UNIT-III: *DECISION TREE LEARNING and INSTANCE-BASED LEARNING*

Table of Content

Decision Tree Algorithm

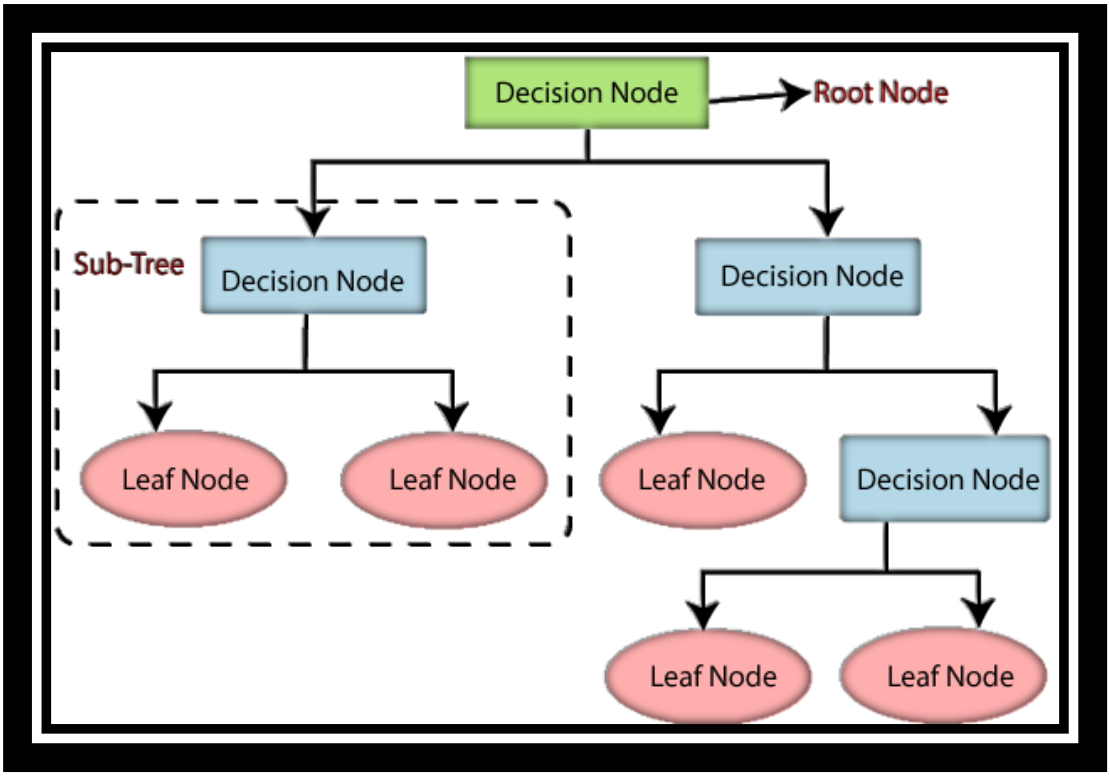
- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

> *A decision tree can contain categorical data (YES/NO) as well as numeric data.*

Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure



Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

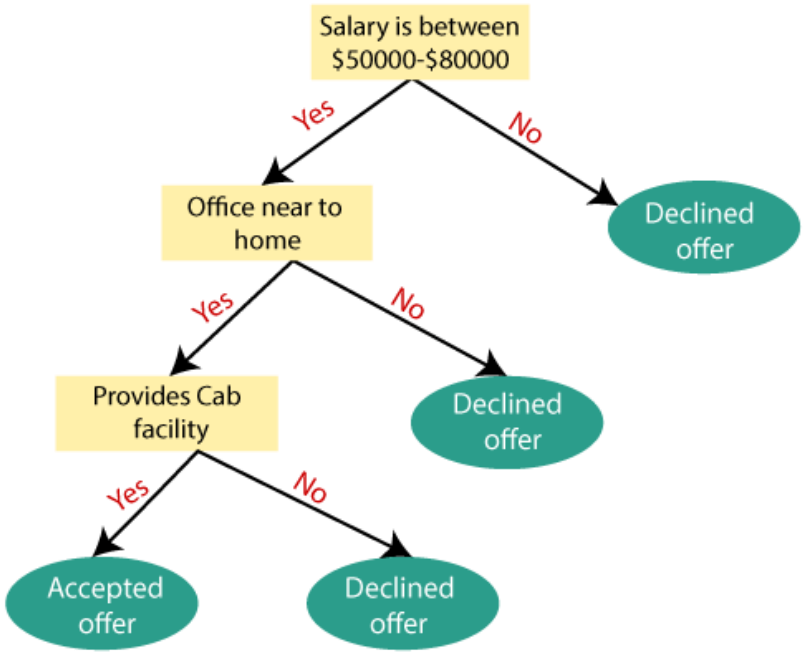
How does the Decision Tree algorithm Work?

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

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

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

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



Attribute Selection Measures

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

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

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

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

Types of Decision Tree Algorithms

The different decision tree algorithms are listed below:

- ID3(Iterative Dichotomiser 3)
- C4.5
- CART(Classification and Regression Trees)
- CHAID (Chi-Square Automatic Interaction Detection)
- MARS(Multivariate Adaptive Regression Splines)

Advantages:

1. **Interpretability**: Decision trees are easily interpretable and can be visualized, making them suitable for explaining the decision-making process to stakeholders and domain experts. The decision rules learned by decision trees are simple to understand and can be represented graphically.
2. **No Assumptions about Data Distribution**: Decision trees make no assumptions about the underlying distribution of the data, unlike parametric models such as linear regression. They can handle both numerical and categorical data without requiring data transformation.
3. **Implicit Feature Selection**: Decision trees automatically perform feature selection by selecting the most informative features at each split. Important features tend to appear closer to the root of the tree, making it easy to identify key predictors.
4. **Handling Non-Linear Relationships**: Decision trees can capture non-linear relationships between features and the target variable without the need for complex transformations or feature engineering. They can model complex decision boundaries with multiple splits.

5. **Easy to Handle Missing Values**: Decision trees can handle missing values in the data by splitting the data based on available features. They do not require imputation or deletion of missing values, simplifying preprocessing steps.

Certainly! Here are the advantages and limitations of the Decision Tree algorithm:

Advantages:

1. **Interpretability**: Decision trees are easily interpretable and can be visualized, making them suitable for explaining the decision-making process to stakeholders and domain experts. The decision rules learned by decision trees are simple to understand and can be represented graphically.
2. **No Assumptions about Data Distribution**: Decision trees make no assumptions about the underlying distribution of the data, unlike parametric models such as linear regression. They can handle both numerical and categorical data without requiring data transformation.
3. **Implicit Feature Selection**: Decision trees automatically perform feature selection by selecting the most informative features at each split. Important features tend to appear closer to the root of the tree, making it easy to identify key predictors.
4. **Handling Non-Linear Relationships**: Decision trees can capture non-linear relationships between features and the target variable without the need for complex transformations or feature engineering. They can model complex decision boundaries with multiple splits.
5. **Easy to Handle Missing Values**: Decision trees can handle missing values in the data by splitting the data based on available features. They do not require imputation or deletion of missing values, simplifying preprocessing steps.

Limitations:

1. **Overfitting**: Decision trees are prone to overfitting, especially when the tree is deep or when the dataset is noisy or contains irrelevant features. Deep trees can capture noise in the training data, leading to poor generalization performance on unseen data.
2. **Instability**: Small changes in the training data can result in different tree structures, leading to high variance and instability. Decision trees are sensitive to the training data, and slight variations in the dataset can produce different trees.
3. **Bias Toward Dominant Classes**: Decision trees tend to favor splits that result in pure subsets, leading to a bias toward dominant classes in the dataset. Imbalanced datasets may result in biased trees that perform poorly on minority classes.

4. **Greedy Nature**: Decision trees use a greedy approach to construct the tree by selecting the best split at each node based on local information. This may not always lead to the globally optimal tree structure and may result in suboptimal solutions.
5. **Limited Expressiveness**: Decision trees may not be expressive enough to capture complex relationships in the data, especially when the decision boundaries are highly non-linear. Ensemble methods such as Random Forests or Gradient Boosting can be used to improve expressiveness.

ID3 Algorithm

- The ID3 algorithm was developed by Ross Quinlan in 1986. It builds a decision tree by recursively partitioning the dataset into smaller subsets until all data points in each subset belong to the same class.
- It employs a top-down approach, selecting features to split the dataset based on **information gain**.
- ID3 primarily deals with categorical properties, making it suitable for problems with discrete input features.
- One of its strengths is its ability to generate interpretable decision trees.
- However, ID3 can be sensitive to noisy data and prone to overfitting.

How Does ID3 Work?

- The ID3 algorithm builds a decision tree in a top-down manner:
  - Start with the root node representing the entire dataset.
  - At each node, select the attribute that provides the most information gain about the target variable.
  - Information gain measures how much uncertainty (entropy) is reduced by splitting based on a particular attribute.
  - Recursively create child nodes for each possible value of the selected attribute.
  - Continue until all data points in a subset belong to the same class or a stopping criterion is met.

Interpretability and Visualization:

- ID3's resulting tree structure is easily understood and visualized, providing insight into the decision-making process.

Example:

Suppose we have a dataset with the following attributes: Outlook, Temperature, Humidity, and Windy, and the target variable is PlayTennis (whether to play tennis or not). Here's a sample of the dataset:

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rain	Mild	High	False	Yes
Rain	Cool	Normal	False	Yes
Rain	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rain	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rain	Mild	High	True	No

Step-by-Step Execution of ID3 Algorithm:

1. **Calculate Entropy of Target Variable**:
   - Calculate the entropy of the target variable PlayTennis, which measures the impurity or uncertainty of the dataset. Entropy is calculated using the formula:

$\text{Entropy}(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$ where p_i is the probability of class i. For our dataset, there are 9 instances of "Yes" and 5 instances of "No". So, $p(\text{Yes}) = \frac{9}{14}$ and $p(\text{No}) = \frac{5}{14}$. $\text{Entropy}(S) = -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) \approx 0.94$

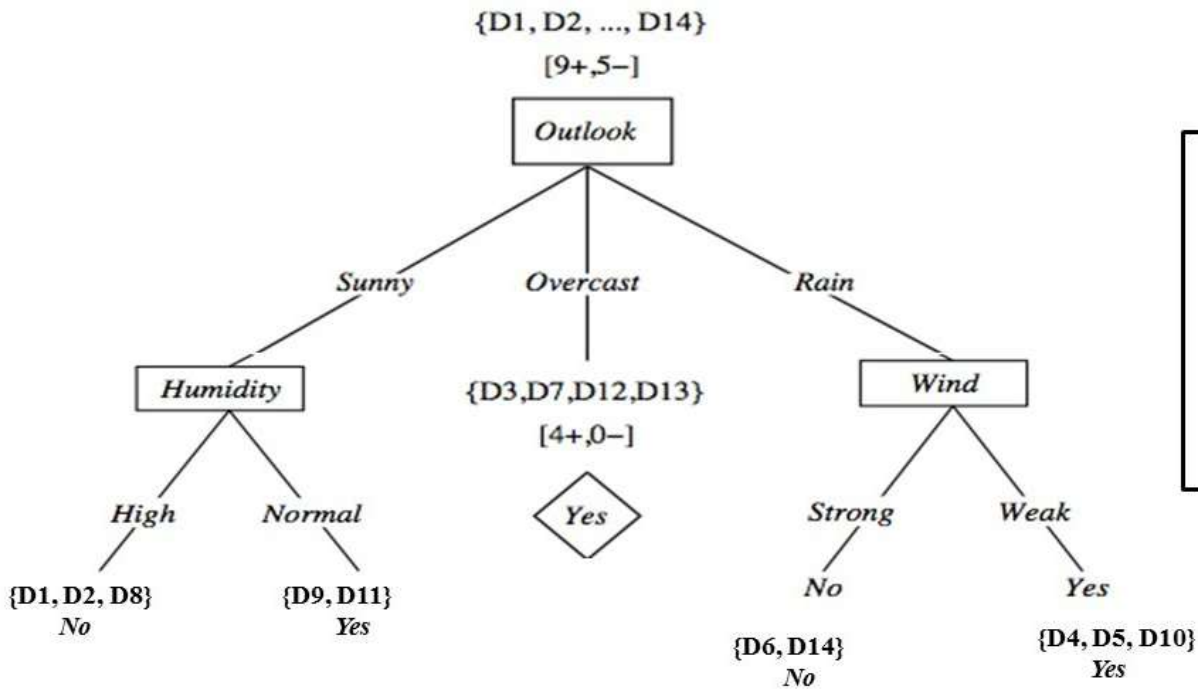
2. **Calculate Information Gain for Each Attribute**:
   - Calculate the information gain for each attribute (Outlook, Temperature, Humidity, Windy) based on the entropy of the target variable.
   - Information gain is calculated using the formula:

$\text{Gain}(A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \times \text{Entropy}(S_v)$ where A is the attribute, S_v is the subset of instances for each value of attribute A. For example, to calculate the gain for the Outlook attribute:

   - Split the dataset based on the values of Outlook (Sunny, Overcast, Rain).
   - Calculate the entropy of each subset (Sunny, Overcast, Rain) and their weighted average entropy.
   - Subtract the weighted average entropy from the overall entropy to get the information gain.

3. **Select the Attribute with the Highest Information Gain**:
   - Select the attribute with the highest information gain as the root node of the decision tree. In our example, let's assume Outlook has the highest information gain.

4. **Repeat Steps 1-3 Recursively**:
   - For each subset created by the split, repeat Steps 1-3 recursively until one of the stopping conditions is met (e.g., all instances in a subset belong to the same class, no more attributes to split, maximum depth reached).



Inductive Bias

In decision tree learning, the inductive bias refers to the assumptions and preferences encoded into the algorithm that guide the construction of the decision tree. These biases influence the choice of attributes for splitting, the structure of the resulting tree, and the generalization capabilities of the model. Here's how inductive bias manifests in decision tree learning:

1. Bias towards Simplicity:
Decision tree algorithms have a bias towards simpler explanations that can be easily understood and interpreted by humans. This bias leads to the preference for smaller, more interpretable trees over larger, more complex ones.

2. Attribute Selection Bias:
Decision tree algorithms use a greedy approach to select attributes for splitting at each node. This bias towards local optimization may lead to suboptimal trees, as the algorithm chooses the best attribute at each step without considering future splits.

3. Bias towards High Information Gain:
Decision tree algorithms, such as ID3 and C4.5, select the attribute that maximizes information gain or another splitting criterion (e.g., Gini impurity or gain ratio) at each node. This bias towards maximizing information gain helps to partition the data into more homogeneous subsets, leading to a better separation of classes.

4. Bias towards Small Trees:
Decision tree algorithms have an inductive bias towards smaller trees to avoid overfitting. They aim to find the simplest tree that adequately captures the underlying patterns in the data. This bias is often implemented through tree pruning techniques that remove unnecessary branches or nodes from the tree.

5. Bias towards Majority Class:
In the absence of a clear split based on the selected attribute, decision tree algorithms may bias towards the majority class. This bias ensures that the resulting tree makes a prediction based on the most prevalent class in the subset, reducing the risk of misclassification.

Importance of Inductive Bias in Decision Trees:
Interpretability: The bias towards simplicity and small trees makes decision trees highly interpretable, allowing users to understand the decision-making process and the factors influencing predictions.
Generalization: The bias towards high information gain and smaller trees helps decision trees generalize well to unseen data, leading to robust and accurate predictions.
Efficiency: The bias towards attribute selection and local optimization ensures that decision tree algorithms are computationally efficient, making them suitable for large datasets and real-time applications.

INSTANCE-BASED LEARNING

Instance-based learning, also known as instance-based learning or lazy learning, is a type of machine learning approach where the model learns from the training instances themselves rather than explicitly building a generalized model. In instance-based learning, the training data is memorized and used directly during the prediction phase.

Instance-based learning (also known as memory-based learning) is a family of machine learning algorithms that operate differently from traditional explicit generalization methods. Instead of

creating explicit models, instance-based learning compares new problem instances with instances seen during training, which are stored in memory

Instance-based learning leverages stored instances to make predictions or classifications, adapting to local patterns in the data. It's a flexible approach that can handle evolving datasets and provides a different perspective from traditional model-based methods

Principles of Instance-Based Learning:

1. **Data Memorization**: Instance-based learning algorithms memorize the entire training dataset, storing it for future reference during the prediction phase.
2. **Lazy Learning**: Instance-based learning is often referred to as lazy learning because it defers the processing of training data until a new instance needs to be classified or predicted.
3. **Similarity Measure**: Instance-based learning relies on a similarity measure (distance metric) to find the most similar instances in the training data to the new instance being classified.
4. **Local Decision Making**: Instead of building a global model, instance-based learning makes decisions based on the local neighborhood of similar instances in the feature space.

Advantages of Instance-Based Learning:

1. **Flexibility**: Instance-based learning can handle complex decision boundaries and non-linear relationships between features, as it directly uses the training instances for prediction.
2. **Adaptability**: Instance-based learning can adapt quickly to changes in the data distribution, as the model does not need to be retrained when new data becomes available.
3. **Interpretability**: Instance-based learning provides transparency, as predictions are based on the actual training instances, making it easier to understand and interpret the model's behavior.

Limitations of Instance-Based Learning:

1. **High Memory Requirements**: Storing the entire training dataset can be memory-intensive, especially for large datasets with many instances and features.
2. **Computational Complexity**: Making predictions with instance-based learning can be computationally expensive, as it requires calculating distances between the new instance and all training instances.

3. **Sensitivity to Noise**: Instance-based learning can be sensitive to noisy data, outliers, and irrelevant features, as it directly uses the training instances for prediction.

Applications of Instance-Based Learning:

1. **Classification**: k-Nearest Neighbors is commonly used for classification tasks, such as text categorization, image recognition, and medical diagnosis.
2. **Regression**: k-Nearest Neighbors can also be applied to regression tasks, such as predicting house prices or stock prices based on similar historical instances.
3. **Anomaly Detection**: Instance-based learning can be used for anomaly detection, where rare or unusual instances are identified based on their dissimilarity to normal instances.

Algorithms in Instance-Based Learning:

1. **k-Nearest Neighbors (k-NN)**: One of the most popular instance-based learning algorithms. Given a new instance, k-NN finds the k nearest neighbors in the training data and assigns the majority class label (for classification) or averages the target values (for regression).
2. **Case-Based Reasoning (CBR)**: CBR is a problem-solving paradigm that relies on past experiences (cases) to solve new problems. It stores a database of cases and retrieves the most similar cases to the current problem to make decisions.
3. **Locally Weighted Learning (LWL)**: Assigns different weights to training instances based on their proximity to the query instance.
4. Radial basis function networks

Comparison between lazy learning (instance-based learning) and eager learning (model-based learning)

Aspect	Lazy Learning (Instance-based Learning)	Eager Learning (Model-based Learning)
Processing	Defers processing of training data until prediction time.	Learns a model during the training phase and uses it for predictions.

Aspect	Lazy Learning (Instance-based Learning)	Eager Learning (Model-based Learning)
Training Time	Low, as no explicit model is constructed during training.	Higher, as the model needs to be trained on the entire dataset.
Prediction Time	Higher, as it requires calculating distances to all training instances.	Lower, as it involves applying the pre-learned model to new data.
Memory Usage	High, as it stores the entire training dataset for future reference.	Lower, as it only needs to store the learned model parameters.
Adaptability	Quick to adapt to changes in the data distribution or new instances.	Less adaptable to changes, as the model needs to be retrained.
Interpretability	Transparent, as predictions are based on the actual training instances.	May be less transparent, as predictions are based on a learned model.
Sensitivity to Noise	Can be sensitive to noisy data, outliers, and irrelevant features.	May be more robust to noise, depending on the chosen model.
Examples	k-Nearest Neighbours (k-NN), Case-Based Reasoning (CBR).	Decision Trees, Logistic Regression, Neural Networks, SVMs.

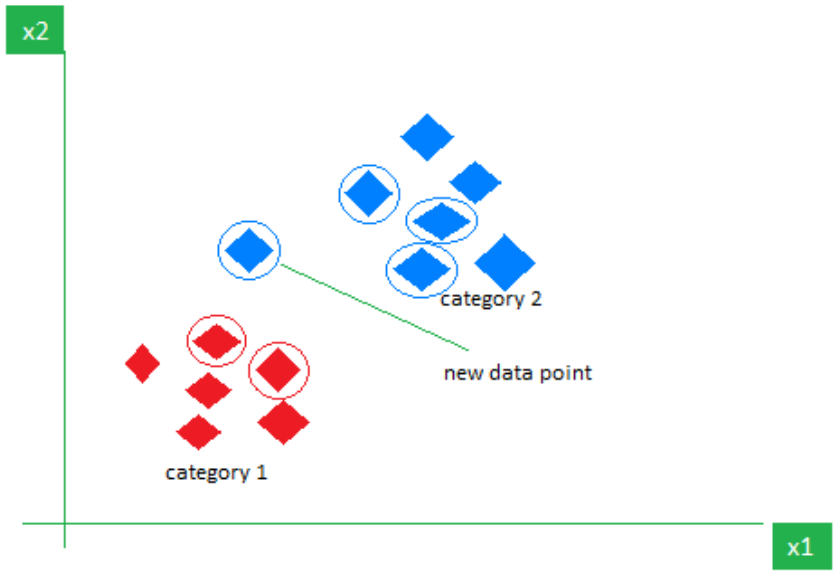
K-Nearest Neighbour (KNN) Algorithm

The **K-Nearest Neighbors (KNN) algorithm** is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which was subsequently expanded by Thomas Cover.

K-Nearest Neighbors Algorithm

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.
It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



Now, given another set of data points (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as 'White'.

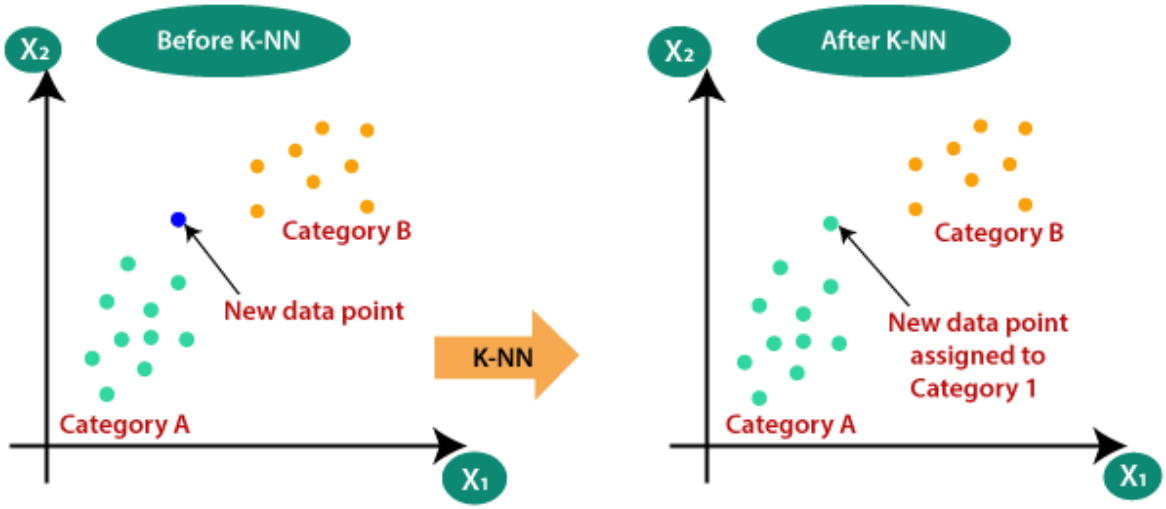
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.

- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

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

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

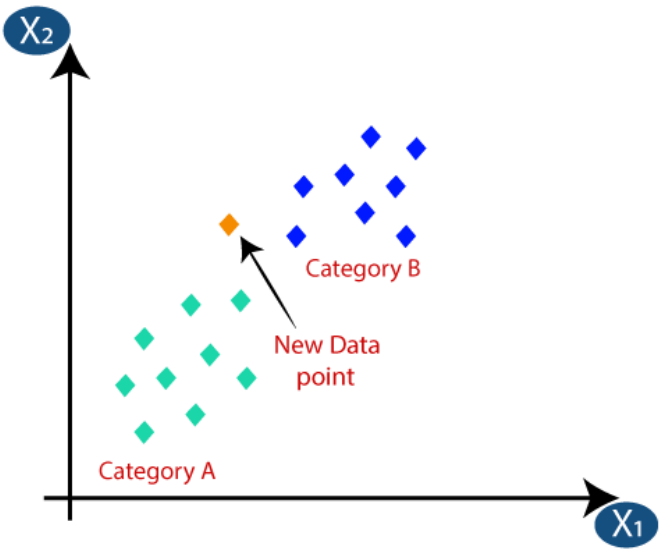


How does K-NN work?

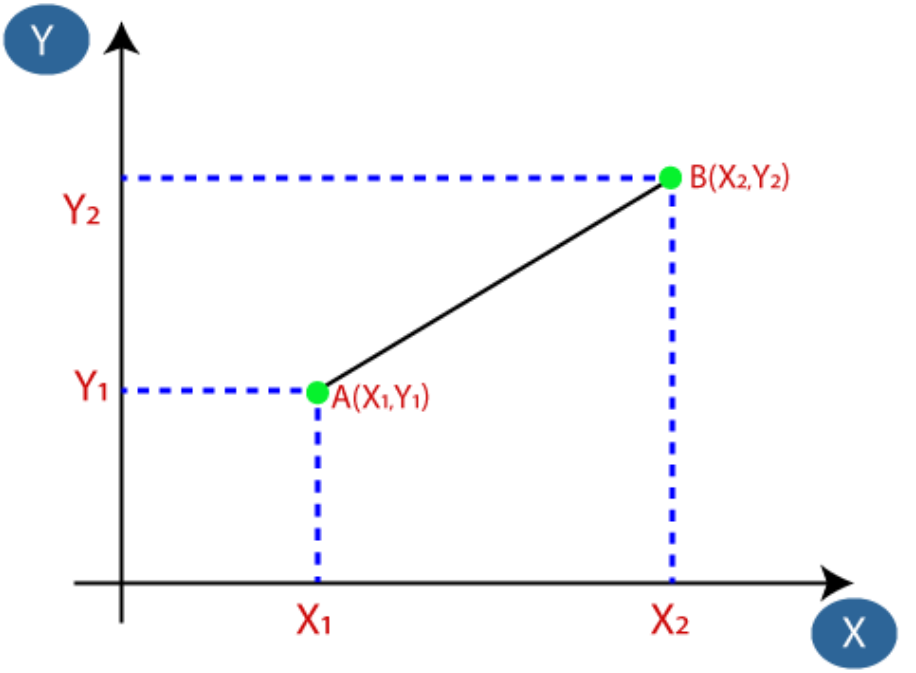
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

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

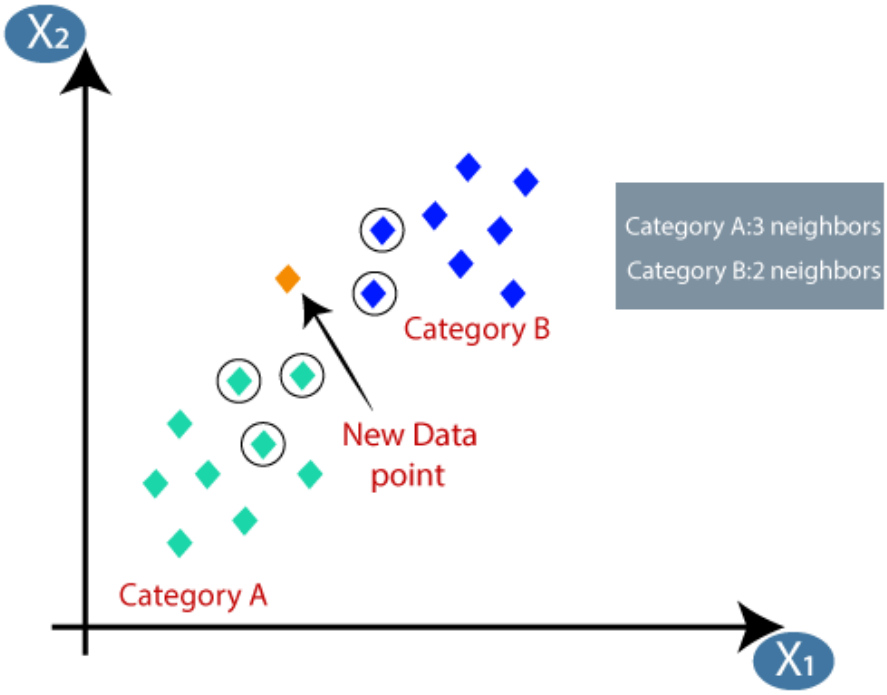


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



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.

Let's walk through a numeric example of k-Nearest Neighbors (k-NN) for classification. Suppose we have a dataset of points in a two-dimensional space, where each point belongs to one of two classes, "A" or "B". We want to classify a new point based on its nearest neighbors using the k-NN algorithm.

Dataset:

Consider the following dataset:

Data Point	Feature 1 (X1)	Feature 2 (X2)	Class Label
1	2	3	A
2	3	4	A
3	5	6	B
4	7	8	B

New Point:
Now, let's say we have a new point with coordinates (4, 5).

k-NN Algorithm:
Calculate Distance: Calculate the distance between the new point and each point in the dataset. For simplicity, let's use Euclidean distance.

$$\text{Distance}=(X1_{new}-X1_i)^2+(X2_{new}-X2_i)^2$$

For the new point (4, 5):

- Distance to point 1: $(4-2)^2+(5-3)^2=2^2+2^2=8\approx2.83$ $(4-2)^2+(5-3)^2=2^2+2^2=8\approx2.83$
- Distance to point 2: $(4-3)^2+(5-4)^2=1^2+1^2=2\approx1.41$ $(4-3)^2+(5-4)^2=1^2+1^2=2\approx1.41$
- Distance to point 3: $(4-5)^2+(5-6)^2=(-1)^2+(-1)^2=2\approx1.41$ $(4-5)^2+(5-6)^2=(-1)^2+(-1)^2=2\approx1.41$
- Distance to point 4: $(4-7)^2+(5-8)^2=(-3)^2+(-3)^2=18\approx4.24$ $(4-7)^2+(5-8)^2=(-3)^2+(-3)^2=18\approx4.24$

2. **Find Nearest Neighbors**: Choose the value of k k. Let's say $k=3$ $k=3$. Select the three nearest neighbors based on the calculated distances:
   - Nearest neighbors: Points 2, 3, and 1.
3. **Majority Vote**: Determine the majority class among the nearest neighbors. In this case, two neighbors belong to class A, and one belongs to class B. Therefore, the predicted class for the new point is A.

Advantages:

1. **Simplicity**: k-NN is easy to understand and implement, making it suitable for beginners.
2. **Non-parametric**: It makes no assumptions about the underlying data distribution, making it versatile and adaptable to various types of data.
3. **Flexibility**: k-NN can handle multi-class classification and regression tasks.
4. **Locally Adaptive**: It can capture complex decision boundaries and adapt to the local structure of the data.

Limitations:

1. **Computational Complexity**: As the size of the training dataset grows, the computational cost of finding the nearest neighbors increases.
2. **Memory Usage**: k-NN requires storing the entire training dataset in memory, which can be memory-intensive for large datasets.
3. **Sensitive to Noise**: It can be sensitive to noisy or irrelevant features, as it relies on the similarity between instances.
4. **Curse of Dimensionality**: Performance may degrade in high-dimensional spaces due to the increased sparsity of data points.

Applications:

1. **Classification**: k-NN is used in various domains such as image recognition, text categorization, and medical diagnosis.
2. **Regression**: It can be applied to regression tasks such as predicting house prices, stock prices, or weather forecasts.
3. **Anomaly Detection**: k-NN can be used for outlier detection or anomaly detection tasks.

Radial Basis Function Networks

The popular type of feed-forward network is the radial basis function (RBF) network. It has two layers, not counting the input layer, and contrasts from a multilayer perceptron in the method that the hidden units implement computations.

The Radial Basis Function (RBF) kernel, also known as the Gaussian kernel, is one of the most widely used kernel functions. It operates by measuring the similarity between data points based on their Euclidean distance in the input space. Mathematically, the RBF kernel between two data points, xx and x'x', is defined as:

$$K(\mathbf{x}, \mathbf{x'}) = \exp\left(-\frac{|\mathbf{x}-\mathbf{x'}|^2}{2\sigma^2}\right)$$

where

where, $|x-x'|2$|x−x'|2 represents the squared Euclidean distance between the two data points.

$\sigma\sigma$ is a parameter known as the bandwidth or width of the kernel, controlling the smoothness of the decision boundary.

Each hidden unit significantly defines a specific point in input space, and its output, or activation, for a given instance based on the distance between its point and the instance, which is only a different point. The closer these two points, the better the activation.

This is implemented by utilizing a nonlinear transformation function to modify the distance into a similarity measure. A bell-shaped Gaussian activation service of which the width can be different for each hidden unit is generally used for this objective. The hidden units are known as RBFs because the points in the instance area for which a given hidden unit makes a similar activation form a hypersphere or hyperellipsoid.

The output layer of an RBF structure is similar to that of a multilayer perceptron – It takes a linear set of the outputs of the hidden units and in classification issues passage it through the sigmoid function.

The parameters that such a network understands are the centers and widths of the RBFs and the weights used to design the linear set of the outputs acquired from the hidden layer. An essential benefit over multilayer perceptrons is that the first group of parameters can be decided independently of the second group and make accurate classifiers.

One method to decide the first group of parameters is to use clustering. The simple k-means clustering algorithm can be applied, clustering each class independently to obtain k-basis functions for each class.

The second group of parameters is understood by keeping the first parameters constant. This includes learning a simple linear classifier using one of the approaches such as linear or logistic regression. If there are long fewer hidden units than training instances, this can be done fastly.

The limitation of RBF networks is that they provide each attribute with a similar weight because all are considered equally in the distance computation unless attribute weight parameters are contained in the complete optimization process.

Therefore, they cannot deal efficiently with inappropriate attributes, against multilayer perceptrons. Support vector machines share similar issues. Support vector machines with Gaussian kernels (i.e., "RBF kernels") are a definite method of RBF network, in which one function is centered on each training instance, all basis functions have a similar width, and the outputs are merged linearly by calculating the maximum-margin hyperplane. This has the result that some of the RBFs have a nonzero weight the ones that define the support vectors.

Advantages:

1. **Non-Linearity**: RBF networks can capture non-linear relationships between input and output variables.
2. **Interpretability**: The centers of the radial basis functions can provide insight into the regions of input space that are most important for prediction.
3. **Generalization**: RBF networks tend to generalize well to unseen data when properly trained.

Limitations:

1. **Scalability**: RBF networks may struggle with scalability for high-dimensional data or large datasets, as the number of parameters increases with the number of dimensions and data points.
2. **Center Selection**: The selection of prototype or reference points can impact the performance of the network, and choosing appropriate centers can be challenging.
3. **Overfitting**: RBF networks are prone to overfitting if not properly regularized or if the number of radial basis functions is too high relative to the size of the training data.

Applications:

1. **Function Approximation**: RBF networks are used for function approximation tasks in engineering, finance, and physics.
2. **Time Series Prediction**: They can be applied to time series prediction tasks in finance, weather forecasting, and other domains.
3. **Classification**: RBF networks can be used for classification tasks in pattern recognition, medical diagnosis, and image processing

Locally Weighted Linear Regression

Locally Weighted Regression (LWR), also known as Locally Weighted Scatterplot Smoothing (LOWESS), is a non-parametric regression method used for fitting a regression line to a dataset. Unlike traditional regression methods that fit a global model to the entire dataset, LWR fits a

separate model to each data point, giving more weight to points that are closer to the point being predicted.

Within the field of machine learning and regression analysis, Locally Weighted Linear Regression (LWLR) emerges as a notable approach that bolsters predictive accuracy through the integration of local adaptation. In contrast to conventional linear regression models, which presume a universal correlation among variables, LWLR acknowledges the significance of localized patterns and relationships present in the data. In the subsequent discourse, we embark on an exploration of the fundamental principles, diverse applications, and inherent advantages offered by Locally Weighted Linear Regression. Our aim is to shed light on its exceptional capacity to amplify predictive prowess and furnish intricate understandings of intricate datasets.
Fundamentally, LWLR manifests as a non-parametric regression algorithm that discerns the connection between a dependent variable and several independent variables. Notably, LWLR's distinctiveness emanates from its dynamic adaptability, which empowers it to bestow distinct weights upon individual data points contingent on their proximity to the target point under prediction. In essence, this algorithm accords greater significance to proximate data points, deeming them as more influential contributors in the prediction process.

Principles of Locally Weighted Linear Regression

LWLR functions on the premise that the association between the dependent and independent variables adheres to linearity; however, this relationship is allowed to exhibit variability across distinct sections within the dataset. This is achieved by employing an individual linear regression model for each prediction, employing a weighted least squares technique. The determination of weights is carried out through a kernel function, which bestows elevated weights upon data points in close proximity to the target point and diminishes the weights for those that are farther away.

Advantages:

1. **Flexibility**: LWR can capture non-linear relationships between input and output variables.
2. **Local Adaptation**: It adapts the regression model to the local structure of the data, giving more weight to nearby points.
3. **Robustness**: LWR is robust to outliers and noise in the data, as it focuses on the local neighborhood of each data point.

Limitations:

1. **Computational Complexity**: LWR can be computationally expensive, especially for large datasets, as it requires fitting a separate model to each data point.

2. **Overfitting**: LWR may overfit the training data if the bandwidth parameter $\tau\tau$ is too small or if there are too few data points in the local neighborhood.
3. **Bandwidth Selection**: Choosing an appropriate value for the bandwidth parameter $\tau\tau$ can be challenging and may require cross-validation or other optimization techniques.

Applications:

1. **Time Series Forecasting**: LWR can be used for time series forecasting tasks, where the relationship between input and output variables may vary over time.
2. **Anomaly Detection**: It can be applied to anomaly detection tasks, where outliers or unusual patterns need to be identified.
3. **Function Approximation**: LWR can be used for function approximation tasks in various domains such as engineering, finance, and physics.

Case-Based Learning (CBL)

Case-Based Reasoning(CBR) resolve new problems by adjusting previously fortunate solutions to alike problems. Roger Schank is widely held to be the beginning of CBR. He proposed a unalike sight on model-based reasoning stimulated by human logical and memory organization.
CBL is a machine learning paradigm that relies on past experiences, called cases, to solve new problems. It is inspired by the way humans learn from previous experiences and apply that knowledge to similar situations.

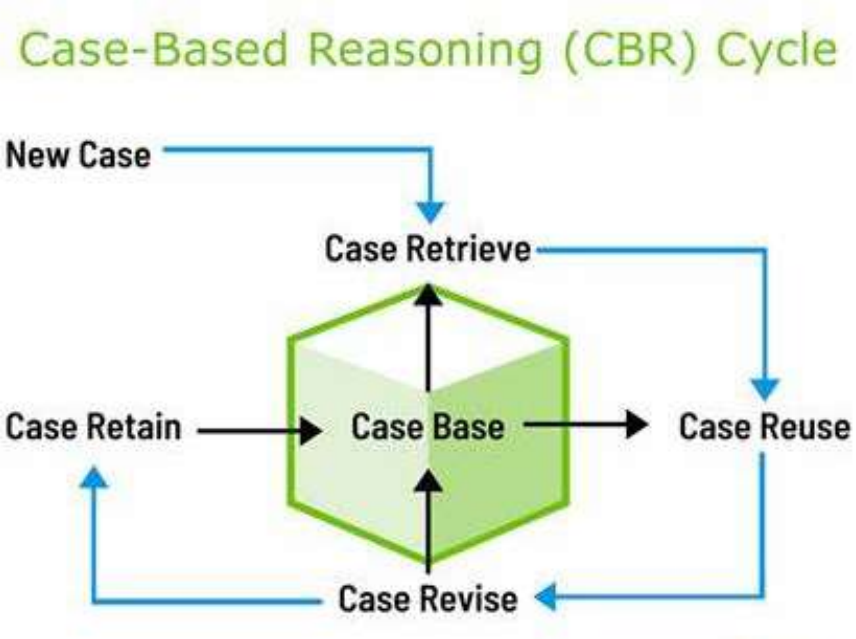
Basis of CBR :

Here, we will discuss the basis key parameters of CBR.

1. Regularity- The identical steps executed under the same circumstances will tend to have the same or alike outcomes.
2. Typicality- Experiences tend to repeat themselves.
3. Consistency-
   Minor switch in the circumstances require merely small changes in the explanation and in the effect.
4. Adaptability- When things replicate, the dissimilarities tend to be minute, and the small differences are uncomplicated to repay for.

Working Cycle of CBR :

Here, we will discuss the working cycle of CBR.



- **Case retrieval –**
  After the issue result has been judged, the best coordinating case is explored in the case base and an estimated solution is retrieved.

- **Case adaptation –**
  The recovered result is adjusted to fit finer the new issue.

- **Solution evaluation –**
  The modified solution can be judged either before the solution is applied to the complication or after the solution has been applied, the modified solution must be adapted again or more cases should be modified.

- **Case- based updating –**
  If the solution was verified as correct the new case may be added to the case.

Knowledge in CBR :

Vocabulary includes the knowledge necessary for choosing the features utilized to describe the cases.

- Case features have to be specified so that they can be helpful in retrieving other cases, which contains useful solutions to similar problems.
- Similarity estimation include the mastery about the similarity measure itself and the grip used to choose the most efficient firm of the employed case base and the most suitable case-retrieval method.
- Modification knowledge includes the knowledge necessary for executing the adaptation and evaluation phases of the CBR working cycle.
- Cases contain knowledge about solved problem instances and, in many CBR systems, this represents the knowledge that the system acquire during use.

Benefits of CBR:

Here, we will discuss the benefits of CBR.

- CBR supports ease of knowledge elicitation.
- CBR works efficiently in the absence of problem solving bias.
- It is suitable for multiplex and not completely formalized result position.
- It holds up ease of explanation.
- It carry ease of maintenance.

Limitations:

Here, we will discuss the limitations of CBR.

- CBR finds it complex to handle large case bases.
- It is almost impossible for CBR to solve dynamic domain problems
- CBR method is unable in handling noisy data