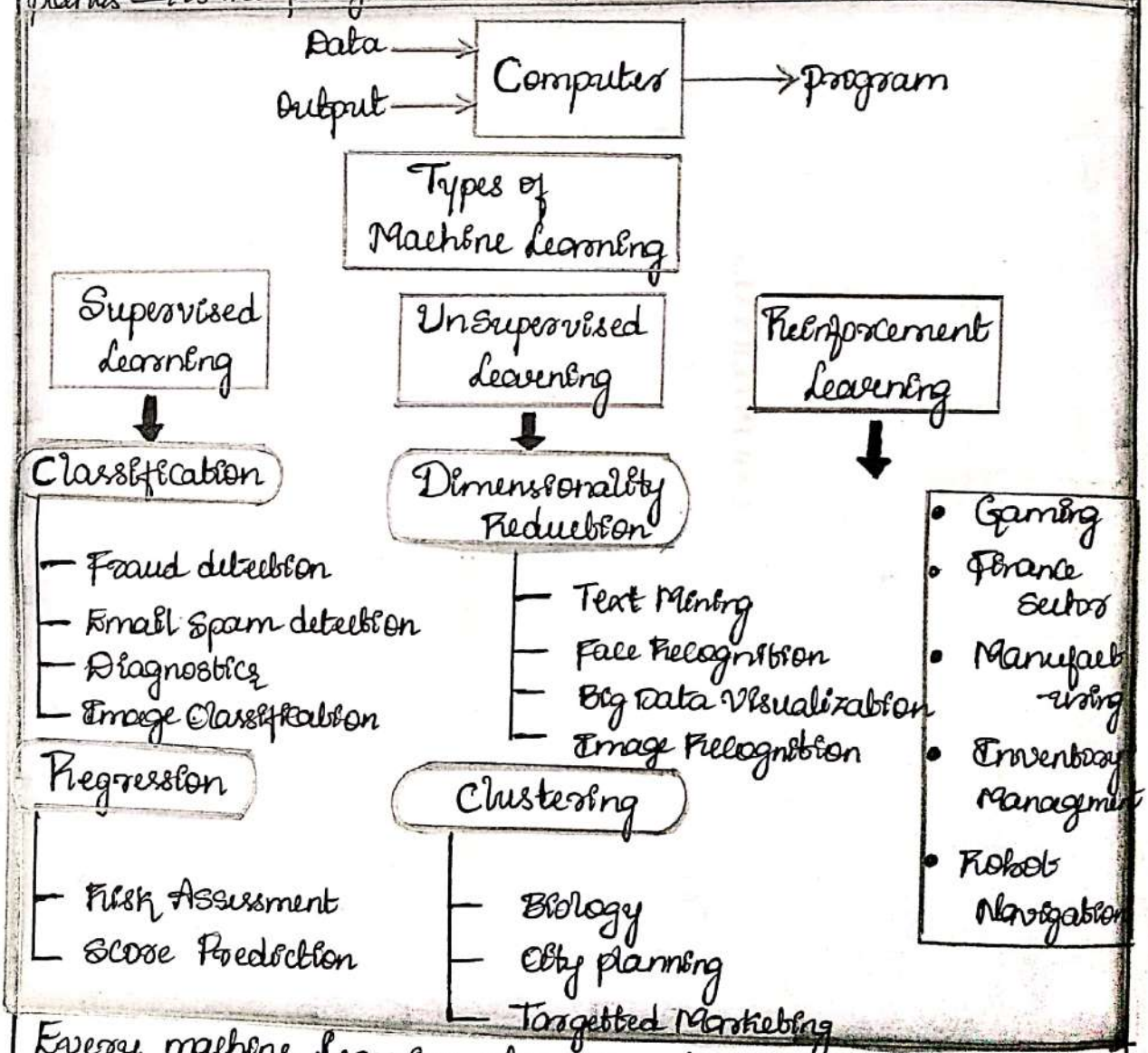
## Machine learning concepts

"Machine Learning" is getting computers to program themselves If programming is automation, then machine learning is automating the process of automation.

Machine Learning:- Data and output is run on the computer to create a program. This program can be used in traditional programming.

Machine Learning is like farming or gardening .. Seeds → is the algorithms, nutrients ← is the data, the gardner → is you and plants → is the programs.




Every machine learning algorithm has three components:

* Representation: how to represent knowledge Ex: Decision tree
* Evaluation: the way to evaluate candidate programs Ex: accuracy
* Optimization: Candidate programs are generated known as the search process Ex: Convex optimization

**Sample programs in Python** Date: 04/10/2021

Q1) Write a program in python to check if input number is prime or not.

```
number = int(input("Enter any number:"))
if number > 1:
    for i in range(2, number):
        if (number % i) == 0):
            print(number, "is not a prime number")
            break
        else:
            print(number, "is a prime number")
else:
    print(number, "is not a prime number")
```

Output:

11 is a prime number

15 is not a prime number

1 is not a prime number

Q2) Write a python program to make a simple Calculator that can add, subtract, multiply, divide using functions.

```
def add(x,y):
    return x+y
def subtract(x,y):
    return x-y
def multiply(x,y):
    return x*y
def divide(x,y):
    return x/y
print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
while True:
    choice=input("Enter choice(1/2/3/4):")
if choice in('1','2','3','4'):
    num1=float(input("Enter first number"))
    num2=float(input("Enter second number"))
if choice=='1':
    print(num1,"+",num2,"=",add(num1,num2))
elif choice=='2':
    print(num1,"-",num2,"=",subtract(num1,num2))
elif choice=='3':
    print(num1,"*",num2,"=",multiply(num1,num2))
elif choice=='4':
    print(num1,"/",num2,"=",divide(num1,num2))
next_calculation=input("proceed Calculation(yes/no)=")
if next_calculation=="no":
    break.
else:
    print("Invalid input")
```

Output:

```
Select operation
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4): 3
Enter first number : 15
Enter second number : 14
15.0 * 14.0 = 210.0
Proceed Calculation (Yes/No): No
```

Q3) Write a python program to find Factorial of a number.

```
num = int(input("Enter a number"))
factorial = 1
if num < 0
    print("Factorial does not exist for negative numbers")
elif num == 0:
    print("the factorial of 0 is 1")
else
  for i in range(1, num+1):
      factorial = factorial * i
  print("The Factorial of", num, "is", factorial)
```

Output:

```
Enter a number : 10
The factorial of 10 is 3628800

Enter a number : 5
The factorial of 5 is 120
```

Q4) Write a program to prompt for score between 0.0 and 1.0, if score is out of range print an error message else print the grade >0.9 then A > 0.8 then B, >0.7 then C, >0.6 then D, <0.5 fail score

```
score = input("Please type a score between 0.0 and 1.0")
try:
  float(score)
  if float(score) >= 0.9 and float(score) <= 1.0:
    print("A")
  elif float(score) >= 0.8 and float(score) <= 0.9:
    print("B")
  elif float(score) >= 0.7 and float(score) <= 0.6:
    print("C")
  elif float(score) >= 0.6 and float(score) <= 0.5:
    print("D")
  elif float(score) > 0 and float(score) <= = 0.6:
    print("fail")
  else:
    print("Badscore please run the program again")
except
    print("B and score please run the program again")
```

Output:

```
please type a score between 0.0 and 1.0: 0.9
A
please type a score between 0.0 and 1.0: 0.8
B
please type a score between 0.0 and 1.0: 0.4
D
please type a score between 0.0 and 1.0: 0.5
C
please type a score between 0.0 and 1.0: 0.2
fail
```

**Sample programs in Python with libraries** Date: 11|10|2021

## NumPy

NumPy, which Stands for Numerical Python. Is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Ex 1: Creating the array

```
import numpy as np
a= np.array([1,2,3])
print(a)
```

output

```
[1 2 3]
```

Ex 2: Creating array by specifying datatype

```
#dtype parameter
import numpy as np
a=np.array([1,2,3], dtype= complex)
print(a)
```

output

```
[1.+0.j 2.+0.j 3.+0.j]
```

Ex 3: Getting the Shape (No. of rows and No. columns)

```
import numpy as np
a=np.array([1,2,3],[4,5,6])
print(a.shape)
```

output

```
(2,3)
```

Ex 4: Reshaping the array

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
b= a.reshape(3,2)
print(b)
```

output

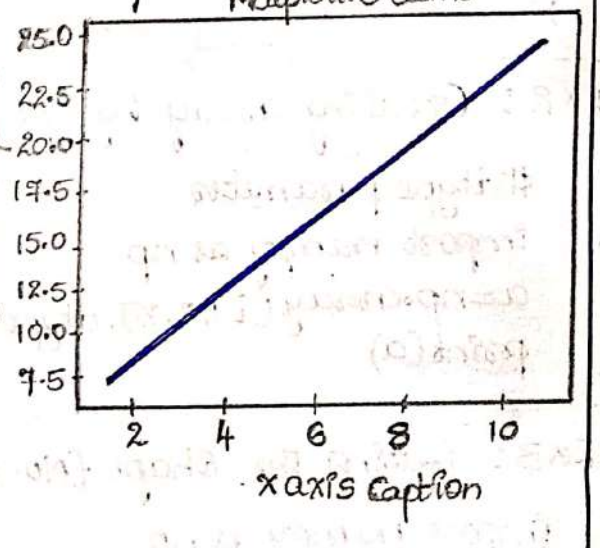
```
[[1 2]
 [3 4]
 [5 6]]
```

# MATPLOTLIB

Matplotlib is a plotting library for python. It is used along with Numpy to provide an environment that is an effective open source alternative for Matlab. It can also be used with graphics toolkits like PyQt and WxPython.

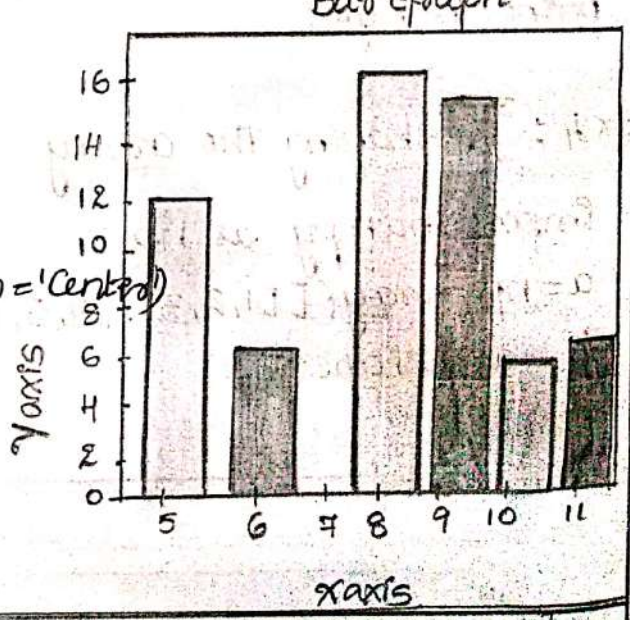
## Ex 1: Drawing the Line Plot

```
import numpy as np
from matplotlib import pyplot as plt
x = np.arrange(1, 11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x, y)
plt.show()
```




## Ex 2: Drawing the Bar plot

```
from matplotlib import pyplot as plt
x = [5, 8, 10]
y = [12, 16, 6]
x2 = [6, 9, 11]
y2 = [6, 15, 7]
plt.bar(x, y, align='center')
plt.bar(x2, y2, color='g', align='center')
plt.title('Bar graph')
plt.ylable('Y axis')
plt.xlable('X axis')
plt.show()
```



# Pandas

Pandas library, it provides high performance, easy to use data structures, and analysis tools for the python programming language. It is an open source python library which provides high performance data manipulation and analysis we use Pandas library to work with data frames.

Ex1: Reading .csv file and 1st column is made the index column and replace missing values with "??"

```
import pandas as pd
data_csv=pd.read_csv("Cars_data.csv")
data_csv=pd.read_csv("Cars_data.csv",index_col=0,na_values=
["??","???"])
```

Ex2: Knowing the number of rows and columns in the dataframe

```
data_csv.index
data_csv.columns
data_csv.shape
```

Ex3: Data types

```
print(data_csv.dtypes)          ; returns the datatype of each column
print(data_csv.info())          ; returns the summary of data type
print(np.unique(data_csv["Automatic"]))   ; returns unique element of the columns
data_csv.isnull().sum()         ; counts the number of missing values.
```

# Seaborn

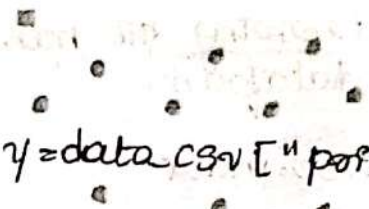
Seaborn library is a python data visualization library which was built on top of matplotlib library that provides a high level interface for drawing attractive and informative statistical graphs.

1. Import pandas as pd
   Import seaborn as sns

```
data_csv = pd.read_csv("Cars_data.csv", index_col=0, na_values=["??","???"])
data_csv.dropna(axis=0, inplace=True)
```

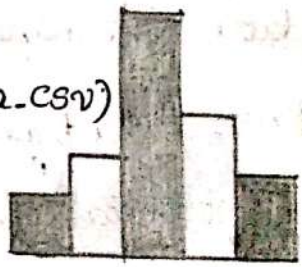
## Scatter plot

```
sns.set(style="darkgrid")
sns.regplot(x=data_csv["Age"], y=data_csv["price"])
```



## Bar plot

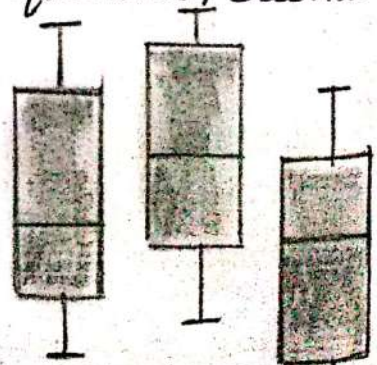
```
sns.countplot(x="Fuel_Type", data=data_csv)
```



## Box plot

The five number summary includes mean, median; The five number summary includes minimum maximum and the three quantiles those are first quantiles, second quantile and third quantile.

```
sns.boxplot(y=data_csv["price"])
```



**Program: 1.** DATE: 29/10/2021

**Implement A* Search algorithm.**

## DESCRIPTION

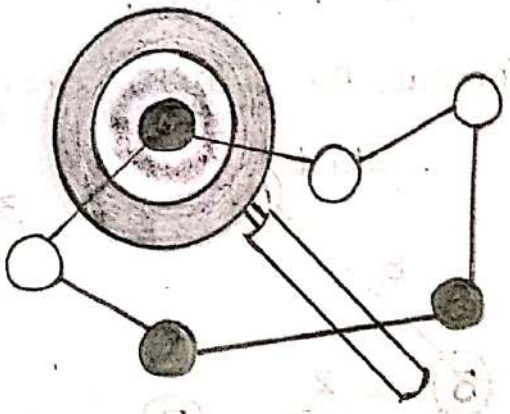
**Heuristic Search:**
- It is a "rule of thumb" used to help guide search.
- It is a technique that improves the efficiency of search process possibly by sacrificing claims of completeness.
- It is involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods.

Heuristic Function:
- It is a function applied to a state in a search space to indicate a likelihood of success if that state is selected.
- It is a function that maps from problem state descriptions to measures of desirability usually represented by numbers - Heuristic function is problem specific.

Heuristic search methods which are the general purpose control strategies for controlling search is often known as "weak methods" because of their generality and because they do not apply a great deal of knowledge.

Weak Methods:

a) Generate and Test
b) Hill Climbing
c) Best First Search
d) Problem Reduction
e) Constraint Satisfaction
f) Means-ends analysis.



Heuristic Search is class of method which is used in order to search a solution space for an optimal solution for a problem.

**A* algorithm():**

- Add Start node to list
- For all the neighbouring nodes, find the least cost Fnode
- Switch to the closed list
  - → For 8 nodes adjacent to the current node.
  - → If the node is not reachable, ignore it. Else
    - ⊛ If the node is not on the open list, move it to the open list and calculate f, g, h.
    - ⊛ If the node is on the open list. Check if the path it offers is less than the current path and change to it if it does so.
- Stop working when
  - → You find the destination
  - → You cannot find the destination going through all possible points.

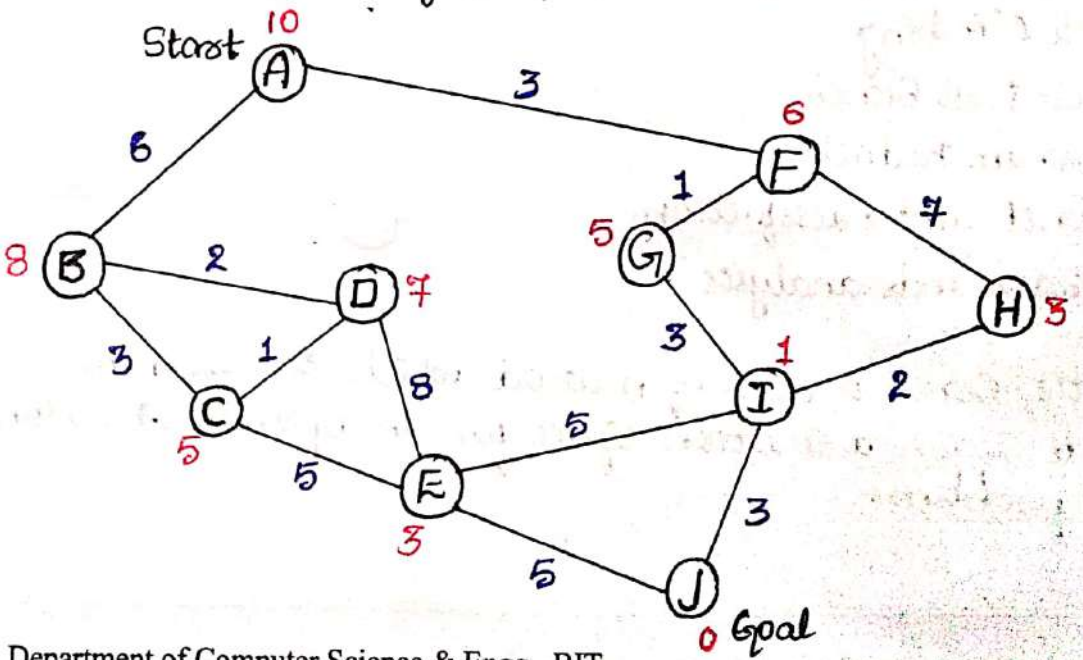
**Illustration of A* algorithm with example**

A* Algorithm : $f^*(n) = g^*(n) + h^*(n)$

$h^*(n)$ (Heuristic factor) = estimate of $h(n)$.

$g^*(n)$ (Depth factor) = approximation of $g(n)$ found by A* so far.

Consider the following graph:-



- The numbers written on edges represent the distance between the nodes.
- The numbers written on nodes represent the heuristic values.
- Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

## Solution:-

**Step-01:**

- We start with node A
- Node B and Node F can be reached from node A.

A* Algorithm calculates f(B) and f(F).

- f(B) = 6+8 = 14
- f(F) = 3+6 = 9

Since f(F) < f(B), so it decides to go to node F.

Path - A → F

**Step-02:**

Node G and Node H can be reached from node F.

A* Algorithm calculates f(G) and f(H)

- f(G) = (3+1)+5 = 9
- f(H) = (3+7)+3 = 13

Since, f(G) < f(H), so it decides to go to node G.

Path - A → F → G

**Step-03:**

Node I can be reached from node G.

A* Algorithm calculates f(I)

f(I) = (3+1+3)+1 = 8 ∴, it decides to go to node I.

Path - A → F → G → I

**Step-04:**

Node E, Node H and Node J can be reached from node I.

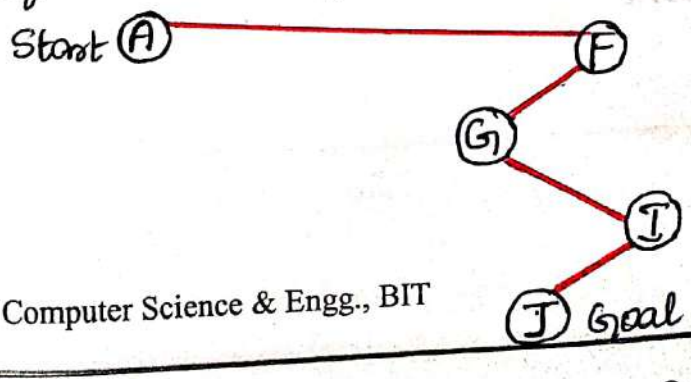
A* Algorithm calculates f(E), f(H) and f(J).

- f(E) = (3+1+3+5)+3 = 15
- f(H) = (3+1+3+2)+3 = 12
- f(J) = (3+1+3+3)+0 = 10

Since, f(J) is least, so it decides to go to node J.

Path - A → F → G → I → J

This is the required shortest path from node A to node J.



## SOURCE CODE

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
    if n == None:
        print('Path does not exist!')
        return None
```

## SOURCE CODE

```
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

def get_neighbors(v):
        if v in Graph_nodes:
            return Graph_nodes[v]
        else:
            return None

def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 5,
            'D': 7,
            'E': 3,
            'F': 6,
            'G': 5,
            'H': 3,
            'I': 1,
            'J': 0
        }
```

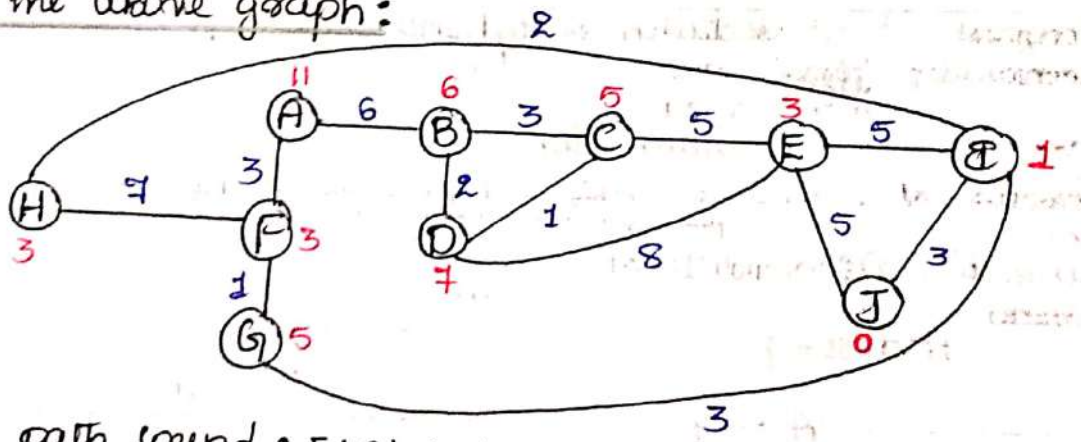
## OUTPUT

```
return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}
aStarAlgo('A', 'J')
```

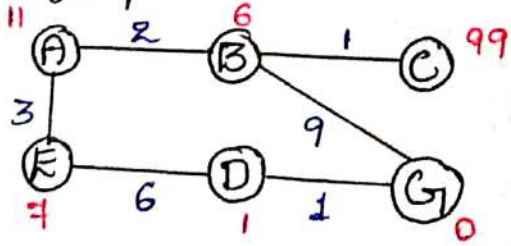
### : Output :

1) For the above graph:




path found: ['A', 'F', 'G', 'I', 'J']

2) For the below graph:



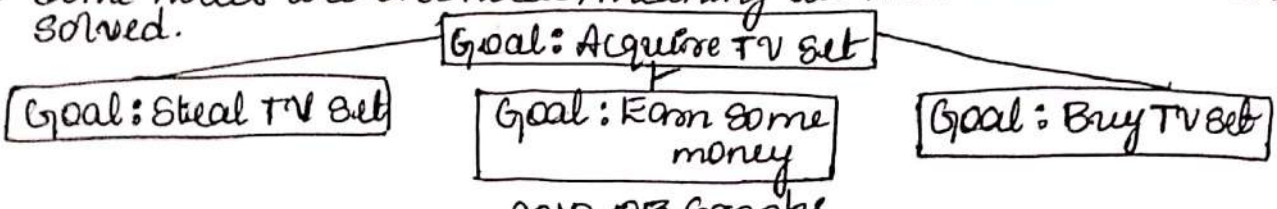

path found: ['A', 'E', 'D', 'G']

**Program: 3.** **DATE:** 12/11/2021

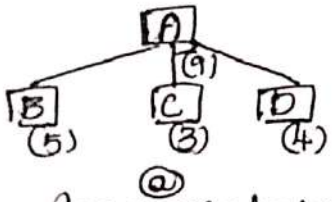
**Implement AO* Search algorithm.**

**Problem reduction:**

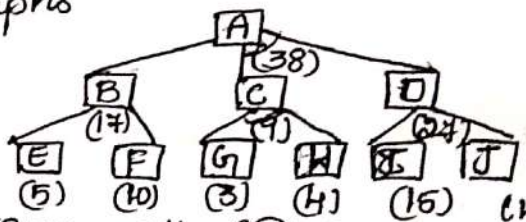
- Breaking a problem down into smaller sub-problems (or sub-goals)
- Can be represented using goal trees (or and-or trees).
- Nodes in the tree represent sub-problems.
- The root node represents the overall problem.
- Some nodes are and nodes, meaning all their children must be solved.




AND-OR Graphs




(a)



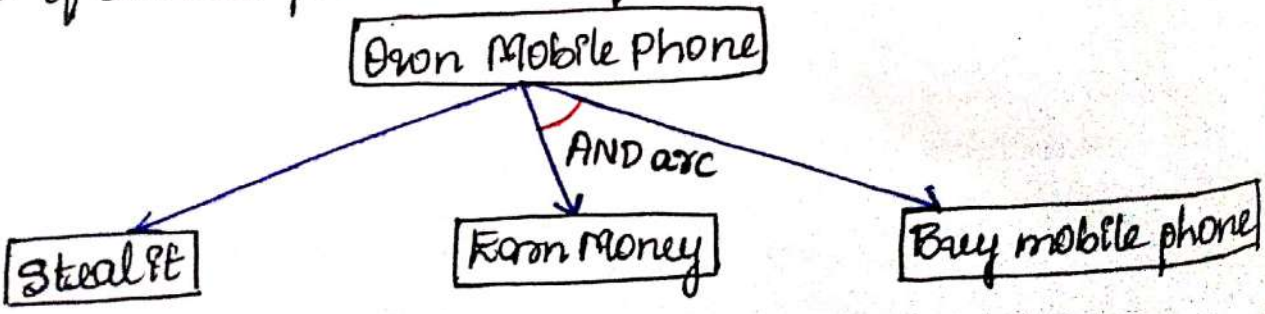

(b)

Examples: Matrix multiplication, Towers of Hanoi (b) Theorem proving

Formulations: — An OR node represents a choice b/w possible decompositions
— An AND node represents a given decomposition.

**AO* Search algorithm.**

- Its an informed search and works as best first search.
- AO* Algorithm is based on problem decomposition (Break down problem into small pieces)
- It represents an AND-OR graph algorithm that is used to find more than one solution.
- Its an efficient method to explore a solution path.
- AND-OR graph is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.



- Node in the graph will point both down to its successors and up to its parent nodes.
- Each node in the graph will also have a heuristic value associated with it.

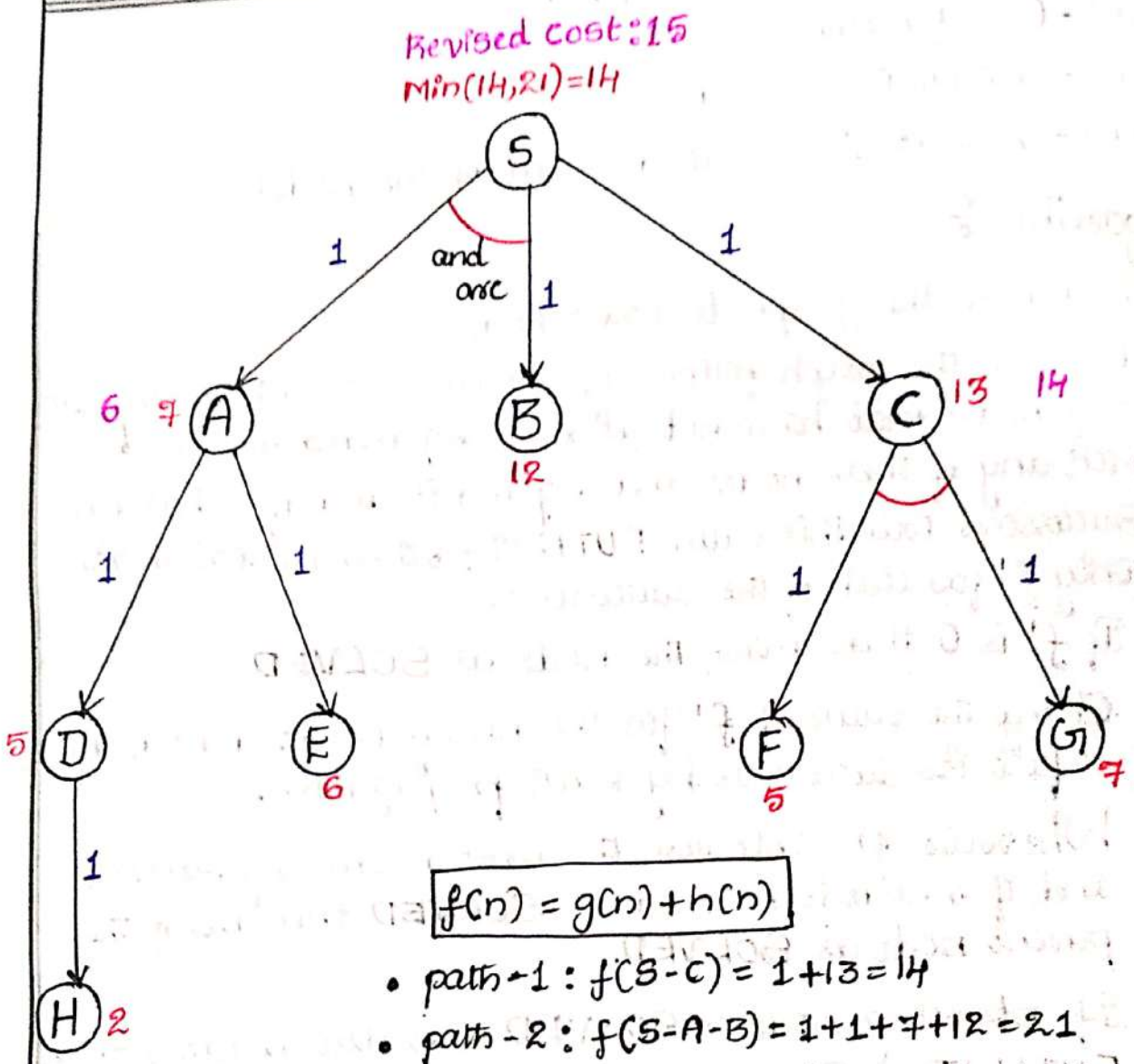
$$f(n) = g(n) + h(n)$$

- $f(n)$ - Cost function.
- $g(n)$ - actual cost or edge value
- $h(n)$ - heuristic/estimated value of the nodes.

## Algorithm:

1. Initialise the graph to start node
2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.
3. Pick any of these nodes and expand it and if it has no successors call this value FUTILITY otherwise calculate only f' for each of the successors.
4. If f' is 0 then mark the node as SOLVED
5. Change the value of f' for the newly created node to reflect its successors by back propagation.
6. Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.
7. If Starting node is SOLVED or value greater than FUTILITY, Stop, else repeat from 2.
8. End the process.

## Illustration of AO* algorithm with example




$$f(n) = g(n) + h(n)$$

- path-1 : $f(S-C) = 1+13 = 14$
- path-2 : $f(S-A-B) = 1+1+7+12 = 21$
- $f(C-F-G) = 1+1+5+7 = 14$
- $f(S-C) = 1+14 = 15$ (revised)

- $f(A-D) = 1+5 = 6$, $f(A-E) = 1+6 = 7$ so revised $f(A) = 6$
- Revised $(S-A-B) = 1+6+1+12 = 20$ ×

## SOURCE CODE

```
Class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph = graph
        self.H = heuristicNodeList
        self.start = startNode
        self.parent = {}
        self.status = {}
        self.solutionGraph = {}

def applyAOStar(self):
    self.aoStar(self.start, False)

def getNeighbours(self, v):
    return self.graph.get(v, "")
def getStatus(self, v):
    return self.status.get(v, 0)
def setStatus(self, v, val):
    self.status[v] = val
def getHeuristicNodeValue(self, n):
    return self.H.get(n, 0)
def setHeuristicNodeValue(self, n, value):
    self.H[n] = value
def printSolution(self):
    print("For graph solution, traverse the graph from the start node:", self.start)
    print("------------------------------------------------------------")
    print(self.solutionGraph)
    print("------------------------------------------------------------")

def computeMinimumCostChildNode(self, v):
    minimumCost = 0
    costToChildNodeListDict = {}
    costToChildNodeListDict[minimumCost] = []
    flag = True

    for nodeInfoTupleList in self.getNeighbours(v):
        cost = 0
        nodeList = []
```

## SOURCE CODE

```
        for C, weight in nodeInfoTripledlist:
            Cost = Cost + self.getHeuristicNodeValue(CC) + weight
            nodelist.append(CC)
        if flag == True:
            minimumCost = Cost
            CostToChildNodeListDict[minimumCost] = nodelist
            flag = False
        else:
            if minimumCost > Cost:
                minimumCost = Cost
                CostToChildNodeListDict[minimumCost] = nodelist
    return minimumCost, CostToChildNodeListDict[minimumCost]

def aoStar(self, v, backTracking):
    print("Heuristic values:", self.H)
    print("Solution Graph:", self.solutionGraph)
    print("processing Node:", v)
    print("-----------------------")

    if self.getStatus(v) >= 0:
        minimumCost, childNodelist = self.computeMinimum
        CostChildNode(v)
        print(minimumCost, childNodelist)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v, len(childNodelist))

        solved = True

        for childnode in childNodelist:
            self.parent[childnode] = v
            if self.getStatus(childnode) != -1:
                solved = solved & False.
```

## SOURCE CODE

```
if solved == True:
    self.setStatus(v, -1)
    self.solutionGraph[v] = childNodeList

if v != self.start:
    self.aoStar(self.parent[v], True)

if backTracking == False:
    for childnode in childNodeList:
        self.setStatus(childnode, 0)
        self.aoStar(childnode, False)
```

```
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7,
      'I': 7, 'J': 1}

graph = {'A': [[('B', 1), ('C', 1)], [('D', 1)]],
         'B': [[('G', 1)], [('H', 1)]],
         'C': [[('J', 1)]],
         'D': [[('E', 1), ('F', 1)]],
         'G': [[('I', 1)]]
    }
G1 = Graph(graph, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

## SOURCE CODE

```
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': H, 'F': H, 'G': 5, 'H': 7}

graph2 = { 'A' : [[('B', 1), ('C', 1)], [('D', 1)]],
           'B' : [[('G', 1)], [('H', 1)]],
           'D' : [[('E', 1), ('F', 1)]]
         }

#G2 = Graph (graph2, h2, 'A')
#G2.applyAOStar()
# G2.printSolution()
```

## OUTPUT

```
Heuristic values : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: A
-------------------
10['B', 'C']
Heuristic values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: B
-------------------
6['G']
Heuristic values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: A
-------------------
10['B', 'C']
Heuristic values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: G
-------------------
8['I']
Heuristic values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: B
-------------------
8['H']
Heuristic values : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: A
-------------------
12['B', 'C']
Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
Solution Graph: {}
processing Node: I
-------------------
0[]
Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}
Solution Graph: {'I': []}
processing Node: G
-------------------
5['B', 'C']
For graph solution, traverse the graph from the start node: A
-------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
```

Program: 3. DATE: 12/11/2021

For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.

## DESCRIPTION

**Version spaces:** Definition: A hypothesis h is consistent with a set of training examples D iff $h(x) = c(x)$ for each example $\langle x, c(x) \rangle$ in D.

The Candidate-Elimination algorithm represents the set of all hypotheses consistent with the observed training examples.

This subset of all hypothesis is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.

Statement: The Version space, denoted $VS_{H,D}$ with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D.

$$VS_{H,D} = \{ h \in H \mid Consistent(h, D) \}$$

**General to specific ordering of hypothesis (with example):**

Example general-to-specific:

Second is less constrained -> it classifies more positive instances

$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$ $h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$

In detail:

- for any instance x in X and hypothesis h in H, we say that x satisfies h if and only if $h(x) = 1$
- Let $h_j$ and $h_k$ be boolean-valued functions defined over X. Then $h_j$ is more-general-than-or-equal-to $h_k$ (written $h_j \geq_g h_k$) if and only if
$$(\forall x \in X)\left[(h_k(x) = 1) \rightarrow (h_j(x) = 1)\right]$$
- Let $h_j$ and $h_k$ be boolean-valued function defined over X. Then $h_j$ is more-general-than $h_k$ (written $h_j >_g h_k$) if and only if $(h_j \geq_g h_k) \wedge \neg (h_k \geq_g h_j)$
- The relations are defined independently of the target concept
- the relation more-general-than-or-equal-to $\geq_g$ defines a partial order over the hypothesis space H
- Informally: there may be pairs of $h_2$ and $h_3$, such that $h_2 \geq_g h_3$ and $h_3 \geq_g h_2$

**List-then-eliminate algorithm**

Version space as list of hypotheses

1. VersionSpace ← a list containing every hypothesis in H
2. For each training example, {x, c(x)}

   Remove from VersionSpace any hypothesis h for which h(x) ≠ c(x)
3. Output the list of hypotheses in VersionSpace.

**Candidate-Elimination algorithm**

* For each training example d, do
  - → If d is a positive example
    - Remove from G any hypothesis inconsistent with d
    - For each hypothesis s in S that is not consistent with d
      - Remove s from S
      - Add to S all minimal generalizations h of s such that
        - h is consistent with d, and some member of G is more general than h
      - Remove from S any hypothesis that is more general than another hypothesis in S
  - → If d is a negative example
    - Remove from S any hypothesis inconsistent with d
    - For each hypothesis g in G that is not consistent with d
      - Remove g from G
      - Add to G all minimal specializations h of g such that
        - h is consistent with d, and some member of S is more specific than h
      - Remove from G any hypothesis that is less general than another hypothesis in G

**Illustration with example**

- S ← maximally general hypotheses in H, G ← maximally general hypothesis in H
  - $S_0 = \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$
  - $G_0 = \langle ?, ?, ?, ?, ?, ? \rangle$

→ Initialize G to the set of maximally general hypotheses in H

→ Initialize S to the set of maximally specific hypotheses in H

Initial values: $S_0 = \{\phi, \phi, \phi, \phi, \phi, \phi\}$ and $G_0 = \{?, ?, ?, ?, ?, ?\}$

| Example | Sky | Temp | Humidity | Wind | Water | Forecast | EnjoySport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | NO |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

Example: Step 1:

after Seeing ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩ +

- $S_0$: ⟨ϕ, ϕ, ϕ, ϕ, ϕ, ϕ⟩
- $S_1$: ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩
- $G_0, G_1$: ⟨?, ?, ?, ?, ?, ?⟩

Step 2:

after Seeing ⟨Rainy, Cold, High, Strong, Warm, Change⟩ −

- $S_2, S_3$: ⟨Sunny, Warm, ?, Strong, Warm, Same⟩
- $G_3$: ⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩ ⟨?, ?, ?, ?, ?, Same⟩
- $G_2$: ⟨?, ?, ?, ?, ?, ?⟩

Step 3:

after Seeing ⟨Sunny, Warm, High, Strong, Cool, Change⟩ +

- $S_3$: ⟨Sunny, Warm, ?, Strong, Warm, Same⟩
- $S_4$: ⟨Sunny, Warm, ?, Strong, ?, ?⟩
- $G_4$: ⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩
- $G_3$: ⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩ ⟨?, ?, ?, ?, ?, Same⟩

✓ The S boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples.

✓ The G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples.

## SOURCE CODE

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('trainingexamples.
                                     csv'))

print('The Dataset is :\n')
print(data)
concepts = np.array(data.iloc[:,0:-1])
print('\n The Concepts are :\n', concepts)
target = np.array(data.iloc[:,-1])
print('\n The target is : \n', target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print(len(specific_h))
    general_h = [["?" for i in range(len(specific_h))] for i in
                 range(len(specific_h))]
    print(general_h)
    for i,h in enumerate(concepts):
        if target[i]=="Y":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    specific_h[x]='?'
                    general_h[x][x]='?'
        if target[i]=="N":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    general_h[x][x]=specific_h[x]
```

## SOURCE CODE

```
        else:
            general_h[x][x] = '?'
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(Concepts, target)
print("\n\n Final S: ", s_final)
print("\n\n Final G: ", g_final)
```

## OUTPUT

The dataset is:

| | Sky | Temperature | humidity | wind | water | forecast | Result |
|---|---|---|---|---|---|---|---|
| 0 | Sunny | warm | Normal | Strong | warm | Same | Y |
| 1 | Sunny | warm | High | Strong | warm | Same | Y |
| 2 | Rainy | Cold | High | Strong | warm | Change | N |
| 3 | Sunny | warm | High | Strong | Cool | change | Y |

The Concepts are:

```
[['Sunny' 'warm' 'Normal' 'Strong' 'warm' 'Same']
 ['Sunny' 'warm' 'High' 'Strong' 'warm' 'Same']
 ['Rainy' 'cold' 'High' 'Strong' 'warm' 'change']
 ['Sunny' 'warm' 'High' 'Strong' 'Cool' 'change']]
```

The target are:

```
['Y' 'Y' 'N' 'Y']
```

```
6
[['?','?','?','?','?','?'], ['?','?','?','?','?','?'], ['?','?','?',
 '?','?','?'], ['?','?','?','?','?','?'], ['?','?','?','?','?','?'],
 ['?','?','?','?','?','?']]
```

Final S:

```
['Sunny' 'warm' '?' 'Strong' '?' 'Same']
```

Final G:

```
[['Sunny','?','?','?','?','?'], ['?','warm','?','?','?','?']]
```

**Program 4.** DATE: 26/11/2021

Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

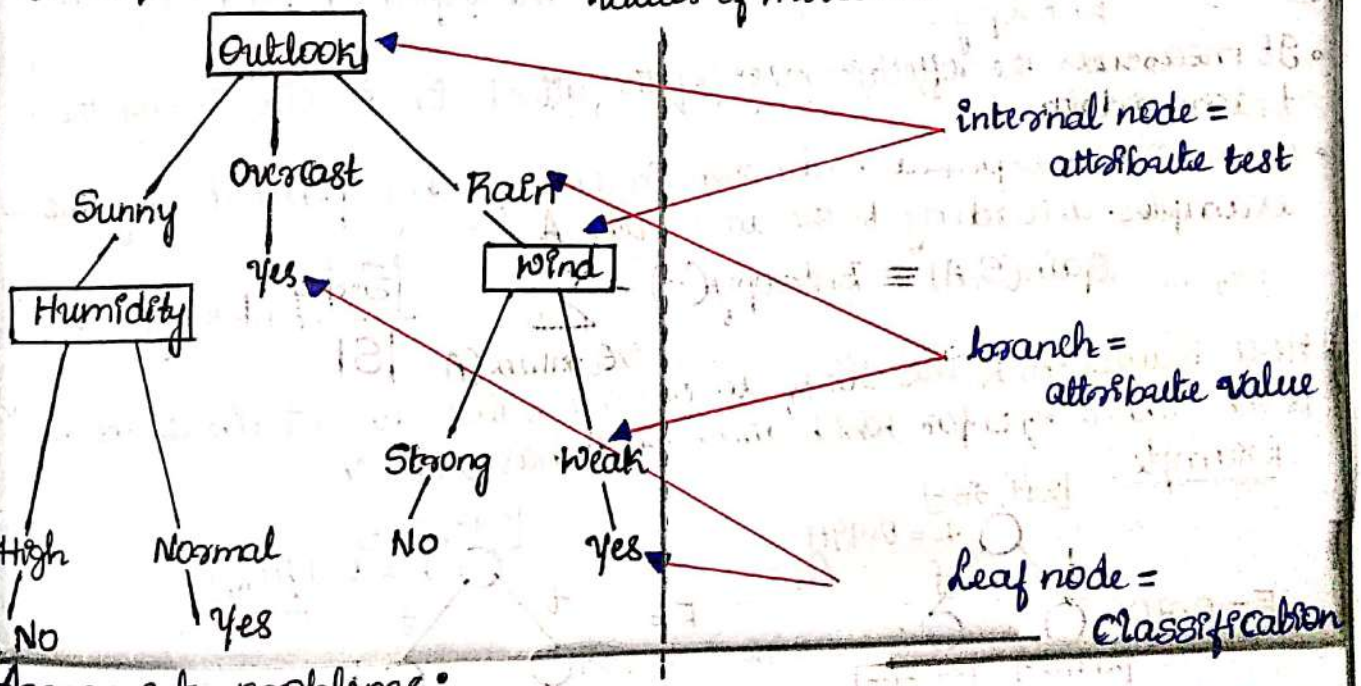
## DESCRIPTION

**Decision tree representation**

- Decision tree learning is one of the most widely used methods for inductive inference
- It is used to approximate discrete valued functions that is robust to noisy data
- It is capable of learning disjunctive expressions.

Representation:

- Each internal node tests an attribute
- Each branch corresponds to an attribute value
- Each leaf node assigns a classification
- An example is classified by sorting it through the tree from the root to the leaf node.
- Example – (Outlook = Sunny, Humidity = High) => (PlayTennis = No)
- In general, DTs represent a disjunction of conjunctions of constraint on the attribute values of instances.




Appropriate problems:

- Instances are represented by attribute-value pairs.
- Target function has discrete output values.
- Disjunctive hypothesis may be required.
- Possibly noisy data * Training data may contain errors.
  * Training data may contain missing attribute values
- Examples – Classification Problems
  * Equipment or medical diagnosis * Credit risk analysis.

**Entropy and Gain (with example)**

## Entropy:

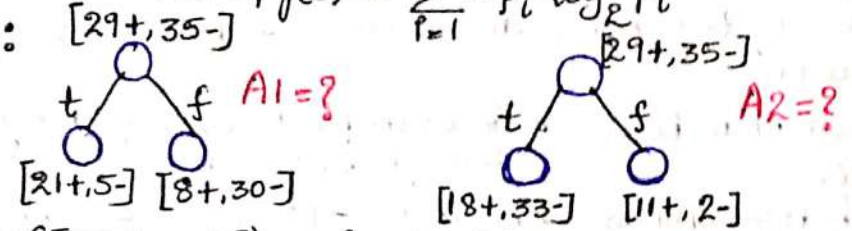
- Entropy(E) is the minimum number of bits needed in order to classify an arbitrary examples as yes or no.
- Entropy is commonly used in information theory. It characterizes the (im)purity of an arbitrary collection of examples.
- S is a sample of training examples
- $P_\oplus$ is the proportion of positive examples in S
- $P_\ominus$ is the proportion of negative examples in S
- then the entropy measures the impurity of S:

$$Entropy(S) \equiv -P_\oplus \log_2 P_\oplus - P_\ominus \log_2 P_\ominus$$

- But if the target attribute can take c different values:

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

Example:




- Entropy([29+,35-]) = -(29/64) log2(29/64) - (35/64) log2(35/64)
  = 0.99H

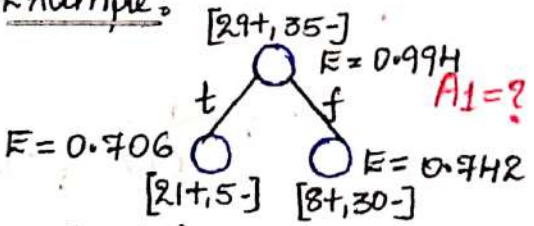
## Gain:

- Information gain measures the expected reduction in Entropy
- It measures the effectiveness of the attribute in classifying the training data
- Gain(S,A) = expected reduction in entropy by partitioning the examples according to the attribute A

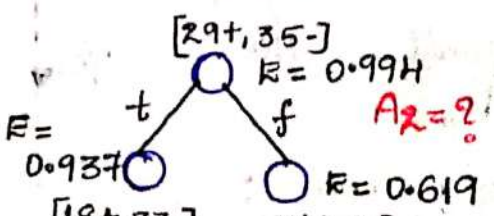
$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Here, values(A) is the set of all possible values for attribute A, $S_v$ is the subset of S for which attribute A has value v

Example:




- Gain(S, A1)
  = 0.99H - (26/64)*.706 - (38/64)* .742
  = 0.266
- Information gained by partitioning along attribute A1 is 0.266




- Gain(S, A2)
  = 0.99H - (51/64)*.937 - (13/64)* .619
  = 0.121
- Information gained by partitioning along attribute A2 is 0.121

## ID3 algorithm

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = −
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples
- Otherwise Begin
  - ⇒ A ← the attribute from Attributes that best* classifies Examples
  - ⇒ The decision attribute for Root ← A
  - ⇒ For each possible value, $v_i$, of A,
    - Add a new tree branch below Root, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of Examples that have value $v_i$ for A.
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
      - Else below this new branch and the subtree ID3($Examples_{v_i}$, Target_attribute, Attributes − {A}))
- End.
- Return Root

## SOURCE CODE

```
import pandas as pd
import numpy as np
import math
class Node:
    def __init__(self, l):
        self.label=l
        self.branch={}

def entropy(data):
    total_ex=len(data)
    p_ex=len(data.loc[data['PlayTennis']=='Yes'])
    n_ex=len(data.loc[data['PlayTennis']=='NO'])
    en=0
    if(p_ex>0):
        en=-(p_ex/float(total_ex))*(math.log(p_ex,2)-
            math.log(total_ex,2))
    if(n_ex>0):
        en+=-(n_ex/float(total_ex))*(math.log(n_ex,2)-
            math.log(total_ex,2))
    return en

def gain(en_S, data_S, attrib):
    values=set(data_S[attrib])
    print(attrib)
    gain=en_S
    for value in values:
        gain-=len(data_S.loc[data_S[attrib]==
value])/float(len(data_S))*entropy(data_S.loc[data_S
[attrib]==value])
    return gain
```

**SOURCE CODE**

```
def get_attr(data):
    en_S=entropy(data)
    attribute=""
    max_gain=0
    for attr in data.columns[:len(data.columns)-1]:
        g=gain(en_S, data, attr)
        if g>max_gain:
            max_gain=g
            attribute=attr
    return attribute

def decision_tree(data):
    root=Node("NULL")
    if(entropy(data)==0):
        if(len(data.loc[data[data.columns[-1]]=="yes"])
            ==len(data)):
            root.label="Yes"
        else:
            root.label="No"
        return root
    if(len(data.columns)==1):
        return
    else:
        attr=get_attr(data)
        root.label=attr
        values=set(data[attr])
        print(attr)
        for value in values:
            root.branch[value]=decision_tree(data.loc[data
[attr]==value].drop(attr,axis=1))
        return root
```

## SOURCE CODE

```
def get_rules(root, rule, rules):
    if not root.branch:
        rules.append(rule[:-1]+"=> "+root.label)
        return rules
    for val in root.branch:
        get_rules(root.branch[val], rule+root.label+"="+
            str(val)+"^", rules)
    return rules

def test(tree, test_str):
    if not tree.branch:
        return tree.label
    print(str(test_str[tree.label]), tree.label, test_str)
    return test(tree.branch[str(test_str[tree.label])],
                test_str)

data = pd.read_csv("tennis.csv")
tree = decision_tree(data)
rules = get_rules(tree, "", [])
for rule in rules:
    print(rule)
test_str = {}
print("Enter the test case input:")
for attr in data.columns[:-1]:
    test_str[attr] = input(attr+":")
print(test_str)
print(test(tree, test_str))
```

## SOURCE CODE

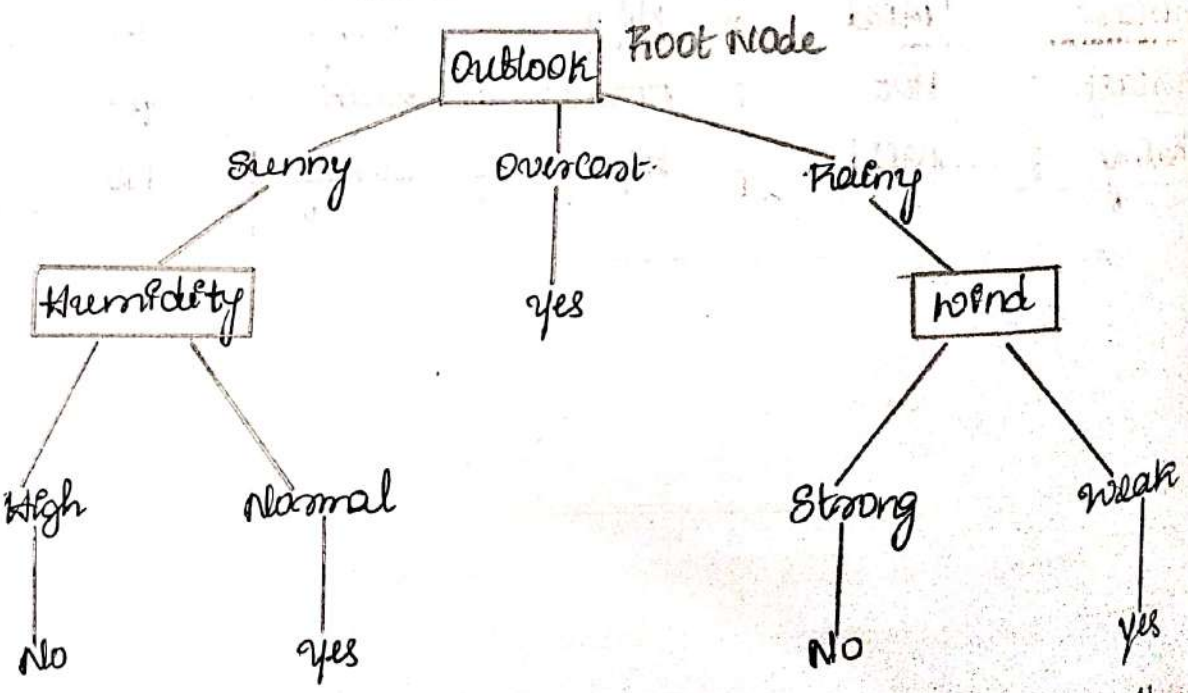
Given Set of Training data examples Stored in a .CSV file

| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | weak | No |
| Sunny | Hot | High | Strong | No |
| OverCast | Hot | High | weak | Yes |
| Rainy | Mild | High | weak | Yes |
| Rainy | Cool | Normal | weak | Yes |
| Rainy | Cool | Normal | Strong | No |
| OverCast | Cool | Normal | Strong | Yes |
| Sunny | Mild | High | weak | No |
| Sunny | Cool | Normal | weak | Yes |
| Rainy | Mild | Normal | weak | Yes |
| Sunny | Mild | Normal | Strong | Yes |
| OverCast | Mild | High | Strong | Yes |
| OverCast | Hot | Normal | weak | Yes |
| Rainy | Mild | High | Strong | No |

## OUTPUT

Outlook = Sunny ^ Humidity = Normal => yes
Outlook = Sunny ^ Humidity = High => no
Outlook = Overcast => yes
Outlook = Rainy ^ Windy = Strong => no
Outlook = Rainy ^ Windy = Weak => yes
Enter the test case input:
Outlook : Sunny
Temperature : Hot
Humidity : High
Windy : Weak
{'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Windy': 'Weak'}
Sunny Outlook {'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Windy': 'Weak'}
High Humidity {'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Windy': 'Weak'}
No

### Final Decision Tree



DATE: 17/12/2021

**Program 5.**

Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.

## DESCRIPTION

**Artificial neural networks**

Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements (neurons) working in unison to solve a specific problem.

Applications of ANN:

* Speech recognition
* Human face recognition
* Financial prediction
* Character recognition
* Image classification

**Neural network representation with diagram**

Prototypical example of ANN learning:

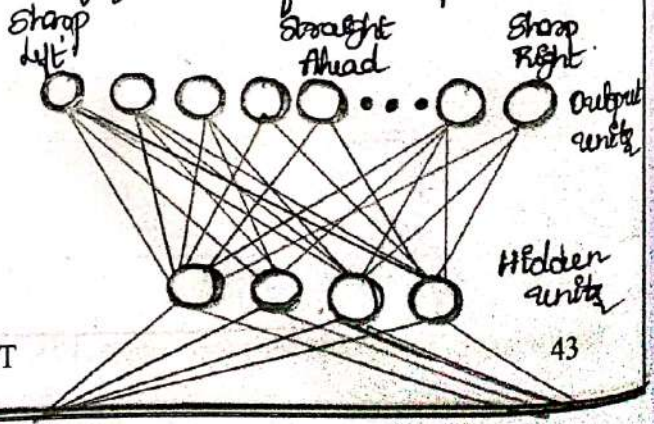
- Pomerleau's (1993) proposed the system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways.

Inputs:

- 30 x 32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.

output:

- Direction in which the vehicle is steered.
- ALVINN has used ANN to successfully drive at speeds up to 70 miles per hour and for distances of 90 miles on public highways.
- The network is shown on the left side of the figure, with the input camera image depicted below it.
- Each node (i.e., circle) in the network diagram corresponds to the output of a single network unit, and the lines entering the node from below are its inputs.
- Four units that receive inputs directly from all of 30 x 32 pixels in the image are called "hidden" units.
- Hidden units computes a single real-valued output based on a weighted combination of the 960 inputs.
- Hidden unit outputs are then used as inputs to a second layer of 30 "output" units.



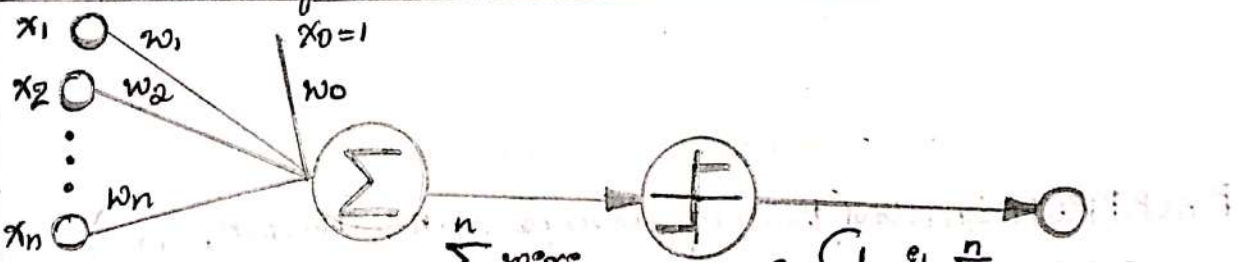
## Perceptron and its representation

- Perceptron is a single layer neural network and is a linear classifier.
- Perceptron takes a vector of real-valued inputs, calculates a (binary) linear combination of these input.
- If the linear combination(Sum) of the inputs is greater than certain threshold then outputs a '1', else a '-1'.
- Given inputs $x1 \ldots xn$ and output $O(x1 \ldots xn)$ can be shown as.

$$O(x1 \ldots xn) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- $w_i$ is a real-valued constant, or weight, that determines the contribution of input $x_i$ to the perceptron output.
- $(-w_0)$ is a threshold.
- perceptron function can be written as $O(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$

  where $\text{sgn}(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise.} \end{cases}$
- The perceptron consists of 4 parts.
  - Input values or One input layer.
  - Weights and Bias.
  - Net Sum
  - Activation function




## Gradient descent and the delta rule

- Determines a weight vector that minimizes E
- The weight vector is modified in small steps after every iteration
- The weight vector is altered in the direction that produces steepest descent along the error surface.
- Continue until the global minimum of the error is reached
- Every iteration is termed as Epoch.

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

## Delta Rule:

- If the training examples are not linearly separable; the delta rule converges toward a best-fit approximation to the target concept.
- Delta rule uses gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- Training error of the weight vector is defined as

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

D → is the set of training examples.
$t_d$ → is the target output for training example d
$o_d$ → is the output of the linear unit for training example d.

**Sigmoid unit**

- It is mostly used for multi-class classification. The Sigmoid or logistic activation function maps the input to the output range between (0,1).
- The Sigmoid unit computes a linear combination of its inputs, and applies a threshold
- The threshold output is continuous function of its input.
- Output o is computed using,

$$o = \sigma(\vec{w} \cdot \vec{x}) \text{ where } \sigma(y) = \frac{1}{1+e^{-y}}$$

$\sigma$ is often called the Sigmoid function

- Since it maps large output domain into small range of outputs, it is referred to as the Squashing function.
- Its derivate is easily expressed in terms of its output

$$\boxed{\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))}$$

**Back propagation algorithm**

BACKPROPAGATION(training_examples, $\eta$, $n_{in}$, $n_{out}$, $n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

$\eta$ is the learning rate (e.g., .05). $n_{in}$ is the number of network inputs, $n_{hidden}$ the number of units in the hidden layer, and $n_{out}$ the number of output units.

The input from unit i into unit j is denoted $x_{ji}$, and the weight from unit i to unit j is denoted $w_{ji}$.

* Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.
* Initialize all network weights to small random numbers (e.g., between −.05 and .05).
* Until the termination condition is met, Do

→ For each $\langle \vec{x}, \vec{t} \rangle$ in training_examples, Do

propogate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output $O_u$ of every unit $u$ in the network.

Propogate the errors backward through the network:

2. For each network output unit $k$, Calculate its error term $\delta_k$

$$\delta_k \leftarrow O_k(1 - O_k)(t_k - O_k)$$

3. For each hidden unit $h$, Calculate its error term $\delta_h$

$$\delta_h \leftarrow O_h(1 - O_h) \sum_{k \in \text{outputs}} w_{kh} \delta_k$$

4. Update each network weight $w_{ji}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where,

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

## SOURCE CODE

```
from math import exp
from random import seed
from random import random

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs+1)]} for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden+1)]} for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
```

## SOURCE CODE

```
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs.

# Calculate the derivate of an neuron output.
def transfer_derivative(output):
    return output*(1.0-output)

# Backpropagate error and store in neurons.
def backward_propogate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors=list()
        if i!=len(network)-1:
            for j in range(len(layer)):
                error =0.0
                for neuron in network[i+1]:
                    error+=(neuron['weights'][j]*neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron=layer[j]
                errors.append(expected[j]-neuron['output'])
        for j in range(len(layer)):
            neuron=layer[j]
            neuron['delta']=errors[j]*transfer_derivative(
                                        neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs=row[:-1]
        if i!=0:
            inputs=[neuron['output'] for neuron in network[i-1]]
        for neuron in network[i]:
```

## SOURCE CODE

```
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] *
                                                    inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

#Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            #print(expected)
            sum_error += sum([(expected[i]-outputs[i])**2 for i in
            range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate,
                                                    sum_error))

#Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3,39651688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
```

## SOURCE CODE

```
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

## OUTPUT

```
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.8508873254395114,
 1.0858178629550297], 'output': 0.029980305604426185,
 'delta': -0.005954660416232625},
{'weights': [0.3771109814246215, -0.06259089445529894,
 0.2765123702642716], 'output': 0.9456229000211323, 'delta':
 0.0026279652850863837}]

[{'weights': [2.515394693978449, -0.3391927502445985,
 -0.9671565426390275], 'output': 0.2364879420235875,
 'delta': -0.04274005927836458},

{'weights': [-2.5584141984844263, 1.0036422106209202,
 0.42883086467582715], 'output': 0.7790535202438367,
 'delta': 0.03803131325964373549}]
```

**PROGRAM 6**

DATE: 31/12/2021

Write a program to implement the **Naive Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

## DESCRIPTION

**Conditional probability**

P(A|B) is the probability of event A occuring, given that event B occurs. For example, given that you drew a red card, what's the probability that it's a four (P(four|red)) = 2/26 = 1/13. So, out of the 26 red cards (given a red card), there are two fours so 2/26 = 1/13.

i.e, P(B|A) = P(A and B) / P(A)

or

P(B|A) = P(A∩B) / P(A)

where,

P = probability,
A = Event A and
B = Event B.

* Conditional probability is defined as the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome.
* It is calculated by multiplying the probability.

**Bayes theorem and concept learning**

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data.
- We can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.
- This section considers a brute-force Bayesian concept learning algorithm, then compares it to concept learning algorithms we considered.
- One interesting result of this comparison is that under certain conditions several algorithms discussed earlier output the same hypothesis as this brute-force Bayesian algorithm, despite the fact that they do not explicitly manipulate probabilities and are considerably more efficient.

## Maximum likelihood hypothesis

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D (or at least one of the maximally probable if there are several).
- Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.
- We can determine the MAP hypothesis by using Bayes theorem to calculate the posterior probability of each candidate hypothesis.

Choosing Hypotheses.

Find most probable hypothesis given training data

Maximum a posteriori hypothesis $h_{MAP}$:

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

P(D) is a constant independent of h.

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Assuming $P(h_i) = P(h_j)$ we can further simplify, and choose the maximum likelihood (ML) hypothesis

$$h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$$

P(D|h) is called the likelihood of D given h and any hypothesis that maximizes this is the maximum likelihood h.

## Naive Bayesian classifier algorithm

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

Working:

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

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

## SOURCE CODE

```
import pandas as pd
#reading the dataset
data = pd.read_csv('data.csv')
#Calculating the total no., no. of positive and no. of negative instances
te = len(data)
np = len(data.loc[data[data.columns[-1]] == 'Yes'])
nn = te-np
# dividing the dataset into training and test
training = data.sample(frac=0.75, replace=False)
# test = pd.concat([data, training, training]).drop_duplicates(keep=False)
test = pd.concat([data, training]).drop_duplicates(keep=False)
print('Training Set: \n', training)
print('\n Test Data Set: \n', test)
#For every value of each attribute calculate the negative and positive probability
prob = {}
for col in training.columns[:-1]:
    prob[col] = {}
    vals = set(data[col])
    for val in vals:
        temp = training.loc[training[col] == val]
        pe = len(temp.loc[temp[temp.columns[-1]] == 'Yes'])
        ne = len(temp) - pe
        prob[col][val] = [pe/np, ne/nn]
#Using Bayes Theorem to predict the output
prediction = []
right_prediction = 0
for i in range(len(test)):
    row = test.iloc[i, :]
    fpp = np/te
    fpn = nn/te
    for col in test.columns[:-1]:
```

## SOURCE CODE

```
        fpp* = prob[col][row[col]][0]
        fpn* = prob[col][row[col]][1]
    if fpp>fpn:
        prediction.append('Yes')
    else:
        prediction.append('NO')
    if prediction[-1]==row[-1]:
        right_prediction+=1
#Output
print('\n Actual Values:', list(test[test.columns[-1]]))
print('Predicted:', prediction)
print('Accuracy:', right_prediction/len(test))
```

## OUTPUT

Training Set:

| | Outlook | Temperature | Humidity | Windy | playTennis |
|---|---|---|---|---|---|
| 5 | Rainy | Cool | Normal | Strong | No |
| 7 | Sunny | Mild | High | weak | No |
| 10 | Sunny | Mild | Normal | Strong | Yes. |
| 13 | Rainy | Mild | High | Strong | No. |
| 1 | Sunny | Hot | High | Strong | No |
| 3 | Rainy | Mild | High | weak | Yes. |
| 0 | Sunny | Hot | High | weak | No |
| 9 | Rainy | Mild | Normal | weak | Yes. |
| 11 | Overcast | Mild | High | Strong | Yes. |
| 12 | Overcast | Hot | Normal | weak | Yes. |

Test Data Set:

| | Outlook | Temperature | Humidity | Windy | playTennis |
|---|---|---|---|---|---|
| 2 | Overcast | Hot | High | weak | Yes. |
| 4 | Rainy | Cool | Normal | weak | Yes. |
| 6 | Overcast | Cool | Normal | Strong | Yes. |
| 8 | Sunny | Cool | Normal | weak | Yes. |

Actual values: ['Yes', 'Yes', 'Yes', 'Yes']

Predicted: ['Yes', 'NO', 'NO', 'NO']

Accuracy: 0.25

**PROGRAM 7**

**DATE:** 7/1/2022.

Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

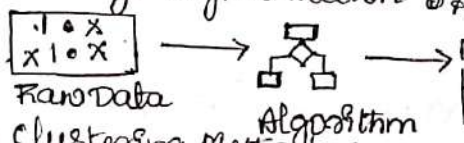
## DESCRIPTION

### Clustering techniques

"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."

The clustering technique is commonly used for statistical data analysis.

Some most common uses of the techniques are: • Market Segmentation • Statistical data analysis • Social Network analysis • Image Segmentation • Anamoly detection, etc.




Types of clustering Methods:

* Partitioning clustering * Distribution Model-Based clustering * Density-Based clustering * Hierarchical & * Fuzzy clustering

### k-Means algorithm

Algorithm K-Means (K number of clusters, D list of data points)

1. Choose K number of random data points as initial Centroids. (Cluster Centers).
2. Repeat till cluster centers stabilize:
   a. Allocate each point in D to the nearest of K Centroids.
   b. Compute centroid for the cluster using all points in the cluster.

**Expectation Maximization algorithm**

- EM Algorithm: Pick random initial $h = \langle \mu_1, \mu_2 \rangle$ then iterate

E step: Calculate the expected value $E[z_{ij}]$ of each hidden variable $z_{ij}$, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

$$E[z_{ij}] = \frac{P(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} P(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

M Step: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable $z_{ij}$ is its expected value $E[z_{ij}]$ calculated above. Replace $h = \langle \mu_1, \mu_2 \rangle$ by $h' = \langle \mu'_1, \mu'_2 \rangle$.

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \cdot x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

- Converges to local maximum likelihood $h$ and provides estimates of hidden variables $z_{ij}$.
- In fact, local maximum on $E[\ln P(Y|h)]$
  - $Y$ is complete (observable plus unobservable variables) data
  - Expected value is taken over possible values of unobserved variables in $Y$.

## SOURCE CODE

### K-means:

```
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import pandas as pd.
import numpy as np.
from sklearn.cluster import KMeans
iris=datasets.load_iris()
x=pd.DataFrame(iris.data)
x.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']
plt.figure(figsize=(14,7))
colormap= np.array(['red','lime','black'])
plt.subplot(1,2,1)
plt.scatter(x.Sepal_Length,x.Sepal_Width, c=colormap[y.Targets], s=40)
plt.title('Sepal')
plt.subplot(1,2,2)
plt.scatter(x.Petal_Length, x.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Petal')
model= KMeans(n_clusters=3)
model.fit(x)
plt.figure(figsize=(14,7))
colormap= np.array(['red','lime','black'])
plt.subplot(1,2,1)
plt.scatter(x.Petal_Length, x.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real')
plt.subplot(1,2,2)
plt.scatter(x.Petal_Length, x.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('Kmeans')
```

## SOURCE CODE

```
print("accuracy_score", accuracy_score(y.Targets, model.labels_))
print("confusion_matrix\n", confusion_matrix(y.Targets, model.labels_))
```

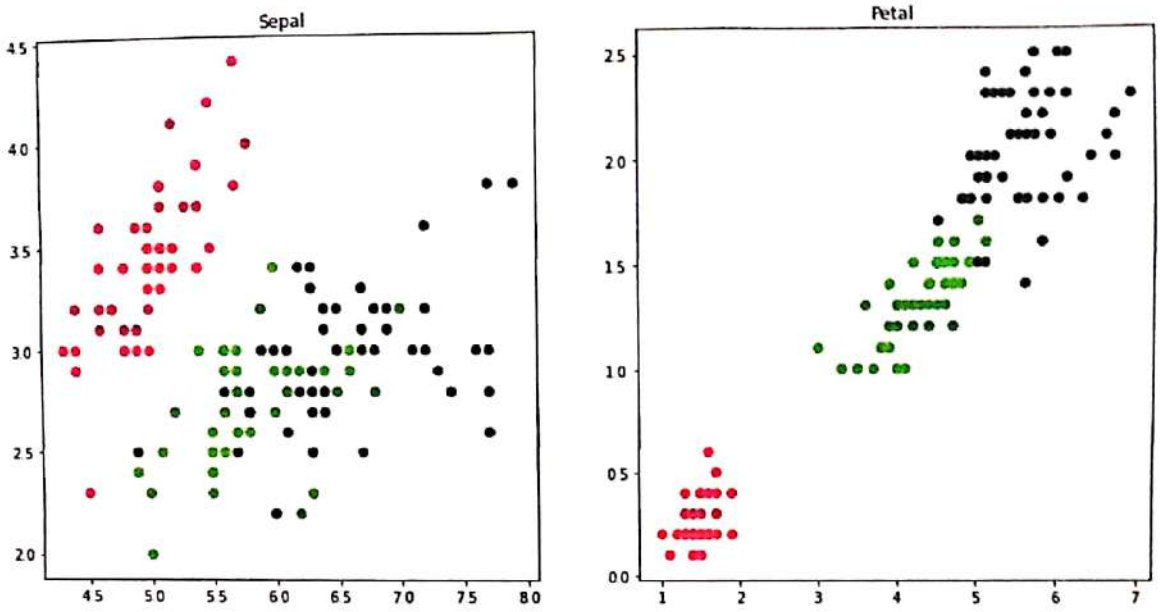
### EM Program:

```
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import pandas as pd.
import numpy as np
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
iris = datasets.load_iris()
x = pd.DataFrame(iris.data)
x.columns = ['Sepal_length', 'Sepal_width', 'petal_length', 'Petal_width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
plt.figure(figSize = (14, 7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(x.Sepal_length, x.Sepal_width, c = colormap[y.Targets], s=40)
plt.title('Sepal')
plt.subplot(1, 2, 2)
plt.scatter(x.Petal_length, x.petal_width, c = colormap[y.Targets], s=40)
plt.title('Petal')
scaler = preprocessing.StandardScaler()
scaler.fit(x)
xsa = scaler.transform(x)
xs = pd.DataFrame(xsa, columns = x.columns)
gmm = GaussianMixture(n_components = 3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
plt.figure(figSize = (14, 7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(x.petal_length, x.petal_width, c=colormap[y.Targets], s=40)
plt.title('Real')
plt.subplot(1, 2, 2)
plt.scatter(x.petal_length, x.petal_width, c=colormap[y_gmm], s=40)
plt.title('EM')
print("accuracy_score", accuracy_score(y.Targets, y_gmm))
print("confusion_matrix\n", confusion_matrix(y.Targets, y_gmm))
```

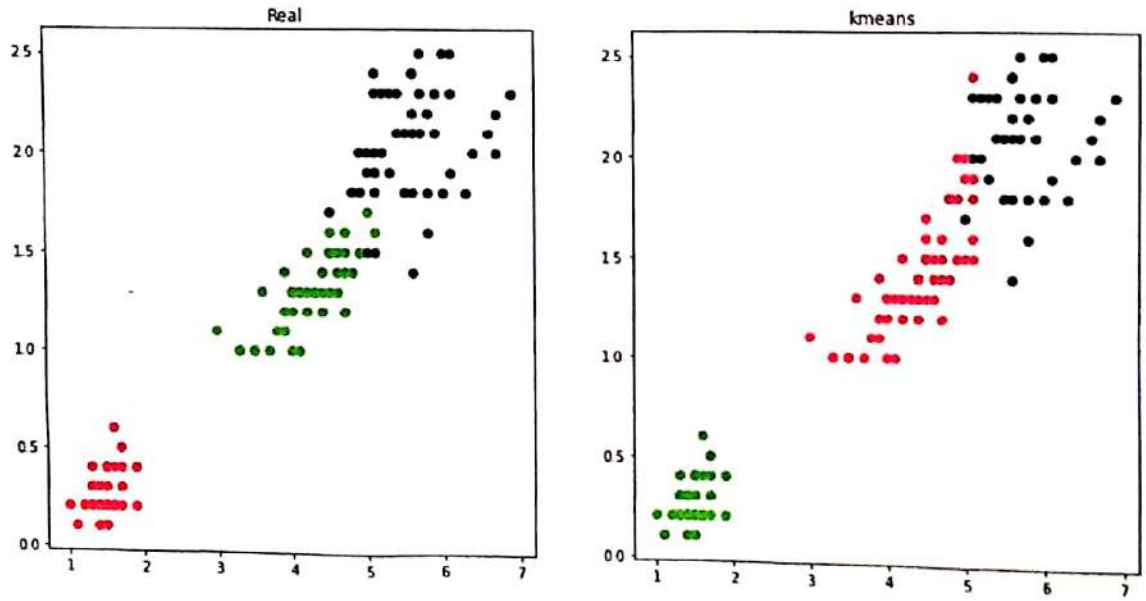
# OUTPUT

## k-Means Algorithm

Text (0.5, 1.0, 'Petal')




Text (0.5, 1.0, 'kmeans')




accuracy_score 0.093333333333334

confusion_matrix

[[ 0 50 0]

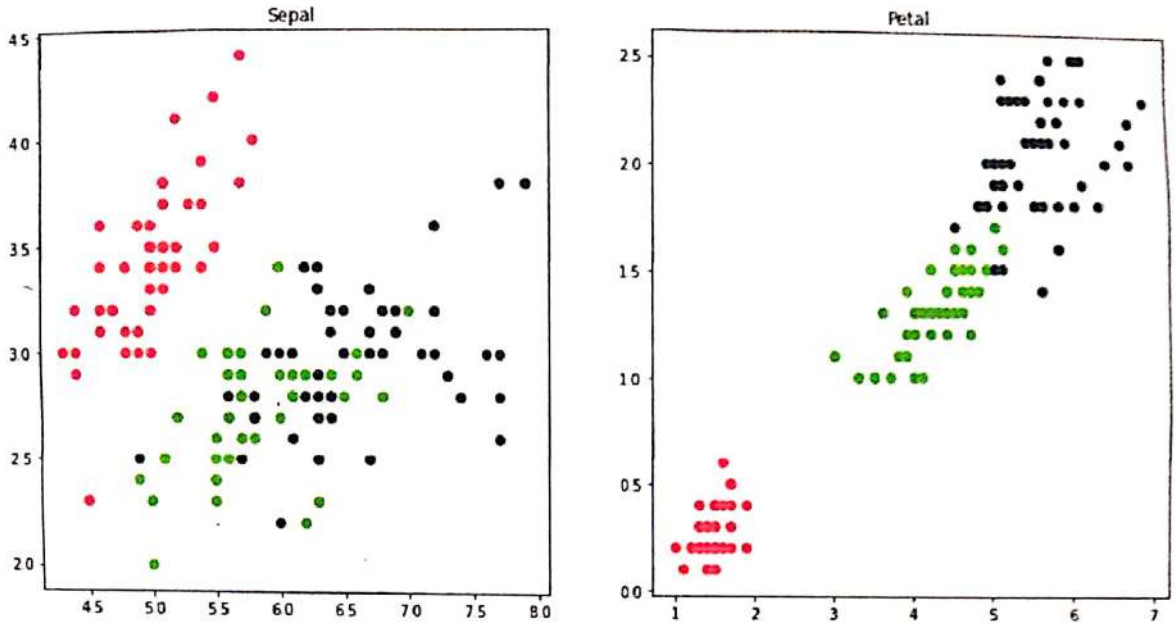
[48 0 2]

[14 0 36]]

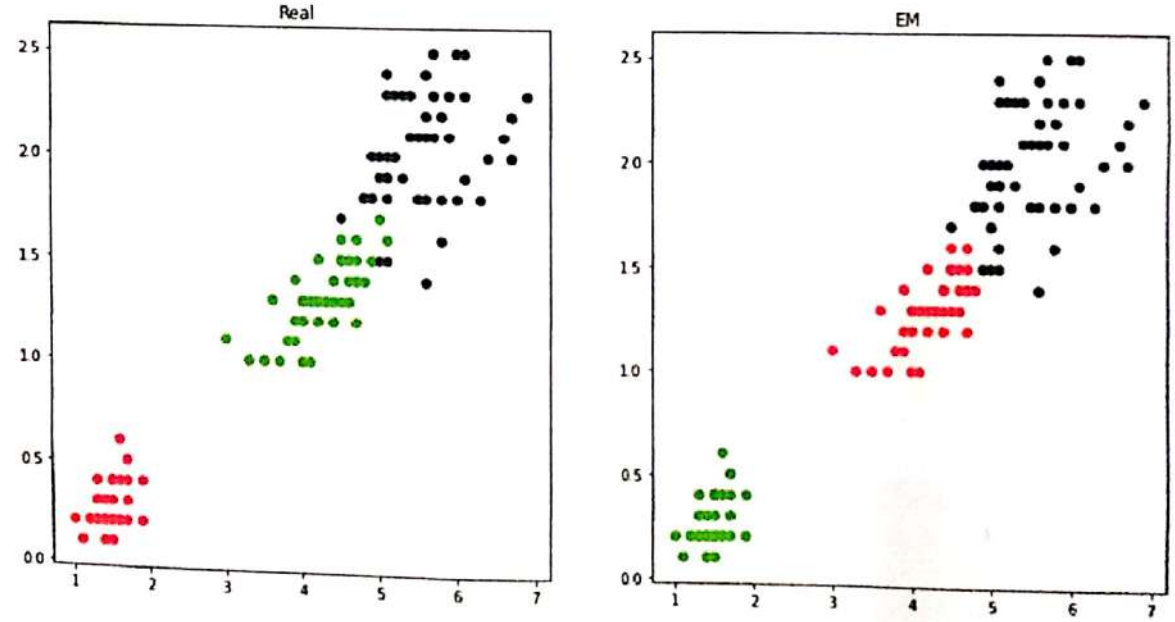
# OUTPUT

## EM Algorithm

Text (0.5, 1.0, 'Petal')




Text (0.5, 1.0, 'EM')




accuracy_score 0.96666666666666667

confusion_matrix

[[ 0 50 0]

[45 0 5]

[ 0 0 50]]

**PROGRAM 8** DATE: 4/01/2022

Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

## DESCRIPTION

### Instance Based Learning

- Instance based learning simply stores the training examples instead of learning explicit description of target function.
- Generalizing examples is postponed till a new example must be classified.
- Relationship between previously stored examples and new query is examined to assign target value to the new instance.
- Includes nearest neighbour, locally weighted regression, case based reasoning methods.
- Can construct a different approximation to the target function for each distinct query instance that must be classified.

Advantages:
* Can use complex symbolic representations for instances.
* Training is very fast.
* Doesn't lose information.

Disadvantages:
* The cost of classifying new instances can be high as nearly all computation takes place at classification time.

### IRIS dataset

- This data sets consists of 3 different types of irises' (- Setosa
  - Versicolour and
  - Virginica) petal and sepal length, stored in a 150×4 numpy.ndarray.
- The rows being the samples and the columns being: Sepal Length, Sepal width, petal length and petal width.
- Load and return the iris dataset(classification).
- The iris dataset is a classic and very easy multi-class classification dataset.

## k-Nearest Neighbour algorithm (with illustration)

- KNN algorithm is a type of Supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well - * Lazy learning Algorithm
  * Non-parametric learning Algorithm
- The most basic instance-based method is the k-Nearest Neighbor learning.
- It assumes all instances correspond to points in the n-dimensional space $\mathbb{R}^n$.
- The nearest neighbours of an instance are defined in terms of the standard Euclidean distance.
- Euclidean distance
  - Calculates the distance between two points (your new sample and all the data you have in your dataset)
  - Let an arbitrary instance x be described by the feature vector $(a_1(x), a_2(x), \ldots, a_n(x))$ where $a_r(x)$ denotes the value of the rth attribute of instance x.
  - The distance between two instances $x_i$ and $x_j$ is defined to be $d(x_i, x_j)$

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

**Training Algorithm:**

- For each training example $\langle x, f(x) \rangle$, add the example to the list training_examples.

**Classification Algorithm:**

- Given a query instance $x_q$ to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training_examples that are nearest to $x_q$.
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a,b) = 1$ if $a = b$ and where $\delta(a,b) = 0$ otherwise.

## SOURCE CODE

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris.

iris_dataset = load_iris()
print("\n IRIS FEATURES \ TARGET NAMES : \n", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
        print("\n [{0}]: [{1}]".format(i, iris_dataset.target_names[i]))
x_train , x_test, y_train , y_test = train_test_split(iris_dataset["data"],
 iris_dataset["target"], random_state=0)
kn = KNeighborsClassifier(n_neighbors=4)
kn.fit(x_train, y_train)
i=1
x = x_test[i]
x_new = np.array([x])
for i in range(len(x_test)):
      x =x_test[i]
      x_new = np.array([x])
       prediction =kn.predict(x_new)
      print("\n Actual: {0} {1}, Predicted: {2} {3}".format
       (y_test[i], iris_dataset.target_names[y_test[i]], prediction,
       iris_dataset.target_names[prediction]))
print("\n TEST SCORE [ACCURACY]: {:.2f} \n".format(kn.score(x_test,
                                                    y_test)))
```

**Output**

```
IRIS FEATURES \TARGET NAMES:
   ['setosa' 'versicolor' 'virginica']
[0]: [Setosa]
[1]: [versicolor]
[2]: [virginica]
TEST SCORE[ACCURACY]: 0.97
```

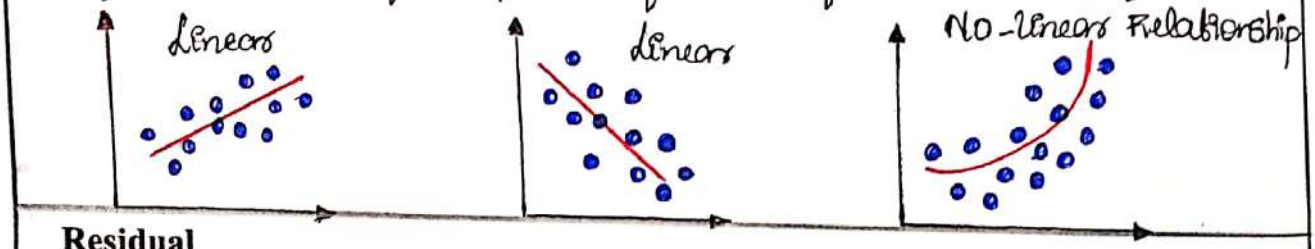
**PROGRAM 9** **DATE: 14/01/2022**

Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

## DESCRIPTION

**Regression**

- Regression means approximating a real-valued target function.
- Regression analysis consists of a set of machine learning methods that allow us to predict a continuous outcome variable ($y$) based on the value of one or multiple predictor variables ($x$).
- The goal of regression model is to build a mathematical equation that defines $y$ as a function of the $x$ variables.




**Residual**

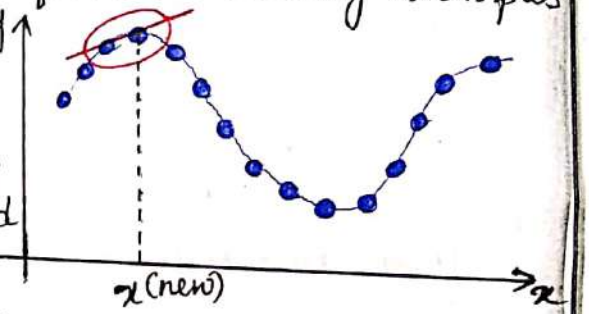
- Residual is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- It is the difference between the two sides of the equation without the expected value, evaluated after just a single transition for $(x, u)$.

**Kernel function**

- Kernel function is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function $K$ such that. $w_i = K(d(x^i, x_q))$.

## Locally Weighted Regression algorithm (with illustration)

- Locally weighted regression (LWR) attempts to fit the training data only in a region around the location of a query example.
- points are weighted by proximity to the current x in question using a kernel. A regression is then computed using the weighted points.
- Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f.
- Consider the diagram, blue dots represent training examples. x (new) is the test point and value of y should be predicted. Using weighting concept only nearby points are considered for prediction.
- Given a new query instance $x_q$, locally weighted regression constructs an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding $x_q$.
- Further calculates value of $\hat{f}(x_q)$.




### Locally weighted linear Regressions:

* Consider target function f approximated near $x_q$ using a linear function of the form,

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \dots + w_n a_n(x)$$

* Coefficients $w_0 \dots w_n$ should be chosen such that it minimizes the error in fitting linear functions to a given set of training examples

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

* The squared error summed over the set D of training examples is given by.

$$E = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

according to gradient descent

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x).$$

* we have to redefine the error criterion E to emphasize fitting the local training examples.
* Three possible criteria are given below.

- Minimize the squared error over just the K-nearest neighbours:

$$E_1(x_q) = \frac{1}{2} \sum_{x \in K \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

* Minimize the squared error

$$E_2(x_q) = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

* Combing both, $E_3(x_q) = \frac{1}{2} \cdot \sum (f(x) - \hat{f}(x))^2 K(d(x_q, x))$
* Weight update rule is given by,

$$\Delta w_j = \eta \cdot \sum_{x \in K \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

## SOURCE CODE

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) #eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
def localWeight(point, xmat, ymat, k):
    wei = kernel(point, xmat, k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    n=np.shape(wei)
    #print(n)
    return W
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    #print(m)
    #print(n)
    ypred=np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
def graphPlot(X, ypred):
    sortIndex = X[:,1].argsort(0)
    xSort=X[sortIndex][:,0]
    fig=plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill, tip, color='green')
    ax.plot(xSort[:,1], ypred[sortIndex], color='red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
```

## SOURCE CODE

```
#Load data points.
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill).
tip = np.array(data.tip)
mbill = np.mat(bill)
mtip = np.mat(tip)
m = np.shape(mbill)[1]
#print(m)
one = np.mat(np.ones(m))
#print(one)
X = np.hstack((one.T, mbill.T)).
#print(X)
ypred = localWeightRegression(X, mtip, 1.5)
graphPlot(X, ypred)
```

## Output