NCR CAMPUS, MODINAGAR

(A Constituent of SRM University, Chennai T.N.)

Delhi-Meerut Road, Sikari Kalan, Modinagar – 201204, GHAZIABAD (U.P.)

ARTIFICIAL INTELLIGENCE LAB

Subject Code:18CSC305J

Lab Record

(Jan-May 2023)

Name of Student	**:**
Reg. No.	**:**
Degree/Branch	**: B. Tech / CSE**
Semester/Section	**: 6th/ J**
Course Code	**: 18CSC305J**
Course Title	**: Artificial Intelligence Lab**
Faculty In charge.	**: Ms. Anjali Yadav**

Department of Computer Science and Engineering

SRM IST, DELHI-NCR CAMPUS, MODINAGAR

Department Of Computer Science and Engineering

REGISTRATION NO.

R	A	2	0	1	1	0	0	3	0	3	0			

BONAFIDE CERTIFICATE

It is to be certified that the bonfide practical record submitted by ________________________________of 6th semester for Bachelor of Technology degree in the Department of Computer Science and Engineering, Delhi-NCR Campus, SRM IST has been done for the course **Artificial Intelligence Lab (18CSC305J)** during the academic semester session January 2023 – May 2023.

Ms. Anjali Yadav

Assistant Professor
Computer Science & Engg.

Dr. R. P. Mahapatra

Head of the Department
Computer Science & Engg.

Submitted for the University Examination held on _____________

Examiner 1

Examiner 2

INDEX

EXPERIMENT - 1

Aim:- Implementation of Toy problem Example-Implement water jug problem.

Algorithm:-

Rule	State	Process
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}
5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

Program:-

```
print("Water jug problem")
x=int(input("Enter X : "))
y=int(input("Enter Y : "))
while True:
    rn=int(input("Enter the rule no. : "))
    if rn==2:
        if y<3:
```

```
        x=0
        y=3
if rn==3:
    if x>0:
        x=0
        y=3
if rn==5:
    if x+y>4:
        x=4
        y=y-(4-x)
if rn==7:
    if x+y<4:
        x=x+y
        y=0
if rn==9:
    x=2
    y=0
print("X=",x)
print("Y=",y)
if x==2:
    print("The result is a goal state")
    break
```

Output:-

```
PS C:\Users\rohan\OneDrive\Documents\C-C++> python -u "c:\Users\rohan\OneDrive\Documents\C-C
++\test.py"
Water jug problem
Enter X : 0
Enter Y : 0
Enter the rule no. : 3
X= 0
Y= 0
Enter the rule no. : 2
X= 0
Y= 3
Enter the rule no. : 3
X= 0
Y= 3
Enter the rule no. : 5
X= 0
Y= 3
Enter the rule no. : 7
X= 3
Y= 0
Enter the rule no. : 9
X= 2
Y= 0
The result is a goal state
PS C:\Users\rohan\OneDrive\Documents\C-C++>
```

RESULT:- Water Jug Problem have been successfully implemented.

EXPERIMENT - 2

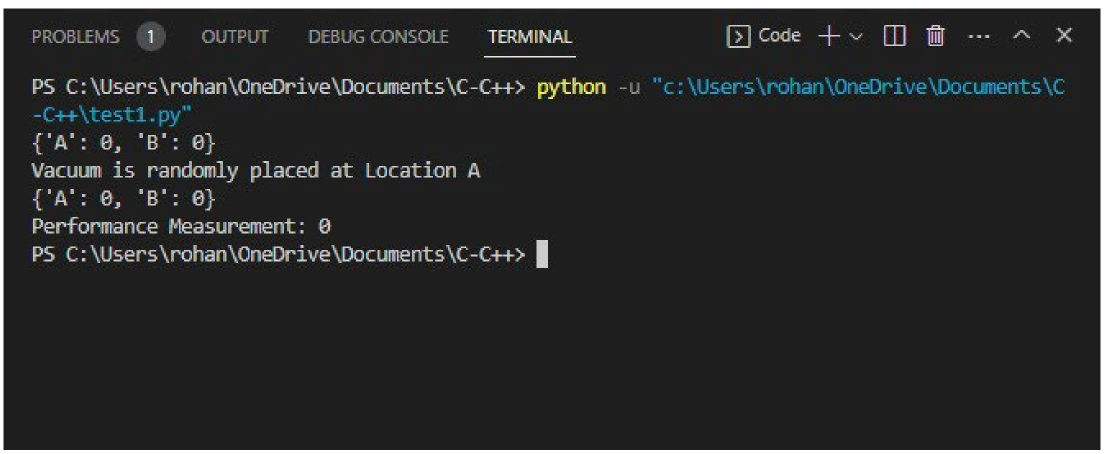
Aim:- Developing Agent Program for Real World Problem.

Program:-

```
import random
class Environment(object):
    def__init_(self):
        self.locationCondition = {'A': '0', 'B': '0'}
        self.locationCondition['A'] = random.randint(0, 1)
        self.locationCondition['B'] = random.randint(0, 1)
class SimpleReflexVacuumAgent(Environment):
    def__init_(self, Environment):
        print (Environment.locationCondition)
        Score = 0
        vacuumLocation = random.randint(0, 1)
        if vacuumLocation == 0:
            print ("Vacuum is randomly placed at Location A")
            if Environment.locationCondition['A'] == 1:
                print ("Location A is Dirty. ")
                Environment.locationCondition['A'] = 0;
                Score += 1
                print ("Location A has been Cleaned. :D")
                if Environment.locationCondition['B'] == 1:
                    print ("Location B is Dirty.")
                    print ("Moving to Location B...")
                    Score -= 1
                    Environment.locationCondition['B'] = 0;
                    Score += 1
                    print ("Location B has been Cleaned :D.")
            else:
                if Environment.locationCondition['B'] == 1:
                    print ("Location B is Dirty.")
                    Score -= 1
                    print ("Moving to Location B...")
                    Environment.locationCondition['B'] = 0;
                    Score += 1
                    print ("Location B has been Cleaned. :D")
        elif vacuumLocation == 1:
            print ("Vacuum is randomly placed at Location B. ")
            if Environment.locationCondition['B'] == 1:
                print ("Location B is Dirty")
                Environment.locationCondition['B'] = 0;
                Score += 1
                print ("Location B has been Cleaned")
                if Environment.locationCondition['A'] == 1:

                    print ("Location A is Dirty")
                    Score -= 1
                    print ("Moving to Location A")
```

```
                    Environment.locationCondition['A'] = 0;
                    Score += 1
                    print ("Location A has been Cleaned")
            else:
                if Environment.locationCondition['A'] == 1:
                    print ("Location A is Dirty")
                    print ("Moving to Location A")
                    Score -= 1
                    Environment.locationCondition['A'] = 0;
                    Score += 1
                    print ("Location A has been Cleaned")
        print (Environment.locationCondition)
        print ("Performance Measurement: " + str(Score))
theEnvironment = Environment()
theVacuum = SimpleReflexVacuumAgent(theEnvironment)
```

Output:-



RESULT:- Vacuum Cleaner Problem have been successfully implemented.

EXPERIMENT - 3

Aim:- Implementation of Constraint satisfaction problem Example: Implement N-Queen Problem

Algorithm:-

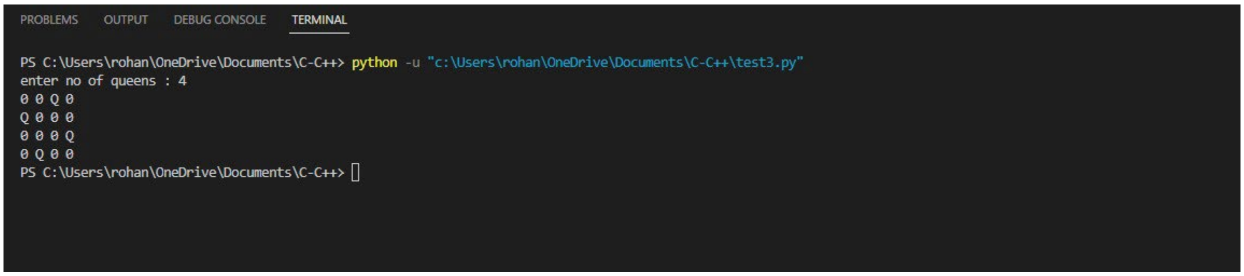
```
while there are untried configurations {
        generate the next configuration
        if queens don't attack in this configuration then {
                print this configuration;
        }
}
```

Program:-

```
global N
N = int(input("Enter no of queen : "))
def printSolution(board):
        for i in range(N):
                for j in range(N):
                        print(board[i][j], end = " ")
                print()
def isSafe(board, row, col):
        for i in range(col):
                if board[row][i] == 1:
                        return False
        for i, j in zip(range(row, -1, -1),
                                        range(col, -1, -1)):
                if board[i][j] == 1:
                        return False
        for i, j in zip(range(row, N, 1),
                                        range(col, -1, -1)):
                if board[i][j] == 1:
                        return False
        return True
def solveNQUtil(board, col):
        if col >= N:
                return True
        for i in range(N):
                if isSafe(board, i, col):
                        board[i][col] = 1
                        if solveNQUtil(board, col + 1) == True:
                                return True
                        board[i][col] = 0
        return False




def solveNQ():
        board = [[0 for i in range(N)] for j in range(N)]
```

```
                if solveNQUtil(board, 0) == False:
                        print ("Solution does not exist")
                        return False
                printSolution(board)
                return True
        solveNQ()
```

Output:-



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\rohan\OneDrive\Documents\C-C++> python -u "c:\Users\rohan\OneDrive\Documents\C-C++\test3.py"
enter no of queens : 4
0 0 Q 0
Q 0 0 0
0 0 0 Q
0 Q 0 0
PS C:\Users\rohan\OneDrive\Documents\C-C++>
```

RESULT:- N QUEEN’S Problem have been successfully implemented.

EXPERIMENT - 4

Aim:- To Implementation and Analysis of BFS and DFS for Application

Algorithm:-

1. Create a node list (Queue) that initially contains the first node N and mark it as visited.
2. Visit the adjacent unvisited vertex of N and insert it in a queue.
3. If there are no remaining adjacent vertices left, remove the first vertex from the queue mark it as visited, display it.
4. Repeat step 1 and step 2 until the queue is empty or the desired node is found.

Program:-

```
graph = {
'S': ['A', 'B'], 'A': ['C', 'D'], 'B': ['G','H'],
'C': ['E','F'], 'D': [],
'G': ['I'],
'H': [],
'E': ['K'],
'F': [],
'I': [],
'K': []
}
visited =[]
queue=[]
def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)
    while queue:
        P=queue.pop(0)
        print(P,end=" ")
        for neighbour in graph[P]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
avisit=set()
def dfs(avisit,graph,node):
    if node not in avisit:
        print(node,end=" ")
        avisit.add(node)
        for neighbour in graph[node]:
            dfs(avisit,graph,neighbour)
print("Breadth first search")
bfs(visited,graph,'S')
print("\nDepth first search")
dfs(avisit,graph,'S')
```

Output:-

```
PROBLEMS  1   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\rohan\OneDrive\Documents\C-C++> python -u "c:\Users\rohan\OneDrive\Documents\C
-C++\test4.py"
Breadth first search
S A B C D G H E F I K
Depth first search
S A C E K F D B G I H
PS C:\Users\rohan\OneDrive\Documents\C-C++>
```

RESULT:- BFS and DFS Algorithm have been successfully implemented.

EXPERIMENT - 5

Aim:- To implement Best First Search and A* algorithm.

Algorithm:-

1. Best First Search-

Step 1: Place the starting node into the OPEN list.
Step 2: If the OPEN list is empty, Stop and return failure.
Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list. If node n is goal then return else
Step 4: Expand the node n, and generate and check the successors of node n. and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 5.
Step 5: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
Step 6: Return to Step 2.

2. A* Alogrithm:-

Step1: Place the starting node in the OPEN list.
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
Step 4:Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
Step 6: Return to Step 2.

Program:-

1. Best First Search:-

```
class Node:
   def__init_(self, v, weight):
      self.v=v
      self.weight=weight
class pathNode:
   def__init_(self, node, parent):
      self.node=node
      self.parent=parent
def addEdge(u, v, weight):
   adj[u].append(Node(v, weight))
adj = []
```

```
def GBFS(h, V, src, dest):
    openList = []
    closeList = []
    openList.append(pathNode(src, None))
    while (openList):
        currentNode = openList[0]
        currentIndex = 0
        for i in range(len(openList)):
            if(h[openList[i].node] < h[currentNode.node]):
                currentNode = openList[i]
                currentIndex = i
        openList.pop(currentIndex)
        closeList.append(currentNode)
        if(currentNode.node == dest):
            path = []
            cur = currentNode
            while(cur != None):
                path.append(cur.node)
                cur = cur.parent
            path.reverse()
            return path
        for node in adj[currentNode.node]:
            for x in openList:
                if(x.node == node.v):
                    continue
            for x in closeList:
                if(x.node == node.v):
                    continue
            openList.append(pathNode(node.v, currentNode))
    return []
""" Making the following graph
             src = 0
            / | \
           /  |  \
          1   2   3
         /\   |   /\
        /  \  | /  \
        4   5 6 7    8
               /
              /
            dest = 9
"""
V = 10
for i in range(V):
    adj.append([])
addEdge(0, 1, 2)
addEdge(0, 2, 1)
addEdge(0, 3, 10)

addEdge(1, 4, 3)
addEdge(1, 5, 2)
```

```
addEdge(2, 6, 9)
addEdge(3, 7, 5)
addEdge(3, 8, 2)
addEdge(7, 9, 5)
h = [20, 22, 21, 10, 25, 24, 30, 5, 12, 0]
path = GBFS(h, V, 0, 9)
for i in range(len(path) - 1):
    print(path[i], end = " -> ")
print(path[(len(path)-1)])
```

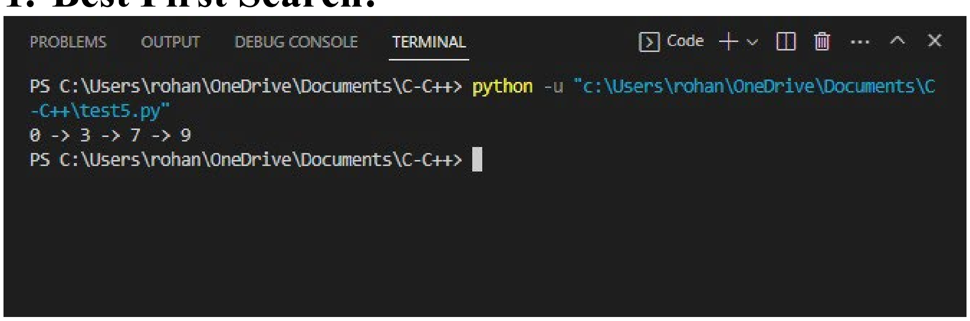
2. A* Algorithm:-

```
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



            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': 99,
            'D': 1,
            'E': 7,
            'G': 0,
        }
        return H_dist[n]
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
aStarAlgo('A', 'G')
```

Output:-

1. Best First Search:-



2. A* Algorithm:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                    Code

PS C:\Users\rohan\OneDrive\Documents\C-C++> python -u "c:\Users\rohan\OneDrive\Documents\C
-C++\test5.py"
Path found: ['A', 'E', 'D', 'G']
PS C:\Users\rohan\OneDrive\Documents\C-C++>
```

RESULT:- Best First Search and A* Algorithm have been successfully implemented.

Experiment 6

Aim – To implement Minimax Algorithm.

Algorithm –

function minimax(node, depth, Player)

1. if depth ==0 or node is a terminal node then
   return value(node)
2. If Player =‘Max’ // for **Maximizer Player**

   set $\alpha = -\infty$ //**worst case value for MAX**

   for each child of node do

   value= minimax(child, depth-1, ’MIN’)

   α= max(α, Value) //gives Maximum of the values

   return (α)

   else // for **Minimizer player**

   set $\alpha = +\infty$ //**worst case value for MIN**

   for each child of node do

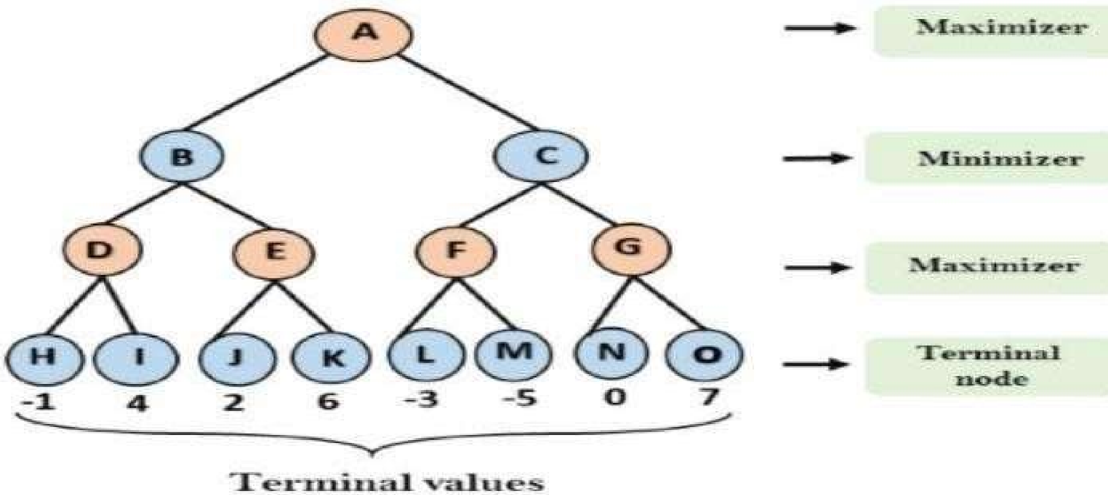
   value= minimax(child, depth-1, ’MAX’)

   α = min(α, Value) //gives minimum of the

   values return (α)

Code –

```
import math
def minimax (curDepth, nodeIndex, maxTurn, scores,targetDepth) :
   if(curDepth==targetDepth):
      return scores[nodeIndex]
   if(maxTurn):
      return max(minimax(curDepth+1,
nodeIndex*2,False,scores,targetDepth),minimax(curDepth+1,
nodeIndex*2+1,False,scores,targetDepth))
   else:
      return min(minimax(curDepth+1,
nodeIndex*2,True,scores,targetDepth),minimax(curDepth+1,
nodeIndex*2+1,True,scores,targetDepth))
scores=[-1,4,2,6,-3,-5,0,7]
treeDepth=math.log(len(scores),2)
print("Optimal value is : ",end=" ")
print(minimax(0,0,True,scores,treeDepth))
```



Result –

```
PS C:\Users\Anupriya Johri> & python "c:/Users/Anupriya Johri/Desktop/Anu college/AI exp/minmax.py"
Optimal value is :  4
PS C:\Users\Anupriya Johri> 
```

Experiment 7

Aim –Implementation of unification and resolution for real-world problems.

Algorithm–

Prolog unification

When programming in Prolog, we spend a lot of time thinking about how variables and rules "match" or "are assigned." There are actually two aspects to this. The first, "unification," regards how terms are matched and variables assigned to make terms match. The second, "resolution," is described in separatenotes. Resolution is only used if rules are involved. You may notice in these notes that no rules are involved since we are only talking about unification.

Terms

Prolog has three kinds of **terms**:

1. Constants like 42 (numbers) and franklin (atoms, i.e., lower-case words).
2. Variables like X and Person (words that start with upper-case).
3. Complex terms like parent(franklin, bo) and baz(X, quux(Y))

Two terms **unify** if they can be matched. Two terms can be matched if:

they are the same term (obviously), or

they contain variables that can be unified so that the two terms without variables are the same.

For example, suppose our knowledge base is:

```
woman(mia).
loves(vincent, angela).
loves(franklin, mia).
```

mia and mia unify
because they are the same.

mia and X unify because X can be given the value mia so that the two terms (without variables) are the same.

woman(mia) and woman(X) unify because X can be set to mia which results in identical terms.

loves(X, mia) and loves(vincent, X) **cannot** unify because there is no assignment for X (given our knowledge base) that makes the two terms identical.

loves(X, mia) and loves(franklin, X) also cannot unify (can you see why?).

We saw in the Prolognotes that we can "query" the knowledge base and get, say, all the people who love mia. When we query with loves(X, mia). we are asking Prolog to give us all the values for X that unify. These values are, essentially, the people who love mia.

term1 and term2 unify whenever:

1. If term1 and term2 are **constants**, then term1 and term2 unify if and only if they are the same atom or the same number.
2. If term1 is a **variable** and term2 is any type of term, then term1 and term2 unify, and term1 is instantiated to term2. (And vice versa.) (If they are both variables, they're both instantiated to each other, and we say that they share values.)
3. If term1 and term2 are **complex terms**, they unify if and only if:

   a. They have the same **functor** and **arity**. The functor is the "function" name (this functor is foo: foo(X, bar)). The arity is the number of arguments for the functor (the arity for foo(X, bar) is 2).

   b. All of their corresponding arguments unify. **Recursion!**

   c. The variable instantiations are compatible (i.e., the same variable is not given two different unifications/values).

4. Two terms unify if and only if they unify for one of the above three reasons (there are no reasons left unstated).

Example

We'll use the = predicate to test if two terms unify. Prolog will answer "Yes" if they do, as well as any sufficient variable assignments to make the unification work.

Do these two terms unify?

1.

```
?- mia = mia.
```

o/p Ans:- Yes from Rule 1

2.

```
?- mia = X.
```

o/p Ans:-Yes, from rule 2.

3.

```
?- X = Y.
```

o/p Yes, from rule 2.

4.

```
?- k(s(g), Y) = k(s(g, X), Y).
```

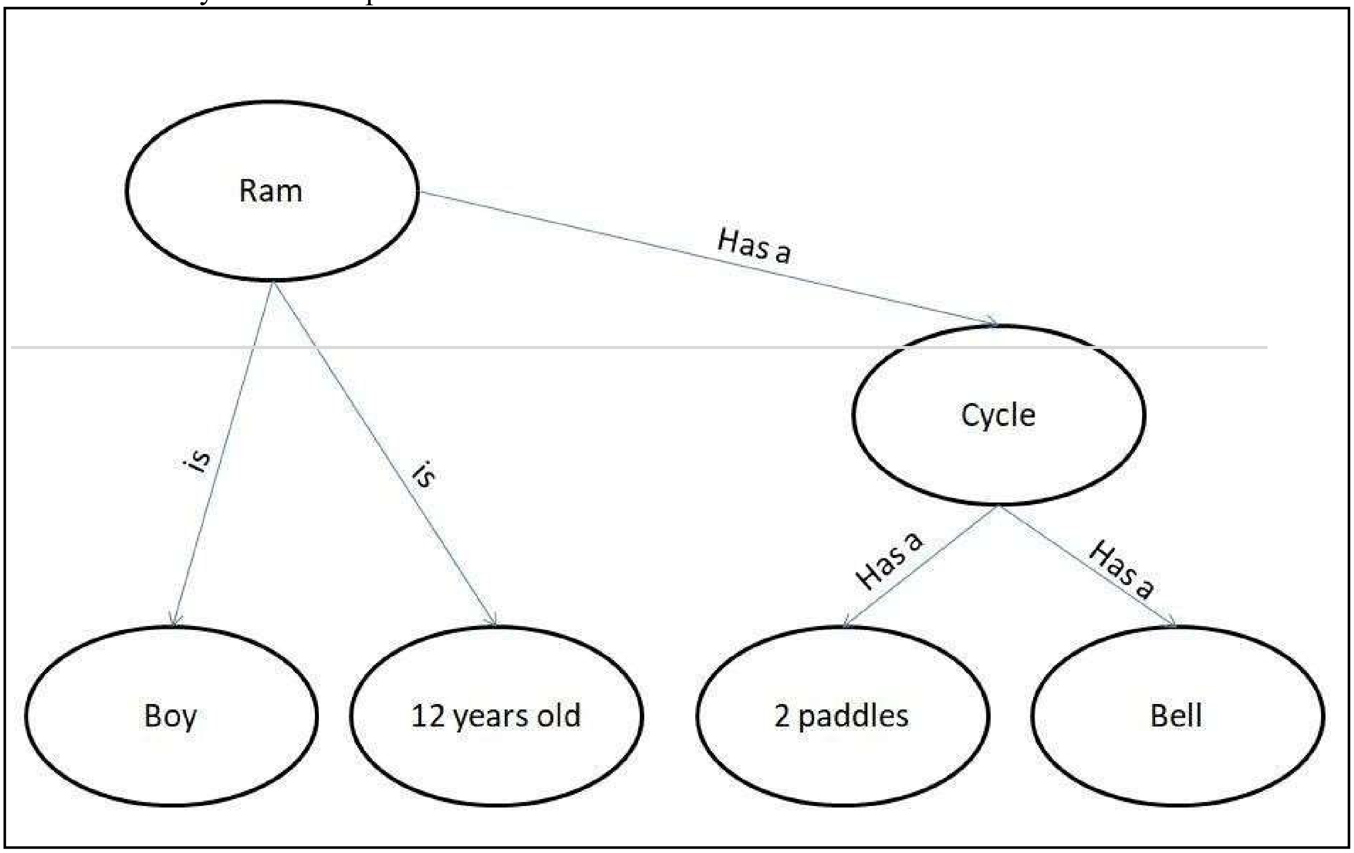
o/p No, these two terms do not unify because arity of s(g) do not match with the arity of s(g,X) due to which rule 3 fails in recursion.

Experiment 8

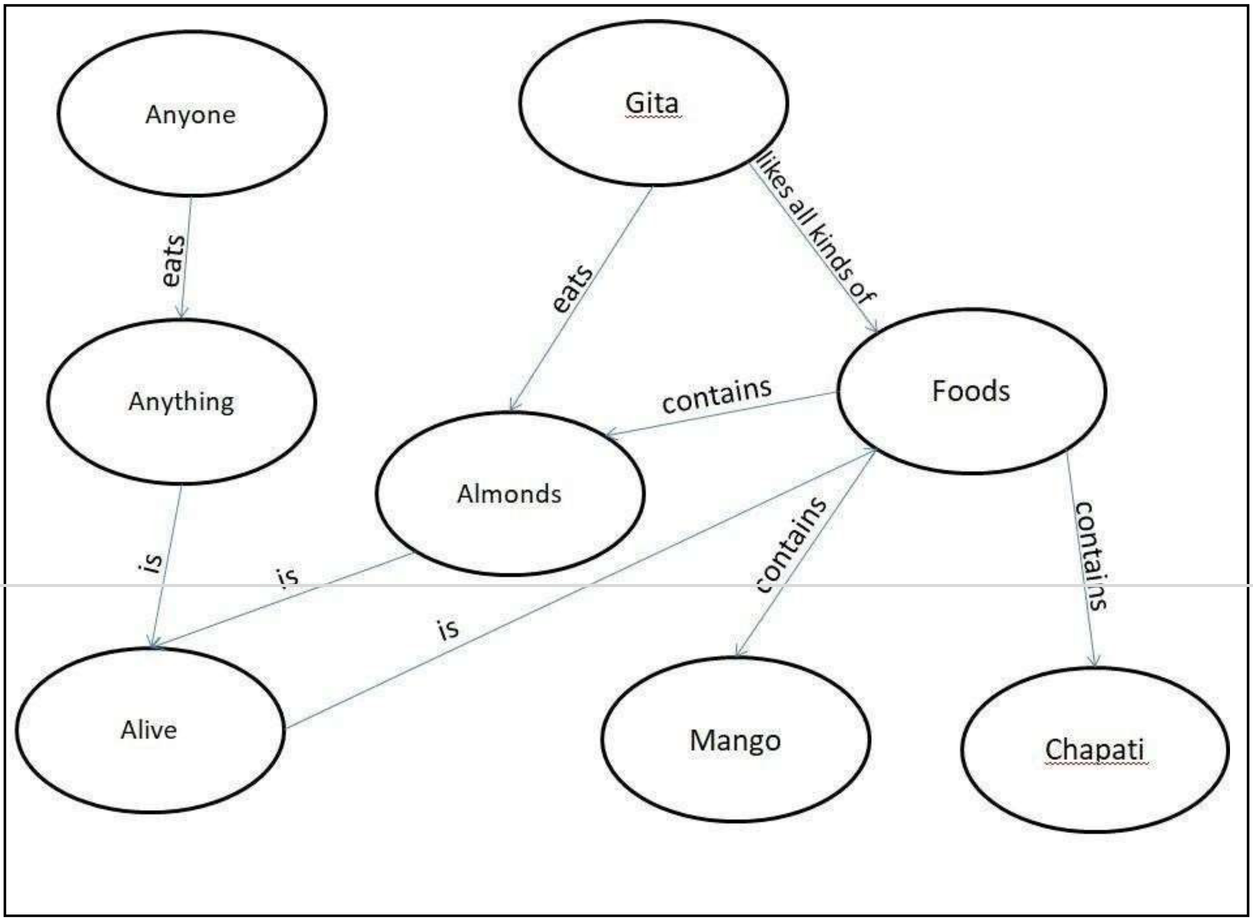
Aim –Implementation of knowledge representation schemes – use cases.

Semantic relations –

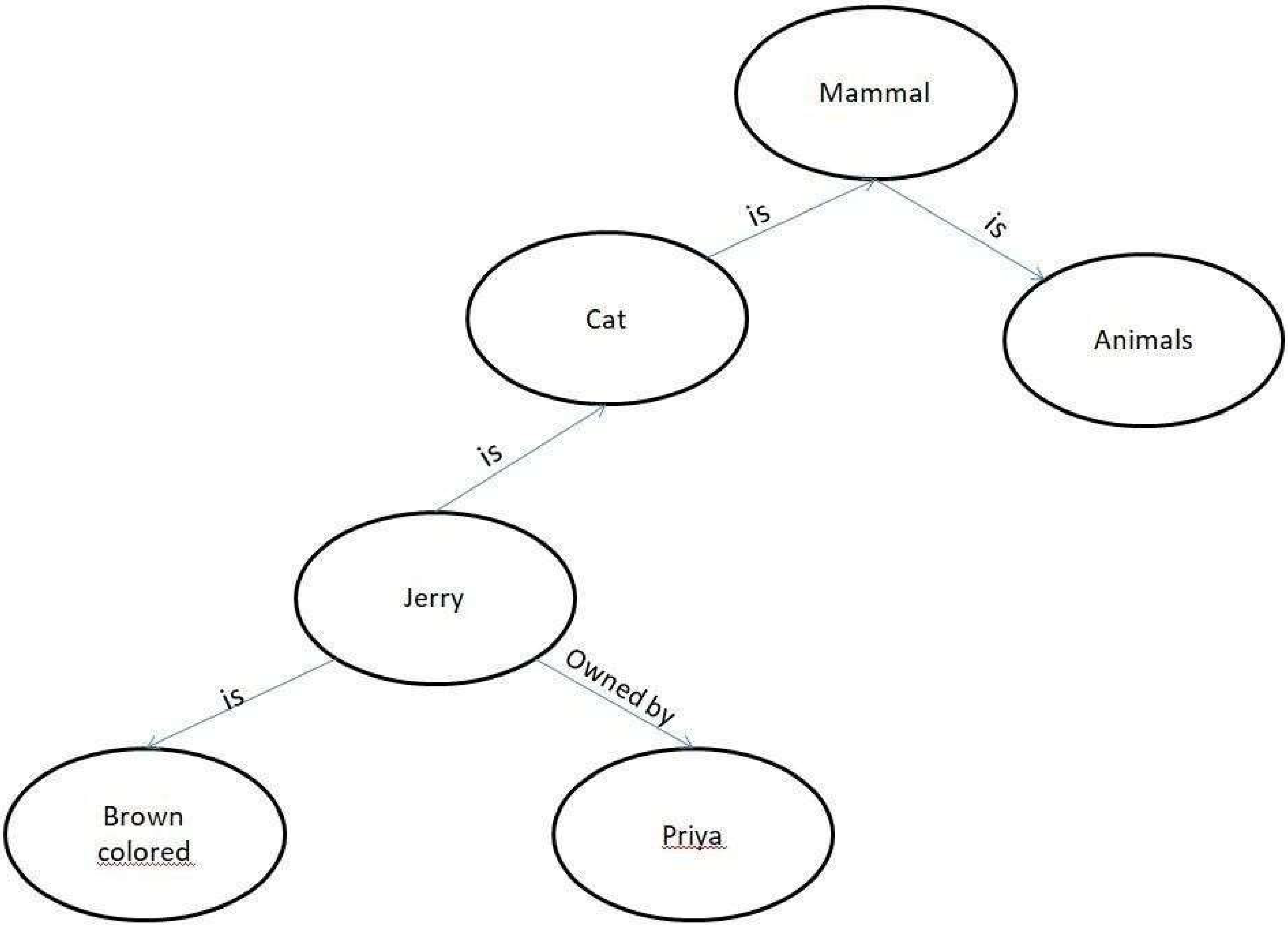
a. 1. Ram has a cycle.
   2. Ram is a boy.
   3. Cycle has a bell.
   4. Ram is 12 years old.
   5. Cycle has two paddles.



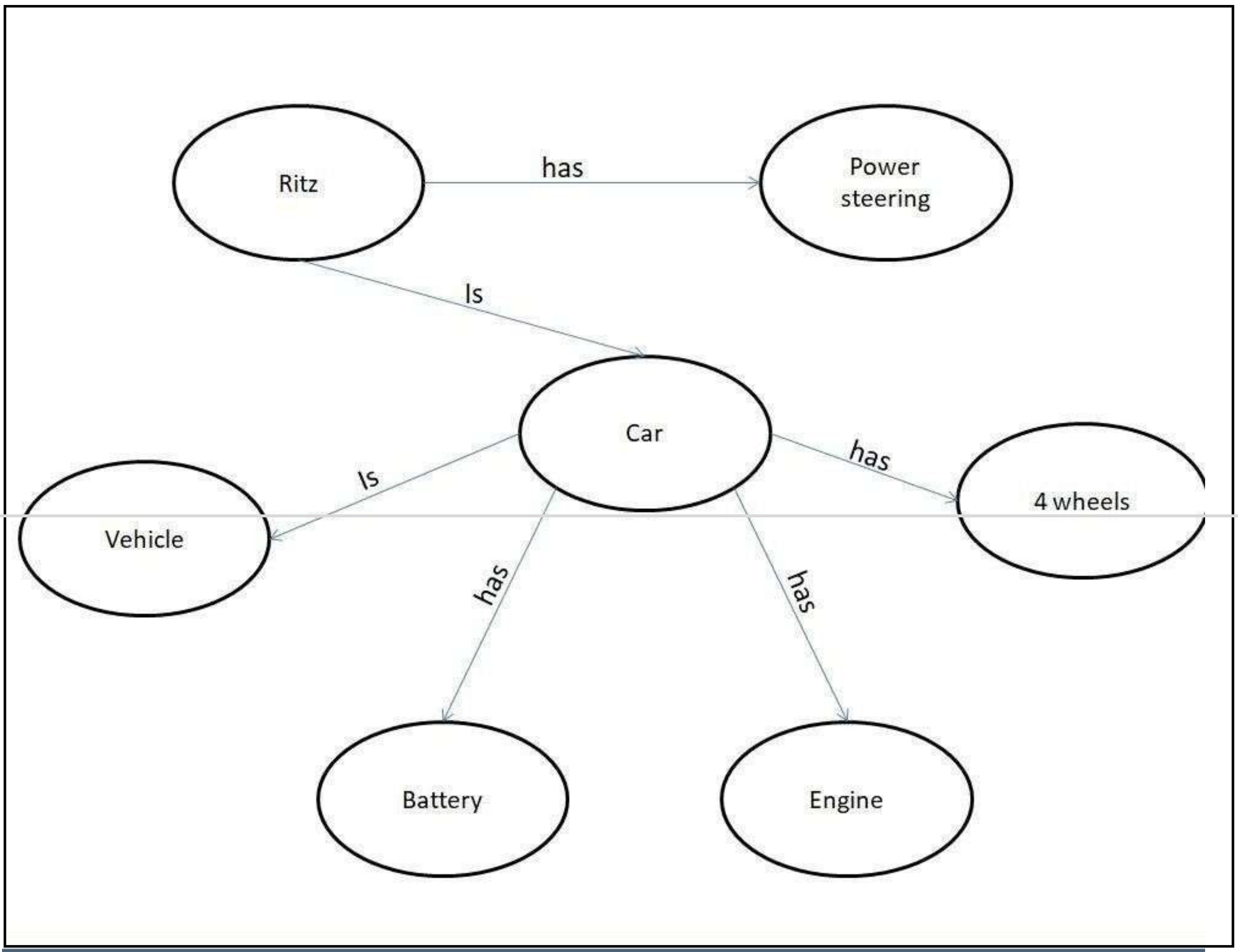
b. 1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.



c. 1. Jerry is a cat.
2. Jerry is a mammal
3. Jerry is owned by Priya.
4. Jerry is brown colored.
5. All Mammals are animal.



d. 1. Ritz is a car.
   2. Car has 4 wheels.
   3. Car is a vehicle.
   4. Car has engine.
   5. Car has battery.
   6. Ritz has power steering.



Experiment 9

Aim –Implementation of uncertain methods for an application.

Algorithm–

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

we can find the probability of an uncertain event by using the above formula.

Code –

Problem1:-

Calculate the Probability of finding how many students got the 60 marks for given data set .

```
import numpy as np
import collections
npArray= np.array([60, 70, 70, 70, 80,90,60])
c=collections.Counter(npArray) # Generate a dictionary {"value":"nbOfOccurrences"}
arraySize=npArray.size
nbOfOccurrences=c[60] #assuming you want the proba to get 10
proba=(nbOfOccurrences/arraySize)*100
print(proba) #print 60.0

output:- 28.57
```

Problem2:-

If In class 80 students and 60 students got 60 % marks then Calculate the Probability of finding how many students got the 60 marks for given data set .

```
#!/usr/bin/env python3
import sys
Marksprob = {}
for line in sys.stdin:
    line = line.strip()
    ClassA, Marks = line.split('\t', 1)
def event_probability(event_outcomes, sample_space):
    probability = (event_outcomes / sample_space) * 100
    return round(probability, 1)
ClassA = 30
Marks = 15
```

```
grade_probability = event_probability(Marks, ClassA)
print(str(grade_probability) + '%')
```

Result –The program has been executed successfully.

Experiment 10

Aim –Implementation of block world problem.

Algorithm –

1. MOVE(B,A)- To lift block from B to A.
2. ON(B,A)- To place block B on A.
3. CLEAR(B)- To lift block B from the table.
4. PLACE(B)- To put the block B on table.

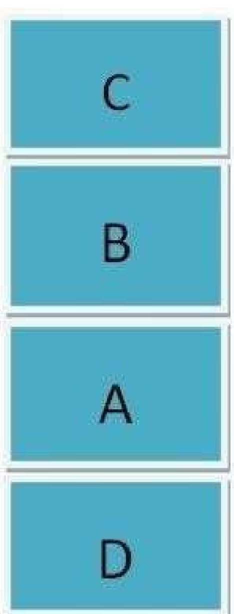
Code –

```
class Strips(object):
 definit(self, name, preconds, effects, cost=1):
    self.name = name
    self.preconds =
    precondsself.effects =
    effects self.cost = cost
defrepr(self):
    return self.name
class STRIPS_domain(object):
 definit(self, feats_vals, actions):
     self.feats_vals =
     feats_valsself.actions = actions
class Planning_problem(object):
 definit(self, prob_domain, initial_state, goal):
     self.prob_domain =
     prob_domainself.initial_state = initial_state
     self.goal = goal
boolean = {True, False}
### blocks world
def move(x,y,z):
    """string for the 'move' action"""
    return 'move_'+x+'_from_'+y+'_to_'+z
def on(x):
    """string for the 'on' feature"""
    return x+'_is_on'
def clear(x):
    """string for the 'clear' feature"""
    return 'clear_'+x
def create_blocks_world(blocks = {'a','b','c','d'}):
    blocks_and_table = blocks | {'table'}
    stmap = {Strips(move(x,y,z),{on(x):y, clear(x):True, clear(z):True},
    {on(x):z, clear(y):True, clear(z):False})}
    for x in blocks:
        for y in blocks_and_table:
            for z in blocks:
                if x!=y and y!=z and z!=x:
                stmap.update({Strips(move(x,y,'table’),{on(x):y,clear(x):True},{on(x):’table’
```

```
                clear(y):True})}) for x in blocks:
        for y in blocks:
            for z in blocks:
                if x!=y:
                    feats_vals = {on(x):blocks_and_table-{x} for x in blocks}
                    feats_vals.update({clear(x):boolean for x in blocks_and_table})

    return STRIPS_domain(feats_vals, stmap)
    blocks1dom = create_blocks_world({'a','b','c'})
    blocks1 = Planning_problem(blocks1dom,
    {on('a'):'table', clear('a'):True,
    on('b'):'c', clear('b'):True,
    on('c'):'table', clear('c'):False}, # initial state
    {on('a'):'b', on('c'):'a'}) #goal
    blocks2dom = create_blocks_world({'a','b','c','d'})
    tower4 = {clear('a'):True, on('a'):'b',
    clear('b'):False, on('b'):'c',
    clear('c'):False, on('c'):'d',
    clear('d'):False, on('d'):'table'}
    blocks2 = Planning_problem(blocks2dom,
    tower4, # initial state
    {on('d'):'c',on('c'):'b',on('b'):'a'}) #goal
    blocks3 = Planning_problem(blocks2dom,
    tower4, # initial state
    {on('d'):'a', on('a'):'b', on('b'):'c'}) #goal
```

Result –Goal achieved.

Output –



Experiment 11

Aim – Implementation of Learning algorithm.

Code –

List of Common Machine Learning Algorithms

Linear Regression
Logistic Regression
Decision Tree
SVM
Naive Bayes
KNN
K-Means
Random Forest

1. Linear Regression

Linear regression is used to estimate real world values like cost of houses, number of calls, total sales etc.

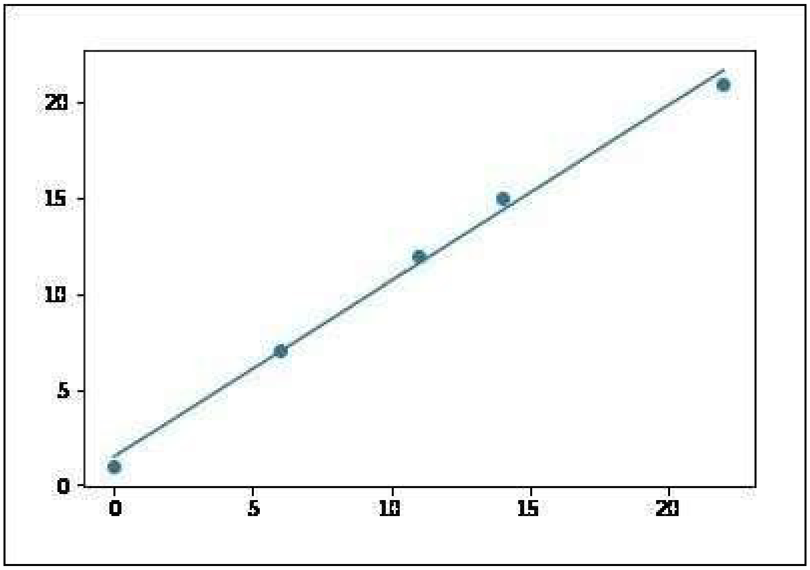
Example

The best way to understand linear regression is by considering an example. Suppose we are asked to arrange students in a class in the increasing order of their weights.

```
# sample points

X = [0, 6, 11, 14, 22]
Y = [1, 7, 12, 15, 21]

# solve for a and b
def best_fit(X, Y):
  xbar = sum(X)/len(X)
  ybar = sum(Y)/len(Y)
  n = len(X) # or len(Y)

  numer = sum([xi*yi for xi,yi in zip(X, Y)]) - n * xbar *
  ybardenum = sum([xi**2 for xi in X]) - n * xbar**2

  b = numer / denum
  a = ybar - b * xbar

  print('best fit line:\ny = {:.2f} + {:.2f}x'.format(a, b))

  return a, b

# solution
a, b = best_fit(X, Y)
#best fit line:
#y = 0.80 + 0.92x
```

```
# plot points and fit line
import matplotlib.pyplot as
pltplt.scatter(X, Y)
yfit = [a + b * xi for xi in X]
plt.plot(X, yfit)
plt.show()
best fit line:
y = 1.48 + 0.92x
```

Output:-



2. KNN (K-Nearest Neighbours)

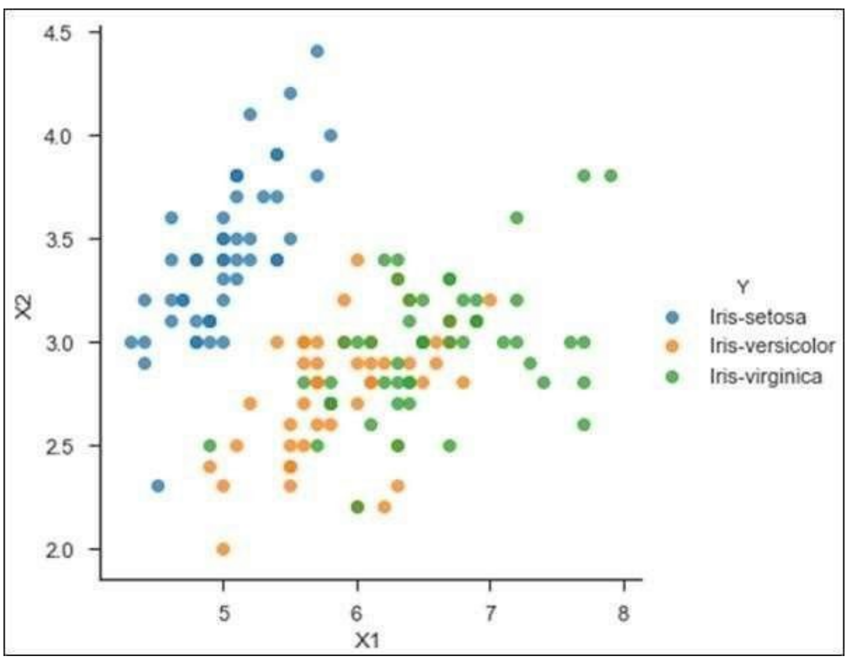
K-Nearest Neighbors, KNN for short, is a supervised learning algorithm specialized in classification. It is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors.

```
#Importing Libraries

from sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset

# Create KNeighbors classifier object model

KNeighborsClassifier(n_neighbors=6) # default value for n_neighborsis 5

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)

from sklearn.neighbors import

KNeighborsClassifierdf = pd.read_csv('iris_df.csv')
```

```
df.columns = ['X1', 'X2', 'X3', 'X4', 'Y']
df = df.drop(['X4', 'X3'], 1)
df.head()
sns.set_context('notebook', font_scale=1.1)
sns.set_style('ticks')
sns.lmplot('X1','X2', scatter=True, fit_reg=False, data=df, hue='Y')
plt.ylabel('X2')
plt.xlabel('X1')
from sklearn.cross_validation import train_test_split
neighbors = KNeighborsClassifier(n_neighbors=5)
X = df.values[:, 0:2]
Y = df.values[:, 2]
trainX, testX, trainY, testY = train_test_split( X, Y, test_size = 0.3)
neighbors.fit(trainX, trainY)
print('Accuracy: \n', neighbors.score(testX, testY))
pred = neighbors.predict(testX)
```

Result:-

('Accuracy: \n', 0.75555555555555554)



Experiment 12

Aim –Development of ensemble model.

Code –

An Ensemble method creates multiple models and combines them to solve it. Ensemble methods help to improve the robustness of the model.

Basic Ensemble Techniques

1 Max Voting
2 Averaging
3 Weighted Average

Problem: Development of ensemble model using Averaging Technique.

Averaging method: It is mainly used for regression problems. The method consists of build multiple models independently and returns the average of the prediction of all the models. In general, the combined output is better than an individual output because variance is reduced.

In the below example, three regression models (linear regression, xgboost, and random forest) are trained and their predictions are averaged. The final prediction output is pred_final.

```
# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# importing machine learning models for prediction
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.linear_model import LinearRegression

# loading train data set in dataframe from train_data.csv file
df = pd.read_csv("train_data.csv")

# getting target data from the dataframe
target = df["target"]

# getting train data from the dataframe
train = df.drop("target")

# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
train, target, test_size=0.20)

# initializing all the model objects with default parameters
model_1 = LinearRegression()
model_2 = xgb.XGBRegressor()
model_3 = RandomForestRegressor()

# training all the model on the training dataset
model_1.fit(X_train, y_target)
model_2.fit(X_train, y_target)
model_3.fit(X_train, y_target)
```

```
# predicting the output on the validation dataset
pred_1 = model_1.predict(X_test)
pred_2 = model_2.predict(X_test)

pred_3 = model_3.predict(X_test)

# final prediction after averaging on the prediction of all 3 models
pred_final = (pred_1+pred_2+pred_3)/3.0

# printing the root mean squared error between real value and predicted value
print(mean_squared_error(y_test, pred_final))
```

Input:-

	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
Rating	5	4	5	4	4	4.4

Output: - final rating will be 4.4

<u>Result</u> – Program compiled successfully.

Experiment 13

Aim – Implementation of NLP problem.

Code –

Problem:-

Count total number of adjective and noun

```
# Import data and tagger
from nltk.corpus import twitter_samples
from nltk.tag import pos_tag_sents

# Load tokenized tweets
tweets_tokens = twitter_samples.tokenized('positive_tweets.json')

# Tag tagged tweets
tweets_tagged = pos_tag_sents(tweets_tokens)

# Set accumulators
JJ_count = 0
NN_count = 0

# Loop through list of tweets
for tweet in tweets_tagged:
    for pair in
        tweet: tag =
        pair[1] if tag
        == 'JJ':
            JJ_count += 1
        elif tag == 'NN':
            NN_count += 1

# Print total numbers for each adjectives and nouns
print('Total number of adjectives = ', JJ_count)
print('Total number of nouns = ', NN_count)
```

Result –

```
- Output
Total number of adjectives =  6094
Total number of nouns =  13180
```

Experiment 14

Aim –Deep learning Project in Python
Code –

The steps to cover in this are as follows:

1. Load Data.
2. Define Keras Model.
3. Compile Keras Model.
4. Fit Keras Model.
5. Evaluate Keras Model.
6. Tie It All Together.
7. Make Predictions

Load Data.

a. **Dataset used –**

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

b. **Code –**

```
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
```

Define Keras Model.

a. **Code –**

```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Compile Keras Model.

a. **Code –**

```
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit Keras Model.

a. Code –

```
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10,verbose=0)
```

Evaluate Keras Model.

a. Code –

```
# evaluate the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
```

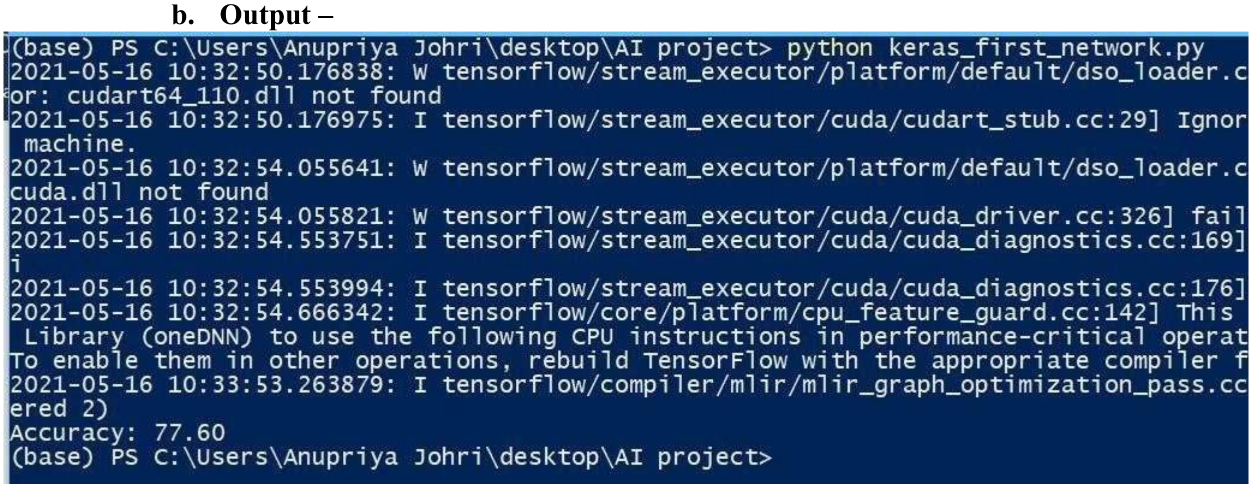
```
print('Accuracy: %.2f' % (accuracy*100))
```

Tie It All Together.

a. **Code-**

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10,verbose=0)

# evaluate the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

b. **Output –**



Make Predictions

a. **Code –**

```
# first neural network with keras make predictions
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
```

```
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10, verbose=0)
# make class predictions with the model
predictions = model.predict_classes(X)
# summarize the first 5 cases
for i in range(5):
    print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

b. Output –

```
(base) PS C:\Users\Anupriya Johri\desktop\AI project> python keras_prediction.py
2021-05-16 10:35:34.935972: W tensorflow/stream_executor/platform/default/dso_loader
or: cudart64_110.dll not found
2021-05-16 10:35:34.936108: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ign
 machine.
2021-05-16 10:35:37.491066: W tensorflow/stream_executor/platform/default/dso_loader
cuda.dll not found
2021-05-16 10:35:37.491273: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] fa
2021-05-16 10:35:37.495274: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:16
i
2021-05-16 10:35:37.495432: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:17
2021-05-16 10:35:37.496001: I tensorflow/core/platform/cpu_feature_guard.cc:142] Thi
 Library (oneDNN) to use the following CPU instructions in performance-critical oper
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
2021-05-16 10:35:38.089867: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.
ered 2)
E:\software\anaconda\lib\site-packages\keras\engine\sequential.py:450: UserWarning:
r 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your
tmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your
 last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
(base) PS C:\Users\Anupriya Johri\desktop\AI project>
```

Keras Project Summary –

In this project, we discovered how to create our first neural network model using the powerful Keras Python library for deep learning.

Specifically, we learnt the six key steps in using Keras to create a neural network or deep learning model, step-by-step including:

How to load data.
How to define a neural network in Keras.
How to compile a Keras model using the efficient numerical backend.
How to train a model on data.
How to evaluate a model on data.
How to make predictions with the model.

Result –The program executed successfully.