects of the same class differs from the mean distance between the objects from different clusters.
- Values of silhouette score lie between -1 to +1 .
- if the value is close to 1 then it corresponds to good clustering results having dense and well-defined clusters



Evaluation Metrics

Evaluation metrics for classification, Regression & Clustering

Silhouette

- if the value is close to -1 then it represents bad clustering.
- Therefore, the higher the silhouette value is, the better the results from clustering.



Class-Imbalance

What is class Imbalance in machine learning?

➤ In machine learning class imbalance is **the issue of target class distribution**. If the target classes are **not equally distributed or not in an equal ratio**, we call the data having an **imbalance data issue**.

Class Imbalance dataset Examples:-

1. Email spam or ham dataset
2. Credit card fraud detection
3. Machine components failure detections
4. Network failure detections

➤ But when it comes to the imbalanced dataset, **the target distribution is not equal**. For email spam or ham, distribution is not equal.



Class-Imbalance

How to deal with imbalance data

➤ To deal with imbalanced data issues, we need to **convert imbalance to balance data** in a meaningful way. Then **we build the machine learning model on the balanced dataset.**

Techniques for handling imbalanced data

➤ top eight ways we can manage the imbalanced classes in our [dataset](#).

- 1 | Changing Performance Metric
- 2 | The More Data, The Better
- 3 | Experiment With Different Algorithms
- 4 | Resampling of Dataset
- 5 | Use of Ensemble Methods
- 6 | Generating Synthetic Samples
- 7 | Multiple Classification System
- 8 | Use of Cost-Sensitive Algorithms



Class-Imbalance

1) Changing Performance Metric

- Performance metrics play a fundamental role while building a machine learning model.
- Implying the incorrect performance metric on an imbalance dataset can yield wrong outcomes.
- **accuracy** is considered to be an important metric for evaluating the performance of a machine learning model, sometimes it can be misleading in case of an imbalanced dataset.
- In such circumstances, one must use other performance metrics such as **Recall, F1-Score, False Positive Rate (FPR), Precision,etc.**



Class-Imbalance

2) The More Data, The Better

➤ Machine learning models are data-hungry. In most cases, researchers spend most of their time in tasks like data cleaning, analysing, visualising, among others during an end-to-end machine learning process and contribute less time in data collection. While all of these steps are important, often the collection of data gets limited to certain numbers. To avoid such circumstances, one must add more data into the dataset. **Collecting more data with relevant examples of the undersampled class of the dataset will help to overcome the issue.**

3) Experiment With Different Algorithms

➤ Another way to handle and manage imbalanced dataset is to **try different algorithms rather than sticking to one particular algorithm**. Experimenting with different algorithms provides a probability to check how the algorithms are performing by a particular dataset.



Class-Imbalance

4) Resampling of Dataset

- To deal with the imbalanced dataset, researchers have introduced a number of resampling techniques. One of the benefits of using these techniques is that they are **external approaches using the existing algorithms**. They can be easily transportable in case of both undersampling and oversampling.
- Some of the popular resampling methods are as follows-

i.) Random Oversampling:

Oversampling seeks to increase the number of minority class members in the training set. Random oversampling is a simple approach to resampling, where one chooses members from the minority class at random. Then, these randomly chosen members are duplicated and added to the new training set.



Class-Imbalance

ii). Random Undersampling:

Undersampling is a process that seeks to reduce the number of majority class members in the training set. [Random undersampling](#) is a popular technique for resampling, where the majority class documents in the training set are randomly eliminated until the ratio between the minority and majority class is at the desired level.

5) Use of Ensemble Methods

Use of ensemble methods is one of the ways to handle the class imbalance problems of the dataset. The learning algorithms construct a set of classifiers and then classify new data points by making a choice of their predictions known as [Ensemble methods](#). It has been discovered that ensembles are often much more accurate than the individual classifiers which make them up. Some of the commonly used Ensemble techniques are Bagging or Bootstrap Aggregation, Boosting and Stacking.



Class-Imbalance

6) Generating Synthetic Samples

- Synthetic Minority Oversampling Technique or SMOTE is one of the most popular approaches for generating synthetic samples. **SMOTE is an oversampling approach in which the minority class is over-sampled by creating synthetic examples rather than by over-sampling with replacement.**
- It is basically a combination of oversampling the minority (abnormal) class and undersampling the majority (normal) class, which is found to achieve better classifier performance (in ROC space) than only undersampling the majority class.



Class-Imbalance

7) Multiple Classification System

- Multiple Classification system is an approach where a classification system is built for imbalanced data based on the combination of several classifiers.
- It is a method for **building multiple classifier systems in which each constituting classifier is trained on a subset of the majority class and on the whole minority class.**
- The basis behind this method is the partition of the set of samples of the majority class in several subsets, each consisting of as many samples as the minority class.



Class-Imbalance

8) Use of Cost-Sensitive Algorithms

- Cost-Sensitive Learning is a type of learning that **takes the misclassification or other types of costs into consideration.**
- Cost-sensitive learning is a popular and common approach to solve the class imbalanced datasets.
- Popular machine learning libraries such as **support vector machines (SVM), random forest, decision trees, logistic regression, among others, can be configured using the cost-sensitive training.**



THANK YOU

Designed by Dr.Subbaiah sir

Visit vitspace.netlify.com for more