UNIT II:

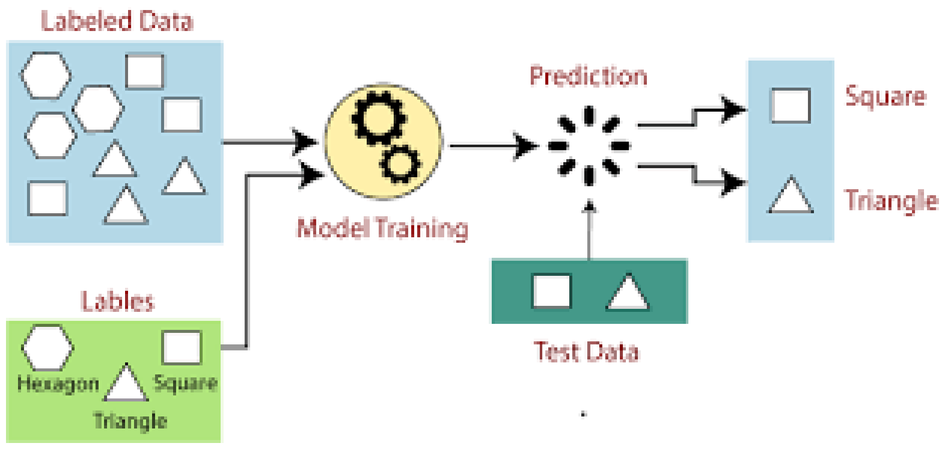
Supervised Learning(Regression/Classification):Basic Methods: Distance based Methods, Nearest Neighbours, Decision Trees, Naive Bayes, **Linear Models:** Linear Regression, Logistic Regression, Generalized Linear Models, Support Vector Machines, **Binary Classification:** Multiclass/Structured outputs, MNIST, Ranking.

SUPERVISED LEARNING

- Supervised Learning is a type of machine learning in which machines are trained using well labelled trained data, and on the basis of this data, machines predict the output. The labelled data means that the input data is already tagged with correct output.
- The aim of supervised learning is to find a mapping function to map the input variable(x) with the output variable(y).
- In the real world, supervised learning can be used for Risk assessment, Image classification, Fraud detection, spam filtering etc.

How does Supervised Learning Works?

- In supervised learning, models are trained using labelled dataset, where the model learns about each type of data.
- Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.
- The working of Supervised learning can be easily understood by the below example and diagram:



Key Points:

- Supervised learning involves training a machine from labelled data.
- Labeled data consists of examples with the correct answer or classification.
- The machine learns the relationship between inputs and outputs.
- The trained machine can then make predictions on new, unlabelled data.

Types of Supervised Learning:

- Supervised learning is classified into two categories of algorithms:

1. Regression

- Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

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

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Applications of Supervised learning:
Supervised learning can be used to solve a wide variety of problems, including:

- **Spam filtering:** Supervised learning algorithms can be trained to identify and classify spam emails based on their content, helping users avoid unwanted messages.
- **Image classification:** Supervised learning can automatically classify images into different categories, such as animals, objects, or scenes, facilitating tasks like image search, content moderation, and image-based product recommendations.
- **Medical diagnosis:** Supervised learning can assist in medical diagnosis by analyzing patient data, such as medical images, test results, and patient history, to identify patterns that suggest specific diseases or conditions.
- **Fraud detection:** Supervised learning models can analyze financial transactions and identify patterns that indicate fraudulent activity, helping financial institutions prevent fraud and protect their customers.
- **Natural language processing (NLP):** Supervised learning plays a crucial role in NLP tasks, including sentiment analysis, machine translation, and text summarization, enabling machines to understand and process human language effectively.

Advantages of Supervised learning:

- Supervised learning allows collecting data and produces data output from previous experiences.
- Helps to optimize performance criteria with the help of experience.
- Supervised machine learning helps to solve various types of real-world computation problems.
- It performs classification and regression tasks.
- It allows estimating or mapping the result to a new sample.
- We have complete control over choosing the number of classes we want in the training data.

Disadvantages of Supervised learning:

- Classifying big data can be challenging.
- Training for supervised learning needs a lot of computation time. So, it requires a lot of time.
- Supervised learning cannot handle all complex tasks in Machine Learning.
- Computation time is vast for supervised learning.
- It requires a labelled data set.
- It requires a training process.

BASIC METHODS:

1. DISTANCE BASED METHODS:

- Distance metrics are a key part of several machine learning algorithms.
- These distance metrics are used in both supervised and unsupervised learning, generally to calculate the similarity between data points.
- An effective distance metric improves the performance of our machine learning model, whether that's for classification tasks or clustering.

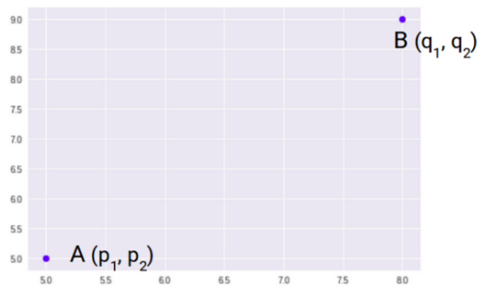
Types of Distance Metrics in Machine Learning

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance

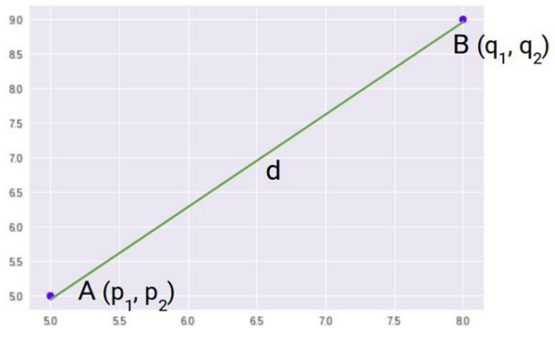
1. Euclidean Distance

- Euclidean Distance represents the shortest distance between two vectors. It is the square root of the sum of squares of differences between corresponding elements.
- The Euclidean distance metric corresponds to the L2-norm of a difference between vectors and vector spaces. The cosine similarity is proportional to the dot product of two vectors and inversely proportional to the product of their magnitudes.

Most machine learning algorithms, including K-Means use this distance metric to measure the similarity between observations. Let's say we have two points, as shown below:



So, the Euclidean Distance between these two points, A and B, will be:



Formula for Euclidean Distance

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

We use this formula when we are dealing with 2 dimensions. We can generalize this for an n-dimensional space as:

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

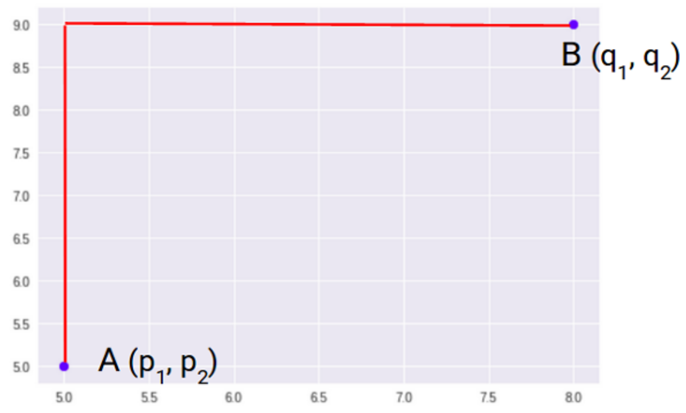
Where,

- n = number of dimensions
- pi, qi = data points

2. Manhattan Distance:

- Manhattan Distance is the sum of absolute differences between points across all the dimensions.

We can represent Manhattan Distance as:



Formula for Manhattan Distance

Since the above representation is 2 dimensional, to calculate Manhattan Distance, we will take the sum of absolute distances in both the x and y directions. So, the Manhattan distance in a 2-dimensional space is given as:

$$d = |p_1 - q_1| + |p_2 - q_2|$$

And the generalized formula for an n-dimensional space is given as:

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

Where,

- n = number of dimensions
- pi, qi = data points

Note that **Manhattan Distance is also known as city block distance.**

3. **Minkowski Distance:**

Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.

Formula for Minkowski Distance

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

Here, p represents the order of the norm.

4. **Hamming Distance:**

- Hamming Distance measures the similarity between two strings of the same length.
- The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.
- Let’s understand the concept using an example. Let’s say we have two strings:

“euclidean” and **“manhattan”**

- Since the length of these strings is equal, we can calculate the Hamming Distance.
- We will go character by character and match the strings. The first character of both the strings (e and m, respectively) is different.
- Similarly, the second character of both the strings (u and a) is different. and so on.
- Look carefully – seven characters are different, whereas two characters (the last two characters) are similar:

euclidean and manhattan

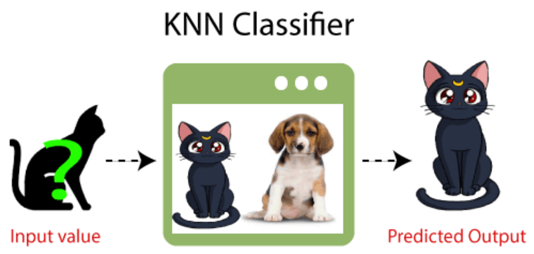
- Hence, the Hamming Distance here will be 7.
- Note that the larger the Hamming Distance between two strings, the more dissimilar those strings will be (and vice versa).

2. **NEAREST NEIGHBOURS:**

- 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** (that do not make strong assumptions about the form of the mapping function), 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.

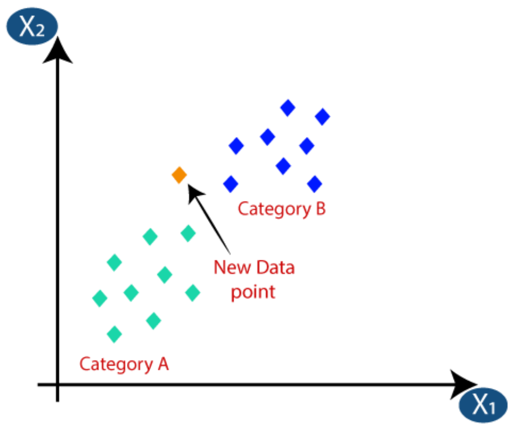
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the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



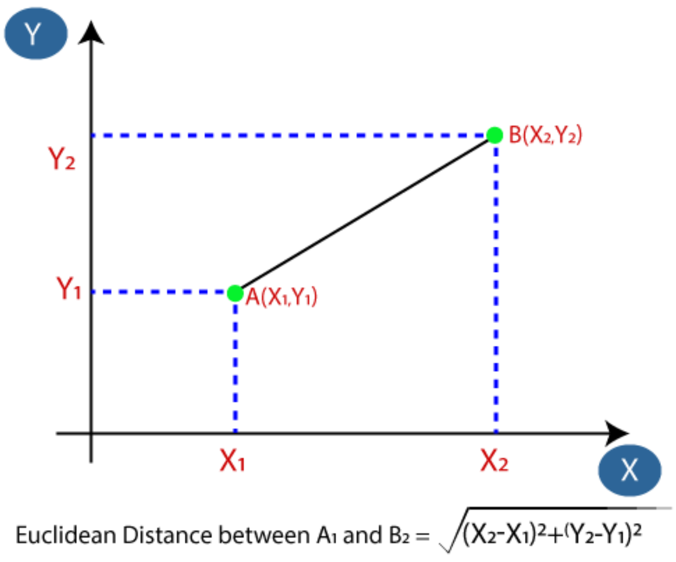
Steps involved in K-NN:

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.

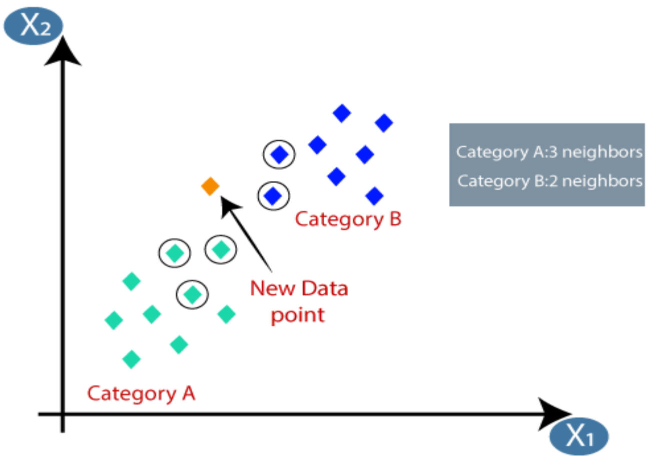
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- 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.

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:

1. There is no structured method to find the best value for "K". We need to find out with various values by trial and error and assuming that training data is unknown.
2. Choosing smaller values for K can be noisy and will have a higher influence on the result.
3. Larger values of K will have smoother decision boundaries which mean lower variance but increased bias. Also, computationally expensive.
4. In general, practice, choosing the value of **k** is **k = sqrt(N)** where **N** stands for the **number of samples in your training dataset**.
5. Try and keep the value of k odd in order to avoid confusion between two classes of data

Advantages of KNN Algorithm:

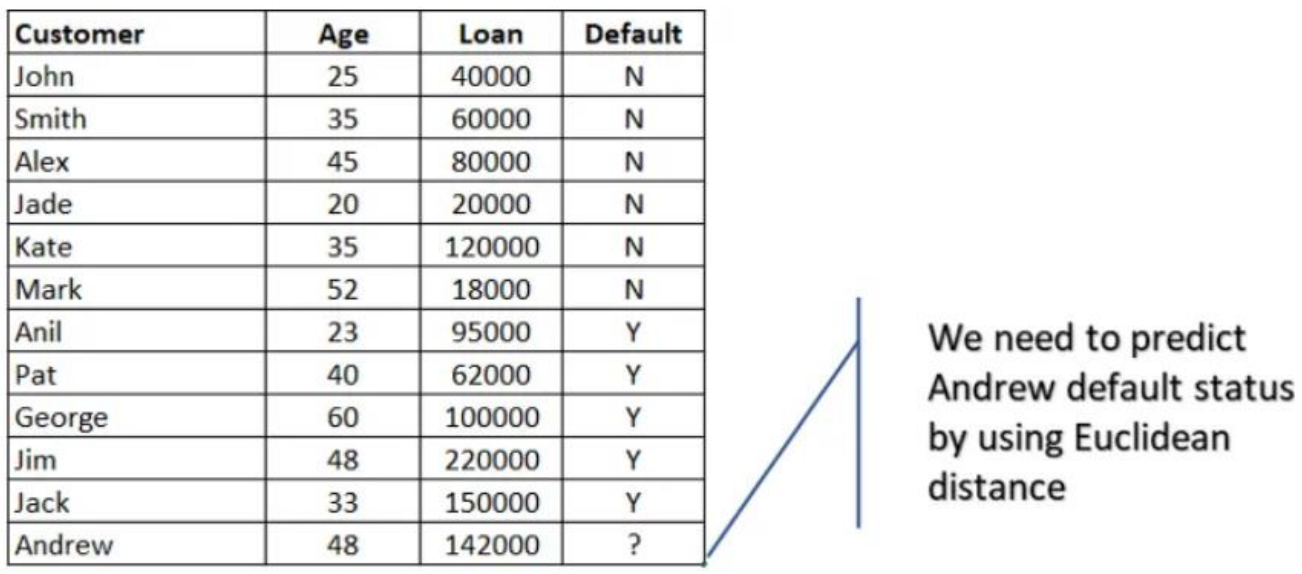
- It is simple to implement.
- It is robust to the noisy training data

- 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.

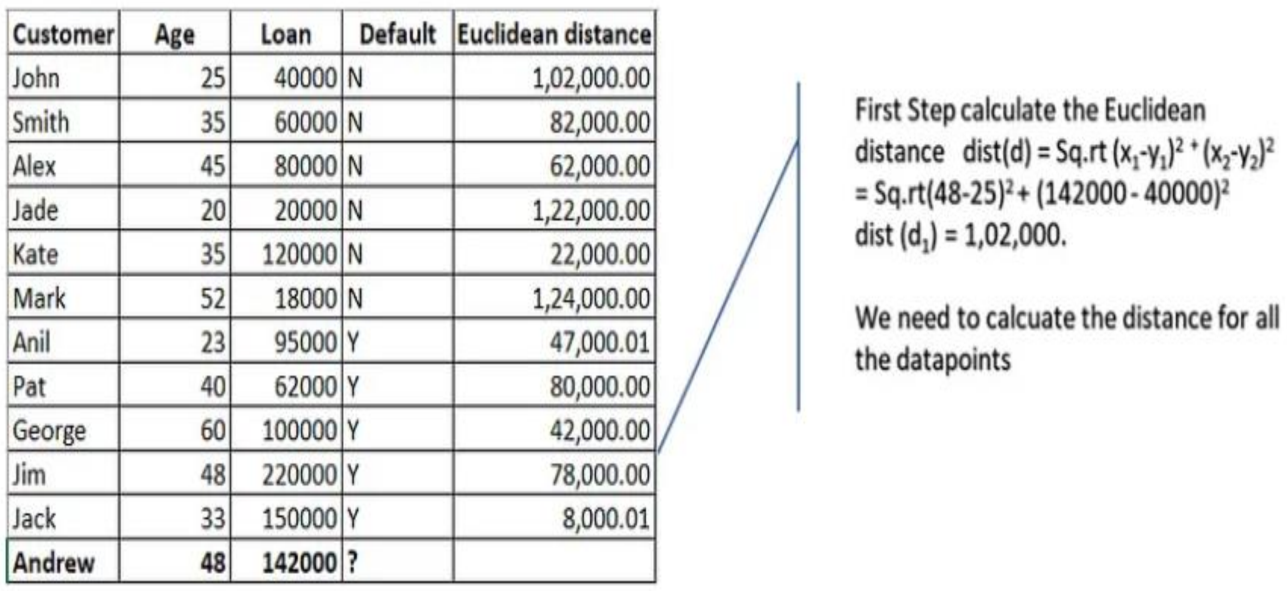
Example: Let's take a small example. Age vs loan.



Customer	Age	Loan	Default
John	25	40000	N
Smith	35	60000	N
Alex	45	80000	N
Jade	20	20000	N
Kate	35	120000	N
Mark	52	18000	N
Anil	23	95000	Y
Pat	40	62000	Y
George	60	100000	Y
Jim	48	220000	Y
Jack	33	150000	Y
Andrew	48	142000	?

We need to predict Andrew default status by using Euclidean distance

We need to predict Andrew default status (Yes or No).

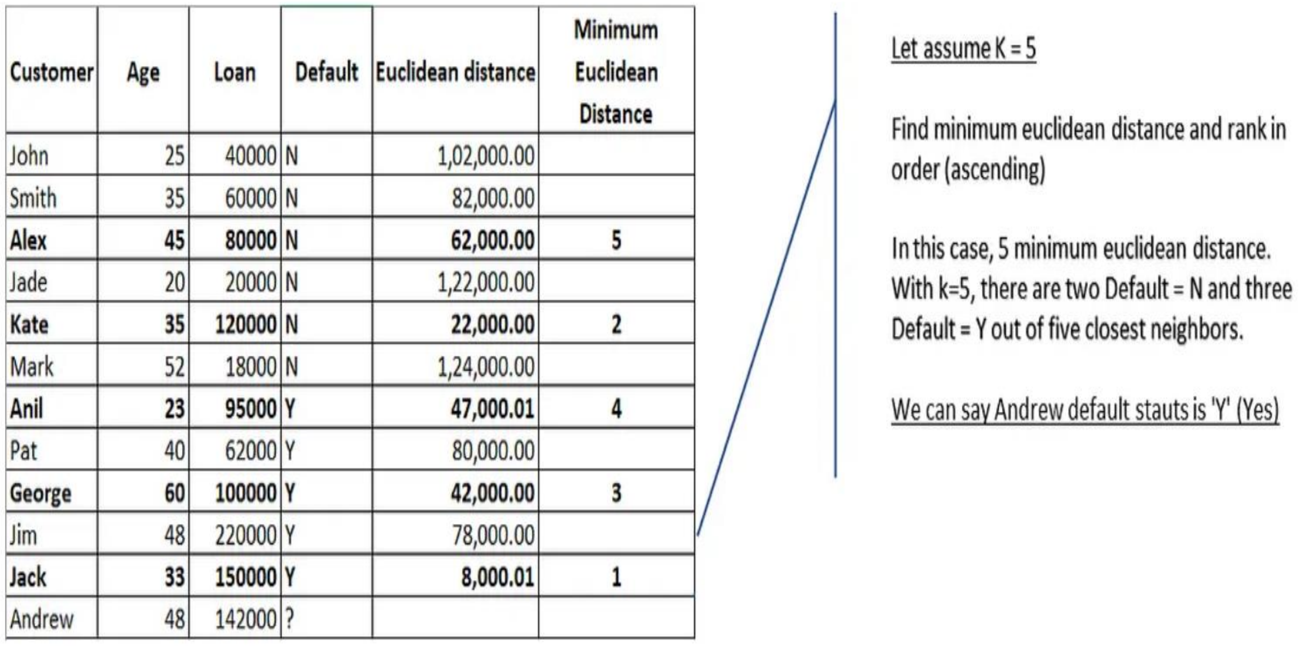


Customer	Age	Loan	Default	Euclidean distance
John	25	40000	N	1,02,000.00
Smith	35	60000	N	82,000.00
Alex	45	80000	N	62,000.00
Jade	20	20000	N	1,22,000.00
Kate	35	120000	N	22,000.00
Mark	52	18000	N	1,24,000.00
Anil	23	95000	Y	47,000.01
Pat	40	62000	Y	80,000.00
George	60	100000	Y	42,000.00
Jim	48	220000	Y	78,000.00
Jack	33	150000	Y	8,000.01
Andrew	**48**	**142000**	?	

First Step calculate the Euclidean distance dist(d) = Sq.rt $(x_1-y_1)^2 + (x_2-y_2)^2$
= Sq.rt$(48-25)^2 + (142000 - 40000)^2$
dist (d_1) = 1,02,000.

We need to calcuate the distance for all the datapoints

Calculate Euclidean distance for all the data points.



Customer	Age	Loan	Default	Euclidean distance	Minimum Euclidean Distance
John	25	40000	N	1,02,000.00	
Smith	35	60000	N	82,000.00	
Alex	**45**	**80000**	**N**	**62,000.00**	**5**
Jade	20	20000	N	1,22,000.00	
Kate	**35**	**120000**	**N**	**22,000.00**	**2**
Mark	52	18000	N	1,24,000.00	
Anil	**23**	**95000**	**Y**	**47,000.01**	**4**
Pat	40	62000	Y	80,000.00	
George	**60**	**100000**	**Y**	**42,000.00**	**3**
Jim	48	220000	Y	78,000.00	
Jack	**33**	**150000**	**Y**	**8,000.01**	**1**
Andrew	48	142000	?		

Let assume K = 5

Find minimum euclidean distance and rank in order (ascending)

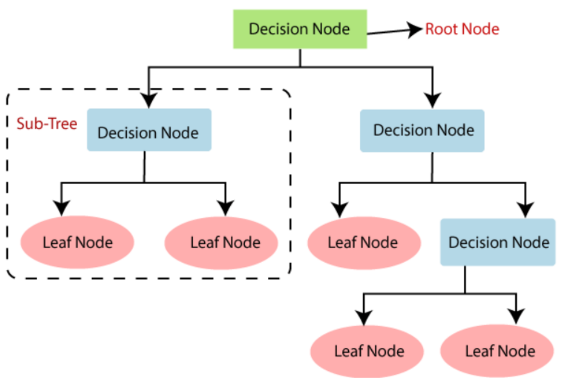
In this case, 5 minimum euclidean distance. With k=5, there are two Default = N and three Default = Y out of five closest neighbors.

We can say Andrew default stauts is 'Y' (Yes)

With K=5, there are two Default=N and three Default=Y out of five closest neighbors. We can say default status for Andrew is 'Y' based on the major similarity of 3 points out of 5.

3. DECISION TREES:

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- 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.**
- 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 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.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

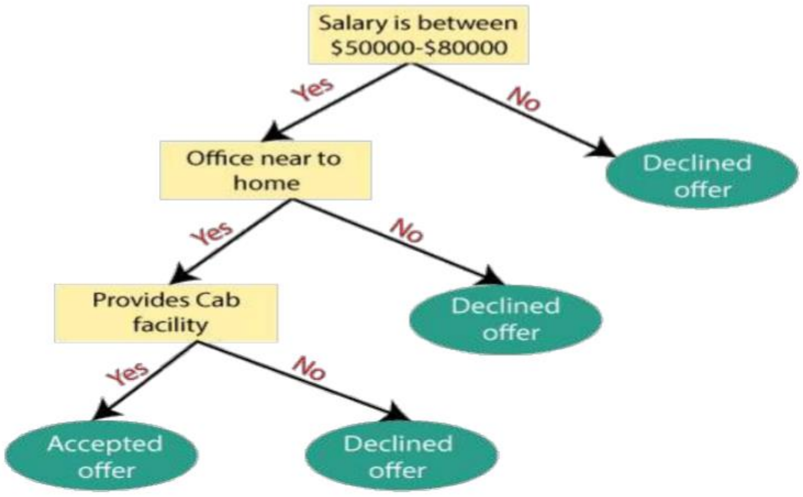
How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

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



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Let node N represent or hold the tuples of partition D.
- The attribute with the highest information gain is chosen as the splitting attribute for node N.
- This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions.
- Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.
- The expected information needed to classify a tuple in D is given by

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

where pi is the nonzero probability that an arbitrary tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|.

- A log function to the base 2 is used, because the information is encoded in bits.
- Info(D) is just the average amount of information needed to identify the class label of a tuple in D. Note that, at this point, the information we have is based solely on the proportions of tuples of each class.
- Info(D) is also known as the entropy of D.
- Now, suppose we were to partition the tuples in D on some attribute A having v dis tinct values, a1, a2,…. , av , as observed from the training data.
- If A is discrete-valued, these values correspond directly to the v outcomes of a test on A.
- Attribute A can be used to split D into v partitions or subsets, D1, D2, , Dv ,where Dj contains those tuples in D that have outcome aj of A.
- How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

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

- The term |Dj|/|D| acts as the weight of the jth partition. InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.
- Information gain is defined as the difference between the original information requirement and the new requirement

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

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

Example: Induction of a decision tree using information gain.

A (root) node N is created for the tuples in D. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Info(D) equation to compute the expected information needed to classify a tuple in D:

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

Next, compute the expected information requirement for each attribute. Let's start with the attribute age. We need to look at the distribution of yes and no tuples for each category of age. For the age category "youth," there are two yes tuples and three no tuples. For the category "middle_aged," there are four yes tuples and zero no tuples. For the category "senior," there are three yes tuples and two no tuples. Using Eq. Infoage(D), the expected information needed to classify a tuple in D if the tuples are partitioned according to age is

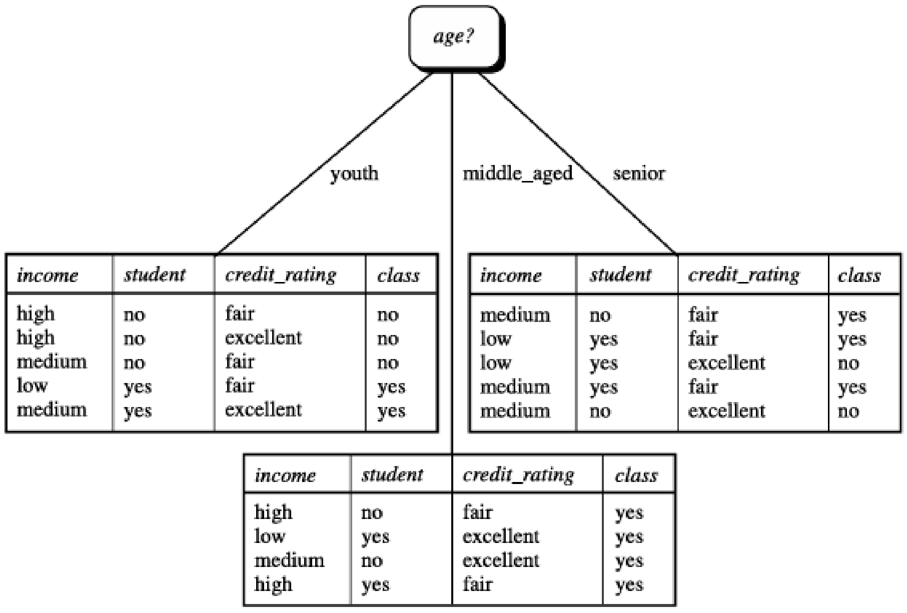
$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$+\frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right)$$
$$+\frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$
$$= 0.694 \text{ bits.}$$

Hence, the gain in information from such a partitioning would be

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

Similarly, we can compute Gain(income)= 0.029bits, Gain(student)= 0.151bits, and Gain(credit rating)= 0.048bits. Because age has the highest information gain among the attributes, it is selected as the splitting attribute.



income	student	credit_rating	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	student	credit_rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

2. Gini Index

Gini Index

The Gini index is used in CART. the Gini index measures the impurity of *D*, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

where *pi* is the probability that a tuple in *D* belongs to class *Ci* and is estimated by |*Ci,D*|/|*D*|. The sum is computed over *m* classes.

The Gini index considers a binary split for each attribute. Let's first consider the case where A is a discrete-valued attribute having v distinct values, {a1, a2, : : : , av}, occurring in D. When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D1 and D2, the Gini index of D given that partitioning is

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2).$$

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

Example: Induction of a decision tree using the Gini index. Let D be the training data shown earlier in Table 8.1, where there are nine tuples belonging to the class buys computer D yes and the remaining five tuples belong to the class buys computer D no. A (root) node N is created for the tuples in D. We first use Eq. for the Gini index to compute the impurity of D:

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

To find the splitting criterion for the tuples in D, we need to compute the Gini index for each attribute. Let's start with the attribute income and consider each of the possible splitting subsets. Consider the subset {low, medium}. This would result in 10 tuples in partition D1 satisfying the condition “income € {low, medium}.” The remaining four tuples of D would be assigned to partition D2. The Gini index value computed based on this partitioning is

$$Gini_{income \in \{low, medium\}}(D)$$
$$= \frac{10}{14}Gini(D_1) + \frac{4}{14}Gini(D_2)$$
$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$
$$= 0.443$$
$$= Gini_{income \in \{high\}}(D).$$

Similarly, the Gini index values for splits on the remaining subsets are 0.458 (for the subsets {low, high} and {medium}) and 0.450 (for the subsets {medium, high} and {low}).
Therefore, the best binary split for attribute income is on {low, medium} (or {high}) because it minimizes the Gini index. Evaluating age, we obtain {youth, senior} (or {middle aged}) as the best split for age with a Gini index of 0.375; the attributes student and credit rating are both binary, with Gini index values of 0.367 and 0.429, respectively.
The attribute age and splitting subset {youth, senior} therefore give the minimum Gini index overall, with a reduction in impurity of 0.459-0.357 = 0.102.
The binary split “age € {youth, senior}” results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion.

Advantages of the Decision Tree

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

Disadvantages of the Decision Tree

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

4. NAIVE BAYES:

- 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**.

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.

Working of Naïve Bayes' Classifier:

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. 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.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	**Play**
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

Frequency table for the Weather Conditions:

Weather	**Yes**	**No**
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	4

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(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

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

Disadvantages of Naïve Bayes Classifier:

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

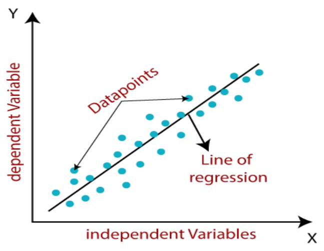
Applications of Naïve Bayes Classifier:

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

LINEAR MODELS:

1. LINEAR REGRESSION:

- Linear regression is one of the easiest and most popular Machine Learning algorithms.
- It is a statistical method that is used for predictive analysis.
- Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.
- Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.
- Linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
- The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



- Mathematically, we can represent a linear regression as:

$$y= a_0+a_1x$$

Here,
y= Dependent Variable (Target Variable)
x= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).

Types of Linear Regression:

Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**

  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

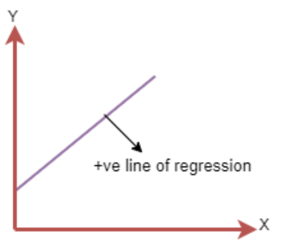
- **Multiple Linear regression:**

  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

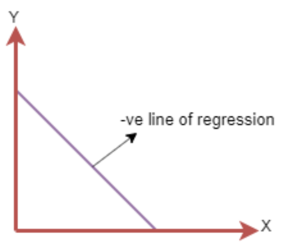
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: **Y= a_0+a_1x**

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: **Y= -a_0+a_1x**

Simple Linear Regression:

- Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable.
- The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.
- The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***.
- However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

The Simple Linear Regression model can be represented using the below equation:

$$y= a_0+a_1x$$

Where,

- y is the dependent variable
- x is the independent variable
- a0= It is the intercept of the Regression line (can be obtained putting x=0)
- a1= It is the slope of the regression line, which tells whether the line is increasing or decreasing.

Assumptions of Linear Regression:

- **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.

Multiple Linear Regression:

- There may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.
- Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:
- *Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.*

Example: Prediction of CO_2 emission based on engine size and number of cylinders in a car.

Key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

- In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x_1, x_2, x_3, ...,x_n. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

$$Y= b_0+b_1x_1+b_2x_2+b_3x_3+----------$$

Where,

- Y= Output/Response variable
- b_0, b_1, b_2, b_3 , b_n....= Coefficients of the model.
- x_1, x_2, x_3, x_4,...= Various Independent/feature variable

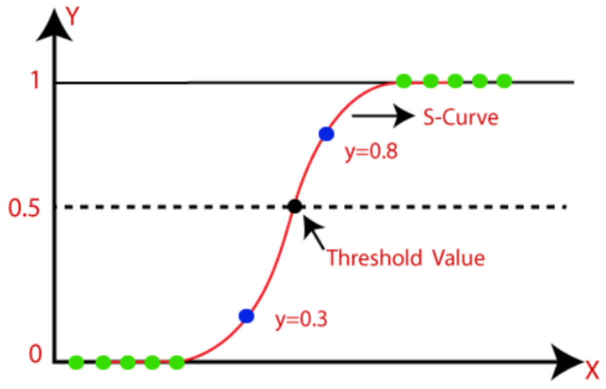
Assumptions for Multiple Linear Regression:

- A **linear relationship** should exist between the Target and predictor variables.
- The regression residuals must be **normally distributed**.
- MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

2. LOGISTIC REGRESSION:

- 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**.
- 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.
- The below image is showing the logistic function:



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.
- 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:

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

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:

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

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

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

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

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

The above equation is the final equation for Logistic Regression.

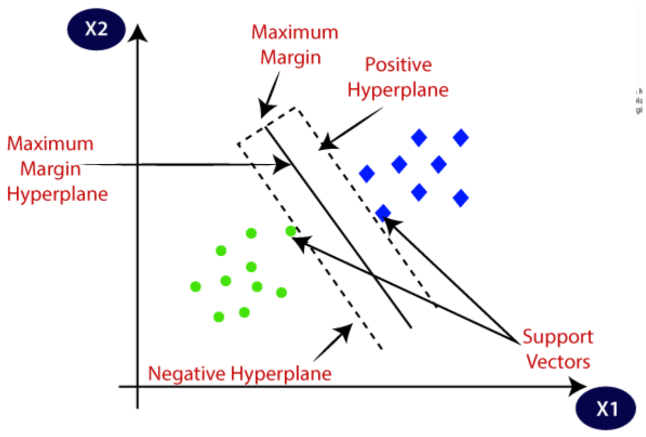
Types of Logistic Regression:

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

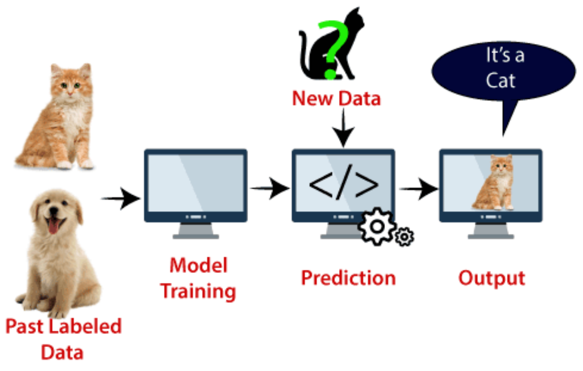
- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs",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

3. SUPPORT VECTOR MACHINES:

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM:

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

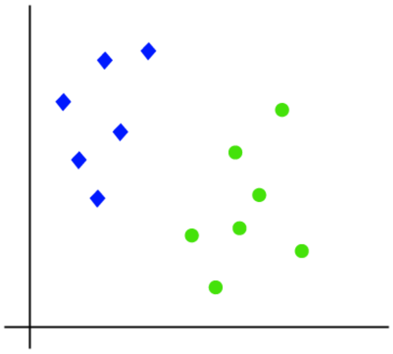
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

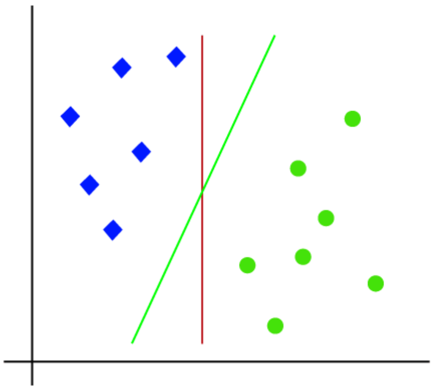
Working of SVM:

Linear SVM:

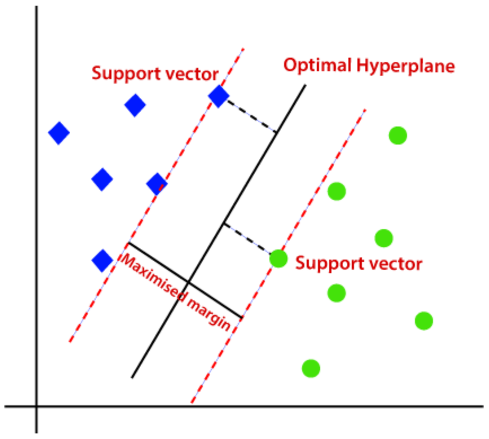
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

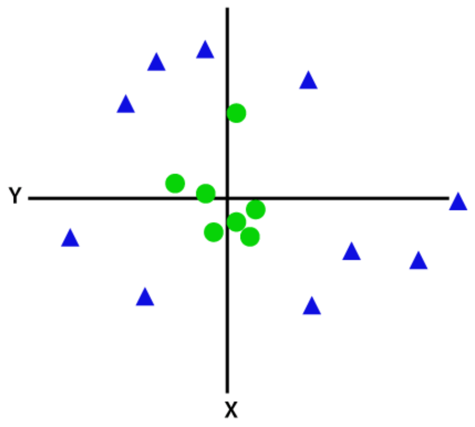


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



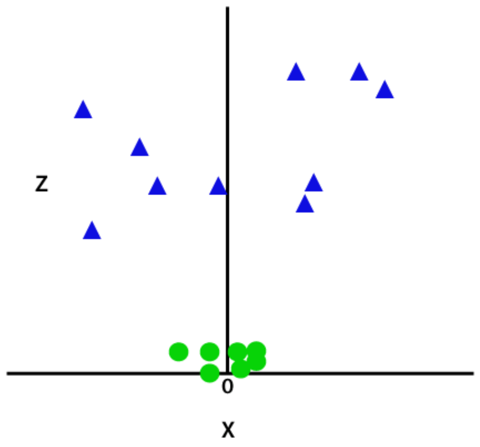
Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

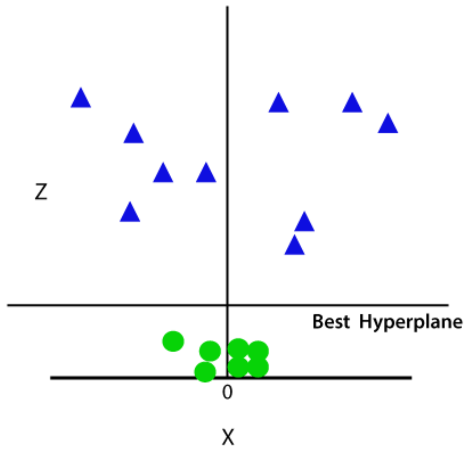


So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as: $z=x^2 + y^2$

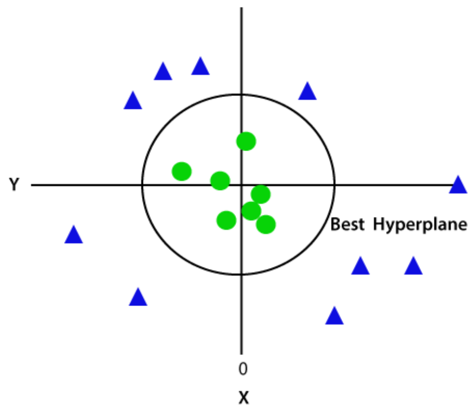
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

BINARY CLASSIFICATION

1. MULTICLASS/STRUCTURED OUTPUTS:

- Classification is a predictive modeling problem that involves assigning a class label to an example.
- Binary classification are those tasks where examples are assigned exactly one of two classes. Multi-class classification is those tasks where examples are assigned exactly one of more than two classes.

- **Binary Classification**: Classification tasks with two classes.
- **Multi-class Classification**: Classification tasks with more than two classes.

Some algorithms are designed for binary classification problems. Examples include:

- Logistic Regression
- Perceptron
- Support Vector Machines

As such, they cannot be used for multi-class classification tasks, at least not directly.

Instead, heuristic methods can be used to split a multi-class classification problem into multiple binary classification datasets and train a binary classification model each.

Two examples of these heuristic methods include:

- One-vs-Rest (OvR)
- One-vs-One (OvO)

One-Vs-Rest for Multi-Class Classification

One-vs-rest (OvR for short, also referred to as One-vs-All or OvA) is a heuristic method for using binary classification algorithms for multi-class classification.

It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident.

For example, given a multi-class classification problem with examples for each class '*red*,' '*blue*,' and '*green*'. This could be divided into three binary classification datasets as follows:

- **Binary Classification Problem 1**: red vs [blue, green]
- **Binary Classification Problem 2**: blue vs [red, green]
- **Binary Classification Problem 3**: green vs [red, blue]

If there are N-classes, then there are N-classifiers .

One-Vs-One for Multi-Class Classification:

One-vs-One (OvO for short) is another heuristic method for using binary classification algorithms for multi-class classification.

Like one-vs-rest, one-vs-one splits a multi-class classification dataset into binary classification problems. Unlike one-vs-rest that splits it into one binary dataset for each class, the one-vs-one approach splits the dataset into one dataset for each class versus every other class.

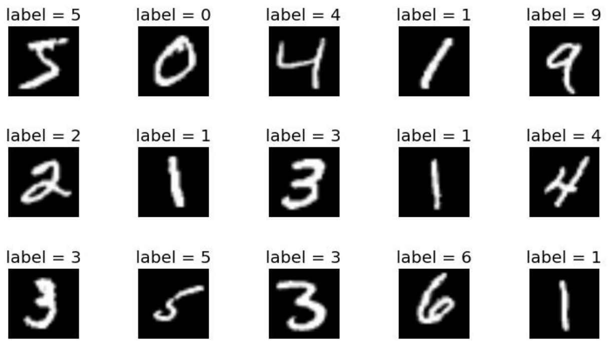
For example, consider a multi-class classification problem with four classes: '*red*,' '*blue*,' and '*green*,' '*yellow*.' This could be divided into six binary classification datasets as follows:

- **Binary Classification Problem 1**: red vs. blue
- **Binary Classification Problem 2**: red vs. green
- **Binary Classification Problem 3**: red vs. yellow
- **Binary Classification Problem 4**: blue vs. green
- **Binary Classification Problem 5**: blue vs. yellow
- **Binary Classification Problem 6**: green vs. yellow

If there are N-classes, then there are N(N-1)/2-classifiers .

2. MNIST:

- The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.
- It was created by "re-mixing" the samples from NIST's original datasets.
- he MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it.
- An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training mnist dataset images, and 40,000 testing mnist dataset images of MNIST dataset of handwritten digits and characters.



MNIST Uses:

- MNIST provides a baseline for testing image processing systems.
- Because MNIST is a labeled dataset that pairs images of hand-written numerals with the name of the respective numeral, it can be used in supervised learning to train classifiers.

3. RANKING:

- Ranking is a machine learning technique to rank items.
- Ranking is useful for many applications in information retrieval such as e-commerce, social networks, recommendation systems, and so on.
- For example, a user searches for an article or an item to buy online. To build a recommendation system, it becomes important that similar articles or items of relevance appear to the user such that the user clicks or purchases the item.
- A simple regression model can predict the probability of a user to click an article or buy an item.
- However, it is more practical to use ranking technique and be able to order or rank the articles or items to maximize the chances of getting a click or purchase.
- The prioritization of the articles or the items influence the decision of the users.
- The ranking technique directly ranks items by training a model to predict the ranking of one item over another item.
- In the training model, it is possible to have items, ranking one over the other by having a "score" for each item.
- Higher ranked items have higher scores and lower ranked items have lower scores.
- Using these scores, a model is built to predict which item ranks higher than the other.