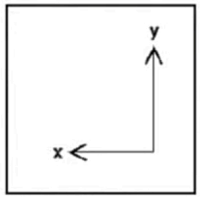
Distance Based Models

- Distance-based models are the second class of Geometric models. Like Linear models, distance-based models are based on the geometry of data.
- Distance-based models work on the concept of distance.
- A good distance based model helps in improving the performance of Classification, Clustering and Information Retrieval process significantly.
- The distance based model helps algorithms to recognize similarities between the contents.

Distance Based Models

- The distance metrics commonly used are **Euclidean**, **Minkowski**, **Manhattan**



Manhattan

Euclidean

Distance Based Models

- Distance is applied through the concept of neighbours and exemplars.
- Neighbours are points in proximity with respect to the distance measure expressed through exemplars.
- Exemplars are either centroids that find a centre of mass according to a chosen distance metric or medoids that find the most centrally located data point.
- The centroid represents the geometric centre of a plane figure, i.e., the arithmetic mean position of all the points in the figure from the centroid point.
- Medoids are similar in concept to means or centroids. Medoids are most commonly used on data when a mean or centroid cannot be defined.

Distance Based Models

➢ **Distance Metrics**

To understand the mathematics behind these Distance metrics identify the machine learning algorithms where use these distance metrics. Below are the commonly used distance metrics -

1. Hamming Distance
2. Euclidean Distance
3. Manhattan Distance (Taxicab or City Block)
4. Minkowski Distance

Distance Based Models

➢ **Role of Distance Measures**

- A distance measure is an objective score that summarizes the relative difference between two objects in a problem domain.
- Popular machine learning algorithms that use distance measures at their core is as follows:

1. K-Nearest Neighbours
2. Learning Vector Quantization (LVQ)
3. Self-Organizing Map (SOM)
4. K-Means Clustering

(KNN CLASSIFIATION SOLVED PROBLEM-2)

Data Warehouse and Data Mining - Lecture Series [Mumbai Univ, Pune Univ, GTU, UPTU, GGSIPU, MDU and other University]

Solved Numerical on KNN Classification:-

Ques:) Perform KNN-classification Algorithm on following dataset and Predict the class for X ($P_1 = 3$ and $P_2 = 7$). K=3 → 3 nearest neighbours.

	P_1	P_2	Class
i)	7	7	False
ii)	7	4	False
iii)	3	4	True
iv)	1	4	True

Euclidean distance $= \sqrt{(x_H - H_1)^2 + (x_W - W)^2 + \ldots}$

Observed Value

Actual Value

$D(x, i) = \sqrt{(3-7)^2 + (7-7)^2} = 4 \longrightarrow N3 \rightarrow FALSE$

$D(x, ii) = \sqrt{(3-7)^2 + (7-4)^2} = \sqrt{16+9} = 5$

$D(x, iii) = \sqrt{(3-3)^2 + (7-4)^2} = 3 \longrightarrow N1 \rightarrow TRUE$

$D(x, iv) = \sqrt{(3-1)^2 + (7-4)^2} = \sqrt{4+9} = 3.6 \longrightarrow N2 \rightarrow TRUE$

2 TRUE > 1 FALSE

X($P_1 = 3$, $P_2 = 7$) will belong to class TRUE

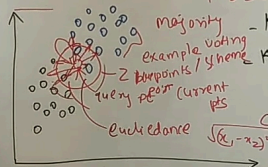
ANS..

k nearest neighbour find eculidin distance mark the true false class and predict the answer

K-Nearest Neighbour KNN

- Identifies data points that are separated into several classes to predict the classification of a new sample point.
- A LAZY algorithm. memorises the process
- classifies new points based on a Similarity measure. euclidean distance

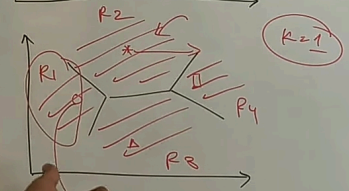
N = 30 K = 3



- K must be ODD
- K must not be a multiple of classes

$C = 7 \times 14$

$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$



Algorithm:

S1: load Data

S2: Initialize K.

S3: For each sample in the training data,

S3.1 Calculate distance between query point & the current point.

S3.2 Add the distance & the index of the example to an ordered Collection.

S4: Sort → ordered collection of distances & indexes from Small to large

S5: Pick first K entries from Sorted collection.

S6: Get the labels of selected K entries.

S7: If regression → return mean of K labels.

S8: If classification → return mode of K labels.

veranoi classification if k=1

Ques) Divide the given Sample Data in two (2) clusters using K-Means Algorithm [Euclidean Distance].

	Height (H)	Weight (W)	
1	185	72	✓
2	170	56	✓
3	168	60	
4	179	68	=
5	182	72	
6	188	77	
7	180	71	
8	180	70	
9	183	84	
10	180	88	
11	180	67	
12	177	76	

$C_1 \rightarrow \{1,$
$C_2 \rightarrow \{2,3$

Made two clusters c1 ,c2 mark the clusters with the centroid then find eculidin distance whonever is less add in the clusters take the mean and calculate the new centroid

$$\sqrt{(X_H - H_1)^2 + (X_W - W_1)^2}$$

O.V

X_H: Observed Value; H_1: Centroid Value; W_1: Centroid Value

i) Initialize two clusters.

	H	W	Centroid
C1	185	72	(185, 72)
C2	170	56	(170, 56)

E.D of Row 3

C_1: $\sqrt{(168-185)^2 + (60-72)^2} = \sqrt{289+144}$ [(20.80)]

C_2: $\sqrt{(168-170)^2 + (60-56)^2} = \sqrt{4+16}$ [(4.48)]

$$C_2\left(\frac{170+168}{2}, \frac{60+56}{2}\right)$$

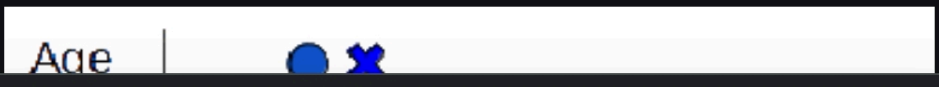
C_2 [169, 58]

Table of contents

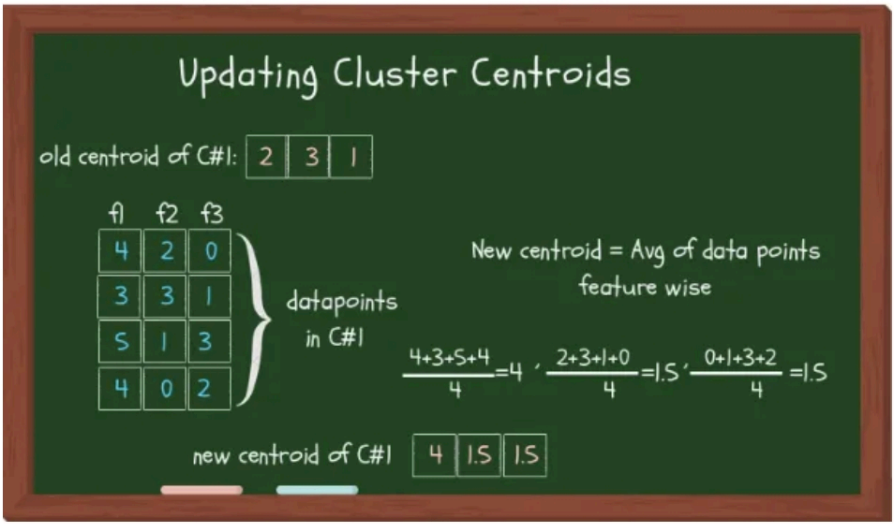
4. Re-initialize centroids

Next, we will re-initialize the centroids by calculating the average of all data points of that cluster.

$$Ci = \frac{1}{|Ni|} \sum xi$$



the updated cluster centorid is the average or the mean value of all the datapoints within that cluster.



Updating cluster centroids

Now if some other algo, like K-Mode, or K-Median was used, instead of taking the average value, mode and median would be taken respectively.

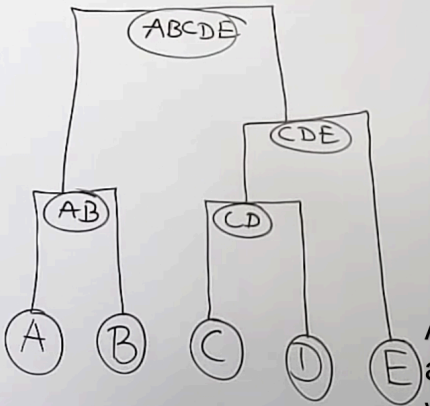
4. Stopping Criterion

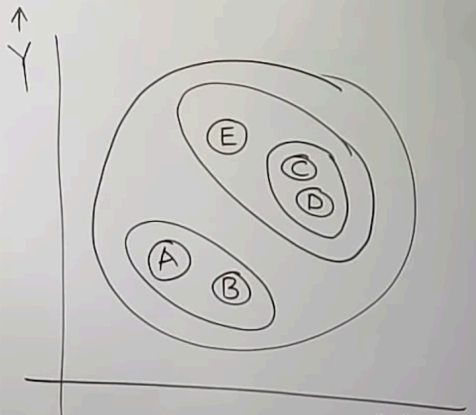
Since step 2 and 3 would be performed iteratively, it would go on forever if we

Hierarchical Clustering

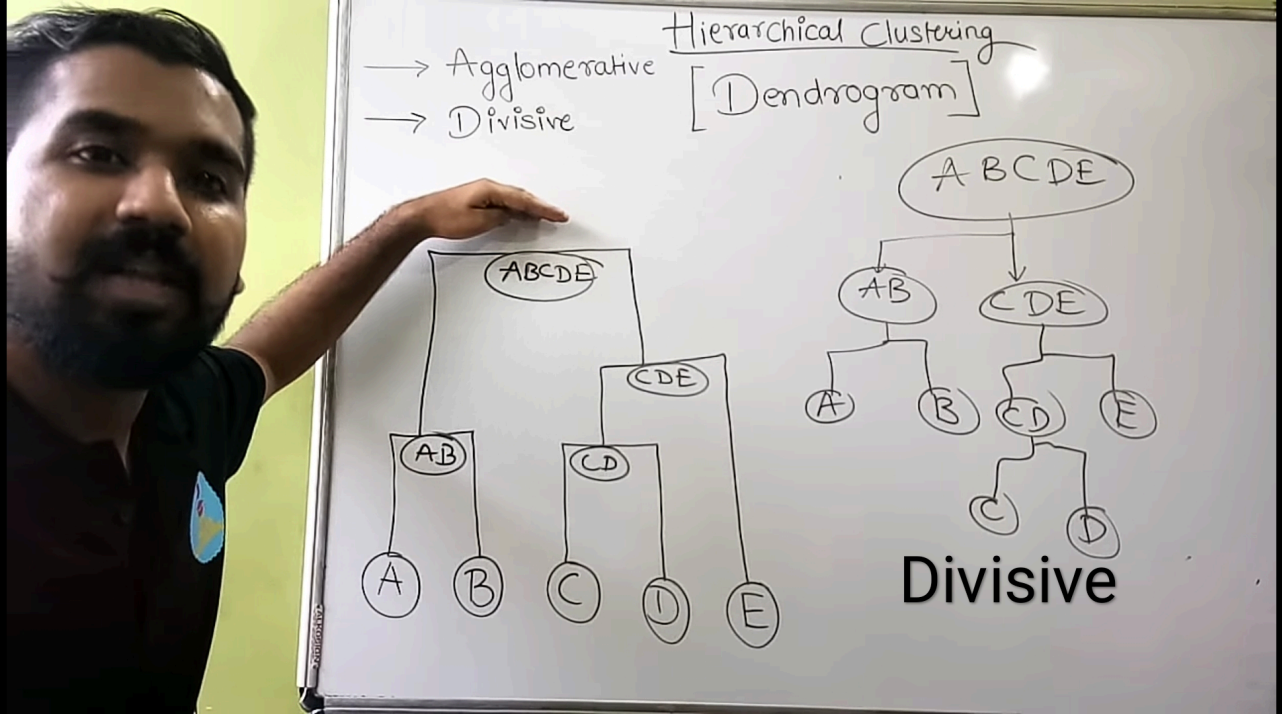
→ Agglomerative
→ Divisive

[Dendrogram]





Agglomerative follows bottom up approach, Called Dendrogram wherr Divisive follow top down approach


Hierarchical Clustering
→ Agglomerative
→ Divisive
[Dendrogram]
ABCDE
AB
CDE
A
B
C
D
E
Divisive

Agglomerative Clustering

	P_1	P_2	P_3	P_4	P_5
P_1	0				
P_2	9	0			
P_3	3	7	0		
P_4	6	5	9	0	
P_5	11	10	2	8	0

$\Rightarrow$

	P_1	P_2	$[P_3, P_5]$	P_4
P_1	0			
P_2	9	0		
$[P_3, P_5]$	3	7	0	
P_4	6	5	8	0

$\Rightarrow d(P_1, [P_3, P_5])$

$\Rightarrow \min(d(P_1, P_3), d(P_1, P_5))$

$\Rightarrow \min(3, 11) \Rightarrow 3$

$\Rightarrow d(P_2, [P_3, P_5])$

$\Rightarrow \min(d(P_2, P_3), d(P_2, P_5))$

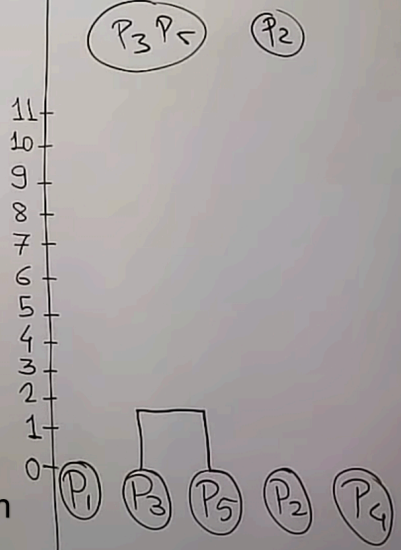
$\Rightarrow \min(7, 10) \Rightarrow 7$

$\Rightarrow d(P_4, [P_3, P_5])$

$\Rightarrow \min(d(P_4, P_3), d(P_4, P_5))$

$\Rightarrow \min(9, 8) \Rightarrow 8$

In single linkage technique use min of distance between p1 to p3, p4



Agglomerative Clustering

	P_1	P_2	P_3	P_4	P_5
P_1	0				
P_2	9	0			
P_3	3	7	0		
P_4	6	5	9	0	
P_5	11	10	2	8	0

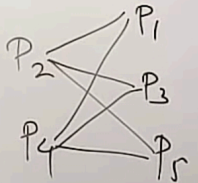
	$[P_1 P_3 P_5]$	P_2	P_4
$[P_1 P_3 P_5]$	0		
P_2	7	0	
P_4	6	5	0

$d(P_2, [P_1 P_3 P_5])$

$\Rightarrow \min(d(P_2,P_1), d(P_2,P_3), d(P_2,P_5))$

$\Rightarrow \min(9, 7, 10) \Rightarrow 7$

	$[P_1 P_3 P_5]$	$[P_2 P_4]$
$[P_1 P_3 P_5]$	0	
$[P_2 P_4]$	6	0



$d([P_1 P_3 P_5], [P_2 P_4])$

$= \min(d(P_2,P_1), d(P_2,P_3), d(P_2,P_5), d(P_4,P_1), d(P_4,P_3), d(P_4,P_5))$

$= \min(9, 7, 10, 6, 9, 8)$

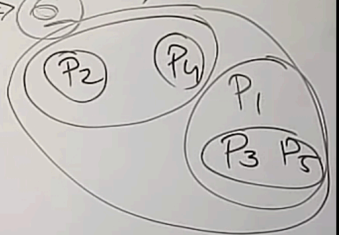
$= 6$

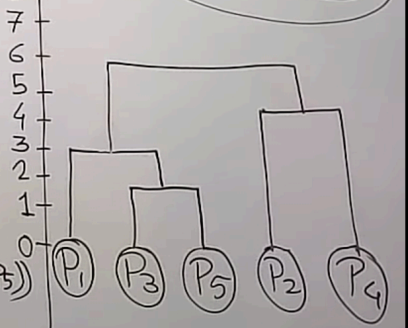
$d(P_4, [P_1 P_3 P_5])$

$\Rightarrow \min(d(P_4,P_1), d(P_4,P_3), d(P_4,P_5))$

$\Rightarrow \min(6, 9, 8)$

$\Rightarrow 6$





Agglomerative Clustering

	P_1	P_2	P_3	P_4	P_5
P_1	0				
P_2	9	0			
P_3	3	7	0		
P_4	6	5	9	0	
P_5	11	10	2	8	0

$\Rightarrow$

	P_1	P_2	$[P_3\ P_5]$	P_4
P_1	0			
P_2	9	0		
$[P_3\ P_5]$	11	10	0	
P_4	6	5	9	0

$d(P_2, [P_3\ P_5])$

$\Rightarrow \max(d(P_2, P_3), d(P_2, P_5)) \Rightarrow \max(7, 10) = 10$

$d(P_1, [P_3\ P_5])$

$\Rightarrow \max(d(P_1, P_3), d(P_1, P_5)) \Rightarrow \max(3, 11) = 11$

$d(P_4, [P_3\ P_5])$

$\Rightarrow \max(d(P_4, P_3), d(P_4, P_5)) \Rightarrow \max(9, 8) = 9$

	$[P_1 P_2 P_4]$	$[P_3\ P_5]$
$[P_1 P_2 P_4]$	0	
$[P_3\ P_5]$	11	0

P_3 P_1 P_2 P_5 P_4

	P_1	$[P_2\ P_4]$	$[P_3\ P_5]$
P_1	0		
$[P_2 P_4]$	9	0	
$[P_3 P_5]$	11	10	0

P_3 P_5

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

P_1 P_2 P_4 P_3 P_5

In complete linkage we use max of distance

ML | K-Medoids clustering with solved example

K-Medoids (also called Partitioning Around Medoid) algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as a point in the cluster, whose dissimilarities with all the other points in the cluster are minimum. The dissimilarity of the medoid(Ci) and object(Pi) is calculated by using E = |Pi – Ci|

The cost in K-Medoids algorithm is given as

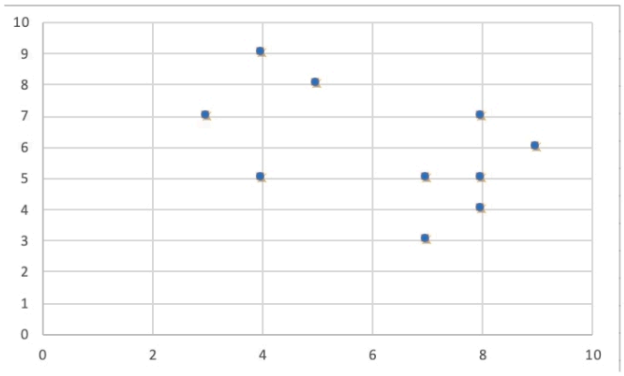
$$c = \sum_{Ci} \sum_{Pi \in Ci} |Pi - Ci|$$

Algorithm:

1. Initialize: select *k* random points out of the *n* data points as the medoids.
2. Associate each data point to the closest medoid by using any common distance metric methods.
3. While the cost decreases: For each medoid m, for each data o point which is not a medoid:
   - Swap m and o, associate each data point to the closest medoid, and recompute the cost.
   - If the total cost is more than that in the previous step, undo the swap.

	X	Y
0	8	7
1	3	7
2	4	9
3	9	6
4	8	5
5	5	8
6	7	3
7	8	4
8	7	5
9	4	5

Let's consider the following example: If a graph is drawn using the above data points, we obtain the following:



Step 1: Let the randomly selected 2 medoids, so select k = 2, and let **C1 -(4, 5)** and **C2 -(8, 5)** are the two medoids.

Step 2: Calculating cost. The dissimilarity of each non-medoid point with the medoids is calculated and tabulated:

3	9	6	6	2
4	8	5	-	-
5	5	8	4	6
6	7	3	5	3
7	8	4	5	1
8	7	5	3	1
9	4	5	-	-

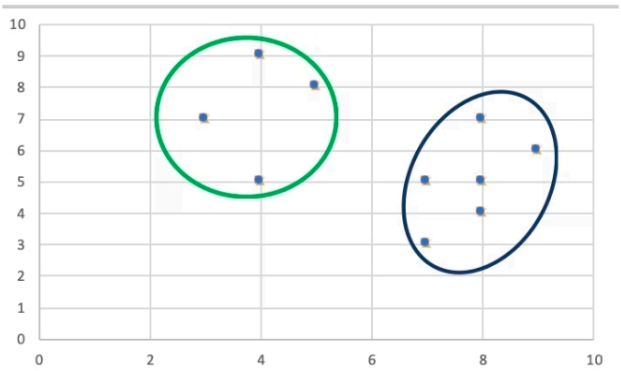
Here we have used Manhattan distance formula to calculate the distance matrices between medoid and non-medoid points. That formula tell that **Distance = |X1-X2| + |Y1-Y2|.**

Each point is assigned to the cluster of that medoid whose dissimilarity is less. Points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. The Cost = (3 + 4 + 4) + (3 + 1 + 1 + 2 + 2) = 20

Step 3: randomly select one non-medoid point and recalculate the cost. Let the randomly selected point be (8, 4). The dissimilarity of each non-medoid point with the medoids – C1 (4, 5) and C2 (8, 4) is calculated and tabulated.

	X	Y	Dissimilarity from C1	Dissimilarity from C2
0	8	7	6	3
1	3	7	3	8
2	4	9	4	9
3	9	6	6	3
4	8	5	4	1
5	5	8	4	7
6	7	3	5	2
7	8	4	-	-
8	7	5	3	2
9	4	5	-	-

Each point is assigned to that cluster whose dissimilarity is less. So, points 1, 2, and 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2. The New cost = (3 + 4 + 4) + (2 + 2 + 1 + 3 + 3) = 22 Swap Cost = New Cost – Previous Cost = 22 – 20 and **2 >0** As the swap cost is not less than zero, we undo the swap. Hence (4, 5) and (8, 5) are the final medoids. The clustering would be in the following way The **time complexity** is $O(k * (n-k)^2)$.



Advantages:

1. It is simple to understand and easy to implement.
2. K-Medoid Algorithm is fast and converges in a fixed number of steps.
3. PAM is less sensitive to outliers than other partitioning algorithms.

Disadvantages:

1. The main disadvantage of K-Medoid algorithms is that it is not suitable for clustering non-spherical (arbitrarily shaped) groups of objects. This is because it relies on minimizing the distances between the non-medoid objects and the medoid (the cluster center) – briefly, it uses compactness as clustering criteria instead of connectivity.
2. It may obtain different results for different runs on the same dataset because the first k medoids are chosen randomly.

Article Tags : Machine Learning ML-Clustering

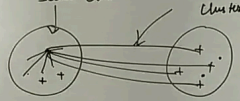
Silhouette cofficient use to find ratio of cohesion and seperat

Cohesion & Separation.

Silhouette Coefficient

measures the distance of one. pt to other pts in the same cluster.

measure of one pt in one cluster. measure with pts in other clusters.



[-1, 1].

-1 ⇒ clustering is wrong.

0 ⇒ indifferent - same.

1 ⇒ both clusters are far away.

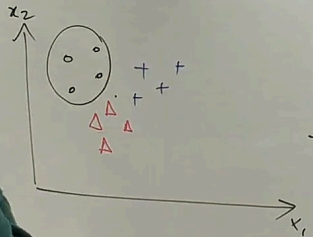
S1: Create a distance matrix. $\|x_i - y_i\|_2$ 6x6.

S2: x, data point

S2.1. a → cohesion. (Intra-class dist)

S2.2. b → other clusters (Separation).

S2.3. $S = \frac{(b-a)}{\max(a,b)}$ ← normalize.



OR.

$a < b$, $1 - \left(\frac{a}{b}\right)$.

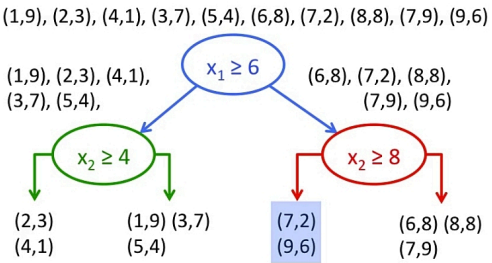
$a > b$ $\left(\frac{b}{a}\right) - 1$

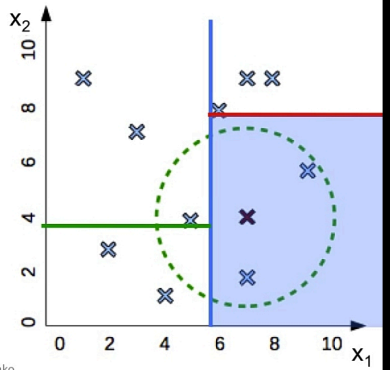
$a = b$ 0.

In kd trees we split the data based on x axis or range values then apply binary search

K-D tree example

- Building a K-D tree from training data:
  - pick random dimension, find median, split data, repeat
- Find NNs for new point (7,4)
  - find region containing (7,4)
  - compare to all points in region





Search and Insertion in K Dimensional tree

What is K dimension tree?

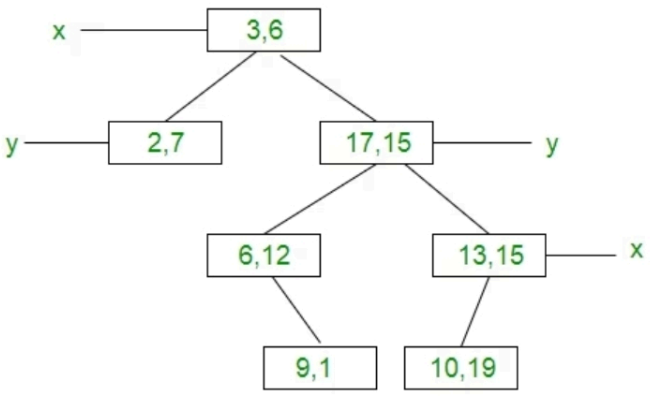
A K-D Tree(also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. In short, it is a space partitioning(details below) data structure for organizing points in a K-Dimensional space. A non-leaf node in K-D tree divides the space into two parts, called as half-spaces. Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree. We will soon be explaining the concept on how the space is divided and tree is formed. For the sake of simplicity, let us understand a 2-D Tree with an example. The root would have an x-aligned plane, the root's children would both have y-aligned planes, the root's grandchildren would all have x-aligned planes, and the root's great-grandchildren would all have y-aligned planes and so on.

Generalization: Let us number the planes as 0, 1, 2, ...(K – 1). From the above example, it is quite clear that a point (node) at depth D will have A aligned plane where A is calculated as: A = D mod K

How to determine if a point will lie in the left subtree or in right subtree? If the root node is aligned in planeA, then the left subtree will contain all points whose coordinates in that plane are smaller than that of root node. Similarly, the right subtree will contain all points whose coordinates in that plane are greater-equal to that of root node.

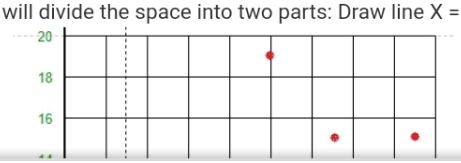
Creation of a 2-D Tree: Consider following points in a 2-D plane: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

1. Insert (3, 6): Since tree is empty, make it the root node.
2. Insert (17, 15): Compare it with root node point. Since root node is X-aligned, the X-coordinate value will be compared to determine if it lies in the right subtree or in the left subtree. This point will be Y-aligned.
3. Insert (13, 15): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, they are equal, this point will lie in the right subtree of (17, 15). This point will be X-aligned.
4. Insert (6, 12): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, 12 < 15, this point will lie in the left subtree of (17, 15). This point will be X-aligned.
5. Insert (9, 1):Similarly, this point will lie in the right of (6, 12).
6. Insert (2, 7):Similarly, this point will lie in the left of (3, 6).
7. Insert (10, 19): Similarly, this point will lie in the left of (13, 15).



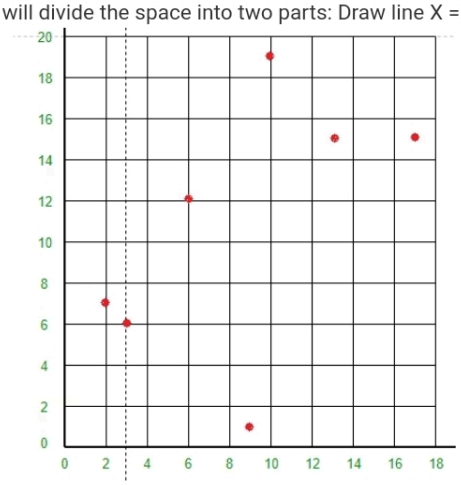
How is space partitioned? All 7 points will be plotted in the X-Y plane as follows:

1. Point (3, 6) will divide the space into two parts: Draw line X = 3.

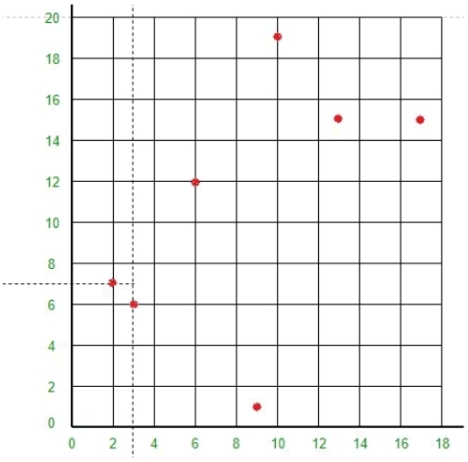


How is space partitioned? All 7 points will be plotted in the X-Y plane as follows:

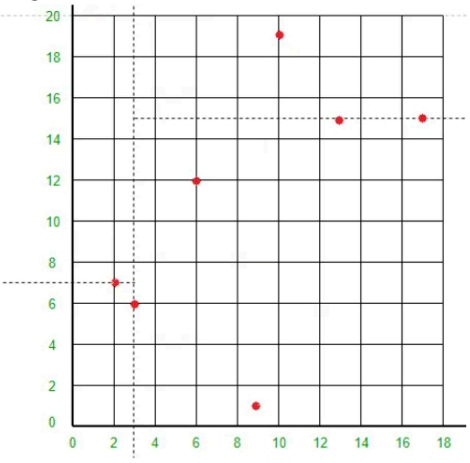
1. Point (3, 6) will divide the space into two parts: Draw line X = 3.



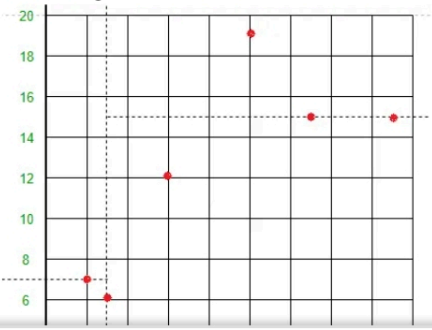
2. Point (2, 7) will divide the space to the left of line X = 3 into two parts horizontally. Draw line Y = 7 to the left of line X = 3.



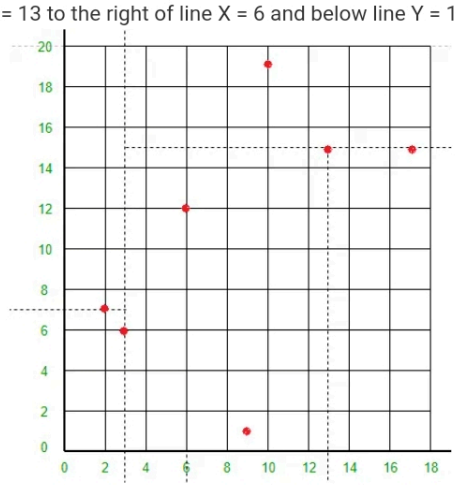
3. Point (17, 15) will divide the space to the right of line X = 3 into two parts horizontally. Draw line Y = 15 to the right of line X = 3.



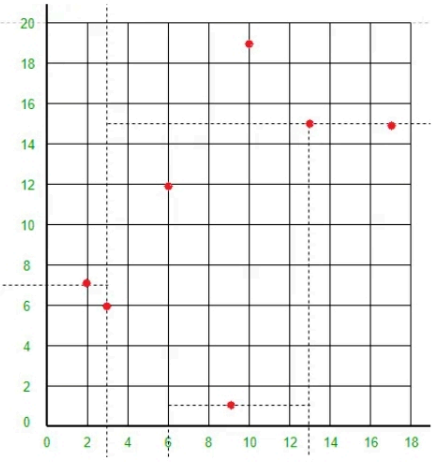
- Point (6, 12) will divide the space below line Y = 15 and to the right of line X = 3 into two parts. Draw line X = 6 to the right of line X = 3 and below line Y = 15.



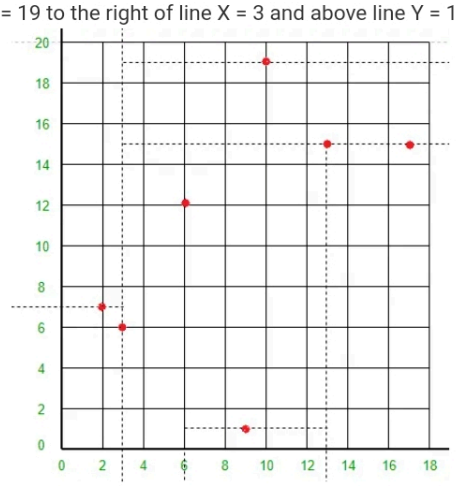
- Point (13, 15) will divide the space below line Y = 15 and to the right of line X = 6 into two parts. Draw line X = 13 to the right of line X = 6 and below line Y = 15.



- Point (9, 1) will divide the space between lines X = 3, X = 6 and Y = 15 into two parts. Draw line Y = 1 between lines X = 3 and X = 13.



- Point (10, 19) will divide the space to the right of line X = 3 and above line Y = 15 into two parts. Draw line Y = 19 to the right of line X = 3 and above line Y = 15.



Following is C++ implementation of KD Tree basic operations like search, insert and delete.

C++ | Java | Python3 | C# | Javascript

```
// A C++ program to demonstrate operations of KD tree
#include<bits/stdc++.h>
using namespace std;

const int k = 2;

// A structure to represent node of kd tree
struct Node
{
    int point[k]; // To store k dimensional point
    Node *left, *right;
```

```
    int n = sizeof(points)/sizeof(points[0]);
    for (int i=0; i<n; i++)
       root = insert(root, points[i]);
    int point1[] = {10, 19};
    (search(root, point1))? cout << "Found\n": cout << "Not Found\n";
    int point2[] = {12, 19};
    (search(root, point2))? cout << "Found\n": cout << "Not Found\n";
    return 0;
}
```

Output:

```
Found
Not Found
```

Time Complexity: O(n)

Auxiliary Space: O(n)

Refer below articles for find minimum and delete operations.

- K D Tree (Find Minimum)
- K D Tree (Delete)

Advantages of K Dimension tree –

K-d trees have several advantages as a data structure:

- **Efficient search**: K-d trees are effective in searching for points in a k-dimensional space, such as in nearest neighbor search or range search.
- **Dimensionality reduction:** K-d trees can be used to reduce the dimensionality of the problem, allowing for faster search times and reducing the memory requirements of the data structure.
- **Versatility:** K-d trees can be used for a wide range of applications, such as in data mining, computer graphics, and scientific computing.
- **Balance:** K-d trees are self-balancing, which ensures that the tree remains efficient even when data is inserted or removed.
- **Incremental construction:** K-d trees can be incrementally constructed, which means that data can be added or removed from the structure without having to rebuild the entire tree.
- **Easy to implement:** K-d trees are relatively easy to implement and can be implemented in a variety of programming languages.

This article is compiled by **Aashish Barnwal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Solve Problems
Solve DSA problems on GfG Practice.

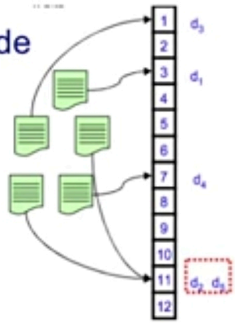
Article Tags : Advanced Data Structure DSA Tree

Recommended Articles

1. m-Way Search Tree | Set-2 | Insertion and Deletion
2. Two Dimensional Binary Indexed Tree or Fenwick Tree
3. Insertion in Binary Search Tree
4. Complexity of different operations in Binary tree, Binary Search Tree and AVL tree
5. Two Dimensional Segment Tree | Sub-Matrix Sum
6. Find minimum in K Dimensional Tree
7. Deletion in K Dimensional Tree
8. Search N elements in an unbalanced Binary Search Tree in O(N * logM) time
9. Insertion in n-ary tree in given order and Level order traversal
10. Proto Van Emde Boas Tree | Set 3 | Insertion and isMember Query
11. Van Emde Boas Tree | Set 2 | Insertion, Find, Minimum and Maximum Queries
12. ScapeGoat Tree | Set 1 (Introduction and Insertion)
13. Insertion in an AVL Tree
14. Insertion in a Binary Tree in level order
15. Optimal sequence for AVL tree insertion (without any rotations)
16. Insertion in Red-Black Tree

Detecting near-duplicates

- Locality-sensitive hashing (LSH)
  - similar documents => similar hash-code
- For each document d:
  - generate K-bit hash-code
  - insert document into hash-table
  - collision => possible duplicate
    - compare to docs in same bucket
- Can miss near-duplicates:
  - similar hash-code ≠ same bucket
  - repeat L times w. different hash-tables (randomised)



Locality-sensitive hashing

In computer science, **locality-sensitive hashing** (**LSH**) is an algorithmic technique that hashes similar input items into the same "buckets" with high probability.[1] (The number of buckets is much smaller than the universe of possible input items.)[1] Since similar items end up in the same buckets, this technique can be used for data clustering and nearest neighbor search. It differs from conventional hashing techniques in that hash collisions are maximized, not minimized. Alternatively, the technique can be seen as a way to reduce the dimensionality of high-dimensional data; high-dimensional input items can be reduced to low-dimensional versions while preserving relative distances between items.

Hashing-based approximate nearest neighbor search algorithms generally use one of two main categories of hashing methods: either data-independent methods, such as locality-sensitive hashing (LSH); or data-dependent methods, such as locality-preserving hashing (LPH).[2][3]

Contents

Definitions

An *LSH family*[1][4][5] $\mathcal{F}$ is defined for

- a metric space $\mathcal{M} = (M, d)$,
- a threshold $R > 0$,
- an approximation factor $c > 1$,
- and probabilities P_1 and P_2.

This family $\mathcal{F}$ is a set of functions $h\colon M \to S$ that map elements of the metric space to buckets $s \in S$. An LSH family must satisfy the following conditions for any two points $p, q \in M$ and any hash function h chosen uniformly at random from $\mathcal{F}$:

- if $d(p, q) \leq R$, then $h(p) = h(q)$ (i.e., p and q collide) with probability at least P_1,
- if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most P_2.

A family is interesting when $P_1 > P_2$. Such a family $\mathcal{F}$ is called (R, cR, P_1, P_2)-*sensitive*.

Alternatively[6] it is defined with respect to a universe of items U that have a similarity function $\phi\colon U \times U \to [0, 1]$. An LSH scheme is a family of hash functions H coupled with a probability distribution D over the functions such that a function $h \in H$ chosen according to D satisfies the property that $Pr_{h \in H}[h(a) = h(b)] = \phi(a, b)$ for any $a, b \in U$.

Locality-preserving hashing

A **locality-preserving hash** is a hash function f that maps points in a metric space $\mathcal{M} = (M, d)$ to a scalar value such that

$$d(p, q) < d(q, r) \Rightarrow |f(p) - f(q)| < |f(q) - f(r)|$$

for any three points $p, q, r \in M$.[*citation needed*]

In other words, these are hash functions where the relative distance between the input values is preserved in the relative distance between the output hash values; input values that are closer to each other will produce output hash values that are closer to each other.

This is in contrast to cryptographic hash functions and checksums, which are designed to have random output difference between adjacent inputs.

The first family of locality-preserving hash functions was devised as a way to facilitate data pipelining in implementations of parallel random-access machine (PRAM) algorithms that use universal hashing to reduce memory contention and network congestion.[7][8]

Locality preserving hashes are related to space-filling curves.[*how?*]

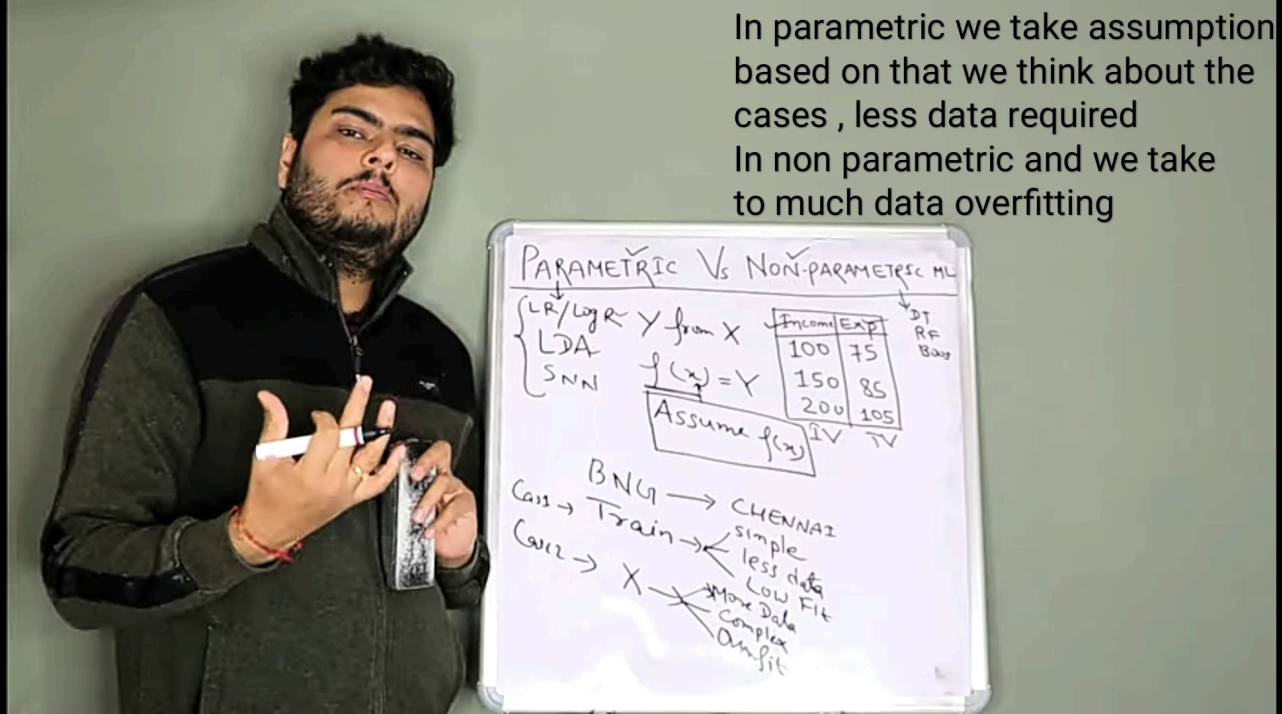
Amplification

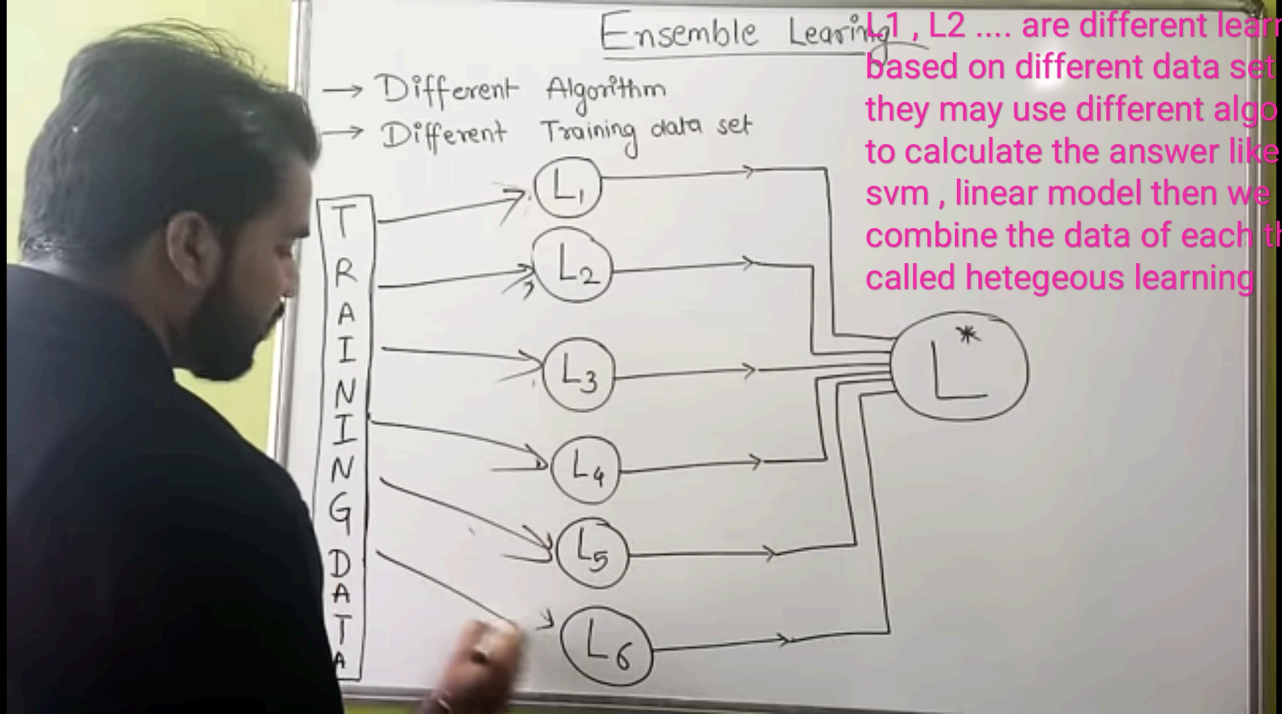
Given a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$, we can construct new families $\mathcal{G}$ by either the AND-construction or OR-construction of $\mathcal{F}$.[1]

To create an AND-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if all $h_i(x) = h_i(y)$ for $i = 1, 2, \ldots, k$. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a (d_1, d_2, p_1^k, p_2^k)-sensitive family.

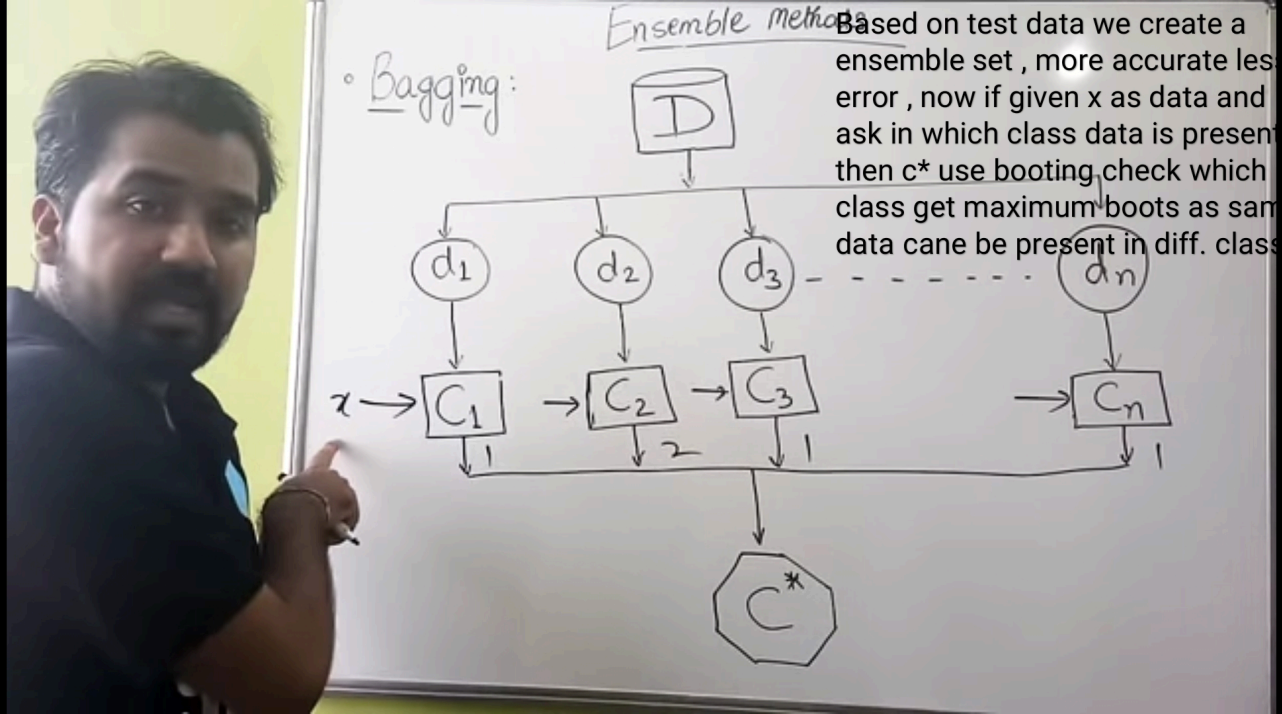
To create an OR-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if $h_i(x) = h_i(y)$ for one or more values of i. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a $(d_1, d_2, 1 - (1 - p_1)^k, 1 - (1 - p_2)^k)$-sensitive family.

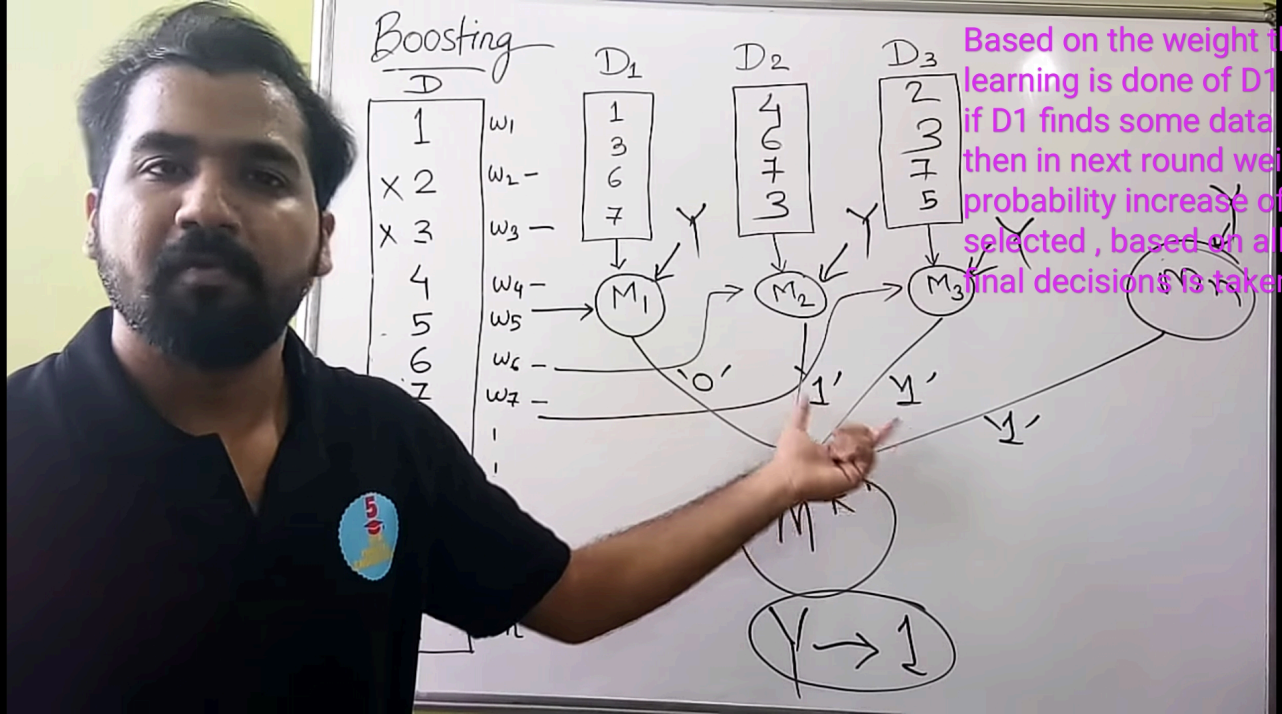
Applications

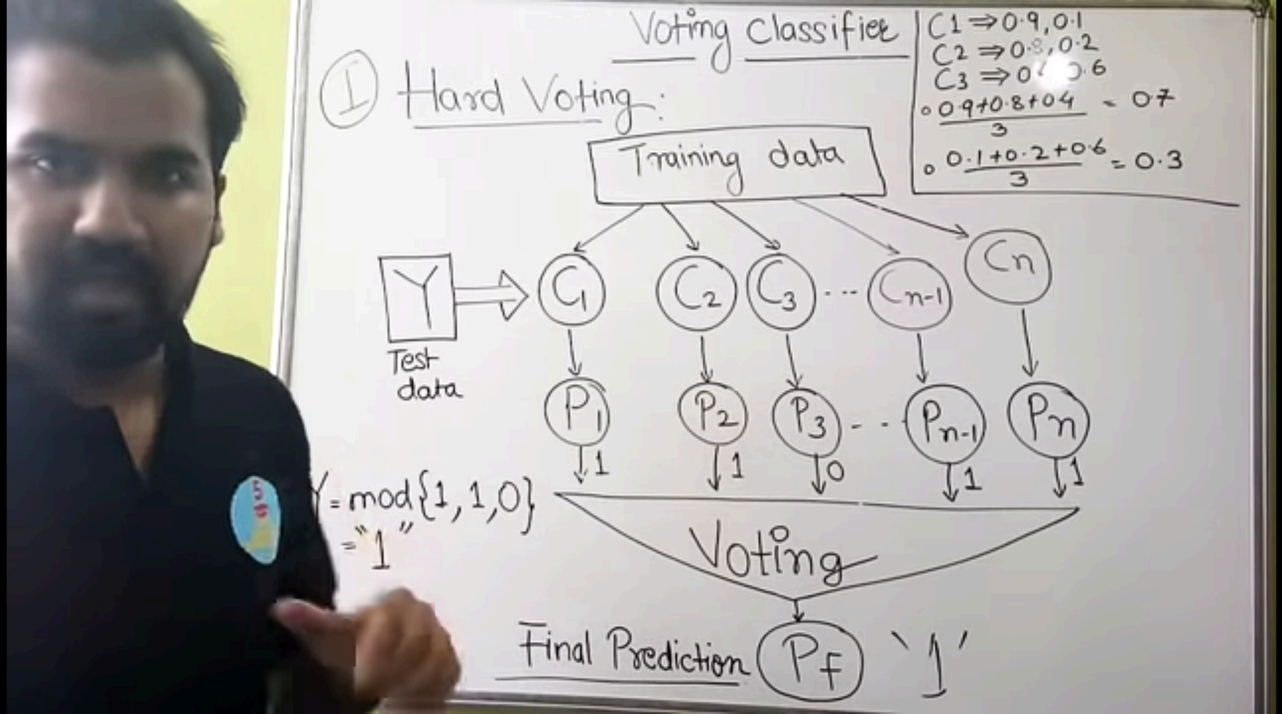

In parametric we take assumption based on that we think about the cases , less data required
In non parametric and we take to much data overfitting
PARAMETRIC Vs NON-PARAMETRIC ML
LR/LogR
LDA
SNN
Y from X
f(x) = Y
Assume f(x)
Income
Exp
100
75
150
85
200
105
IV
IV
DT
RF
BNG → CHENNAI
Case → Train → simple, less data, Low Fit
Case2 → X → More Data, Complex, Overfit

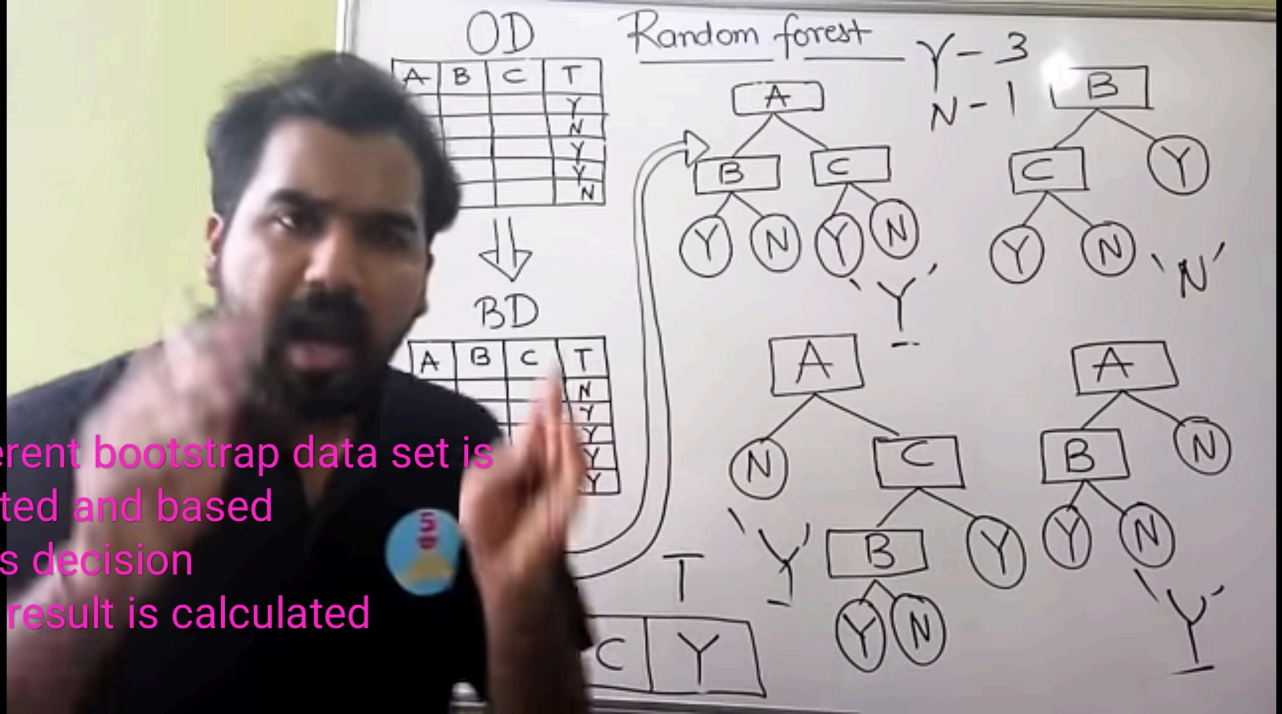


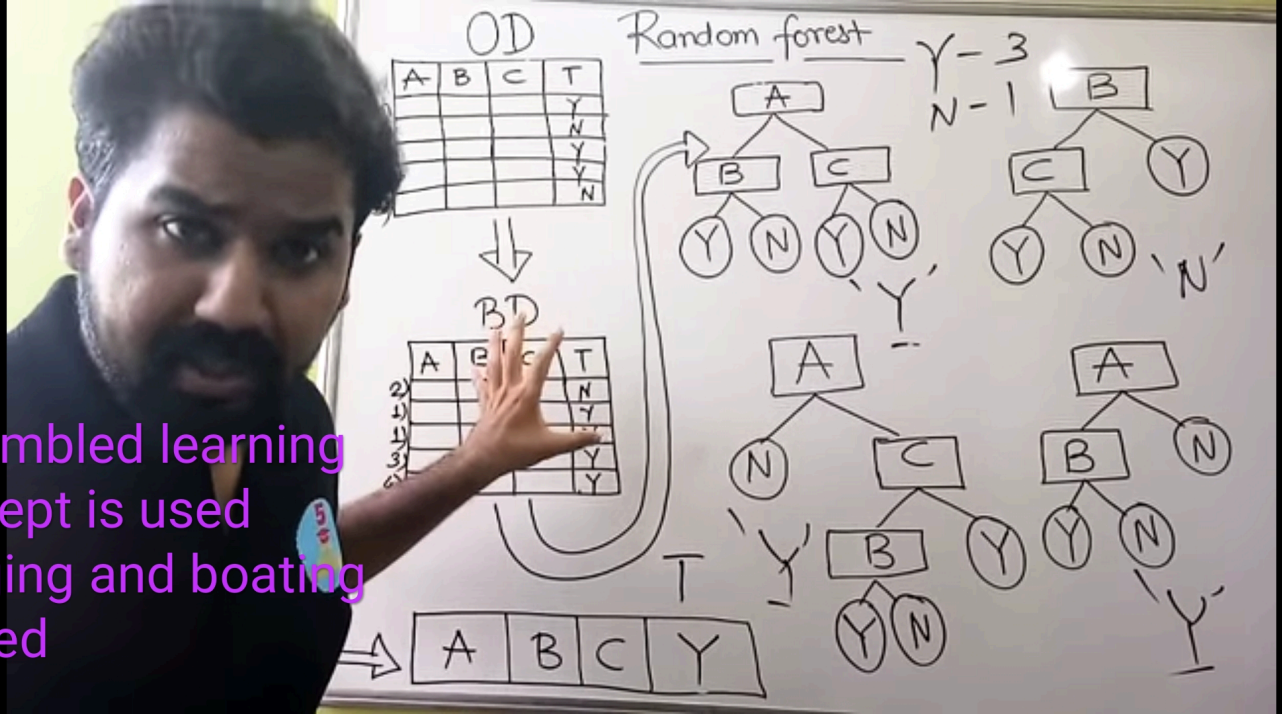
L1 , L2 .... are different learnings based on different data set they may use different algo to calculate the answer like svm , linear model then we combine the data of each this is called hetegeous learning


Ensemble methods
Bagging:
D
d1
d2
d3
dn
x
C1
C2
C3
Cn
1
2
1
1
C*
Based on test data we create a
ensemble set , more accurate les
error , now if given x as data and
ask in which class data is present
then c* use booting check which
class get maximum boots as sam
data cane be present in diff. class


Boosting
D
D1
D2
D3
1
x 2
x 3
4
5
6
7
w1
w2
w3
w4
w5
w6
w7
1 3 6 7
4 6 7 3
2 3 7 5
M1
M2
M3
'0'
'1'
'1'
'1'
Y → 1
Based on the weight the intial learning is done of D1 , D2-- Dn if D1 finds some data wrong then in next round weight / probability increase of geeting selected , based on all the data final decisions is taken


Voting Classifier
C1 ⇒ 0.9, 0.1
C2 ⇒ 0.8, 0.2
C3 ⇒ 0.4, 0.6
0.9+0.8+0.4 / 3 = 0.7
0.1+0.2+0.6 / 3 = 0.3
I Hard Voting:
Training data
Y
Test data
C1
C2
C3
Cn-1
Cn
P1
P2
P3
Pn-1
Pn
1
1
0
1
1
Y = mod{1,1,0}
= "1"
Voting
Final Prediction
Pf
'1'


OD
A B C T
Y
N
Y
Y
N
BD
A B C T
N
Y
Y
Y
Y
Random forest
Y - 3
N - 1
A
B
C
Y
N
Y
N
Y
N
Y
N
Y
N
Y
A
N
C
B
Y
Y
N
Y
A
B
N
Y
N
Y
T
C
Y
Different bootstrap data set is
created and based
on its decision
final result is calculated

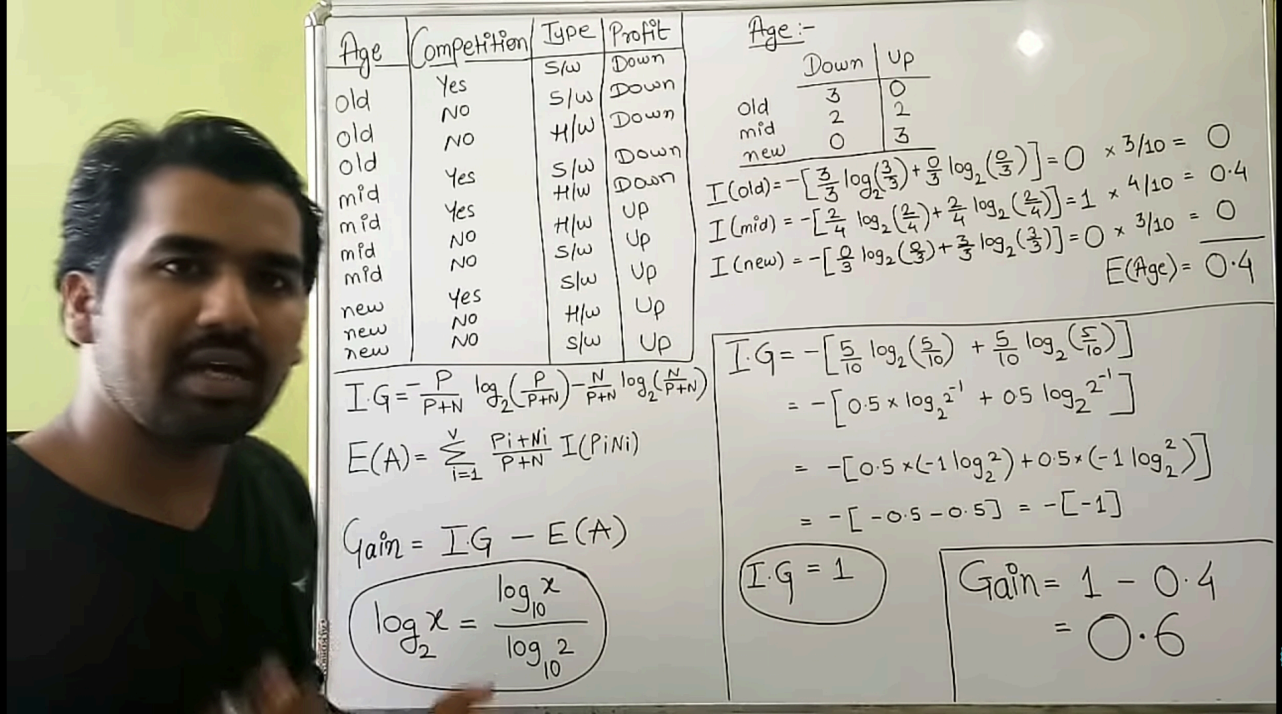


Assembled learning concept is used
Bagging and boating is used

What is Meta-learning?

Meta-learning, described as “learning to learn”, is a subset of machine learning in the field of computer science. It is used to improve the results and performance of the learning algorithm by changing some aspects of the learning algorithm based on the results of the experiment. Meta-learning helps researchers understand which algorithms generate the best/better predictions from datasets. Meta-learning algorithms use learning algorithm metadata as input. They then make predictions and provide information about the performance of these learning algorithms as output. An image’s metadata in a learning model can include, for example, its size, resolution, style, creation date, and owner.

Systematic experiment design in meta-learning is the most important challenge.



Age	Competition	Type	Profit
old	Yes	S/w	Down
old	NO	S/w	Down
old	NO	H/w	Down
mid	Yes	S/w	Down
mid	Yes	H/w	Up
mid	NO	H/w	Up
mid	NO	S/w	Up
new	Yes	S/w	Up
new	NO	H/w	Up
new	NO	S/w	Up

$$IG = -\frac{P}{P+N}\log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N}\log_2\left(\frac{N}{P+N}\right)$$

$$E(A) = \sum_{i=1}^{v} \frac{Pi+Ni}{P+N} I(PiNi)$$

$$Gain = IG - E(A)$$

$$\log_2 x = \frac{\log_{10} x}{\log_{10} 2}$$

Age :-

	Down	Up
old	3	0
mid	2	2
new	0	3

$$I(old) = -\left[\frac{3}{3}\log_2\left(\frac{3}{3}\right) + \frac{0}{3}\log_2\left(\frac{0}{3}\right)\right] = 0 \times 3/10 = 0$$

$$I(mid) = -\left[\frac{2}{4}\log_2\left(\frac{2}{4}\right) + \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right] = 1 \times 4/10 = 0.4$$

$$I(new) = -\left[\frac{0}{3}\log_2\left(\frac{0}{3}\right) + \frac{3}{3}\log_2\left(\frac{3}{3}\right)\right] = 0 \times 3/10 = 0$$

$$E(Age) = 0.4$$

$$IG = -\left[\frac{5}{10}\log_2\left(\frac{5}{10}\right) + \frac{5}{10}\log_2\left(\frac{5}{10}\right)\right]$$

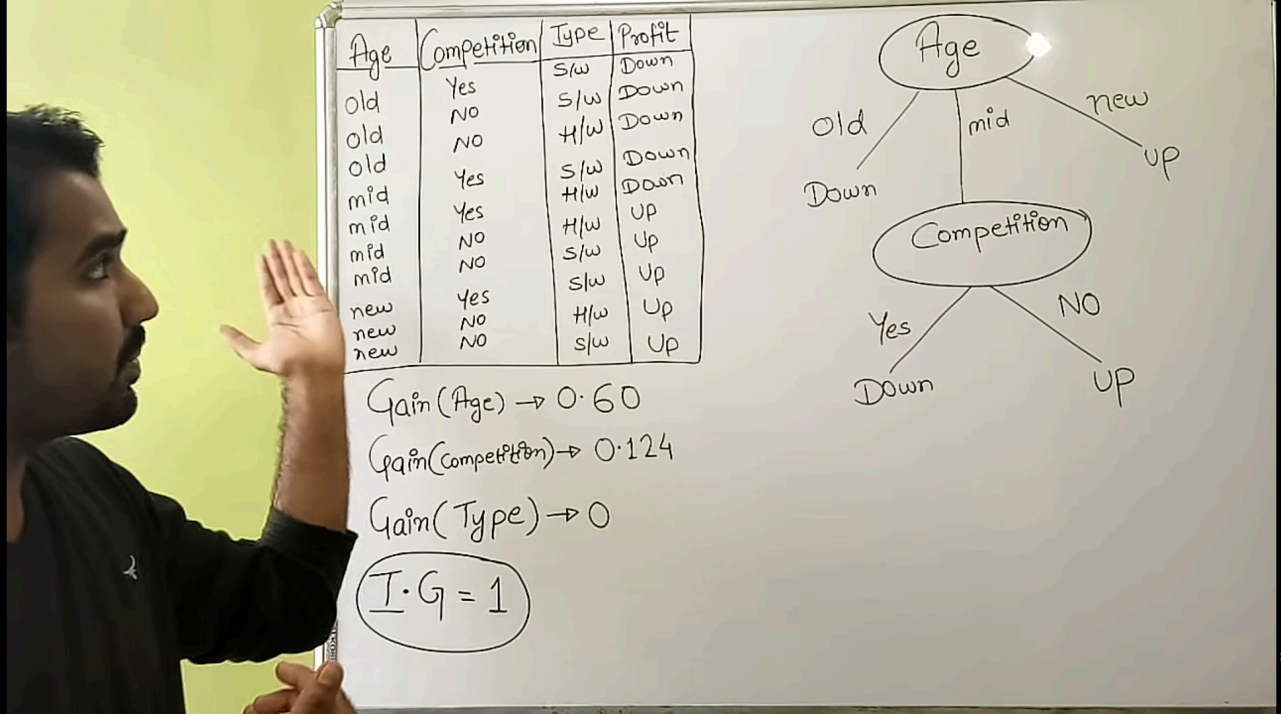
$$= -\left[0.5 \times \log_2 2^{-1} + 0.5\log_2 2^{-1}\right]$$

$$= -\left[0.5 \times (-1\log_2 2) + 0.5 \times (-1\log_2 2)\right]$$

$$= -[-0.5 - 0.5] = -[-1]$$

$$I.G = 1$$

$$Gain = 1 - 0.4 = 0.6$$



Age	Competition	Type	Profit
old	Yes	S/w	Down
old	NO	S/w	Down
old	NO	H/w	Down
mid	Yes	S/w	Down
mid	Yes	H/w	Down
mid	NO	H/w	Up
mid	NO	S/w	Up
new	Yes	S/w	Up
new	NO	H/w	Up
new	NO	S/w	Up

Gain (Age) → 0.60

Gain (Competition) → 0.124

Gain (Type) → 0

I.G = 1

Reinforcement learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

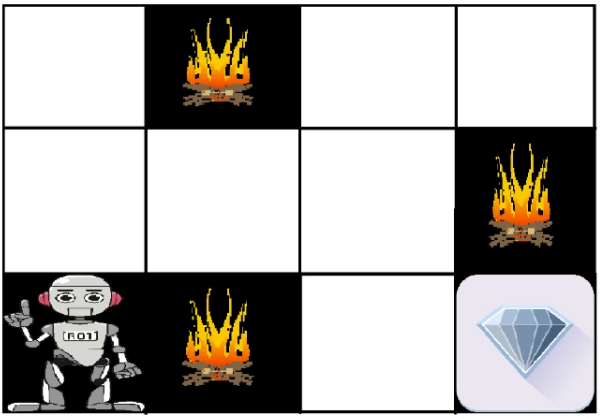
Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Example:

The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions	In Supervised learning, the decision is

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	Example: Object recognition,spam detetction

Types of Reinforcement:

There are two types of Reinforcement:

1. **Positive:** Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
   Advantages of reinforcement learning are:
   - Maximizes Performance
   - Sustain Change for a long period of time
   - Too much Reinforcement can lead to an overload of states which can diminish the results
2. **Negative:** Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.
   Advantages of reinforcement learning:
   - Increases Behavior
   - Provide defiance to a minimum standard of performance
   - It Only provides enough to meet up the minimum behavior

Elements of Reinforcement Learning

Reinforcement learning elements are as follows:

1. Policy
2. Reward function
3. Value function
4. Model of the environment

Policy: Policy defines the learning agent behavior for given time period. It is a mapping from perceived states of the environment to actions to be taken when in those states.

Reward function: Reward function is used to define a goal in a reinforcement learning problem.A reward function is a function that provides a numerical score based on the state of the environment

Value function: Value functions specify what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Model of the environment: Models are used for planning.

⚫ **Credit assignment problem:** Reinforcement learning algorithms learn to generate an internal value for the intermediate states as to how good they are in leading to the goal. The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent.

The agent has sensors to decide on its state in the environment and takes action that modifies its state.

⚫ The reinforcement learning problem model is an agent continuously interacting with an environment. The agent and the environment interact in a sequence of time steps. At each time step t, the agent receives the state of the environment and a scalar numerical reward for the previous action, and then the agent then selects an action.

Reinforcement learning is a technique for solving Markov decision problems.

⚫ Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem.

Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials

to be a simple way of representing essential features of the artificial intelligence problem.

Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

Application of Reinforcement Learnings

1. Robotics: Robots with pre-programmed behavior are useful in structured environments, such as the assembly line of an automobile manufacturing plant, where the task is repetitive in nature.

2. A master chess player makes a move. The choice is informed both by planning, anticipating possible replies and counter replies.

3. An adaptive controller adjusts parameters of a petroleum refinery's operation in real time.

RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization)
3. The only way to collect information about the environment is to interact with it.

Advantages and Disadvantages of Reinforcement Learning

Advantages of Reinforcement learning

1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.

2. The model can correct the errors that occurred during the training process.

3. In RL, training data is obtained via the direct interaction of the agent with the environment

4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.

5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.

6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems.

2. Reinforcement learning needs a lot of data and a lot of computation

3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.

4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

Implementation:

Python3

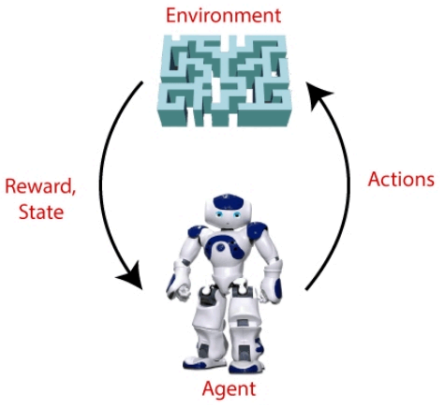
```
import gym
import numpy as np

# Define the Q-table and learning rate
q_table = np.zeros((state_size, action_size))
alpha = 0.8
gamma = 0.95

# Train the Q-Learning algorithm
for episode in range(num_episodes):
    state = env.reset()
    done = False
    while not done:
        # Choose an action
        action = np.argmax(
            q_table[state, :] + np.random.randn(1, action_size) * (1. / (epis

        # Take the action and observe the new state and reward
        next_state, reward, done, _ = env.step(action)

        # Update the Q-table
        q_table[state, action] = (1 - alpha) * q_table[state, action] + \
            alpha * (reward + gamma * np.max(q_table[next_state, :]))
```

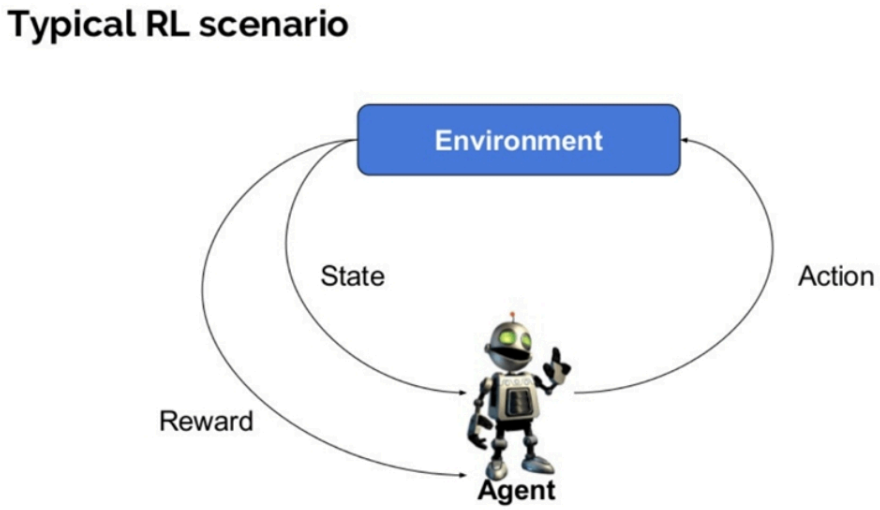


Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement

Important Components of Deep Reinforcement Learning Method



Here are some important terms used in Reinforcement AI:

- **Agent:** It is an assumed entity which performs actions in an environment to gain some reward.
- **Environment (e):** A scenario that an agent has to face.
- **Reward (R):** An immediate return given to an agent when he or she performs specific action or task.
- **State (s):** State refers to the current situation returned by the environment.
- **Policy (π):** It is a strategy which applies by the agent to decide the

- **Policy (π):** It is a strategy which applies by the agent to decide the next action based on the current state.
- **Value (V):** It is expected long-term return with discount, as compared to the short-term reward.
- **Value Function:** It specifies the value of a state that is the total amount of reward. It is an agent which should be expected beginning from that state.
- **Model of the environment:** This mimics the behavior of the environment. It helps you to make inferences to be made and also determine how the environment will behave.
- **Model based methods:** It is a method for solving reinforcement learning problems which use model-based methods.
- **Q value or action value (Q):** Q value is quite similar to value. The only difference between the two is that it takes an additional parameter as a current action.

How Reinforcement Learning

Regular Expression

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

For instance:

In a regular expression, x* means zero or more occurrence of x. It can generate {e, x, xx, xxx, xxxx, ...}

In a regular expression, x^+ means one or more occurrence of x. It can generate {x, xx, xxx, xxxx, ...}

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union L U M is also a

NOM → Noun
NOM → Noun NOM
VP → Verb
VP → Verb NP

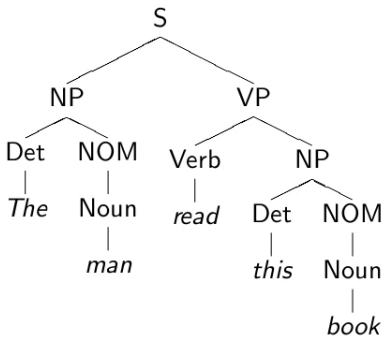
}

Application of grammar rewrite rules

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

Parse tree

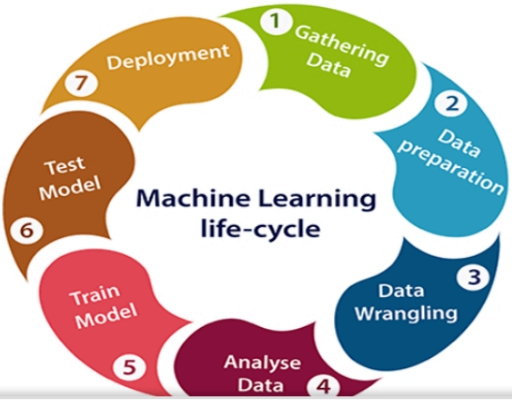


Machine learning Life cycle

Machine learning has given the computer systems the abilities to automatically learn without being explicitly programmed. But how does a machine learning system work? So, it can be described using the life cycle of machine learning. Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project.

Machine learning life cycle involves seven major steps, which are given below:

- **Gathering Data**
- **Data preparation**
- **Data Wrangling**
- **Analyse Data**
- **Train the model**
- **Test the model**
- **Deployment**



The most important thing in the complete process is to understand the problem and to know the purpose of the problem. Therefore, before starting the life cycle, we need to understand the problem because the good result depends on the better understanding of the problem.

In the complete life cycle process, to solve a problem, we create a machine learning system called "model", and this model is created by providing "training". But to train a model, we need data, hence, life cycle starts by collecting data.

1. Gathering Data:

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.

In this step, we need to identify the different data sources, as data can be collected from various sources such as **files**, **database**, **internet**, or **mobile devices**. It is one of the most important steps of the

1. Gathering Data:

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.

In this step, we need to identify the different data sources, as data can be collected from various sources such as **files**, **database**, **internet**, or **mobile devices**. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.

This step includes the below tasks:

- **Identify various data sources**
- **Collect data**
- **Integrate the data obtained from different sources**

By performing the above task, we get a coherent set of data, also called as a **dataset**. It will be used in further steps.

---

2. Data preparation

After collecting the data, we need to

2. Data preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data.

This step can be further divided into two processes:

- **Data exploration:**
  It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data.
  A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.
- **Data pre-processing:**
  Now the next step is preprocessing of data for its analysis.

3. Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to

Machine Learning Steps

The task of imparting intelligence to machines seems daunting and impossible. But it is actually really easy. It can be broken down into 7 major steps :

1. Collecting Data:

As you know, machines initially learn from the data that you give them. It is of the utmost importance to collect reliable data so that your machine learning model can find the correct patterns. The quality of the data that you feed to the machine will determine how accurate your model is. If you have incorrect or outdated data, you will have wrong outcomes or predictions which

Machine learning models, such as supervised learning algorithms, can be more efficient compared to feature-based classification techniques for sorting fruits based on type in a dealer's warehouse. Here are some reasons to justify this:

1. Automation: Machine learning models automate the process of fruit sorting, eliminating the need for manual feature engineering and classification rule creation. Instead of manually defining rules based on individual features like size, color, and texture, the model can learn and extract relevant features on its own, reducing human effort and potential biases.
2. Adaptability: Machine learning models can adapt and generalize well to new and unseen data. As the dealer's warehouse may receive different types of fruits over time, the model can learn from the historical data and adjust its classification boundaries accordingly. It can handle variations in fruit characteristics, account for seasonal changes, and accommodate new fruit types without requiring explicit modification of feature-based rules.
3. Handling complex patterns: Machine learning models can identify complex patterns and relationships among multiple features that may not be obvious to human observers. Fruits can exhibit subtle variations in color, size, shape, and texture, and these features may interact in intricate ways to determine the fruit type. Machine learning models can automatically learn these patterns and make accurate predictions based on the learned relationships.
4. Scalability: Machine learning models can handle large amounts of data efficiently. In a warehouse setting, there may be thousands or even millions of fruit samples to process and classify. Feature-based classification techniques may struggle to handle such large datasets, as they require manual feature selection and rule creation for each fruit type. In contrast, machine learning models can handle high-dimensional data and scale well with increased data size.
5. Continuous improvement: Machine learning models can continuously improve their performance over time. By receiving feedback on their predictions and retraining periodically, the models can refine their classification accuracy and adapt to changes in fruit characteristics. This ongoing learning process can lead to better sorting accuracy and efficiency in the long run.

Overall, machine learning models offer advantages of automation, adaptability, pattern detection, scalability, and continuous improvement, making them more efficient compared to feature-based classification techniques for sorting fruits in a dealer's warehouse.

8. How to Choose Between DFS and BFS?

All this said, the actual question we're interested in is how to choose between DFS and BFS.

If we know that the goal is deep, we should use DFS because it reaches deep nodes faster than BFS. A good example is constraint satisfaction. In those problems, we have to assign values to several variables such that the constraints on the variables and their values are satisfied. Each node in the resulting search graph is a list of assignments. Since we're interested in complete solutions, it makes more sense to reach the full assignment nodes as fast as possible with DFS than to slowly search through incomplete assignments with BFS.

On the other hand, if we know that the goal may appear at a shallow level in the search tree, BFS is a better choice. The same goes if we

8. How to Choose Between DFS and BFS?

All this said, the actual question we're interested in is how to choose between DFS and BFS.

If we know that the goal is deep, we should use DFS because it reaches deep nodes faster than BFS. A good example is constraint satisfaction. In those problems, we have to assign values to several variables such that the constraints on the variables and their values are satisfied. Each node in the resulting search graph is a list of assignments. Since we're interested in complete solutions, it makes more sense to reach the full assignment nodes as fast as possible with DFS than to slowly search through incomplete assignments with BFS.

On the other hand, if we know that the goal may appear at a shallow level in the search tree, BFS is a better choice. The same goes if we cannot accept a sub-optimal path and are ready to spend more memory to

On the other hand, if we know that the goal may appear at a shallow level in the search tree, BFS is a better choice. The same goes if we cannot accept a sub-optimal path and are ready to spend more memory to get the theoretical guarantees. But, the caveat is that BFS can require a lot of memory and will prove inefficient if the branching factor is too large or the goal is deep.

It's precisely the memory complexity that makes BFS impractical even though it's complete and optimal in the cases where DFS isn't. For that reason, DFS has become the working horse of AI. We can even sort out the problems of its incompleteness and sub-optimal solutions. Iterative deepening, which limits the depth of DFS and runs it with incremental limiting depths, is complete, optimal, and of the same time complexity as BFS. However, its memory complexity is the same as that of DFS.

8. Conclusion

Rule-based Classifier

A technique for collecting records using a collection of IF ... THEN .. (rules) → disjunctive NF

$R = (r_1 \vee r_2 \vee \cdots r_K)$

R → Ruleset, disjuncts or classification rules.

r_1 : (GB = N) ∧ (Aerial creature = Y) → Birds
r_2 : (GB = N) ∧ (Aq. creature = Y) → Fish
r_3 : (GB = Y) ∧ (BT = WB) → Mammals
r_4 : (GB = N) ∧ (Aerial creature = N) → Reptiles
r_5 : (Aq. creature = Semi) → Amphibians

r_i : (Condition$_i$) → y_i

rule antecedent / precondition

Ai (op) vi

< <= > >= = ≠ comparisons.

y_i → rule consequent.

① Coverage(r) = fraction of rec D. which triggers rule r

$= \frac{|A|}{|D|}$ → rule antecedent

② Accuracy / Confidence factor(r) = frac of rec. in D. which is triggered by rule which has same class label as y

Dataset : Vertebrate

Name	BT	SC	GB	Aq. Creature	Aerial Creature	Legs	Hibernates
lemur	WB	fur	Y	N	N	Y	Y
Turtle	CB	scales	N	Semi	N	Y	N
Dogfish shark	CB	scales	Y	Y	N	N	N

Solution to make the rule set exhaustive:

- Use a default class

Ordered Rule Set

An ordered rule set is known as a decision list. Rules are rank ordered according to their **priority**. For example, when a test record is presented to the classifier, it is assigned to the class label of the highest ranked rule it has triggered. If none of the rules fired, it is assigned to the default class.

That is, if more than one rule is triggered, need **conflict resolution**:

- **Size ordering** - assign the highest priority to the triggering rules that has the “toughest” requirement (i.e., with the most attribute test)
- **Class-based ordering** - decreasing order of prevalence or misclassification cost per class
- **Rule-based ordering** (decision list) - rules are organized into one long priority list, according to some measure of **rule quality** or by experts

Building Rules Through Direct Method

Direct Method extract rules directly from data. **Sequential Covering** such as CN2 Algorithm and RIPPER Algorithm are common direct methods for building classification rules.

Take Ripper method as example. For **2-class**

Ordered Rule Set

An ordered rule set is known as a decision list. Rules are rank ordered according to their **priority**. For example, when a test record is presented to the classifier, it is assigned to the class label of the highest ranked rule it has triggered. If none of the rules fired, it is assigned to the default class.

That is, if more than one rule is triggered, need **conflict resolution**:

- **Size ordering** - assign the highest priority to the triggering rules that has the “toughest” requirement (i.e., with the most attribute test)
- **Class-based ordering** - decreasing order of prevalence or misclassification cost per class
- **Rule-based ordering** (decision list) - rules are organized into one long priority list, according to some measure of **rule quality** or by experts

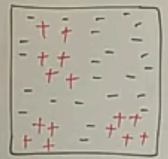
Building Rules Through Direct Method

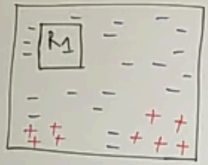
Direct Method extract rules directly from data. **Sequential Covering** such as CN2 Algorithm and RIPPER Algorithm are common direct methods for building classification rules.

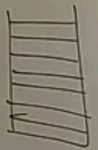
Take Ripper method as example. For **2-class problem**, choose one of the classes as positive class, and the other as negative class, learn rules for positive class, and negative class will be

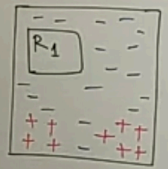
Sequential Covering Algorithm

- Extract rules directly from Dataset
- Rules grown in Greedy way based on certain evaluation measure (accuracy/conf factor, coverage)
- Extract one rule at a time for 2 classes
- Which class to be generated depends upon class prevalence or cost of misclassifying records from a given class









$r_i : (\text{Condition}_i) \rightarrow y_i$

ALGORITHM :- Ordered rules

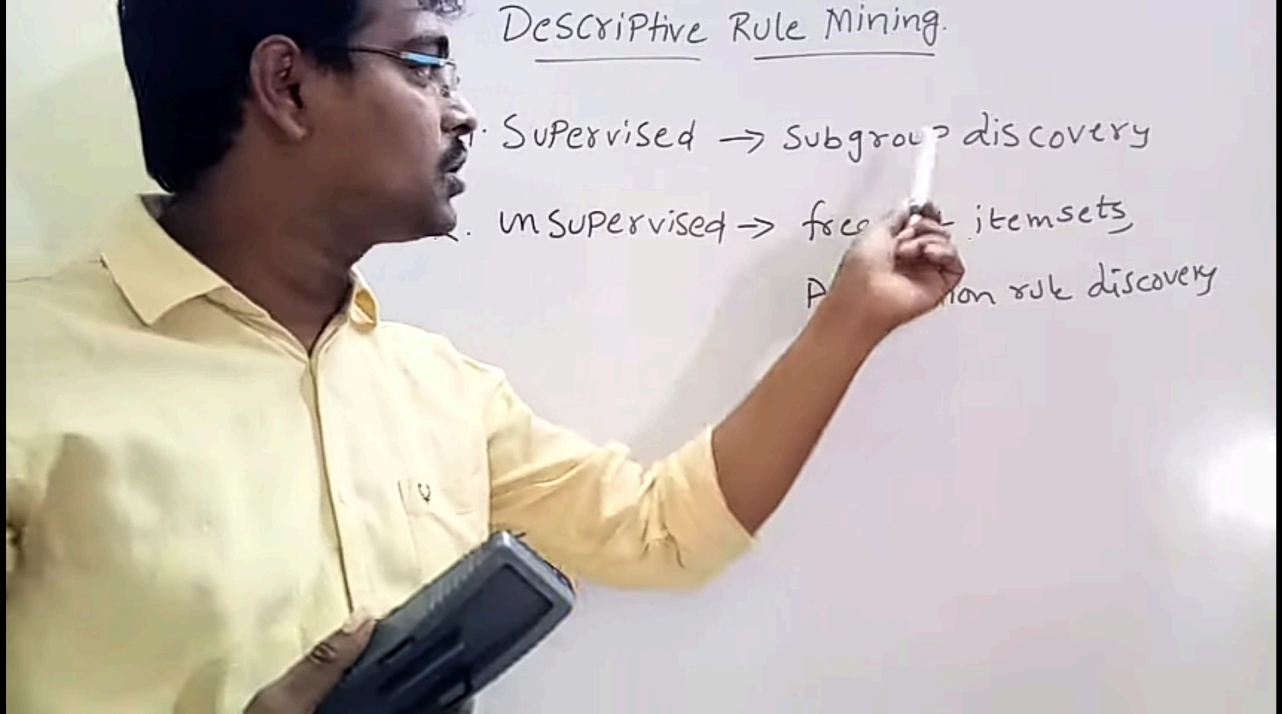
S1: Create empty decision list, 'R'

S2: Learn-One-Rule function → extract best rule for class 'y'
All train rec ∈ class 'y' ⇒ +
train rec ∉ class 'y' ⇒ −

S3: Rule : Desirable when it covers most of + obs.

S4: Find such a rule then eliminate training records

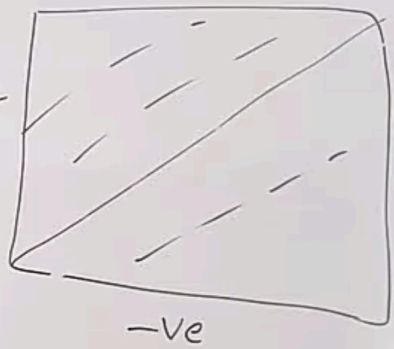
S5: New rule added to bottom of 'R'


Descriptive Rule Mining.
1. Supervised → Subgroup discovery
2. Unsupervised → frequent itemsets
Association rule discovery

Rule learing for Subgroup discovery

1. Precision:

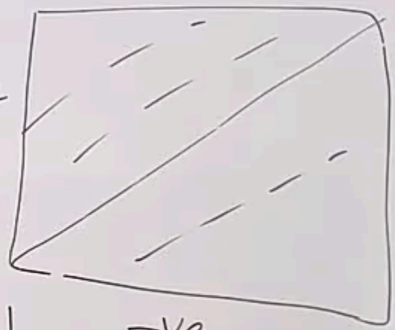
$|Prec - Pos|$



e learing for Subgroup discovery

ecision:

| Prec - POS |

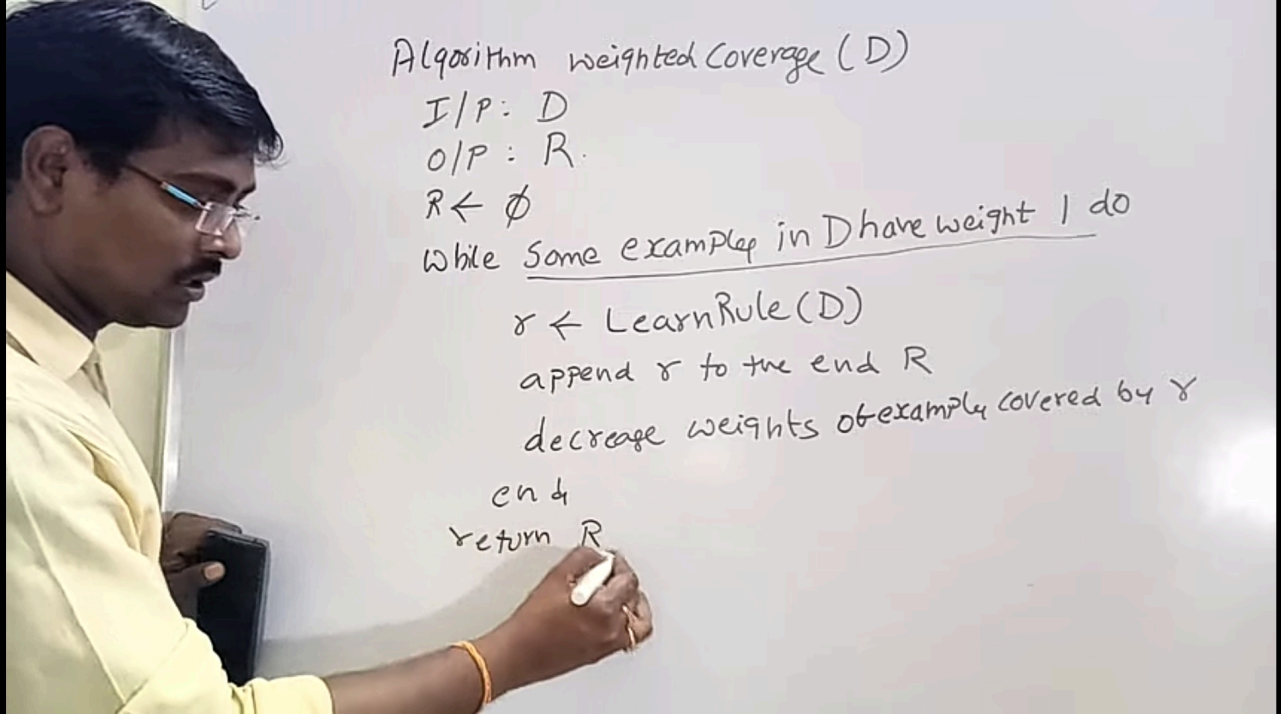


verage - recall (0.5)

| avg rec − 0.5 |

ghted relative accuracy

= POS · Neg. (tpr − fpr)

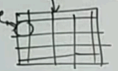

Algorithm weighted Coverage (D)
I/P: D
O/P: R.
R ← ∅
While Some examples in D have weight 1 do
r ← LearnRule(D)
append r to the end R
decrease weights of examples covered by r
end
return R

item: any obj / ele.

Itemset: 0^* items → ϕ, {}, null

Frequent itemset: Set of items occuring frequently coz 'interesting relationship'

Market Basket Analysis



rows → txns } assymetric
items → Columns } binary variable

Key challenges: 10 → 10K 100k. 50%. 60%.

① Discover patterns → interestingness measure.

② Misleading rules: Spurious info.

Txn width: no. of items ∈

Support Count: Set of itemset frequently occ...

$$\sigma(X) = |t_i| X \subseteq t_i, t_i \in T$$

$$\sigma(X)$$

ASSOCIation Rules : set of interesting rules from freq items.

interestingness measure.

$A \longrightarrow C$ Consequent.

Antecedent

$A \cap C = \phi$

Bin { Be, Di } Cola.
1 1 0 0

$$\text{Support}(X \to Y) = \frac{\sigma(X \cup Y)}{N}$$

eliminate uninteresting rules.

6 Rules
4 rules
2 rules.

Confidence: ✓ inference of a rule & its reliability.

Conditional prob Y given X $P(Y|X)$

$$\text{Confidence}(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

minsup = 0.3
minconf = 0.5

Support(X) ≥ minsup

$d \; 2^d - 3^{d+1} + 1$

...txn. ① Reduce the gen. candidate items
② Min the comparisons

item: any obj / ele.

Itemset: 0^* items → ϕ, {}, null

Frequent itemset: Set of items occuring frequency coz 'interesting relationship'

Market Basket Analysis: tuple

attributes ≥ 10

101

rows → txns
items → Colums
} assymetric binary variable

Key challenges: 10 → 10K 100k. 50%. 60%.

① Discover patters → interestingness measure.

② Misleading rules: Spurious info.

Txn width: no. of items ∈

Support Count: Set of itemset frequently occuring in a txn.

$$\sigma(X) = |t_i| X \subseteq t_i, t_i \in T|$$

$$\frac{\sigma(X)}{N} = \text{Support}(X)$$ itemset.

Association Rules: set of interesting rules from freq items.

interestingness measure.

Machine Learning | Laplace Correction

Laplace Correction

Conditional prob Y given X P(Y|X)

$$\text{Confidence}(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

minsup = 0.3
minconf = 0.5

Support(X) ≥ minsup

$d \; 2^d - 3^{d+1} + 1$

Strong rules

① Reduce the gen. Candidate items

② Min the comparisons

≥ minconf.

Association Rule Learning

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are

Representing Simple Facts in FOPL: → models world in terms of objects.

Real-world facts can be represented as logical propositions written as well-formed formulas in propositional logic.

Symbols:-
1. Objects
2. Properties
3. Relation.

} Representation.

Symbols are formed of the following:-

1. Set of all uppercase english Alphabets.
2. Set of digits from 0 to 9.
3. Underscore character.

[All dogs are brown] Can't be written in propositional logic
↳ first-order predicate logic.

Defining a Sentence :-

Every atomic sentence is a sentence.
↳ is defined as predicate constant of arity 'n', followed by $t_1, t_2, \ldots t_n$ terms enclosed in parentheses and separated by commas.

↳ (1) if 'S' is sentence then $\neg S$ is sentence.

(2) if S1, S2 are sentences] $S1 \wedge S2$ (Conjuction)

(3) " "] $S1 \vee S2$ (disjunction)

(4) " "] $S1 \rightarrow S2$ (implication)

(5) " "] $S1 \equiv S2$ (equivalence)

(6) if x is Var. and s is Sentence the $\forall x$ s is a Sentence.

(7) " " $\rightarrow \exists x$ s is a Sentence.

Quantifiers in Predicate Calculus:-

There are two Quantifiers used in first-order predicate calculus:-

1. Universal] for all 'x' such that — $\forall x$ (→)
2. Existensial] for some 'x' such that — $\exists x$

↳ Constrain the meaning of sentence containing a Variable.

↳ Quantifier is followed by a Variable and a Sentence.

① All Boys like football. } → $\forall x : Boys(x) \rightarrow Like(x, football)$

② Some Boys like football. } → $\exists x : Boys(x) \wedge like(x, footba$

Quantifiers in Predicate Calculus:-

There are two Quantifiers used in first-order predicate calculus:-

1. Universal] for all 'x' such that — $\forall x$ (→)
2. Existensial] for some 'x' such that — $\exists x$

↳ Constrain the meaning of sentence containing a Variable.

↳ Quantifier is followed by a Variable and a Sentence.

① All Boys like football. } → $\forall x$: Boys(x) → like(x, football)

② Some Boys like football. } → $\exists x$: Boys(x) ∧ like(x, football)

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Example:

The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.

The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	Example: Object recognition,spam detetction

Types of Reinforcement:

There are two types of Reinforcement:

1. **Positive:** Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
   Advantages of reinforcement learning are:
   - Maximizes Performance
   - Sustain Change for a long period of time
   - Too much Reinforcement can lead to an overload of states which can diminish the results
2. **Negative:** Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.
   Advantages of reinforcement learning:
   - Increases Behavior
   - Provide defiance to a minimum standard of performance
   - It Only provides enough to meet up the minimum behavior

Elements of Reinforcement Learning

Reinforcement learning elements are as follows:

1. Policy
2. Reward function
3. Value function
4. Model of the environment

Policy: Policy defines the learning agent behavior for given time period. It is a mapping from perceived states of the environment to actions to be taken when in those states.

Reward function: Reward function is used to define a goal in a reinforcement learning problem.A reward function is a function that provides a numerical score based on the state of the environment

Value function: Value functions specify what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Model of the environment: Models are used for planning.

● **Credit assignment problem:** Reinforcement learning algorithms learn to generate an internal value for the intermediate states as to how good they are in leading to the goal. The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent.

The agent has sensors to decide on its state in the environment and takes action that modifies its state.

● The reinforcement learning problem model is an agent continuously interacting with an environment. The agent and the environment interact in a sequence of time steps. At each time step t, the agent receives the state of the environment and a scalar numerical reward for the previous action, and then the agent then selects an action.

Reinforcement learning is a technique for solving Markov decision problems.

● Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem.

Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

Application of Reinforcement Learnings

1. Robotics: Robots with pre-programmed behavior are useful in structured environments, such as the assembly line of an automobile manufacturing plant, where the task is repetitive in nature.

2. A master chess player makes a move. The choice is informed both by planning, anticipating possible replies and counter replies.

3. An adaptive controller adjusts parameters of a petroleum refinery's operation in real time.

RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization)
3. The only way to collect information about the environment is to interact with it.

Advantages and Disadvantages of Reinforcement Learning

Advantages of Reinforcement learning

1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.

2. The model can correct the errors that occurred during the training process.

3. In RL, training data is obtained via the direct interaction of the agent with the environment

4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.

5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.

6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems

Advantages and Disadvantages of Reinforcement Learning

Advantages of Reinforcement learning

1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.

2. The model can correct the errors that occurred during the training process.

3. In RL, training data is obtained via the direct interaction of the agent with the environment

4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.

5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.

6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems.

2. Reinforcement learning needs a lot of data and a lot of computation

3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.

4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

Implementation:

Python3

```
import gym
import numpy as np

# Define the Q-table and learning rate
q_table = np.zeros((state_size, action_size))
alpha = 0.8
gamma = 0.95

# Train the Q-Learning algorithm
for episode in range(num_episodes):
    state = env.reset()
    done = False
    while not done:
        # Choose an action
        action = np.argmax(
            q_table[state, :] + np.random.randn(1, action_size) * (1. / (episode + 1

        # Take the action and observe the new state and reward
        next_state, reward, done, _ = env.step(action)

        # Update the Q-table
        q_table[state, action] = (1 - alpha) * q_table[state, action] + \
            alpha * (reward + gamma * np.max(q_table[next_state, :]))

        state = next_state

# Test the trained Q-Learning algorithm
state = env.reset()
done = False
while not done:
    # Choose an action
    action = np.argmax(q_table[state, :])

    # Take the action
    state, reward, done, _ = env.step(action)
    env.render()
```

Last Updated : 18 Apr, 2023

Similar Reads

Markov Decision Process

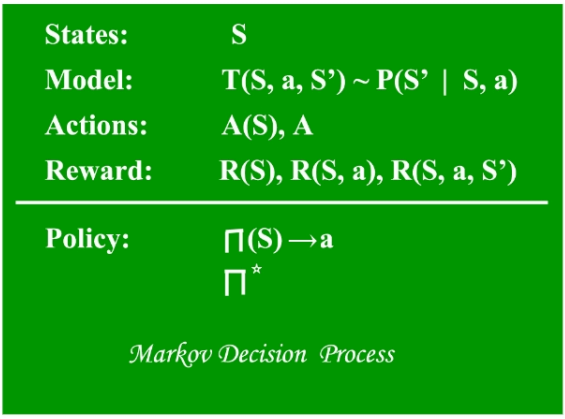
Reinforcement Learning :

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**.



What is a State?

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

What is a 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.

What are 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.

What is a Reward?

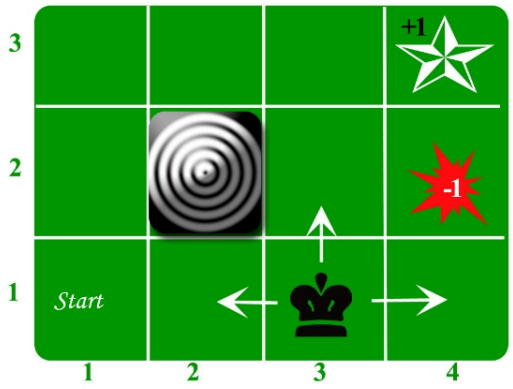
A **Reward** is a real-valued reward function. R(s) indicates the reward for simply being in the state S. R(S,a) indicates the reward for being in a state S and taking an action 'a'. R(S,a,S') indicates the reward for being in a state S, taking an action 'a' and ending up in a state S'.

What is a 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'.

What is a 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:-

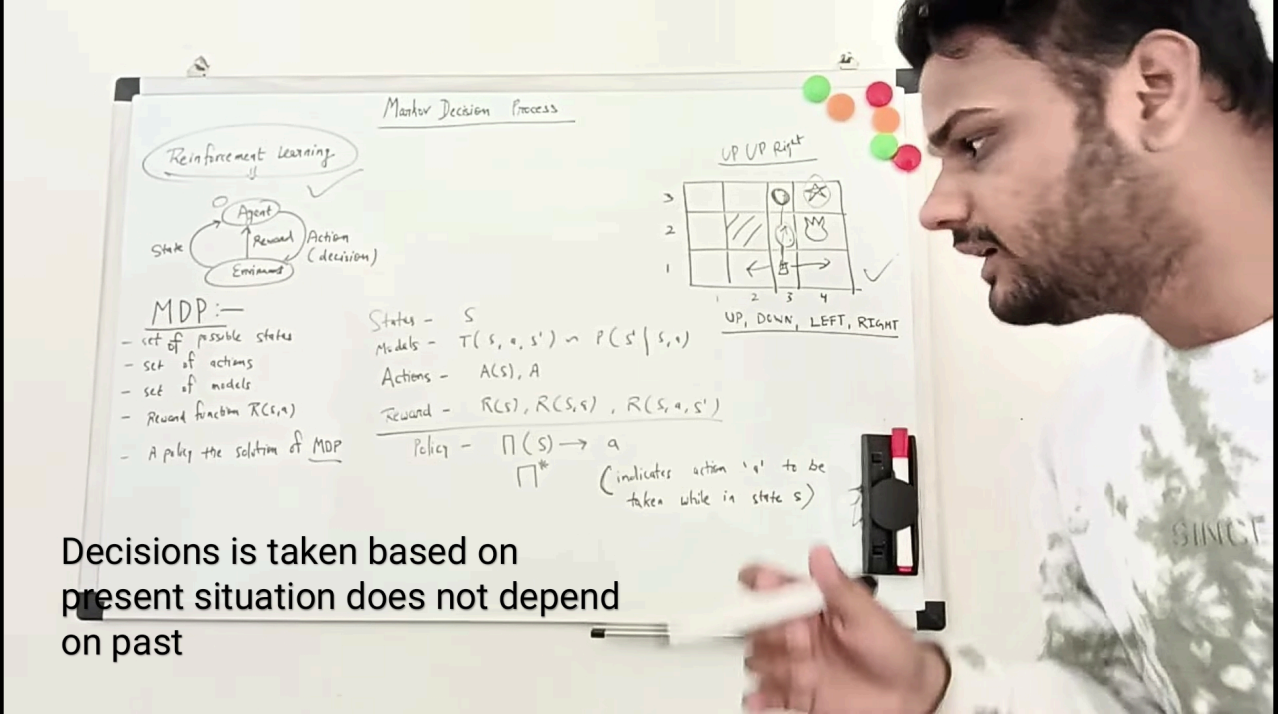
- Small reward each step (can be negative when can also be term as punishment, in the above example entering the Fire can have a reward of -1).
- Big rewards come at the end (good or bad).
- The goal is to Maximize the sum of rewards.

References: http://reinforcementlearning.ai-depot.com/
http://artint.info/html/ArtInt_224.html

Article Tags : Computer Subject | Machine Learning

Recommended Articles

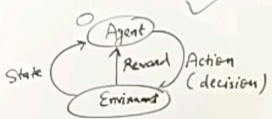
1. Markov Chain
2. Hidden Markov Model in Machine learning
3. Decision Threshold In Machine Learning
4. ML | Logistic Regression v/s Decision Tree Classification
5. Weighted Product Method - Multi Criteria Decision Making
6. Decision Tree Classifiers in R Programming



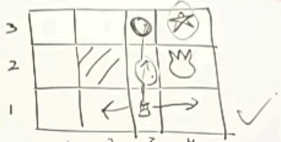
Decisions is taken based on present situation does not depend on past

Markov Decision Process

Reinforcement Learning



UP UP Right



UP, DOWN, LEFT, RIGHT

MDP :—

- set of possible states
- set of actions
- set of models
- Reward function $R(s,a)$
- A policy the solution of MDP

States - S

Models - $T(s, a, s') \sim P(s' \mid s, a)$

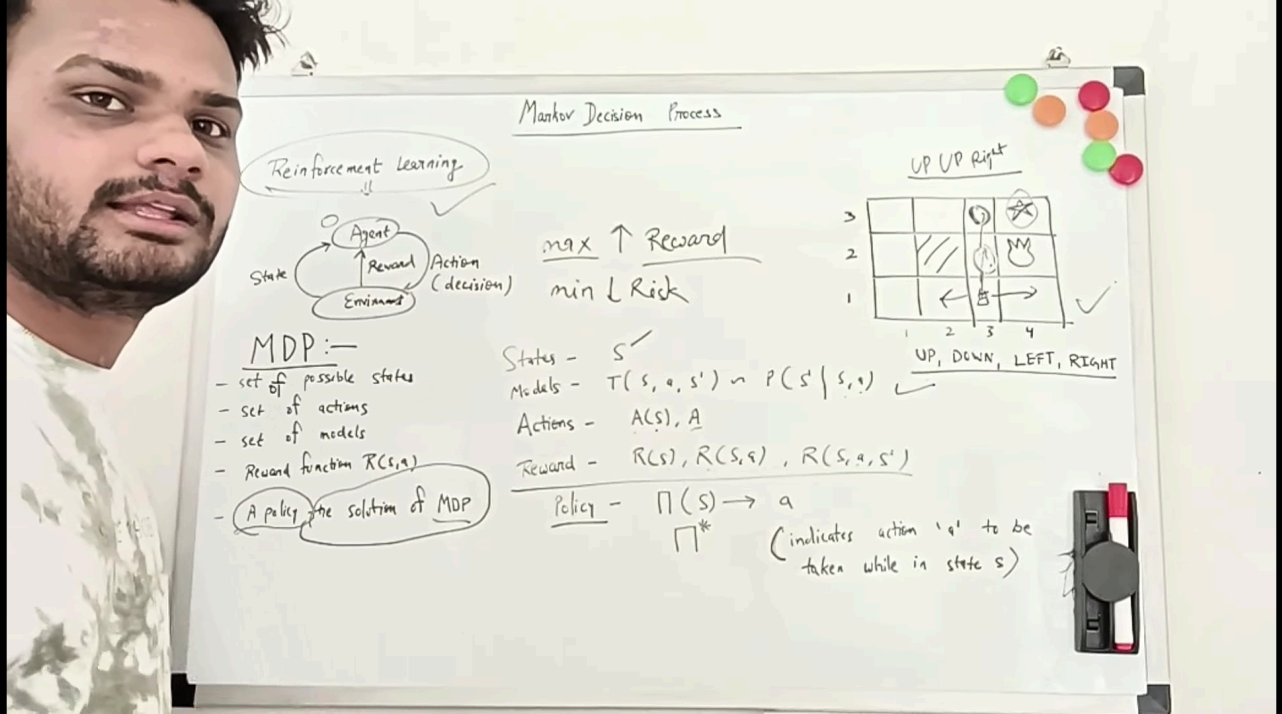
Actions - $A(s)$, A

Reward - $R(s)$, $R(s,a)$, $R(s,a,s')$

Policy - $\Pi(s) \rightarrow a$

Π^*

(indicates action 'a' to be taken while in state s)


Markov Decision Process
Reinforcement Learning
Agent
State
Reward
Action
(decision)
Environment
max ↑ Reward
min ↓ Risk
UP UP Right
UP, DOWN, LEFT, RIGHT
MDP :—
- set of possible states
- set of actions
- set of models
- Reward function R(s,a)
- A policy the solution of MDP
States - S
Models - T(s, a, s') ~ P(s' | s, a)
Actions - A(s), A
Reward - R(s), R(s,a), R(s,a,s')
Policy - Π(s) → a
Π*
(indicates action 'a' to be taken while in state s)

Active and Passive Differences

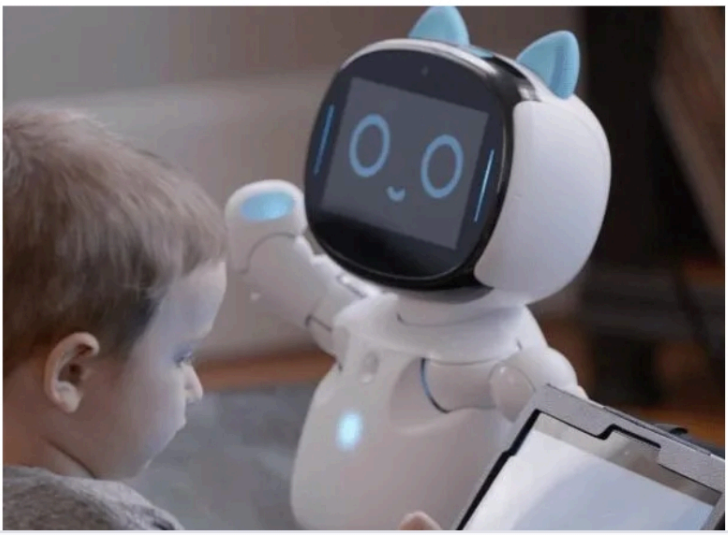
Active reinforcement learning is when the agent actively chooses the actions to perform based on the current state of the environment. This means that the agent has complete control over its actions and is free to explore different options to determine the best way to maximize its reward.

For example, in a game of chess, an active reinforcement learning agent would be responsible for deciding which move to make based on its evaluation of the current state of the game.

When playing chess, an agent (player) can select actions and respond to their environment (the board) to bring them closer to their goal. 📷: Getty Images

Passive reinforcement learning, on the other hand, occurs when the agent does not have control over its actions. Instead, the actions are determined by an external agent, such as a human operator or a pre-programmed algorithm. In these situations, the agent simply receives rewards or penalties based on its actions, and its goal is to learn from this feedback to improve its performance.

A good example of passive reinforcement learning is in robotics, where an external agent might provide rewards for reaching a target location or completing a specific task. The robot would receive feedback in the form of a reward or penalty for its actions, but it would not have the ability to choose its own actions.



A robot is told what to do, and over time can learn more efficient methods of performing those actions. 📷: Cite

Which One to Use

So, when should you use active reinforcement learning and when should you use passive reinforcement learning? The choice between these two approaches depends on the particular problem you are trying to solve.

Active reinforcement learning is ideal for situations where the agent needs to have control over its actions and be able to explore different options to find the best solution. This type of reinforcement learning is often used in game playing, where the agent needs to be able to make decisions based on the current state of the game.

Passive reinforcement learning, on the other hand, is well-suited for situations where the agent does not have control over its actions. This approach is often used in robotics, where the robot needs to learn from its actions, but the actions themselves are determined by an external agent.

Another key difference between active and passive reinforcement learning is the type of feedback the agent receives. In active reinforcement learning, the agent receives feedback in the form of rewards or penalties for its actions, allowing it to adjust its behavior accordingly. In passive reinforcement learning, the agent only receives feedback in the form of rewards or penalties at the end, and it must use this information to make predictions about the best action to take in the future.

. . .

In conclusion, both active and passive reinforcement learning have their own strengths and weaknesses, and the choice between the two depends on the specific problem you are trying to solve.

> *If the agent needs to have control over its actions and be able to explore different options, then **active reinforcement learning** may be the best approach.*
>
> *If the agent does not have control over its actions and must learn from feedback, then **passive reinforcement learning** may be the more appropriate choice.*

Thanks for reading and be sure to check out my other AI / tech articles!

Reinforcement Learning | Active Reinforcement | Passive Reinforcement | Reinforcement Examples | Ai Reinforcement

Passive Reinforcement Learning

- In passive learning, **the agent's policy π is fixed**:
- In state s, agent already has a fixed policy π(s) that determines its actions.
- The agent trying to learn Utility Function $U^{\pi}(s)$, the expected total discounted reward,
- if policy π is executed in beginning of state s then the agent computes the expected total discounted reward.
- this agent is called as **passive learning agent**.
- The passive learning task is similar to the **policy evaluation** task

Passive Reinforcement Learning…

- Each transition is annotated with both the action taken and the reward received at the next state.
- The agent used the information about rewards to learn the expected utility $U^{\pi}(s)$ associated with each nonterminal state s.
- The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$$

time

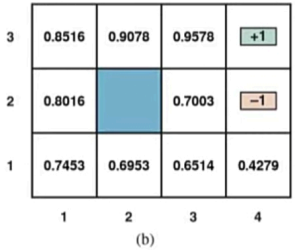
- where R(St,π(St),St+1) is the reward received when action π(St) is taken in state S_t
- And reaches state S_{t+1}
- St is the state reached at time t when executing policy π.
- Include a **discount factor** γ in all equations, but for the 4×3 world, set γ =1.

Passive Reinforcement Learning…

- Direct utility estimation
- Adaptive dynamic programming
- Temporal-difference learning

Direct Utility Estimation (Model Free)

- The utility of a state is defined as, the expected total reward from that state onward (called the **expected reward-to-go**),
- Estimate ***Uπ(s) as average total reward of goals*** containing s (calculating from s to end of goal)
- At each trial the algorithm
  - calculates the observed reward-to-go for each state and
  - updates the estimated utility for that state,
- by keeping a running average for each state in a table.



Direct Utility Estimation…

- The direct utility estimation is just an instance of supervised learning where each example has
  - the state as input and
  - the observed reward-to-go as output.
- This means the reinforcement learning is reduced to a standard inductive learning problem.
- The computed expected reward can be applied directly to the observed data.

Adaptive Dynamic Programming (ADP) (Model Based)

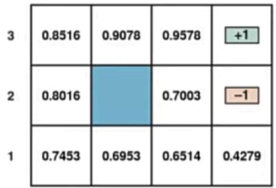
- An **Adaptive Dynamic Programming (**ADP) agent adapted the constraints in Bellman Equation.
- It learns the transition model and solve the corresponding Markov decision process.
- That is, the learned transition model P(s' | s, π(s)) and the observed rewards R(s) into the **Bellman equations**

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s')$$

- to calculate the utilities of the states.
- these equations are linear, so they can be solved using any linear algebra package.

Adaptive Dynamic Programming (ADP)…

- The environment is fully observable hence the learning the model is easy
- In this supervised learning, where
  - the input is a state–action pair and
  - the output is the resulting state.
- Represent the transition model as a table of probabilities.
- Keep track of, how often each action outcome occurs and estimate the transition probability P(s'| s, a) from the frequency with which s' is reached when executing a in s.



(b)

Adaptive dynamic programming (ADP)…

- ADP is a smarter method than Direct Utility Estimation.
- It runs trials to learn the model of the environment.
- It adopts the approach of modified policy iteration.
- It updates the utility estimates after each change.
- ADP is a model based approach and requires the transition model of the environment.

Temporal-difference (TD) Learning

- The *TD* is much simpler than direct utility estimation and Adaptive Dynamic Programming.
- It requires much less computation per observation.
- *It does not need a transition model to perform its updates*.
- The environment itself supplies the connection between neighboring states, in the form of observed transitions.

Temporal-difference learning...

uncertain, infinite

- **Temporal-difference** (TD) Equation
- A transition occurs from state s to state s' via action π(s),
- apply the following update to $U^{\pi}(s)$,

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)].$$

- Here, α is the **learning rate** parameter.
- This update rule uses the difference in utilities between successive states (s, s')
- it is called the **temporal-difference** (TD) equation

Temporal-difference learning…

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)].$$

- TD term $R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)$ is an error signal, and
- the update is used to reduce the error.
- All temporal-difference methods work by adjusting the utility estimates when the estimates are correct
- notice that the update involves only the observed successor s',
-

Comparison

- Direct Estimation (Model Free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints and converges slowly
- Adaptive Dynamic Programming (Model Based)
  - Harder to implement
  - Each update is full policy evaluation (expensive)
  - Fully exploits Bellman Constraints
  - Fast convergence (in terms of times)
- Temporal Difference Learning (model free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman Constraints – adjusts state to 'agree' with observed successor
  - Convergence in between direct estimation and ADP

1. ***Direct Utility Estimation***:

In this method, the agent executes a sequence of trials or runs (sequences of states-actions transitions that continue until the agent reaches the terminal state). Each trial gives a sample value and the agent estimates the utility based on the samples values. Can be calculated as running averages of sample values. The main drawback is that this method makes a wrong assumption that *state utilities are independent* while in reality they are Markovian. Also, it is slow to converge.

Suppose we have a 4x3 grid as the environment in which the agent can move either Left, Right, Up or Down(set of available actions). An

Suppose we have a 4x3 grid as the environment in which the agent can move either Left, Right, Up or Down(set of available actions). An example of a run,

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

Total reward starting at *(1,1)* = 0.72

2. Adaptive Dynamic Programming(ADP)

ADP is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

sum of reward for being in that state and the expected discounted reward of being in the next state.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

Where $R(s)$ = reward for being in state s, $P(s'|s, \pi(s))$ = transition model, γ = discount factor and $U^{\pi}(s)$ = utility of being in state s'.

It can be solved using value-iteration algorithm. The algorithm converges fast but can become quite costly to compute for large state spaces. ADP is a model based approach and requires the transition model of the environment. A model-free approach is Temporal Difference Learning.

3. ***Temporal Difference Learning (TD)***

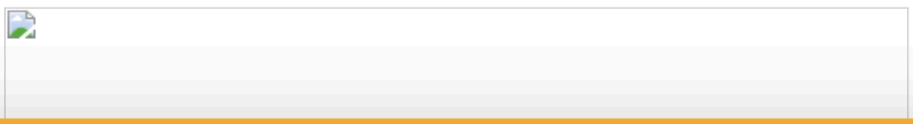
3. *Temporal Difference Learning (TD)*

TD learning does not require the agent to learn the transition model. The update occurs between successive states and agent only updates states that are directly affected.

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma(U^{\pi}(s') - U^{\pi}(s))$$

Where α = learning rate which determines the convergence to true utilities.

While ADP adjusts the utility of s with all its successor states, TD learning adjusts it with that of a single successor state s'. TD is slower in convergence but much simpler in terms of computation.



chess (source:

https://robotnor.no/expertise/fields-of-competence/robot-learning/)

Active Learning

1. ***ADP with exploration function***

As the goal of an active agent is to learn an optimal policy, the agent needs to learn the expected utility of each state and update its policy.

Can be done using a passive ADP agent and then using value or policy iteration it can learn optimal actions. But this approach results into a greedy agent.

Hence, we use an approach that gives higher weights to unexplored actions and lower weights to actions with lower utilities.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A} f\Big(\sum_{s'} P(s'|s,a) U_i(s'), N(s,a)\Big)$$

Where *f(u,n)* is the exploration function that increases with expected value *u* and decreases with number of tries *n*

$$f(u,n) = \begin{cases} R^+, if n < N_e \\ u, otherwise \end{cases}$$

R^+ is an optimisic reward and N_e is the number of times we want an agent to be forced to pick an action in every state. The exploration function

state. The exploration function *converts a passive agent into an active one.*

2. *Q-Learning*

Q-learning is a TD learning method which does not require the agent to learn the transitional model, instead learns Q-value functions *Q(s, a)* .

$$U(s) = \max_a Q(s, a)$$

Q-values can be updated using the following equation,

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big)$$

Next action can be selected using the following policy,

$$a_{next} = arg\max_{a'} f(Q(s', a'), N(s', a'))$$

Again this is simpler to compute but slower than ADP.

Table 1: Comparison of active and passive learning methods

	Fixed Policy (Active)	Policy not fixed (Passive)
Model-free (real world)	Temporal Difference Learning (TD)	Q-learning
Model-based (simulation)	Adaptive Dynamic Programming(ADP)	ADP with proper exploration function

I'd recommend the following resources

1. Passive Learning

Fully observable environment

Policy : π, State: s, Action: π(s)

Goal : How good the policy is

Utility function: $U^{\pi}(S)$

2. Adaptive dynamic estimation

B

Learning transition model of environment

solving the corresponding Markov decision process

Adaptive dynamic estimation

Adopt the approach of **modified policy iteration** update the utility estimates after each change

3. Temporal Difference Learning

Use the observed transitions to adjust the values of the observed states

Suppose that, as a result of the filrst trial, the utility estimates are $U^{\pi}(1,3) = 0.84$ and $U^{\pi}(2,3) = 0.92$.

Now, if this transition occurred all the time, we would expect the utilities to obey:

$$U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$$

so $U^{\pi}(1,3)$ would be 0.88.

Exploration

Solution is:

Right

Right

Up

Up

Right

$$U^{+}(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s'} P(s' \mid s, a) U^{+}(s'),\ N(s, a)\right) .$$

Exploitation and Exploration in Machine Learning

Exploitation and exploration are the key concepts in Reinforcement Learning, which help the agent to build online decision making in a better way. Reinforcement learning is a machine learning method in which an intelligent agent (computer program) learns to interact with the environment and take actions to maximize rewards in a specific situation. This ML method is currently being used in so many industries such as automobile, healthcare, medicine, education, etc.

As in Reinforcement learning, the agent is not aware of the different states, actions for each state, associate rewards, and transition to the next state, but it learns it

As in Reinforcement learning, the agent is not aware of the different states, actions for each state, associate rewards, and transition to the next state, but it learns it by exploring the environment. However, the knowledge of an agent about the state, actions, rewards, and resulting states is partial, and this results in **Exploration-Exploitation Dilemma**. In this topic, "**Exploitation and Exploration in Machine Learning**," we will discuss both these terms in detail with suitable examples. But before starting the topic, let's first understand reinforcement learning in ML.

What is 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,

Key points in Reinforcement Learning

- 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.

Hence, we can define reinforcement learning as:

"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."

What are Exploration and Exploitation in Reinforcement Learning

Before going to a brief description of

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

Definition of generalization?

In machine learning, generalization is a definition to demonstrate how well is a trained model to classify or forecast unseen data. Training a generalized machine learning model means, in general, it works for all subset of unseen data. An example is when we train a model to classify between dogs and cats. If the model is provided with dogs images dataset with only two breeds, it may obtain a good performance. But, it possibly gets a low classification score when it is tested by other breeds of dogs as well. This issue can result to classify an actual dog image as a cat from the unseen dataset. Therefore, data diversity is very important factor in order to make a good prediction. In the sample above, the model may obtain 85% performance score when it is tested by only two dog breeds and gains 70% if trained by all breeds. However, the first possibly gets a very low score (e.g. 45%) if it is evaluated by an unseen dataset with all breed dogs. This for the latter can be unchanged given than it has been trained by high data diversity including all possible breeds.

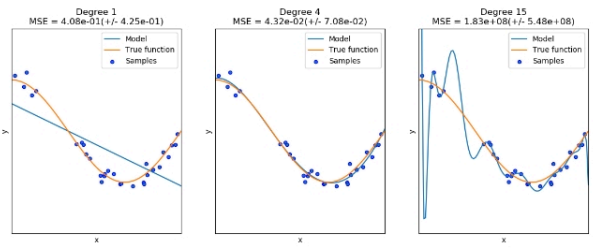
It should be taken into account that data diversity is not the only point to care in order to have a generalized model. It can be resulted by nature of a machine learning algorithm, or by poor hyper-parameter configuration. In this post we explain all determinant factors. There are some methods (regularization) to apply during model training to ensure about generalization. But before, we explain bias and variance as well as underfitting and overfitting.

Variance and bias (overfitting and underfitting)

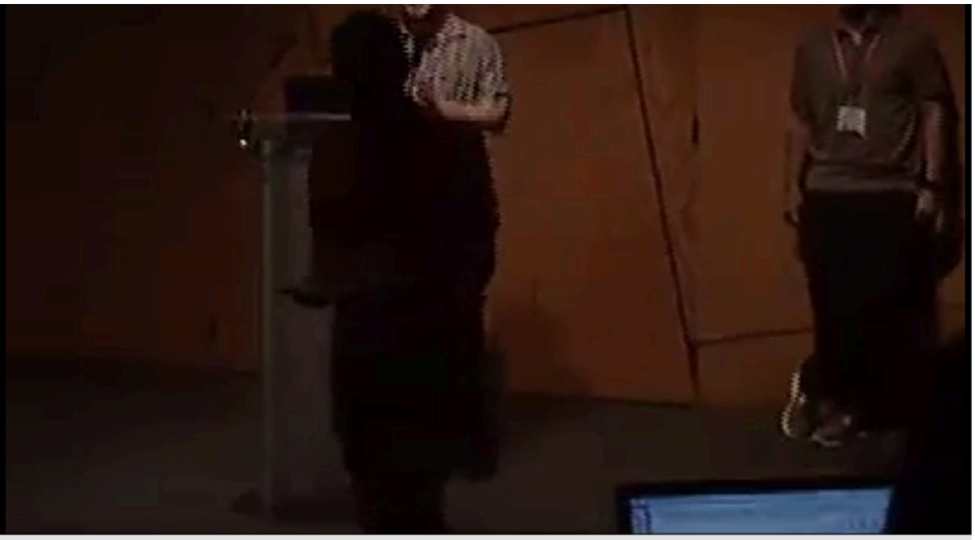
Variance and bias are two important terms in machine learning. Variance means the variety of predictions values made by a machine learning model (target function). Bias means the distance of the predictions from the actual (true) target values. A high-biased model means its prediction values (average) are far from the actual values. Also, high-variance prediction means the prediction values are highly varied.

Variance-bias trade-off

The prediction results of a machine learning model stand somewhere between a) low-bias, low-variance, b) low-bias, high-variance c) high-bias, low-variance, and d) high-bias, high-variance. A low-biased, high-variance model is called overfit and a high-biased, low-variance model is called underfit. By generalization, we find the best trade-off between underfitting and overfitting so that a trained model obtains the best performance. An overfit model obtains a high prediction score on seen data and low one from unseen datsets. An underfit model has low performance in both seen and unseen datasets.



Three models with underfitting (left), goodfit (middle), and overfitting (center).
Credit: https://scikit-learn.org/



Lecture popularity: ☆☆☆☆☆ You need to login to cast your vote.

Watch videos: *(click on thumbnail to launch)*

Part 1 1:07:38
!NOW PLAYING

Part 2 1:11:15

Description

Policy search is a subfield in reinforcement learning which focuses on finding good parameters for a given policy parametrization. It is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning. We review recent successes of both model-free and model-based policy search in robot learning. Model-free policy search is a general approach to learn policies based on sampled trajectories. We classify model-free methods based on their policy evaluation strategy, policy update strategy, and exploration strategy and present a unified view on existing algorithms. Learning a policy is often easier than learning an accurate forward model, and, hence, model-free methods are more frequently used in practice. How- ever, for each sampled trajectory, it is necessary to interact with the robot, which can be time consuming and challenging in practice. Model-based policy search addresses this problem by first learning a simulator of the robot's dynamics from data. Subsequently, the simulator generates trajectories that are used for policy learning. For both model- free and model-based policy search methods, we review their respective properties and their applicability to robotic systems.

Link this page

Would you like to put a link to this lecture on your homepage?
Go ahead! Copy the HTML snippet !

Write your own review or comment:

Name:

Game Playing in Artificial Intelligence

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game. Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

Game playing is a popular application of artificial intelligence that involves the development of computer programs to play games, such as chess, checkers, or Go. The goal of game playing in artificial intelligence is to develop algorithms that can learn how to play games and make decisions that will lead to winning outcomes.

1. One of the earliest examples of successful game playing AI is the chess program Deep Blue, developed by IBM, which defeated the world champion Garry Kasparov in 1997. Since then, AI has been applied to a wide range of games, including two-player games, multiplayer games, and video games.

There are two main approaches to game playing in AI, rule-based systems and machine learning-based systems.

1. **Rule-based systems** use a set of fixed rules to play the game.
2. **Machine learning-based systems** use algorithms to learn from experience and make decisions based on that experience.

In recent years, machine learning-based systems have become increasingly popular, as they are able to learn from experience and improve over time, making them well-suited for complex games such as Go. For example, AlphaGo, developed by DeepMind, was the first machine learning-based system to defeat a world champion in the game of Go.

Game playing in AI is an active area of research and has many practical applications, including game development, education, and military training. By simulating game playing scenarios, AI algorithms can be used to develop more effective decision-making systems for real-world applications.

The most common search technique in game playing is Minimax search procedure. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Minimax algorithm uses two functions –

MOVEGEN : It generates all the possible moves that can be generated from the current position.

STATICEVALUATION : It returns a value depending upon the goodness from the viewpoint of two-player

This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move. It is a recursive algorithm, as same procedure occurs at each level.

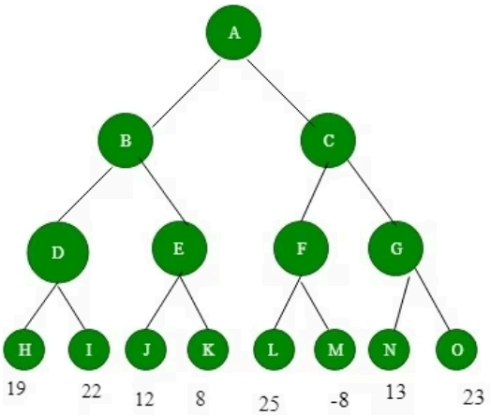


Figure 1: Before backing-up of values

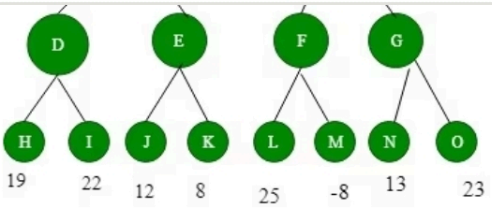


Figure 1: Before backing-up of values

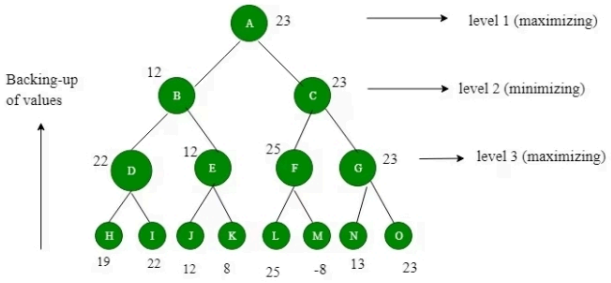


Figure 2: After backing-up of values We assume that PLAYER1 will start the game.

4 levels are generated. The value to nodes H, I, J, K, L, M, N, O is provided by STATICEVALUATION function. Level 3 is maximizing level, so all nodes of level 3 will take maximum values of their children. Level 2 is minimizing level, so all its nodes will take minimum values of their children. This process continues. The value of A is 23. That means A should choose C move to win.

Reference : Artificial Intelligence by Rich and Knight

Advantages of Game Playing in Artificial Intelligence:

1. **Advancement of AI:** Game playing has been a driving force behind the development of artificial intelligence and has led to the creation of new algorithms and techniques that can be applied to other areas of AI.
2. **Education and training:** Game playing can be used to teach AI techniques and algorithms to students and professionals, as well as to provide training for military and emergency response personnel.
3. **Research:** Game playing is an active area of research in AI and provides an opportunity to study and develop new techniques for decision-making and problem-solving.
4. **Real-world applications:** The techniques and algorithms developed for game playing can be applied to real-world applications, such as robotics, autonomous systems, and decision support systems.

Disadvantages of Game Playing in Artificial Intelligence:

1. **Limited scope:** The techniques and algorithms developed for game playing may not be well-suited for other types of applications and may need to be adapted or modified for different domains.
2. **Computational cost:** Game playing can be computationally expensive, especially for complex games such as chess or Go, and may require powerful computers to achieve real-time performance.

Reference books:

1. “Artificial Intelligence and Games” edited by Michael Buro.
2. “Game AI Pro: Collected Wisdom of Game AI Professionals” edited by Steve Rabin.
3. “Artificial Intelligence with Python” by Prateek Joshi

Article Tags : Computer Subject | Artificial Intelligence

Recommended Articles

1. Difference Between Artificial Intelligence and Human Intelligence

5 Current Machine Learning Applications in Robotics

** Terms in **italics and bold** are defined further in the glossary at the bottom of this post.*

1 – Computer Vision

Though related, some would argue the correct term is machine vision or robot vision rather than computer vision, because "robots seeing" involves more than just computer algorithms; engineers and roboticists also have to account for camera hardware that allow robots to process physical data. Robot vision is very closely linked to machine vision, which can be given credit for the emergence of robot guidance and automatic inspection systems. The slight difference between the two may be in ***kinematics*** as applied to robot vision, which encompasses reference frame calibration and a robot's ability to physically affect its environment.

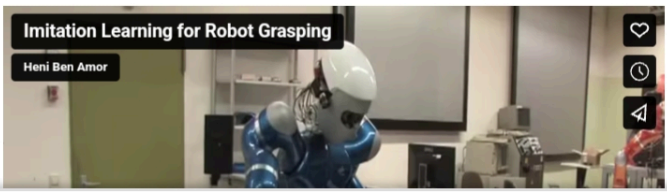
An influx of big data i.e. visual information available on the web (including annotated/labeled photos and videos) has propelled advances in computer vision, which in turn has helped further machine-learning based structured prediction learning techniques at universities like Carnegie Mellon and elsewhere, leading to robot vision applications like identification and sorting of objects. One offshoot example of this is anomaly detection with unsupervised learning, such as building systems capable of finding and assessing faults in silicon wafers using convolutional neural networks, as engineered by researchers at the Biomimetic Robotics and Machine Learning Lab, which is part of the nonprofit Assistenzrobotik e.V. in Munich.

Extrasensory technologies like radar, lidar, and ultrasound, like those from Nvidia, are also driving the development of 360-degree vision-based systems for autonomous vehicles and drones.

2 – Imitation Learning

Imitation learning is closely related to observational learning, a behavior exhibited by infants and toddlers. Imitation learning is also an umbrella category for reinforcement learning, or the challenge of getting an agent to act in the world so as to maximize its rewards. ***Bayesian or probabilistic models*** are a common feature of this machine learning approach. The question of whether imitation learning could be used for humanoid-like robots was postulated as far back as 1999.

Imitation learning has become an integral part of field robotics, in which characteristics of mobility outside a factory setting in domains like domains like construction, agriculture, search and rescue, military, and others, make it challenging to manually program robotic solutions. Examples include inverse optimal control methods, or "programming by demonstration,"which has been applied by CMU and other organizations in the areas of humanoid robotics, legged locomotion, and off-road rough-terrain mobile navigators. Researchers from Arizona State published this video two years ago showing a humanoid robot using imitation learning to acquire different grasping techniques:



the "iCub" humanoid robot.

3 – Self-Supervised Learning

Self-supervised learning approaches enable robots to generate their own training examples in order to improve performance; this includes using *a priori* training and data captured close range to interpret "long-range ambiguous sensor data." It's been incorporated into robots and optical devices that can detect and reject objects (dust and snow, for example); identify vegetables and obstacles in rough terrain; and in 3D-scene analysis and modeling vehicle dynamics

Watch-Bot is a concrete example, created by researchers from Cornell and Stanford, that uses a 3D sensor (a Kinect), a camera, laptop and laser pointer to detect 'normal human activity', which are patterns that it learns through probabilistic methods. Watch-Bot uses a laser pointer to target the object as a reminder (for example, the milk that was left out of the fridge). In initial tests, the bot was able to successfully remind humans 60 percent of time (it has no conception of what it's doing or why), and the researchers expanded trials by allowing its robot to learn from online videos (called project RoboWatch).

Other examples of self-supervised learning methods applied in robotics include a road detection algorithm in a front-view monocular camera with a road probabilistic distribution model (RPDM) and fuzzy support vector machines (FSVMs), designed at MIT for autonomous vehicles and other mobile on-road robots.

Autonomous learning, which is a variant of self-supervised learning involving deep learning and unsupervised methods, has also been applied to robot and control tasks.

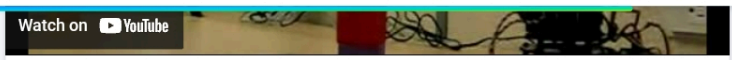
A team at Imperial College in London, collaborating with researchers from University of Cambridge and University of Washington, has created a new method for speeding up learning that incorporates model uncertainty (a probabilistic model) into long-term planning and controller learning, reducing the effect of model errors when learning a new skill.

This statistical machine learning approach is put into action by the team's manipulator in the video below:

4 – Assistive and Medical Technologies

An assistive robot (according to Stanford's David L. Jaffe) is a device that can sense, process sensory information, and perform actions that benefit people with disabilities and seniors (though smart assistive technologies also exist for the general population, such as driver assistance tools). Movement therapy robots provide a diagnostic or therapeutic benefit. Both of these are technologies that are largely (and unfortunately) still confined to the lab, as they're still cost-prohibitive for most hospitals in the U.S. and abroad.

Early examples of assistive technologies included the DeVAR, or



4 – Assistive and Medical Technologies

An assistive robot (according to Stanford's David L. Jaffe) is a device that can sense, process sensory information, and perform actions that benefit people with disabilities and seniors (though smart assistive technologies also exist for the general population, such as driver assistance tools). Movement therapy robots provide a diagnostic or therapeutic benefit. Both of these are technologies that are largely (and unfortunately) still confined to the lab, as they're still cost-prohibitive for most hospitals in the U.S. and abroad.

Early examples of assistive technologies included the DeVAR, or desktop vocational assistant robot, developed in the early 1990s by Stanford and the Palo Alto Veterans Affairs Rehabilitation Research and Development. More recent examples of machine learning-based robotic assistive technologies are being developed that include combining assistive machines with more autonomy, such as the MICO robotic arm (developed at Northwester University) that observes the world through a Kinect Sensor. The implications are more complex yet smarter assistive robots that adapt more readily to user needs but also require partial autonomy (i.e. a sharing of control between the robot and human).

In the medical world, advances in machine learning methodologies applied to robotics are fast advancing, even though not readily available in many medical facilities. A collaboration through the Cal-MR: Center for Automation and Learning for Medical Robotics, between researchers at multiple universities and a network of physicians (collaborations with researchers at multiple universities and physicians led to the creation of the Smart Tissue Autonomous Robot (STAR), piloted through the Children's National Health System in DC. Using innovations in autonomous learning and 3D sensing, STAR is able to stitch together "pig intestines" (used in lieu of human tissue) with better precision and reliability than the best human surgeons. Researchers and physicians make the statement that STAR is not a replacement for surgeons – who for the foreseeable future would remain nearby to handle emergencies – but offer major benefits in performing similar types of delicate surgeries.

5 – Multi-Agent Learning

Coordination and negotiation are key components of multi-agent learning, which involves machine learning-based robots (or agents – this technique has been widely applied to games) that are able to adapt to a shifting landscape of other robots/agents and find "equilibrium strategies." Examples of multi-agent learning approaches include no-regret learning tools, which involve weighted algorithms that "boost" learning outcomes in multi-agent planning, and learning in market-based, distributed control systems.

A more concrete example is an algorithm for distributed agents or robots created by researchers from MIT's Lab for Information and Decision Systems in late 2014. Robots collaborated to build a better and more inclusive learning model than could be done with one robot (smaller chunks of information processed and then combined), based on the concept of exploring a building and its room layouts and autonomously building a knowledge base.

Each robot built its own catalog, and combined with other robots' data sets, the distributed algorithm outperformed the standard algorithm in creating this knowledge base. While not a perfect system, this type of machine learning approach allows robots to compare catalogs or data sets, reinforce mutual observations and correct omissions or over-generalizations, and will undoubtedly play a near-future role in several robotic applications, including multiple autonomous land and airborne vehicles.

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.

Advantages of CART

- Results are simplistic.
- Classification and regression trees are Nonparametric and Nonlinear.
- Classification and regression trees implicitly perform feature selection.
- Outliers have no meaningful effect on CART.
- It requires minimal supervision and produces easy-to-understand models.

Limitations of CART

- Overfitting.
- High Variance.
- low bias.
- the tree structure may be unstable.

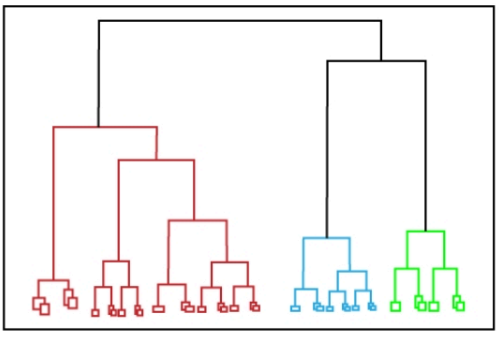
Applications of the CART algorithm

- For quick Data insights.
- In Blood Donors Classification.
- For environmental and ecological data.
- In the financial sectors.

Article Tags : Machine Learning, ML-Classification, ML-Regression

Hierarchical Clustering

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



Fuzzy Clustering

Fuzzy clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be

Regression Algorithm	Classification Algorithm
In Regression, the output variable must be of continuous nature or real value.	In Classification, the output variable must be a discrete value.
The task of the regression algorithm is to map the input value (x) with the continuous output variable(y).	The task of the classification algorithm is to map the input value(x) with the discrete output variable(y).
Regression Algorithms are used with continuous data.	Classification Algorithms are used with discrete data.
In Regression, we try to find the best fit line, which can predict the output more accurately.	In Classification, we try to find the decision boundary, which can divide the dataset into different classes.
Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, etc.	Classification Algorithms can be used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
The regression Algorithm can be further divided into Linear and Non-linear Regression.	The Classification algorithms can be divided into Binary Classifier and Multi-class Classifier.