List of Experiments

EX.NO: 1

Simple AI Technique implementation

AIM

To implement Simple AI techniques - Water Jug Problem.

PYTHON CODE:

```
x = 0
y = 0
m = 4
n = 3
print("Initial state = (0,0)")
print("Capacities = (4,3)")
print("Goal state = (2,y)")
while x != 2:
  r = int(input("Enter rule"))
  if(r == 1):
    x = m
  elif(r == 2):
    y = n
  elif(r == 3):
    x = 0
  elif(r == 4):
    y = 0
  elif(r == 5):
    t = n - y
    y = n
    x -= t
  elif(r == 6):


    t = m - x
    x = m
```

```
    y -= t
  elif(r == 7):
    y += x
    x = 0
  elif(r == 8):
    x += y
    y = 0
  print (x, y)
```

OUTPUT:

Initial state = (0,0)

Capacities = (4,3)

Goal state = (2,y)

Enter rule2

0 3

Enter rule8

3 0

Enter rule2

3 3

Enter rule6

4 2

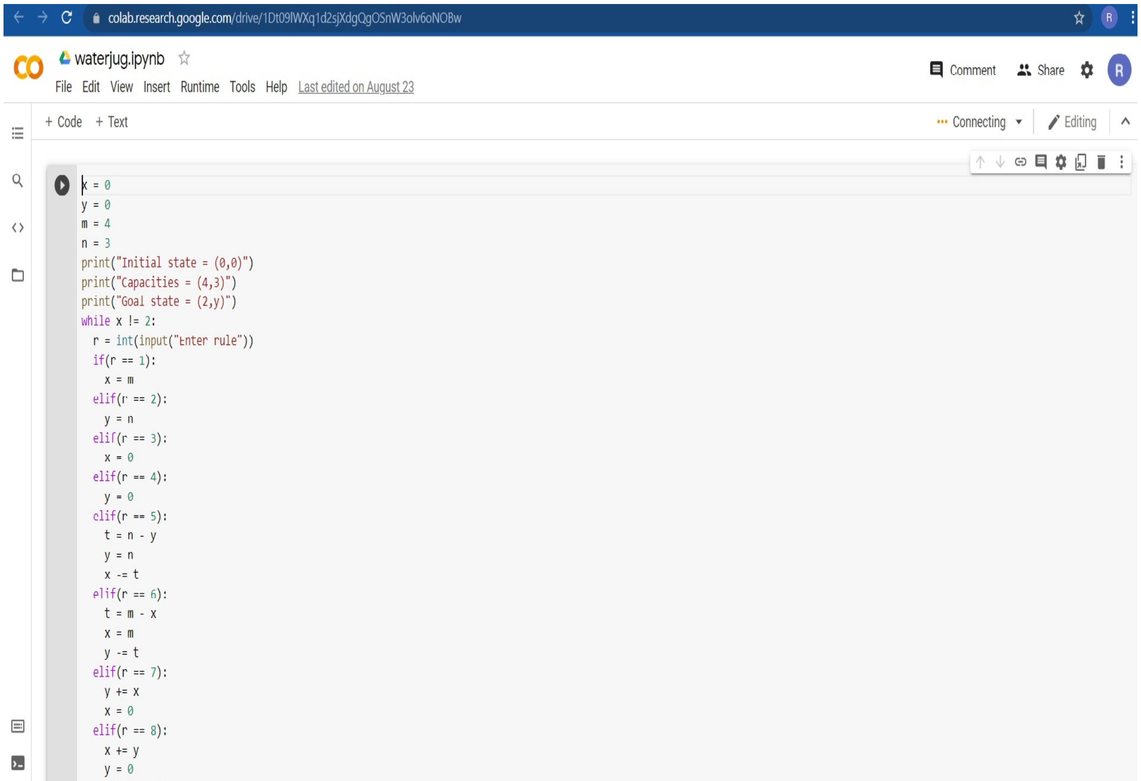
Enter rule3

0 2

Enter rule8

2 0

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 2

Implementation of Tic-Tac-Toe Game

AIM

To Implement Tic Tac Toe Game

PYTHON CODE:

```
def tic_tac_toe():
    board = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    end = False
    win_commbinations = ((0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4,
6))

    def draw():
        print(board[0], board[1], board[2])
        print(board[3], board[4], board[5])
        print(board[6], board[7], board[8])
        print()

    def p1():
        n = choose_number()
        if board[n] == "X" or board[n] == "O":
            print("\nYou can't go there. Try again")
            p1()
        else:

             board[n] = "X"
               def p2():
        n = choose_number()
        if board[n] == "X" or board[n] == "O":
            print("\nYou can't go there. Try again")
            p2()
        else:
```

```
        board[n] = "O"

def choose_number():
    while True:
        while True:
            a = input()
            try:
                a  = int(a)
                a -= 1
                if a in range(0, 9):
                    return a
                else:
                    print("\nThat's not on the board. Try again")
                    continue
            except ValueError:
               print("\nThat's not a number. Try again")
               continue

def check_board():
    count = 0
    for a in win_commbinations:
        if board[a[0]] == board[a[1]] == board[a[2]] == "X":
            print("Player 1 Wins!\n")
            print("Congratulations!\n")
            return True
        if board[a[0]] == board[a[1]] == board[a[2]] == "O":
            print("Player 2 Wins!\n")
            print("Congratulations!\n")
            return True
    for a in range(9):
        if board[a] == "X" or board[a] == "O":
            count += 1
        if count == 9:
            print("The game ends in a Tie\n")
            return True
```

```
    while not end:
        draw()
        end = check_board()
        if end == True:
            break
        print("Player 1 choose where to place a cross")
        p1()
        print()
        draw()
        end = check_board()
        if end == True:
            break
        print("Player 2 choose where to place a nought")
        p2()
        print()

    if input("Play again (y/n)\n") == "y":
        print()
        tic_tac_toe()
tic_tac_toe()
```

OUTPUT:

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

Player 1 choose where to place a cross

1 2 X
4 5 6
7 8 9
```

Player 2 choose where to place a nought

O 2 X

4 5 6

7 8 9

Player 1 choose where to place a cross

O X X

4 5 6

7 8 9

Player 2 choose where to place a nought

O X X

4 5 O

7 8 9

Player 1 choose where to place a cross

O X X

4 5 O

7 8 X

Player 2 choose where to place a nought

O X X

4 O O

7 8 X

Player 1 choose where to place a cross

O X X

X O O

7 8 X

Player 2 choose where to place a nought

O X X

X O O

O 8 X

Player 1 choose where to place a cross

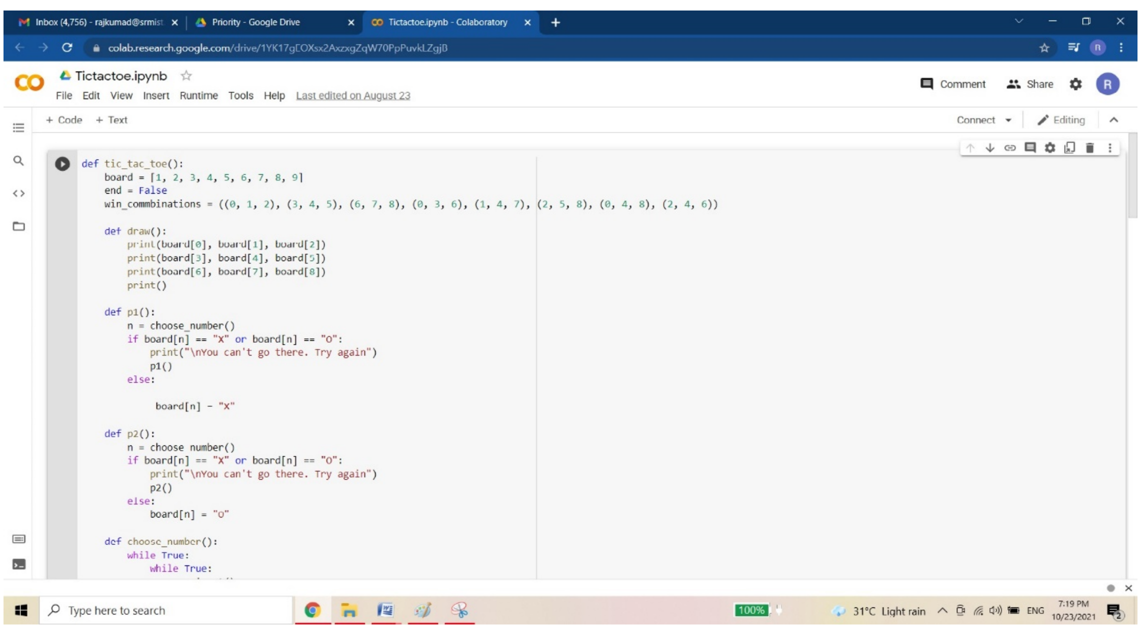
O X X

X O O

O X X

The game ends in a Tie

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 3

Implementation of Travelling Sales man problem

AIM

To Implement Travelling Salesman Problem

PYTHON CODE:

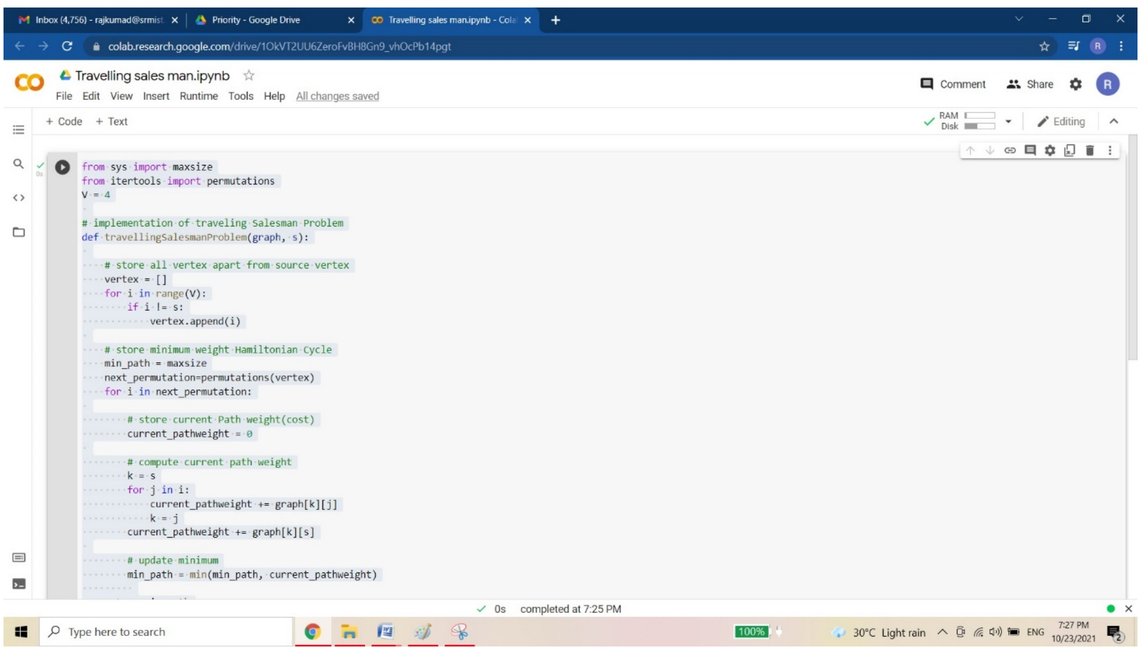
```
from sys import maxsize
from itertools import permutations
V = 4
# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):
    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    # store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        # store current Path weight(cost)
        current_pathweight = 0
        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j


        current_pathweight += graph[k][s]
        # update minimum
```

```
        min_path = min(min_path, current_pathweight)
    return min_path
# Driver Code
if __name__ == "__main__":
    # matrix representation of graph
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
            [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

OUTPUT:

80

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 4

Knowledge implementation

AIM

To implement knowledge to intelligent agents

PYTHON CODE:

```
size = 9
#empty cells have value zero
matrix = [
    [5,3,0,0,7,0,0,0,0],
    [6,0,0,1,9,5,0,0,0],
    [0,9,8,0,0,0,0,6,0],
    [8,0,0,0,6,0,0,0,3],
    [4,0,0,8,0,3,0,0,1],
    [7,0,0,0,2,0,0,0,6],
    [0,6,0,0,0,0,2,8,0],
    [0,0,0,4,1,9,0,0,5],
    [0,0,0,0,8,0,0,7,9]]

#print sudoku
def print_sudoku():
    for i in matrix:
        print (i)

#assign cells and check
def number_unassigned(row, col):
    num_unassign = 0
    for i in range(0,size):



        for j in range (0,size):
            #cell is unassigned
            if matrix[i][j] == 0:
```

```
                row = i
                col = j
                num_unassign = 1
                a = [row, col, num_unassign]
                return a
    a = [-1, -1, num_unassign]
    return a
#check validity of number
def is_safe(n, r, c):
    #checking in row
    for i in range(0,size):
        #there is a cell with same value
        if matrix[r][i] == n:
            return False
    #checking in column
    for i in range(0,size):
        #there is a cell with same value
        if matrix[i][c] == n:
            return False
    row_start = (r//3)*3
    col_start = (c//3)*3;
    #checking submatrix
    for i in range(row_start,row_start+3):
        for j in range(col_start,col_start+3):
            if matrix[i][j]==n:
                return False
    return True

#check validity of number




def solve_sudoku():
    row = 0
    col = 0
```

```
    #if all cells are assigned then the sudoku is already solved
    #pass by reference because number_unassigned will change the values of row and col
    a = number_unassigned(row, col)
    if a[2] == 0:
        return True
    row = a[0]
    col = a[1]
    #number between 1 to 9
    for i in range(1,10):
        #if we can assign i to the cell or not
        #the cell is matrix[row][col]
        if is_safe(i, row, col):
            matrix[row][col] = i
            #backtracking
            if solve_sudoku():
                return True
            #f we can't proceed with this solution
            #reassign the cell
            matrix[row][col]=0
    return False

if solve_sudoku():
    print_sudoku()
else:
    print("No solution")
```

OUTPUT

[5, 3, 4, 6, 7, 8, 9, 1, 2]

[6, 7, 2, 1, 9, 5, 3, 4, 8]

[1, 9, 8, 3, 4, 2, 5, 6, 7]

[8, 5, 9, 7, 6, 1, 4, 2, 3]

[4, 2, 6, 8, 5, 3, 7, 9, 1]

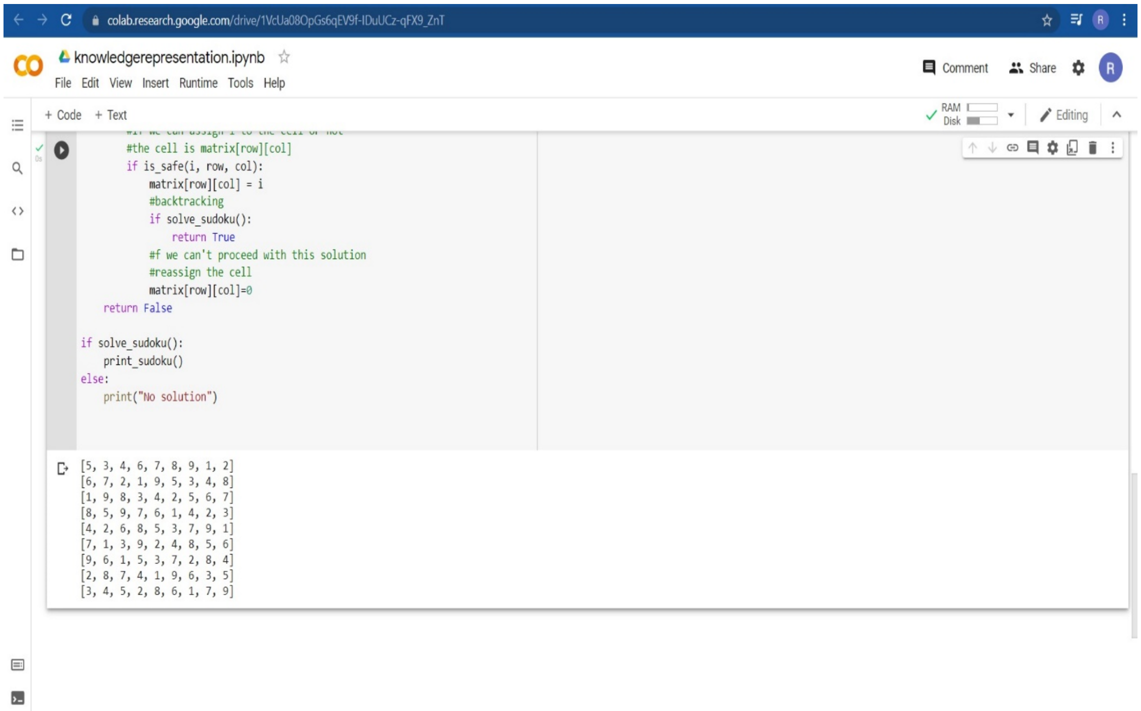
[7, 1, 3, 9, 2, 4, 8, 5, 6]

[9, 6, 1, 5, 3, 7, 2, 8, 4]

[2, 8, 7, 4, 1, 9, 6, 3, 5]

[3, 4, 5, 2, 8, 6, 1, 7, 9]

OUTPUT SCREENSHOT



RESULT: The above python code has been successfully executed.

EX.NO: 5

Implementation of FOPL and Rules

AIM:

To implement First order predicate logic and its rules.

PYTHON CODE:

```
# Import libraries
import aima.utils
import aima.logic
# The main entry point for this module
def main():
    # Create an array to hold clauses
    clauses = []
    # Add first-order logic clauses (rules and fact)
    clauses.append(aima.utils.expr("(American(x) & Weapon(y) & Sells(x, y, z) & Hostile(z)) ==>
Criminal(x)"))
    clauses.append(aima.utils.expr("Enemy(Nono, America)"))
    clauses.append(aima.utils.expr("Owns(Nono, M1)"))
    clauses.append(aima.utils.expr("Missile(M1)"))
    clauses.append(aima.utils.expr("(Missile(x) & Owns(Nono, x)) ==> Sells(West, x, Nono)"))
    clauses.append(aima.utils.expr("American(West)"))
    clauses.append(aima.utils.expr("Missile(x) ==> Weapon(x)"))
    # Create a first-order logic knowledge base (KB) with clauses
    KB = aima.logic.FolKB(clauses)
    # Add rules and facts with tell
    KB.tell(aima.utils.expr('Enemy(Coco, America)'))
    KB.tell(aima.utils.expr('Enemy(Jojo, America)'))
    KB.tell(aima.utils.expr("Enemy(x, America) ==> Hostile(x)"))
    # Get information from the knowledge base with ask
    hostile = aima.logic.fol_fc_ask(KB, aima.utils.expr('Hostile(x)'))
    criminal = aima.logic.fol_fc_ask(KB, aima.utils.expr('Criminal(x)'))
    # Print answers
    print('Hostile?')
```

```
    print(list(hostile))
    print('\nCriminal?')
    print(list(criminal))
    print()
# Tell python to run main method
if __name__ == "__main__": main()
```

OUTPUT:

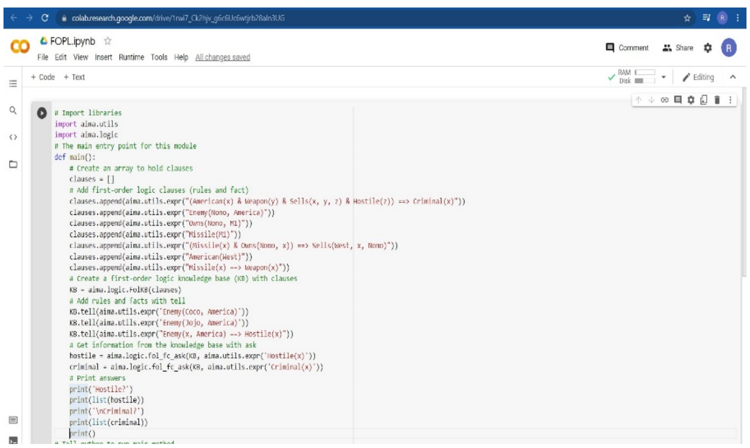
Hostile?

[{x: Coco}, {x: Nono}, {x: Jojo}]

Criminal?

[{x: West}]

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 6

Implementation of Ontology and FOL

AIM

To implement ontology and First order logic

PROCEDURE:

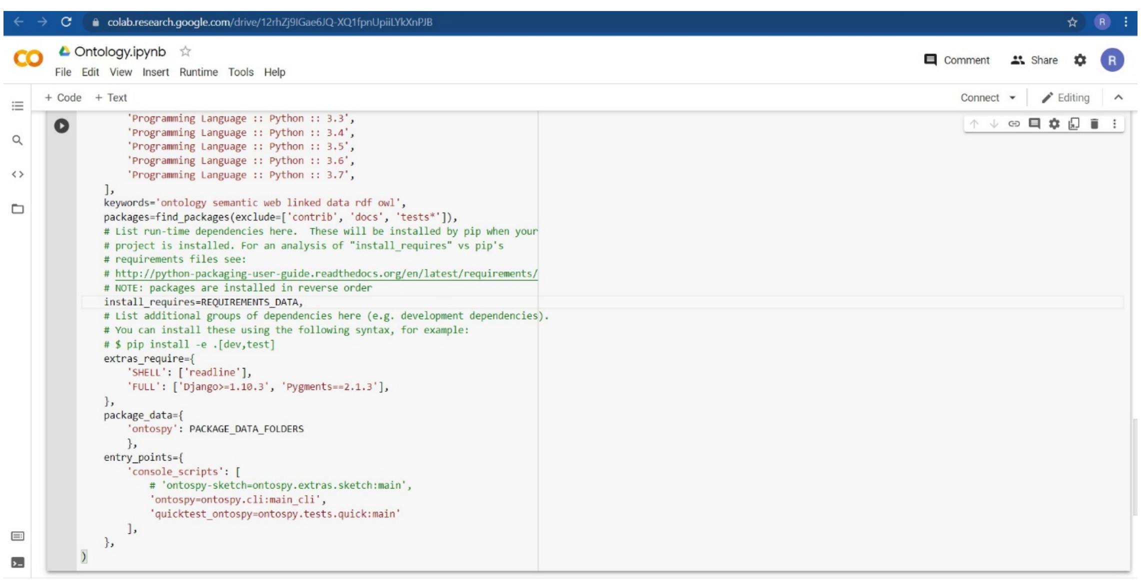
```
from setuptools import setup, find_packages  # Always prefer setuptools over distutils
from codecs import open  # To use a consistent encoding
from os import path
import os
# trick to manage package versions in one place only
# http://stackoverflow.com/questions/458550/standard-way-to-embed-version-into-python-
package
import re
VERSIONFILE="ontospy/VERSION.py"
verstrline = open(VERSIONFILE, "rt").read()
VSRE = r"^__version__ = ['\"]([^'\"]*)['\"]"
mo = re.search(VSRE, verstrline, re.M)
if mo:
    VERSIONSTRING = mo.group(1)
else:
    raise RuntimeError("Unable to find version string in %s." % (VERSIONFILE,))
# Get the long description from the README file
with open(path.join(HERE, 'README.md'), encoding='utf-8') as f:
    long_description = f.read()

# Parse requirements.txt file so to have one single source of truth
REQUIREMENTS_DATA = []
with open(path.join(HERE, 'requirements.txt'), encoding='utf-8') as f:
    for l in f.readlines():
        if not l.startswith("#"):
            if (">=" in l):
                REQUIREMENTS_DATA.append([l.split(">=")[0]])
```

```
        elif ("=" in l):
            REQUIREMENTS_DATA.append([l.split("=")[0]])


def get_package_folders(top_folder, root_path):
    """
    Utility to generate dynamically the list of folders needed by the package_data setting
    ..
        package_data={
            'ontospy': ['viz/static/*.*', 'viz/templates/*.*', 'viz/templates/shared/*.*',
'viz/templates/splitter/*.*', 'viz/templates/markdown/*.*'],
        },
    ...
    """
    _dirs = []
    out = []
    for root, dirs, files in os.walk(top_folder):
        for dir in dirs:
            _dirs.append(os.path.join(root, dir))
    for d in _dirs:
        _d = os.path.join(d, "*.*")
        out.append(_d.replace(root_path+"/", ""))
    return out
PROJECT_ROOT = os.path.join(HERE, "ontospy") # should be top level always
DATA_STATIC_FILES = os.path.join(PROJECT_ROOT, "ontodocs", "media", "static")
DATA_TEMPLATE_FILES = os.path.join(PROJECT_ROOT, "ontodocs", "media", "templates")
# dynamically generate list of data folders
PACKAGE_DATA_FOLDERS = get_package_folders(
    DATA_STATIC_FILES, PROJECT_ROOT) + get_package_folders(
        DATA_TEMPLATE_FILES, PROJECT_ROOT)
if True:
    print(PACKAGE_DATA_FOLDERS)
setup(
    name='ontospy',
    version=VERSIONSTRING,
    description=
```

```
'Query, inspect and visualize knowledge models encoded as RDF&OWL ontologies.',
long_description=long_description,
long_description_content_type='text/markdown',
url='https://github.com/lambdamusic/ontospy',
author='Michele Pasin',
author_email='michele.pasin@gmail.com',
license='MIT',
# See https://pypi.python.org/pypi?%3Aaction=list_classifiers
classifiers=[
   # How mature is this project? Common values are
   #   3 - Alpha
   #   4 - Beta
   #   5 - Production/Stable
   'Development Status :: 5 - Production/Stable',

   # Indicate who your project is intended for
   'Intended Audience :: Science/Research',
   'Intended Audience :: Developers',
   'Topic :: Scientific/Engineering :: Artificial Intelligence',
   # Pick your license as you wish (should match "license" above)
   'License :: OSI Approved :: MIT License',

   # Specify the Python versions you support here. In particular, ensure
   # that you indicate whether you support Python 2, Python 3 or both.
   'Programming Language :: Python :: 3',
   'Programming Language :: Python :: 3.2',
   'Programming Language :: Python :: 3.3',
   'Programming Language :: Python :: 3.4',
   'Programming Language :: Python :: 3.5',
   'Programming Language :: Python :: 3.6',
   'Programming Language :: Python :: 3.7',
],
# requirements files see:
# http://python-packaging-user-guide.readthedocs.org/en/latest/requirements/
# NOTE: packages are installed in reverse order
```

```
install_requires=REQUIREMENTS_DATA,
# List additional groups of dependencies here (e.g. development dependencies).
# You can install these using the following syntax, for example:
# $ pip install -e .[dev,test]
extras_require={
    'SHELL': ['readline'],
    'FULL': ['Django>=1.10.3', 'Pygments==2.1.3'],
},
package_data={
    'ontospy': PACKAGE_DATA_FOLDERS
    },
entry_points={
    'console_scripts': [
        # 'ontospy-sketch=ontospy.extras.sketch:main',
        'ontospy=ontospy.cli:main_cli',
        'quicktest_ontospy=ontospy.tests.quick:main'
    ],
},)
```

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 7

Concept Learning task

AIM:

To implement Concept learning task in machine learning.

PYTHON CODE:

```
import pandas as pd
import numpy as np
#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")
#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)
#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis
#obtaining the final hypothesis
```

```
print("n The final hypothesis is:",train(d,target))
```

OUTPUT:

Time Weather Temperature Company Humidity Wind Goes

0 Morning Sunny Warm Yes Mild Strong Yes

1 Evening Rainy Cold No Mild Normal No

2 Morning Sunny Moderate Yes Normal Normal Yes

3 Evening Sunny Cold Yes High Strong Yes n

n The attributes are: [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']

['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']

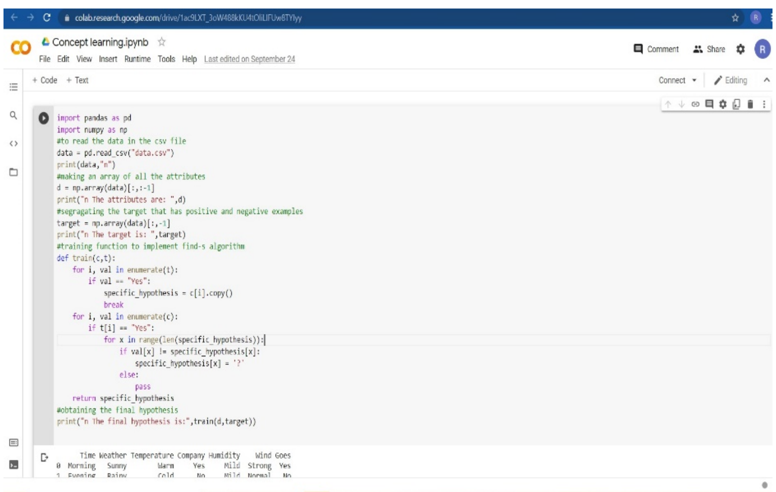
['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']

['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

n The target is: ['Yes' 'No' 'Yes' 'Yes']

n The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO:8

Design a Learning System

AIM

To design a learning system in machine learning

PYTHON CODE:

```
import sys
print('Python: {}'.format(sys.version))
import scipy
print('scipy: {}'.format(scipy.__version__))
import numpy
print('numpy: {}'.format(numpy.__version__))
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
import pandas pd
print('pandas: {}'.format(pandas.__version__))
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)
print(dataset.shape)
print(dataset.head(20))
print(dataset.describe())
print(dataset.groupby('class').size())
dataset.hist()
pyplot.show()
scatter_matrix(dataset)
pyplot.show()
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
```

```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
dataset.hist()
pyplot.show()
scatter_matrix(dataset)
pyplot.show()
```

OUTPUT:

```
Python: 3.7.12 (default, Sep 10 2021, 00:21:48)
[GCC 7.5.0]
scipy: 1.4.1
numpy: 1.19.5
matplotlib: 3.2.2
pandas: 1.1.5
sklearn: 0.22.2.post1
(150, 5)
   sepal-length  sepal-width  petal-length  petal-width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
5           5.4          3.9           1.7          0.4  Iris-setosa
6           4.6          3.4           1.4          0.3  Iris-setosa
7           5.0          3.4           1.5          0.2  Iris-setosa
8           4.4          2.9           1.4          0.2  Iris-setosa
9           4.9          3.1           1.5          0.1  Iris-setosa
10          5.4          3.7           1.5          0.2  Iris-setosa
11          4.8          3.4           1.6          0.2  Iris-setosa
12          4.8          3.0           1.4          0.1  Iris-setosa
13          4.3          3.0           1.1          0.1  Iris-setosa
14          5.8          4.0           1.2          0.2  Iris-setosa
15          5.7          4.4           1.5          0.4  Iris-setosa
```

16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

class

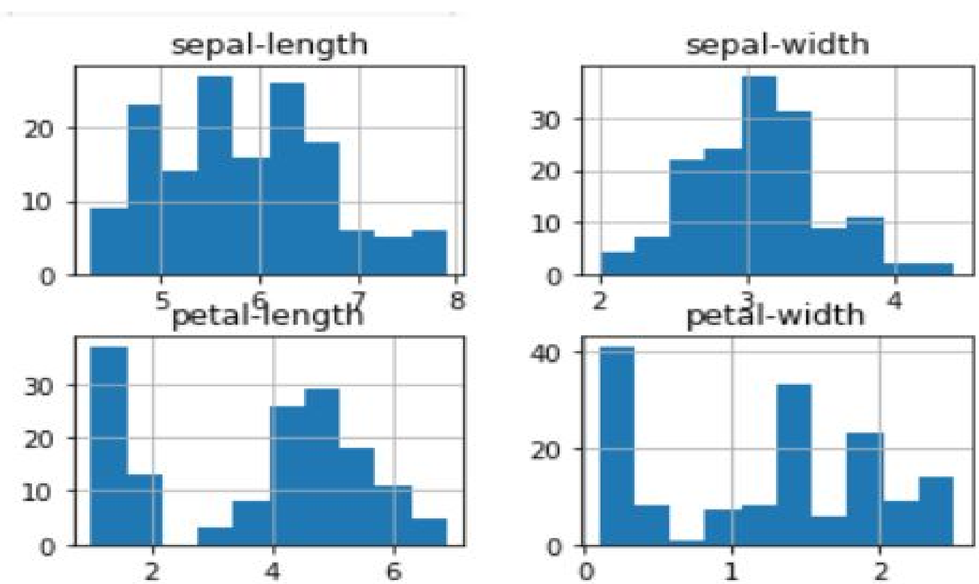
Iris-setosa 50

Iris-versicolor 50

Iris-virginica 50

dtype: int64

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 9

Implementation of candidate elimination algorithm

AIM

To implement candidate elimination algorithm

PYTHON CODE:

```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)
    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
```

```
                general_h[x][x] = '?'
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")
    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("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

OUTPUT:

```
Instances 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']]
Target Values are:  ['yes' 'yes' 'no' 'yes']
Initialization of specific_h and genearal_h
Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
```

Specific Bundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 is ['sunny' 'warm' 'high' 'strong' 'cool' 'change']

Instance is Positive

Specific Bundary after 4 Instance is ['sunny' 'warm' '?' 'strong' '?' '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

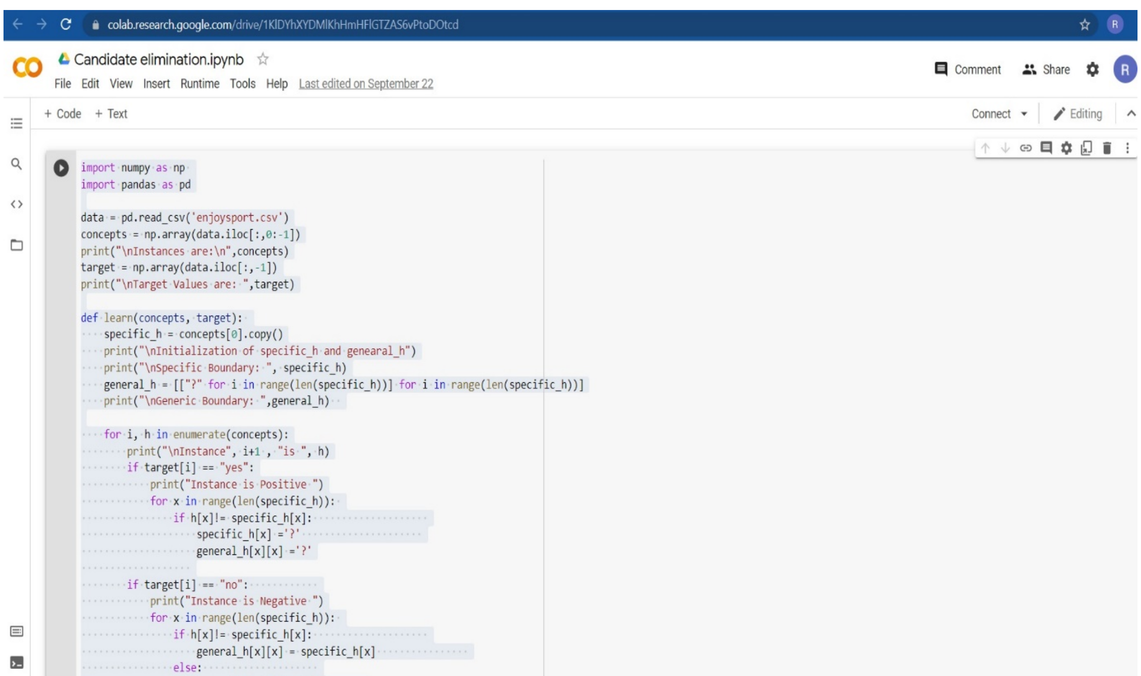
Final Specific_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 10

Decision tree implementation

AIM:

To implement Decision tree algorithm

PYTHON CODE:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("pima-indians-diabetes.csv", header = None, names = col_names)
pima.head()
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)

from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
```

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data, filled=True, rounded=True,
special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Pima_diabetes_Tree.png')
Image(graph.create_png())
```

OUTPUT:

Confusion Matrix:

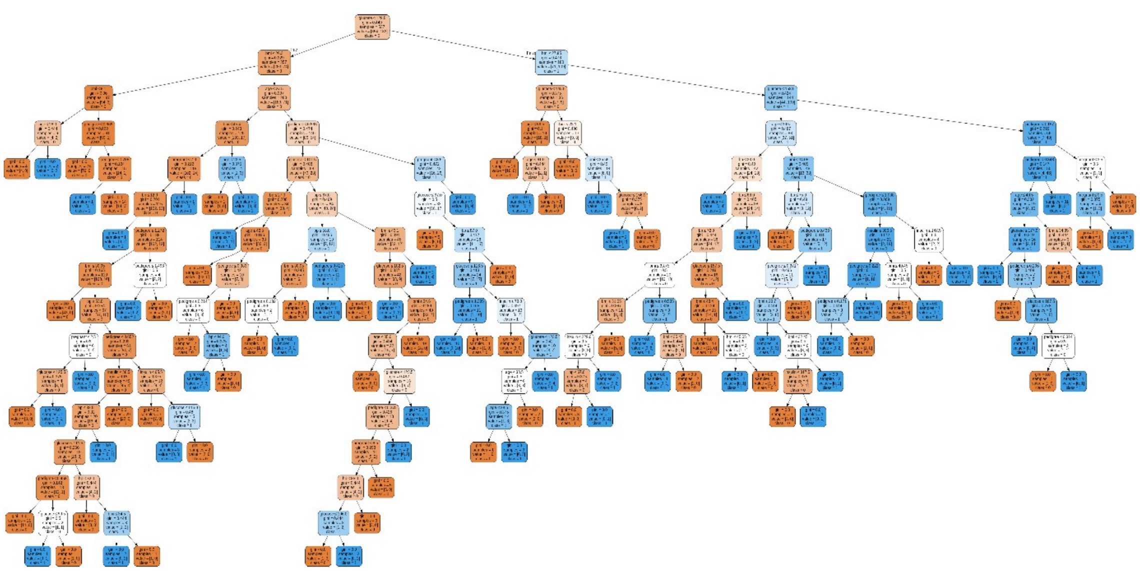
[[117 29]

 [44 41]]

Classification Report:

	precision	recall	f1-score	support
0	0.73	0.80	0.76	146
1	0.59	0.48	0.53	85
accuracy			0.68	231
macro avg	0.66	0.64	0.65	231
weighted avg	0.67	0.68	0.68	231

OUTPUT SCREENSHOT:



RESULT: The above python code has been successfully executed.

EX.NO: 11

Implementation of K- Means algorithm

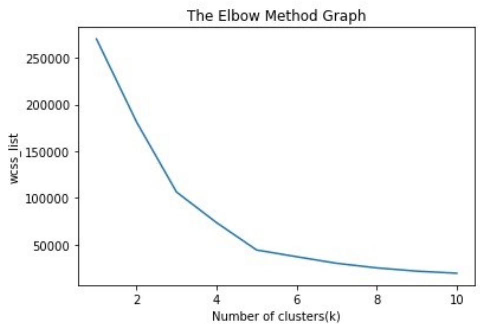
AIM:

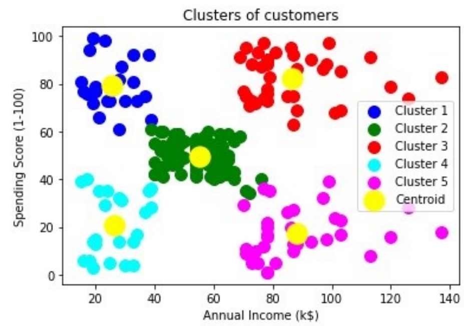
To implement k-means algorithm

PYTHON CODE:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
dataset = pd.read_csv('Mall_Customers.csv')
x = dataset.iloc[:, [3, 4]].values
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS
  for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elbow Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for
first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for
second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for
third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for
fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
#for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow',
label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')



mtp.legend()
mtp.show()
```

OUTPUT SCREENSHOT:





RESULT: The above python code has been successfully executed.

EX.NO: 12

Implementation of ID3 algorithm

AIM

To implement ID3 algorithm

PYTHON CODE:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import pandas as pd
dataset = pd.read_csv("pima-indians-diabetes.csv")
dataset.head()
dataset.shape
features = dataset.drop(["Outcome"], axis=1)
X = np.array(features)
y = np.array(dataset["Outcome"])
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=0, test_size=0.20)
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
tree.tree_.max_depth
validation_prediction = tree.predict(X_val)
training_prediction = tree.predict(X_train)
print('Accuracy training set: ', accuracy_score(y_true=y_train, y_pred=training_prediction))
print('Accuracy validation set: ', accuracy_score(y_true=y_val, y_pred=validation_prediction))
from sklearn.tree import export_graphviz
feature_names = features.columns
dot_data = export_graphviz(tree, out_file=None,
                        feature_names=feature_names,
                         class_names=True,
                         filled=True, rounded=True,
                         special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

OUTPUT:

Accuracy training set: 0.7964169381107492

Accuracy validation set: 0.8116883116883117

RESULT: The above python code has been successfully executed.

EX.NO: 13

Neural Network model implementation

AIM

To implement Neural Network.

PYTHON CODE:

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt(', 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)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

OUTPUT:

Epoch 1/200

77/77 [==============================] - 1s 1ms/step - loss: 9.3462 - accuracy: 0.5404

Epoch 2/200

77/77 [==============================] - 0s 1ms/step - loss: 2.6633 - accuracy: 0.4896

Epoch 3/200

77/77 [==============================] - 0s 1ms/step - loss: 1.4994 - accuracy: 0.5391

Epoch 4/200

77/77 [==============================] - 0s 1ms/step - loss: 1.0731 - accuracy: 0.5938

Epoch 5/200

77/77 [==============================] - 0s 1ms/step - loss: 0.8820 - accuracy: 0.6367

Epoch 6/200

77/77 [==============================] - 0s 2ms/step - loss: 0.8197 - accuracy: 0.6289

Epoch 7/200

77/77 [==============================] - 0s 1ms/step - loss: 0.7773 - accuracy: 0.6432

Epoch 8/200

77/77 [==============================] - 0s 1ms/step - loss: 0.7406 - accuracy: 0.6523

Epoch 9/200

77/77 [==============================] - 0s 1ms/step - loss: 0.7005 - accuracy: 0.6471

Epoch 10/200

77/77 [==============================] - 0s 1ms/step - loss: 0.7090 - accuracy: 0.6549

Epoch 11/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6890 - accuracy: 0.6341

Epoch 12/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6738 - accuracy: 0.6615

Epoch 13/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6460 - accuracy: 0.6719

Epoch 14/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6386 - accuracy: 0.6680

Epoch 15/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6327 - accuracy: 0.6836

Epoch 16/200

77/77 [==============================] - 0s 2ms/step - loss: 0.6208 - accuracy: 0.6680

Epoch 17/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6492 - accuracy: 0.6667

Epoch 18/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6164 - accuracy: 0.6914

Epoch 19/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6257 - accuracy: 0.6706

Epoch 20/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6058 - accuracy: 0.6771

Epoch 21/200

77/77 [==============================] - 0s 1ms/step - loss: 0.5965 - accuracy: 0.6862

Epoch 22/200

77/77 [==============================] - 0s 2ms/step - loss: 0.5961 - accuracy: 0.7044

Epoch 23/200

77/77 [==============================] - 0s 1ms/step - loss: 0.5835 - accuracy: 0.7083

Epoch 24/200

77/77 [==============================] - 0s 1ms/step - loss: 0.6045 - accuracy: 0.6901

Epoch 25/200

77/77 [==============================] - 0s 2ms/step - loss: 0.5788 - accuracy: 0.7070

……………………………………………………………………………..

RESULT: The above python code has been successfully executed.

EX.NO: 14

Implementation of Multi-layer neural network

AIM

To implement multi layer Neural Network.

PYTHON CODE:

```
from sklearn.base import BaseEstimator, ClassifierMixin, RegressorMixin
import random
class MultiLayerPerceptron(BaseEstimator, ClassifierMixin):
    def __init__(self, params=None):
        if (params == None):
            self.inputLayer = 4                        # Input Layer
            self.hiddenLayer = 5                       # Hidden Layer
            self.outputLayer = 3                       # Outpuy Layer
            self.learningRate = 0.005                  # Learning rate
            self.max_epochs = 600                      # Epochs
            self.iasHiddenValue = -1                   # Bias HiddenLayer
            self.BiasOutputValue = -1                  # Bias OutputLayer
            self.activation = self.ativacao['sigmoid'] # Activation function
            self.deriv = self.derivada['sigmoid']
        else:
            self.inputLayer = params['InputLayer']
            self.hiddenLayer = params['HiddenLayer']
            self.OutputLayer = params['OutputLayer']
            self.learningRate = params['LearningRate']
            self.max_epochs = params['Epocas']
            self.BiasHiddenValue = params['BiasHiddenValue']

            self.BiasOutputValue = params['BiasOutputValue']
            self.activation = self.ativacao[params['ActivationFunction']]
            self.deriv = self.derivada[params['ActivationFunction']]

        'Starting Bias and Weights'
```

```
        self.WEIGHT_hidden = self.starting_weights(self.hiddenLayer, self.inputLayer)
        self.WEIGHT_output = self.starting_weights(self.OutputLayer, self.hiddenLayer)
        self.BIAS_hidden = np.array([self.BiasHiddenValue for i in range(self.hiddenLayer)])
        self.BIAS_output = np.array([self.BiasOutputValue for i in range(self.OutputLayer)])
        self.classes_number = 3

    pass
     def starting_weights(self, x, y):
        return [[2  * random.random() - 1 for i in range(x)] for j in range(y)]
    ativacao = {
         'sigmoid': (lambda x: 1/(1 + np.exp(-x))),
            'tanh': (lambda x: np.tanh(x)),
            'Relu': (lambda x: x*(x > 0)),
               }
    derivada = {
         'sigmoid': (lambda x: x*(1-x)),
            'tanh': (lambda x: 1-x**2),
            'Relu': (lambda x: 1 * (x>0))
               }
    def Backpropagation_Algorithm(self, x):
        DELTA_output = []
        'Stage 1 - Error: OutputLayer'
        ERROR_output = self.output - self.OUTPUT_L2
        DELTA_output = ((-1)*(ERROR_output) * self.deriv(self.OUTPUT_L2))
arrayStore = []
        'Stage 2 - Update weights OutputLayer and HiddenLayer'

        for i in range(self.hiddenLayer):
            for j in range(self.OutputLayer):
                self.WEIGHT_output[i][j] -= (self.learningRate * (DELTA_output[j] *
self.OUTPUT_L1[i]))
                self.BIAS_output[j] -= (self.learningRate * DELTA_output[j])
        'Stage 3 - Error: HiddenLayer'
        delta_hidden = np.matmul(self.WEIGHT_output, DELTA_output)*
self.deriv(self.OUTPUT_L1)
```

```
    'Stage 4 - Update weights HiddenLayer and InputLayer(x)'
    for i in range(self.OutputLayer):
        for j in range(self.hiddenLayer):
            self.WEIGHT_hidden[i][j] -= (self.learningRate * (delta_hidden[j] * x[i]))
            self.BIAS_hidden[j] -= (self.learningRate * delta_hidden[j])
def show_err_graphic(self,v_erro,v_epoca):
    plt.figure(figsize=(9,4))
    plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
    plt.xlabel("Number of Epochs")
    plt.ylabel("Squared error (MSE) ");
    plt.title("Error Minimization")
    plt.show()
def predict(self, X, y):
    'Returns the predictions for every element of X'
    my_predictions = []
    'Forward Propagation'
    forward = np.matmul(X,self.WEIGHT_hidden) + self.BIAS_hidden
    forward = np.matmul(forward, self.WEIGHT_output) + self.BIAS_output
    for i in forward:
        my_predictions.append(max(enumerate(i), key=lambda x:x[1])[0])
    print(" Number of Sample  | Class |  Output |  Hoped Output  ")
    for i in range(len(my_predictions)):

        if(my_predictions[i] == 0):
            print("id:{}    | Iris-Setosa  |  Output: {}  ".format(i, my_predictions[i], y[i]))
        elif(my_predictions[i] == 1):
            print("id:{}    | Iris-Versicolour    |  Output: {}  ".format(i, my_predictions[i], y[i]))
        elif(my_predictions[i] == 2):
            print("id:{}    | Iris-Iris-Virginica   |  Output: {}  ".format(i, my_predictions[i], y[i]))
    return my_predictions
    pass
def fit(self, X, y):
    count_epoch = 1
    total_error = 0
```

```
        n = len(X);
        epoch_array = []
        error_array = []
        W0 = []
        W1 = []
        while(count_epoch <= self.max_epochs):
            for idx,inputs in enumerate(X):
                self.output = np.zeros(self.classes_number)
                'Stage 1 - (Forward Propagation)'
                self.OUTPUT_L1 = self.activation((np.dot(inputs, self.WEIGHT_hidden) +
self.BIAS_hidden.T))
                self.OUTPUT_L2 = self.activation((np.dot(self.OUTPUT_L1, self.WEIGHT_output) +
self.BIAS_output.T))
                'Stage 2 - One-Hot-Encoding'
                if(y[idx] == 0):
                    self.output = np.array([1,0,0]) #Class1 {1,0,0}
                elif(y[idx] == 1):
                    self.output = np.array([0,1,0]) #Class2 {0,1,0}
                elif(y[idx] == 2):

                    self.output = np.array([0,0,1]) #Class3 {0,0,1}
                square_error = 0
                for i in range(self.OutputLayer):
                    erro = (self.output[i] - self.OUTPUT_L2[i])**2
                    square_error = (square_error + (0.05 * erro))
                    total_error = total_error + square_error
                'Backpropagation : Update Weights'
                self.Backpropagation_Algorithm(inputs)
            total_error = (total_error / n)
            if((count_epoch % 50 == 0)or(count_epoch == 1)):
                print("Epoch ", count_epoch, "- Total Error: ",total_error)
                error_array.append(total_error)
                epoch_array.append(count_epoch)
            W0.append(self.WEIGHT_hidden)
            W1.append(self.WEIGHT_output)
```

```
            count_epoch += 1
        self.show_err_graphic(error_array,epoch_array)
        plt.plot(W0[0])
        plt.title('Weight Hidden update during training')
        plt.legend(['neuron1', 'neuron2', 'neuron3', 'neuron4', 'neuron5'])
        plt.ylabel('Value Weight')
        plt.show()
        plt.plot(W1[0])
        plt.title('Weight Output update during training')
        plt.legend(['neuron1', 'neuron2', 'neuron3'])
        plt.ylabel('Value Weight')
        plt.show()
        return self
```

OUTPUT:

```
Epoch  1 - Total Error:  0.08939914265311079
Epoch  50 - Total Error:  0.05376852115527734
Epoch  100 - Total Error:  0.03773105860576402
Epoch  150 - Total Error:  0.0311644504972821
Epoch  200 - Total Error:  0.028406909135682064
Epoch  250 - Total Error:  0.02669256194214157
Epoch  300 - Total Error:  0.02524003405939519
Epoch  350 - Total Error:  0.02384525367109564
Epoch  400 - Total Error:  0.022427464754471382
Epoch  450 - Total Error:  0.020948415886624147
Epoch  500 - Total Error:  0.01936513137210054
Epoch  550 - Total Error:  0.017597424493928912
Epoch  600 - Total Error:  0.01579217059564909
Epoch  650 - Total Error:  0.014194648016335552
Epoch  700 - Total Error:  0.01285558332929372
```

RESULT: The above python code has been successfully executed.

EX.NO: 15

Applying Backpropagation and genetic algorithm

AIM

To implement genetic algorithm

PYTHON CODE:

```
# genetic algorithm search of the one max optimization problem
from numpy.random import randint
from numpy.random import rand
# objective function
def onemax(x):
	return -sum(x)
# tournament selection
def selection(pop, scores, k=3):
	# first random selection
	selection_ix = randint(len(pop))
	for ix in randint(0, len(pop), k-1):
		# check if better (e.g. perform a tournament)
		if scores[ix] < scores[selection_ix]:
```

```
                selection_ix = ix
        return pop[selection_ix]
# crossover two parents to create two children
def crossover(p1, p2, r_cross):
        # children are copies of parents by default
        c1, c2 = p1.copy(), p2.copy()
        # check for recombination
        if rand() < r_cross:
                # select crossover point that is not on the end of the string

        pt = randint(1, len(p1)-2)
                # perform crossover
                c1 = p1[:pt] + p2[pt:]
                c2 = p2[:pt] + p1[pt:]
        return [c1, c2]
# mutation operator
def mutation(bitstring, r_mut):
        for i in range(len(bitstring)):
                # check for a mutation
                if rand() < r_mut:
                        # flip the bit
                        bitstring[i] = 1 - bitstring[i]
# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
        # initial population of random bitstring
        pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
        # keep track of best solution
        best, best_eval = 0, objective(pop[0])
        # enumerate generations
        for gen in range(n_iter):
                # evaluate all candidates in the population
                scores = [objective(c) for c in pop]
                # check for new best solution
                for i in range(n_pop):
                        if scores[i] < best_eval:
```

```
                best, best_eval = pop[i], scores[i]
                print(">%d, new best f(%s) = %.3f" % (gen,  pop[i], scores[i]))
        # select parents
        selected = [selection(pop, scores) for _ in range(n_pop)]
        # create the next generation
        children = list()
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in crossover(p1, p2, r_cross):
                # mutation
                mutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
    return [best, best_eval]
# define the total iterations
n_iter = 100
# bits
n_bits = 20
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
# perform the genetic algorithm search
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

OUTPUT:

>0, new best f([1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1]) = -13.000

>0, new best f([0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1]) = -14.000

>0, new best f([1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0]) = -15.000

>1, new best f([1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -17.000

>2, new best f([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1]) = -18.000

>5, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -19.000

>6, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000

Done!

f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000000

RESULT: The above python code has been successfully executed.