Unit-IV

Introduction to Pattern Recognition and Machine Learning: Patterns, features, pattern representation, the curse of dimensionality, dimensionality reduction. Classification—linear and non-linear. Bayesian, Perceptron, Nearest neighbor classifier, Logistic regression, Naïve-Bayes, decision trees and random forests; boosting and bagging.Clustering---partitional and hierarchical; k-means clustering. Regression. Cost functions, training and testing a classifier. Cross-validation, Class-imbalance – ways of handling, Confusion matrix, evaluation metrics.

Introduction to Pattern Recognition and Machine Learning

Patterns, features, pattern representation:-

Feature?

A **Feature** is an individual measurable property or characteristic of a phenomenon.

Choosing informative, discriminating and independent features is a crucial element of effective algorithms in pattern recognition, classification and regression.

To understand in more simple way, lets take an example, where you can consider one column of your data set to be one feature which is also know as "variables or attributes" and the more number of features are known as dimensions.

Examples – Eno, Stud_Rno, Play_tennis etc…

What is a pattern?

• A pattern represents a physical object or an abstract notion. For example, 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.

What are classes?

• The patterns belong to two or more classes.

• The task of pattern recognition pertains to finding the class to which

a pattern belongs.

• The attributes or features used to represent the patterns should be discriminatory attributes. This means that they help in classifying the patterns.

• The task of finding the discriminatory features is called feature extraction/selection.

Representation of patterns:

- Patterns can be represented in a number of ways.
- All the ways pertains to giving the values of the features used for that particular pattern.
- For supervised learning, where a training set is given, each pattern in the training set will also have the class of the pattern given.

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.

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. As can be seen,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.

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

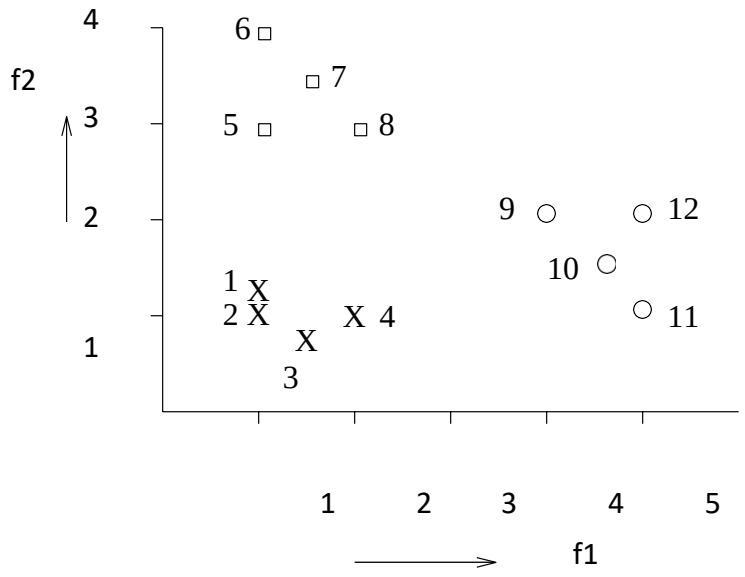


Figure 2: Dataset of three classes

Pattern 1 : (1,1.25,1)	Pattern 2 : (1,1,1)
Pattern 3 : (1.5,0.75,1)	Pattern 4 : (2,1,1)
Pattern 5 : (1,3,2)	Pattern 6 : (1,4,2)
Pattern 7 : (1.5,3.5,2)	Pattern 8 : (2,3,2)
Pattern 9 : (4,2,3)	Pattern 10 : (4.5,1.5,3)
Pattern 11 : (5,1,3)	Pattern 12 : (5,2,3)

Each triplet consists of feature 1, feature 2 and the class label. This isshown in Figure 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 formVHLTPEEK ....
- Each string of characters represents a pattern. Operations like pattern matching or finding the similarity between strings are carried out withthese patterns.
- More details on proteins and genes can be got from [1].

Representing patterns by using logical operators

- Here each pattern is represented by a sentence(well formed formula) ina logic.
- An example would be
  if (beak(x) = red) and (colour(x) = green) then parrot(x)
  This is a rule where the antecedent is a conjunction of primitives andthe consequent is the class label.
- Another example would be
  if (has-trunk(x)) and (colour(x) = black) and (size(x) = large) thenelephant(x)

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 heightwhich 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. An example of this would be the pattern

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

  The above example gives a pattern with 4 features. The 4th featureis 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. When a particular feature of a pattern is missing, looking at other patterns, we can find a range of values which this feature can take. This can be represented as an interval.
  The example pattern given above has the second feature as a linguistic value. The first feature is an integer and the third feature is a real value.
- Rough sets are used to represent classes. So, 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.
- For example, consider Figure 3. 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). 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.

- Not just the features, each pattern can have grades of membership toevery class instead of belonging to one class. In other words, each

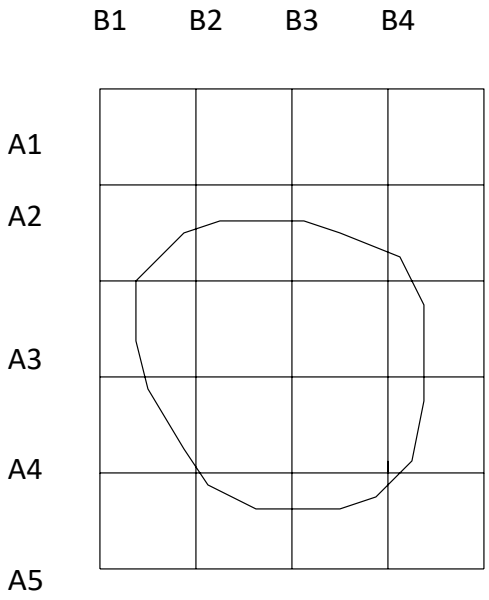


Figure 3: Representation of an object

pattern has a fuzzy label which consists of *c* values in [0,1] where each component gives the grade of membership of the pattern to one class. Here *c* gives the number of classes. For example, consider a collection of documents. It is possible that each of the documents may be associated with more than one category. A paragraph in a document, for instance, may be associated with *sport* and another with *politics*.

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 ofsmall and big is different for different people.

The curse of dimensionality:-

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.

Gathering a huge number of data may lead to the dimensionality problem where highly noisy dimensions with fewer pieces of information and without significant benefit can be obtained due to the large data. The exploding nature of spatial volume is at the forefront is the reason for the curse of dimensionality.

The difficulty in analysing high-dimensional data results from the conjunction of two effects.

- High-dimensional spaces have geometrical properties which are counter-intuitive, and far from the properties which can be observed in two-or three-dimensional spaces.
- Data analysis tools are most often designed having in mind intuitive properties and examples in low-dimensional spaces and usually, data analysis tools are best illustrated in 2-or 3-dimensional spaces.

The difficulty is that those tools are also used when data are high-dimensional and more complex and hence, there is a probability to lose the intuition of the behavior of the tool and thus drawing incorrect conclusions.

A careful choice of the number of dimensions (features) to be used is the prerogative of the data scientist training the network. In general the smaller the size of the training set, the fewer features she should use. She must keep in mind that each features increases the data set requirement exponentially.

Dimensionality Reduction*:-*

Dimensionality reduction is a method of converting the high dimensional variables into lower dimensional variables without changing the specific information of the variables. To overcome the issue of the curse of dimensionality, Dimensionality Reduction is used to reduce the feature space with consideration by a set of principal features. Dimensionality Reduction contains no extra variables that make the data analyzing easier and simple for machine learning algorithms and resulting in a faster outcome from the algorithms.

Approaches of Dimension Reduction

There are two ways to apply the dimension reduction technique, which are given below:

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

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:

- Correlation
- Chi-Square Test
- ANOVA
- Information Gain, etc.

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:

- Forward Selection
- Backward Selection
- Bi-directional Elimination

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:

- LASSO
- Elastic Net
- Ridge Regression, etc.

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:

a. Principal Component Analysis
b. Linear Discriminant Analysis

c. Kernel PCA
d. Quadratic Discriminant Analysis

Classification

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.

Linear Classification:

Most classifiers we've seen use linear functions to separate classes.

- Perceptron
- Logistic regression
- Support vector machines (unless kernelized)

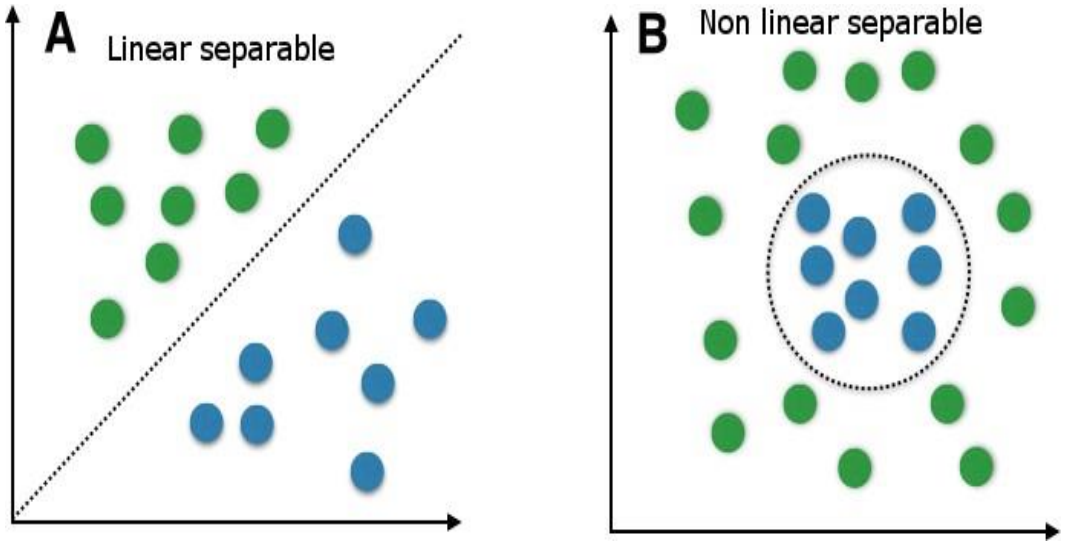
Nonlinear Classification:

Nonlinear functions can be used to separate instances that are not linearly separable.

We've seen two nonlinear classifiers:

- k-nearest-neighbors (kNN)
- Kernel SVM

Kernel SVMs are still implicitly learning a linear separator in a higher dimensional space, but the separator is nonlinear in the original feature space.



Decision Tree Induction:-

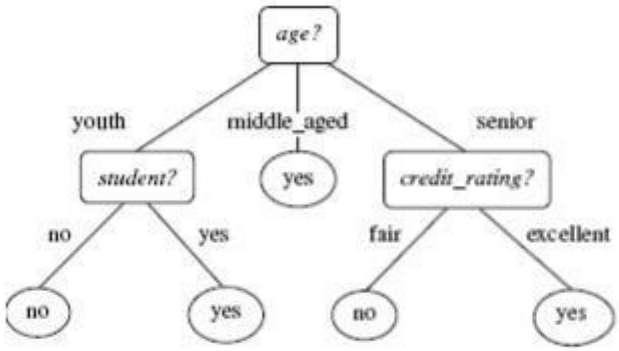
A decision tree is a flowchart-like tree structure, where

- **– each internal node (non-leaf node) denotes a test on an attribute**
- **– each branch represents an outcome of the test**
- **– each leaf node (or terminal node) holds a class label.**
- **– The topmost node in a tree is the root node.**

This example represents the concept buys _computer, that is, it predicts whether a customer at AllElectronics is likely to purchase a computer.

Table 8.1 Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no



A decision tree for the concept *buys_computer*, indicating whether a customer at *AllElectronics* is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

How are decision trees used for classification?

– Given a tuple, X, for which the associated class label is unknown,
– The attribute values of the tuple are tested against the decision tree
– A path is traced from the root to a leaf node, which holds the class prediction for that tuple

- **Basic Algorithm for Decision Tree Induction**

  – Tree is constructed in a top-down recursive divideand-conquer manner
  – At start, all the training examples are at the root
  – Attributes are categorical (if continuous-valued, they are discretized in advance)
  – Examples are partitioned recursively based on selected attributes
  – Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

- **Conditions for stopping partitioning**

  – All samples for a given node belong to the same class
  – There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  – There are no samples left

Attribute Selection Measures -

Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.

– Information gain
– Gain ratio
– Gini index

The notation used herein is as follows.

– Let D, the data partition, be a training set of classlabeled tuples.
– Suppose the class label attribute has m distinct values defining m distinct classes, Ci(for i = 1, … , m)
– Let C be the set of tuples of class C in D.
– Let Ci,D be the set of tuples of class Ci in D.
– Let |D| and |Ci,D| denote the number of tuples in D and Ci,D, respectively

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Gain(A) = Info(D) - Info_A(D).$$

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits}.$$

Table 6.1 Class-labeled training tuples from the *AllElectronics* customer database.

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14} \times (-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) + \frac{4}{14} \times (-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}) + \frac{5}{14} \times (-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}) = 0.694 \text{ bits.}$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits. Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node *N* is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 6.5.

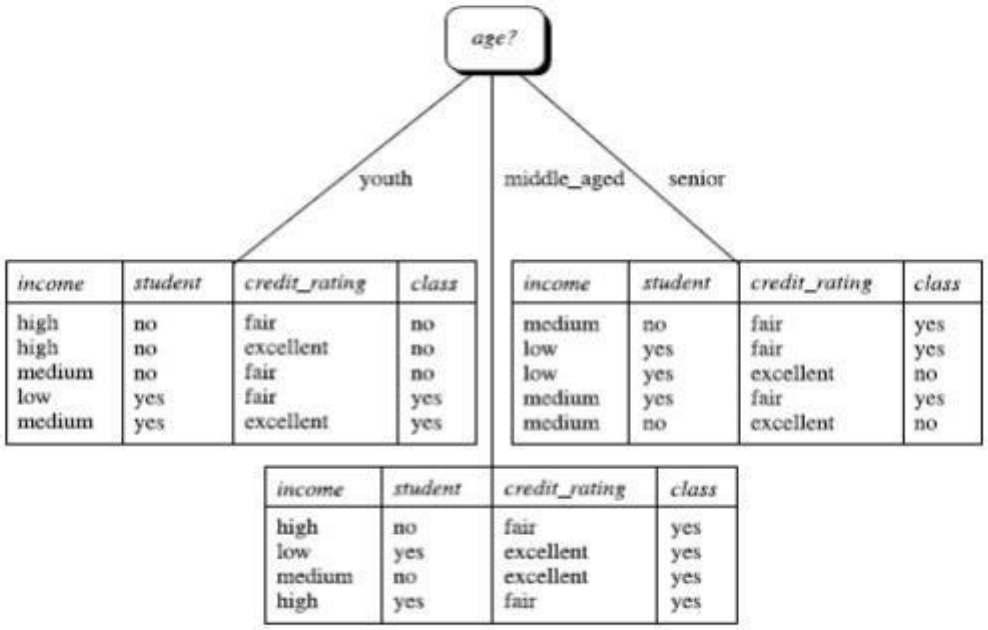


Figure 6.5 The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

Naïve Bayes Classifier:-

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

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

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

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:

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

Where,

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.

How to calculate these probabilities?

Bayes Theorem provides a way of calculating the posterior probability, P(H|X) from P(X|H), P(X) and P(H).

Naive Bayesian Classification:

1. Let a training data set D consists of tuples(many) and let X is one of them with n attributes A1, A2, A3,….., An and m associated class labels C1, C2, C3, C4,…, Cm.
2. Bayesian classifier predicts that tuple X belongs to the class Ci if and only if P(Ci|X)it is maximum among all i.e, we want to maximize P(Ci|X). The class Ci for which P(Ci|X) is maximized is called the maximum posteriori hypothesis.
3. Now the question arises how to maximize this ? we know P(Ci|X)=(P(X|Ci)*P(Ci))/P(X). Since P(X) is constant therefore we need to maximize P(X|Ci)*P(Ci) only.
4. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely(it means we have equal number of each class label), that is, P(C1)=P(C2)=….=P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)*P(Ci).
5. P(Ci)=|Ci,D|/|D| where |Ci,D| is the number of training tuples of class Ci in D.
6. For a given data set with many attributes it would be computationally expensive to calculate P(X|Ci)because we need to consider each attribute separately so we. “To reduce computation in evaluating P(X|Ci), the naive assumption of class-conditional independence is made. This presumes that the attributes’ values are conditionally independent of one another” P(X|Ci)=P(x1|Ci)*P(x2|Ci)*P(x3|Ci)=..=P(xn|Ci). It is actually the product of all the probabilities for n attributes.
7. Recall that here xk refers to the value of attribute Ak for tuple X.To compute P(X|Ci), we consider the following cases:

- If Ak is categorical, P(X|Ci) then is the number of tuples of class Ci in D having the value xk for Ak, divided by |Ci,D| the number of tuples of class Ci in D.
- If Ak is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean(u) and standard deviation(sd), defined by

for P(X|Ci) in the above equation put X =xk, u= mean of Ci ,and standard deviation= sd(Ci).

It was all the raw theory now consider an example for better understanding

Table 8.1 Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Suppose we have a tuple X =(age =youth, income =medium, student =yes, credit rating =fair) and we need to predict its class label (yes or no).P(Ci) i=1,2 the prior probability of each class, can be computed based on the training tuples:

Here we have x1=age ,x2=income, x3=student , x4=credit_rating

P(C1)=P(buys_computer =yes) =9/14 =0.643 (since total 9 rows of yes)

P(C2)=P(buys_computer =no) =5/14 =0.357.

To compute P(X|Ci), for i=1, 2, we compute the following conditional probabilities:

P(age =youth |buys_computer =yes) =2/9 =0.222 (it P(x1|C1)=prob(age=youth and buys_computer=yes)/prob(buys_computer=yes)=(2/14)/(9/14)=2/9)

P(age =youth | buys_computer =no) =3/5 =0.600 (it P(x1|C2))

P(income =medium | buys_computer =yes) =4/9 =0.444 (it P(x2|C1))

P(income =medium | buys_computer =no) =2/5 =0.400 (it P(x2|C2))

P(student =yes | buys_computer =yes) =6/9 =0.667

P(student =yes |buys_computer D=no) =1/5 =0.200

P(credit_rating =fair |buys_computer =yes) =6/9 =0.667

P(credit_rating =fair |buys_computer =no) =2/5 =0.400

Using these probabilities, we obtain P(X|buys_computer =yes)= P(age =youth |buys computer =yes)*

P(income =medium |buys_computer =yes)*P(student =yes |buys_computer =yes)* P(credit rating =fair

|buys_computer =yes)=0.222*0.444*0.667*0.667 =0.044.

Similarly, P(X|buys_computer =no)=0.600*0.400*0.200*0.400 =0.019.
To find the class, Ci , that maximizes P(X|Ci)*P(Ci), we compute

P(X|buys_computer =yes)*P(buys-computer =yes)= 0.044*0.643 =0.028
P(X|buys_computer =no)*P(buys_computer =no)=0.019*0.357 =0.007

Therefore, the naive Bayesian classifier predicts buys_computer =yes for tuple X.

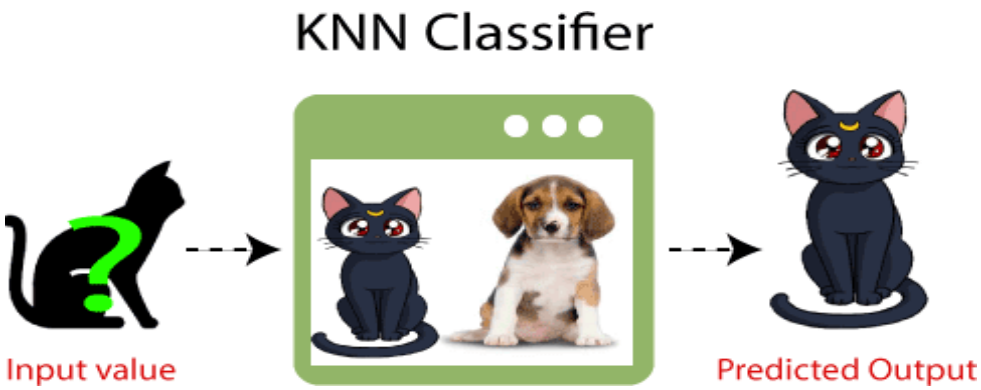
To avoid computing probability values of zero. Suppose that for the class buys computer =yes in some training database, D, containing 1000 tuples, we have 0 tuples with income =low, 990 tuples with income =medium, and 10 tuples with income =high. The probabilities of these events are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively. Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair, for each income-value pair. In this way, we instead obtain the following probabilities (rounded up to three decimal places):

1/1003=0.001 ,999/1003=0.988, 11 /1003=0.011 respectively. The “corrected” probability estimates are close to their “uncorrected”counterparts, yet the zero probability value is avoided.

K-Nearest Neighbor (KNN):-

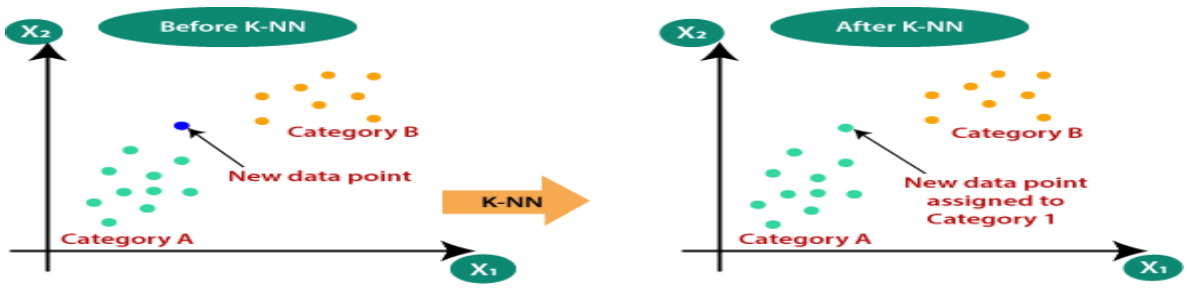
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm 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-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.
- **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.



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 x1, 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:

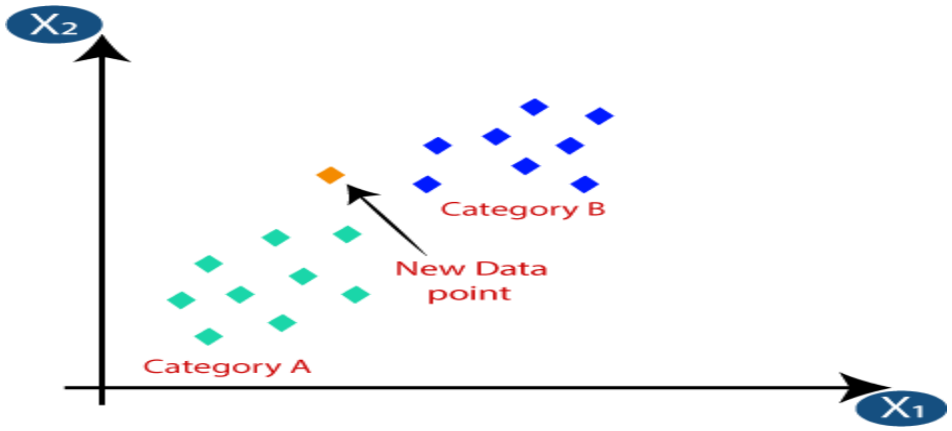


How does K-NN work?

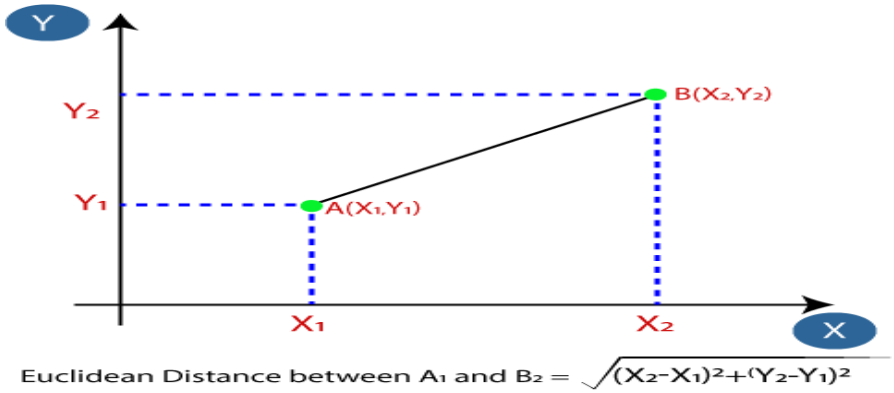
The K-NN working can be explained on the basis of the below algorithm:

- **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.

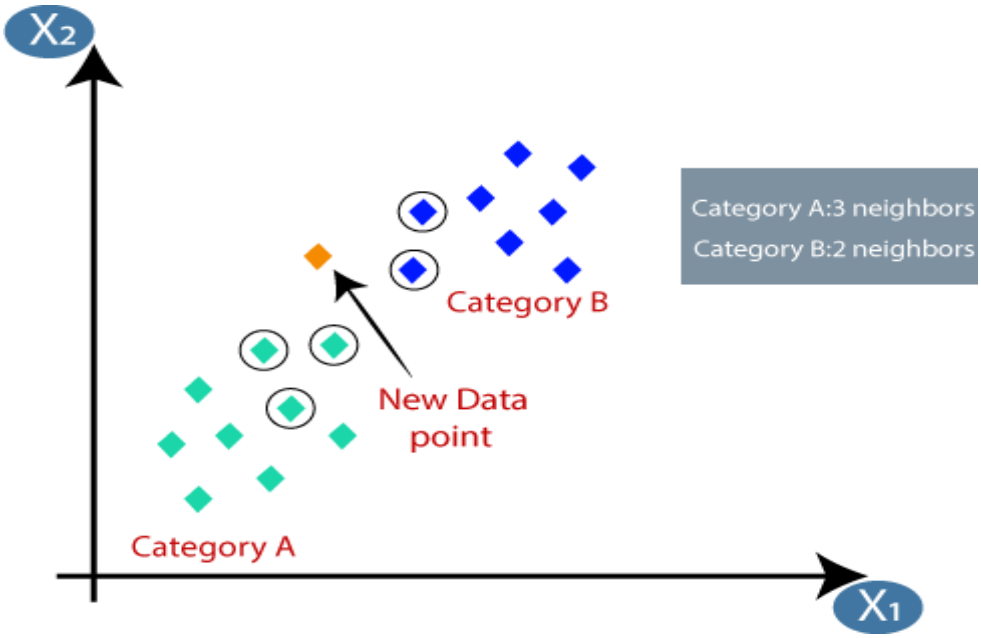
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Perceptron:-

Perceptron is a single layer neural network *and a multi-layer perceptron is called Neural Networks.*

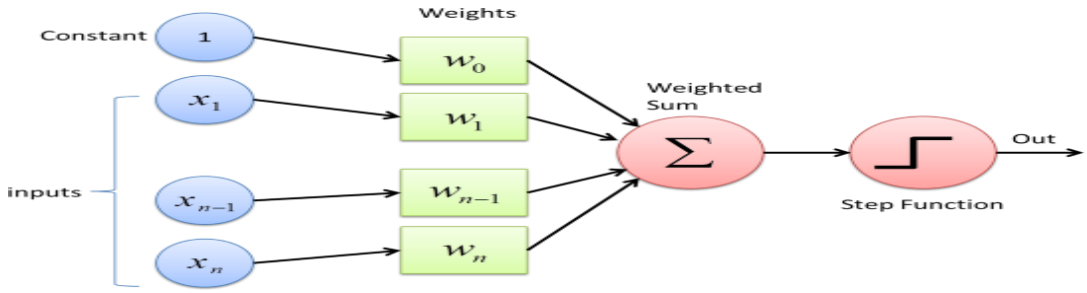
The **Perceptron** is a linear machine learning algorithm for binary classification tasks

Also, it is used in supervised learning. It helps to classify the given input data.

This algorithm enables neurons to learn and processes elements in the training set one at a time

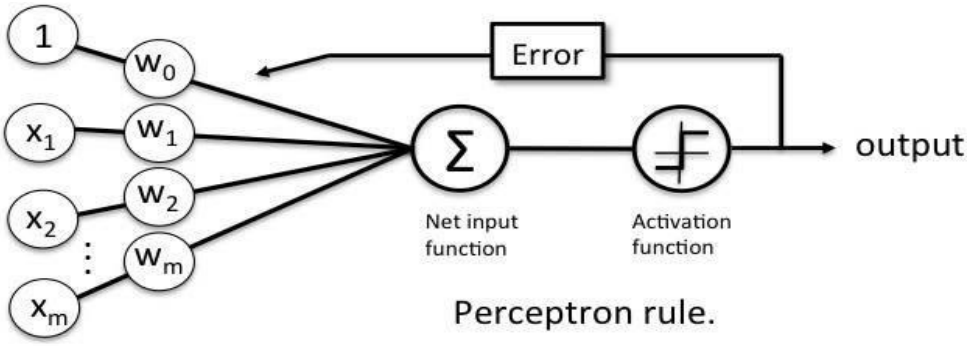
The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

This enables you to distinguish between the two linearly separable classes +1 and 0.



Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.



The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context of supervised learning and classification, this can then be used to predict the class of a sample.

Perceptron Function

Perceptron is a function that maps its input "x," which is multiplied with the learned weight coefficient; an output value "f(x)"is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the equation given above:

- "w" = vector of real-valued weights

- "b" = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
- "x" = vector of input x values

$$\sum_{i=1}^{m} w_i x_i$$

- "m" = number of inputs to the Perceptron

The output can be represented as "1" or "0." It can also be represented as "1" or "-1" depending on which activation function is used.

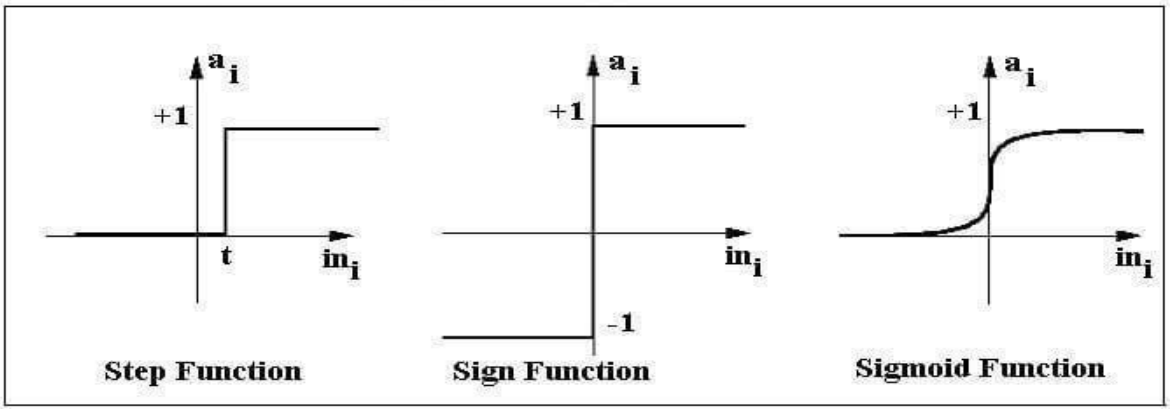
Inputs of a Perceptron

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The image below shows a Perceptron with a Boolean output.
A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function "∑" multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Activation Functions of Perceptron

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



For example:
If ∑ wixi> 0 => then final output "o" = 1 (issue bank loan)
Else, final output "o" = -1 (deny bank loan)

Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Output of Perceptron

Perceptron with a Boolean output:
Inputs: x1…xn
Output: o(x1….xn)

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

Error in Perceptron

In the Perceptron Learning Rule, the predicted output is compared with the known output. If it does not match, the error is propagated backward to allow weight adjustment to happen.

Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated. The weights of the model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process is repeated for all examples in the training dataset, called an epoch. This process of updating the model using examples is then repeated for many epochs.

Model weights are updated with a small proportion of the error each batch, and the proportion is controlled by a hyperparameter called the learning rate, typically set to a small value. This is to ensure learning does not occur too quickly, resulting in a possibly lower skill model, referred to as premature convergence of the optimization (search) procedure for the model weights.

Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed.

The initial values for the model weights are set to small random values. The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using heuristics or hyperparameter tuning.

Example -

$\overline{w}$ – weight vector

α – learning rate

t - target (0 (or) 1)

$\overline{x}$ - input vector

p($\overline{x}$)- perceptron output (0 (or) 1)

$w_{new} = w_{old} + \alpha\ (t-p(x))\ x_i$

assume α=0.1

$\overline{x}$	T	W_1	W_2	W_0
X_1, X_2				
Start		0	0	0
1, 1	1	0.1	0.1	0.1
2, 0.4	0	−0.1	0.06	0
0, 1	1	−0.1	0.06	0

Logistic regression:-

Logistic Regression is used when the dependent variable(target) is categorical.

It is the go-to method for binary classification problems (problems with two class values).

For example

- To predict whether an email is spam (1) or (0)
- Whether the tumor is malignant (1) or not (0)

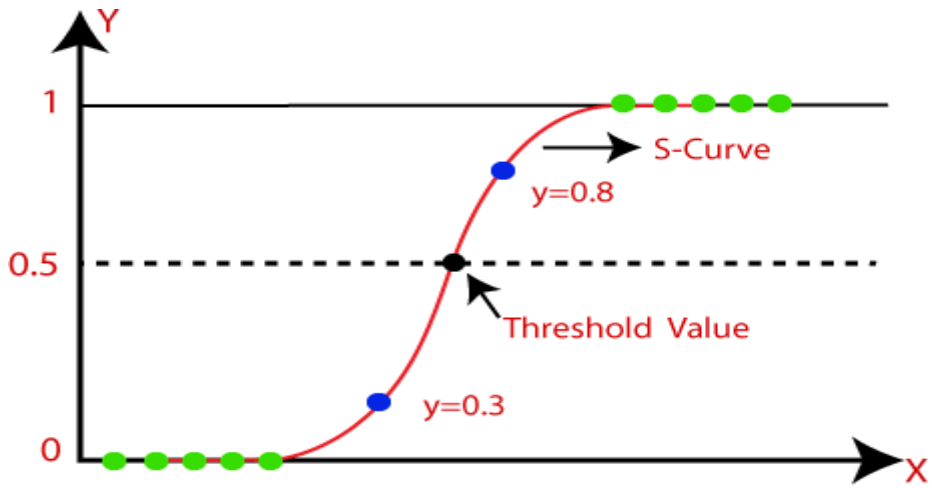
Logistic Function -

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of

the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)



Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

$$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

Logistic Regression Predicts Probabilities

Logistic regression models the probability of the default class (e.g. the first class).

For example, if we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

P(sex=male|height)

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

P(X) = P(Y=1|X)

Note that the probability prediction must be transformed into a binary values (0 or 1) in order to actually make a probability prediction.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

p(X) = e^(b0 + b1*X) / (1 + e^(b0 + b1*X))

Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data.

The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class.

It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data.

Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

Let's make this concrete with a specific example.

Let's say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.

We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150). We will use EXP() for e, because that is what you can use if you type this example into your spreadsheet:

$$y = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$$

$$y = \exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))$$

$$y = 0.0000453978687$$

Or a probability of near zero that the person is a male.

In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

0 if p(male) < 0.5

1 if p(male) >= 0.5

Now that we know how to make predictions using logistic regression, let's look at how we can prepare our data to get the most from the technique.

Random Forest:-

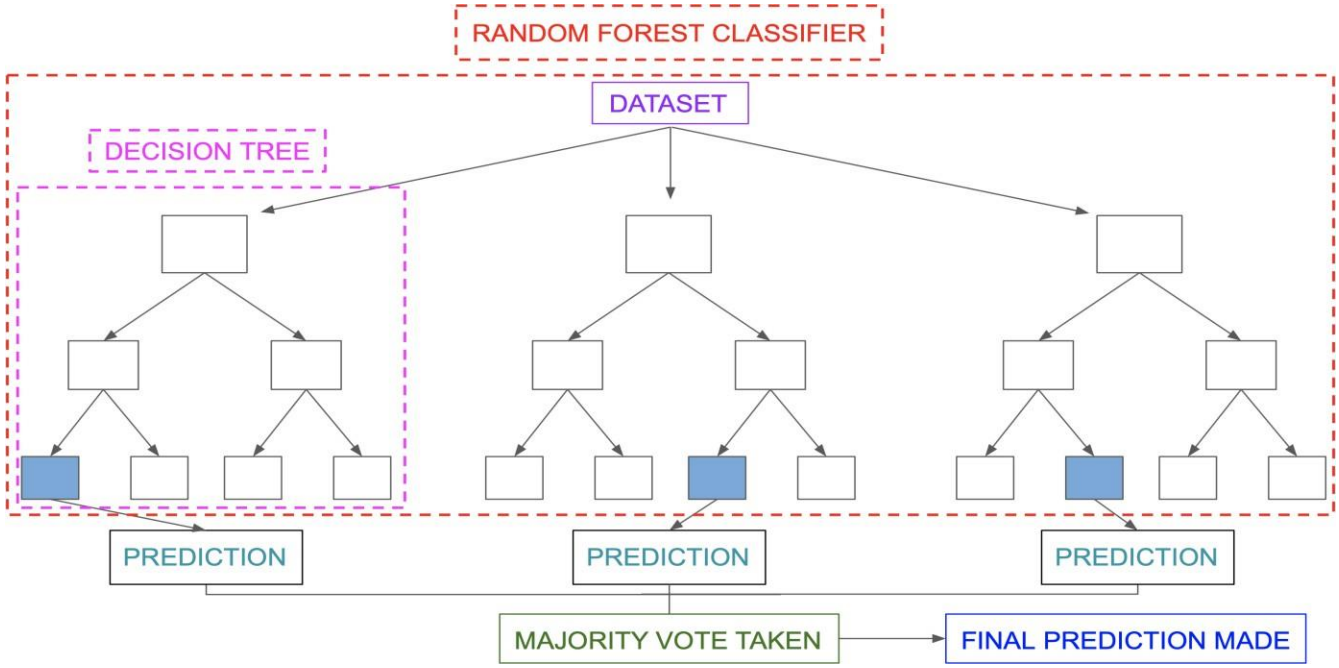
Random forest is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random forest has a variety of applications, such as recommendation engines, image classification and feature selection.

How does the algorithm work?

It works in four steps:

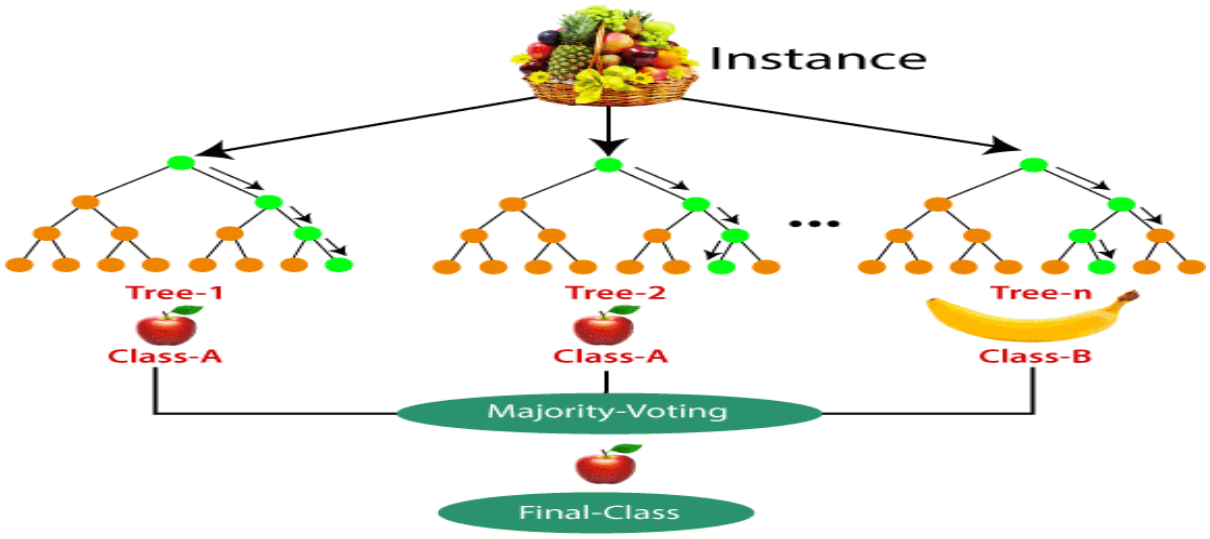
1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.



Example -

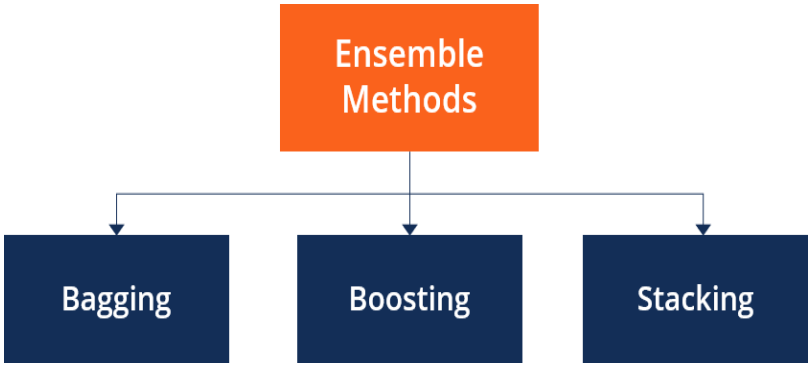
Let's take an example of a training dataset consisting of various fruits such as bananas, apples, pineapples, and mangoes. The random forest classifier divides this dataset into subsets. These subsets are given to every decision tree in the random forest system. Each decision tree produces its specific output. For example, the prediction for trees 1 and 2 is *apple*.

Another decision tree (n) has predicted *banana* as the outcome. The random forest classifier collects the majority voting to provide the final prediction. The majority of the decision trees have chosen *apple* as their prediction. This makes the classifier choose *apple* as the final prediction.



Ensemble methods:-

Ensemble method is a machine learning technique that combines several base models in order to produce one optimal predictive model. Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly.



Bagging: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.

Boosting: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Bagging -

Bootstrap Aggregating also know as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

Description of the Technique

Suppose a set D of d tuples, at each iteration i, a training set Di of d tuples is sampled with replacement from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D < i. Each classifier Mi returns its class prediction. The bagged classifier M* counts the votes and assigns the class with the most votes to X (unknown sample).

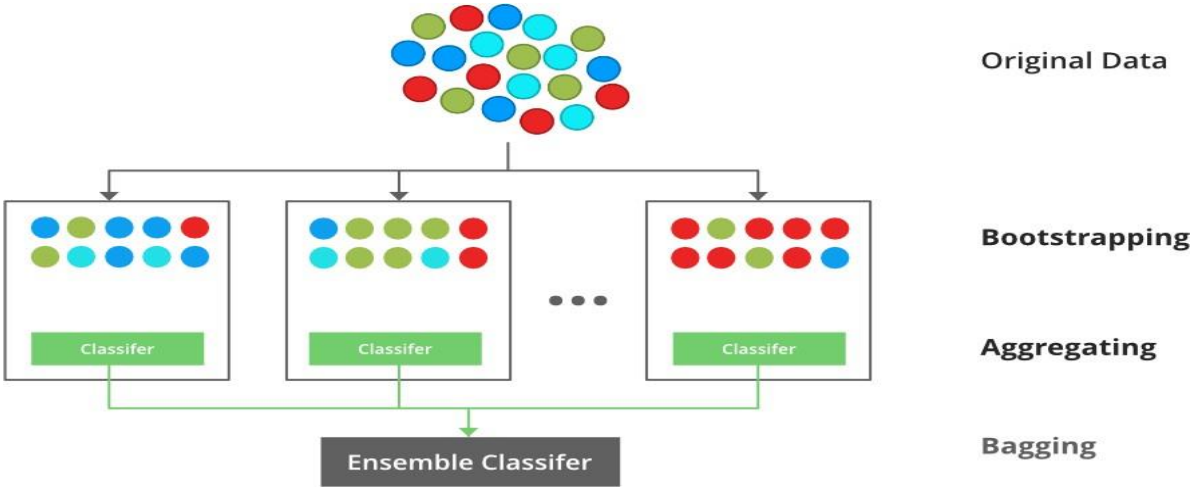
Implementation Steps of Bagging

Step 1: Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.

Step 2: A base model is created on each of these subsets.

Step 3: Each model is learned in parallel from each training set and independent of each other.

Step 4: The final predictions are determined by combining the predictions from all the models.



Example –

Random Forest model

Boosting -

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

Boosting Algorithms

1 Initialise the dataset and assign equal weight to each of the data point.

2 Provide this as input to the model and identify the wrongly classified data points.

3 Increase the weight of the wrongly classified data points.

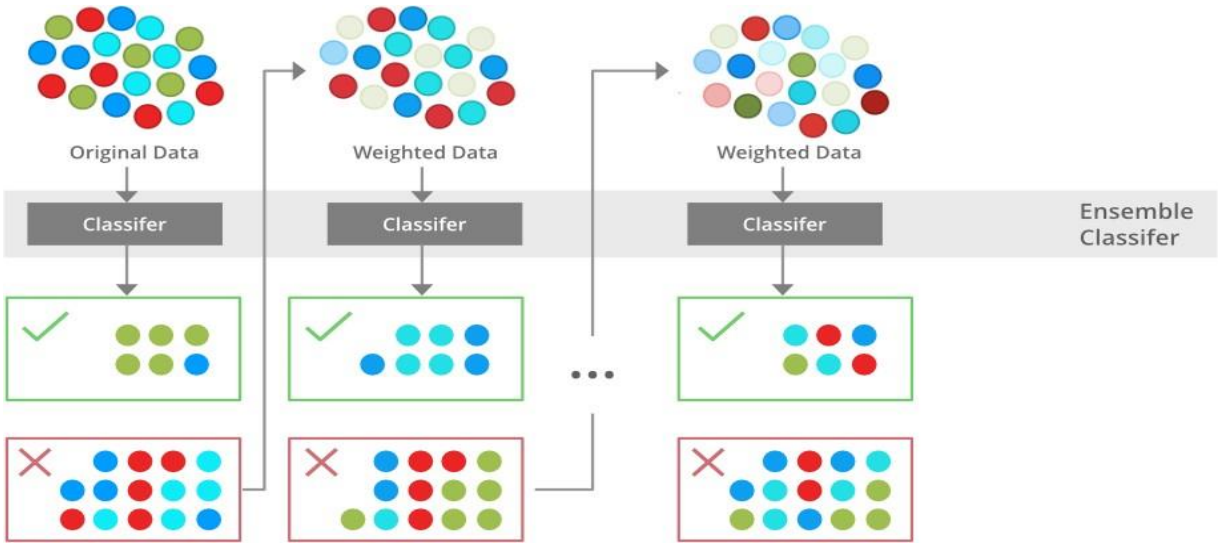
4 if (got required results)

Goto step 5

else

Goto step 2

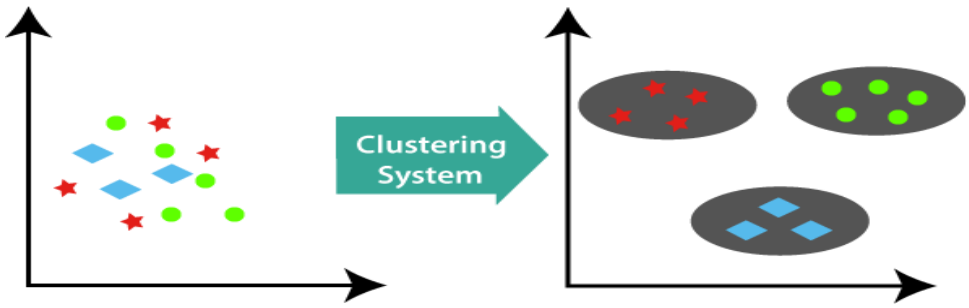
End



Clustering

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.



Partitioning-based clustering methods:-

K-means clustering is a partitioning method and as anticipated, this method decomposes a dataset into a set of disjoint clusters. Given a dataset, a partitioning method constructs several partitions of this data, with each partition representing a cluster. These methods relocate instances by moving them from one cluster to another, starting from an initial partitioning.

Partitioning Algorithms used in Clustering –

K-Means Clustering Algorithm (A centroid based Technique): -

It is one of the most commonly used algorithm for partitioning a given data set into a set of k groups (i.e. *k clusters*), where k represents the number of groups. It classifies objects in multiple groups (i.e., clusters), such that objects within the same cluster are as similar as possible (i.e., high *intra-class similarity*), whereas objects from different clusters are as dissimilar as possible (i.e., low *inter-class similarity*). In k-means clustering, each cluster is represented by its center (i.e, *centroid*) which corresponds to the mean of points assigned to the cluster. The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized.

Steps involved in K-Means Clustering:

1. The first step when using k-means clustering is to indicate the number of clusters (k) that will be generated in the final solution.
2. The algorithm starts by randomly selecting k objects from the data set to serve as the initial centers for the clusters. The selected objects are also known as cluster means or centroids.
3. Next, each of the remaining objects is assigned to it's closest centroid, where closest is defined using the Euclidean distance between the object and the cluster mean. This step is called "cluster assignment step".

4. After the assignment step, the algorithm computes the new mean value of each cluster. The term cluster "centroid update" is used to design this step. Now that the centers have been recalculated, every observation is checked again to see if it might be closer to a different cluster. All the objects are reassigned again using the updated cluster means.
5. The cluster assignment and centroid update steps are iteratively repeated until the cluster assignments stop changing (i.e until *convergence* is achieved). That is, the clusters formed in the current iteration are the same as those obtained in the previous iteration.

Step 1 ->

Subject	A	B
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Step 2 ->

	Individual	Mean Vector (centroid)
Group 1	1	(1.0, 1.0)
Group 2	4	(5.0, 7.0)

Step 3 ->

	Cluster 1		Cluster 2	
Step	Individual	Mean Vector (centroid)	Individual	Mean Vector (centroid)
1	1	(1.0, 1.0)	4	(5.0, 7.0)
2	1, 2	(1.2, 1.5)	4	(5.0, 7.0)
3	1, 2, 3	(1.8, 2.3)	4	(5.0, 7.0)
4	1, 2, 3	(1.8, 2.3)	4, 5	(4.2, 6.0)
5	1, 2, 3	(1.8, 2.3)	4, 5, 6	(4.3, 5.7)
6	1, 2, 3	(1.8, 2.3)	4, 5, 6, 7	(4.1, 5.4)

Step 4 ->

	Individual	Mean Vector (centroid)
Cluster 1	1, 2, 3	(1.8, 2.3)
Cluster 2	4, 5, 6, 7	(4.1, 5.4)

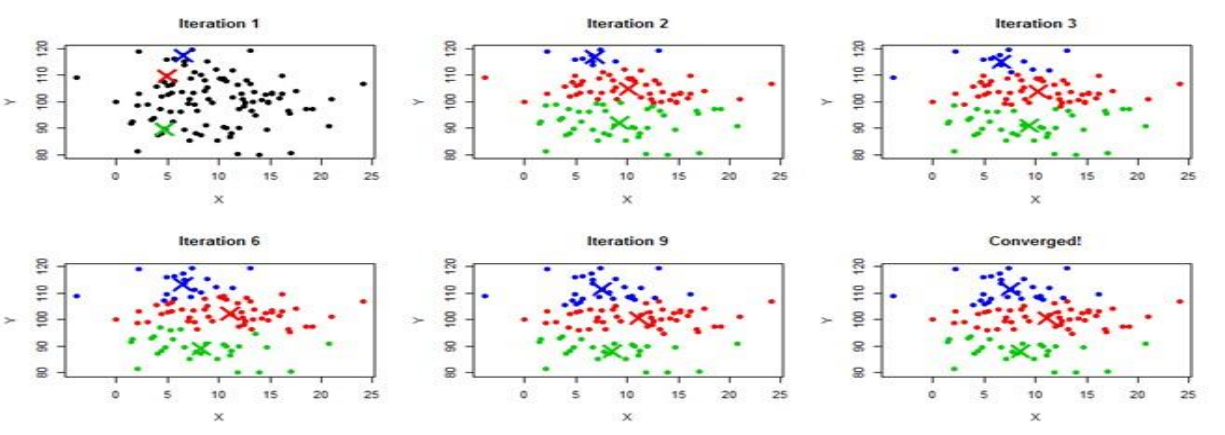
Step 5 ->

Individual	Distance to mean (centroid) of Cluster 1	Distance to mean (centroid) of Cluster 2
1	1.5	5.4
2	0.4	4.3
3	2.1	1.8
4	5.7	1.8
5	3.2	0.7
6	3.8	0.6
7	2.8	1.1

Step 6 ->

	Individual	Mean Vector (centroid)
Cluster 1	1, 2	(1.3, 1.5)
Cluster 2	3, 4, 5, 6, 7	(3.9, 5.1)

Example for K-Means Clustering



Hierarchical Clustering methods:-

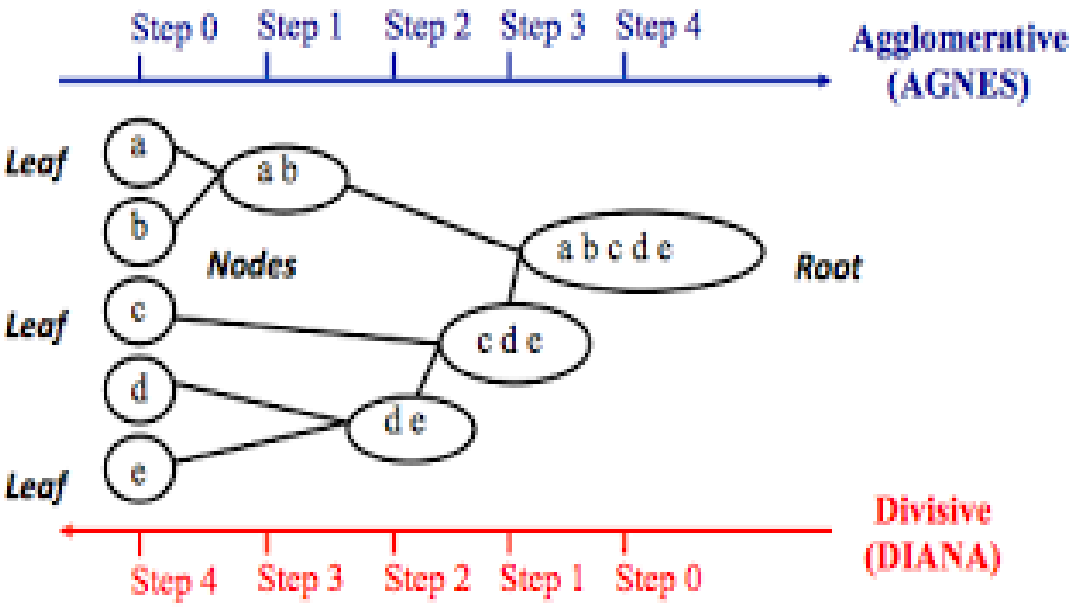
Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

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**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

Hierarchical clustering can be divided into two main types: *agglomerative* and *divisive*.

1. **Agglomerative clustering:** It's also known as AGNES (Agglomerative Nesting). It works in a bottom-up manner. That is, each object is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This procedure is iterated until all points are member of just one single big cluster (root) (see figure below). The result is a tree which can be plotted as a dendrogram.
2. **Divisive hierarchical clustering:** It's also known as DIANA (Divise Analysis) and it works in a top-down manner. The algorithm is an inverse order of AGNES. It begins with the root, in which all objects are included in a single cluster. At each step of iteration, the most heterogeneous cluster is divided into two. The process is iterated until all objects are in their own cluster (see figure below).



HIERARCHICAL CLUSTERING ALGORITHMS -

1. Balanced Iterative Reducing and Clustering Using Hierarchies Algorithm (BIRCH)
2. Robust Clustering Using Links (ROCK).
3. CHAMELEON Algorithm

Regression:-

It falls under supervised learning wherein the algorithm is trained with both input features and output labels.

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, 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.

In Regression, we plot a graph between the variables which best fit the given data points.

In naïve words, ***"Regression shows a line or curve that passes through all the data points on a target-predictor graph in such a way that the vertical distance between the data points and the regression line is minimum."***

Types of Regression models

1. Linear Regression
2. Polynomial Regression
3. Logistics Regression

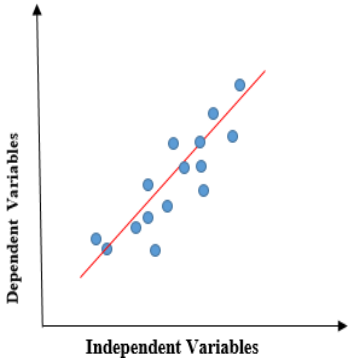
Linear Regression:

Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression.

If there is a single input variable (x), such linear regression is called ***simple linear regression****.*

And if there is more than one input variable, such linear regression is called ***multiple linear regression*****.**

The linear regression model gives a sloped straight line describing the relationship within the variables.



The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (**independent variable**) increases, the value of y (**dependent variable**) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best.

To calculate best-fit line linear regression uses a traditional slope-intercept form.

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

y= Dependent Variable.

x= Independent Variable.

a0= intercept of the line.

a1 = Linear regression coefficient.

Cost function -

The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points.
Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to find the accuracy of the **mapping function** that maps the input variable to the output variable. This mapping function is also known as **the Hypothesis function**.
In Linear Regression, **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the predicted values and actual values.
By simple linear equation y=mx+b we can calculate MSE as:
Let's y = actual values, y_i = predicted values

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Model parameters ***xi, b (a_0,a_1)*** can be manipulated to minimize the cost function. These parameters can be determined using the gradient descent method so that the cost function value is minimum.

Residuals -

The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.

- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
  Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any

correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Cross-validation:-

In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the cross-validation technique.

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.

The three steps involved in cross-validation are as follows:

Reserve some portion of sample data-set.

Using the rest data-set train the model.

Test the model using the reserve portion of the data-set.

Methods of Cross Validation:

Validation -

In this method, we perform training on the 50% of the given data-set and rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

LOOCV (Leave One Out Cross Validation) -

In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also.

An advantage of using this method is that we make use of all data points and hence it is low bias.

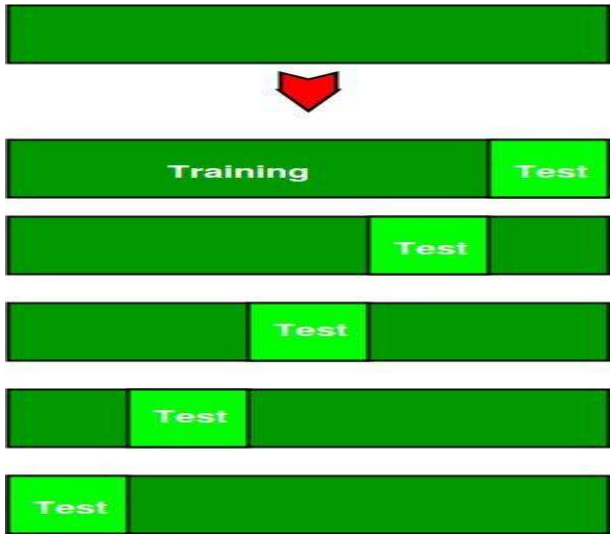
The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

K-Fold Cross Validation -

In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Example

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so on.



Total instances: 25

Value of k : 5

No. Iteration	Training set observations	Testing set observations
1	[5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[0 1 2 3 4]
2	[0 1 2 3 4 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[5 6 7 8 9]
3	[0 1 2 3 4 5 6 7 8 9 15 16 17 18 19 20 21 22 23 24]	[10 11 12 13 14]
4	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 20 21 22 23 24]	[15 16 17 18 19]
5	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]	[20 21 22 23 24]

Comparison of train/test split to cross-validation

Advantages of train/test split:

This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.

Advantages of cross-validation:

More accurate estimate of out-of-sample accuracy.

More "efficient" use of data as every observation is used for both training and testing.

Class-imbalance – ways of handling:-

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally.
For example, you may have a 2-class (binary) classification problem with 100 instances (rows). A total of 80 instances are labeled with Class-1 and the remaining 20 instances are labeled with Class-2.
This is an imbalanced dataset and the ratio of Class-1 to Class-2 instances is 80:20 or more concisely 4:1.
You can have a class imbalance problem on two-class classification problems as well as multi-class classification problems.
Most classification data sets do not have exactly equal number of instances in each class, but a small difference often does not matter.
However, most machine learning algorithms do not work very well with imbalanced datasets. The following seven techniques can help you, to train a classifier to detect the abnormal class.

1. Use the right evaluation metrics

Applying inappropriate evaluation metrics for model generated using imbalanced data can be dangerous. Imagine our training data is the one illustrated in graph above. If accuracy is used to measure the goodness of a model, a model which classifies all testing samples into "0" will have an excellent accuracy (99.8%), but obviously, this model won't provide any valuable information for us.

In this case, other alternative evaluation metrics can be applied such as:

Precision/Specificity: how many selected instances are relevant.

Recall/Sensitivity: how many relevant instances are selected.

F1 score: harmonic mean of precision and recall.

MCC: correlation coefficient between the observed and predicted binary classifications.

AUC: relation between true-positive rate and false positive rate.

2. Resample the training set

Apart from using different evaluation criteria, one can also work on getting different dataset. Two approaches to make a balanced dataset out of an imbalanced one are under-sampling and over-sampling.

Under-sampling

Under-sampling balances the dataset by reducing the size of the abundant class. This method is used when quantity of data is sufficient. By keeping all samples in the rare class and randomly selecting an equal number of samples in the abundant class, a balanced new dataset can be retrieved for further modelling.

Over-sampling

On the contrary, oversampling is used when the quantity of data is insufficient. It tries to balance dataset by increasing the size of rare samples. Rather than getting rid of abundant samples, new rare samples are generated by using e.g. repetition, bootstrapping or SMOTE (Synthetic Minority Over-Sampling Technique) [1].

Note that there is no absolute advantage of one resampling method over another. Application of these two methods depends on the use case it applies to and the dataset itself. A combination of over- and under-sampling is often successful as well.

3. Use K-fold Cross-Validation in the right way

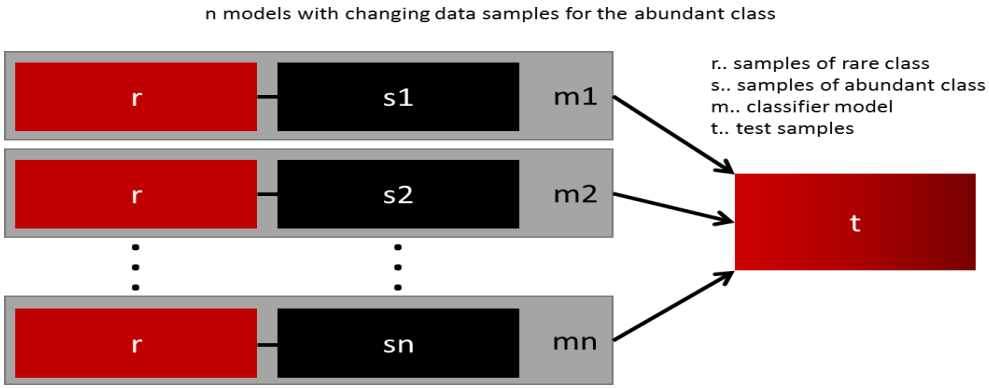
It is noteworthy that cross-validation should be applied properly while using over-sampling method to address imbalance problems.

Keep in mind that over-sampling takes observed rare samples and applies bootstrapping to generate new random data based on a distribution function. If cross-validation is applied after over-sampling, basically what we are doing is overfitting our model to a specific artificial bootstrapping result. That is why cross-validation should always be done before over-sampling the data, just as how feature selection should be implemented. Only by resampling the data repeatedly, randomness can be introduced into the dataset to make sure that there won't be an overfitting problem.

4. Ensemble different resampled datasets

The easiest way to successfully generalize a model is by using more data. The problem is that out-of-the-box classifiers like logistic regression or random forest tend to generalize by discarding the rare

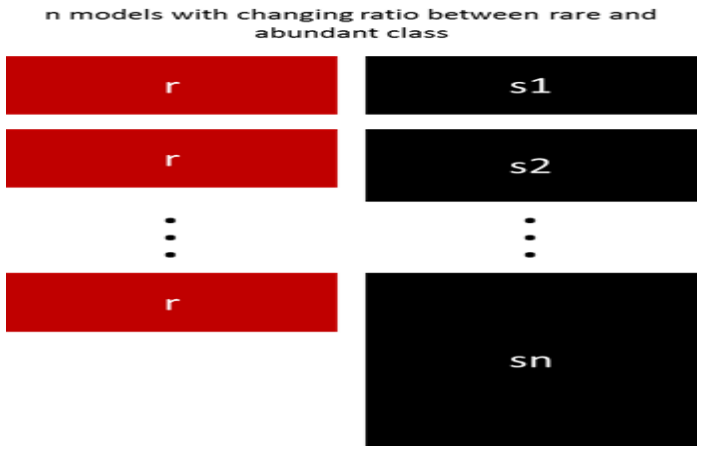
class. One easy best practice is building n models that use all the samples of the rare class and n-differing samples of the abundant class. Given that you want to ensemble 10 models, you would keep e.g. the 1.000 cases of the rare class and randomly sample 10.000 cases of the abundant class. Then you just split the 10.000 cases in 10 chunks and train 10 different models.



This approach is simple and perfectly horizontally scalable if you have a lot of data, since you can just train and run your models on different cluster nodes. Ensemble models also tend to generalize better, which makes this approach easy to handle.

5. Resample with different ratios

The previous approach can be fine-tuned by playing with the ratio between the rare and the abundant class. The best ratio heavily depends on the data and the models that are used. But instead of training all models with the same ratio in the ensemble, it is worth trying to ensemble different ratios. So if 10 models are trained, it might make sense to have a model that has a ratio of 1:1 (rare:abundant) and another one with 1:3, or even 2:1. Depending on the model used this can influence the weight that one class gets.



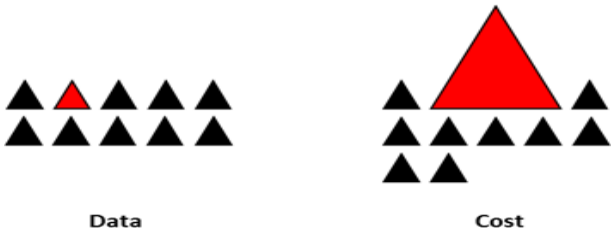
6. Cluster the abundant class

An elegant approach was proposed by Sergey on Quora [2]. Instead of relying on random samples to cover the variety of the training samples, he suggests clustering the abundant class in r groups, with r being the number of cases in r. For each group, only the medoid (centre of cluster) is kept. The model is then trained with the rare class and the medoids only.

7. Design your own models

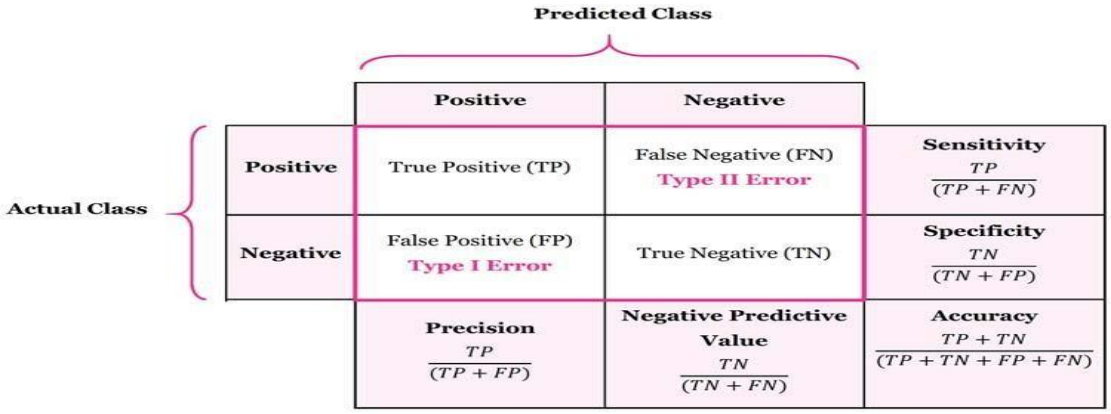
All the previous methods focus on the data and keep the models as a fixed component. But in fact, there is no need to resample the data if the model is suited for imbalanced data. The famous XGBoost is already a good starting point if the classes are not skewed too much, because it internally takes care that the bags it trains on are not imbalanced. But then again, the data is resampled, it is just happening secretly.

By designing a cost function that is penalizing wrong classification of the rare class more than wrong classifications of the abundant class, it is possible to design many models that naturally generalize in favour of the rare class. For example, tweaking an SVM to penalize wrong classifications of the rare class by the same ratio that this class is underrepresented.



Confusion Matrix:-

The million-dollar question – what, after all, is a confusion matrix?
A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.
For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:



Let's decipher the matrix:

- The target variable has two values: **Positive** or **Negative**
- The **columns** represent the **actual values** of the target variable
- The **rows** represent the **predicted values** of the target variable

But wait – what's TP, FP, FN and TN here? That's the crucial part of a confusion matrix. Let's understand each term below.
Understanding True Positive, True Negative, False Positive and False Negative in a Confusion Matrix

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value
- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value
- Also known as the **Type 2 error**

Precision - Precision tells us how many of the correctly predicted cases actually turned out to be positive.

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

This would determine whether our model is reliable or not.

Recall - Recall tells us how many of the actual positive cases we were able to predict correctly

with our model.

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

F1-Score -

In practice, when we try to increase the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value:

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

But there is a catch here. The interpretability of the F1-score is poor. This means that we don't know what our classifier is maximizing – precision or recall? So, we use it in combination with other evaluation metrics which gives us a complete picture of the result.

This is how we'll calculate the accuracy:

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

Evaluation metrics:-

Evaluation metrics are tied to machine learning tasks. There are different metrics for the tasks of classification, regression, ranking, clustering, topic modeling, etc. Some metrics, such as precision-recall, are useful for multiple tasks. Classification, regression, and ranking are examples of supervised learning, which constitutes a majority of machine learning applications.

1 Model Accuracy -

Model accuracy in terms of classification models can be defined as the ratio of correctly classified samples to the total number of samples:

Accuracy = Number of correct predictions

Total number of predictions

Or for binary classification models, the accuracy can be defined as:

Accuracy = TP +TN

TP + TN + FP + FN

Predicted

Actual		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

Confusion Matrix

True Positive (TP) — A true positive is an outcome where the model *correctly* predicts the positive class.

True Negative (TN)—A true negative is an outcome where the model *correctly* predicts the negative class.

False Positive (FP)—A false positive is an outcome where the model *incorrectly* predicts the positive class.

False Negative (FN)—A false negative is an outcome where the model *incorrectly* predicts the negative class.

2 Precision and Recall -

In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class).

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

Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been).

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

3 F1 Score -

The F1 score is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$

4 ROC Curve -

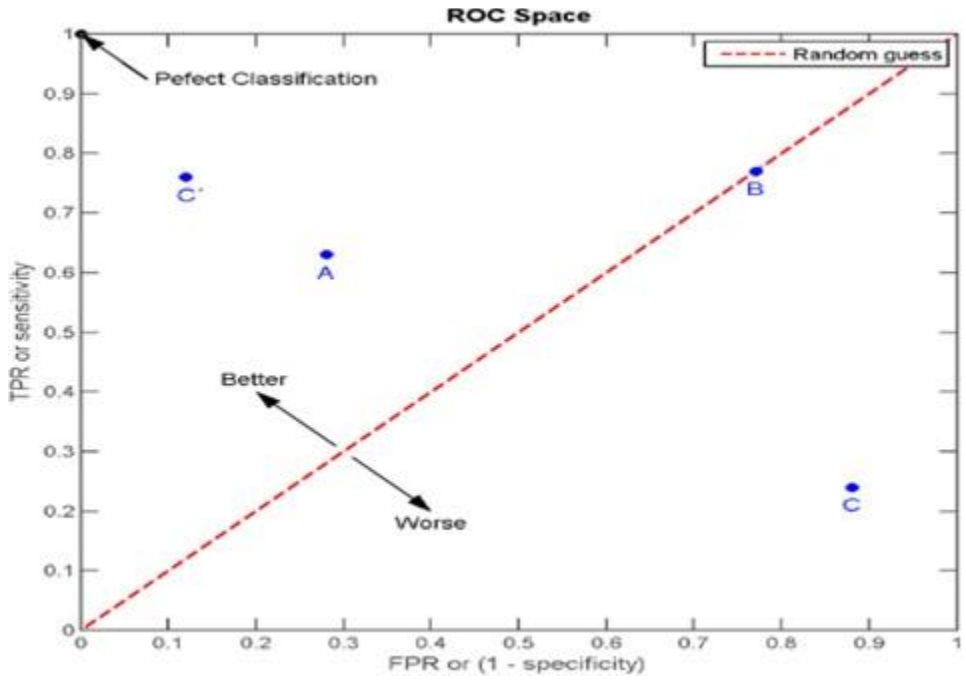
A **receiver operating characteristic** curve, or **ROC** curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true positive rate is also known as sensitivity, recall, or probability of detection in machine learning. The false-positive rate is also known as the fall-out, or probability of false alarm.

$$\text{True Positive Rate (TPR)} = \frac{TP}{(TP+FN)}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{(FP+TN)}$$

`



5 AUC — Area under the ROC Curve -

An ROC curve is a two-dimensional depiction of classifier performance. To compare classifiers, we may want to reduce ROC performance to a single scalar value representing expected performance. A common method is to calculate the area under the ROC curve, abbreviated AUC.
Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0. However, because random guessing produces the diagonal line between (0, 0) and (1, 1), which has an area of 0.5, **no realistic classifier should have an AUC less than 0.5**