

S.I.E.S College of Arts, Science and Commerce,
Sion(W), Mumbai - 400 022.

CERTIFICATE

This is to certify that Mr./ Miss. Vinay Srinivas Eligeti has successfully completed the necessary course of experiments in the subject of Artificial Intelligence during the academic year **2023- 2024** complying with the requirements of **University of Mumbai**, for the course of **T.Y.BSc. Computer Science [Semester-V]**

Head of the Department
(Computer Science)

-Dr. Manoj Singh

Prof. In-Charge

Mrs. Shivani Deopa

Date: 1th September 2023

College Seal

Index Page

Practical No.	Practical Name	Page No.	Signature
1.	Implementing Simple Search 1 Algorithm.	3	
2.	Implementing Simple Search 2 Algorithm.	5	
3.	Implementing Simple Search 3 Algorithm.	7	
4.	Implement Breadth first search algorithm for map problem.	9	
5.	Implement Iterative deep depth first search for map problem.	11	
6.	Implement A* search algorithm for Romanian map problem.	12	
7.	Implement recursive best-first search algorithm for Romanian map problem.	14	
8.	Implement decision tree learning algorithm.	16	
9.	To Implement Support Vector Machine Algorithm in Python .	18	
10.	Implement Adaboost ensemble learning algorithm.	20	

Practical-1

Implementing Simple Search 1 Algorithm.

Code snippet:

```
import random
OPEN=['S']
map_list={'S':['A','B','C'],
          'A':['S','D'],
          'B':['S','E'],
          'C':['S','F'],
          'D':['A','G'],
          'E':['B','G','F'],
          'F':['C','E'],
          'G':['D','E']}

def movegen(node):
    return map_list[node]
def goaltest(node):
    return node=='G'
def ss1():
    while len(OPEN)>0:
        random.shuffle(OPEN)
        N=OPEN.pop()
        if goaltest(N):
            return "Found"
        else:
            n=movegen(N)
            for i in n:
                if i not in OPEN:
                    OPEN.append(i)
            print("OPEN_LIST",OPEN)
    return "NOT FOUND"
print(ss1())
```

Output:

Thonny - E:\TVBSCCS 23-24\AI\Practical-1\Simple Searching algorithm-1.py @ 5:25

File Edit View Run Tools Help

Simple Searching algorithm-1.py Simple Searching algorithm-2.py Simple Searching algorithm-3.py

```
1 import random
2 OPEN=['S']
3 map_list={'S':['A','B','C'],
4           'A':['S','D'],
5           'B':['S','E'],
6           'C':['S','F'],
7           'D':['A','G'],
8           'E':['B','G','F'],
9           'F':['C','E'],
10          'G':['D','E']}
11
12 def movegen(node):
13     return map_list[node]
14 def goaltest(node):
15     return node=='G'
16 def ssl():
```

Shell

```
>>> %Run 'Simple Searching algorithm-1.py'
OPEN_LIST [['S', None]]
Picked: ['S']
OPEN_LIST [['C', ['S', None]], ['B', ['S', None]], ['A', ['S', None]]]
Picked: ['S', 'A']
OPEN_LIST [['B', ['S', None]], ['D', ['A', ['S', None]], ['C', ['S', None]]]
Picked: ['S', 'A', 'C']
OPEN_LIST [['F', ['C', ['S', None]], ['B', ['S', None]], ['D', ['A', ['S', None]]]
Picked: ['S', 'A', 'C', 'D']
OPEN_LIST [['B', ['S', None]], ['F', ['C', ['S', None]], ['G', ['D', ['A', ['S', None]]]]
Picked: ['S', 'A', 'C', 'D', 'G']
GOAL Found
Path: SADG
G
>>>
```

Activate Windows
Go to Settings to activate Windows.

Python 3.7.9

Type here to search

28°C 20:07 13-09-2023

Practical-2

Implementing Simple Search 2 Algorithm.

Code snippet:

```
import random
OPEN=['S']
CLOSED=[]
map_list={'S':['A','B','C'],
          'A':['S','D'],
          'B':['S','E'],
          'C':['S','F'],
          'D':['A','G'],
          'E':['B','G','F'],
          'F':['C','E'],
          'G':['D','E']}

def movegen(node):
    return map_list[node]
def goaltest(node):
    return node=='G'
def ss2():
    while len(OPEN)>0:
        random.shuffle(OPEN)
        N=OPEN.pop()
        CLOSED.append(N)
        if goaltest(N):
            return "FOUND"
        else:
            n=movegen(N)
            for i in n:
                if i not in CLOSED and i not in OPEN:
                    OPEN.append(i)
            print("OPEN_LIST",OPEN)
            print("CLOSED_LIST",CLOSED)
    return "NOT FOUND"
print(ss2())
```

Output:

Thonny - E:\TVBSCCS 23-24\AI\Practical-1\Simple Searching algorithm-2.py @ 3:10

File Edit View Run Tools Help

Simple Searching algorithm-1.py Simple Searching algorithm-2.py Simple Searching algorithm-3.py

```
1 import random
2 OPEN=['S']
3 CLOSED=[]
4 map_list={'S':['A','B','C'],
5           'A':['S','D'],
6           'B':['S','E'],
7           'C':['S','F'],
8           'D':['A','G'],
9           'E':['B','G','F'],
10          'F':['C','E'],
11          'G':['D','E']}
12
13 def movegen(node):
14     return map_list[node]
15 def goaltest(node):
16     return node=='G'
```

Shell

```
>>> %Run 'Simple Searching algorithm-2.py'
OPEN_LIST ['A', 'B', 'C']
CLOSED_LIST ['S']
OPEN_LIST ['C', 'A', 'E']
CLOSED_LIST ['S', 'B']
OPEN_LIST ['A', 'C', 'G', 'F']
CLOSED_LIST ['S', 'B', 'E']
OPEN_LIST ['F', 'G', 'A']
CLOSED_LIST ['S', 'B', 'E', 'C']
OPEN_LIST ['A', 'G']
CLOSED_LIST ['S', 'B', 'E', 'C', 'F']
OPEN_LIST ['G', 'D']
CLOSED_LIST ['S', 'B', 'E', 'C', 'F', 'A']
FOUND
>>>
```

Activate Windows
Go to Settings to activate Windows.

Python 3.7.9

Type here to search

Wat...

2008
13-09-2023

Practical-3**Implementing Simple Search 3 Algorithm.****Code snippet:**

```

import random
OPEN=[['S',None]]
CLOSED=[]
map_list={'S':['A','B','C'],
          'A':['S','D'],
          'B':['S','E'],
          'C':['S','F'],
          'D':['A','G'],
          'E':['B','G','F'],
          'F':['C','E'],
          'G':['D','E']}

def movegen(node):
    return map_list[node]
def goaltest(node):
    return node=='G'
def returnpath(path):
    if path is not None:
        return str(path[0])+returnpath(path[1])
    else:
        return ""
def ss3():
    while len(OPEN)>0:
        random.shuffle(OPEN)
        print("OPEN_LIST",OPEN)
        M=OPEN.pop()
        N=M[0]
        CLOSED.append(N)
        print("Picked: ",CLOSED)
        if goaltest(N):
            print("GOAL Found")
            print("Path: ",returnpath(M[::-1]))
            return N
        else:
            neigh=movegen(N)
            for node in neigh:
                if node not in CLOSED and node not in OPEN:
                    new_list=[node,M]
                    OPEN.append(new_list)
    print("Not Found")
print(ss3())

```

Output:

Thonny - E:\TVBSCCS 23-24\AI\Practical-1\Simple Searching algorithm-3.py @ 4:29

File Edit View Run Tools Help

Simple Searching algorithm-1.py Simple Searching algorithm-2.py Simple Searching algorithm-3.py

```
1 import random
2 OPEN=[['S',None]]
3 CLOSED=[]
4 map_list={'S':['A','B','C'],
5           'A':['S','D'],
6           'B':['S','E'],
7           'C':['S','F'],
8           'D':['A','G'],
9           'E':['B','G','F'],
10          'F':['G','D']}
```

Shell

```
>>> %Run 'Simple Searching algorithm-3.py'
OPEN_LIST [['S', None]]
Picked: ['S']
OPEN_LIST [['A', ['S', None]], ['B', ['S', None]], ['C', ['S', None]]]
Picked: ['S', 'C']
OPEN_LIST [['B', ['S', None]], ['F', ['C', ['S', None]], ['A', ['S', None]]]
Picked: ['S', 'C', 'A']
OPEN_LIST [['D', ['A', ['S', None]], ['F', ['C', ['S', None]], ['B', ['S', None]]]
Picked: ['S', 'C', 'A', 'B']
OPEN_LIST [['D', ['A', ['S', None]], ['F', ['C', ['S', None]], ['E', ['B', ['S', None]]]
Picked: ['S', 'C', 'A', 'B', 'E']
OPEN_LIST [['F', ['E', ['B', ['S', None]]], ['G', ['E', ['B', ['S', None]], ['D', ['A', ['S', None]], ['F', ['C', ['S', None]]]
Picked: ['S', 'C', 'A', 'B', 'E', 'F']
OPEN_LIST [['G', ['E', ['B', ['S', None]], ['F', ['E', ['B', ['S', None]], ['D', ['A', ['S', None]]]
Picked: ['S', 'C', 'A', 'B', 'E', 'F', 'D']
OPEN_LIST [['F', ['E', ['B', ['S', None]], ['G', ['E', ['B', ['S', None]], ['G', ['D', ['A', ['S', None]]]
Picked: ['S', 'C', 'A', 'B', 'E', 'F', 'D', 'G']
GOAL Found
Path: SADG
G
>>>
```

Activate Windows
Go to Settings to activate Windows.

Python 3.7.9

Type here to search

Wat...

20:09
13-09-2023

Practical-4

Implement Breadth first search algorithm for map problem.

Code snippet:

```
graph = {
    'S':['A','B','C'],
    'A':['S','D'],
    'B':['S','E'],
    'C':['S','F'],
    'D':['A','G'],
    'E':['B','G','F'],
    'F':['C','E'],
    'G':['D','E']
}
visited = []
queue = []

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, 'S')
```

Output:

Thonny - E:\TYBSCCS 23-24\AI\Practical-2\BFS.py @ 3:14

File Edit View Run Tools Help

BFS.py x

```
1 graph = {
2     'S': ['A', 'B', 'C'],
3     'A': ['S', 'D'],
4     'B': ['S', 'E'],
5     'C': ['S', 'F'],
6     'D': ['A', 'G'],
7     'E': ['B', 'G', 'F'],
8     'F': ['C', 'E'],
9     'G': ['D', 'E']
10 }
11 visited = []
12 queue = []
13
14 def bfs(visited, graph, node):
15     visited.append(node)
16     queue.append(node)
17
18     while queue:
19         m = queue.pop(0)
20         print(m, end = " ")
21
22         for neighbour in graph[m]:
```

Shell x

```
>>> %cd 'E:\TYBSCCS 23-24\AI\Practical-2'
>>> %Run BFS.py

Following is the Breadth-First Search
S A B C D E F G

>>>
```

Activate Windows
Go to Settings to activate Windows.

Python 3.7.9

28°C 20:10 13-09-2023

Practical-5

Implement Iterative deep depth first search for map problem.

Code snippet:

```
graph={
    'S':['A','B','C'],
    'A':['S','D'],
    'B':['S','E'],
    'C':['S','F'],
    'D':['A','G'],
    'E':['B','G','F'],
    'F':['C','E'],
    'G':['D','E']
}
def dfs(visited, graph, node):
    if node not in visited:
        print (node,end=" ")
        visited.append(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
print("Following is the Depth-First Search")
dfs([], graph, 'S')
```

Output:

The screenshot shows the Thonny IDE interface. The top pane displays the Python code for the DFS algorithm. The bottom pane shows the execution output in the shell.

```
graph={
    'S':['A','B','C'],
    'A':['S','D'],
    'B':['S','E'],
    'C':['S','F'],
    'D':['A','G'],
    'E':['B','G','F'],
    'F':['C','E'],
    'G':['D','E']
}
def dfs(visited, graph, node):
    if node not in visited:
        print (node,end=" ")
        visited.append(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
print("Following is the Depth-First Search")
dfs([], graph, 'S')
```

Shell Output:

```
>>> %Run DFS.py
Following is the Depth-First Search
S A D G E B F C
>>>
>>>
>>>
>>> |
```

Python 3.7.9

Practical-6**Implement A* search algorithm for Romanian map problem.****Code snippet:**

```

#A star
nodeList={'mumbai':[("delhi",1200),("nasik",350),("goa",800),("pune",130)],
          "delhi":[("nasik",375),("mumbai",1200)],
          "nasik":[("indore",600),("delhi",375),("mumbai",350),("nagpur",600)]
          ,
          "indore":[("nasik",400)],
          "nagpur":[("nasik",600),("pune",450)],
          "pune":[("mumbai",130),("nagpur",450),("blore",550)],
          "blore":[("hyd",110),("goa",750)],
          "goa":[("blore",750),("hyd",850),("mumbai",800)],
          "hyd":[("blore",750),("goa",750)]}
hd={"mumbai":790,"delhi":1515,"nasik":1140,"indore":1540,"nagpur":1110,"pune":
660,"blore":110,"goa":850,"hyd":0}
openList=[("mumbai",700)]
closedList=[]
def goalTest(node):
    return node=="hyd"
def moveGen(node):
    return nodeList[node[0]]
def sort(mylist):
    for i in range(len(mylist)):
        for j in range(0,len(mylist)-i-1):
            if (mylist[j][1]>mylist[j+1][1]):
                temp=mylist[j]
                mylist[j]=mylist[j+1]
                mylist[j+1]=temp
    return (mylist)
def AStar():
    while(len(openList)>0):
        sort(openList)
        print("Open List Contains:",openList)
        node=openList.pop(0)
        closedList.append((node[0],hd[node[0]]))
        print("Picked node:",node)
        if (goalTest(node[0])):
            return "Goal Found"
        else:
            neighbours=moveGen(node)
            print("Neighbours of ",node," are: ",neighbours)
            for node in neighbours:
                if node not in openList and node[0] not in closedList[0]:
                    tup=(node[0],(node[1]+hd[node[0]]))
                    openList.append(tup)

```

```

        return "Goal Not Found"
AStar()
'''
Goal='hyd'
finding hd->>selecting mumbai
hd1=mumbai->delhi(1200)->nasik(375)->indore(600)->nasik(375)loop hence
discarded
hd2=mumbai->nasik(350)->discarded
hd3=mumbai->goa(800)->blore(750)->hyd
hd4=
hd5=
hd6=
'''

```

Output:

```

Thonny - E:\TVBSCCS 23-24\AI\AStar_Algorithm.py @ 10:45
File Edit View Run Tools Help

AStar_Algorithm.py x
1 #A star
2 nodeList=[('mumbai': [('delhi', 1200), ('nasik', 350), ('goa', 800), ('pune', 130)],
3             'delhi': [('nasik', 375), ('mumbai', 1200)],
4             'nasik': [('indore', 600), ('delhi', 375), ('mumbai', 350), ('nagpur', 600)],
5             'indore': [('nasik', 400)],
6             'nagpur': [('nasik', 600), ('pune', 450)],
7             'pune': [('mumbai', 130), ('nagpur', 450), ('blore', 550)],
8             'blore': [('hyd', 110), ('goa', 750)],
9             'goa': [('blore', 750), ('hyd', 850), ('mumbai', 800)],
10            'hyd': [('blore', 750), ('goa', 750)]]
11 hd=({'mumbai': 790, 'delhi': 1515, 'nasik': 1140, 'indore': 1540, 'nagpur': 1110, 'pune': 660, 'blore': 110, 'goa': 850, 'hyd': 0})
12 openList=[('mumbai', 700)]
13 closedList=[]
14 def goalTest(node):
15     return node==hd"

Shell x
>>> %cd 'E:\TVBSCCS 23-24\AI'
>>> %Run AStar_Algorithm.py

Open List Contains: [('mumbai', 700)]
Picked node: ('mumbai', 700)
Neighbours of ('mumbai', 700) are: [('delhi', 1200), ('nasik', 350), ('goa', 800), ('pune', 130)]
Open List Contains: [('pune', 790), ('nasik', 1490), ('goa', 1650), ('delhi', 2715)]
Picked node: ('pune', 790)
Neighbours of ('pune', 790) are: [('mumbai', 130), ('nagpur', 450), ('blore', 550)]
Open List Contains: [('blore', 660), ('nasik', 1490), ('nagpur', 1560), ('goa', 1650), ('delhi', 2715)]
Picked node: ('blore', 660)
Neighbours of ('blore', 660) are: [('hyd', 110), ('goa', 750)]
Open List Contains: [('hyd', 110), ('nasik', 1490), ('nagpur', 1560), ('goa', 1600), ('goa', 1650), ('delhi', 2715)]
Picked node: ('hyd', 110)

>>>

```

Activate Windows
Go to Settings to activate Windows.

Python 3.7.9

Practical-7

Implement recursive best-first search algorithm for Romanian map problem.

Code snippet:

```
#Best First Search Algorithm
map_list={ 'Mumbai':[( 'Pune',750),('Delhi',1500),('Goa',1300)],
           'Goa':[( 'Mumbai',1200)],
           'Delhi':[( 'Mumbai',1200),('Guwahati',100),('Pune',750)],
           'Chennai':[( 'Pune',750)],
           'Kolkata':[( 'Guwahati',100),('Pune',750)],
           'Pune':[( 'Mumbai',1200),('Kolkata',0),('Chennai',1600),('Delhi',1500
)],
           'Guwahati':[( 'Delhi',1500),('Kolkata',0)]
}
OPEN=[[( 'Mumbai',1200),None]]
CLOSED=[]
def movegen(node):
    return map_list[node]
def goaltest(node):
    return node=='Kolkata'
final=[]
def reconstructpath(path):
    if path is None:
        return ""
    else:
        final.append(path[0][0])
        reconstructpath(path[1])
        return final
def sort(a):
    for i in range(len(a)):
        for j in range(0,len(a)-i-1):
            if ((a[j][0][1]),a[j+1][0][1]):
                (a[j],a[j+1])=(a[j+1],a[j])
    return a
def best():
    while len(OPEN)>0:
        print("Open List:",OPEN)
        x=sort(OPEN)
        seen=x.pop(0)
        N=seen[0][0]
        CLOSED.append(N)
        print("Closed List cotains",CLOSED)
        print("Node Picked:",N)
        if goaltest(N):
            print(reconstructpath(seen)[::-1])
            return "Found"
```

```

else:
    neigh=movegen(N)
    for i in neigh:
        if i[0] not in CLOSED and i not in OPEN:
            new=[i,seen]
            OPEN.append(new)
    return "Not Found"
best()

```

Output:

```

AStar_Algorithm.py x Best First Search Algorithm.py x
1 #Best First Search Algorithm
2 map_list={'Mumbai': [('Pune', 750), ('Delhi', 1500), ('Goa', 1300)],
3           'Goa': [('Mumbai', 1200)],
4           'Delhi': [('Mumbai', 1200), ('Guwahati', 100), ('Pune', 750)],
5           'Chennai': [('Pune', 750)],
6           'Kolkata': [('Guwahati', 100), ('Pune', 750)],
7           'Guwahati': []}
8
9 def movegen(N):
10     neigh=[]
11     for i in N:
12         for j in map_list[i]:
13             new=[j, N[i][0]]
14             if new not in N:
15                 neigh.append(new)
16     return neigh
17
18 def best():
19     OPEN=[]
20     CLOSED=[]
21     if map_list['Mumbai']:
22         OPEN.append(['Mumbai', 1200])
23     while OPEN:
24         node=OPEN[0]
25         CLOSED.append(node)
26         node=node[0]
27         if node=='Mumbai':
28             return node
29         OPEN.remove(node)
30         for i in map_list[node]:
31             new=[i, node]
32             if new not in OPEN and new not in CLOSED:
33                 OPEN.append(new)
34     return "Not Found"
35
36 if __name__ == '__main__':
37     best()
38
Shell x
>>> %cd 'E:\TYBSCCS 23-24\AI\Practical-3'
>>> %Run 'Best First Search Algorithm.py'
Open List: [('Mumbai', 1200), None]
Closed List contains ['Mumbai']
Node Picked: Mumbai
Open List: [('Pune', 750), [('Mumbai', 1200), None], [('Delhi', 1500), [('Mumbai', 1200), None], [('Goa', 1300), [('Mumbai', 1200), None]]]
Closed List contains ['Mumbai', 'Goa']
Node Picked: Goa
Open List: [('Delhi', 1500), [('Mumbai', 1200), None], [('Pune', 750), [('Mumbai', 1200), None]]]
Closed List contains ['Mumbai', 'Goa', 'Pune']
Node Picked: Pune
Open List: [('Delhi', 1500), [('Mumbai', 1200), None], [('Kolkata', 0), [('Pune', 750), [('Mumbai', 1200), None], [('Chennai', 1600), [('Pune', 750), [('Mumbai', 1200), None]]]]]
Closed List contains ['Mumbai', 'Goa', 'Pune', 'Delhi']
Node Picked: Delhi
Open List: [('Chennai', 1600), [('Pune', 750), [('Mumbai', 1200), None], [('Kolkata', 0), [('Pune', 750), [('Mumbai', 1200), None], [('Chennai', 1600), [('Pune', 750), [('Mumbai', 1200), None]]]]]]]
Closed List contains ['Mumbai', 'Goa', 'Pune', 'Delhi', 'Guwahati']
Node Picked: Guwahati
Open List: [('Delhi', 1500), [('Mumbai', 1200), None], [('Kolkata', 0), [('Pune', 750), [('Mumbai', 1200), None], [('Chennai', 1600), [('Pune', 750), [('Mumbai', 1200), None]]]]]]]
Closed List contains ['Mumbai', 'Goa', 'Pune', 'Delhi', 'Guwahati', 'Kolkata']
Node Picked: Kolkata
['Mumbai', 'Pune', 'Delhi', 'Guwahati', 'Kolkata']
>>>

```

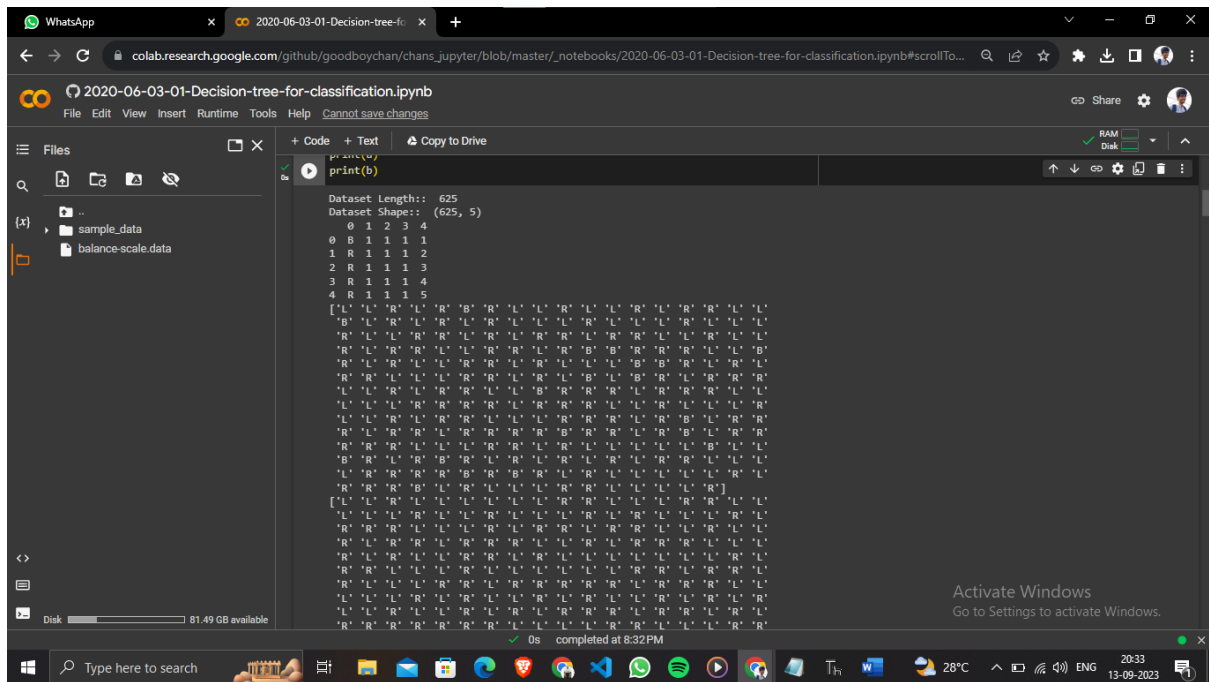
Practical-8

Implement decision tree learning algorithm.

Code snippet:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
balance_data=pd.read_csv("balance-scale.data",sep=" ",header=None)
print("Dataset Length:: ",len(balance_data))
print("Dataset Shape:: ",balance_data.shape)
print(balance_data.head())
X=balance_data.values[:, 1:5]#predicator
Y=balance_data.values[:,0]#target attr (B,R,L)
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.4,random_state=100)
clf_entropy=DecisionTreeClassifier(criterion="entropy",random_state=100,max_depth=3,min_samples_leaf=5)
clf_entropy.fit(X_train,y_train)
clf_gini=DecisionTreeClassifier(criterion="gini",random_state=100,max_depth=3,min_samples_leaf=5)
clf_gini.fit(X_train,y_train)
print(y_test)
y_pred_en=clf_entropy.predict(X_test)
y_pred_gini=clf_gini.predict(X_test)
print(y_pred_en)
print(y_pred_gini)
a=accuracy_score(y_pred_en,y_test)*100
b=accuracy_score(y_pred_gini,y_test)*100
print(a)
print(b)
```

Output:



```
print(a)
print(b)

Dataset Length:: 625
Dataset Shape:: (625, 5)
0 1 2 3 4
0 8 1 1 1 1
1 1 1 1 1 2
2 1 1 1 1 3
3 1 1 1 1 4
4 1 1 1 1 5

['L' 'L' 'R' 'L' 'R' 'B' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'L'
 'B' 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'L'
 'R' 'L' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'B' 'B' 'R' 'R' 'L' 'L' 'B'
 'R' 'L' 'R' 'L' 'L' 'R' 'R' 'L' 'R' 'L' 'L' 'B' 'B' 'R' 'L' 'R' 'L'
 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'B' 'R' 'R' 'L' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R' 'L' 'R' 'B' 'L' 'R' 'R'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'B' 'R' 'L' 'R' 'L' 'B' 'L' 'R' 'R'
 'R' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'B' 'L' 'L'
 'B' 'R' 'L' 'R' 'B' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'L'
 'L' 'R' 'R' 'R' 'R' 'B' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'R' 'L'
 'R' 'R' 'R' 'B' 'L' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'L'
 ['L' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'L'
 'R' 'L' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L'
 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'L' 'B'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'L' 'L'
 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
```

Practical-9

To Implement Support Vector Machine Algorithm in Python.

Code snippet:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn import datasets
balance_data=pd.read_csv('balance-scale.data',sep=",",header=None)
print("Dataset Length:: ",len(balance_data))
print("Dataset Shape:: ",balance_data.shape)
print(balance_data.head())
X=balance_data.values[:,1:5]
Y=balance_data.values[:,0]
X_train, X_test, y_train,
y_test=train_test_split(X,Y,test_size=0.3,random_state=100)
svclassifier=SVC(kernel="linear")
svclassifier.fit(X_train,y_train)
print(y_test)
y_pred=svclassifier.predict(X_test)
print(y_pred)
print(accuracy_score(y_pred,y_test)*100)
```

Or

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm
from sklearn import datasets
cancer_data=datasets.load_breast_cancer()
X_train, X_test, y_train,
y_test=train_test_split(cancer_data.data,cancer_data.target,
test_size=0.4,random_state=109)
cls=svm.SVC(kernel="linear")
cls.fit(X_train,y_train)
pred=cls.predict(X_test)
print("accuracy: ",metrics.accuracy_score(y_test,y_pred=pred)*100)
print(metrics.classification_report(y_test,y_pred=pred))
```

Output:

2020-06-03-01-Decision-tree-for-classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Files

- sample_data
- balance-scale.data

```
[2] 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R' 'L' 'R' 'B' 'L' 'R' 'R'
      'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'B' 'R' 'R' 'B' 'R' 'B' 'L' 'R' 'R'
      'R' 'R' 'R' 'L' 'L' 'L' 'R' 'R']
      91.48936170212765
```

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm
from sklearn import datasets
cancer_data=datasets.load_balance_scale()
X_train, X_test, y_train, y_test=train_test_split(cancer_data.data,cancer_data.target, test_size=0.4,random_state=109)
cls=svm.SVC(kernel="linear")
cls.fit(X_train,y_train)
pred=cls.predict(X_test)
print("accuracy: ",metrics.accuracy_score(y_test,y_pred=pred)*100)
print(metrics.classification_report(y_test,y_pred=pred))
```

accuracy: 96.49122807017544

	precision	recall	f1-score	support
0	0.97	0.94	0.96	90
1	0.96	0.98	0.97	138
accuracy			0.96	228
macro avg	0.97	0.96	0.96	228
weighted avg	0.96	0.96	0.96	228

Activate Windows
Go to Settings to activate Windows.

import pandas as pd

0s completed at 8:47 PM

Practical-10**Implement Adaboost ensemble learning algorithm.****Code snippet:**

```

import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names= ['preg','plas','pres', 'skin', 'test','mass', 'pedi','age', 'class']
dataframe = pandas.read_csv(url, names=names)
array=dataframe.values
print("Dataset Length:: ", len(dataframe))
print("Dataset Shape:: ", dataframe.shape)
print(dataframe.head())
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees=30
kfold = model_selection.KFold(n_splits=10)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean()*100)

```

Or

```

import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names= ['preg','plas','pres', 'skin', 'test','mass', 'pedi','age', 'class']
dataframe = pandas.read_csv(url, names=names)
array=dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10)
# create the sub models
estimators = []
model1=LogisticRegression (solver='lbfgs', max_iter=1000)
estimators.append(('logistic', model1))
model2=DecisionTreeClassifier()
estimators.append(('cart', model2))

```

```

model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble=VotingClassifier (estimators)
results = model_selection.cross_val_score (ensemble, X, Y, cv=kfold)
print(results.mean()*100)

```

Output:

The screenshot shows a Jupyter Notebook titled "2020-06-03-01-Decision-tree-for-classification.ipynb" in a Google Colab environment. The code in the notebook imports necessary libraries, loads the 'pima-indians-diabetes' dataset, and uses an AdaBoostClassifier with 30 estimators (SVC) for cross-validation. The output displays the dataset's length and shape, followed by a preview of the first five rows of the data.

```

import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array=dataframe.values
print("Dataset Length: ", len(dataframe))
print("Dataset Shape: ", dataframe.shape)
print(dataframe.head())
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees=30
kfold = model_selection.KFold(n_splits=10)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean()*100)

```

Dataset Length: 768
Dataset Shape: (768, 9)

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

76.04579630895421