UNIT-I

Introduction: Introduction to Machine learning, Supervised learning, Unsupervised learning, Reinforcement learning. Deep learning. Feature Selection: Filter, Wrapper Embedded methods. Feature Normalization:- min-max normalization, z-score normalization, and constant factor normalization Introduction to Dimensionality Reduction : Principal Component Analysis(PCA), Linear Discriminant Analysis(LDA)

Introduction

1.1 Definition of Machine Learning

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. He defined machine learning as "the field of study that gives computers the ability to learn without being explicitly programmed." However, there is no universally accepted definition for machine learning. Different authors define the term differently. We give below two more definitions.

Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience.
The field of study known as machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

In the above definitions we have used the term **"model"** and we will be using this term at several contexts later. It appears that there is no universally accepted one sentence definition of this term. Loosely, it may be understood as some mathematical expression or equation, or some mathematical structures such as graphs and trees, or a division of sets into disjoint subsets, or a set of logical "if . . . then . . . else . . ." rules, or some such thing. It may be noted that this is not an exhaustive list.

Definition of learning

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Examples:

i) Handwriting recognition learning problem

• Task T: Recognising and classifying handwritten words within images

• Performance P: Percent of words correctly classified

• Training experience E: A dataset of handwritten words with given classifications

ii) A robot driving learning problem

• Task T: Driving on highways using vision sensors

• Performance measure P: Average distance traveled before an error

• training experience: A sequence of images and steering commands recorded while observing a human driver

iii) A chess learning problem

• Task T: Playing chess

• Performance measure P: Percent of games won against opponents

• Training experience E: Playing practice games against itself

A computer program which learns from experience is called a machine learning program or simply a learning program. Such a program is sometimes also referred to as a learner.

1.1.2 How machines learn

Basic components of learning process:

The learning process, whether by a human or a machine, can be divided into four components, namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the various components and the steps involved in the learning process.

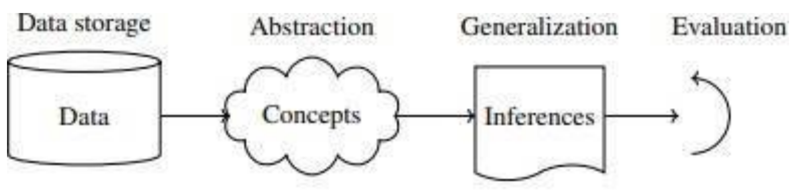


Figure 1.1: Components of learning process

Data storage: Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

• In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.

• Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

Abstraction

The second component of the learning process is known as abstraction. Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models.

The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

Generalization

The third component of the learning process is known as generalisation.

The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

Evaluation

Evaluation is the last component of the learning process. It is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then utilised to effect improvements in the whole learning process.

1.1.3 Applications of machine learning

Application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having high predictive accuracy.

The following is a list of some of the typical applications of machine learning.

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.
4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon and Go.

1.2 Different types of learning

In general, machine learning algorithms can be classified into three types.

Supervised learning:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

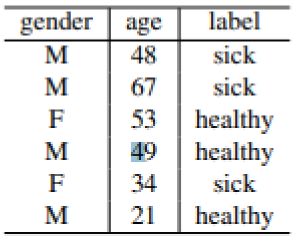
In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value. A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples. In the optimal case, the function will correctly determine the class labels for unseen instances. Both classification and regression problems are supervised learning problems.

A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.

A "supervised learning" is so called because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers (that is, the correct outputs), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Example :

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labelled as "healthy" or "sick".



gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	healthy
F	34	sick
M	21	healthy

Unsupervised learning

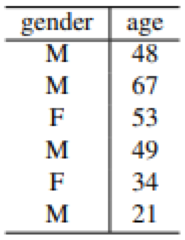
Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

In unsupervised learning algorithms, a classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Example :

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.



gender	age
M	48
M	67
F	53
M	49
F	34
M	21

Based on this data, can we infer anything regarding the patients entering the clinic?

Reinforcement learning

Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards.

A learner (the program) is not told what actions to take as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards.

For example, consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs. Reinforcement learning is different from supervised learning. Supervised learning is learning from examples provided by a knowledgeable expert.

1.3 Feature Selection

“Feature selection is a way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, or noisy features.”

While developing the machine learning model, only a few variables in the dataset are useful for building the model, and the rest features are either redundant or irrelevant. If we input the dataset with all these redundant and irrelevant features, it may negatively impact and reduce the overall performance and accuracy of the model. Hence it is very important to identify and select the most appropriate features from the data and remove the irrelevant or less important features, which is done with the help of feature selection in machine learning.

Feature selection is one of the important concepts of machine learning, which highly impacts the performance of the model. As machine learning works on the concept of "Garbage In Garbage Out", so we always need to input the most appropriate and relevant dataset to the model in order to get a better result.

In this topic, we will discuss different feature selection techniques for machine learning. But before that, let's first understand some basics of feature selection.

What is Feature Selection?

A feature is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the model is known as feature selection. Each machine learning process depends on feature engineering, which mainly contains two processes; which are Feature Selection and Feature Extraction.

Although feature selection and extraction processes may have the same objective, both are completely different from each other. The main difference between them is that **feature selection** is about selecting the subset of the original feature set, whereas **feature extraction** creates new features.

Feature selection is a way of reducing the input variable for the model by using only relevant data in order to reduce over fitting in the model.

So, we can define feature Selection as, "***It is a process of automatically or manually selecting the subset of most appropriate and relevant features to be used in model***

building." Feature selection is performed by either including the important features or excluding the irrelevant features in the dataset without changing them.

Need for Feature Selection:

Before implementing any technique, it is really important to understand, need for the technique and so for the Feature Selection. As we know, in machine learning, it is necessary to provide a pre-processed and good input dataset in order to get better outcomes. We collect a huge amount of data to train our model and help it to learn better. Generally, the dataset consists of noisy data, irrelevant data, and some part of useful data. Moreover, the huge amount of data also slows down the training process of the model, and with noise and irrelevant data, the model may not predict and perform well. So, it is very necessary to remove such noises and less-important data from the dataset and to do this, and Feature selection techniques are used.

Selecting the best features helps the model to perform well. For example, Suppose we want to create a model that automatically decides which car should be crushed for a spare part, and to do this, we have a dataset. This dataset contains a Model of the car, Year, Owner's name, Miles. So, in this dataset, the name of the owner does not contribute to the model performance as it does not decide if the car should be crushed or not, so we can remove this column and select the rest of the features(column) for the model building.

Below are some benefits of using feature selection in machine learning:

It helps in avoiding the curse of dimensionality.

It helps in the simplification of the model so that it can be easily interpreted by the researchers.

It reduces the training time.

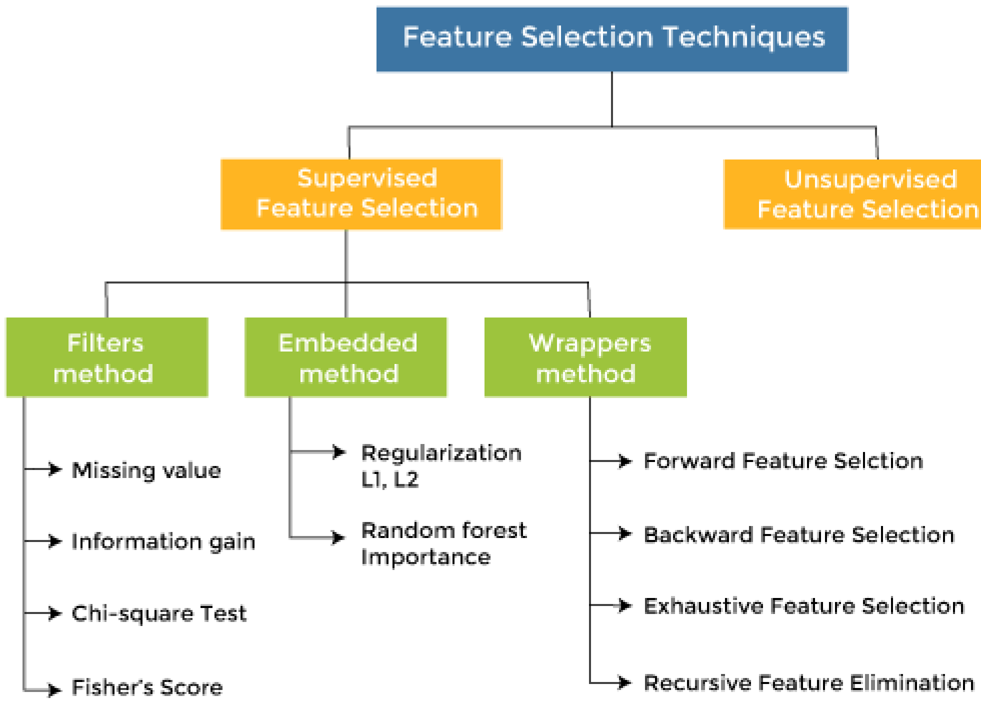
It reduces over fitting hence enhance the generalization.

1.2 Feature Selection Techniques:

There are mainly two types of Feature Selection techniques, which are:

Supervised Feature selection techniques consider the target variable and can be used for the labelled dataset.

Unsupervised Feature selection techniques ignore the target variable and can be used for the unlabelled dataset.

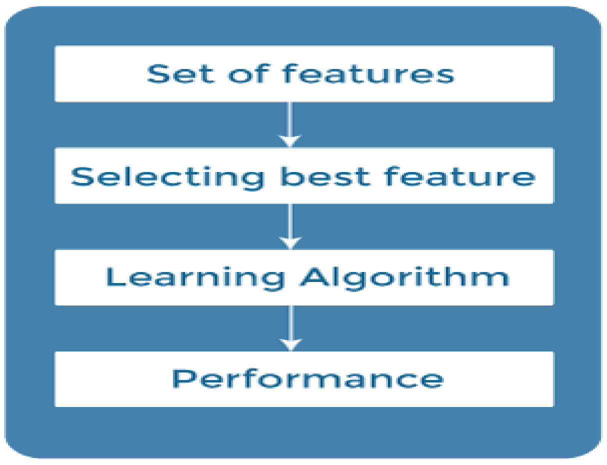


1.2.1 Filter Methods:

In Filter Method, features are selected on the basis of statistics measures. This method does not depend on the learning algorithm and chooses the features as a pre-processing step.

The filter method filters out the irrelevant feature and redundant columns from the model by using different metrics through ranking.

The advantage of using filter methods is that it needs low computational time and does not over fit the data.



Some common techniques of Filter methods are as follows::

information Gain

Chi-square Test

Fisher's Score

Missing Value Ratio

Information Gain: Information gain determines the reduction in entropy while transforming the dataset. It can be used as a feature selection technique by calculating the information gain of each variable with respect to the target variable.

Chi-square Test: Chi-square test is a technique to determine the relationship between the categorical variables. The chi-square value is calculated between each feature and the target variable, and the desired number of features with the best chi-square value is selected.

Fisher's Score:

Fisher's score is one of the popular supervised techniques of features selection. It returns the rank of the variable on the fisher's criteria in descending order. Then we can select the variables with a large fisher's score.

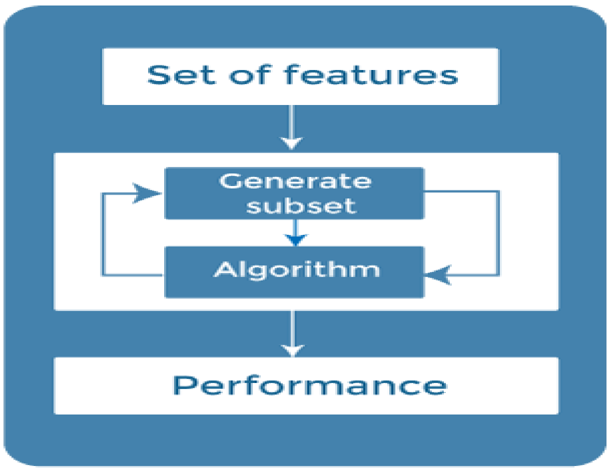
Missing Value Ratio:

The value of the missing value ratio can be used for evaluating the feature set against the threshold value. The formula for obtaining the missing value ratio is the number of missing values in each column divided by the total number of observations. The variable is having more than the threshold value can be dropped.

$$\text{Missing Value Ratio} = \frac{\textit{Number of Missing values} * 100}{\textit{Total number of observations}}$$

1.2.2 Wrapper Methods:

In wrapper methodology, selection of features is done by considering it as a search problem, in which different combinations are made, evaluated, and compared with other combinations. It trains the algorithm by using the subset of features iteratively.



On the basis of the output of the model, features are added or subtracted, and with this feature set, the model has trained again.

Some techniques of wrapper methods are:

Forward selection - Forward selection is an iterative process, which begins with an empty set of features. After each iteration, it keeps adding on a feature and evaluates the performance to check whether it is improving the performance or not. The process continues until the addition of a new variable/feature does not improve the performance of the model.

Backward elimination - Backward elimination is also an iterative approach, but it is the opposite of forward selection. This technique begins the process by considering all the features and removes the least significant feature. This elimination process continues until removing the features does not improve the performance of the model.

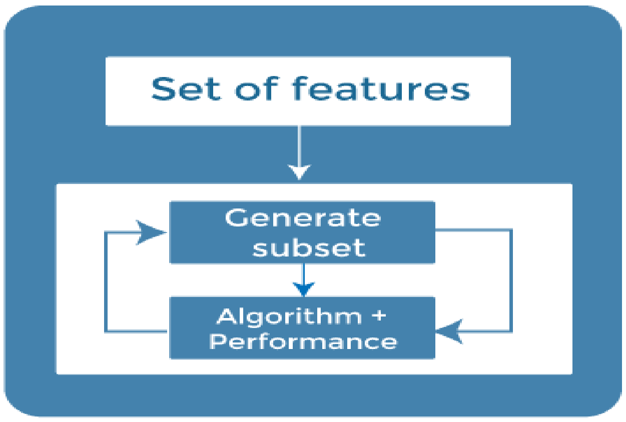
Exhaustive Feature Selection- Exhaustive feature selection is one of the best feature selection methods, which evaluates each feature set as brute-force. It means this method tries & make each possible combination of features and return the best performing feature set.

Recursive feature elimination

Recursive feature elimination is a recursive greedy optimization approach, where features are selected by recursively taking a smaller and smaller subset of features. Now, an estimator is trained with each set of features, and the importance of each feature is determined using *coef_attribute* or through a *feature_importances_attribute.*

1.2.3 Embedded Methods

Embedded methods combined the advantages of both filter and wrapper methods by considering the interaction of features along with low computational cost. These are fast processing methods similar to the filter method but more accurate than the filter method.



These methods are also iterative, which evaluates each iteration, and optimally finds the most important features that contribute the most to training in a particular iteration. Some techniques of embedded methods are:

Regularization- Regularization adds a penalty term to different parameters of the machine learning model for avoiding overfitting in the model. This penalty term is added to the coefficients; hence it shrinks some coefficients to zero. Those features with zero coefficients can be removed from the dataset. The types of regularization techniques are L1 Regularization (Lasso Regularization) or Elastic Nets (L1 and L2 regularization).

Random Forest Importance - Different tree-based methods of feature selection help us with feature importance to provide a way of selecting features. Here, feature importance specifies which feature has more importance in model building or has a great impact on the target variable. Random Forest is such a tree-based method, which is a type of bagging algorithm that aggregates a different number of decision trees. It automatically ranks the nodes by their performance or decrease in the impurity (Gini impurity) over all the trees. Nodes are arranged as per the impurity values, and thus it allows to pruning of trees below a specific node. The remaining nodes create a subset of the most important features.

1.3 Feature normalization:

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

1.3.1 Min-max normalization

Min-max normalization (usually called **feature scaling**) performs a linear transformation on the original data. This technique gets all the scaled data in the range (0, 1). The formula to achieve this is the following:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

For the three example values, min = 28 and max = 46. Therefore, the min-max normalized values are:

28: (28 - 28) / (46 - 28) = 0 / 18 = 0.00

46: (46 - 28) / (46 - 28) = 18 / 18 = 1.00

34: (34 - 28) / (46 - 28) = 6 / 18 = 0.33

The min-max technique results in values between 0.0 and 1.0 where the smallest value is normalized to 0.0 and the largest value is normalized to 1.0.

1.3.2 Z-score normalization refers to the process of normalizing every value in a dataset such that the mean of all of the values is 0 and the standard deviation is 1.

We use the following formula to perform a z-score normalization on every value in a dataset:

New value = (x – μ) / σ

where:

x: Original value

μ: Mean of data

σ: Standard deviation of data

For the three example values, mean(**μ**) = (28 + 46 + 34) / 3 = 108 / 3 = 36.0. The standard deviation of a set of values is the square root of the sum of the squared difference of each value and the mean, divided by the number of values, and so is:

σ = sqrt([(28 - 36.0)^2 + (46 - 36.0)^2 + (34 - 36.0)^2] / 3)

= sqrt([(-8.0)^2 + (10.0)^2 + (-2.0)^2] / 3)

= sqrt([64.0 + 100.0 + 4.0] / 3)

= sqrt(168.0 / 3)

= sqrt(56.0)

= 7.48

Therefore, the z-score normalized values are:

```
28: (28 - 36.0) / 7.48 = -1.07

46: (46 - 36.0) / 7.48 = +1.34

34: (34 - 36.0) / 7.48 = -0.27
```

A z-score normalized value that is positive corresponds to an x value that is greater than the mean value, and a z-score that is negative corresponds to an x value that is less than the mean.

1.3.3 Constant Factor Normalization:

The simplest normalization technique is constant factor normalization. Expressed as a math equation constant factor normalization is x' = x / k, where x is a raw value, x' is the normalized value, and k is a numeric constant. If k = 100, the constant factor normalized values are:

```
28: 28 / 100 = 0.28

46: 46 / 100 = 0.46

34: 34 / 100 = 0.34
```

1.4 Dimensionality Reduction

Dimensionality reduction or dimension reduction is the process of reducing the number of variables under consideration by obtaining a smaller set of principal variables.
Dimensionality reduction may be implemented in two ways.

• Feature selection

In feature selection, we are interested in finding k of the total of n features that give us the most information and we discard the other (n−k) dimensions. We are going to discuss subset selection as a feature selection method.

• Feature extraction

In feature extraction, we are interested in finding a new set of k features that are the combination of the original n features. These methods may be supervised or unsupervised depending on whether or not they use the output information. The best known and most widely used feature extraction methods are Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA), which are both linear projection methods, unsupervised and supervised respectively.

Why dimensionality reduction is useful?

There are several reasons why we are interested in reducing dimensionality.

- In most learning algorithms, the complexity depends on the number of input dimensions, d, as well as on the size of the data sample, N, and for reduced memory and computation, we are interested in reducing the dimensionality of the problem. Decreasing d also decreases the complexity of the inference algorithm during testing.
- When an input is decided to be unnecessary, we save the cost of extracting it.
- Simpler models are more robust on small datasets. Simpler models have less variance, that is, they vary less depending on the particulars of a sample, including noise, outliers, and so forth.
- When data can be explained with fewer features, we get a better idea about the process that underlies the data, which allows knowledge extraction.
- When data can be represented in a few dimensions without loss of information, it can be plotted and analyzed visually for structure and outliers.

1.4.1 Principal component analysis:

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

Computation of the principal component vectors

(PCA algorithm)

The following is an outline of the procedure for performing a principal component analysis on a given data. The procedure is heavily dependent on mathematical concepts. A knowledge of these concepts is essential to carry out this procedure.

Step 1. Data

We consider a dataset having n features or variables denoted by X1;X2; : : : ;Xn. Let there be N examples. Let the values of the i-th feature Xi be Xi1;Xi2; : : : ;XiN (see Table 4.1).

Features	Example 1	Example 2	$\cdots$	Example N
X_1	X_{11}	X_{12}	$\cdots$	X_{1N}
X_2	X_{21}	X_{22}	$\cdots$	X_{2N}
$\vdots$				
X_i	X_{i1}	X_{i2}	$\cdots$	X_{iN}
$\vdots$				
X_n	X_{n1}	X_{n2}	$\cdots$	X_{nN}

Table 4.1: Data for PCA algorithm

Step 2. Compute the means of the variables

We compute the mean _Xi of the variable Xi:

$$\bar{X}_i = \frac{1}{N}(X_{i1} + X_{i2} + \cdots + X_{iN}).$$

Step 3. Calculate the covariance matrix

Consider the variables Xi and Xj (i and j need not be different). The covariance of the Ordered pair (Xi;Xj) is defined as

$$\text{Cov}(X_i, X_j) = \frac{1}{N-1}\sum_{k=1}^{N}(X_{ik} - \bar{X}_i)(X_{jk} - \bar{X}_j). \qquad (4.1)$$

We calculate the following $n \times n$ matrix S called the covariance matrix of the data. The element in the i-th row j-th column is the covariance $\text{Cov}(X_i, X_j)$:

$$S = \begin{bmatrix} \text{Cov}(X_1,X_1) & \text{Cov}(X_1,X_2) & \cdots & \text{Cov}(X_1,X_n) \\ \text{Cov}(X_2,X_1) & \text{Cov}(X_2,X_2) & \cdots & \text{Cov}(X_2,X_n) \\ \vdots & & & \\ \text{Cov}(X_n,X_1) & \text{Cov}(X_n,X_2) & \cdots & \text{Cov}(X_n,X_n) \end{bmatrix}$$

Step 4. Calculate the eigen values and eigenvectors of the covariance matrix

Let S be the covariance matrix and let I be the identity matrix having the same dimension as the dimension of S.

i) Set up the equation:

$$\det(S - \lambda I) = 0. \tag{4.2}$$

This is a polynomial equation of degree n in λ. It has n real roots (some of the roots may be repeated) and these roots are the eigenvalues of S. We find the n roots $\lambda_1, \lambda_2, \ldots, \lambda_n$ of Eq. (4.2).

ii) If $\lambda = \lambda'$ is an eigenvalue, then the corresponding eigenvector is a vector

$$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

such that

$$(S - \lambda' I)U = 0.$$

(This is a system of n homogeneous linear equations in u_1, u_2, ..., u_n and it always has a nontrivial solution.) We next find a set of n orthogonal eigenvectors $U_1, U_2, \ldots, U_n$ such that U_i is an eigenvector corresponding to λ_i.[2]

iii) We now normalise the eigenvectors. Given any vector X we normalise it by dividing X by its length. The length (or, the norm) of the vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

is defined as

$$||X|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Given any eigenvector U, the corresponding normalised eigenvector is computed as

$$\frac{1}{||U||}U.$$

We compute the n normalised eigenvectors $e_1, e_2, \ldots, e_n$ by

We compute the n normalised eigenvectors $e_1, e_2, \dots, e_n$ by

$$e_i = \frac{1}{||U_i||} U_i, \quad i = 1, 2, \dots, n.$$

Step 5. Derive new data set

Order the eigenvalues from highest to lowest. The unit eigenvector corresponding to the largest eigenvalue is the first principal component. The unit eigenvector corresponding to the next highest eigenvalue is the second principal component, and so on.

i) Let the eigenvalues in descending order be $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and let the corresponding unit eigenvectors be $e_1, e_2, \dots, e_n$.

ii) Choose a positive integer p such that $1 \leq p \leq n$.

iii) Choose the eigenvectors corresponding to the eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_p$ and form the following $p \times n$ matrix (we write the eigenvectors as row vectors):

$$F = \begin{bmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_p^T \end{bmatrix},$$

where T in the superscript denotes the transpose.

iv) We form the following $n \times N$ matrix:

$$X = \begin{bmatrix} X_{11} - \bar{X}_1 & X_{12} - \bar{X}_1 & \cdots & X_{1N} - \bar{X}_1 \\ X_{21} - \bar{X}_2 & X_{22} - \bar{X}_2 & \cdots & X_{2N} - \bar{X}_2 \\ \vdots & & & \\ X_{n1} - \bar{X}_n & X_{n2} - \bar{X}_n & \cdots & X_{nN} - \bar{X}_n \end{bmatrix}$$

v) Next compute the matrix:

$$X_{\text{new}} = FX.$$

Note that this is a $p \times N$ matrix. This gives us a dataset of N samples having p features.

Step 6. New dataset

The matrix X_{new} is the new dataset. Each row of this matrix represents the values of a feature. Since there are only p rows, the new dataset has only features.

Step 7. Conclusion

This is how the principal component analysis helps us in dimensional reduction of the dataset. Note that it is not possible to get back the original n-dimensional dataset from the new dataset.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

- Improved Visualization: High dimensional data is difficult to visualize, and dimensionality reduction techniques can help in visualizing the data in 2D or 3D, which can help in better understanding and analysis.
- Overfitting Prevention: High dimensional data may lead to overfitting in machine learning models, which can lead to poor generalization performance. Dimensionality reduction can help in reducing the complexity of the data, and hence prevent overfitting.
- Feature Extraction: Dimensionality reduction can help in extracting important features from high dimensional data, which can be useful in feature selection for machine learning models.
- Data Preprocessing: Dimensionality reduction can be used as a preprocessing step before applying machine learning algorithms to reduce the dimensionality of the data and hence improve the performance of the model.
- Improved Performance: Dimensionality reduction can help in improving the performance of machine learning models by reducing the complexity of the data, and hence reducing the noise and irrelevant information in the data.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.
- Interpretability: The reduced dimensions may not be easily interpretable, and it may be difficult to understand the relationship between the original features and the reduced dimensions.
- Overfitting: In some cases, dimensionality reduction may lead to overfitting, especially when the number of components is chosen based on the training data.
- Sensitivity to outliers: Some dimensionality reduction techniques are sensitive to outliers, which can result in a biased representation of the data.
- Computational complexity: Some dimensionality reduction techniques, such as manifold learning, can be computationally intensive, especially when dealing with large datasets.

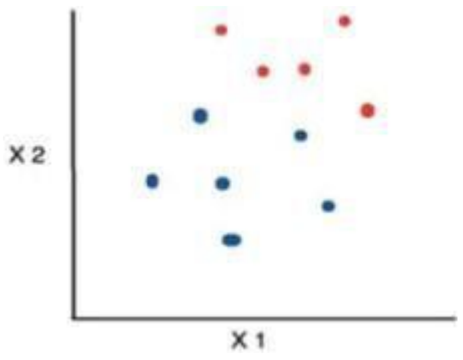
1.4.2 Linear Discriminant Analysis (LDA):

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

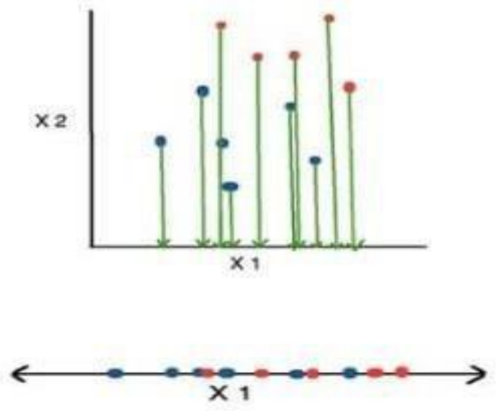
Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.

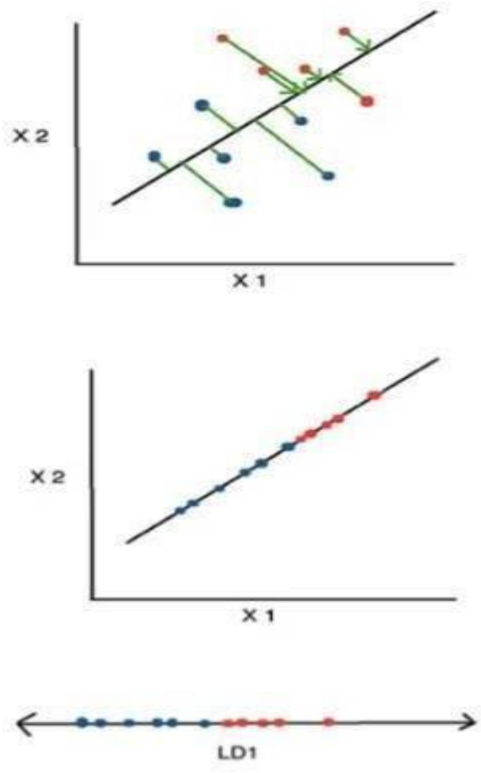
Consider a situation where you have plotted the relationship between two variables where each color represents a different class. One is shown with a red color and the other with blue.



If you are willing to reduce the number of dimensions to 1, you can just project everything to the x-axis as shown below:



This approach neglects any helpful information provided by the second feature. However, you can use LDA to plot it. The advantage of LDA is that it uses information from both the features to create a new axis which in turn minimizes the variance and maximizes the class distance of the two variables.



Drawbacks of Linear Discriminant Analysis (LDA)

Although, LDA is specifically used to solve supervised classification problems for two or more classes which are not possible using logistic regression in machine learning. But LDA also fails in some cases where the Mean of the distributions is shared. In this case, LDA fails to create a new axis that makes both the classes linearly separable. **Real-world Applications of LDA**

Some of the common real-world applications of Linear discriminant Analysis are given below:

- **FaceRecognition**
  Face recognition is the popular application of computer vision, where each face is represented as the combination of a number of pixel values. In this case, LDA is used to minimize the number of features to a manageable number before going through the classification process. It generates a new template in which each dimension consists of a linear combination of pixel values. If a linear combination is generated using Fisher's linear discriminant, then it is called Fisher's face.
- **Medical**
  In the medical field, LDA has a great application in classifying the patient disease on the basis of various parameters of patient health and the medical treatment which is going on. On such parameters, it classifies disease as mild, moderate, or severe. This classification helps the doctors in either increasing or decreasing the pace of the treatment.
- **Customer Identification**
  In customer identification, LDA is currently being applied. It means with the help of LDA; we can easily identify and select the features that can specify the group of customers who are likely to purchase a specific product in a shopping mall. This can be helpful when we want to identify a group of customers who mostly purchase a product in a shopping mall.
- **For Predictions**
  LDA can also be used for making predictions and so in decision making. For example, "will you buy this product" will give a predicted result of either one or two possible classes as a buying or not.
- **InLearning**
  Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

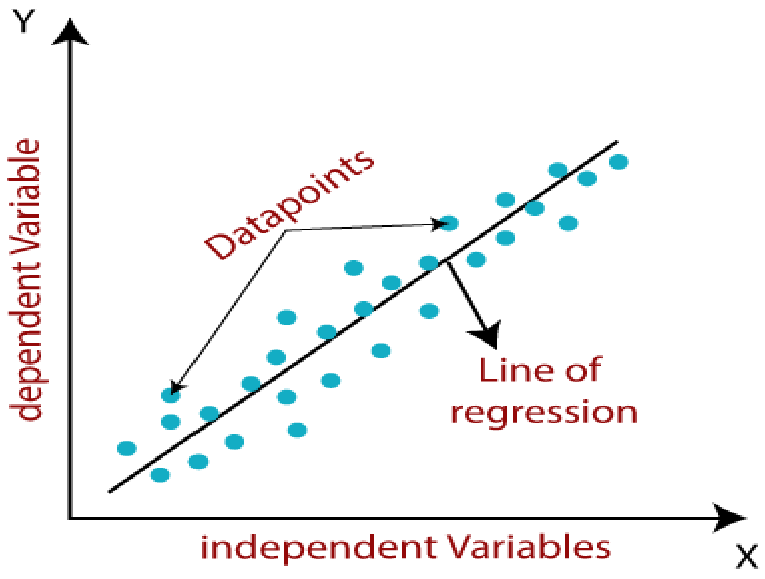
UNIT – II

Supervised Learning – I (Regression/Classification) Regression models: Simple Linear Regression, multiple linear Regression. Cost Function, Gradient Descent, Performance Metrics: Mean Absolute Error(MAE),Mean Squared Error(MSE) R-Squared error, Adjusted R Square. Classification models: Decision Trees-ID3,CART, Naive Bayes, K-Nearest-Neighbours (KNN), Logistic Regression, Multinomial Logistic Regression Support Vector Machines (SVM) - Nonlinearity and Kernel Methods

Linear regression:

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.

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:

$y= a_0+a_1x+ \varepsilon$

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).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Regression Models

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.

2.1 Linear regression

Linear regression in simple term is answering a question on "How can I use X to predict Y?" where X is some information that you have, and Y is some information that you want.

Let's say you wanted a sell a house and you wanted to know how much you can sell it for. You have information about the house that is your X and the selling price that you wanted to know will be your Y.

Linear regression creates an equation in which you input your given numbers (X) and it outputs the target variable that you want to find out (Y).

Linear Regression model representation

Linear regression is such a useful and established algorithm, that it is both a statistical model and a machine learning model. Linear regression tries a draw a best fit line that is close to the data by finding the slope and intercept.

Linear regression equation is,

Y=a+bx

In this equation:

- y is the output variable. It is also called the target variable in machine learning or the dependent variable.
- x is the input variable. It is also referred to as the feature in machine learning or it is called the independent variable.
- a is the constant

- b is the coefficient of independent variable

2.2 Multiple linear regression

Multiple Linear Regression assumes there is a linear relationship between two or more independent variables and one dependent variable.

The Formula for multiple linear regression:

Y=B0+B0X1+B2X2+……+BnXn+e

- **Y** = the predicted value of the dependent variable
- **B0** = the y-intercept (value of y when all other parameters are set to 0)
- **B1X1**= the regression coefficient (B1) of the first independent variable (**X1**)
- **BnXn** = the regression coefficient of the last independent variable
- **e** = model error

2.3 Cost-function

The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.

2.4 Gradient Descent

It is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

2.4.1 Types of Gradient Descent

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into **Batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.** Let's understand these different types of gradient descent:

1. Batch Gradient Descent:

Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

Advantages of Batch gradient descent:

- It produces less noise in comparison to other gradient descent.
- It produces stable gradient descent convergence.
- It is Computationally efficient as all resources are used for all training samples.

2. Stochastic gradient descent

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration.

3. MiniBatch Gradient Descent:

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

Where,

$\hat{y}$ $-$ *predicted value of y*

$\bar{y}$ $-$ *mean value of y*

- **Mean Squared Error** represents the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$

- **Root Mean Squared Error** is the square root of Mean Squared error. It measures the standard deviation of residuals.

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

- The coefficient of determination or R-squared represents the proportion of the variance in the dependent variable which is explained by the linear regression model. It is a scale-free score i.e. irrespective of the values being small or large, the value of R square will be less than one.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

- **Adjusted R** squared is a modified version of R square, and it is adjusted for the number of independent variables in the model, and it will always be less than or equal to R^2.In the formula below **n** is the number of observations in the data and **k** is the number of the independent variables in the data.

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n-1)}{n-k-1} \right]$$

2.5 Evaluation Metrics

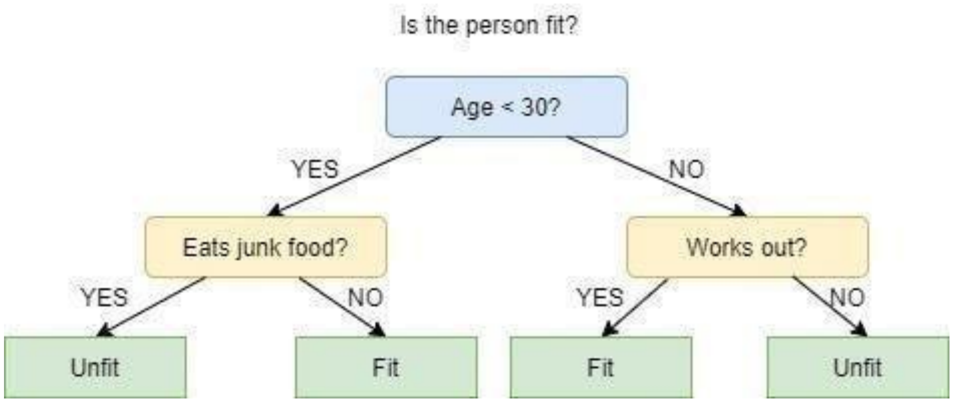
- Mean Squared Error(MSE) and Root Mean Square Error penalizes the large prediction errors vi-a-vis Mean Absolute Error (MAE). However, RMSE is widely used than MSE to evaluate the performance of the regression model with other random models as it has the same units as the dependent variable (Y-axis).
- MSE is a differentiable function that makes it easy to perform mathematical operations in comparison to a non-differentiable function like MAE. Therefore, in many models, RMSE is used as a default metric for calculating Loss Function despite being harder to interpret than MAE.
- The lower value of MAE, MSE, and RMSE implies higher accuracy of a regression model. However, a higher value of R square is considered desirable.
- **R Squared & Adjusted R** Squared are used for explaining how well the independent variables in the linear regression model explains the variability in the dependent variable. R Squared value always increases with the addition of the independent variables which might lead to the addition of the redundant variables in our model. However, the adjusted R-squared solves this problem.
- Adjusted R squared takes into account the number of predictor variables, and it is used to determine the number of independent variables in our model. The value of Adjusted R squared decreases if the increase in the R square by the additional variable isn’t significant enough.

- For comparing the accuracy among different linear regression models, RMSE is a better choice than R Squared.

2.6 Decision Trees

In simple words, a decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponds to records). Each node is either used to *make a decision (*known as decision node) or *represent an outcome* (known as leaf node).

Decision tree Example



The picture above depicts a decision tree that is used to classify whether a person is **Fit** or **Unfit.**

The decision nodes here are questions like '' *'Is the person less than 30 years of age?'*, *'Does the person eat junk?'*, etc. and the leaves are one of the two possible outcomesviz. **Fit** and **Unfit**.

Looking at the Decision Tree we can say make the following decisions: if a person is less than 30 years of age and doesn't eat junk food then he is Fit, if a person is less than 30 years of age and eats junk food then he is Unfit and so on.

The initial node is called the **root node** *(colored in blue)*, the final nodes are called the **leaf nodes** *(colored in green)* and the rest of the nodes are called **intermediate** or **internal** nodes. The root and intermediate nodes represent the decisions while the leaf nodes represent the outcomes.

2.6.1 ID3

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.

Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.

Most generally ID3 is only used for classification problems with nominal features only.

ID3 Steps

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset **S** into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the classas its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

2.6.2 CART Algorithm

The CART algorithm works via the following process:

- The best split point of each input is obtained.
- Based on the best split points of each input in Step 1, the new "best" split point is identified.
- Split the chosen input according to the "best" split point.
- Continue splitting until a stopping rule is satisfied or no further desirable splitting is available.

CART algorithm uses Gini Impurity to split the dataset into a decision tree .It does that by searching for the best homogeneity for the sub nodes, with the help of the Gini index criterion.

Gini index/Gini impurity

The Gini index is a metric for the classification tasks in CART. It stores the sum of squared probabilities of each class. It computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient. It works on categorical variables, provides outcomes either "successful" or "failure" and hence conducts binary splitting only.

The degree of the Gini index varies from 0 to 1,

- Where 0 depicts that all the elements are allied to a certain class, or only one class exists there.
- The Gini index of value 1 signifies that all the elements are randomly distributed across various classes, and
- A value of 0.5 denotes the elements are uniformly distributed into some classes.

Classification tree

A classification tree is an algorithm where the target variable is categorical. The algorithm is then used to identify the “Class” within which the target variable is most likely to fall. Classification trees are used when the dataset needs to be split into classes that belong to the response variable(like yes or no)

Regression tree

A Regression tree is an algorithm where the target variable is continuous and the tree is used to predict its value. Regression trees are used when the response variable is continuous. For example, if the response variable is the temperature of the day.

Pseudo-code of the CART algorithm

```
d = 0, endtree = 0
Note(0) = 1, Node(1) = 0, Node(2) = 0
while endtree < 1
   if Node($2^d$-1) + Node($2^d$) +..... + Node($2^{d+1}$-2) = 2 - $2^{d+1}$
      endtree = 1
   else
      do i = $2^d$-1, $2^d$, ..... , $2^{d+1}$-2
         if Node(i) > -1
            Split tree
         else
            Node(2i+1) = -1
            Node(2i+2) = -1
         end if
      end do
   end if
d = d + 1
end while
```

CART model representation

CART models are formed by picking input variables and evaluating split points on those variables until an appropriate tree is produced.

Steps to create a Decision Tree using the CART algorithm:

- ***Greedy algorithm***: In this The input space is divided using the Greedy method which is known as a recursive binary spitting. This is a numerical method within which all of the values are aligned and several other split points are tried and assessed using a cost function.
- ***Stopping Criterion:*** As it works its way down the tree with the training data, the recursive binary splitting method described above must know when to stop splitting. The most frequent halting method is to utilize a minimum amount of training data allocated to every leaf node. If the count is smaller than the specified threshold, the split is rejected and also the node is considered the last leaf node.
- ***Tree pruning:*** Decision tree's complexity is defined as the number of splits in the tree. Trees with fewer branches are recommended as they are simple to grasp and less prone to cluster the data. Working through each leaf node in the tree and evaluating the effect of deleting it using a hold-out test set is the quickest and simplest pruning approach.
- ***Data preparation for the CART*****:** No special data preparation is required for the CART algorithm.

2.7 Naïve Bayes Classifier Algorithm

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

Why is it called Naïve Bayes?

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

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. 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:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

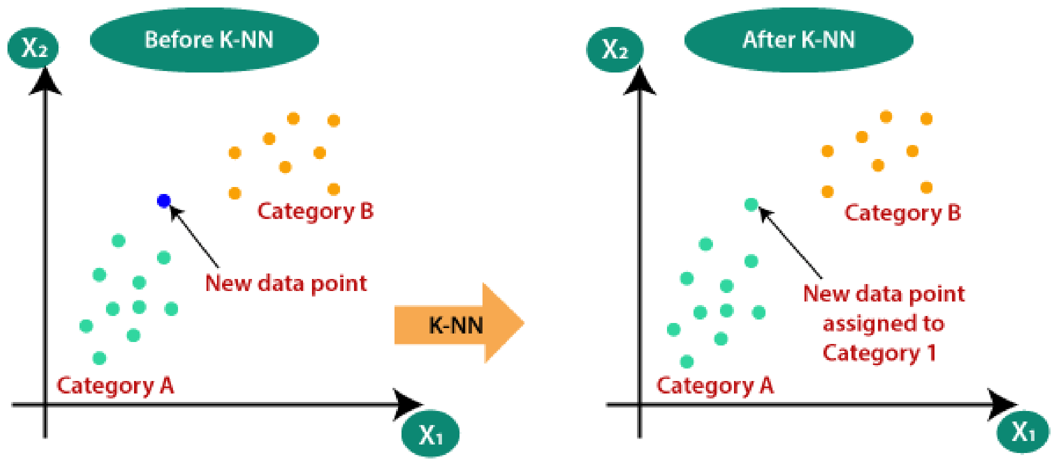
1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

2.8 K-Nearest Neighbor(KNN) Algorithm

- 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.

<u>Why do we need a K-NN Algorithm?</u>

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:

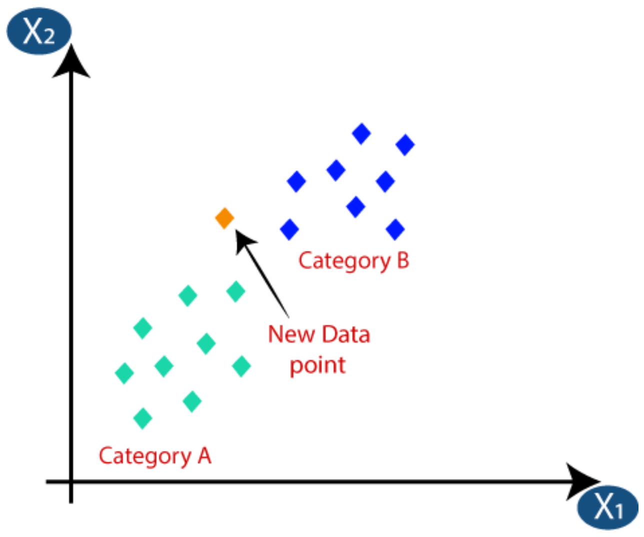


How does K-NN work?

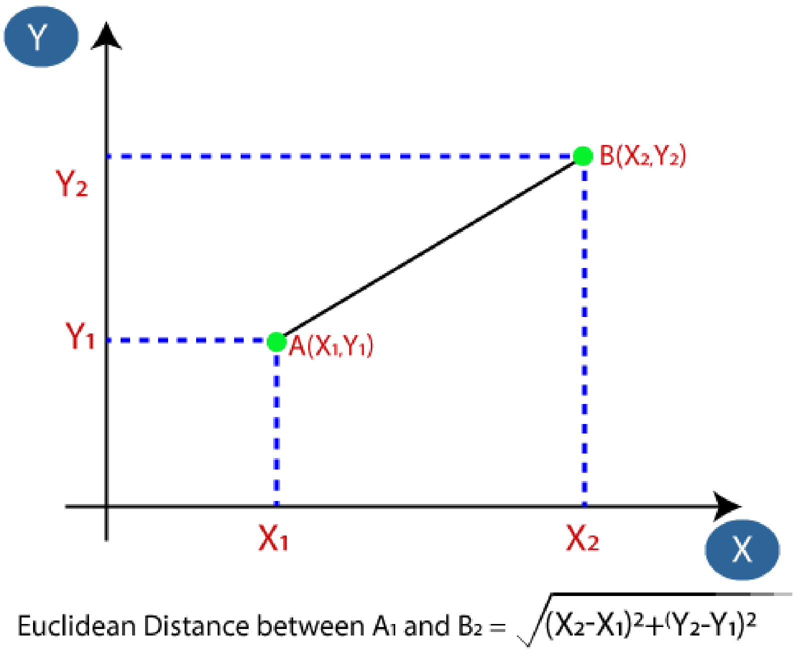
The K-NN working can be explained on the basis of the below algorithm:

- **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.

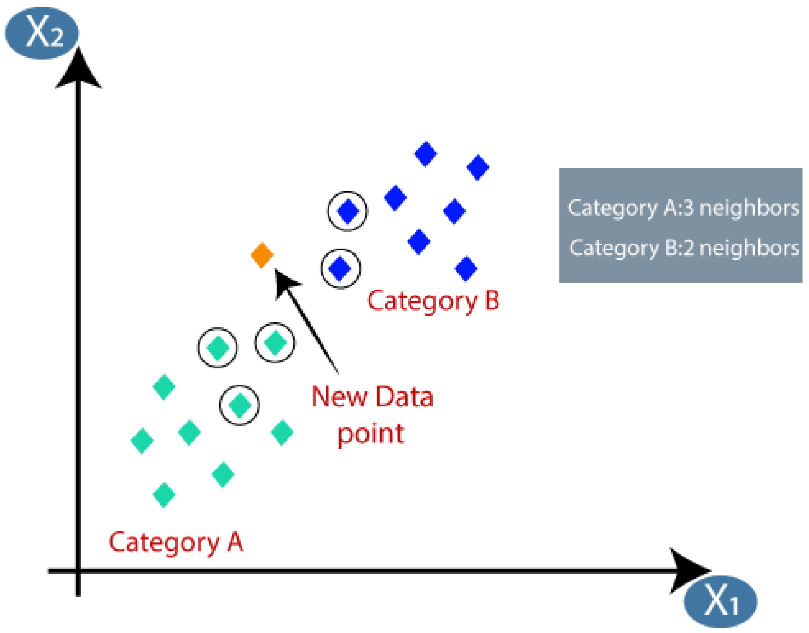
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- 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:


Y
X
Y_2
Y_1
X_1
X_2
$B(X_2,Y_2)$
$A(X_1,Y_1)$
Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- 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.

<u>How to select the value of K in the K-NN Algorithm?</u>

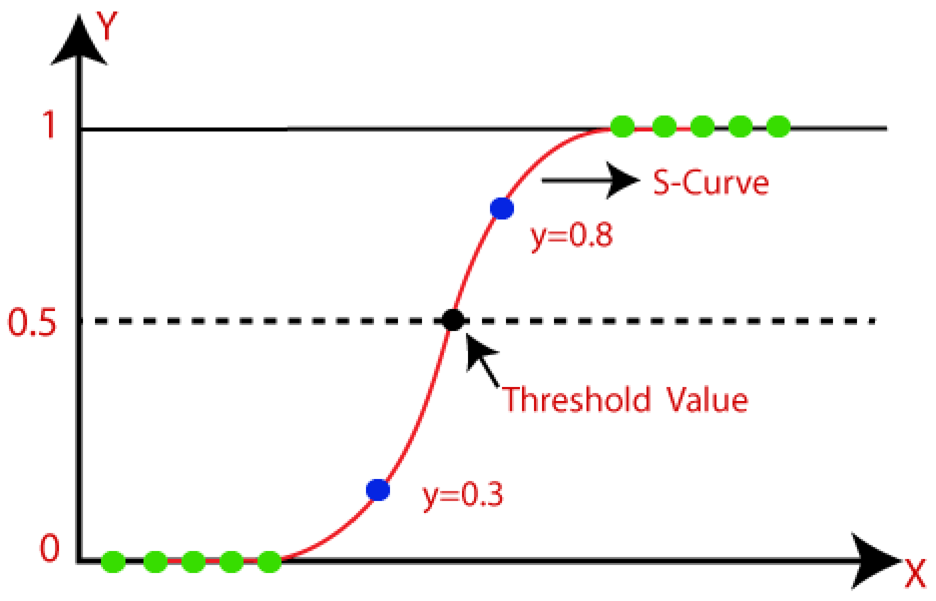
Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.

- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

2.9 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.

Type 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".

2.10 Multinomial Logistic Regression

Multinomial Logistic Regression is a classification technique that extends the logistic regression algorithm to solve multiclass possible outcome problems, given one or more independent variables.

Example for Multinomial Logistic Regression:

(a) Which Flavor of ice cream will a person choose?

Dependent Variable:

- Vanilla
- Chocolate
- Butterscotch
- Black Current

Independent Variables:

- Gender
- Age
- Occasion
- Happiness
- Etc.

Multinomial Logistic Regression is also known as multiclass logistic regression, softmax regression, polytomous logistic regression, multinomial logit, maximum entropy (MaxEnt) classifier and conditional maximum entropy model.

Dependent Variable:

The dependent Variable can have two or more possible outcomes/classes.

The dependent variables are nominal in nature means there is no any kind of ordering in target dependent classes i.e. these classes cannot be meaningfully ordered.

The dependent variable to be predicted belongs to a limited set of items defined.

Basic Steps

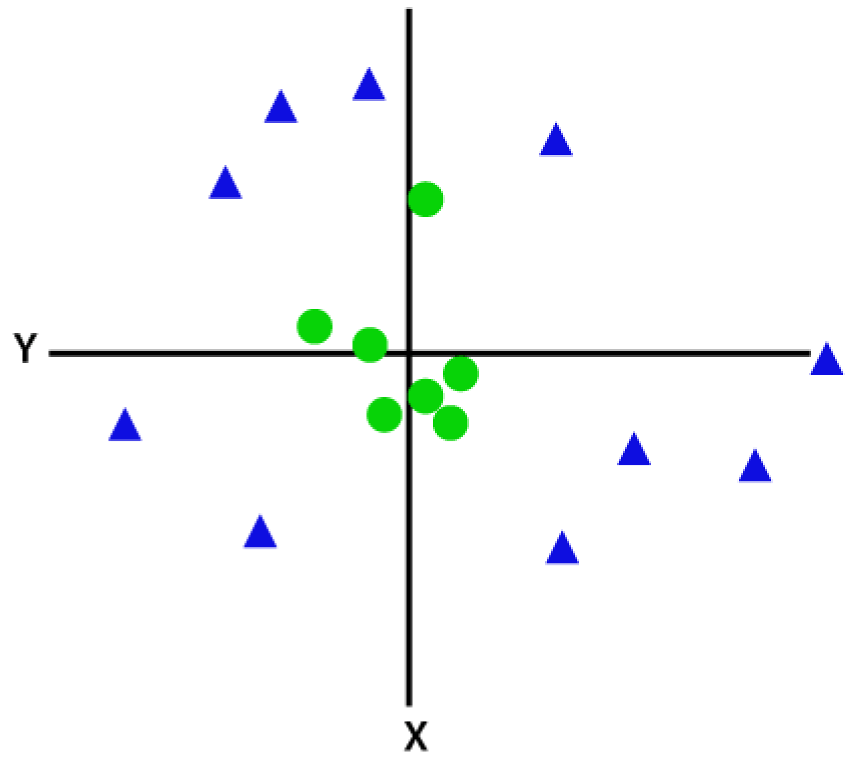
The basic steps of the SVM are:

1. select **two hyperplanes** (in 2D) which separates the data **with no points between them** (red lines)
2. **maximize their distance** (the margin)
3. the **average line** (here the line half way between the two red lines) will be the **decision boundary**

This is very nice and easy, but finding the best margin, the optimization problem is not trivial (it is easy in 2D, when we have only two attributes, but what if we have N dimensions with N a very big number).

Non-Linear SVM:

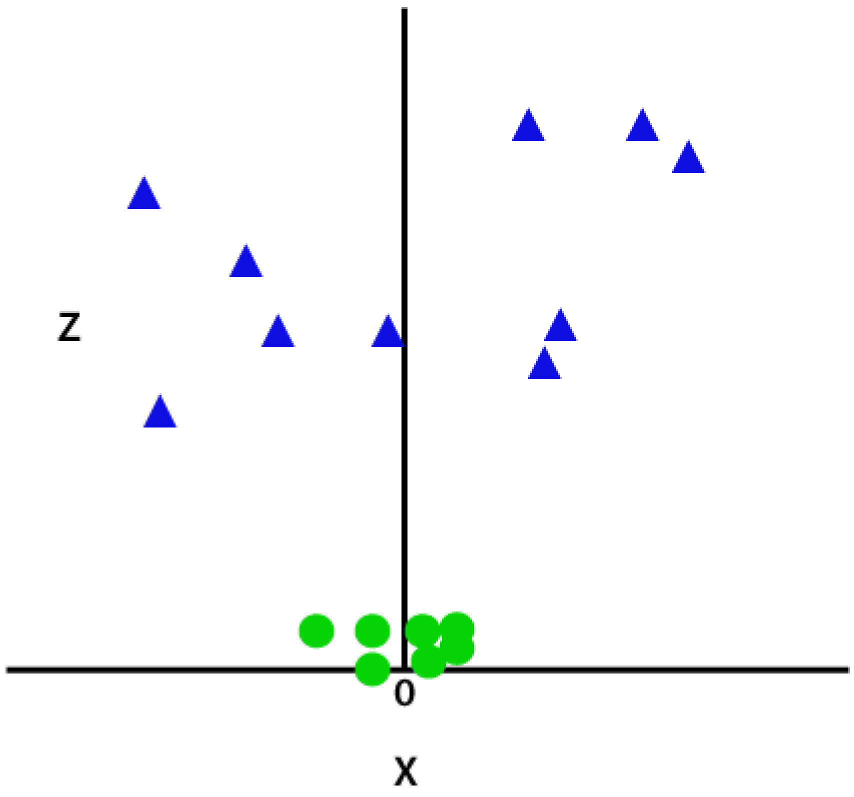
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



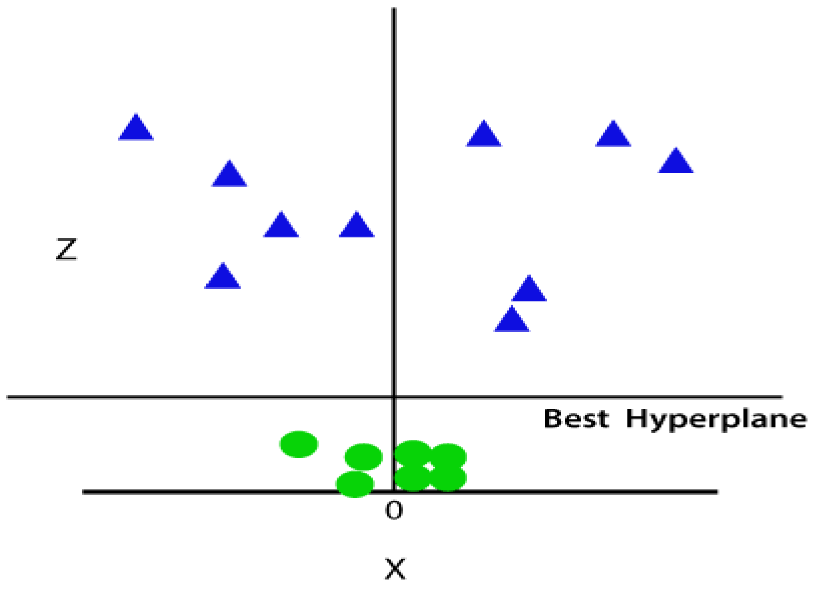
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

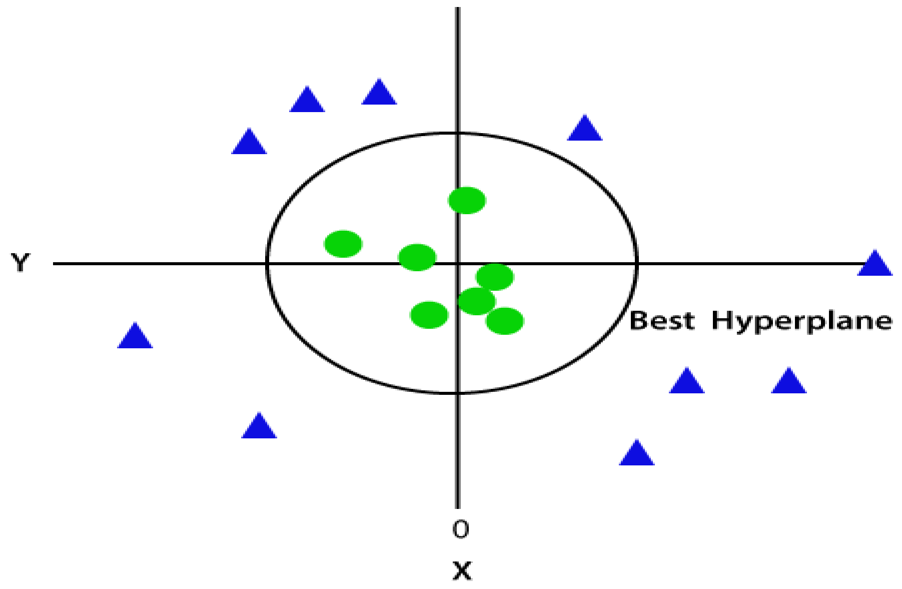
By adding the third dimension, the sample space will become as below image:



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.

Kernel Methods

Kernels or kernel methods (also called Kernel functions) are sets of different types of algorithms that are being used for pattern analysis. They are used to solve a non-linear problem by using a linear classifier. Kernels Methods are employed in SVM (Support Vector Machines) which are used in classification and regression problems. The SVM uses what is called a “Kernel Trick” where the data is transformed and an optimal boundary is found for the possible outputs.

The Need for Kernel Method and its Working

Before we get into the working of the Kernel Methods, it is more important to understand support vector machines or the SVMs because kernels are implemented in SVM models. So, Support Vector Machines are supervised **machine learning <u>algorithms</u>** that are used in classification and regression problems such as classifying an apple to class fruit while classifying a Lion to the class animal.

we have 2 dimension which represents the ambient space but the lone which divides or classifies the space is one dimension less than the ambient space and is called hyperplane.

But what if we have input like this:

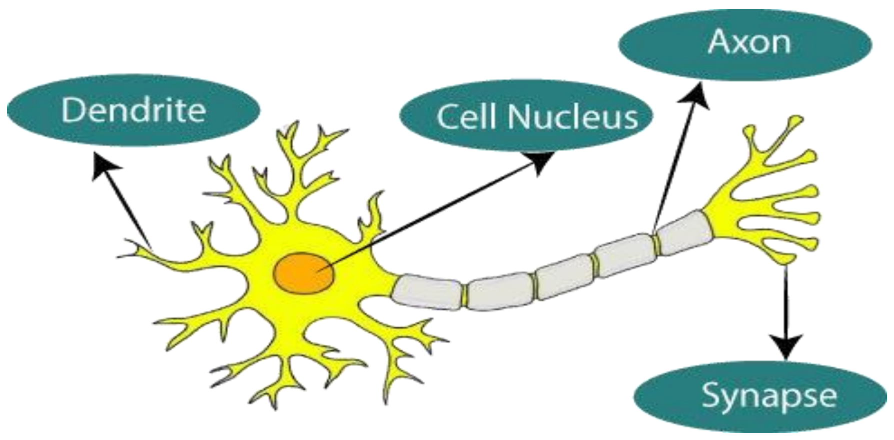
It is very difficult to solve this classification using a linear classifier as there is no good linear line that should be able to classify the red and the green dots as the points are randomly

UNIT – III

Supervised Learning – II (Neural Networks)
Neural Network Representation – Problems – Perceptrons , Activation Functions, ArtificialNeural Networks (ANN) , Back Propagation Algorithm.
Convolutional Neural Networks - Convolution and Pooling layers, , Recurrent NeuralNetworks (RNN).

3.1 Neural Network Representation

The term " **Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are

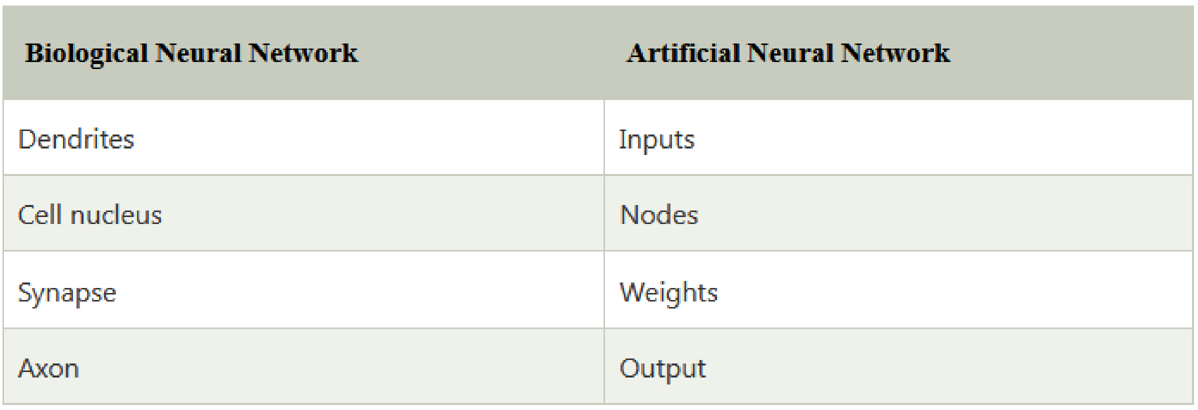


interconnected to one another in various layers of the networks. These neurons are known as nodes.

The given figure illustrates the typical diagram of Biological Neural Network.
The typical Artificial Neural Network looks something like the given figure

Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

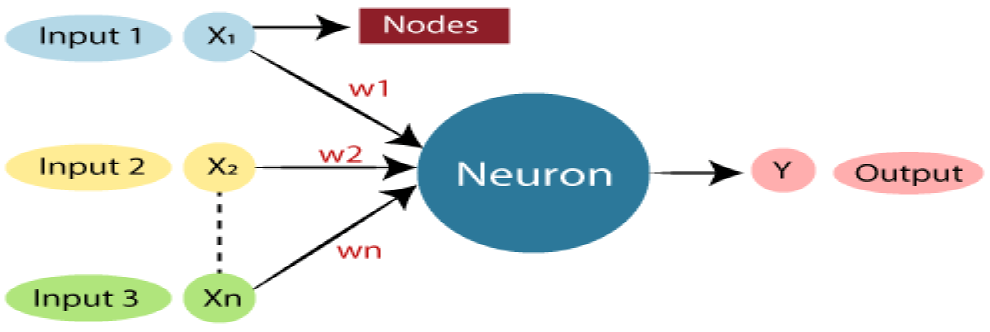


Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

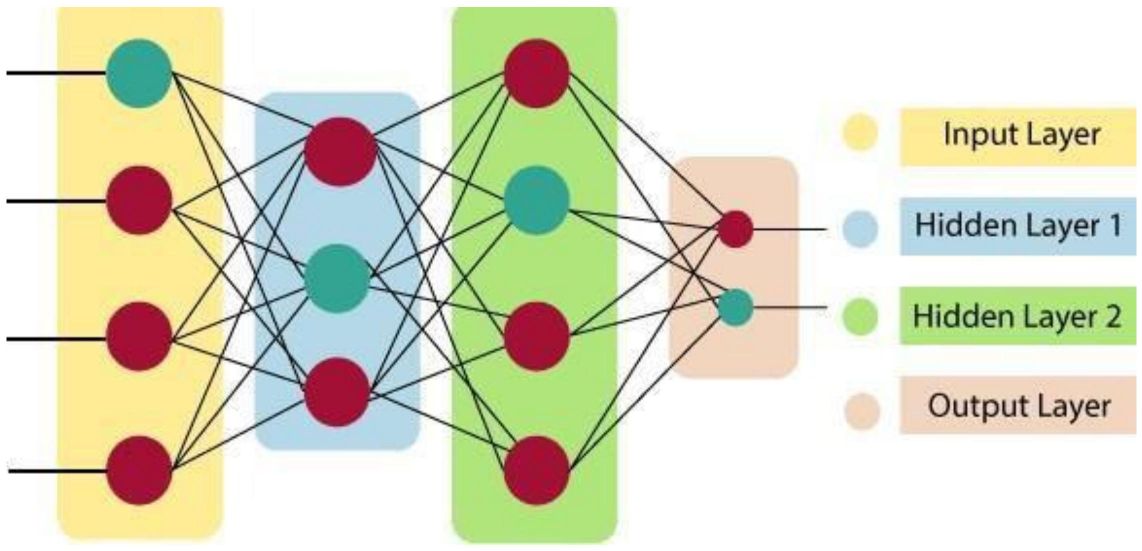
An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input.



Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."



The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:

Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Perceptrons

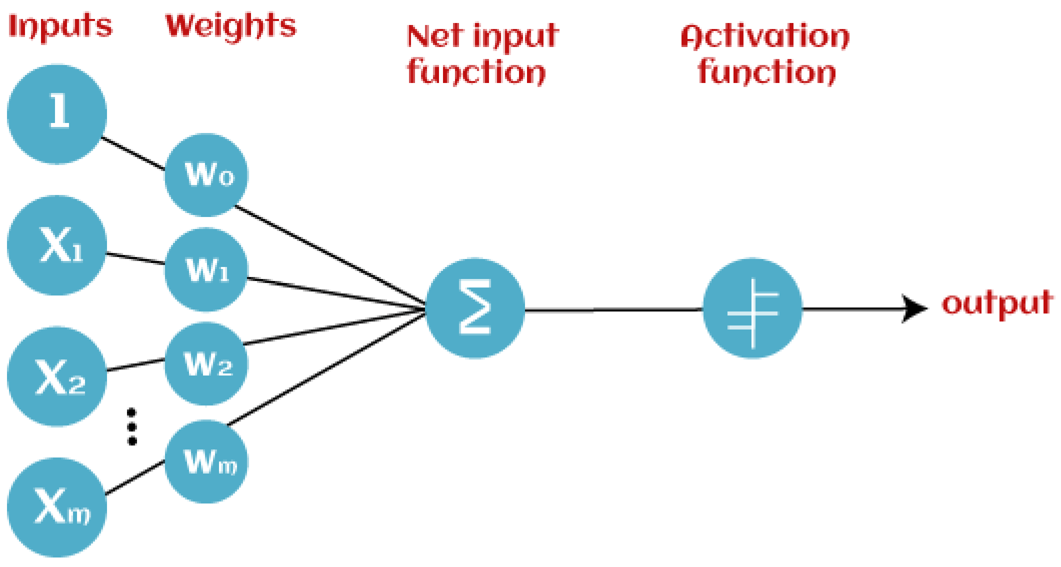
Perceptron is a building block of an Artificial Neural Network. Initially, in the mid of 19th century, **Mr. Frank Rosenblatt** invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence. Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers. This algorithm enables neurons to learn elements and processes them one by one during preparation. In this tutorial, "Perceptron in Machine Learning," we will discuss in-depth knowledge of Perceptron and its basic functions in brief. Let's start with the basic introduction of Perceptron..

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

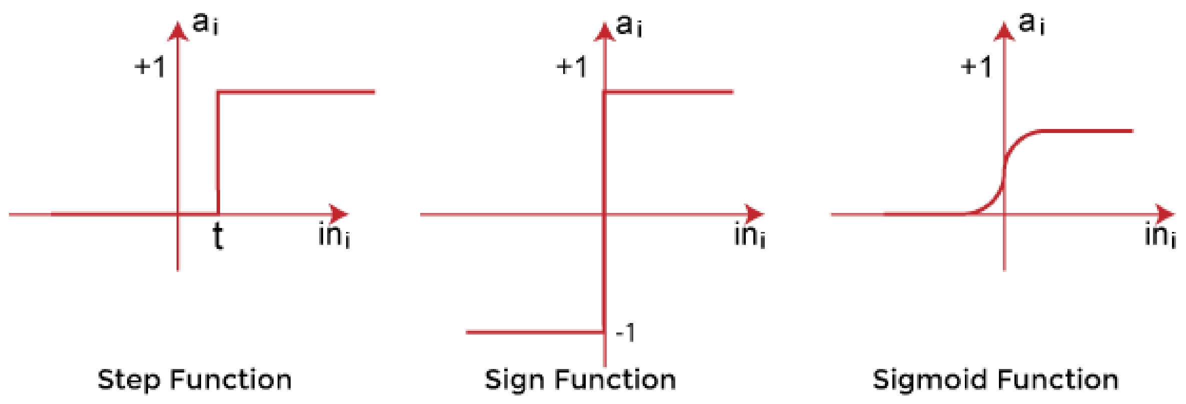
- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and

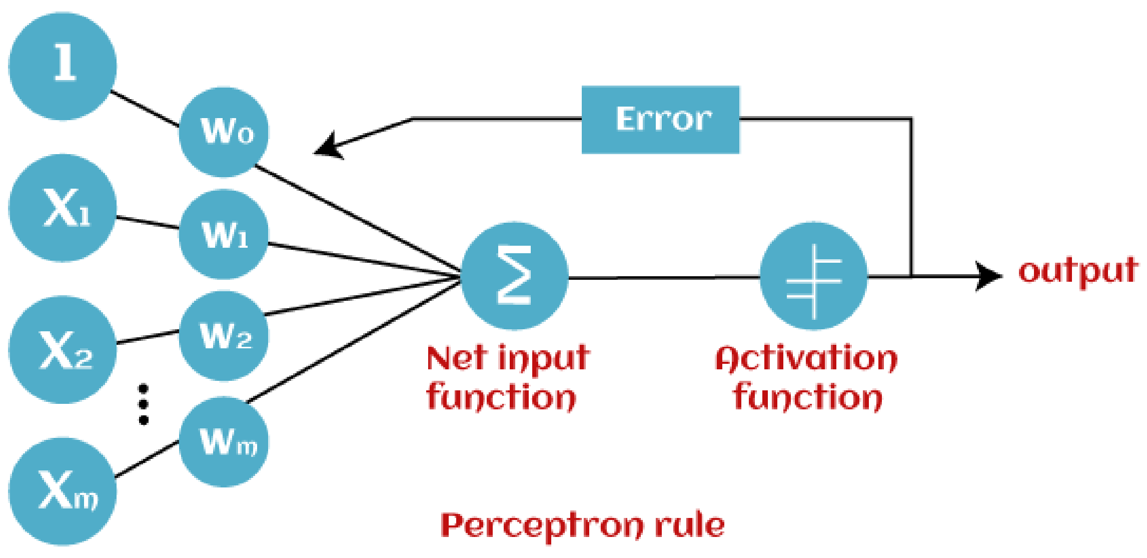
- Sigmoid function



The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.



This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$\boldsymbol{\sum wi*xi + b}$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

Y = f(∑wi*xi + b)

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

"Single-layer perceptron can learn only linearly separable patterns.".

Multi-Layered Perceptron Model:

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

f(x)=1; if w.x+b>0

otherwise, f(x)=0

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

Limitations of Perceptron Model

A perceptron model has limitations as follows:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

Activation Functions

Activation function also helps to normalize the output of any input in the range between **1 to -1**. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points.

Without an activation function, a neural network will become a linear regression model. But introducing the activation function the neural network will perform a non-linear transformation to the input and will be suitable to solve problems like image classification, sentence prediction, or langue translation.

The neuron is basically is a weighted average of input, then this sum is passed through an activation function to get an output.

$$Y = \sum (weights*input + bias)$$

Here Y can be anything for a neuron between range **-infinity to +infinity**. So, we have to bound our output to get the desired prediction or generalized results.

$$Y = \text{Activation function}(\sum (weights*input + bias))$$

So, we pass that neuron to activation function to bound output values.

Why do we need Activation Functions

Without activation function, weight and bias would only have a linear transformation, or neural network is just a linear regression model, a linear equation is polynomial of one degree only which is simple to solve but limited in terms of ability to solve complex problems or higher degree polynomials.

But opposite to that, the addition of activation function to neural network executes the non-linear transformation to input and make it capable to solve complex problems such as language translations and image classifications.

In addition to that, Activation functions are differentiable due to which they can easily implement back propagations, optimized strategy while performing backpropagations to measure gradient loss functions in the neural networks.

Types of Activation Functions

The two main categories of activation functions are:

- Linear Activation Function
- Non-linear Activation Functions

Linear Neural Network Activation Function

Linear Activation Function

Equation: A linear function's equation, which is y = x, is similar to the eqn of a single direction.

The ultimate activation function of the last layer is nothing more than a linear function of input from the first layer, regardless of how many levels we have if they are all linear in nature. -inf to +inf is the range.

Uses: The output layer is the only location where the activation function's function is applied.

If we separate a linear function to add non-linearity, the outcome will no longer depend on the input "x," the function will become fixed, and our algorithm won't exhibit any novel behaviour.

A good example of a regression problem is determining the cost of a house. We can use linear activation at the output layer since the price of a house may have any huge or little value. The neural network's hidden layers must perform some sort of non-linear function even in this circumstance.

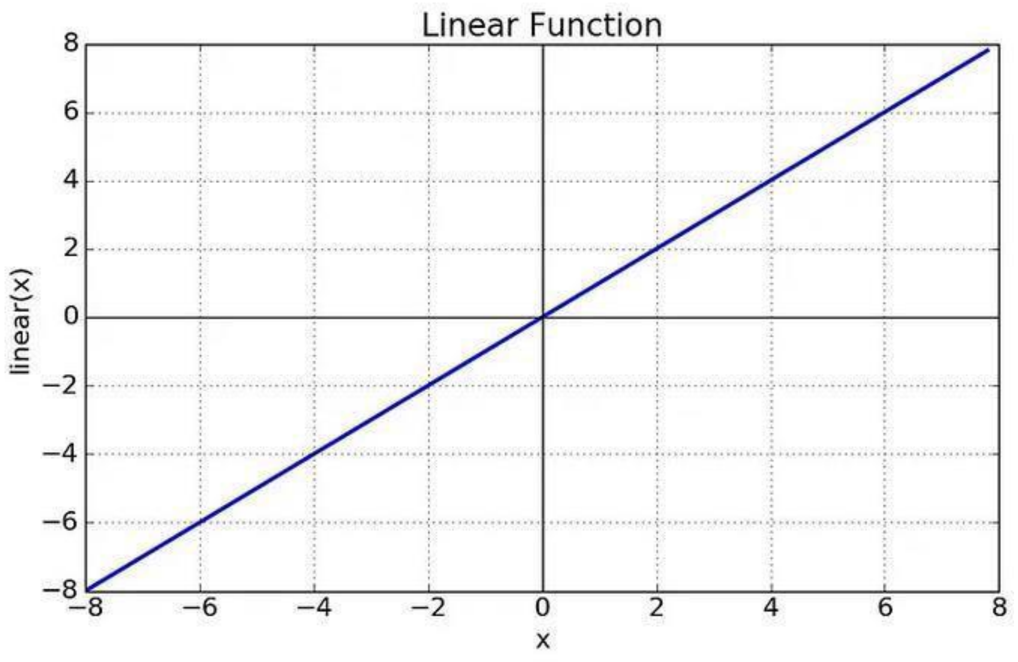


Fig: Linear Activation Function

Equation : f(x) = x

Range : (-infinity to infinity)

It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

<u>Non Linear Neural Network Activation Function</u>

1. Sigmoid or Logistic Activation Function

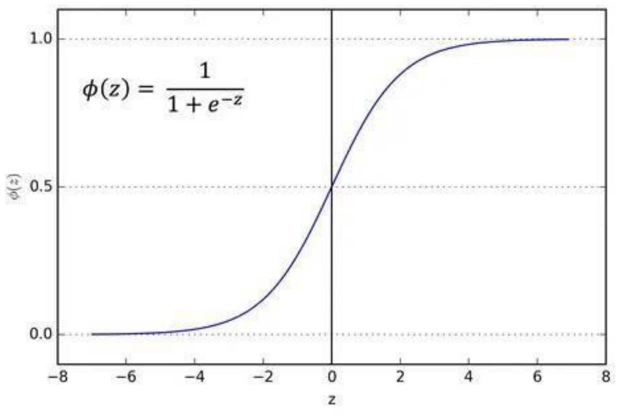


Fig: Sigmoid Function

It is a functional that is graphed in a "S" shape.

A is equal to 1/(1 + e-x).

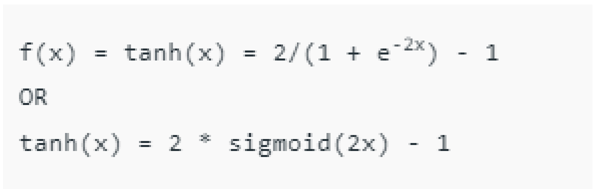
Non-linear in nature. Observe that while Y values are fairly steep, X values range from -2 to 2. To put it another way, small changes in x also would cause significant shifts in the value of Y. spans from 0 to 1.

Uses: Sigmoid function is typically employed in the output nodes of a classi?cation, where the result may only be either 0 or 1. Since the value for the sigmoid function only ranges from 0 to 1, the result can be easily anticipated to be 1 if the value is more than 0.5 and 0 if it is not.

- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** A = 1/(1 + e^{-x})
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function

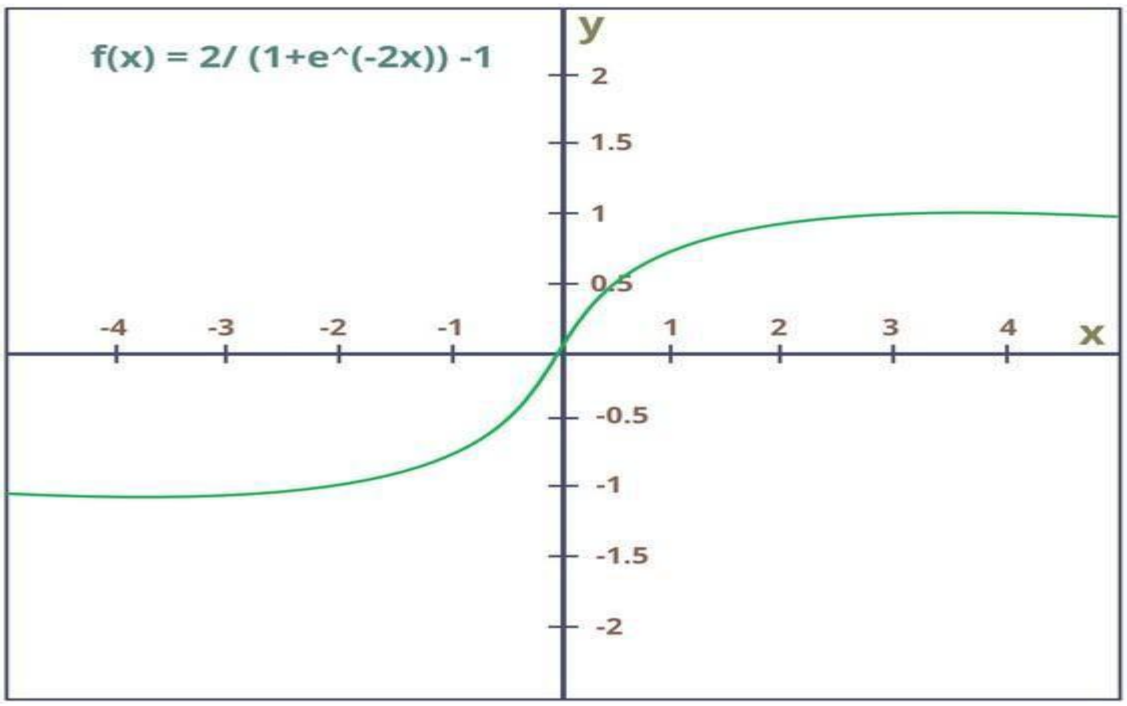
The activation that consistently outperforms sigmoid function is known as tangent hyperbolic function. It's actually a sigmoid function that has been mathematically adjusted. Both are comparable to and derivable from one another.



```
f(x) = tanh(x) = 2/(1 + e^(-2x)) - 1
OR
tanh(x) = 2 * sigmoid(2x) - 1
```

Range of values: -1 to +1. non-linear nature

Uses: - Since its values typically range from -1 to 1, the mean again for hidden layer of a neural network will be 0 or very near to it. This helps to centre the data by getting the mean close to 0. This greatly facilitates learning for the following layer.



- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

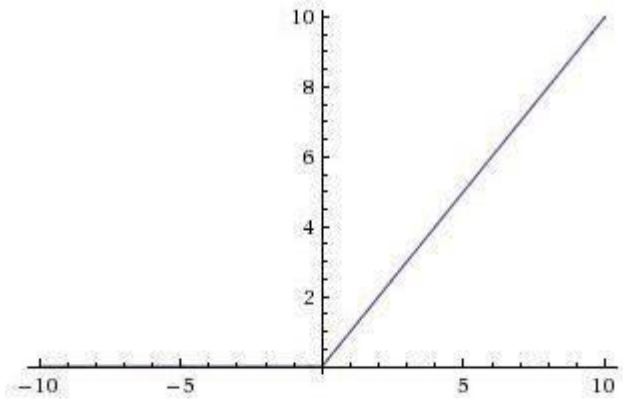
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

ReLU (Rectified Linear Unit) Activation Function

Currently, the ReLU is the activation function that is employed the most globally. Since practically all convolutional neural networks and deep learning systems employ it.

The derivative and the function are both monotonic.

However, the problem is that all negative values instantly become zero, which reduces the model's capacity to effectively fit or learn from the data. This means that any negative input to a ReLU activation function immediately becomes zero in the graph, which has an impact on the final graph by improperly mapping the negative values.



- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function

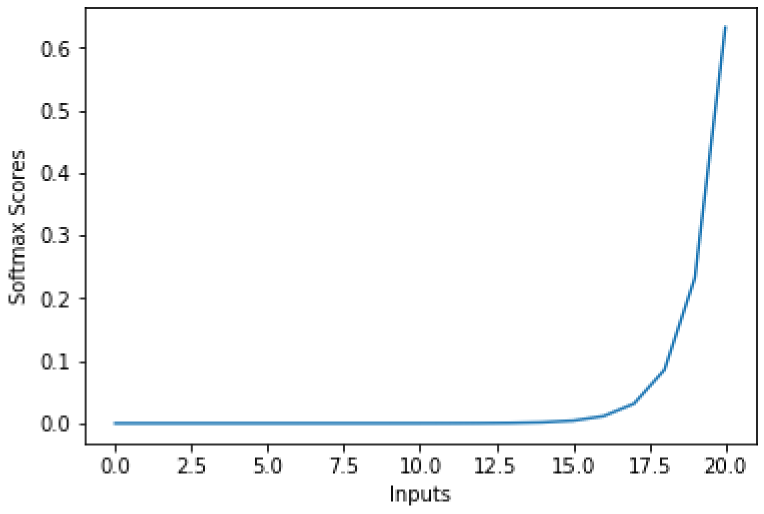
Although it is a subclass of the sigmoid function, the softmax function comes in handy when dealing with multiclass classification issues.

Used frequently when managing several classes. In the output nodes of image classification issues, the softmax was typically present. The softmax function would split by the sum of the outputs and squeeze all outputs for each category between 0 and 1.

The output unit of the classifier, where we are actually attempting to obtain the probabilities to determine the class of each input, is where the softmax function is best applied.

The usual rule of thumb is to utilise RELU, which is a usual perceptron in hidden layers and is employed in the majority of cases these days, if we really are unsure of what encoder to apply.

A very logical choice for the output layer is the sigmoid function if your input is for binary classification. If our output involves multiple classes, Softmax can be quite helpful in predicting the odds for each class.



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax

function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

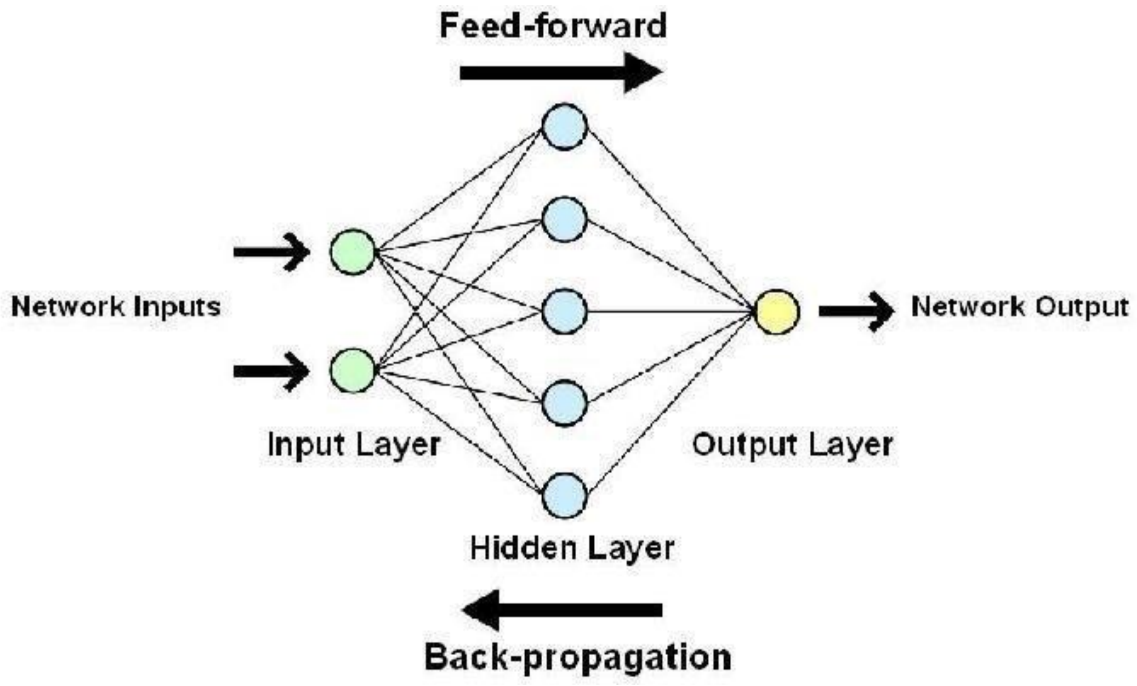
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

3.2 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) are algorithms based on brain function and are used to model complicated patterns and forecast issues. The Artificial Neural Network (ANN) is a deep learning method that arose from the concept of the human brain Biological Neural Networks. The development of ANN was the result of an attempt to replicate the workings of the human brain. The workings of ANN are extremely similar to those of biological neural networks, although they are not identical. ANN algorithm accepts only numeric and structured data.

Artificial Neural Networks Architecture

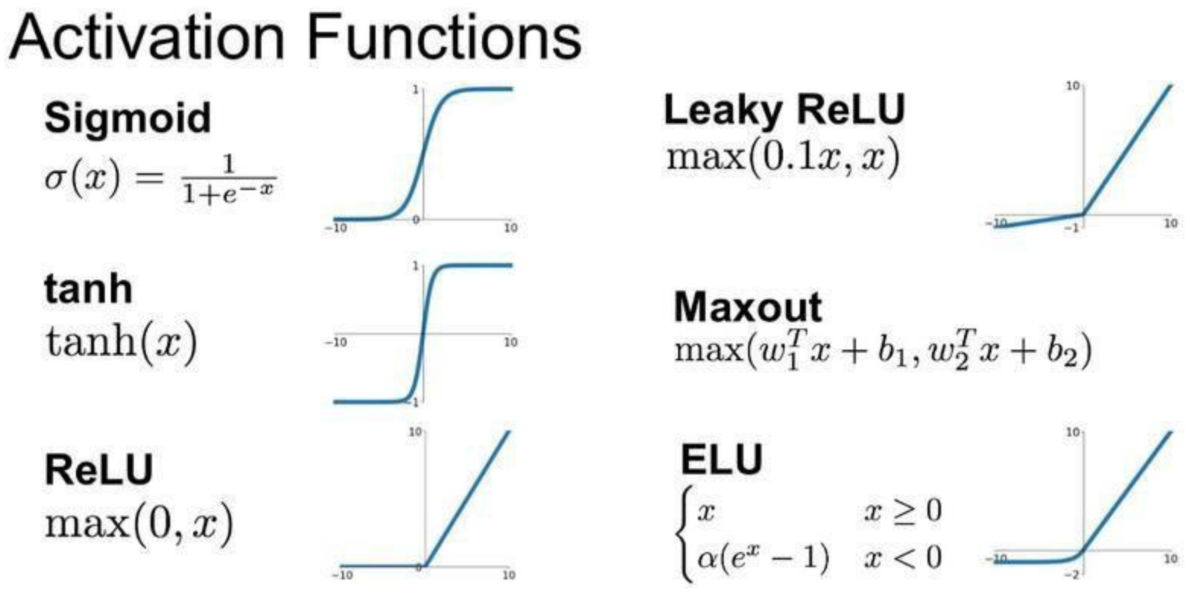
1. There are three layers in the network architecture: the input layer, the hidden layer (more than one), and the output layer. Because of the numerous layers are sometimes referred to as the MLP (Multi-Layer Perceptron).



2. It is possible to think of the hidden layer as a "distillation layer," which extracts some of the most relevant patterns from the inputs and sends them on to the next layer for further analysis. It accelerates and improves the efficiency of the network by recognizing just the most important information from the inputs and discarding the redundant information.

3. The activation function is important for two reasons:

- This model captures the presence of non-linear relationships between the inputs.
- It contributes to the conversion of the input into a more usable output.

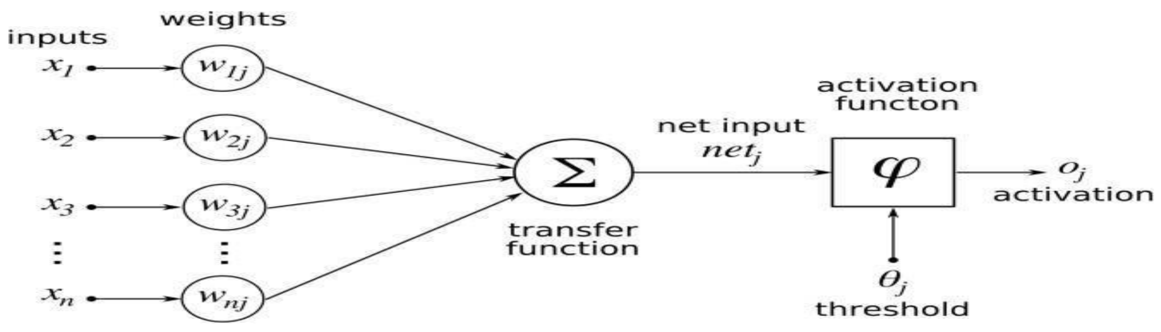


4. Finding the "optimal values of W — weights" that minimize prediction error is critical to building a successful model. The "backpropagation algorithm" does this by converting ANN into a learning algorithm by learning from mistakes.

5. The optimization approach uses a "gradient descent" technique to quantify prediction errors. To find the optimum value for W, small adjustments in W are tried, and the impact on prediction errors is examined. Finally, those W values are chosen as ideal since further W changes do not reduce mistakes.

How artificial neural networks functions

The core component of ANNs is artificial neurons. Each neuron receives inputs from several other neurons, multiplies them by assigned weights, adds them and passes the sum to one or more neurons. Some artificial neurons might apply an activation function to the output before passing it to the next variable.



At its core, this might sound like a very trivial math operation. But when you place hundreds, thousands and millions of neurons in multiple layers and stack them up on top of each other, you'll obtain an artificial neural network that can perform very complicated tasks, such as classifying images or recognizing speech.

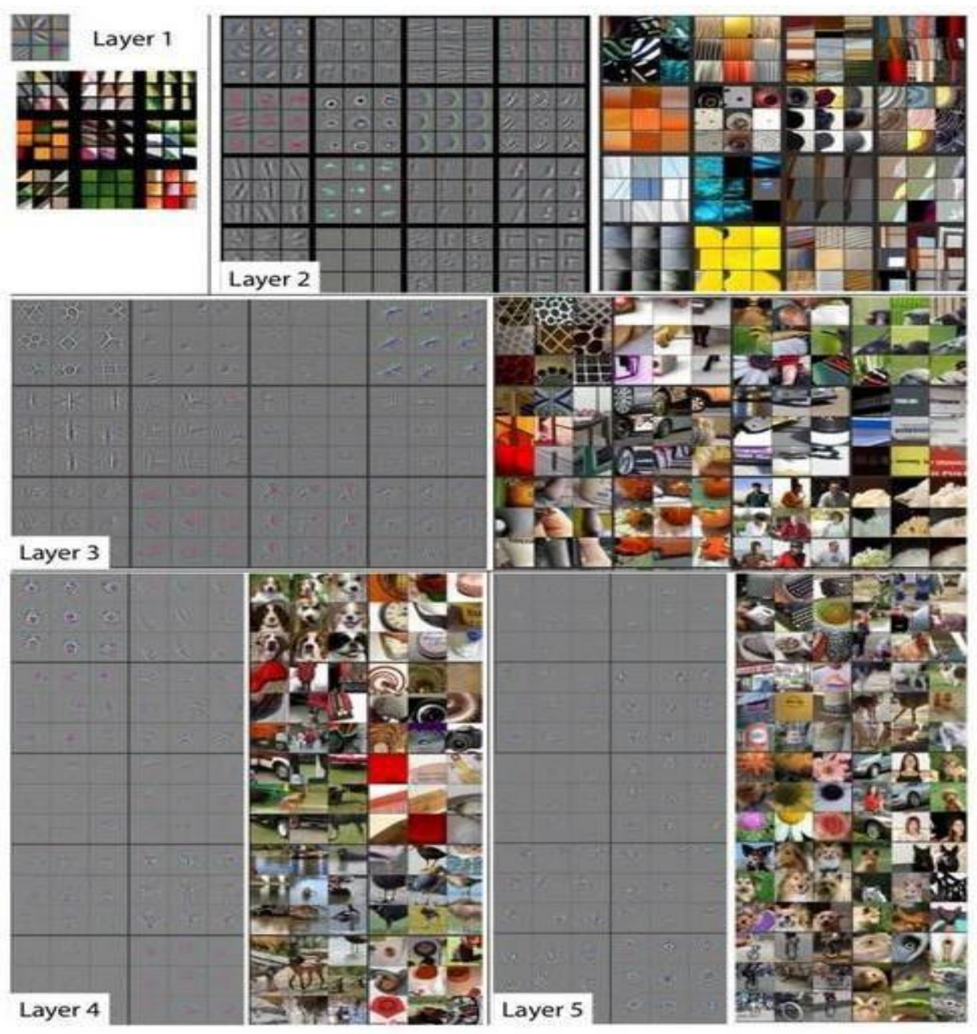
Artificial neural networks are composed of an input layer, which receives data from outside sources (data files, images, hardware sensors, microphone…), one or more hidden layers that process the data, and an output layer that provides one or more data points based on the function of the network. For instance, a neural network that detects persons, cars and animals will have an output layer with three nodes. A network that classifies bank transactions between safe and fraudulent will have a single output.

Training artificial neural networks

Artificial neural networks start by assigning random values to the weights of the connections between neurons. The key for the ANN to perform its task correctly and accurately is to adjust these weights to the right numbers. But finding the right weights is not very easy, especially when you're dealing with multiple layers and thousands of neurons.

This calibration is done by "training" the network with annotated examples. For instance, if you want to train the image classifier mentioned above, you provide it with multiple photos, each labeled with its corresponding class (person, car or animal). As you provide it with more and more training examples, the neural network gradually adjusts its weights to map each input to the correct outputs.

Basically, what happens during training is the network adjust itself to glean specific patterns from the data. Again, in the case of an image classifier network, when you train the AI model with quality examples, each layer detects a specific class of features. For instance, the first layer might detect horizontal and vertical edges, the next layers might detect corners and round shapes. Further down the network, deeper layers will start to pick out more advanced features such as faces and objects.



When you run a new image through a well-trained neural network, the adjusted weights of the neurons will be able to extract the right features and determine with accuracy to which output class the image belongs.

One of the challenges of training neural networks is to find the right amount and quality of training examples. Also, training large AI models requires vast amounts of computing resources. To overcome this challenge, many engineers use “transfer learning,” a training technique where you take a pre-trained model and fine-tune it with new, domain-specific examples. Transfer learning is especially efficient when there’s already an AI model that is close to your use case.

3.3 Backpropagation Algorithm

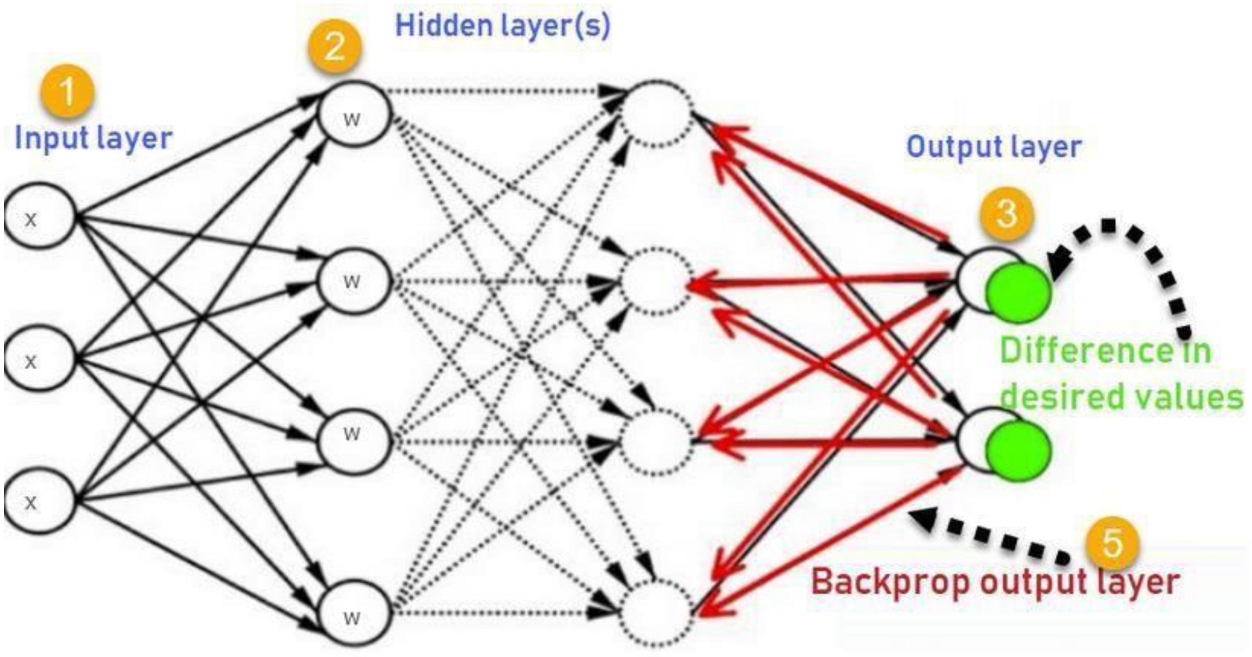
Backpropagation is an algorithm that backpropagates the errors from the output nodes to the input nodes. Therefore, it is simply referred to as the backward propagation of errors. It uses in the vast applications of neural networks in data mining like Character recognition, Signature verification, etc.

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

How Backpropagation Algorithm Works

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



How Backpropagation Algorithm Works

Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

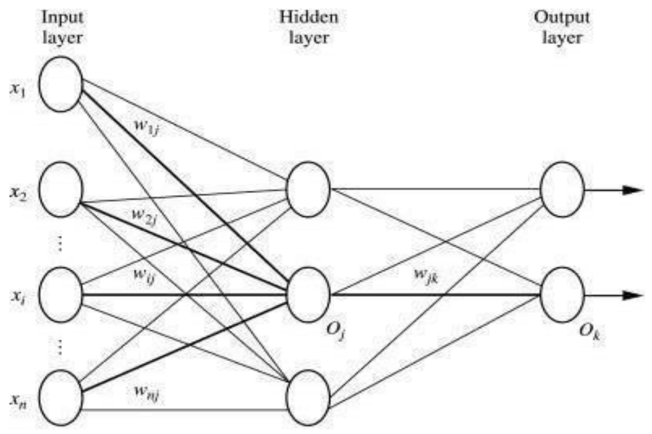
Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.



Parameters :

- x = inputs training vector x=(x_1,x_2,............x_n).
- t = target vector t=(t_1,t_2..............t_n).
- δ_k = error at output unit.
- δ_j = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j.

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the stepsstopping condition is to be false do step 3 to 10.

Step 3: For each training pair do step 4 to 9 (Feed-Forward).

Step 4: Each input unit receives the signal unit and transmitsthe signal x_i signal to all the units.

Step 5 : Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input

$$z_{inj} = v_{0j} + \Sigma x_i v_{ij} \quad (i=1 \text{ to } n)$$

Applying activation function $z_j = f(z_{inj})$ and sends this signals to all units in the layer about i.e output units

For each output l=unit y_k = (k=1 to m) sums its weighted input signals.

$$y_{ink} = w_{0k} + \Sigma\, z_i w_{jk} \quad (j=1 \text{ to } a)$$

and applies its activation function to calculate the output signals.

$$y_k = f(y_{ink})$$

Backpropagation Error :

Step 6: Each output unit y_k (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as:

$$\delta_k = (t_k – y_k) + y_{ink}$$

Step 7: Each hidden unit Z_j (j=1 to a) sums its input from all units in the layer above

$$\delta_{inj} = \Sigma\, \delta_j\, w_{jk}$$

The error information term is calculated as :

$$\delta_j = \delta_{inj} + z_{inj}$$

Updation of weight and bias :

Step 8: Each output unit y_k (k=1 to m) updates its bias and weight (j=1 to a). The weight correction term is given by :

$$\Delta w_{jk} = \alpha \delta_k z_j$$

and the bias correction term is given by $\Delta w_k = \alpha \delta_{k.}$

therefore $w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$

$$w_{0k(new)} = w_{ok(old)} + \Delta w_{ok}$$

for each hidden unit z_j (j=1 to a) update its bias and weights (i=0 to n) the weight connection term

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and the bias connection on term

$$\Delta v_{0j} = \alpha \delta_j$$

Therefore $v_{ij(new)} = v_{ij(old)} + \Delta v_{ij}$

$$v_{0j(new)} = v_{0j(old)} + \Delta v_{0j}$$

Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

Need for Backpropagation:

Backpropagation is “backpropagation of errors” and is very useful for training neural networks. It’s fast, easy to implement, and simple. Backpropagation does not require any parameters to be set, except the number of inputs. Backpropagation is a flexible method because no prior knowledge of the network is required.

Types of Backpropagation

There are two types of backpropagation networks.

- **Static backpropagation:** Static backpropagation is a network designed to map static inputs for static outputs. These types of networks are capable of solving static classification problems such as OCR (Optical Character Recognition).
- **Recurrent backpropagation:** Recursive backpropagation is another network used for fixed-point learning. Activation in recurrent backpropagation is feed-forward until a fixed value is reached. Static backpropagation provides an instant mapping, while recurrent backpropagation does not provide an instant mapping.

Advantages:

- It is simple, fast, and easy to program.
- Only numbers of the input are tuned, not any other parameter.
- It is Flexible and efficient.
- No need for users to learn any special functions.

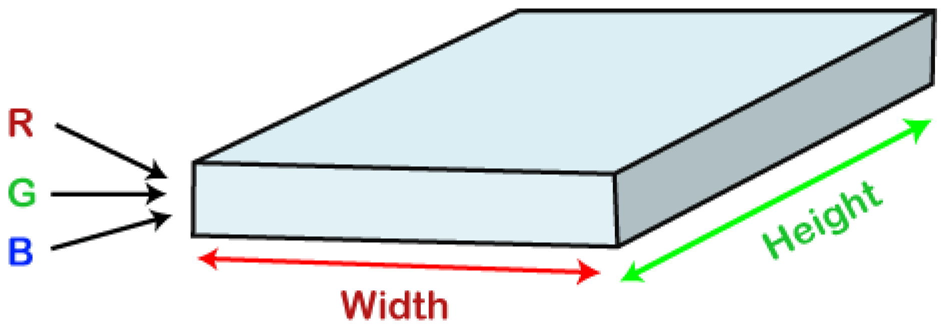
Disadvantages:

- It is sensitive to noisy data and irregularities. Noisy data can lead to inaccurate results.
- Performance is highly dependent on input data.
- Spending too much time training.
- The matrix-based approach is preferred over a mini-batch.

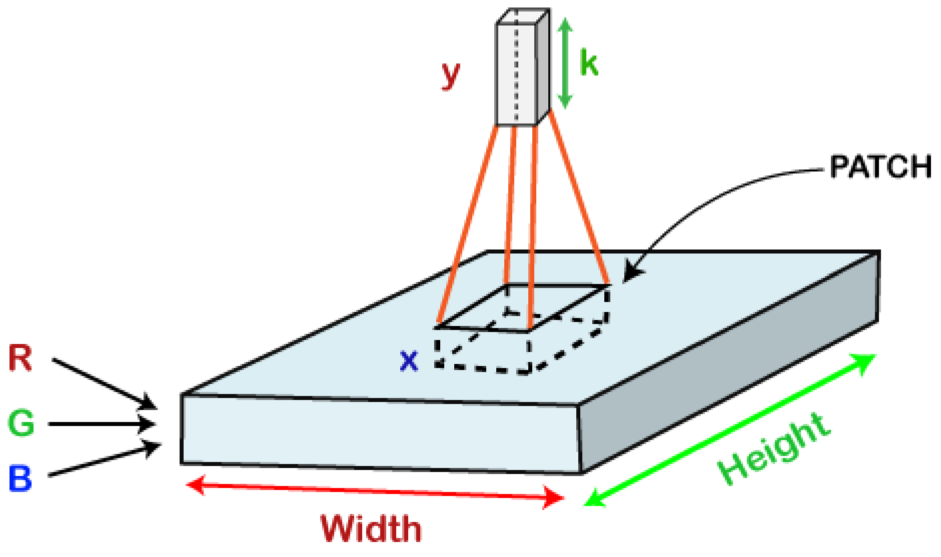
3.4 Convolutional Neural Network

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It's also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image.

The Convolutional Neural Networks, which are also called as covnets, are nothing but neural networks, sharing their parameters. Suppose that there is an image, which is embodied as a cuboid, such that it encompasses length, width, and height. Here the dimensions of the image are represented by the Red, Green, and Blue channels, as shown in the image given below.



Now assume that we have taken a small patch of the same image, followed by running a small neural network on it, having k number of outputs, which is represented in a vertical manner. Now when we slide our small neural network all over the image, it will result in another image constituting different width, height as well as depth. We will notice that rather than having R, G, B channels, we have come across some more channels that, too, with less width and height, which is actually the concept of Convolution. In case, if we accomplished in having similar patch size as that of the image, then it would have been a regular neural network. We have some wights due to this small patch.



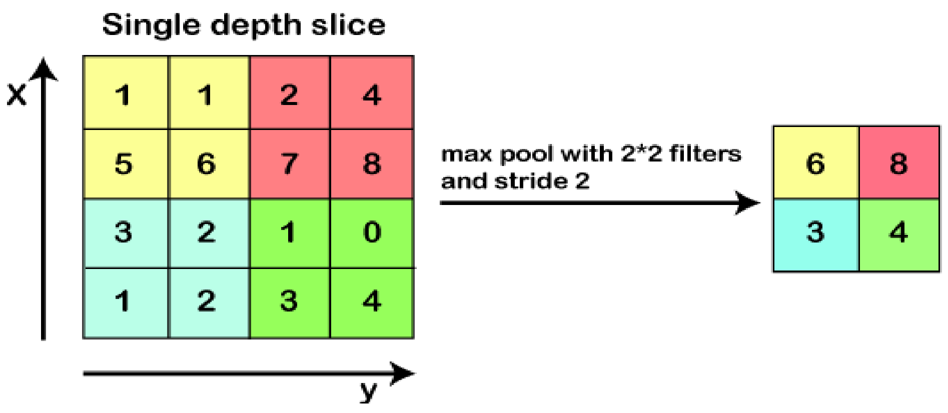
Mathematically it could be understood as follows;

- The Convolutional layers encompass a set of learnable filters, such that each filter embraces small width, height as well as depth as that of the provided input volume (if the image is the input layer then probably it would be 3).
- Suppose that we want to run the convolution over the image that comprises of 34x34x3 dimension, such that the size of a filter can be axax3. Here a can be any of the above 3, 5, 7, etc. It must be small in comparison to the dimension of the image.
- Each filter gets slide all over the input volume during the forward pass. It slides step by step, calling each individual step as a stride that encompasses a value of 2 or 3 or 4 for higher-dimensional images, followed by calculating a dot product in between filter's weights and patch from input volume.
- It will result in 2-Dimensional output for each filter as and when we slide our filters followed by stacking them together so as to achieve an output volume to have a similar depth value as that of the number of filters. And then, the network will learn all the filters.

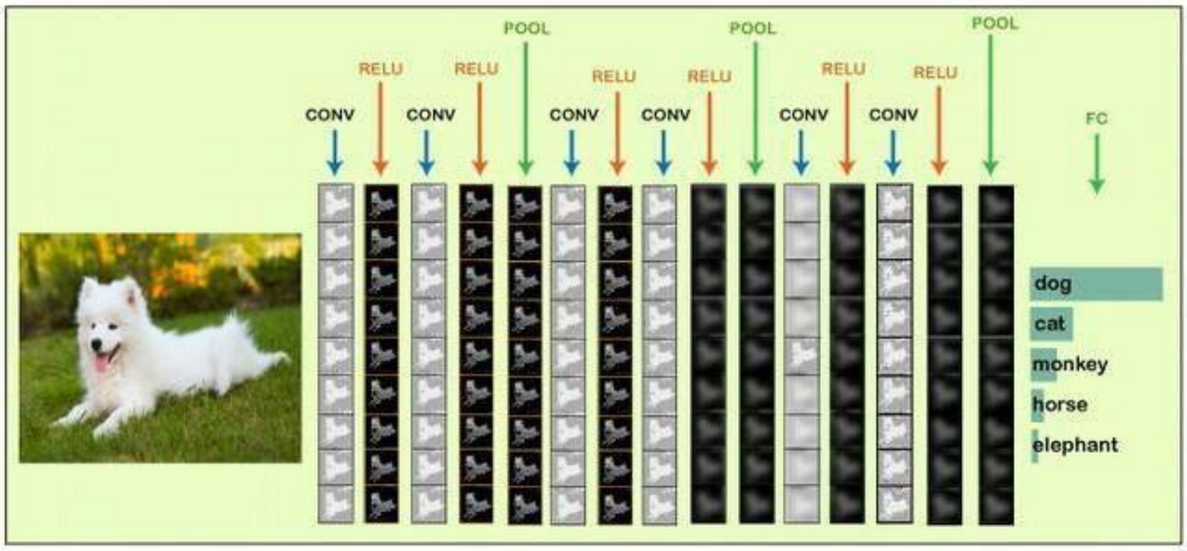
Working of CNN

Generally, a Convolutional Neural Network has three layers, which are as follows;

- **Input:** If the image consists of 32 widths, 32 height encompassing three R, G, B channels, then it will hold the raw pixel([32x32x3]) values of an image.
- **Convolution:** It computes the output of those neurons, which are associated with input's local regions, such that each neuron will calculate a dot product in between weights and a small region to which they are actually linked to in the input volume. For example, if we choose to incorporate 12 filters, then it will result in a volume of [32x32x12].
- **ReLU Layer:** It is specially used to apply an activation function elementwise, like as max (0, x) thresholding at zero. It results in ([32x32x12]), which relates to an unchanged size of the volume.
- **Pooling:** This layer is used to perform a downsampling operation along the spatial dimensions (width, height) that results in [16x16x12] volume.

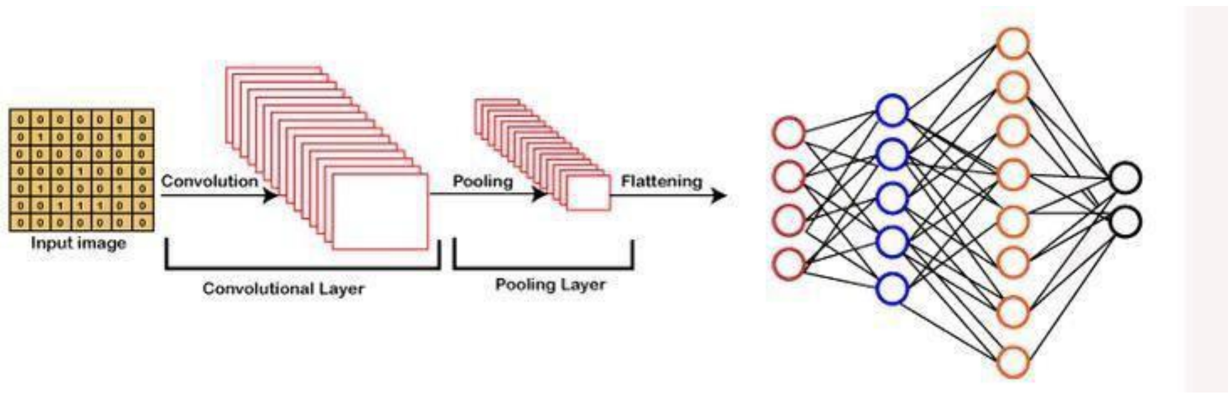


Locally Connected: It can be defined as a regular neural network layer that receives an input from the preceding layer followed by computing the class scores and results in a 1-Dimensional array that has the equal size to that of the number of classes.



We will start with an input image to which we will be applying multiple feature detectors, which are also called as filters to create the feature maps that comprises of a Convolution layer. Then on the top of that layer, we will be applying the ReLU or Rectified Linear Unit to remove any linearity or increase non-linearity in our images.

Next, we will apply a Pooling layer to our Convolutional layer, so that from every feature map we create a Pooled feature map as the main purpose of the pooling layer is to make sure that we have spatial invariance in our images. It also helps to reduce the size of our images as well as avoid any kind of overfitting of our data. After that, we will flatten all of our pooled images into one long vector or column of all of these values, followed by inputting these values into our artificial neural network. Lastly, we will feed it into the locally connected layer to achieve the final output.



Pooling Layers

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.
For a feature map having dimensions $\mathbf{n_h} \times \mathbf{n_w} \times \mathbf{n_c}$, the dimensions of output obtained after a pooling layer is

$(n_h - f + 1) / s \times (nw - f + 1)/s \times nc$

where,

-> $\mathbf{n_h}$ - height of feature map
-> $\mathbf{n_w}$ - width of feature map
-> $\mathbf{n_c}$ - number of channels in the feature map
-> **f** - size of filter
-> **s** - stride length

A common CNN model architecture is to have a number of convolution and pooling layers stacked one after the other.

Why to use Pooling Layers?

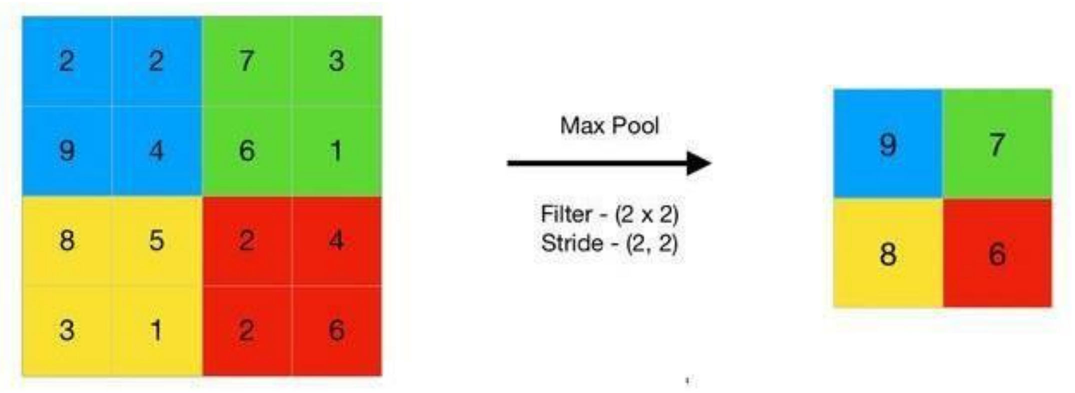
- Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Types of Pooling Layers:

Max Pooling

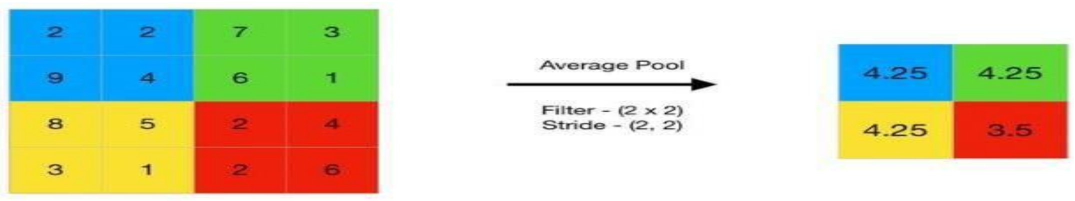
1. Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



This can be achieved using MaxPooling2D layer in keras as follows

Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



Global Pooling

1. Global pooling reduces each channel in the feature map to a single value. Thus, an $\mathbf{n_h \times n_w \times n_c}$ feature map is reduced to $\mathbf{1 \times 1 \times n_c}$ feature map. This is equivalent to using a filter of dimensions $\mathbf{n_h \times n_w}$ i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

In convolutional neural networks (CNNs), the pooling layer is a common type of layer that is typically added after convolutional layers. The pooling layer is used to reduce the spatial dimensions (i.e., the width and height) of the feature maps, while preserving the depth (i.e., the number of channels).

1. The pooling layer works by dividing the input feature map into a set of non-overlapping regions, called pooling regions. Each pooling region is then transformed into a single output value, which represents the presence of a particular feature in that region. The most common types of pooling operations are max pooling and average pooling.
2. In max pooling, the output value for each pooling region is simply the maximum value of the input values within that region. This has the effect of preserving the most salient features in each pooling region, while discarding less relevant information. Max pooling is often used in CNNs for object recognition tasks, as it helps to identify the most distinctive features of an object, such as its edges and corners.
3. In average pooling, the output value for each pooling region is the average of the input values within that region. This has the effect of preserving more information than max pooling, but may also dilute the most salient features. Average pooling is often used in CNNs for tasks such as image segmentation and object detection, where a more fine-grained representation of the input is required.

Pooling layers are typically used in conjunction with convolutional layers in a CNN, with each pooling layer reducing the spatial dimensions of the feature maps, while the convolutional layers extract increasingly complex features from the input. The resulting feature maps are then passed to a fully connected layer, which performs the final classification or regression task.

Advantages of Pooling Layer:

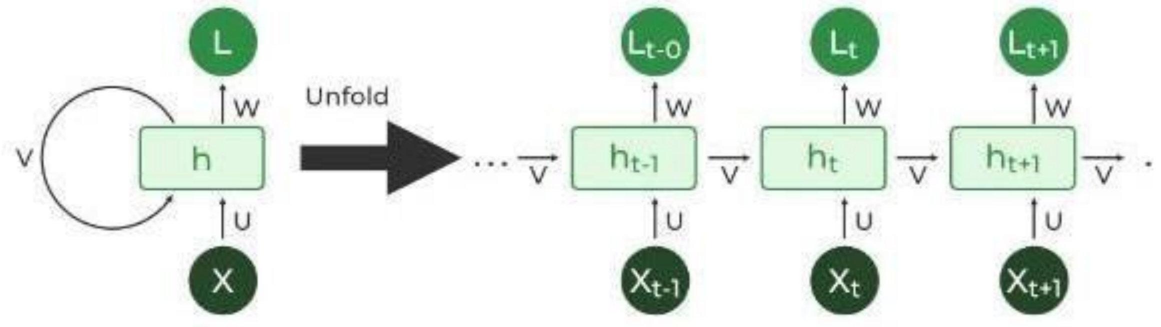
1. Dimensionality reduction: The main advantage of pooling layers is that they help in reducing the spatial dimensions of the feature maps. This reduces the computational cost and also helps in avoiding overfitting by reducing the number of parameters in the model.
2. Translation invariance: Pooling layers are also useful in achieving translation invariance in the feature maps. This means that the position of an object in the image does not affect the classification result, as the same features are detected regardless of the position of the object.
3. Feature selection: Pooling layers can also help in selecting the most important features from the input, as max pooling selects the most salient features and average pooling preserves more information.

Disadvantages of Pooling Layer:

1. Information loss: One of the main disadvantages of pooling layers is that they discard some information from the input feature maps, which can be important for the final classification or regression task.
2. Over-smoothing: Pooling layers can also cause over-smoothing of the feature maps, which can result in the loss of some fine-grained details that are important for the final classification or regression task.
3. Hyperparameter tuning: Pooling layers also introduce hyperparameters such as the size of the pooling regions and the stride, which need to be tuned in order to achieve optimal performance. This can be time-consuming and requires some expertise in model building.

3.5 Recurrent Neural Network(RNN)

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.



Architecture Of Recurrent Neural Network

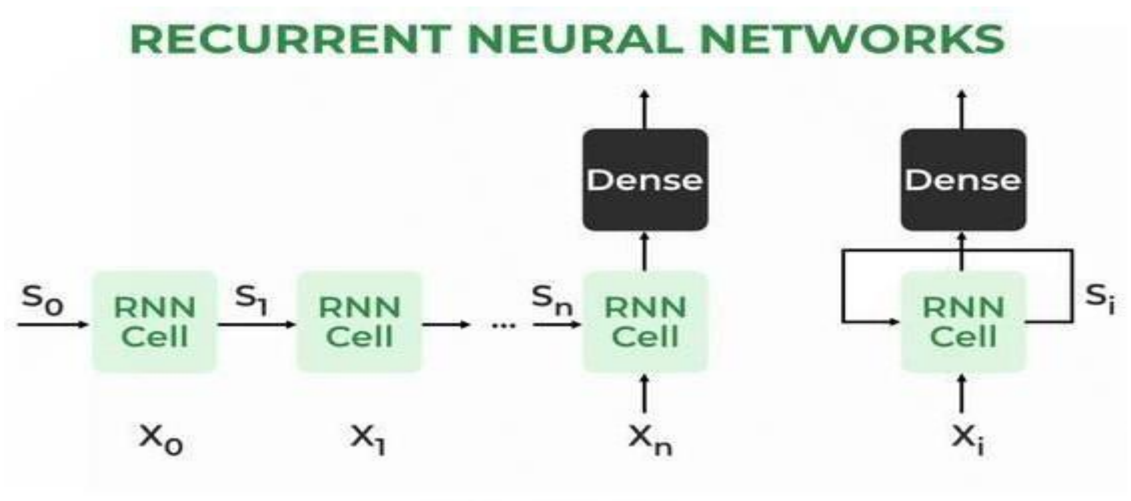
RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state H_i for every input X_i . **By using the following formulas:**

h= σ(UX + Wh$_{-1}$ + B)

Y = O(Vh + C) Hence

Y = f (X, h , W, U, V, B, C)

Here S is the State matrix which has element si as the state of the network at timestep i
The parameters in the network are W, U, V, c, b which are shared across timestep



What is Recurrent Neural Network

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where:

h_t -> current state
h_{t-1} -> previous state
x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

w_{hh} -> weight at recurrent neuron
w_{xh} -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output

W_{hy} -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using back propagation through time

3.6 Confusion Matrix in Machine Learning

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

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

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

UNIT-IV

Model Validation in Classification : Cross Validation - Holdout Method, K-Fold, Stratified K-Fold, Leave-One-Out Cross Validation. Bias-Variance tradeoff, Regularization , Overfitting, Underfitting. Ensemble Methods: Boosting, Bagging, Random Forest.

4.1 CROSS VALIDATION

To test the performance of a classifier, we need to have a number of training/validation set pairs from a dataset X. To get them, if the sample X is large enough, we can randomly divide it then divide each part randomly into two and use one half for training and the other half for validation. Unfortunately, datasets are never large enough to do this. So, we use the same data split differently; this is called cross-validation.

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

During the evaluation of machine learning (ML) models, the following question might arise:

- Is this model the best one available from the hypothesis space of the algorithm in terms of generalization error on an unknown/future data set?
- What training and testing techniques are used for the model?
- What model should be selected from the available ones?

4.2 Methods used for Cross-Validation:

4.2.1 Holdout Method

Consider training a model using an algorithm on a given dataset. Using the same training data, you determine that the trained model has an accuracy of 95% or even 100%. What does this mean? Can this model be used for prediction?

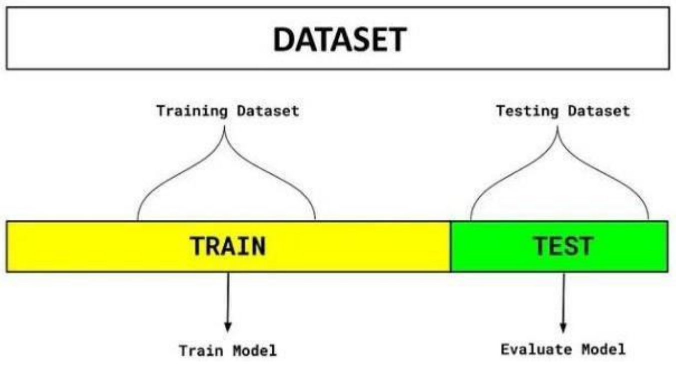
No. This is because your model has been trained on the given data, i.e. it knows the data and has generalized over it very well. In contrast, when you try to predict over a new set of data, it will most likely give you very bad accuracy because it has never seen the data before and thus cannot generalize well over it. To deal with such problems, hold-out methods can be employed.

The hold-out method involves splitting the data into multiple parts and using one part for training the model and the rest for validating and testing it. It can be used for both model evaluation and selection.

In cases where every piece of data is used for training the model, there remains the problem of selecting the best model from a list of possible models. Primarily, we want to identify which model has the lowest generalization error or which model makes a better prediction on future or unseen datasets than all of the others. There is a need to have a mechanism that allows the model to be trained on one set of data and tested on another set of data. This is where hold-out comes into play.

Hold-Out Method for Model Evaluation

Model evaluation using the hold-out method entails splitting the dataset into training and test datasets, evaluating model performance, and determining the most optimal model. This diagram illustrates the hold-out method for model evaluation.

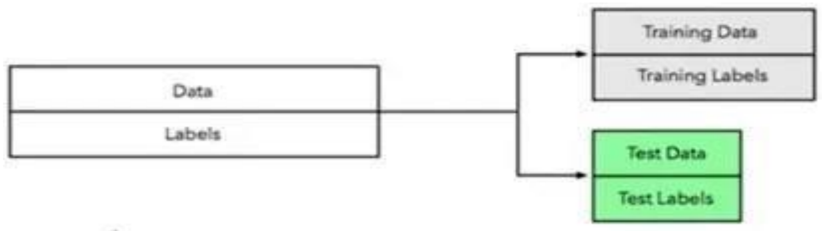


There are two parts to the dataset in the diagram above. One split is held aside as a training set. Another set is held back for testing or evaluation of the model. The percentage of the split is determined based on the amount of training data available. A typical split of 70–30% is used in which 70% of the dataset is used for training and 30% is used for testing the model.

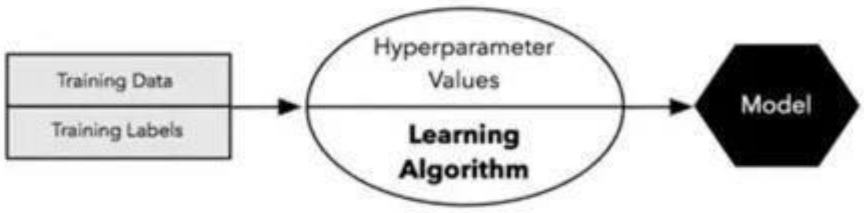
The objective of this technique is to select the best model based on its accuracy on the testing dataset and compare it with other models. There is, however, the possibility that the model can be well fitted to the test data using this technique. In other words, models are trained to improve model accuracy on test datasets based on the assumption that the test dataset represents the population. As a result, the test error becomes an optimistic estimation of the generalization error. Obviously, this is not what we want. Since the final model is trained to fit well (or overfit) the test data, it won't generalize well to unknowns or future datasets.

Follow the steps below for using the hold-out method for model evaluation:

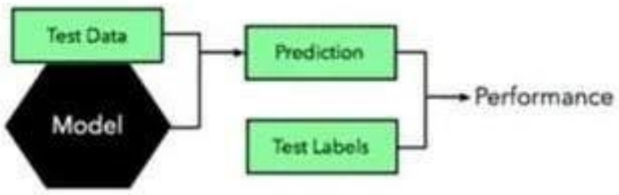
1. Split the dataset in two (preferably 70–30%; however, the split percentage can vary and should be random).



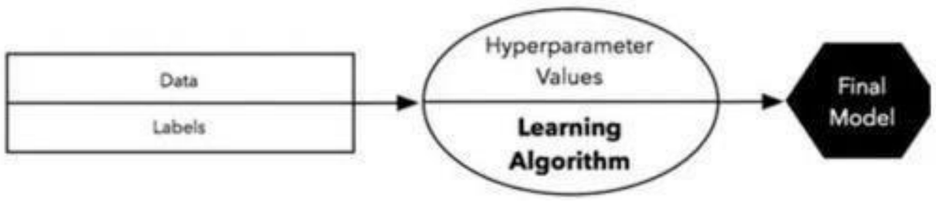
2. Now, we train the model on the training dataset by selecting some fixed set of hyperparameters while training the model.



3. Use the hold-out test dataset to evaluate the model.



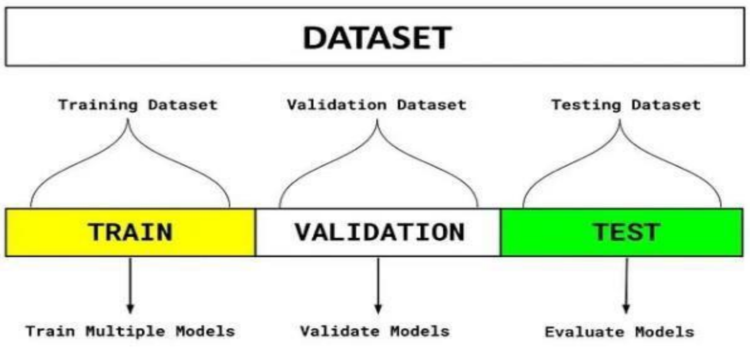
4. Use the entire dataset to train the final model so that it can generalize better on future datasets.



In this process, the dataset is split into training and test sets, and a fixed set of hyperparameters is used to evaluate the model. There is another process in which data can also be split into three sets, and these sets can be used to select a model or to tune hyperparameters. We will discuss that technique next.

Hold-Out Method for Model Selection

Sometimes the model selection process is referred to as hyperparameter tuning. During the hold-out method of selecting a model, the dataset is separated into three sets — training, validation, and test.



Follow the steps below for using the hold-out method for model selection:

1. Divide the dataset into three parts: training dataset, validation dataset, and test dataset.
2. Now, different machine learning algorithms can be used to train different models. You can train your classification model, for example, using logistic regression, random forest, and XGBoost.
3. Tune the hyperparameters for models trained with different algorithms. Change the hyperparameter settings for each algorithm mentioned in step 2 and come up with multiple models.
4. On the validation dataset, test the performance of each of these models (associating with each of the algorithms).
5. Choose the most optimal model from those tested on the validation dataset. The most optimal model will be set up with the most optimal hyperparameters. Using the example above, let's suppose the model trained with XGBoost with the most optimal hyperparameters is selected.
6. Finally, on the test dataset, test the performance of the most optimal model.

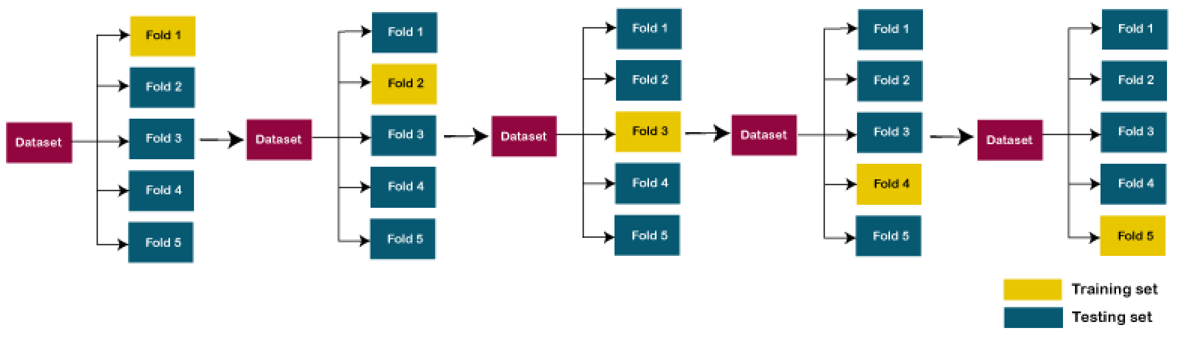
4.2.2 K-Fold Cross-Validation

K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called **folds**. For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set. This approach is a very popular CV approach because it is easy to understand, and the output is less biased than other methods.

The steps for k-fold cross-validation are:

- Split the input dataset into K groups
- For each group:
  - Take one group as the reserve or test data set.
  - Use remaining groups as the training dataset
  - Fit the model on the training set and evaluate the performance of the model using the test set.

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1st iteration, the first fold is reserved for test the model , and rest are used to train the model. On 2nd iteration, the second fold is used to test the model, and rest are used to train the model. This process will continue until each fold is not used for the test fold.



4.2.3 Stratified k-fold cross-validation:

This technique is similar to k-fold cross-validation with some little changes. This approach works on stratification concept, it is a process of rearranging the data to ensure that each fold or group is a good representative of the complete dataset. To deal with the bias and variance, it is one of the best approaches.

It can be understood with an example of housing prices, such that the price of some houses can be much high than other houses. To tackle such situations, a stratified k-fold cross-validation technique is useful.

4.2.4 Leave one out cross-validation

This method is similar to the leave-p-out cross-validation, but instead of p, we need to take 1 dataset out of training. It means, in this approach, for each learning set, only one data point is reserved, and the remaining dataset is used to train the model. This process repeats for each data point. Hence for n samples, we get n different training set and n test set. It has the following features:

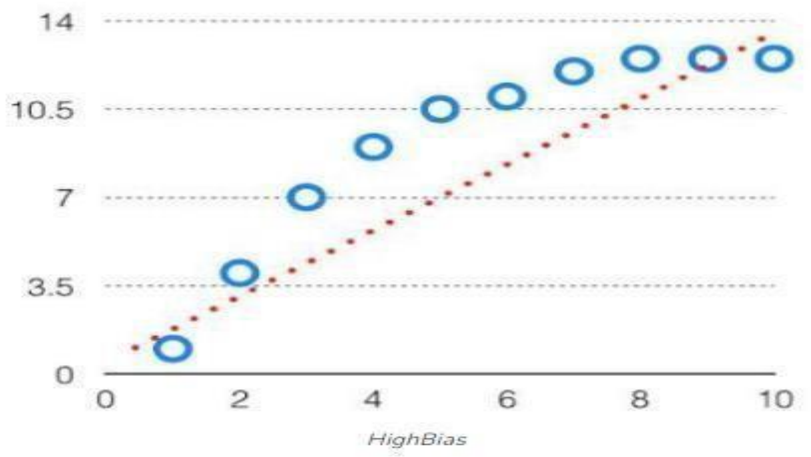
- In this approach, the bias is minimum as all the data points are used.
- The process is executed for n times; hence execution time is high.
- This approach leads to high variation in testing the effectiveness of the model as we iteratively check against one data point.

4.3 Bias-Variance Trade off

It is important to understand prediction errors (bias and variance) when it comes to accuracy in any machine learning algorithm. There is a tradeoff between a model's ability to minimize bias and variance which is referred to as the best solution for selecting a value of **Regularization** constant. Proper understanding of these errors would help to avoid the overfitting and underfitting of a data set while training the algorithm

Bias

The bias is known as the difference between the prediction of the values by the ML model and the correct value. Being high in biasing gives a large error in training as well as testing data. Its recommended that an algorithm should always be low biased to avoid the problem of underfitting.By high bias, the data predicted is in a straight line format, thus not fitting accurately in the data in the data set. Such fitting is known as **Underfitting of Data**. This happens when the hypothesis is too simple or linear in nature. Refer to the graph given below for an example of such a situation.



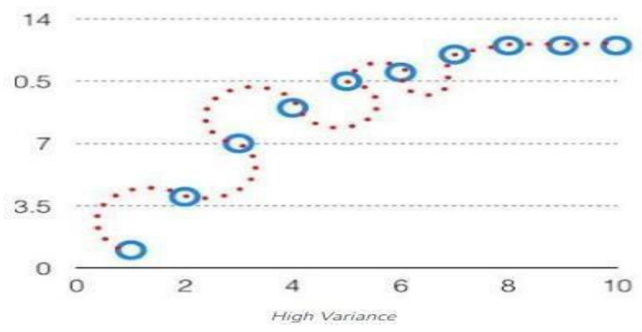
In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Variance

The variability of model prediction for a given data point which tells us spread of our data is called the variance of the model. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.When a model is high on variance, it is then said to as **Overfitting of Data**. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high. While training a data model variance should be kept low.

The high variance data looks like follows.



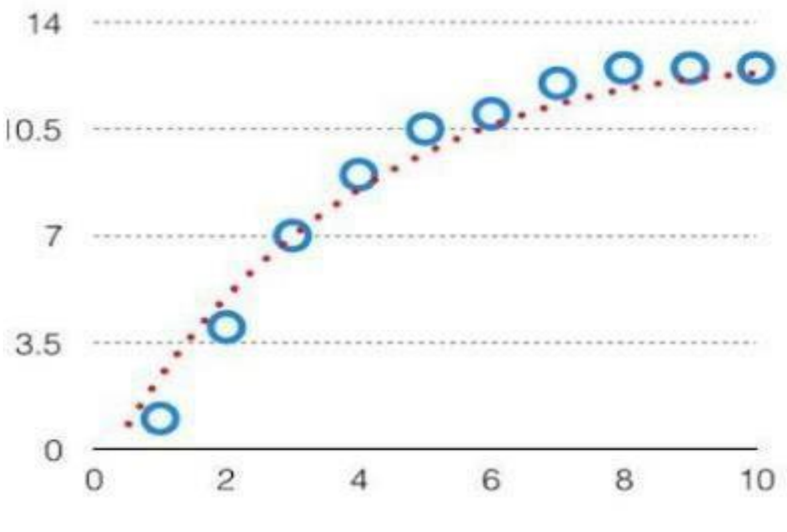
In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Bias Variance Tradeoff

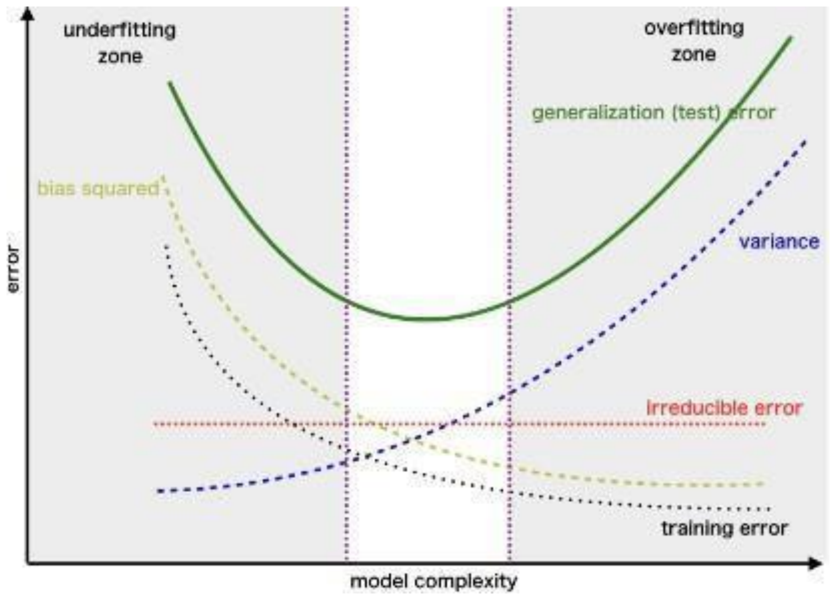
If the algorithm is too simple (hypothesis with linear eq.) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree eq.) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as Trade-off or Bias Variance Trade-off.

This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time. For the graph, the perfect tradeoff will be like.



The best fit will be given by hypothesis on the tradeoff point.

The error to complexity graph to show trade-off is given as –



This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

4.4 Regularization :

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

it mainly regularizes or reduces the coefficient of features toward zero. In simple words, " *In regularization technique, we reduce the magnitude of the features by keeping the same number of features."*

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

y= β0+β1x1+β2x2+β3x3+⋯+βnxn +b

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

$$\sum_{i=1}^{M}\left(y_i - y'_i\right)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n} \beta_j * Xij\right)^2$$

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

- **Ridge Regression**
- **Lasso Regression**

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can

calculate it by multiplying with the lambda to the squared weight of each individual feature.

- The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}\beta_j^2$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- It helps to solve the problems if we have more parameters than samples.

Lasso Regression:

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

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}|\beta_j|$$

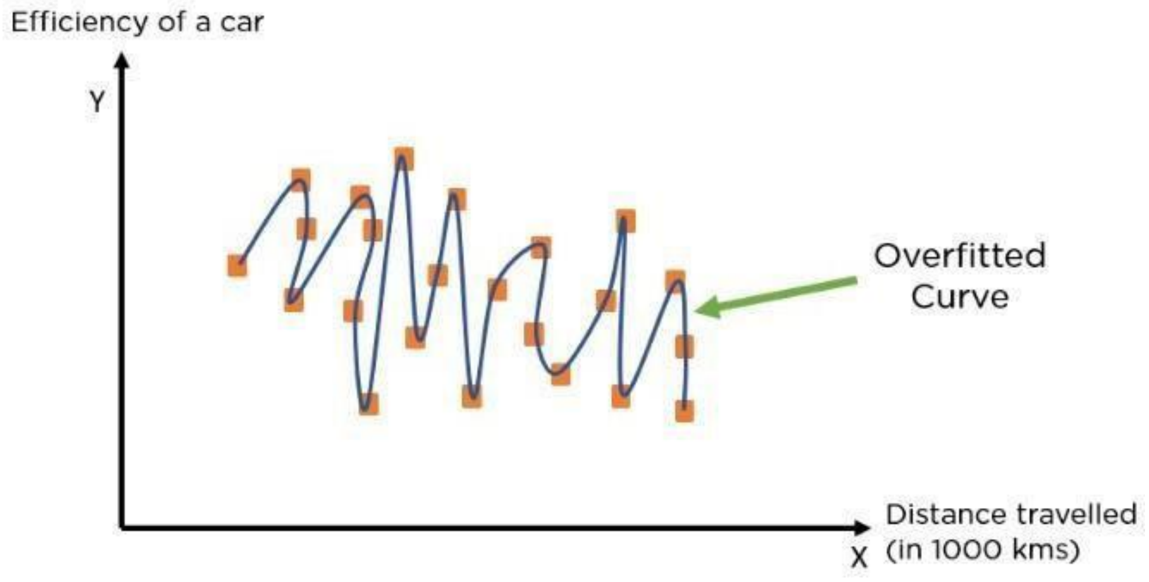
- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

4.5 Overfitting and Under fitting:

To train our machine learning model, we give it some data to learn from. The process of plotting a series of data points and drawing the best fit line to understand the relationship between the variables is called Data Fitting. Our model is the best fit when it can find all necessary patterns in our data and avoid the random data points and unnecessary patterns called Noise.

4.5.1 Overfitting

When a model performs very well for training data but has poor performance with test data (new data), it is known as overfitting. In this case, the machine learning model learns the details and noise in the training data such that it negatively affects the performance of the model on test data. Overfitting can happen due to low bias and high variance.



Reasons for Overfitting

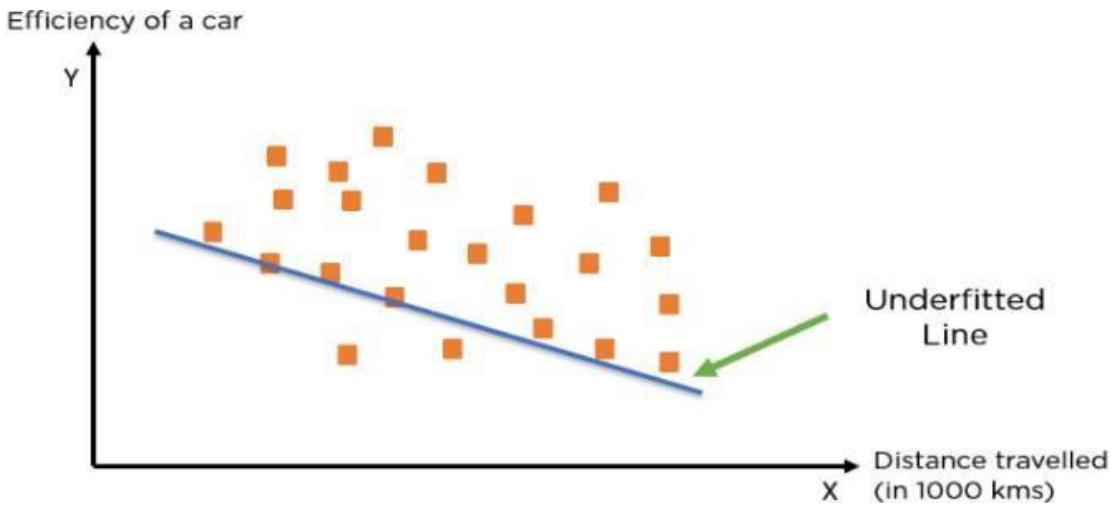
- Data used for training is not cleaned and contains noise (garbage values) in it
- The model has a high variance
- The size of the training dataset used is not enough
- The model is too complex

Ways to Tackle Overfitting

- Using K-fold cross-validation
- Using Regularization techniques such as Lasso and Ridge
- Training model with sufficient data
- Adopting ensembling techniques

4.5.2 Underfitting:

When a model has not learned the patterns in the training data well and is unable to generalize well on the new data, it is known as underfitting. An underfit model has poor performance on the training data and will result in unreliable predictions. Underfitting occurs due to high bias and low variance.



Reasons for Underfitting

- Data used for training is not cleaned and contains noise (garbage values) in it
- The model has a high bias
- The size of the training dataset used is not enough

- The model is too simple

Ways to Tackle Underfitting

- Increase the number of features in the dataset
- Increase model complexity
- Reduce noise in the data
- Increase the duration of training the data

4.6 Ensemble Methods:

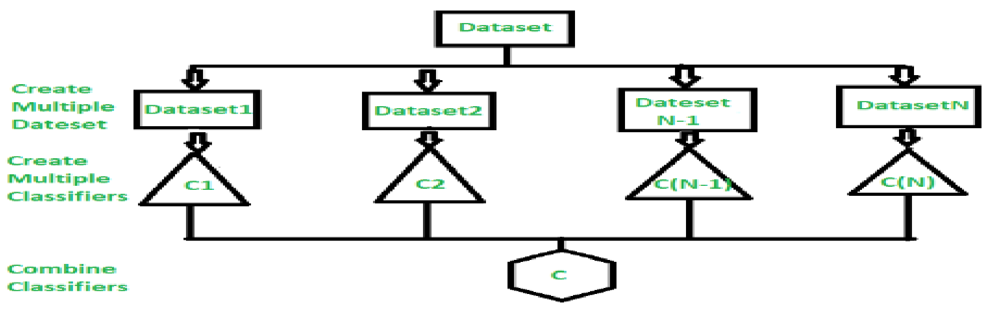
When you want to purchase a new car, will you walk up to the first car shop and purchase one based on the advice of the dealer? It's highly unlikely.

You would likely browser a few web portals where people have posted their reviews and compare different car models, checking for their features and prices. You will also probably ask your friends and colleagues for their opinion. In short, you wouldn't directly reach a conclusion, but will instead make a decision considering the opinions of other people as well.

Ensemble models in machine learning operate on a similar idea. They combine the decisions from multiple models to improve the overall performance.

Advantage : Improvement in predictive accuracy.

Disadvantage : It is difficult to understand an ensemble of classifiers.



Ensembles overcome three problems –

- **Statistical Problem** –

The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!

- **Computational Problem –**
  The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis.
- **Representational Problem –**
  The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es).

Types of Ensemble Classifier –

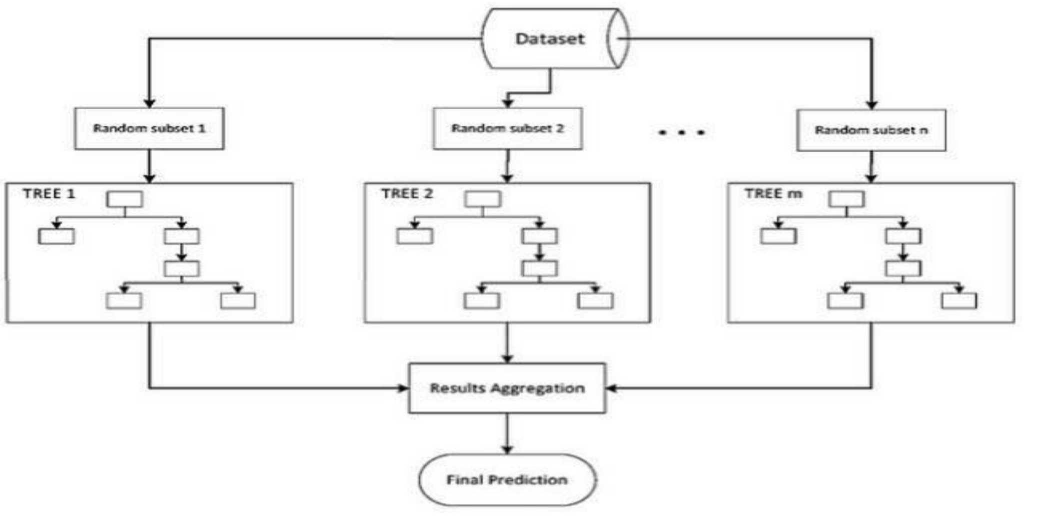
1)Bagging

2)Boosting

3)Random Forest

4.6.1 Bagging:

B*AGG*ing, or ***B***ootstrap ***AGG***regating. **BAGG**ing gets its name because it combines ***B***ootstrapping and ***Agg***regation to form one ensemble model. Given a sample of data, multiple bootstrapped subsamples are pulled. A Decision Tree is formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. The image below will help explain:

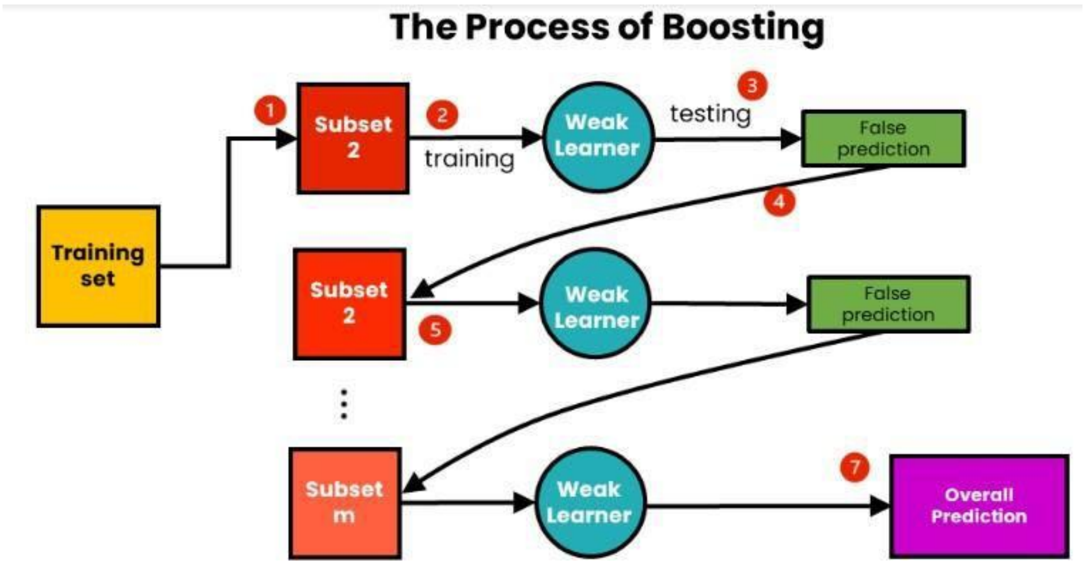


Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

4.6.2 Boosting :

Unlike bagging, which aggregates prediction results at the end, boosting aggregates the results at each step. They are aggregated using weighted averaging.

Weighted averaging involves giving all models different weights depending on their predictive power. In other words, it gives more weight to the model with the highest predictive power. This is because the learner with the highest predictive power is considered the most important.



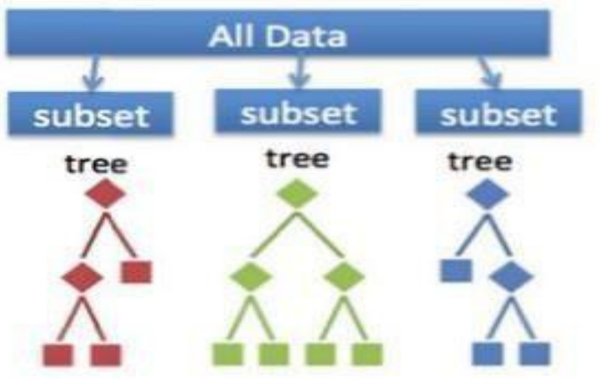
Boosting works with the following steps:

1. We sample m-number of subsets from an initial training dataset.
2. Using the first subset, we train the first weak learner.

3. We test the trained weak learner using the training data. As a result of the testing, some data points will be incorrectly predicted.
4. Each data point with the wrong prediction is sent into the second subset of data, and this subset is updated.
5. Using this updated subset, we train and test the second weak learner.
6. We continue with the following subset until the total number of subsets is reached.
7. We now have the total prediction. The overall prediction has already been aggregated at each step, so there is no need to calculate it.

4.6.3 Random Forest Models.

Random Forest Models can be thought of as **BAGG**ing, with a slight tweak. When deciding where to split and how to make decisions, BAGGed Decision Trees have the full disposal of features to choose from. Therefore, although the bootstrapped samples may be slightly different, the data is largely going to break off at the same features throughout each model. In contrary, Random Forest models decide where to split based on a random selection of features. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor. Refer to the image for a better understanding.



UNIT-V

Unsupervised Learning : Clustering-K-means, K-Modes, K-Prototypes, Gaussian MixtureModels, Expectation-Maximization.

Reinforcement Learning: Exploration and exploitation trade-offs, non-associative learning,Markov decision processes, Q-learning.

5.1 Unsupervised Learning:

Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

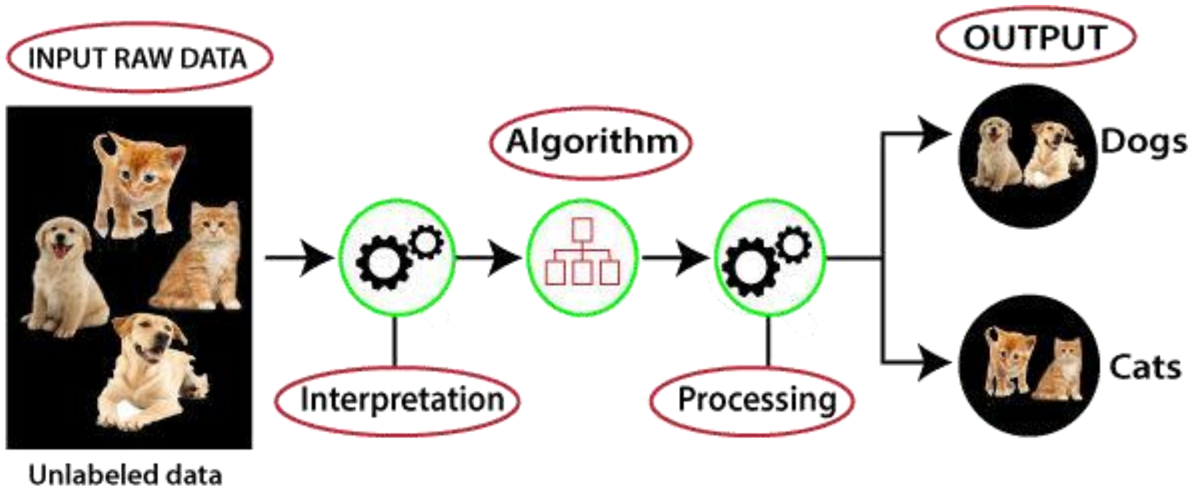
Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of unsupervised learning can be understood by the below diagram:

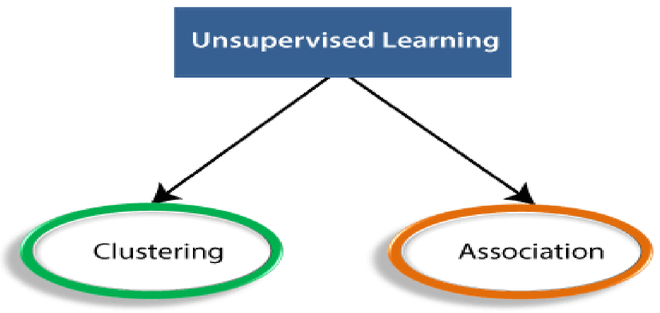


-

Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

The unsupervised learning algorithm can be further categorized into two types of problems:

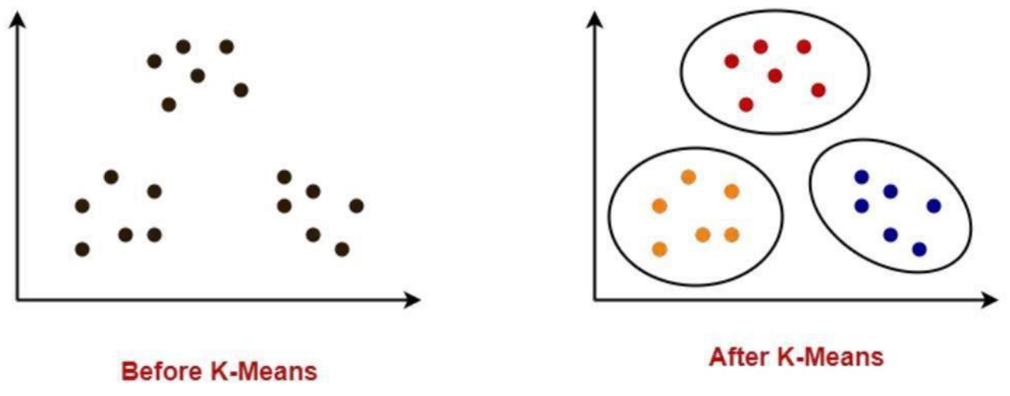


- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis

5.1 K-Means Clustering

- K-Means clustering is an unsupervised iterative clustering technique.
- It partitions the given data set into k predefined distinct clusters.

- A cluster is defined as a collection of data points exhibiting certain similarities.



- It partitions the data set such that-
- Each data point belongs to a cluster with the nearest mean.
- Data points belonging to one cluster have high degree of similarity.
- Data points belonging to different clusters have high degree of dissimilarity.

K-Means Clustering Algorithm-

- K-Means Clustering Algorithm involves the following steps-

Step-01:

1. Choose the number of clusters K.

Step-02:

1. Randomly select any K data points as cluster centers.
2. Select cluster centers in such a way that they are as farther as possible from each other.

Step-03:

1. Calculate the distance between each data point and each cluster center.
2. The distance may be calculated either by using given distance function or by using euclidean distance formula.

Step-04:

1. Assign each data point to some cluster.
2. A data point is assigned to that cluster whose center is nearest to that data point.

Step-05:

1. Re-compute the center of newly formed clusters.

2. The center of a cluster is computed by taking mean of all the data points contained in that cluster.

Step-06:

Keep repeating the procedure from Step-03 to Step-05 until any of the following stopping criteria is met-

1. Center of newly formed clusters do not change
2. Data points remain present in the same cluster
3. Maximum number of iterations are reached

Problem:

Cluster the following eight points (with (x, y) representing locations) into three clusters:

A1(2, 10), A2(2, 5), A3(8, 4), A4(5, 8), A5(7, 5), A6(6, 4), A7(1, 2), A8(4, 9)

Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2).

The distance function between two points a = (x1, y1) and b = (x2, y2) is defined as-

P(a, b) = |x2 - x1| + |y2 - y1|

Use K-Means Algorithm to find the three cluster centers after the second iteration.

Solution-

We follow the above discussed K-Means Clustering Algorithm-

Iteration-01:

- We calculate the distance of each point from each of the center of the three clusters.
- The distance is calculated by using the given distance function.

The following illustration shows the calculation of distance between point A1(2, 10) and each of the center of the three clusters-

Calculating Distance Between A1(2, 10) and C1(2, 10)-

P(A1, C1)

= |x2 – x1| + |y2 – y1|

= |2 – 2| + |10 – 10|

= 0

Calculating Distance Between A1(2, 10) and C2(5, 8)-

P(A1, C2)

= |x2 – x1| + |y2 – y1|

= |5 – 2| + |8 – 10|

= 3 + 2

= 5

Calculating Distance Between A1(2, 10) and C3(1, 2)-

P(A1, C3)

= |x2 – x1| + |y2 – y1|

= |1 – 2| + |2 – 10|

= 1 + 8

= 9

In the similar manner, we calculate the distance of other points from each of the center of the three clusters.

Next,

- We draw a table showing all the results.
- Using the table, we decide which point belongs to which cluster.
- The given point belongs to that cluster whose center is nearest to it.

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (5, 8) of Cluster-02	Distance from center (1, 2) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	5	9	C1
A2(2, 5)	5	6	4	C3
A3(8, 4)	12	7	9	C2
A4(5, 8)	5	0	10	C2
A5(7, 5)	10	5	9	C2
A6(6, 4)	10	5	7	C2
A7(1, 2)	9	10	0	C3
A8(4, 9)	3	2	10	C2

From here, New clusters are-

Cluster-01:

First cluster contains points-

- A1(2, 10)

Cluster-02:

Second cluster contains points-

- A3(8, 4)
- A4(5, 8)
- A5(7, 5)
- A6(6, 4)
- A8(4, 9)

Cluster-03:

Third cluster contains points-

- A2(2, 5)
- A7(1, 2)

Now,

- We re-compute the new cluster clusters.
- The new cluster center is computed by taking mean of all the points contained in that cluster.

For Cluster-01:

- We have only one point A1(2, 10) in Cluster-01.
- So, cluster center remains the same.

For Cluster-02:

Center of Cluster-02

= ((8 + 5 + 7 + 6 + 4)/5, (4 + 8 + 5 + 4 + 9)/5)

= (6, 6)

For Cluster-03:

Center of Cluster-03

= ((2 + 1)/2, (5 + 2)/2)

= (1.5, 3.5)

This is completion of Iteration-01.

Iteration-02:

- We calculate the distance of each point from each of the center of the three clusters.
- The distance is calculated by using the given distance function.

The following illustration shows the calculation of distance between point A1(2, 10) and each of the center of the three clusters-

Calculating Distance Between A1(2, 10) and C1(2, 10)-

P(A1, C1)

= |x2 – x1| + |y2 – y1|

= |2 – 2| + |10 – 10|

= 0

Calculating Distance Between A1(2, 10) and C2(6, 6)-

P(A1, C2)

= |x2 – x1| + |y2 – y1|

= |6 – 2| + |6 – 10|

= 4 + 4

= 8

Calculating Distance Between A1(2, 10) and C3(1.5, 3.5)-

P(A1, C3)

= |x2 – x1| + |y2 – y1|

= |1.5 – 2| + |3.5 – 10|

= 0.5 + 6.5

= 7

In the similar manner, we calculate the distance of other points from each of the center of the three clusters.

Next,

- We draw a table showing all the results.
- Using the table, we decide which point belongs to which cluster.
- The given point belongs to that cluster whose center is nearest to it.

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (6, 6) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	8	7	C1
A2(2, 5)	5	5	2	C3
A3(8, 4)	12	4	7	C2
A4(5, 8)	5	3	8	C2
A5(7, 5)	10	2	7	C2
A6(6, 4)	10	2	5	C2
A7(1, 2)	9	9	2	C3
A8(4, 9)	3	5	8	C1

From here, New clusters are-

Cluster-01:

First cluster contains points-

- A1(2, 10)
- A8(4, 9)

Cluster-02:

Second cluster contains points-

- A3(8, 4)
- A4(5, 8)
- A5(7, 5)
- A6(6, 4)

Cluster-03:

Third cluster contains points-

- A2(2, 5)
- A7(1, 2)

Now,

- We re-compute the new cluster clusters.
- The new cluster center is computed by taking mean of all the points contained in that cluster.

For Cluster-01:

Center of Cluster-01

= ((2 + 4)/2, (10 + 9)/2)

= (3, 9.5)

For Cluster-02:

Center of Cluster-02

= ((8 + 5 + 7 + 6)/4, (4 + 8 + 5 + 4)/4)

= (6.5, 5.25)

For Cluster-03:

Center of Cluster-03

= ((2 + 1)/2, (5 + 2)/2)

= (1.5, 3.5)

This is completion of Iteration-02.

After second iteration, the center of the three clusters are-

- C1(3, 9.5)
- C2(6.5, 5.25)
- C3(1.5, 3.5)

5.2 K MODES

KModes is a clustering algorithm used to group similar data points into clusters based on their categorical attributes. Unlike traditional clustering algorithms that use distance metrics, KModes works by identifying the modes or most frequent values within each cluster to determine its centroid. KModes is ideal for clustering categorical data such as customer demographics, market segments, or survey responses

In K-means clustering when we used categorical data after converting it into a numerical form. it doesn’t give a good result for high-dimensional data. So, Some changes are made for categorical data t.

- Replace Euclidean distance with Dissimilarity metric
- Replace Mean by Mode for cluster centers.
- Apply a frequency-based method in each iteration to update the mode.

And then this is called K-MODE Clustering because of MODE.

Similarity and dissimilarity measurements are used to determine the distance between the data objects in the dataset. In the case of K-modes, these distances are calculated using a dissimilarity measure called the Hamming distance. The Hamming distance between two data objects is the number of categorical attributes that differ between the two objects.

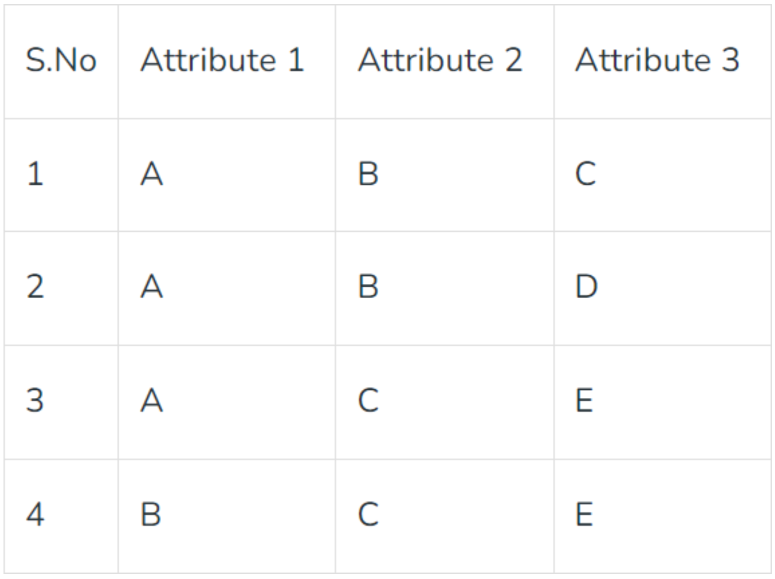
Let x and y be two categorical data objects defined by m features or attributes.

$$d(x, y) = \sum_{j=1}^{m} \delta(x_j, y_j)$$

Where,

$$\delta(x_j, y_j) = \begin{cases} 0 & \text{if } x_j = y_j \\ 1 & \text{if } x_j \neq y_j \end{cases}$$

For example, consider the following dataset with three categorical attributes:



S.No	Attribute 1	Attribute 2	Attribute 3
1	A	B	C
2	A	B	D
3	A	C	E
4	B	C	E

To calculate the Hamming distance between data objects 1 and 2, we compare their values for each attribute and count the number of differences. In this case, there is one difference (Attribute 3 is C for object 1 and D for object 2), so the Hamming distance between objects 1 and 2 is 1.

To calculate the Hamming distance between objects 1 and 3, we again compare their values for each attribute and count the number of differences. In this case, there are two differences (Attribute 2 is B for object 1 and C for object 3, and Attribute 3 is C for object 1 and E for object 3), so the Hamming distance between objects 1 and 3 is 2.

To calculate the Hamming distance between objects 1 and 4, we again compare their values for each attribute and count the number of differences. In this case, there are three differences (Attribute 1 is A for objects 1 and B for object 4, Attribute 2 is B for object 1 and C for object 4, and Attribute 3 is C for objects 1 and E for object 4), so the Hamming distance between objects 1 and 4 is 3.

Data objects with a smaller Hamming distance are considered more similar, while objects with a larger Hamming distance is considered more dissimilar.

Let X be a set of categorical data objects of $X = \begin{bmatrix} x_{11}, & \dots & x_{1n} \\ \dots & \dots & \dots \\ x_{n1}, & \dots & x_{nm} \end{bmatrix}$ that can be denoted as $X = [X_1, X_2, \dots, X_n]$. And the mode of Z is a vector $Q = [q_1, q_2, \dots, q_m]$ then, minimize

$$D(X, Q) = \sum_{i=1}^{n} d(X_i, Q)$$

Apply dissimilarity metric equation for data objects

$$D(X, Q) = \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(x_{ij}, Q)$$

Suppose we want to K cluster, Then we have Q = [q_{k1},q_{k1},....,q_{km}] \epsilon Q

$$C(Q) = \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(x_{ij}, q_{kj})$$

The main task for K-Modes algorithm is to minimize this C(Q) cost function.

It consists of the following steps.

1. Select K data objects for each cluster.
2. Calculate dissimilarities D(X,Q) and allocate each data object to nearest cluster.
3. Calculate the new modes for all clusters.
4. Repeat step 2 and 3 until the cluster will become stable.

Some variations of the K-modes algorithm may use different methods for updating the centroids (modes) of the clusters, such as taking the weighted mode or the median of the objects within each cluster.

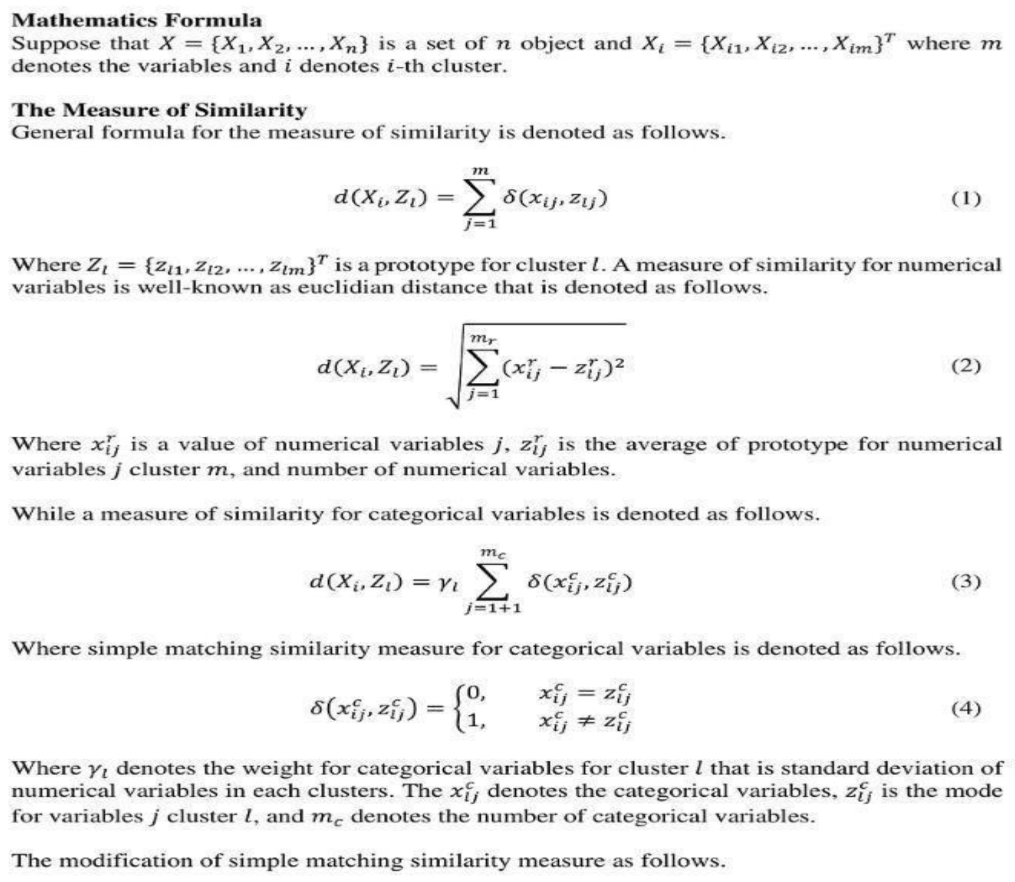
Overall, the goal of K-modes clustering is to minimize the dissimilarities between the data objects and the centroids (modes) of the clusters, using a measure of categorical similarity such as the Hamming distance.

5.3 K PROTOTYPE:

One of the conventional clustering methods commonly used in clustering techniques and efficiently used for large data is the K-Means algorithm. However, its method is not good and suitable for data that contains categorical variables. This problem happens when the cost function in K-Means is calculated using the Euclidian distance that is only suitable for numerical data. While K-Mode is only suitable for categorical data only, not mixed data types.

Facing these problems, Huang proposed an algorithm called K-Prototype which is created in order to handle clustering algorithms with the mixed data types (numerical and categorical variables). K-Prototype is a clustering method based on partitioning. Its algorithm is an improvement of the K-Means and K-Mode clustering algorithm to handle clustering with the mixed data types**.**

K-Prototype has an advantage because it's not too complex and is able to handle large data and is better than hierarchical based algorithms



Mathematics Formula

Suppose that $X = \{X_1, X_2, \dots, X_n\}$ is a set of n object and $X_i = \{X_{i1}, X_{i2}, \dots, X_{im}\}^T$ where m denotes the variables and i denotes i-th cluster.

The Measure of Similarity

General formula for the measure of similarity is denoted as follows.

$$d(X_i, Z_l) = \sum_{j=1}^{m} \delta(x_{ij}, z_{lj}) \qquad (1)$$

Where $Z_l = \{z_{l1}, z_{l2}, \dots, z_{lm}\}^T$ is a prototype for cluster l. A measure of similarity for numerical variables is well-known as euclidian distance that is denoted as follows.

$$d(X_i, Z_l) = \sqrt{\sum_{j=1}^{m_r} (x_{ij}^r - z_{lj}^r)^2} \qquad (2)$$

Where x_{ij}^r is a value of numerical variables j, z_{lj}^r is the average of prototype for numerical variables j cluster m, and number of numerical variables.

While a measure of similarity for categorical variables is denoted as follows.

$$d(X_i, Z_l) = \gamma_l \sum_{j=1+1}^{m_c} \delta(x_{ij}^c, z_{lj}^c) \qquad (3)$$

Where simple matching similarity measure for categorical variables is denoted as follows.

$$\delta(x_{ij}^c, z_{lj}^c) = \begin{cases} 0, & x_{ij}^c = z_{lj}^c \\ 1, & x_{ij}^c \neq z_{lj}^c \end{cases} \qquad (4)$$

Where γ_l denotes the weight for categorical variables for cluster l that is standard deviation of numerical variables in each clusters. The x_{ij}^c denotes the categorical variables, z_{lj}^c is the mode for variables j cluster l, and m_c denotes the number of categorical variables.

The modification of simple matching similarity measure as follows.

The modification of simple matching similarity measure as follows.

$$\delta(x_{ij}^c, z_{lj}^c) = \begin{cases} 1 - \omega(x_{ij}^c, l), & x_{ij}^c = z_{lj}^c \\ 1 \quad , & x_{ij}^c \neq z_{lj}^c \end{cases} \tag{5}$$

The above formula increases the object similarity within cluster with categorical variables so that the result will be better where $\omega(x_{ij}^c, l)$ denotes the weight for x_{ij}^c where

$$\omega(x_{ij}^c, l) = \frac{f(x_{ij}^c | c_l)}{|c_l| \cdot f(x_{ij}^c | D)} \tag{6}$$

Where $f(x_{ij}^c | c_l)$ is the frequency of x_{ij}^c in cluster l and $|c_l|$ is the number of object in cluster l, and $f(x_{ij}^c | c_l)$ is the frequency of x_{ij}^c in the whole of data.

According to the equation (1) to (5), it obtains the measure of similarity prior to the data with numerical and categorical variables as follows.

$$d(X_i, Z_l) = \sqrt{\sum_{j=1}^{m_r} (x_{ij}^r - z_{lj}^r)^2 + \gamma_l \sum_{j=1+1}^{m_c} \delta(x_{ij}^c, z_{lj}^c)} \tag{7}$$

Huang Cost Function
Huang declared that cost function equation for mixed data type (numerical and categorical) is as follows.

$$Cost_l = \sum_{l=1}^{k} u_{il} \sum_{j=1}^{m_r} (x_{ij}^r - z_{lj}^r)^2 + \gamma_l \sum_{j=1}^{m_c} u_{il} \sum_{j=1}^{m_c} \delta(x_{ij}^c, z_{lj}^c) \tag{8}$$
$$Cost_l = Cost_l^r + Cost_l^c$$

Where $Cost_l^r$ denotes the total cost of all the numerical variables for the entire objects within cluster l. $Cost_l^r$ is minimized while z_{lj} being calculated with following equation.

$$z_{lj} = \frac{1}{n_l} \sum_{i=1}^{n} u_{il} \cdot x_{ij} \quad \text{for } j = 1, 2, \ldots, m \tag{9}$$

Where $n_l = \sum_{i=1}^{n} u_{il} \cdot x_{ij}$ is the number of objects within cluster l.

Further, the categorical variables e.g. C_j is a set of unique value in each categorical variables j and $p(q_{ij}^c \in C_j | l)$ is the probability for c_j within cluster l. So, $Cost_l^c$ can be rewritten as follows.

$$Cost_l^c = \gamma_l \sum_{j=1}^{m_c} n_l \left(1 - p(q_{ij}^c \in C_j | l)\right) \tag{10}$$

where n_l denotes the objects within cluster l. The solution in order to minimize the $Cost_l^c$ is explained clearly in **lemma 1**.

Lemma 1
For special cluster l, $Cost_l^c$ is minimized if and only if $p(z_{lj}^c \in C_j | l) \geq p(c_j \in C_j | l)$ for $z_{lj}^c \neq c_j$ to all categorical variables. So that *cost function* can be rewritten as follows.

$$Cost = \sum_{l=1}^{K} (Cost_l^r + Cost_l^c)$$
$$Cost = \sum_{l=1}^{k} Cost_l^r + \sum_{l=1}^{k} Cost_l^c \tag{11}$$
$$Cost = Cost^r + Cost^c$$

Because $Cost^r$ and $Cost^c$ are non-negative, $Cost$ minimalization can be done by minimizing the $Cost^r$ and $Cost^c$.

5.4 Normal or Gaussian Distribution

In real life, many datasets can be modeled by Gaussian Distribution (Univariate or Multivariate). So it is quite natural and intuitive to assume that the clusters come from different Gaussian Distributions. Or in other words, it tried to model the dataset as a mixture of several Gaussian Distributions. This is the core idea of this model.
In one dimension the probability density function of a Gaussian Distribution is given by

$$G(X|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

where μ and σ^2 are respectively the mean and variance of the distribution. For Multivariate (let us say d-variate) Gaussian Distribution, the probability density function is given by

$$G(X|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)|\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(X-\mu)^T \mathbf{\Sigma}^{-1}(X-\mu)\right)$$

Here μ is a d dimensional vector denoting the mean of the distribution and Σ is the d X d covariance matrix.

Gaussian Mixture Model

Suppose there are K clusters (For the sake of simplicity here it is assumed that the number of clusters is known and it is K). So and are also estimated for each k. Had it been only one distribution, they would have been estimated by the **maximum-likelihood method**. But since there are K such clusters and the probability density is defined as a linear function of densities of all these K distributions, i.e.

$$p(X) = \sum_{k=1}^{K} \pi_k G(X|\mu_k, \Sigma_k)$$

where π_k is the mixing coefficient for kth distribution. For estimat
parameters by the maximum log-likelihood method, compute p(

$$\begin{aligned} \ln p(X|\mu, \Sigma, \pi) &= \sum_{i=1}^{N} p(X_i) \\ &= \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k G(X_i|\mu_k, \Sigma_k) \end{aligned}$$

Now define a random variable $\gamma_k(X)$ such that $\gamma_k(X)$=p(k|X).

From Bayes theorem,

$$\begin{aligned}\gamma_k(X) &= \frac{p(X|k)p(k)}{\sum_{k=1}^{K} p(k)p(X|k)} \\ &= \frac{p(X|k)\pi_k}{\sum_{k=1}^{K} \pi_k p(X|k)}\end{aligned}$$

Now for the log-likelihood function to be maximum, its derivative of $p(X|\mu,\Sigma,\pi)$ with respect to μ, Σ, and π should be zero. So equating the derivative of $p(X|\mu,\Sigma,\pi)$ with respect to μ to zero and rearranging the terms,

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n)x_n}{\sum_{n=1}^{N} \gamma_k(x_n)}$$

Similarly taking the derivative with respect to Σ and pi respectively, one can obtain the following expressions.

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n)(x_n-\mu_k)(x_n-\mu_k)^T}{\sum n=1^N \gamma_k(x_n)}$$

And

$$\pi_k = \frac{1}{N}\sum_{n=1}^{N} \gamma_k(x_n)$$

Note: *$\sum_{n=1}^{N} \gamma_k(x_n)$ denotes the total number of sample points in the kth cluster. Here it is assumed that there is a total N number of samples and each sample containing d features is denoted by x_i.*

So it can be clearly seen that the parameters cannot be estimated in closed form. This is where the **Expectation-Maximization algorithm** is beneficial.

5.5 Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is an iterative way to find maximum-likelihood estimates for model parameters when the data is incomplete or has some missing data points or has some hidden variables. EM chooses some random values for the missing data points and estimates a new set of data. These new values are then recursively used to estimate a better first date, by filling up missing points, until the values get fixed.

These are the two basic steps of the EM algorithm, namely the E Step, or Expectation Step or Estimation Step, and M Step, or Maximization Step.

Estimation step

Initialize μ_k, Σ_k and π_k by some random values, or by K means clustering results or by hierarchical clustering results. Then for those given parameter values, estimate the value of the latent variables

(i.e γ_k)

Maximization Step

Update the value of the parameters(i.e. μ_k, $\Sigma_k$andπ_k) calculated using the ML method.

5.6 Reinforcement Learning

Unlike supervised and unsupervised learning, reinforcement learning is a **feedback-based approach** in which agent learns by performing some actions as well as their outcomes. Based on action status (good or bad), the agent gets positive or negative feedback. Further, for each positive feedback, they get rewarded, whereas, for each negative feedback, they also get penalized.

Def: "Reinforcement learning is a type of machine learning technique, where an intelligent agent (computer program) interacts with the environment, explore it by itself, and makes actions within that."

- Reinforcement learning does not require any labeled data for the learning process. It learns through the feedback of action performed by the agent. Moreover, in reinforcement learning, agents also learn from past experiences.

- Reinforcement learning methods are used to solve tasks where decision-making is sequential and the goal is long-term, e.g., robotics, online chess, etc.
- Reinforcement learning aims to get maximum positive feedback so that they can improve their performance.
- Reinforcement learning involves various actions, which include taking action, changing/unchanged state, and getting feedback. And based on these actions, agents learn and explore the environment.

5.6.1 Exploration and Exploitation in Reinforcement Learning:

Before going to a brief description of exploration and exploitation in machine learning, let's first understand these terms in simple words. In reinforcement learning, whenever agents get a situation in which they have to make a difficult choice between whether to continue the same work or explore something new at a specific time, then, this situation results in Exploration-Exploitation Dilemma because the knowledge of an agent about the state, actions, rewards and resulting states is always partial.

Now we will discuss exploitation and exploration in technical terms.

Exploitation in Reinforcement Learning

Exploitation is defined as a greedy approach in which agents try to get more rewards by using estimated value but not the actual value. So, in this technique, ***agents make the best decision based on current information.***

Exploration in Reinforcement Learning

Unlike exploitation, in exploration techniques, agents primarily focus on improving their knowledge about each action instead of getting more rewards so that they can get long-term benefits. So, in this technique, ***agents work on gathering more information to make the best overall decision.***

Examples of Exploitation and Exploration in Machine Learning

Let's understand exploitation and exploration with some interesting real-world examples.

Coal mining:

Let's suppose people A and B are digging in a coal mine in the hope of getting a diamond inside it. Person B got success in finding the diamond before person A and walks off happily. After seeing him, person A gets a bit greedy and thinks he too might get success in finding diamond at the same place where person B was digging coal. This action performed by person A is called **greedy action**, and this policy is known as **a greedy policy.** But person A was unknown because a bigger diamond was buried in that place where he was initially digging the coal, and this greedy policy would fail in this situation.

In this example, person A only got knowledge of the place where person B was digging but had no knowledge of what lies beyond that depth. But in the actual scenario, the diamond can also be buried in the same place where he was digging initially or some completely another place. Hence, with this partial knowledge about getting more rewards, our reinforcement learning agent will be in a dilemma on whether to exploit the partial knowledge to receive some rewards or it should explore unknown actions which could result in many rewards.

However, both these techniques are not feasible simultaneously, but this issue can be resolved by using Epsilon Greedy Policy (Explained below).

here are a few other examples of Exploitation and Exploration in Machine Learning as follows:

Example 1: Let's say we have a scenario of online restaurant selection for food orders, where you have two options to select the restaurant. In the first option, you can choose your favorite restaurant from where you ordered food in the past; this is called **exploitation** because here, you only know information about a specific restaurant. And for other options, you can try a new restaurant to explore new varieties and tastes of food, and it is called exploration. However, food quality might be better in the first option, but it is also possible that it is more delicious in another restaurant.

Example 2: Suppose there is a game-playing platform where you can play chess with robots. To win this game, you have two choices either play the move that you believe is best, and for the other choice, you can play an experimental move. However, you are playing the best possible move, but who knows new move might be more strategic to win this game. Here, the first choice is called exploitation, where you know about your game strategy, and the second choice is called exploration, where you are exploring your knowledge and playing a new move to win the game.

5.6.2 Non-Associative Learning

In reinforcement learning, non-associative learning refers to a type of learning that does not involve forming associations or relationships between different stimuli or actions. It is a simpler form of learning compared to associative learning, which involves linking different stimuli or actions together.

Non-associative learning is typically observed in situations where an agent's behavior changes in response to a single stimulus or repeated exposure to the same stimulus. There are two common types of non-associative learning: habituation and sensitization.

1. Habituation: Habituation occurs when an agent's response to a particular stimulus decreases over time with repeated exposure. It is a form of adaptive behavior where the agent learns to ignore irrelevant or harmless stimuli. For example, if a robot is repeatedly exposed to a loud noise that is not associated with any reward or punishment, it may gradually stop reacting to the noise and become habituated to it.

2. Sensitization: Sensitization is the opposite of habituation and occurs when an agent's response to a stimulus increases over time with repeated exposure. It involves an increased sensitivity or responsiveness to a stimulus. For example, if a robot is repeatedly exposed to a painful stimulus, it may become more sensitive to that stimulus and show an increased response.

Non-associative learning is not directly related to reinforcement learning, as reinforcement learning primarily focuses on associative learning, where agents learn to associate their actions with rewards or punishments. However, non-associative learning mechanisms can play a role in shaping an agent's behavior and influencing its responses to stimuli, which can indirectly impact the learning process in reinforcement learning scenarios.

Examples

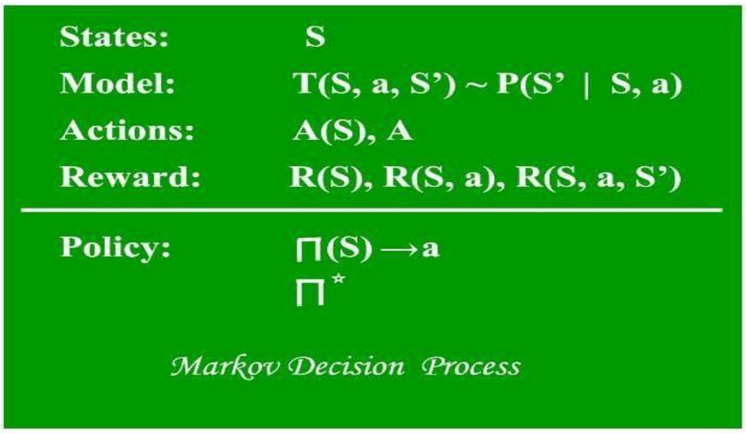
- Living near an airport for a year and getting used to the sound of airplanes passing overhead ─ **Habituation**
- Hearing loud thunder when at home alone at night and then becoming easily startled by bright flashes of light ─ **Sensitization**

5.6.3 Markov-Decision Process

Reinforcement Learning is a type of Machine Learning. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed to decide the best action to select based on his current state. When this step is repeated, the problem is known as a **Markov Decision Process**.
A **Markov Decision Process (MDP)** model contains:

A set of possible world states S.
- A set of Models.
- A set of possible actions A.
- A real-valued reward function R(s,a).
- A policy the solution of **Markov Decision Process**.



States:	**S**
Model:	$T(S, a, S') \sim P(S' \mid S, a)$
Actions:	$A(S), A$
Reward:	$R(S), R(S, a), R(S, a, S')$
Policy:	$\Pi(S) \rightarrow a$ Π^*

Markov Decision Process

State:

A **State** is a set of tokens that represent every state that the agent can be in.

Model:

A **Model** (sometimes called Transition Model) gives an action's effect in a state. In particular, T(S, a, S') defines a transition T where being in state S and taking an action 'a' takes us to state S' (S and S' may be the same). For stochastic actions (noisy, non-deterministic) we also define a probability P(S'|S,a) which represents the probability of reaching a state S' if action 'a' is taken in state S. Note Markov property states that the effects of an action taken in a state depend only on that state and not on the prior history.

Actions

An **Action** A is a set of all possible actions. A(s) defines the set of actions that can be taken being in state S.

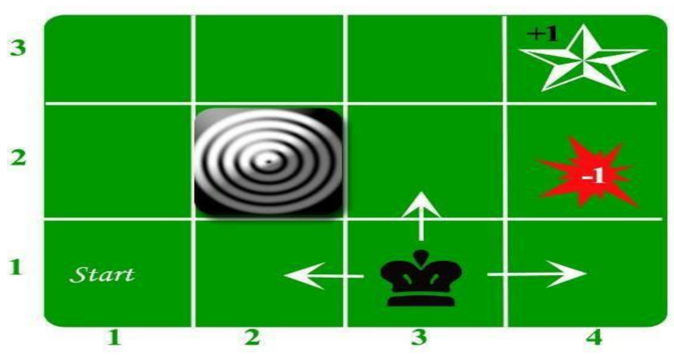
Reward

A **Reward** is a real-valued reward function. R(s) indicates the reward for simply being in the state S. R(S,a) indicates the reward for being in a state S and taking an action 'a'. R(S,a,S') indicates the reward for being in a state S, taking an action 'a' and ending up in a state S'.

Policy

A **Policy** is a solution to the Markov Decision Process. A policy is a mapping from S to a. It indicates the action 'a' to be taken while in state S.

Let us take the example of a grid world:



An agent lives in the grid. The above example is a 3*4 grid. The grid has a START state(grid no 1,1). The purpose of the agent is to wander around the grid to finally reach the Blue Diamond (grid no 4,3). Under all circumstances, the agent should avoid the Fire grid (orange color, grid no 4,2). Also the grid no 2,2 is a blocked grid, it acts as a wall hence the agent cannot enter it.

The agent can take any one of these actions: **UP, DOWN, LEFT, RIGHT**
Walls block the agent path, i.e., if there is a wall in the direction the agent would have taken, the agent stays in the same place. So for example, if the agent says LEFT in the START grid he would stay put in the START grid.

First Aim: To find the shortest sequence getting from START to the Diamond. Two such sequences can be found:

- **RIGHT RIGHT UP UPRIGHT**
- **UP UP RIGHT RIGHT RIGHT**

Let us take the second one (UP UP RIGHT RIGHT RIGHT) for the subsequent discussion. The move is now noisy. 80% of the time the intended action works correctly. 20% of the time the action agent takes causes it to move at right angles. For example, if the agent says UP the probability of going UP is 0.8 whereas the probability of going LEFT is 0.1, and the probability of going RIGHT is 0.1 (since LEFT and RIGHT are right angles to UP).

The agent receives rewards each time step:-

- Small reward each step (can be negative when can also be term as punishment, in the above example entering the Fire can have a reward of -1).
- Big rewards come at the end (good or bad).
- The goal is to Maximize the sum of rewards.

5.6.4 Q-learning

Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The "Q" stands for quality. Quality represents how valuable the action is in maximizing future rewards.

The **model-based** algorithms use transition and reward functions to estimate the optimal policy and create the model. In contrast, **model-free** algorithms learn the consequences of their actions through the experience without transition and reward function.

The **value-based** method trains the value function to learn which state is more valuable and take action. On the other hand, **policy-based** methods train the policy directly to learn which action to take in a given state.

In the **off-policy**, the algorithm evaluates and updates a policy that differs from the policy used to take an action. Conversely, the **on-policy** algorithm evaluates and improves the same policy used to take an action

Before we jump into how Q-learning works, we need to learn a few useful terminologies to understand Q-learning's fundamentals.

- **States(s)**: the current position of the agent in the environment.
- **Action(a)**: a step taken by the agent in a particular state.
- **Rewards**: for every action, the agent receives a reward and penalty.
- **Episodes**: the end of the stage, where agents can't take new action. It happens when the agent has achieved the goal or failed.
- **Q(S_{t+1}, a)**: expected optimal Q-value of doing the action in a particular state.
- **Q(S_t, A_t)**: it is the current estimation of Q(S_{t+1}, a).
- **Q-Table**: the agent maintains the Q-table of sets of states and actions.
- **Temporal Differences(TD)**: used to estimate the expected value of Q(S_{t+1}, a) by using the current state and action and previous state and action.

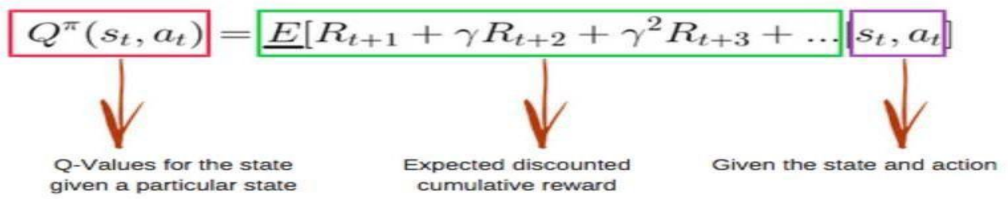
We will learn in detail how Q-learning works by using the example of a frozen lake. In this environment, the agent must cross the frozen lake from the start to the goal, without falling into the holes. The best strategy is to reach goals by taking the shortest path

Q-Table

The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. In simple words, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

Q-Function

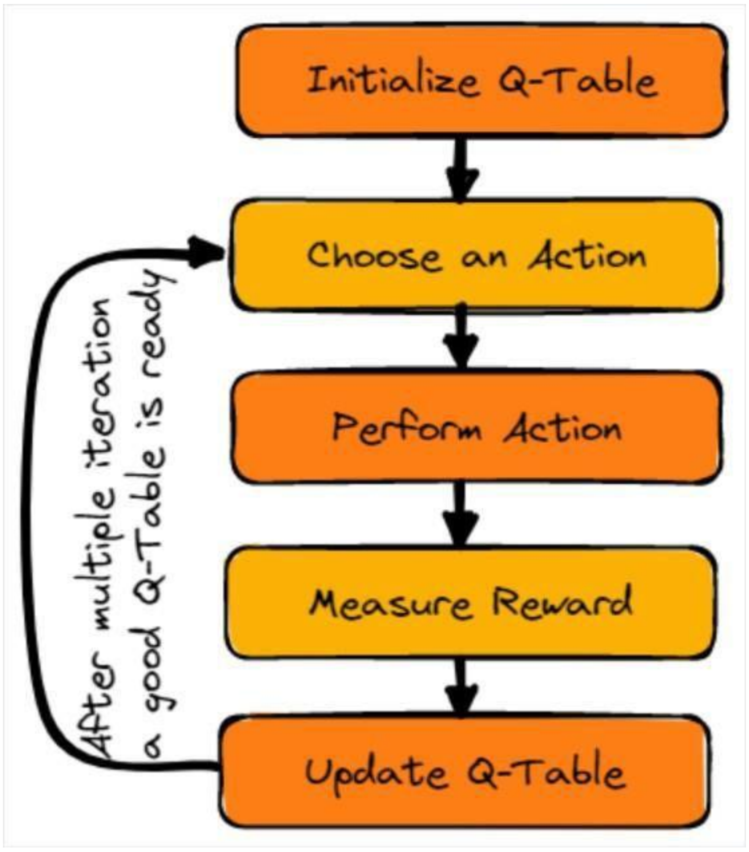
The Q-function uses the Bellman equation and takes state(s) and action(a) as input. The equation simplifies the state values and state-action value calculation.



$$\boxed{Q^{\pi}(s_t, a_t)} = \boxed{\underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots} \boxed{s_t, a_t}]$$

Q-Values for the state given a particular state
Expected discounted cumulative reward
Given the state and action

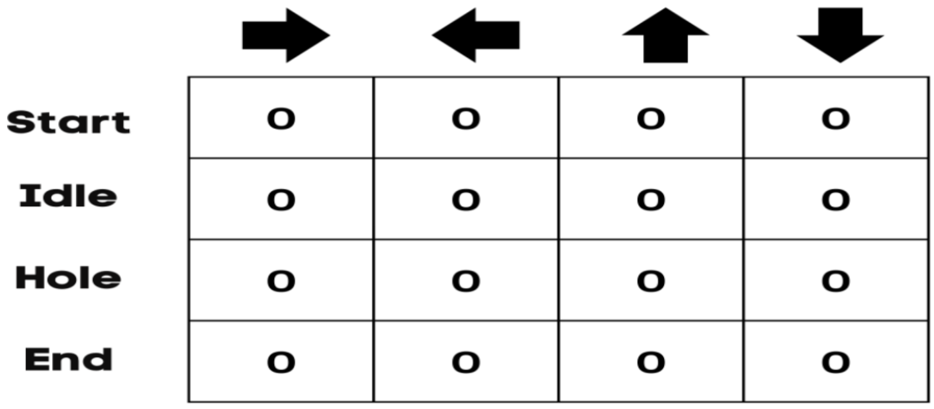
Q-learning algorithm



Initialize Q-Table

We will first initialize the Q-table. We will build the table with columns based on the number of actions and rows based on the number of states.

In our example, the character can move up, down, left, and right. We have four possible actions and four states(start, Idle, wrong path, and end). You can also consider the wrong path for falling into the hole. We will initialize the Q-Table with values at 0.



	→	←	↑	↓
Start	0	0	0	0
Idle	0	0	0	0
Hole	0	0	0	0
End	0	0	0	0

Choose an Action

The second step is quite simple. At the start, the agent will choose to take the random action(down or right), and on the second run, it will use an updated Q-Table to select the action.

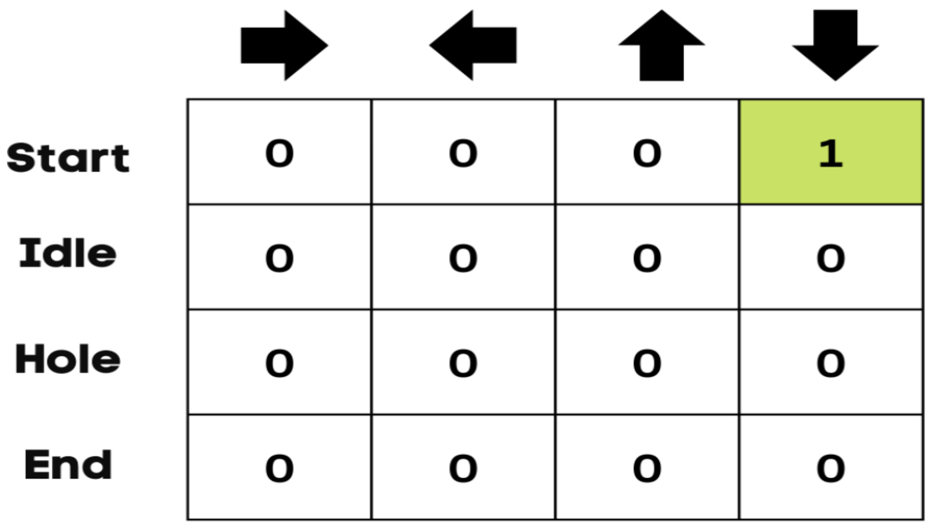
Perform an Action

Choosing an action and performing the action will repeat multiple times until the training loop stops. The first action and state are selected using the Q-Table. In our case, all values of the Q-Table are zero.

Then, the agent will move down and update the Q-Table using the Bellman equation. With every move, we will be updating values in the Q-Table and also using it for determining the best course of action.

Initially, the agent is in exploration mode and chooses a random action to explore the environment. The Epsilon Greedy Strategy is a simple method to balance exploration and exploitation. The epsilon stands for the probability of choosing to explore and exploits when there are smaller chances of exploring.

At the start, the epsilon rate is higher, meaning the agent is in exploration mode. While exploring the environment, the epsilon decreases, and agents start to exploit the environment. During exploration, with every iteration, the agent becomes more confident in estimating Q-values



	→	←	↑	↓
Start	0	0	0	1
Idle	0	0	0	0
Hole	0	0	0	0
End	0	0	0	0

In the frozen lake example, the agent is unaware of the environment, so it takes random action (move down) to start. As we can see in the above image, the Q-Table is updated using the Bellman equation.

Measuring the Rewards

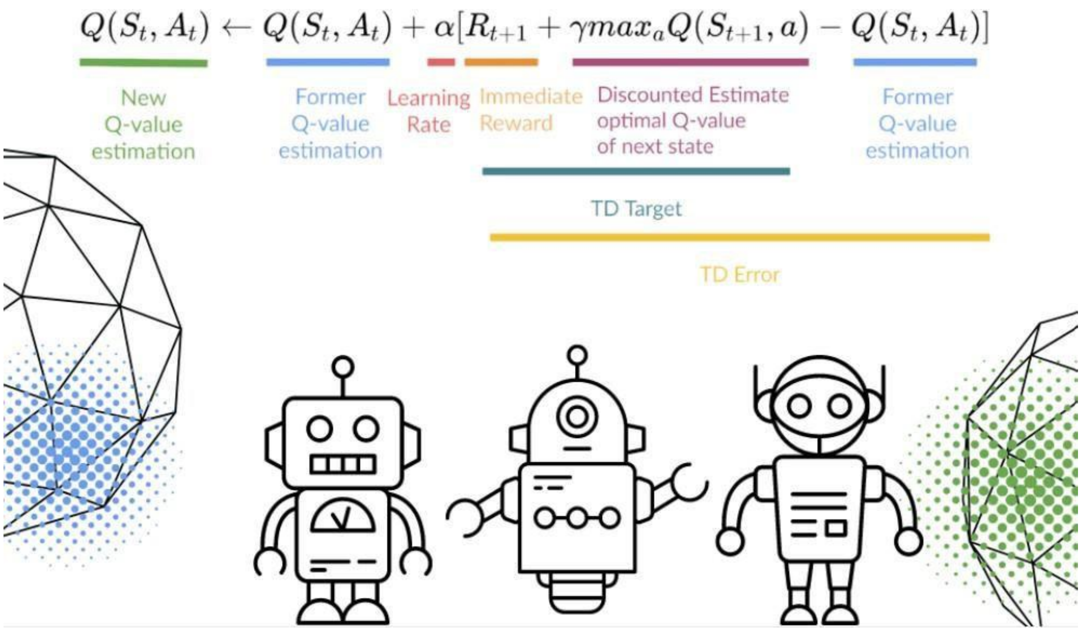
After taking the action, we will measure the outcome and the reward.

- The reward for reaching the goal is +1
- The reward for taking the wrong path (falling into the hole) is 0
- The reward for Idle or moving on the frozen lake is also 0.

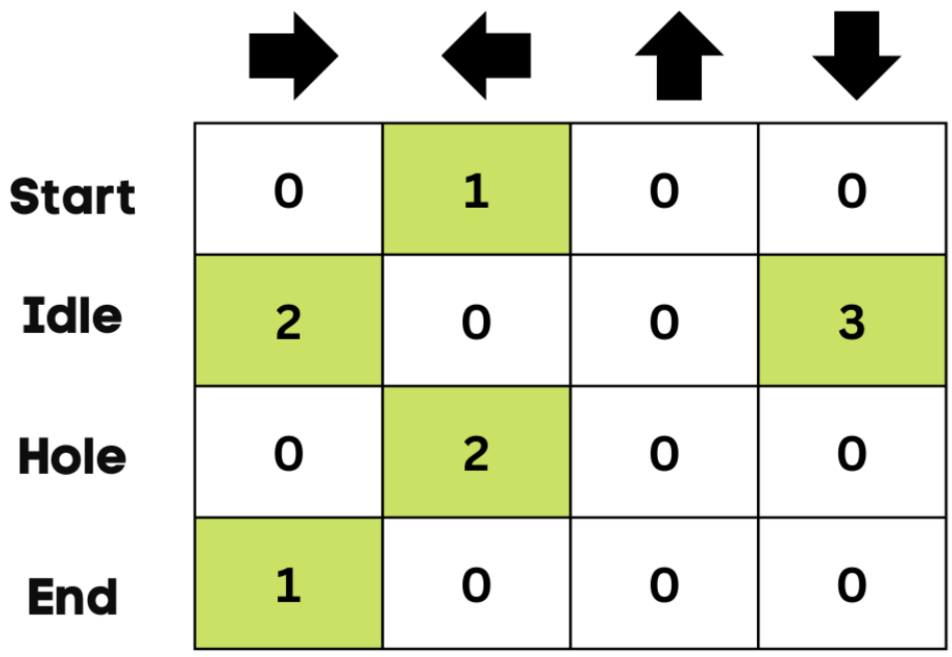
Update Q-Table

We will update the function Q(S_t, A_t) using the equation. It uses the previous episode's estimated Q-values, learning rate, and Temporal Differences error. Temporal Differences error is calculated using Immediate reward, the discounted maximum expected future reward, and the former estimation Q-value.

The process is repeated multiple times until the Q-Table is updated and the Q-value function is maximized.


$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

At the start, the agent is exploring the environment to update the Q-table. And when the Q-Table is ready, the agent will start exploiting and start taking better decisions.



	→	←	↑	↓
Start	0	1	0	0
Idle	2	0	0	3
Hole	0	2	0	0
End	1	0	0	0

In the case of a frozen lake, the agent will learn to take the shortest path to reach the goal and avoid jumping into the holes.