DMC 8212

MASTER OF COMPUTER APPLICATION

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

CENTRE FOR DISTANCE AND ONLINE EDUCATION
ANNA UNIVERSITY
CHENNAI - 600 025

Course Writer
Dr.Arockia Xavier Annie. R
Associate Professor
Department of CSE
Anna University
Chennai – 600 025.
9677032001

Course Reviewer
Dr.R.S.Bhuvaneswaran
Professor
Ramanujan Computing Centre
Anna University, Chennai.
9962524665 /8001

Printed by

Gunasundari Modern
Art Printers,18A/34,
Ammaiappan Street,
Chennai - 600 021.
For

CENTRE FOR DISTANCE AND ONLINE EDUCATION
ANNA UNIVERSITY
CHENNAI – 600 015.

ARTIFICIAL INTELLIGENCEAND MACHINE LEARNING LABORATORY

SYLLABUS

OBJECTIVES:

- To familiarize with the machine learning algorithms and implement in practical situations.
- To involve the students to practice AI algorithms and techniques.
- Learn to use different algorithms for real time data sets.

OUTCOMES:

- Apply the techniques of Problem Solving in Artificial Intelligence.
- Implement Knowledge and Reasoning for real world problems.
- Model the various Learning features of Artificial Intelligence
- Analyze the working model and features of Decision tree
- Apply k-nearest algorithm for appropriate research problem.

LIST OF EXPERIMENTS :

- Write a program to illustrate problem solving as a search.
- Write a program to illustrate local search algorithms.
- Write a program to demonstrate logical agents.
- Evaluate forward chainer and rule base on at least four different databases Try to create at least one database that demonstrates an interesting feature of the domain, or an interesting feature of forwardchaining in general.
- Demonstrate agent based on propositional logic.
- Write a program to implement the naïve Bayesian classifier for a sample training data set. Compute the accuracy of the classifier, considering few test data sets.
- Write a program to construct a Bayesian network considering medicaldata. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.
- Apply EM algorithm to cluster a set of data stored in a .CSV file.
- Write a program to implement k-Nearest Neighbor algorithm to classify the data set

- Apply the technique of pruning for a noisy data monk2 data, and derive the decision tree from this data. Analyze the results by comparing the structure of pruned and unpruned tree.
- Build an Artificial Neural Network by implementing the Backpropogation algorithm and test the same using appropriate data sets.
- Implement support Vector Classification for linear kernel.
- Implement Logistic Regression to classify the problems such as spam detection. Diabetes predictions so on.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

CONTENT

LESSON - I WRITE A PROGRAM TO ILLUSTRATE PROBLEM SOLVING IN UNINFORMED SEARCH

CONTENTS

UNINFORMED SEARCH

There may be many solutions to a particular problem. If you can think of the task you want your agent to perform in these terms,then you will need to write aproblem solving agent which uses search. It is important to identify the scope ofyour task in terms of the problems which will need to be solved.

BREADTH FIRST SEARCH - BFS

BFS uses the first node that it finds and it is in FIFO ordering for node expansion i.e.,it uses a FIFO queue as the queuing mechanism.

Properties of BFS:

- **Completeness:** Yes. BFS is guaranteed to find a solution if one exists.
- **Optimality:** Only if action costs are all 1. BFS finds the shallowest goal node,the one requiring the least number of actions or edges. This is optimal only if all actions have the same cost.
- **Time complexity: $O(b^S)$,**where s is the depth of the shallowest solution.
- **Space complexity: $O(b^S)$**

BFS Algorithm

Listing 1: BFS Algorithm

BFS(G,s)//Where G is the graph and s is the source node let Q be queue.
Q.enqueue(s)//Insertings in queue until all its neighbour vertices are marked. marks as visited.
while(Q is not empty)
//Removing that vertex from queue,whose neighbour will be visited nowv
dequeue ( ) // p r o c e s s i n g a l l the neighbours o f v
for all neighbours w of v in Graph G
if w is not visited

BFS IN PYTHON

Listing 2: BFS code

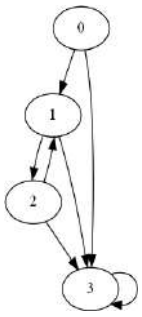
```
# −*− coding : utf −8 −*−
# Python3 Program to print BFS traversal from a given source vertex.# BFS(
i n t gs ) traverses vertices
from gs.from collectionsimport
defaultdict import pydot
# This class represents a directed graph using adjacency list representationclass
Graph :
# Constructor
def __ init__ ( s e l f ) :
# defaultdictionary to store graph uses listself .
graph = defaultdict(list)
# f u n c t i o n to add an edge to graph
def add Edge ( s e l f ,u ,v ) :
self . graph [ u ].append ( v )
# Function to p r i n t a BFS o f graphdef
BFS ( s e l f ,gs ) :
# Mark a l l the vertices as not visited
visited = [ False ] * ( max( s e l f . graph ) + 1 )
```

```
# Create a queue f o r BFS
queue = [ ]
# Mark the so ur ce node as visited and
enqueue i t queue . append ( gs )
visited [ gs ] = True
while queue :
# Dequeue a v e r t e x from queue and p r i n t i t
gs = queue . pop ( 0 )
p r i n t ( gs ,end = " " )
# Get a l l adjacent vertices o f the dequeued vertex s .
# If a adjacent has not been visited,then mark i t visited and enqueue i t
f o r i in s e l f . graph [ gs ] :
i f visited [ i ] == False :
queue . append ( i )
visited [ i ] = True
#  Driver /  Main  code
#Prepare the  nodes  f o r  visualizing  the  input  graph
vis Graph=pydot . Dot ( graph_type=' digraph ' )
node0 = pydot . Node ( " 0 " )
vis Graph . add_node ( node0 )
node1 = pydot . Node ( " 1 " )
vis Graph . add_node ( node1 )
node2 = pydot . Node ( " 2 " )
vis Graph . add_node ( node2 )
node3 = pydot . Node ( " 3 " )
vis Graph . add_node ( node3 )
# Construct the graph
#Also add the edges to the v i s u a l Graph
g = Graph ( )
g . addEdge ( 0 ,1 )
vis Graph . add_edge ( pydot . Edge ( node0 ,node1 ) )
g . addEdge ( 0 ,3 )
```

```
vis Graph . add_edge ( pydot . Edge ( node0 ,node3 ) )
g . addEdge ( 1 ,2 )
vis Graph . add_edge ( pydot . Edge ( node1 ,node2 ) )
g . addEdge ( 1 ,3 )
vis Graph . add_edge ( pydot . Edge ( node1 ,node3 ) )
g . addEdge ( 2 ,1 )
vis Graph . add_edge ( pydot . Edge ( node2 ,node1 ) )
g . addEdge ( 2 ,3 )
vis Graph . add_edge ( pydot . Edge ( node2 ,node3 ) )
g . addEdge ( 3 ,3 )
vis Graph . add_edge ( pydot . Edge ( node3 ,node3 ) )
vis Graph . write_png ( ' graph . png ' )
p r i n t ( " Breadth F i r s t Traversal i s :
" " ( s t a r t i n g from v e r t e x 0 ) " )
#run the BFS f u n c t i o n
g . BFS( 0 )
```

Input Graph

Figure 1: Output




```
Breadth First Traversal is : (starting from vertex 2) 2 1 3
Breadth First Traversal is : (starting from vertex 0) 0 1 3 2
```

DEPTH FIRST SEARCH - DFS

DFS chooses the most recently found nodes for expansion,i.e.,it uses LIFO ordering for node expansion. In other terms,it uses a stack as the queuing mechanism.

Properties of DFS:

- **Completeness:** DFS is complete only if we prevent cycles i.e.,Graph search version of DFS is complete.
- **Optimality:** DFS is not optimal. It does not always find the cheapest solution. It only returns the first solution it finds,no matter how expensive it is. Hence optimal solution is not available with DFS.
- **Time complexity:** O(bm),where b is the branching factor and m is the maximum depth.
- **Space complexity:** O(bm),for the tree search version and O(bm) for the graph search version

DFS Algorithm

Listing 3: Depth First Search

```
DFS− iterative ( G,s ) : // Where G i s graph and s i s source vertexlet S
be stack
S . push ( s )// Insertings in stackmark
s as visited .
while ( S i s not empty )//Pop a vertex from stack tovis
it next
v = S . top ( )
S . pop ( )// Push a l l the neighbours o f v in sta
ck that are not visited f o r a l l neighbours wof v in
Graph G:
if w is not visited :
S . push ( w ) mark w
as visited
```

DFS IN PYTHON

Listing 4: Depth First Search in Python

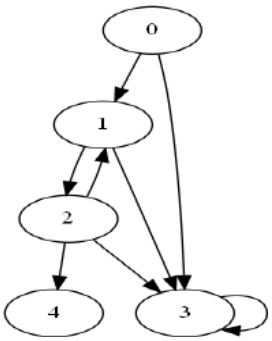
```
# −*− coding : utf −8 −*−
from collections import default dict
import pydot
# This class represents adirected graph using adjacency list
representation
c l a s s Graph :
```

```
# Constructor
 def __ init__ ( self) :
#default dictionary to store graph
self. graph = defaultdict( l i s t )
# f u n c t i o n to add an edge to graph
def add Edge ( s e l f ,u ,v ) :
s e l f . graph [ u ] . append ( v )
# A function used by DFS
def DFSUtil ( s e l f ,v ,visited ) :
# Mark the current node as visited and print it
visited . add ( v )
p r i n t ( v ,end= ‘ ‘ )
#recursively find for all the vertices adjacent to
thisvertex
for neighbour in self . graph [ v ] :
i f neighbour not in visited :
self . DFSUtil ( neighbour ,visited )
# The function to do DFS traversal.It uses recursive
DFSUtil ( )
def DFS ( self ,v ) :
# Create a set to store visited vertices
visited = s e t ( )
# Callthe recursive helper function to p r i n t DFS
traversal
self. DFSUtil ( v ,visited )
# Driver code
#Prepare the nodes for visualizing the input graph
vis Graph=pydot.Dot ( graph_type=’ digraph ‘ )
node0 = pydot . Node ( “ 0 “ )
vis Graph . add_node ( node0 )
node1 = pydot . Node ( “ 1 “ )
vis Graph . add_node ( node1 )
```

```
node2 = pydot . Node ( “ 2 “ )
vis Graph . add_node ( node2 )
node3 = pydot . Node ( “ 3 “ )
vis Graph . add_node ( node3 )
node4 = pydot . Node ( “ 4 “ )
vis Graph . add_node ( node4 )
# Construct the graph
 #Also add the edges to the v i s u a l Graph
g = Graph ( )
g . addEdge ( 0 ,1 )
vis Graph . add_edge ( pydot . Edge ( node0 ,node1 ) )
g . addEdge ( 0 ,3 )
vis Graph . add_edge ( pydot . Edge ( node0 ,node3 ) )
g . addEdge ( 1 ,2 )
vis Graph . add_edge ( pydot . Edge ( node1 ,node2 ) )
g . addEdge ( 1 ,3 )
vis Graph . add_edge ( pydot . Edge ( node1 ,node3 ) )
g . addEdge ( 2 ,1 )
vis Graph . add_edge ( pydot . Edge ( node2 ,node1 ) )
g . addEdge ( 2 ,3 )
vis Graph . add_edge ( pydot . Edge ( node2 ,node3 ) )
g . addEdge ( 2 ,4 )
vis Graph . add_edge ( pydot . Edge ( node2 ,node4 ) )
g . addEdge ( 3 ,3 )
vis Graph . add_edge ( pydot . Edge ( node3 ,node3 ) )
vis Graph . write_png ( ‘ DFSgraph . png ‘ )
p r i n t ( “ Following i s DFS from ( s t a r t i n g from v e r t e x 2 ) “ )
g . DFS( 2 )
```

Input Graph

Figure 2: Output



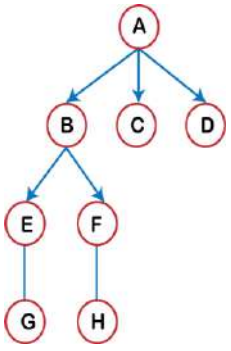
Following is DFS from (starting from vertex 2) 2 1 3 4

CONCLUSION

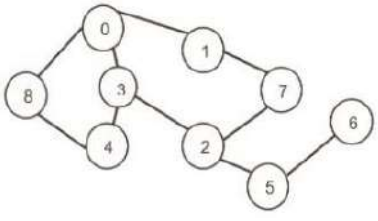
The uninformed search strategies such as BFS and DFS were explained in this section and the example run was done in graph search. For graph searches,the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space.

EXERCISES

1. Find if the node E is present in the given tree. Input Graph



2. Order the nodes of the given graph using (i) BFS and (ii) DFS Input Graph



LESSON - 2 WRITE A PROGRAM TO ILLUSTRATE INFORMED SEARCH

CONTENTS

INFORMED SEARCH

The key idea behind informed search algorithms is that of heuristics. A search using domain-specific knowledge is Informed Search. Suppose that we have a way to estimate how close a state is to the goal,with an evaluation function. The general strategy is to expand the best state in the open list first. It's called a best-first search or ordered state-space search. In general the evaluation function is imprecise,which makes the method a heuristic (works well in most cases). The evaluation is often based on empirical observations.As an example,the routing problem (finding a path from one pt to another),we could use Euclidean distance as heuristic.

A* SEARCH

A* combines the ideas behind both greedy search and Uniform Cost Search. It uses the sum of path cost from start state and the heuristic estimate to a goal state,to order nodes for expansion,i.e.,A* orders nodes by g(n)+f(n),where g(n) is the cumulative path cost (backward cost) from start state to state n and f(n) is the heuristic estimate (forward cost) from state n to a goal node.

Properties of A* Search:

- **Completeness:** Yes
- **Optimality:** Yes,if we use an admissible heuristic
- **Time Complexity**: Highly dependent on the heuristic
- **Space complexity:** Highly dependent on the heuristic

A* SEARCH ALGORITHM

Listing 5: A* Algorithm

```
# −*− coding : utf −8 −*−
```

```
import heapq
class Priority Queue :
def __ init__ (self,priorityFunction):
self.priorityFunction=priorityFunction
self. heap = [ ]
def push ( s e l f ,item ) :
heapq . heappush ( s e l f . heap ,
(self.priorityFunction(item),item))
def pop ( s e l f ) :
(_,item ) = heapq . heappop ( self . heap )
return item
def empty ( self ) :
return len( self. heap ) == 0
def astar Graph Search ( problem ,heuristic ) :
# A* uses path cost from startstate + heuristic estimate to a goal
totalCost = lambda state : len ( state [ 1 ] ) + heuristic ( state )
return graph Search ( problem ,Priority Queue ( totalCost) )
# problem i s an instance of PacmanProblem
# food i s the position o f the food pellet ( r ,c )
def pacmanPath F i n d e r ( problem ,food ) :
manhattan Distance Heuristic = lambda s t a t e :
abs ( s t a t e [ 0 ] [ 0 ] − food [ 0 ] ) + abs ( s t a t e [ 0 ] [ 1 ] − food [ 1 ] )
return astar Graph Search ( problem ,
manhattan Distance Heuristic )
```

Listing 6: A* Search in Python

```
# −*− coding : utf −8 −*−
c l a s s Node ( ) :
“””A node c l a s s f o r A*Pathfinding “””
def__ init__ ( s e l f ,parent=None ,position=None ) :
s e l f . parent = parent
self.position=position
self.g=0
```

```
self.h=0
self.f=0
def__ eq__ ( s e l f ,other ) :
returnself.position==other.position
def astar ( maze ,s t a r t ,end ) :
“”” Returns a l i s t o f t u p l e s as a path from the given s t a r t to the given end
in the given maze”””
# Create s t a r t and end node
start_ node = Node ( None ,s t a r t )
start_ node . g = start_ node . h = start_ node . f = 0
end_node = Node ( None ,end )
end_node . g = end_node . h = end_node . f = 0
# Initializebothopenandclosedlistopen_list=[ ]
closed_list=[ ]
# Add the start node
open_ l i s t . append ( start_ node )
# Loop until you find the end
while l e n ( open_ l i s t ) > 0 :
# Get the current node
current_ node = open_ l i s t [ 0 ]
current_ index = 0
f o r index ,item in enumerate ( open_ l i s t ) :
i f item . f < current_ node . f :
current_ node = item
current_ index = index
# Pop current offopen list,add to closedl i s t
open_list.pop(current_index)
closed _ list.append ( current_ node )
# Found the g o a l
i f current_ node == end_node :
path = [ ]
current = current_ node
```

```
while c u r r e n t i s not None :
path . append ( current.position )
current=current.parent
return path [ : : − 1 ]
# Return reversed path
# Generate children
children = [ ]
f o r new_ position in [ ( 0 ,−1) ,( 0 ,1 ) ,( −1 ,0 ) ,( 1 ,
0 ) ,( −1 ,−1) ,( −1 ,1 ) ,( 1 ,−1) ,( 1 ,1 ) ] : #
Adjacent sq ua r es
# Get node position
node_ position = ( current_ node . position [ 0 ] +
new_ position [ 0 ] ,current_ node . position [ 1 ] + new_ position [ 1 ] )
# Make sure within range
i f node_ position [ 0 ] > ( l e n ( maze ) − 1 ) or
node_ position [ 0 ] < 0 or node_ position [ 1 ] > ( l e n ( maze [ l e n ( maze )
−1]) −1) or
node_ position [ 1 ] < 0 :
continue
# Make sure walkable terrain
i f maze [ node_ position [ 0 ] ] [ node_ position [ 1 ] ] != 0 :
continue
# Create new node
new_node = Node ( current_node ,node_ position )
# Append
c h i l d r e n . append ( new_node )
# Loop through children
for child in children:
# Child i s on the closedlist
forclosed_childinclosed_list:
i f c h i l d == c l o s e d _ c h i l d :
continue
```

```
# Create the f ,g ,and h values
child. g = current_ node . g + 1
child. h = ( ( child.position[ 0 ] −
end_node . position[ 0 ] ) ** 2 ) +
( ( child.position[1]−end_node.position[1])** 2 )
c h i l d . f = c h i l d . g + c h i l d . h
# Child is already in the open l ist for open_node in open_ l i s t :
i f c h i l d == open_node and c h i l d . g >
open_node . g :
continue
# Add the child to the open l ist open_ list . append ( child)
def main ( ) :
maze = [ [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 ] ,
         [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ] ]
start = ( 0 ,0 ) end = ( 7 ,6 )
path = astar ( maze ,s t a r t ,end )
p r i n t ( path )
i f_name_== ‘_main_‘ :
main ( )
```

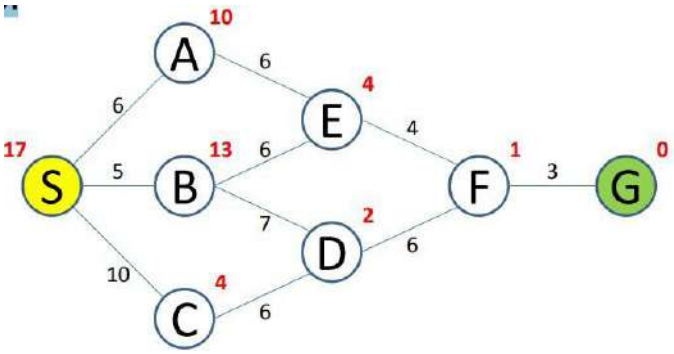
Output: [(0,0),(1,1),(2,2),(3,3),(4,3),(5,4),(6,5),(7,6)]

CONCLUSION

In A* Search,we can eliminate the repeated work by maintaining a closed set of expanded nodes and adding only those nodes to the priority queue which are not in the closed set,thus yielding an optimal graph search algorithm.

EXERCISE

1. Perform the A* Algorithm on the following figure. Explicitly print the queue at each step. Input Graph



LESSON - 3 WRITE A PROGRAM TO DEMONSTRATE LOGICAL AGENTS

CONTENTS

LOGICAL AGENTS

In Computer Science,Logic generally refers to formal languages that can represent information such that conclusions can be drawn from them. The idea of Logical Agents is that they can represent knowledge of its world,its goals and the current situation by sentences in logic and decide what to do by inferring an action or course of actions that help to achieve its goals.

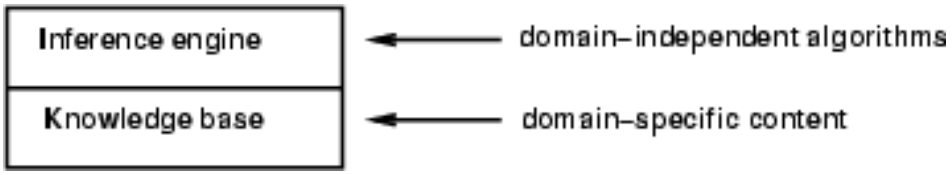


Figure 3: Basic components of a Logical Agent

Referring Figure 3 ,the Knowledge Base consists of the set of sentences in a formal language that represents the Truths of the world of the agent. The Inference Engine consists of the algorithms that can process and manipulate the information in the Knowledge Base,i.e.,it performs the Reasoning using the agents knowledge.

Basically,Knowledge is represented using either Propositional Logic or First Order Logic. In propositional logic,complex sentences are constructed from simpler sentences using logical connectives like negation (¬),conjunction (*) etc. First-Order

Logic is represented using natural language phrases,unlike propositional logic. It's basic representation uses Constants

(Dev,King Richard),Predicates(Brother,>..),Functions (sqrt,LeftLegOf,..) and so on.

PYTHON-SAT

This is a Python library providing a simple interface to a number of state-of-art Boolean satisfiability (SAT) solvers and a few types of cardinality and pseudo-Boolean encodings. The purpose of PySAT is to enable researchers working on SAT and its applications and generalizations to easily prototype with SAT oracles in Python while exploiting incrementally the power of the original low-level implementations of modern SAT solvers.

With PySAT it should be easy for you to implement a MaxSAT solver,an MUS/ MCS extractor/enumerator,or any tool solving an application problem with the (potentially multiple) use of a SAT oracle.

Reference: https://pypi.org/project/python-sat/

A LOGICAL AGENT IN PYTHON - TO EXIT A MINEFIELD

Here we have a program,which derives inference from a Minefield environment. The characteristics of the minefield is depicted in figure 4,where if the person enters the square with the mine,he will lose his life. The actions the person can take are: Move Left,Right,Up,Down.

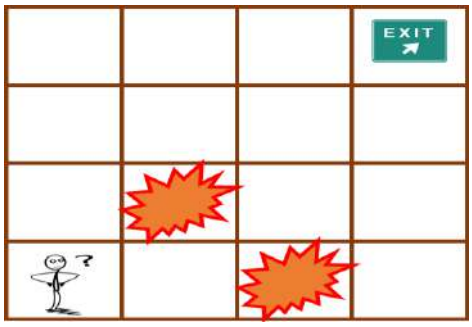


Figure 4: The Mine Field World in the program

Listing 7: Logical Agent in Python

```
# −*− coding: utf−8 −*−
class Agent:
def init (self): self._mineFieldWorld = [
                    [‘’,’’,’’,’’],# Rooms [1,4] to [4,4]
                    [‘’,’’,’’,’’],# Rooms [1,3] to [4,3]
                    [‘’,’M’,’’,’’],# Rooms [1,2] to [4,2]
                    [‘’,’’,’M’,’’],# Rooms [1,1] to [4,1]
                    ] # This is the mine field world
# A different instance of the mine field world will be used for evaluation. self.
```

```
curLoc = [1,1]
self._isAlive = True self._hasExited = False
def _FindIndicesForLocation(self,loc): x,y = loc
i,j = 4−y,x−1 return i,j
def _CheckForMine(self):
mf = self._mineFieldWorld
i,j = self._FindIndicesForLocation(self._curLoc) if 'M' in mf[i][j]:
print(mf[i][j])
self._isAlive = False print('Agent is DEAD.')
return self._isAlive
def TakeAction(self,action): # The function takes an action and returns whether
the Agent is alive
# after taking the action. validActions = ['Up','Down','Left','Right']
assert action in validActions,'Invalid Action.'
if self._isAlive == False:
print('Action cannot be performed. Agent is DEAD. Location:{0}'.format(self._
curLoc))
return False
if self._hasExited == True:
print('Action cannot be performed.  Agent has exited the Mine field
world.'.format(self._curLoc))
return False
index = validActions.index(action) validMoves = [[0,1],[0,−1],[−1,0],[1,0]]
move = validMoves[index]
newLoc = []
for v,inc in zip(self._curLoc,move):
z = v + inc #increment location index
z = 4 if z>4 else 1 if z<1 else z #Ensure that index is between 1 and 4 newLoc.
append(z)
self._curLoc = newLoc
print('Action Taken: {0},Current Location {1}'.format(action,self._curLoc)) if
self._curLoc[0]==4 and self._curLoc[1]==4:
self._hasExited=True return self._CheckForMine()
```

```
def _FindAdjacentRooms(self): cLoc = self._curLoc
validMoves = [[0,1],[0,−1],[−1,0],[1,0]]
adjRooms = []
for vM in validMoves: room = []
valid = True
for v,inc in zip(cLoc,vM): z = v + inc
if z<1 or z>4: valid = False break
else:
room.append(z) if valid==True:
adjRooms.append(room) return adjRooms
def PerceiveCurrentLocation(self): #This function perceives the current location.
#It detects whether mine is present in adjacent rooms.
percept = None
mf = self._mineFieldWorld if self._isAlive == False:
print('Agent cannot perceive.Agent is DEAD. Location:{0}'.format(self._
curLoc))
return None
if self._hasExited == True:
print('Agentcannotperceive.Agent hasexitedthe Mine field World.'.format(self._
curLoc))
return None
adjRooms = self._FindAdjacentRooms() count=0
for room in adjRooms:
i,j = self._FindIndicesForLocation(room) if 'M' in mf[i][j]:
count+=1
percept = '=0' if count==0 else '=1' if count==1 else '>1' return percept
def FindCurrentLocation(self):
return self._curLoc
def main():
        ag = Agent()
    print('curLoc',ag.FindCurrentLocation())
    print('Percept ',ag.PerceiveCurrentLocation()
```

```
        ag.TakeAction('Right')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Left')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Up')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Up')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Up')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Right')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Right')
        print('Percept',ag.PerceiveCurrentLocation())
        ag.TakeAction('Right')
        print('Percept',ag.PerceiveCurrentLocation())
if _name_=='_main'_: main()
```

Listing 8: mine_field.py

```
# −*− coding: utf−8 −*−
from pysat.solvers import Glucose3 from Agent.py import*
#[Left,right,Up,Down] where 65 is a dummy node ( always false )
neighbour={1:[65,2,5,65],2: [1,3,6,65],3: [2,4,7,65],4: [3,65,8,65],5:
[65,1,6,9],6: [5,
7,2,10],7: [6,8,3,11],8:[4,7,12,65],9: [65,5,10,13],10: [9,11,6,14],11: [10,
12,7,15],12: [8,11,16,65],13: [65,9,65,14],14: [13,15,65,10],15: [14,16,65,11],
16: [15,65,65,12]}
g = Glucose3()
def percept0(l,r,u,d,p0): clauses = []
g.add_clause([−1*d,−1*p0 ])
clauses.append([−1*d,−1*p0 ])
g.add_clause([1*d,1*l,1*p0,1*r,1*u])
clauses.append([1*d,1*l,1*p0,1*r,1*u])
```

```
g.add_clause([−1*l,−1*p0] )
clauses.append([−1*l,−1*p0] )
g.add_clause([−1*r,−1*p0] )
clauses.append([−1*r,−1*p0] )
g.add_clause([−1*u,−1*p0] )
clauses.append([−1*u,−1*p0] )
return clauses
def percept1(l,r,u,d,p1): clauses = []
append([−1*l,−1*p1,−1*r])
g.add_clause([−1*l,−1*p1,−1*u])
clauses.append([−1*l,−1*p1,−1*u])
g.add_clause([−1*u,−1*p1,−1*r])
clauses.append([−1*u,−1*p1,−1*r])
return clauses
def percept2(l,r,u,d,p2): clauses= []
g.add_clause([−1*d,−1*l,p2])
clauses.append([−1*d,−1*l,p2])
g.add_clause([−1*d,−1*r,p2])
clauses.append([−1*d,−1*r,p2])
g.add_clause([−1*d,−1*u,p2])
clauses.append([−1*d,−1*u,p2])
g.add_clause([d,l,−1*p2,r])
clauses.append([d,l,−1*p2,r])
g.add_clause([d,l,−1*p2,u])
clauses.append([d,l,−1*p2,u])
g.add_clause([d,−1*p2,r,u ])
clauses.append([d,−1*p2,r,u ])
g.add_clause([−1*l,p2,−1*r ])
clauses.append([−1*l,p2,−1*r ])
g.add_clause([−1*l,p2,−1*u])
clauses.append([−1*l,p2,−1*u])
g.add_clause([l,−1*p2,r,u ])
```

```
clauses.append([l,−1*p2,r,u ])
g.add_clause([p2,−1*r,−1*u])
clauses.append([p2,−1*r,−1*u])
return clauses
def Explore_rooms(ag,visited,goalLoc,dfsVisited,path): #dfs to new safe room
curr_pos=ag.FindCurrentLocation()
curLoc= 4*(curr_pos[1]−1)+curr_pos[0] #convert to location index
if(curLoc==goalLoc or curLoc == 16 ):
return True dfsVisited[curLoc]=True
if curr_pos[1]+1 <= 4:
if (visited[curLoc+4]==True ):
if dfsVisited[curLoc+4]==False:
 ag.TakeAction('Up')
cell = ag.FindCurrentLocation()
path.append(cell)
if Explore_rooms(ag,visited,goalLoc,dfsVisited,path): return True
path.remove(cell) ag.TakeAction('Down')
if curr_pos[0]+1 <= 4:
if (visited[curLoc+1]==True ):
if dfsVisited[curLoc+1]==False: ag.TakeAction('Right')
cell = ag.FindCurrentLocation()
path.append(cell)
if Explore_rooms(ag,visited,goalLoc,dfsVisited,path): return True
path.remove(cell) ag.TakeAction('Left')
if curr_pos[1]−1 >= 1:
if (visited[curLoc−4]==True ):
if dfsVisited[curLoc−4]==False: ag.TakeAction('Down')
cell = ag.FindCurrentLocation()
path.append(cell)
if Explore_rooms(ag,visited,goalLoc,dfsVisited,path): return True
path.remove(cell) ag.TakeAction('Up')
if curr_pos[0]−1 >= 1:
```

```
if (visited[curLoc−1]==True ):
if dfsVisited[curLoc−1]==False: ag.TakeAction('Left')
cell = ag.FindCurrentLocation()
path.append(cell)
if Explore_rooms(ag,visited,goalLoc,dfsVisited,path): return True
path.remove(cell) ag.TakeAction('Right')
return False
def ExitMineField(ag,clauses ):
clauses.append([−65]) # 65 represents dummy node which never has mine. clauses.append([−16]) # 16 represents the final state which can never have a mine. clauses.append([−1]) # start can never have a mine.
path = [] # will store the path
visited = [False for i in range(17)] #Rooms Visited till now visited[1] = True
while(ag.FindCurrentLocation()!=[4,4]):
percept= ag.PerceiveCurrentLocation() curr_pos = ag.FindCurrentLocation() curLocIndex= 4*(curr_pos[1]−1)+ curr_pos[0] visited[curLocIndex]=True
if percept==’=0’:
g.add_clause( [16+ curLocIndex] ) # 16−32 represents percept 0 clauses. append([16+ curLocIndex])
g.add_clause([ −1* (32+curLocIndex) ])
clauses.append([ −1* (32+curLocIndex) ])
g.add_clause([ −1* (48+curLocIndex) ])
clauses.append([ −1* (48+curLocIndex) ])
elif percept == '=1':
g.add_clause( [32+ curLocIndex] ) # 32−48 represents percept 1 clauses. append([32+ curLocIndex])
g.add_clause([ −1* (48+curLocIndex) ])
clauses.append([ −1* (48+curLocIndex) ])
g.add_clause([ −1* (16+curLocIndex) ])
clauses.append([ −1* (16+curLocIndex) ])
else:
g.add_clause( [48+ curLocIndex] ) # 48−64 represents percept 2 clauses. append([48+ curLocIndex])
```

```
g.add_clause([ −1* (16+curLocIndex) ])
clauses.append([ −1* (16+curLocIndex) ])
g.add_clause([ −1* (32+curLocIndex) ])
clauses.append([ −1* (32+curLocIndex) ])
for newLoc in [16,15,12,14,11,8,13,10,7,4,9,6,3,5,2,1]:
if visited[newLoc]==False: tempclauses= clauses tempclauses. append([newLoc]) g1 = Glucose3() g1.append_formula(tempclauses) tempclauses.remove([newLoc])
if g1.solve() == False:
#Room is safe clauses.append([−1*newLoc]) dfsVisited = [False for i in range(17)] visited[newLoc] = True
roomReachable=Explore_rooms(ag,visited,newLoc,dfsVisited,path)
#dfs to new safe Room if roomReachable:
break
return path
def initialise_knowledge():
# 1−16 is mine locations ( T/F )
# 17−32 is percept 0 ( T/F )
# 33−48 is percept 1 ( T/F )
# 49−64 is percept 2 ( T/F )
mega_clause = []
for i in range (1,17):
mega_clause.extend(percept0(neighbour[i][0],neighbour[i][1],neighbour[i][2],
neighbour[i][3],i+16 ))
mega_clause.extend(percept1(neighbour[i][0],neighbour[i][1],neighbour[i][2],
neighbour[i][3],i+32 ))
mega_clause.extend(percept2(neighbour[i][0],neighbour[i][1],neighbour[i][2],
neighbour[i][3],i+48 ))
return mega_clause
def main():
cl = initialise_knowledge() ag = Agent()
print('Start Location: {0}'.format(ag.FindCurrentLocation())) path =
ExitMineField(ag,cl)
```

```
print('{0} reached. Exiting the Mine Field.'.format(ag.FindCurrentLocation()))
main()
```

Output:

curLoc [1,1]

Percept =0

Action Taken: Right,Current Location [2,1] Percept >1

Action Taken: Left,Current Location [1,1] Percept =0

Action Taken: Up,Current Location [1,2] Percept =1

Action Taken: Up,Current Location [1,3] Percept =0

Action Taken: Up,Current Location [1,4] Percept =0

Action Taken: Right,Current Location [2,4] Percept =0

Action Taken: Right,Current Location [3,4] Percept =0

Action Taken: Right,Current Location [4,4]

Agent cannot perceive. Agent has exited the Mine field World. Percept None

CONCLUSION

Here we have demonstrated a logical agent using python,that perceives a Mine Field environment,and performs actions in order to exit the Mine field alive. The PySAT package was used for implementation.

EXERCISE

1. Write a program to demonstrate the game of Tic Tac Toe.

LESSON - 4

EVALUATE FORWARD CHAINER AND RULE BASE ON AT LEAST FOUR DIFFERENT DATABASES. TRY TO CREATE AT LEAST ONE DATABASE THAT DEMONSTRATES AN INTERESTING FEATURE OF THE DOMAIN, OR AN INTERESTING FEATURE OF FORWARD CHAINING

CONTENTS

FORWARD CHAINING

Forward Chaining is a part of the Inference Engine of a Logical Agent in Artificial Intelligence. It is a reasoning process,that starts with atomic sentences/clauses in the Knowledge Base(KB),and applies Modus Ponens rules in the forward direction,until a goal is reached [3].

HORN CLAUSE AND DEFINITE CLAUSE

These are sentences in the KB. To be able to apply Forward chaining or Backward chaining on the sentences,they should be in Definite Clause Format.

•Horn Clause

– A clause which is a disjunction of literals with at most one positive literal is known as horn clause. All the definite clauses are horn clauses.

•Definite Clause

– A clause that is a disjunction of literals with exactly one positive literal.

– ¬a v ¬b v.....v ¬y v z,where only z is a positive literal and the others are negative literals

– A definite clause is logically equivalent to the expression a^b^. ^y => z

To solve a problem using forward Chaining,when we get the problem description,convert all the facts available in the problem into First Order Logic (FOL) definite clauses. The inference is done using the following operations

1. Convert the sentences and facts to First Order Logic
2. Convert every sentence to Conjunctive Normal Form
   - Eliminate Implications
   - Move the negation ¬ inwards for all quantifiers
3. Standardize variables apart,i.e,rename variables if there are same variables
4. Remove Quantifiers if any
5. Distribute v over ^
6. Apply Forward Chaining,by starting with the known true facts.
   a. Construct the proof,by matching with the remaining clauses,and Substi- tute for variables.
   b. Repeat Matching,Substitution and Elimination of Complementary predic ates,until the goal is achieved.

FORWARD CHAINING PROGRAM IN PYTHON

Input Files

In this program,the Query and the KB sentences are included in a single inputfile. The format of the input file is as follows:

1. Number of queries
2. The Queries to be asked
3. Number of Sentences in KB
4. The Sentences to represent the Scenario/Problem

Listing 9: Forward Chaining Resolution in Python

```
# −*− coding: utf−8 −*−
import time
start_time = time.time()
import re
import itertools import
collectionsimport copy
import queue
p=open(“Test cases/input22.txt”,”r”)
data=list()
data1= p.readlines()
```

```
count=0
n=int(data1[0])
queries=list()
for i in range(1,n+1):
queries.append(data1[i].rstrip())
k=int(data1[n+1])
kbbefore=list()
def CNF(sentence):
temp=re.split("=>",sentence) temp1=temp[0].split('&')
for i in range(0,len(temp1)): if temp1[i][0]=='~':
temp1[i]=temp1[i][1:]
else:
temp1[i]='~'+temp1[i]
temp2='|'.join(temp1)
temp2=temp2+'|'+temp[1]
return temp2
variableArray = list("abcdefghijklmnopqrstuvwxyz")
variableArray2 = []
variableArray3 = []
variableArray5 = []
variableArray6 = []
for eachCombination in itertools.permutations(variableArray,2):
variableArray2.append(eachCombination[0] + eachCombination[1])
for eachCombination in itertools.permutations(variableArray,3):
variableArray3.append(eachCombination[0] + eachCombination[1] +
eachCombination[2])
for eachCombination in itertools.permutations(variableArray,4):
variableArray5.append(eachCombination[0] + eachCombination[1] +
eachCombination[2]+ eachCombination[3])
for eachCombination in itertools.permutations(variableArray,5):
variableArray6.append(eachCombination[0] + eachCombination[1] +
eachCombination[2] + eachCombination[3] + eachCombination[4])
```

```
variableArray=variableArray+variableArray2 +variableArray3 +variableArray5
+ variableArray6
capitalVariables = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”
number=0
def standardizationnew(sentence):
newsentence=list(sentence)
i=0
global number
variables=collections.OrderedDict()
positionsofvariable=collections.OrderedDict()
lengthofsentence=len(sentence)
for i in range(0,lengthofsentence−1):
if(newsentence[i]==’,’ or newsentence[i]==’(‘):
if newsentence[i+1] not in capitalVariables:
substitution=variables.get(newsentence[i+1])
positionsofvariable[i+1]=i+1
if not substitution :
variable[newsentence[i+1]]=variableArray[number]
newsentence[i+1]=variableArray[number]
number+=1
else:
newsentence[i+1]=substitution
return “”.join(newsentence)
def insidestandardizationnew(sentence):
lengthofsentence=len(sentence)
newsentence=sentence
variables=collections.OrderedDict()
positionsofvariable=collections.OrderedDict()
global number
i=0
while i <=len(newsentence)−1 :
if(newsentence[i]==’,’ or newsentence[i]==’(‘):
```

```
if newsentence[i+1] not in capitalVariables:
j=i+1
while(newsentence[j]!=',' and newsentence[j]!=')' ):
j+=1
substitution=variables.get(newsentence[i+1:j]) if not substitution :
variables[newsentence[i+1:j]]=variableArray[number]
newsentence=newsentence[:i+1]+variableArray[number]+newsentence[j:]
i=i+len(variableArray[number])
number+=1
i+=1
else:
newsentence=newsentence[:i+1]+substitution+newsentence[j:]
i=i+len(substitution)
return newsentence
def replace(sentence,theta):
lengthofsentence=len(sentence)
newsentence=sentence i=0
while i <=len(newsentence)−1 :
if(newsentence[i]==',' or newsentence[i]=='(‘):
if newsentence[i+1] not in capitalVariables:
j=i+1
while(newsentence[j]!=',' and newsentence[j]!=')' ):
j+=1
nstemp=newsentence[i+1:j] substitution=theta.get(nstemp) if substitution :
newsentence=newsentence[:i+1]+substitution+newsentence[j:]
i=i+len(substitution)
i+=1
return newsentence repeatedsentencecheck=collections.OrderedDict()
def insidekbcheck(sentence): lengthofsentence=len(sentence)
newsentence=pattern.split(sentence) newsentence.sort() newsentence="|".
join(newsentence) global repeatedsentencecheck
i=0
while i <=len(newsentence)−1 :
```

```
if(newsentence[i]==',' or newsentence[i]=='('):
if newsentence[i+1] not in capitalVariables:
j=i+1
while(newsentence[j]!=',' and newsentence[j]!=')' ): j+=1
newsentence=newsentence[:i+1]+'x'+newsentence[j:]
i+=1
repeatflag=repeatedsentencecheck.get(newsentence) if repeatflag :
return True repeatedsentencecheck[newsentence]=1
return False
for i in range(n+2,n+2+k):
data1[i]=data1[i].replace(" ","") if "=>" in data1[i]:
data1[i]=data1[i].replace(" ","")
sentencetemp=CNF(data1[i].rstrip())
kbbefore.append(sentencetemp) else:
kbbefore.append(data1[i].rstrip())
for i in range(0,k):
kbbefore[i]=kbbefore[i].replace(" ","")
kb={}
pattern=re.compile("\||&|=>") #we can remove the '\|'' to speed up as 'OR' doesnt come in the KB
pattern1=re.compile("[(,]") for i in range(0,k):
kbbefore[i]=standardizationnew(kbbefore[i])
temp=pattern.split(kbbefore[i])
lenoftemp=len(temp)
for j in range(0,lenoftemp):
clause=temp[j]
clause=clause[:−1]
predicate=pattern1.split(clause)
argumentlist=predicate[1:]
lengthofpredicate=len(predicate)−1 if predicate[0] in kb:
if lengthofpredicate in kb[predicate[0]]:
kb[predicate[0]][lengthofpredicate].append([kbbefore[i],temp,j,predicate[1:]])
```

```
else:
kb[predicate[0]][lengthofpredicate]=[kbbefore[i],temp,j,predicate[1:]]
else:
kb[predicate[0]]={lengthofpredicate:[[kbbefore[i],temp,j,predicate[1:]]]}
for qi in range(0,n): queries[qi]=standardizationnew(queries[qi])
def substituevalue(paramArray,x,y):
for index,eachVal in enumerate(paramArray): if eachVal == x:
paramArray[index] = y return paramArray
def unificiation(arglist1,arglist2):
theta = collections.OrderedDict()
for i in range(len(arglist1)):
if arglist1[i] != arglist2[i] and (arglist1[i][0] in capitalVariables) and (arglist2[i]
[0] in capitalVariables):
return []
elif arglist1[i] == arglist2[i] and (arglist1[i][0] in capitalVariables) and (arglist2[i]
[0] in capitalVariables):
if arglist1[i] not in theta.keys():
theta[arglist1[i]] = arglist2[i]
elif (arglist1[i][0] in capitalVariables) and not (arglist2[i][0] in capitalVariables):
if arglist2[i] not in theta.keys():
theta[arglist2[i]] = arglist1[i]
arglist2 = substituevalue(arglist2,arglist2[i],arglist1[i])
elif not (arglist1[i][0] in capitalVariables) and (arglist2[i][0] in
capitalVariables):
if arglist1[i] not in theta.keys():
theta[arglist1[i]] = arglist2[i]
arglist1 = substituevalue(arglist1,arglist1[i],arglist2[i])
elif not (arglist1[i][0] in capitalVariables) and not (arglist2[i][0] in
capitalVariables):
if arglist1[i] not in theta.keys():
theta[arglist1[i]] = arglist2[i]
arglist1 = substituevalue(arglist1,arglist1[i],arglist2[i])
```

```
else:
argval=theta[arglist1[i]] theta[arglist2[i]]=argval
arglist2 = substituevalue(arglist2,arglist2[i],argval)
return [arglist1,arglist2,theta]
def resolution():
global repeatedsentencecheck answer=list()
qrno=0
for qr in queries:
qrno+=1
repeatedsentencecheck.clear()
q=queue.Queue()
query_start=time.time()
kbquery=copy.deepcopy(kb)
ans=qr
if qr[0]==’~’:
ans=qr[1:] else:
ans=’~’+qr q.put(ans) label:
outerloop currentanswer=”FALSE” counter=0
while True:
counter+=1 if q.empty(): break ans=q.get()
label:outerloop1
ansclauses=pattern.split(ans)
lenansclauses=len(ansclauses)
flagmatchedwithkb=0
innermostflag=0
for ac in range(0,lenansclauses):
insidekbflag=0
ansclausestruncated=ansclauses[ac][:−1]
ansclausespredicate=pattern1.split(ansclausestruncated)
lenansclausespredicate=len(ansclausespredicate)−1
if ansclausespredicate[0][0]==’~’:
anspredicatenegated=ansclausespredicate[0][1:]
```

```
else:
anspredicatenegated="~"+ansclausespredicate[0]
x=kbquery.get(anspredicatenegated,{}).get(lenansclausespredicate) if not x:
continue else:
lenofx=len(x)
for numofpred in range(0,lenofx):
insidekbflag=0
putinsideq=0 sentenceselected=x[numofpred]
thetalist=unificiation(copy.deepcopy(sentenceselected[3]),copy.
deepcopy(ansclause if(len(thetalist)!=0):
for key in thetalist[2]:
tl=thetalist[2][key] tl2=thetalist[2].get(tl) if tl2:
thetalist[2][key]=tl2
flagmatchedwithkb=1
notincludedindex=sentenceselected[2]
senclause=copy.deepcopy(sentenceselected[1]) mergepart1=""
del senclause[notincludedindex]
ansclauseleft=copy.deepcopy(ansclauses) del ansclauseleft[ac]
for am in range(0,len(senclause)):
senclause[am]=replace(senclause[am],thetalist[2])
mergepart1=mergepart1+senclause[am]+'|'
for remain in range(0,len(ansclauseleft)):
listansclauseleft=ansclauseleft[remain]
ansclauseleft[remain]=replace(listansclauseleft,thetalist[2]) if
ansclauseleft[remain] not in senclause:
mergepart1=mergepart1+ansclauseleft[remain]+'|'mergepart1=mergepart1[:-1]
if mergepart1=="":
currentanswer="TRUE" break
ckbflag=insidekbcheck(mergepart1) if not ckbflag:
mergepart1=insidestandardizationnew(mergepart1)
ans=mergepart1
temp=pattern.split(ans) lenoftemp=len(temp)
for j in range(0,lenoftemp):
```

```
clause=temp[j]
clause=clause[:−1]
predicate=pattern1.split(clause)
argumentlist=predicate[1:]
lengthofpredicate=len(predicate)−1
if predicate[0] in kbquery:
if lengthofpredicate in kbquery[predicate[0]]:
kbquery[predicate[0]][lengthofpredicate].
append([mergepart1,temp,j,argumentlist])
else:
kbquery[predicate[0]][lengthofpredicate]= [
[mergepart1,temp,j,argumentlist]]
else:
kbquery[predicate[0]]={lengthofpredicate:[
                [mergepart1,temp,j,argumentlist]]}
q.put(ans)
if(currentanswer==”TRUE”): break
if(currentanswer==”TRUE”): break
if(counter==2000 or (time.time()−query_start)>20): break
answer.append(currentanswer) return answer
if _name_== ‘_main_‘:
finalanswer=resolution() o=open(“output.txt”,”w+”) wc=0
while(wc < n−1): o.write(finalanswer[wc]+”\n”) print(finalanswer[wc])
wc+=1 o.write(finalanswer[wc]) o.close()
```

Outputs:

Input Content 1

```
1
Criminal(West) 8
American(x)& Weapon(y)& Sells(x,y,z)& Hostile(z) =>Criminal(x)
Owns(Nono,M1) Missile(M1)
Missile(x)& Owns(Nono,x) => Sells(West,x,Nono)
Missile(x) => Weapon(x) Enemy(x,America) =>Hostile(x)
```

American(West) Enemy(Nono,America)

Output 1

TRUE

Input Content 2

2

Ancestor(Liz,Billy) Ancestor(Liz,Bob) 6

Mother(Liz,Charley) Father(Charley,Billy) Mother(x,y) | Parent(x,y) Father(x,y) |Parent(x,y) Parent(x,y) | Ancestor(x,y)

Parent(x,y) | Ancestor(y,z) | Ancestor(x,z)

Output 2

TRUE FALSE

Input Content 3

1

Eats(jo,fish) 3

Cat(x) =>Likes(x,fish)

Cat(y) & Likes(y,z) =>Eats(y,z)

Cat(jo)

Output 3 TRUE

Input Content 4

2

Ancestor(Liz,Billy) Ancestor(Liz,Bob) 6

Mother(Liz,Charley) Father(Charley,Billy) Mother(x,y) | Parent(x,y) Father(x,y) | Parent(x,y) Parent(x,y) | Ancestor(x,y)

Parent(x,y) Ancestor(y,z) => Ancestor(x,z)

Output 4 TRUE FALSE

CONCLUSION

A Logical Agent has been implemented and demonstrated to use the Forward Chaining Algorithm for making inferences,on 4 different cases.

EXERCISE

The following is the rule set of a simple weather forecast expert system:

1. IF cyclone THEN clouds
2. IF anticyclone THEN clear sky
3. IF pressure is low THEN cyclone

4. IF pressure is high THEN anticyclone
5. IF arrow is down THEN pressure is low
6. IF arrow is up THEN pressure is high

Use forward chaining to reason about the weather if the working memory contains the fact: arrow is down. Show your answer in a table naming the rules matching the current working memory (conflict set),which rule you apply,and how the working memory contents changes on the next cycle after a rule has fired: Cycle | Working Memory | Conflict set | Rule fired ..

LESSON - 5

DEMONSTRATE AGENT BASED ON PROPOSITIONAL LOGIC

CONTENTS

PROPOSITIONAL LOGIC (PL)

This is the simplest way of representing information in the Knowledge Base. It consists of atomic sentences,that use single propositional symbols,represented by letters. Complex sentences are constructed from the atomic symbols of literals using logical connectives,as discussed in section 3.1.Although propositional logic is simple,it lacks expressive power,due to which complex semantics cannot be represented using PL.

RESOLUTION IN PROPOSITIONAL LOGIC

1. Apply Modus Ponens on a Rule
   - **Modus Ponens**

   P =>Q,P

   □ Q (1)
2. Apply And Elimination : From a Conjunction,any of the conjuncts can be inferred
   - **And-Elimination**

   OR

   A ^ B/ A (2)

   A ^ B /B (3)
3. Apply Unit Resolution: Where complementary literals are eliminated.
   - **Unit Resolution**

$$\frac{l_1 \vee l_2, \neg l_2 \vee l_3}{l_1 \vee l_3} \quad (4)$$

4. Repeat steps 1 to 3 until no more literals need to be resolved or you arrive at an empty clause

CONJUNCTIVE NORMAL FORM (CNF)

A sentence expressed as a conjunction of disjunctions of literals is said to be in conjunctive normal form or CNF. The resolution rule applies only to disjunctions of literals. Every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals.

To convert a sentence to CNF,the following rules need to be applied:

1. Eliminate Biconditionals in a sentence $(\Leftrightarrow)$
2. Eliminate Implications $(\Rightarrow)$
3. Apply De Morgans Rule to the result.

$\neg (P \vee Q) \iff (\neg P) \wedge (\neg Q)$ and $\neg (P \wedge Q) \iff (\neg P) \vee (\neg Q)$

4. Apply distributive law and flatten the sentence

The text-book example for propositional logic based Logical Agent is the Wumpus world. Here we can refer to the figure 5 to model our Knowledge Base:

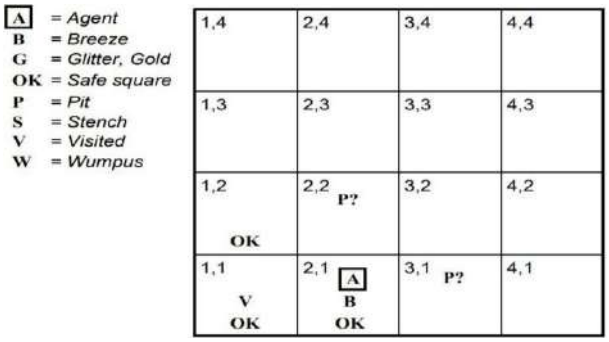


Figure 5: Wumpus World Reference from AI-A Modern Approach

PROPOSITIONAL LOGIC IN PYTHON

To implement this in Python,we have used the library files from AIMA (Artificial Intelligence - A Modern approach) Python code [4] . We have also used the example from the same book [5]. To implement this program,copy the file named 'logic' into your system,and create your file inside this folder. Install required packages like the following:

jupyter

pandas matplotlib pillow Image ipython

ipythonblocks

Listing 10: Python Code to demonstrate Propositional Logic Agent

```
# −*− coding: utf−8 −*−
from utils import *
from logic import *
from notebook import psource
#Create a propositional KB for Wumpus World wumpus_kb = PropKB()
#Create the required symbols to represent the world
P11,P12,P21,P22,P31,B11,B21 = expr('P11,P12,P21,P22,P31,B11,B21')
#TELL the KB the specifications in the Wumpus World description
#No pit in P(1,1) wumpus_kb.tell(~P11)
#A square is breezy if and only if there is a pit in a neighboring square wumpus_
kb.tell(B11 | '<=>' | ((P12 | P21)))
wumpus_kb.tell(B21 | '<=>' | ((P11 | P22 | P31)))
#Include the breeze percepts for the first two squares wumpus_kb.tell(~B11)
wumpus_kb.tell(B21)
#Check the clauses stored in a KB by accessing its clauses variable print("Wumpus
World KB:")
print( wumpus_kb.clauses)
#Given a percept,a KB Agent adds the percept to its knowledge base,asks the
knowledge base for the best action and
#tells the KB that it has taken the action
def KB_AgentProgram(KB):
steps=itertools.count()
def program(percept):
t= next(steps) KB.tell(make_percept_sentence(percept,t))
action = KB.ask(make_action_query(t))
KB.tell(make_action_sentence(action,t))
retmake_action_query
def make_percept_sentence(percept,t):
return Expr("Percept")(percept,t)
def make_action_query(t):
```

```
return expr("ShouldDo(action,{}").format(t)
def make_action_sentence(action,t):
return Expr("Did")(action[expr('action')],t)
 return program
#Checking whether KB=>A
#Inference in Propositional Logic
def tt_check_all(kb,alpha,symbols,model):
"""Auxiliary routine to implement tt_entails."""" if not symbols:
if pl_true(kb,model):
result = pl_true(alpha,model)
assert result in (True,False)
return result
else:
return True
else:
P,rest = symbols[0],symbols[1:]
return (tt_check_all(kb,alpha,rest,extend(model,P,True)) and tt_check_
all(kb,alpha,rest,extend(model,P,False)))
def tt_entails(kb,alpha):
"""Does kb entail the sentence alpha? Use truth tables. For propositional kb's
and sentences. Note that the 'kb' should be an
Expr which is a conjunction of clauses.
>>> tt_entails(expr('P & Q'),expr('Q'))
True """
assert not variables(alpha)
symbols = list(prop_symbols(kb & alpha)) return tt_check_
all(kb,alpha,symbols,{})
print("Test_Query_Result for P&Q|Q:")
print(tt_entails(P & Q,Q))
print("IS ~P11 true?")
print(wumpus_kb.ask_if_true(~P11))
print("IS P11 true?")
```

print(wumpus_kb.ask_if_true(P11))

print("IS ~P22 true?")

print(wumpus_kb.ask_if_true(~P22))

print("IS P22 true?")

print(wumpus_kb.ask_if_true(P22))

#Proof by Resolution

#Conversion to Conjunctive Normal form def to_cnf(s):

"""Convert a propositional logical sentence to conjunctive normal form. That is,to the form ((A | ~B | ...) & (B | C | ...) & ...) [p. 253]

>>> to_cnf('~(B | C)') (~B & ~C)

"""

s = expr(s)

if isinstance(s,str): s = expr(s)

s = eliminate_implications(s) # Steps 1,2 s = move_not_inwards(s) # Step 3

return distribute_and_over_or(s) # Step 4

def eliminate_implications(s):

"""Change implications into equivalent form with only &,|,and ~ as logical operators."""

s = expr(s)

if not s.args or is_symbol(s.op):

return s # Atoms are unchanged.

args = list(map(eliminate_implications,s.args))

a,b = args[0],args[−1]

if s.op == '==>':

return b | ~a elif s.op == '<==':

return a | ~b elif s.op == '<=>':

return (a | ~b) & (b | ~a) elif s.op == '^':

assert len(args) == 2 # TODO:

relax this restriction

return (a & ~b) | (~a & b) else:

assert s.op in ('&','|','~')

return Expr(s.op,*args) def move_not_inwards(s):

```
“””Rewrite sentence s by moving negation sign inward.
>>> move_not_inwards(~(A | B)) (~A & ~B)”””
s = expr(s)
if s.op == ‘~’: def NOT(b):
return move_not_inwards(~b) a = s.args[0]
if a.op == ‘~’:
return move_not_inwards(a.args[0]) # ~~A ==> A if a.op == ‘&’:
return associate(‘|’,list(map(NOT,a.args))) if a.op == ‘|’:
return associate(‘&’,list(map(NOT,a.args))) return s
elif is_symbol(s.op) or not s.args: return s
else:
return Expr(s.op,*list(map(move_not_inwards,s.args)))
def distribute_and_over_or(s):
“””Given a sentence s consisting of conjunctions and disjunctions of literals,return
an equivalent sentence in CNF.
>>> distribute_and_over_or((A & B) | C) ((A | C) & (B | C))
“””
s = expr(s)
if s.op == ‘|’:
s = associate(‘|’,s.args) if s.op != ‘|’:
return distribute_and_over_or(s) if len(s.args) == 0:
return False
if len(s.args) == 1:
return distribute_and_over_or(s.args[0])
conj = first(arg for arg in s.args if arg.op == ‘&’) if not conj:
return s
others = [a for a in s.args if a is not conj]
rest = associate(‘|’,others)
return associate(‘&’,[distribute_and_over_or(c | rest)
for c in conj.args])
elif s.op == ‘&’:
return associate(‘&’,list(map(distribute_and_over_or,s.args))) else:
```

return s

#Test some queries to check the working print("Testing CNF conversion for some sentences:") A,B,C,D = expr('A,B,C,D')

print("Convert A <=> B to CNF")

print(to_cnf(A |'<=>'| B))

print("Convert A <=> (B & C) to CNF")

print(to_cnf(A |'<=>'| (B & C)))

#Resolution of Problems

def pl_resolution(KB,alpha):

"""Propositional−logic resolution:

say if alpha follows from KB. [Figure 7.12]""" clauses = KB.clauses + conjuncts(to_cnf(~alpha))

new = set() while True:

n = len(clauses)

pairs = [(clauses[i],clauses[j])

for i in range(n) for j in range(i+1,n)] for (ci,cj) in pairs:

resolvents = pl_resolve(ci,cj) if False in resolvents:

return True

new = new.union(set(resolvents)) if new.issubset(set(clauses)):

return False for c in new:

if c not in clauses: clauses.append(c)

#Check a few Wumpus World queries for Resolution

print("Query: Given the Wumpus World Knowledge base,there is no pit in P11") print(pl_resolution(wumpus_kb,~P11))

print("Query: Given the Wumpus World Knowledge base,there is a pit in P11") print(pl_resolution(wumpus_kb,P11))

print("Query: Given the Wumpus World Knowledge base,there is no pit in P22") print(pl_resolution(wumpus_kb,~P22))

print("Query: Given the Wumpus World Knowledge base,there is a pit in P22") print(pl_resolution(wumpus_kb,P22))

Output

Wumpus World KB:

[¬P 11,(¬P 12|B11),(¬P 21|B11),(P 12|P 21|¬B11),(¬P 11|B21),(¬P 22|B21),(¬P 31|B21),(P 11|P 22|P 31|¬B21),¬B11,B21]

Test_Query_Result for P &Q|Q :

True

IS ¬P 11 true?

True

IS P11 true? False

IS ¬P 22 true?

False

IS P22 true? False

Testing CNF conversion for some sentences:

Convert A|′ ⟺ ′|B to CNF

((A|¬B)&(B|¬A))

Convert A|′ ⟺ ′|(B&C) to CNF

((A|¬B|¬C)&(B|¬A)&(C|¬A))

Query: Given the Wumpus World Knowledge base,there is no pit in P11 True

Query: Given the Wumpus World Knowledge base,there is a pit in P11 False

Query: Given the Wumpus World Knowledge base,there is no pit in P22 False

Query: Given the Wumpus World Knowledge base,there is a pit in P22 False

CONCLUSION

A Propositional Logic based Agent was implemented and demonstrated successfully in Python using the AIMA-Python Libraries.

EXERCISE

For more exercises follow the link given: https://aimacode.github.io/aima-exercises/knowledge- logic-exercises/

LESSON - 6

WRITE A PROGRAM TO IMPLEMENT THE NAIVE BAYESIAN CLASSIFI FOR A SAMPLE TRAINING DATASET COMPUTE THE ACCURACY OF THE CLASSIFI CONSIDERING FEW DATA- SETS.

CONTENTS

NAIVE BAYES CLASSIFIER

Naïve Bayes Classifier is a classification algorithm used on Binary (two-class) and Multi-class classification Problems. It is a simple and powerful algorithm for pre- dictive modeling. Here we assume that the values of each attribute are conditionally independent.

BAYES THEOREM

Bayes theorem is about finding probability of a hypothesis “h” given Prior Knowledge

or data “d”.

$$P(\underline{\underline{h}}) = \frac{P(\underline{\underline{d}}) * P_{\overline{h}}(h)}{P(d)} \quad (5)$$

where

- Select the hypothesis with the highest probability,called MAP or Maximum A Posteriori

$$MAP(h) = max\lceil P(\frac{H}{d})\rceil \quad (6)$$

Assumptions Naïve Bayes Classifier assumes that all features are unrelated or independent of eachother.

Types of Naïve Bayes Algorithms There are 3 types of Naïve Bayes Algorithms:

Gaussian Naïve Bayes Used when attribute values are continuous. Assumption is made that values associated with each class are distributed according to Gaussian or Normal distribution.

Multinomial Naïve Bayes Used on data that is multinomially distributed. Here the feature vectors respresent the frequencies with which events have been generated by a multinomial. Useful in text categorization problems.

Bernoulli Naïve Bayes Used where classification is based on binary values. Useful in document classification where Term occurrence is specified rather than term frequencies.

GAUSSIAN NAÏVE BAYES CLASSIFIER

If x is a continuous attribute variable

1. Segment the data by class
2. Compute mean μi and variance σi of x in each class i
3. For an observation value xi the probability distribution of xi given a class i is computed by equation:

$$P(x_i|y_i) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{\frac{-(x_i-\mu_j)^2}{2\sigma_j^2}} \quad (7)$$

NAÏVE BAYES CLASSIFIER STEPS

Here we implement the Naive Bayes Classifier using Gaussian method and the sklearn package from Python.

1. Preprocess the data
2. Fit the Naive Bayes classifier to the Training set
3. Predict the test result
4. Test the accuracy of the result

DATASET DESCRIPTION

The dataset contains the list of people who may or may not have diabetes. So the main idea of this project is to predict whether a person is susceptible to diabetes or not. For that the attributes the dataset contains are:

- Pregnancies
- Glucose
- Blood Pressure
- Skin Thickness
- Insulin
- BMI
- Diabetes Pedigree function
- Age

Listing 11: Naive Bayes Classifier Program

```
# −*− coding: utf−8 −*−
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('diabetes.csv')
x = dataset.iloc[:,[0,7]].values
y = dataset.iloc[:,8].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
 x_test = sc.transform(x_test)
 # Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
```

```
classifier = GaussianNB()
classifier.fit(x_train,y_train)
# Predicting the Test set results
y_pred = classifier.predict(x_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print("Confusion Matrix: \n",cm)
# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set,y_set = x_train,y_train
X1,X2 = nm.meshgrid(nm.arange(start = x_set[:,0].min() − 1,stop = x_set[:,0].
max() + 1,step = 0.01),
nm.arange(start = x_set[:,1].min() − 1,stop = x_set[:,1].max()
+ 1,step = 0.01))
mtp.contourf(X1,X2,classifier.predict(nm.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),
alpha = 0.75,cmap = ListedColormap(('purple','green'))) mtp.xlim(X1.min(),X1.
max())
mtp.ylim(X2.min(),X2.max())
for i,j in enumerate(nm.unique(y_set)): mtp.scatter(x_set[y_set == j,0],x_set[y_set
== j,1],
color = ListedColormap(('purple','green'))(i),label = j) mtp.title('Naive Bayes
(Training set)')
mtp.xlabel('Age') mtp.ylabel('Diabetic') mtp.legend()
mtp.show()
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set,y_set = x_test,y_test
X1,X2 = nm.meshgrid(nm.arange(start = x_set[:,0].min() − 1,stop = x_set[:,0].
max() + 1,step = 0.01),
nm.arange(start = x_set[:,1].min() − 1,stop = x_set[:,1].max()
+ 1,step = 0.01))
```

mtp.contourf(X1,X2,classifier.predict(nm.array([X1.ravel(),X2.ravel()]).T). reshape(X1.shape),

alpha = 0.75,cmap = ListedColormap(('purple','green'))) mtp.xlim(X1.min(),X1. max())

mtp.ylim(X2.min(),X2.max())

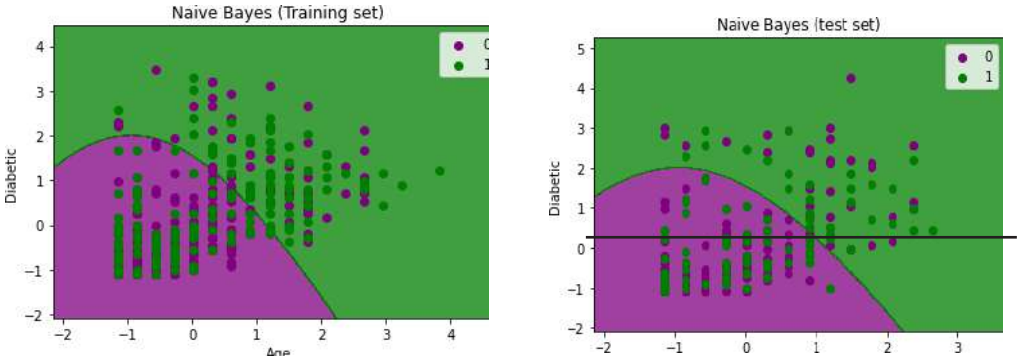
for i,j in enumerate(nm.unique(y_set)): mtp.scatter(x_set[y_set == j,0],x_set[y_set == j,1],

color = ListedColormap(('purple','green'))(i),label = j) mtp.title('Naive Bayes (test set)')

mtp.xlabel('Age') mtp.ylabel('Diabetic') mtp.legend() mtp.show()

Output:

Confusion Matrix: 134 33
59 28



CONCLUSION

The Naive Bayes Classifier was trained on the Diabetes Dataset,and the classification of population was done based on their details and the prediction was whether they are non diabetic or diabetic.

EXERCISE

Apply Naive Bayes Technique on any datasets like Iris,Pima etc.,.

LESSON - 7

WRITE A PROGRAM TO CONSTRUCT A BAYESIAN NET- WORK CONSIDERING MEDICAL DATA. USE THIS MODEL TO DEMONSTRATE THE DIAGNOSIS OF HEART PATIENTS USING STANDARD HEART DISEASE DATASET.

CONTENTS

BAYESIAN NETWORK

Bayesian Networks are graphical models that are paired with their conditional probabilities,for each node. Each node in a Bayesian network corresponds to a random variable,and each edge represents the conditional probability for the corresponding random variables. It is similar to a Directed Acyclic Graph(DAG) since it represents dependencies and conditional probabilities. It is called the Bayesian Network since it measures the conditional probability using Bayes theorem,shown in equation 8.

$$P(A|B) = \frac{P(A|B)P(A)}{P(B)} \quad (8)$$

DATA SET DESCRIPTION

The dataset we use here has attributes pertaining to heart disease,as explained below.

Attributes:

- age: age in years

- sex: sex (1 = male; 0 = female)
- cp: chest pain type
  - Value 1: typical angina
  - Value 2: atypical angina
  - Value 3: non-anginal pain
  - Value 4: asymptomatic
- trestbps: resting blood pressure (in mm Hg on admission to the hospital)
- chol: serum cholestoral in mg/dl
- fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg: resting electrocardiographic results
  - Value 0: normal
  - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
  - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
- thalach: maximum heart rate achieved
- exang: exercise induced angina (1 = yes; 0 = no)
- oldpeak = ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment
  - Value 1: upsloping
  - Value 2: flat
  - Value 3: downsloping
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
- Heartdisease: It is integer valued from 0 (no presence) to 4.

Download the dataset from the link given here: Heart Disease Diagnosis

BAYESIAN NETWORK IN PYTHON

Listing 12: Bayesian Network implementation in Python

```
# −*− coding: utf−8 −*−
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
```

```
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
model= BayesianNetwork([('age','heartdisease'),('gender','heartdisease'),
('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg') q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ') q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

```
age gender cp trestbps chol ... oldpeak slope ca thal heartdisease
0 63 1 1 145 233 ... 2.3 3 0 6 0
1 67 1 4 160 286 ... 1.5 2 3 3 2
2 67 1 4 120 229 ... 2.6 2 2 7 1
3 37 1 3 130 250 ... 3.5 3 0 3 0
4 41 0 2 130 204 ... 1.4 1 0 3 0
[5 rows x 14 columns]
Attributes and datatypes
age int64
gender int64
cp int64
trestbps int64
```

chol int64
fbs int64
restecg int64
thalach int64
exang int64
oldpeak float64
slope int64
ca object
thal object
heartdisease int64

dtype: object

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:
1.Probability of HeartDisease given evidence= restecg
0%| | 0/4 [00:00<?,?it/s]
0%| | 0/4 [00:00<?,?it/s]

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

2.Probability of HeartDisease given evidence= cp

0%| | 0/3 [00:00<?,?it/s]
0%| | 0/3 [00:00<?,?it/s]

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537

```
+------------------+----------------------+
| heartdisease(4) | 0.1321 |
+------------------+----------------------+
```

CONCLUSION

Here we implemented a Bayesian Network,modelled it on the Heart Disease Database,and predicted the chance of heart disease in the patients,based on given attributes.

EXERCISE

Apply Bayesian Network to any of the datasets in UCI Repository and find the related predictions.

LESSON - 8

APPLY EM ALGORITHM TO CLUSTER A SET OF DATA STORED IN A .CSV FILE

CONTENTS

EXPECTATION MAXIMIZATION (EM) ALGORITHM

When Machine Learning problems are faced with scenarios where several relevant features are available to build our model,but only a few of them are observable,we can use the EM algorithm to determine optimum values for these latent variables,and thereby obtain the model parameters. It is an iterative method that alternates between an Expectation step and a Maximization step.

1. E-Step: Estimate the missing variables in the dataset.
2. M-Step: Maximize the parameters of the model in the presence of the data.

Thus,parameters are computed which then helps to determine the distribution of latent/hidden variables in the next Expectation step.

GAUSSIAN MIXTURE MODEL (GMM)

The Gaussian Mixture Model helps to model a problem on the Expectation Maximization Algorithm,and thereby,makes it easier to find a solution to the problem. Basically,a GMM is where,we assume that the probability distribution of the different classes in a training set is a Gaussian or Normal distribution. Considering that each class in a dataset has it's own Gaussian distribution(in which case data is called multi-modal),we can estimate the parameters based on as many Gaussians as there are classes. Although other probability distributions can be used,Gaussian is the most common choice. The output for a datapoint will be the sum of the values expected by all of the Gaussians.

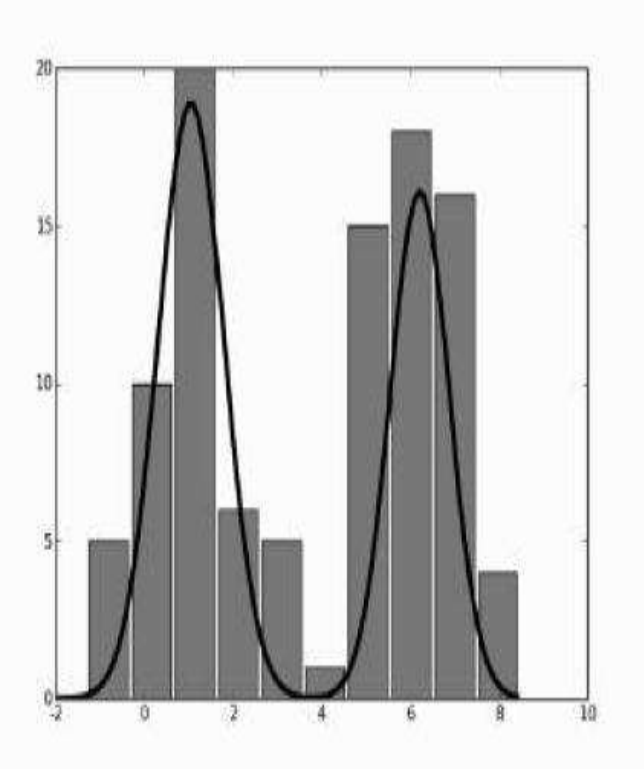


Figure 6: A Gaussian Mixture Model Histogram

The probability density function of a Gaussian Distribution for a 1-dimensional space is given by

$$P(X|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{\frac{-(x_j-\mu_j)^2}{2\sigma_j^2}} \quad (9)$$

For a d-dimensional feature space,the p.d.f. would be

$$G(X|\mu,\Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left(-\frac{1}{2}(X-\mu)'\Sigma(X-\mu)\right) \quad (10)$$

where

x - input vector

μ - 2-D Mean vector

Σ - 2x2 covariance matrix

GENERAL STEPS IN EM ALGORITHM

- Initialisation
  - Initialize the parameters with a set of initial values,assuming the observed values come from a certain probability distribution (here we assume Gaussian probability)
- Repeat until convergence:
  - E-Step- Compute the Expectation,guess the values of the missing data and update the variables
  - M-Step- Estimate the new parameters,and update the hypothesis
- Stop when convergence occurs

DATASET DESCRIPTION

We use a famous dataset,called the Iris Dataset,which contains attributes with respect to physical features of flowers. The attributes in the Iris dataset are:

- Sepal-Length
- Sepal-Width
- Petal-Length
- Petal-Width
- Species

– setosa

– versicolor

– virginica

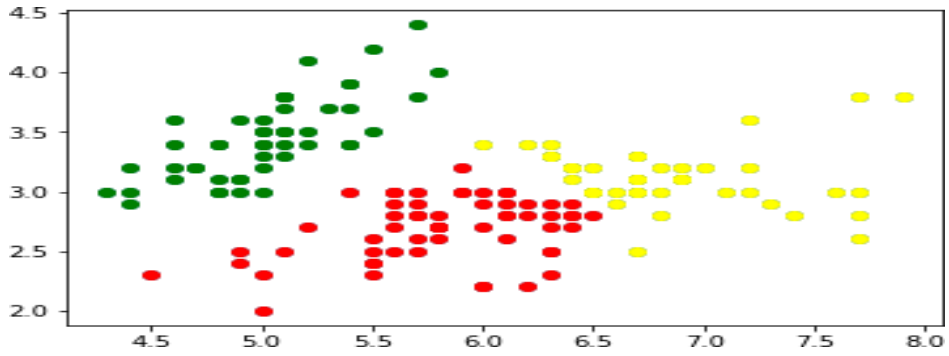
EM ALGORITHM IN PYTHON

Listing 13: Expectation Maximization in python

```
# −*− coding: utf−8 −*−
 import numpy as np import pandas as pd
import matplotlib.pyplot as plt from pandas
import DataFrame
from sklearn import datasets
from sklearn.mixture import GaussianMixture
# load the iris dataset
iris = datasets.load_iris()
# select first two columns
```

```
X = iris.data[:,:2]
# turn it into a dataframe
d = pd.DataFrame(X)
# plot the data
plt.scatter(d[0],d[1])
gmm = GaussianMixture(n_components = 3)
# Fit the GMM model for the dataset
# which expresses the dataset as a
# mixture of 3 Gaussian Distribution gmm.fit(d)
# Assign a label to each sample
labels = gmm.predict(d)
d['labels']= labels
d0 = d[d['labels']== 0]
d1 = d[d['labels']== 1]
d2 = d[d['labels']== 2]
# plot three clusters in same plot
plt.scatter(d0[0],d0[1],c ='r')
plt.scatter(d1[0],d1[1],c ='yellow')
# print the converged log−likelihood value
print(gmm.lower_bound_)
# print the number of iterations needed
# for the log−likelihood value to converge
print(gmm.n_iter_)
plt.scatter(d2[0],d2[1],c ='g')
```

Output



CONCLUSION

The EM algorithm was implemented using the Gaussian Mixture Model (GMM) for estimation,and the data was clustered and visualized.

EXERCISE

Apply the EM algorithm on heart disease dataset used in Bayesian Network

LESSON - 9

WRITE A PROGRAM TO IMPLEMENT K-NEARESTNEIGH- BOURALGORITHM TO CLASSIFY THE DATA SET.

CONTENTS

K-NEAREST NEIGHBOUR ALGORITHM

k-Nearest Neighbour(KNN) Algorithm is a supervised machine learning algorithm that can solve classification as well as regression problems. The basic idea is that,items that are similar are found in close proximity of each other. In KNN algorithm,the Euclidean distance between two points in a graph is calculated.

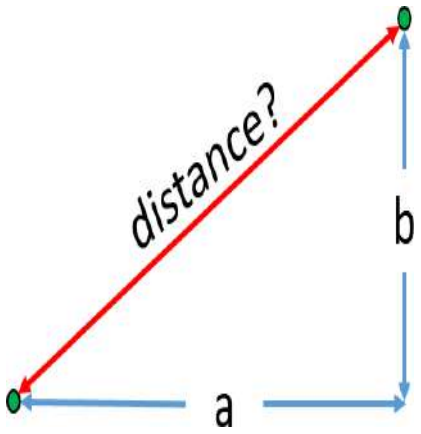


Figure 7: Euclidean Distance between two points to find similar items

When a and b are known,the straight line distance is calculated using the formula

$$distance = \sqrt{a^2 + b^2} \quad (11)$$

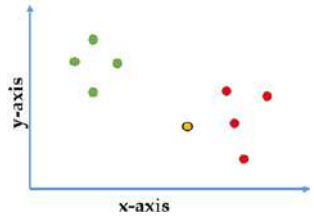


Figure 8: Sample Data points plotted on a 2-dimensional feature space

In Figure 8 we can see two sets of datapoints,green and red. to find the class of the yellow datapoint,we find the distance from it to k neighbours. Based on the class of the k neighbours,the yellow datapoint is classified. Here,it belongs to the set of red datapoints,based on the distance from them.

KNN PSEUDOCODE

1. Load the training data.
2. Prepare data by scaling,missing value treatment,and dimensionality reduction as required.
3. Find the optimal value for K:
4. Predict a class value for new data:
   (a) Calculate distance(X,Xi) from i=1,2,3,. . . .,n. where X= new data point,Xi= training data,distance as per your chosen distance metric.
   (b) Sort these distances in increasing order with corresponding train data.
   (c) From this sorted list,select the top 'K' rows.
   (d) Find the most frequent class from these chosen 'K' rows. This will be your predicted class.

DATASET DESCRIPTION

We use a famous dataset,called the Iris Dataset,which contains attributes with respect to physical features of flowers. The attributes in the Iris dataset are:

- Sepal-Length
- Sepal-Width
- Petal-Length
- Petal-Width
- Species

– setosa

– versicolor
– virginica

K-NEAREST NEIGHBOURS (KNN) IN PYTHON

Listing 14: KNN in Python

```
#Code −1
from random import seed
from random import randrange from csv import reader
from math import sqrt
# Load a CSV file
def load_csv(filename):
dataset = list()
with open(filename,'r') as file:
csv_reader = reader(file)
for row in csv_reader:
if not row:
continue dataset.append(row)
return dataset
# Convert string column to float
def str_column_to_float(dataset,column):
for row in dataset:
row[column] = float(row[column].strip())
# Convert string column to integer
def str_column_to_int(dataset,column):
class_values = [row[column] for row in dataset] unique = set(class_values)
lookup = dict()
for i,value in enumerate(unique):
lookup[value] = i
for row in dataset:
row[column] = lookup[row[column]]
return lookup
# Find the min and max values for each column
```

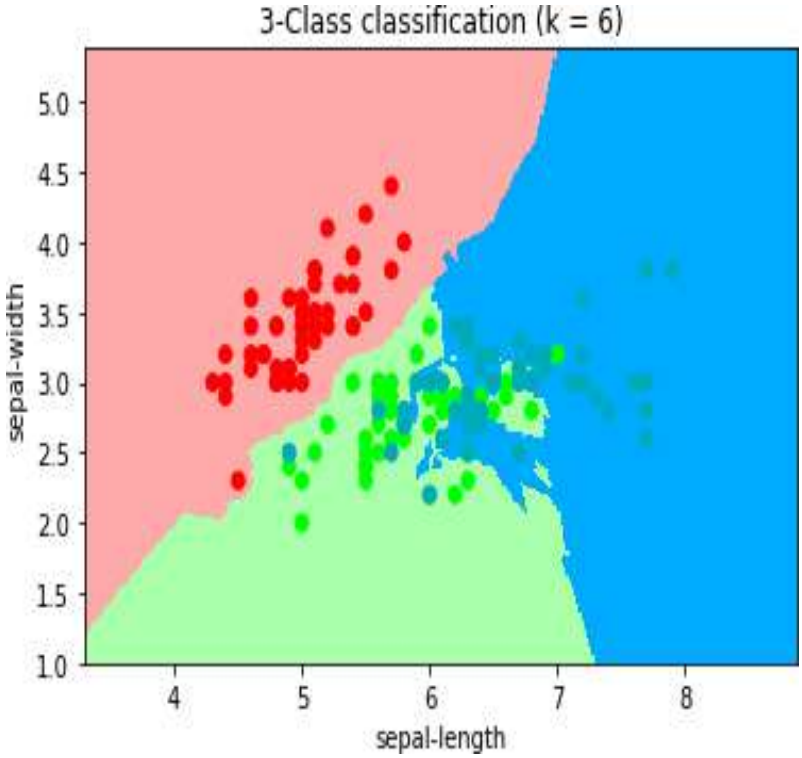
```
def dataset_minmax(dataset):
minmax = list()
for i in range(len(dataset[0])):
col_values = [row[i] for row in dataset]
value_min = min(col_values)
value_max = max(col_values)
minmax.append([value_min,value_max])
return minmax
# Rescale dataset columns to the range 0−1 def normalize_dataset(dataset,min-
max):
for row in dataset:
for i in range(len(row)):
row[i] = (row[i] − minmax[i][0]) / (minmax[i][1] − minmax[i][0])
# Split a dataset into k folds
def cross_validation_split(dataset,n_folds):
dataset_split = list()
dataset_copy = list(dataset)
fold_size = int(len(dataset) / n_folds) for _ in range(n_folds):
fold = list()
while len(fold) < fold_size:
index = randrange(len(dataset_copy))
fold.append(dataset_copy.pop(index))
dataset_split.append(fold)
return dataset_split
# Calculate accuracy percentage
def accuracy_metric(actual,predicted): correct = 0
for i in range(len(actual)):
if actual[i] == predicted[i]: correct += 1
return correct / float(len(actual)) * 100.0
# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset,algorithm,n_folds,*args):
 folds = cross_validation_split(dataset,n_folds)
```

```
scores = list() for fold in folds:
train_set = list(folds)
train_set.remove(fold)
train_set = sum(train_set,[])
test_set = list()
for row in fold:
row_copy = list(row)
test_set.append(row_copy)
row_copy[−1] = None
predicted = algorithm(train_set,test_set,*args)
actual = [row[−1] for row in fold]
accuracy = accuracy_metric(actual,predicted)
scores.append(accuracy) return scores
# Calculate the Euclidean distance between two vectors def euclidean_distance(row1,row2):
distance = 0.0
for i in range(len(row1)−1):
distance += (row1[i] − row2[i])**2 return sqrt(distance)
# Locate the most similar neighbors
def get_neighbors(train,test_row,num_neighbors):
distances = list()
for train_row in train:
dist = euclidean_distance(test_row,train_row)
distances.append((train_row,dist))
distances.sort(key=lambda tup: tup[1])
neighbors = list()
for i in range(num_neighbors):
neighbors.append(distances[i][0])
return neighbors
# Make a prediction with neighbors
def predict_classification(train,test_row,num_neighbors):
neighbors = get_neighbors(train,test_row,num_neighbors)
```

```
output_values = [row[−1] for row in neighbors]
prediction = max(set(output_values),key=output_values.count)
return prediction
# kNN Algorithm
def k_nearest_neighbors(train,test,num_neighbors):
 predictions = list()
for row in test:
output = predict_classification(train,row,num_neighbors)
predictions.append(output)
return(predictions)
# Test the kNN on the Iris Flowers dataset seed(1)
filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])−1):
str_column_to_float(dataset,i)
# convert class column to integers str_column_to_int(dataset,len(dataset[0])−1)
# evaluate algorithm
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset,k_nearest_neighbors,n_folds,num_neigh-
bors) print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
# −*− coding: utf−8 −*−
#Code −2 import matplotlib
#matplotlib.use('GTKAgg')
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors
import ListedColormap from sklearn
import neighbors,datasets
n_neighbors = 6
# import some data to play with iris = datasets.load_iris()
```

```
names = ['sepal−length','sepal−width','petal−length','petal−width','Class']
# prepare data
X = iris.data[:,:2] y = iris.target
h = .02
# Create color maps
cmap_light = ListedColormap(['#FFAAAA','#AAFFAA','#00AAFF'])
cmap_bold = ListedColormap(['#FF0000','#00FF00','#00AABB'])
# we create an instance of Neighbours Classifier and fit the data.
clf = neighbors.KNeighborsClassifier(n_neighbors,weights='distance')
clf.fit(X,y)
# calculate min,max and limits
x_min,x_max = X[:,0].min() − 1,X[:,0].max() + 1
y_min,y_max = X[:,1].min() − 1,X[:,1].max() + 1 xx,yy = np.meshgrid(np.
arange(x_min,x_max,h),np.arange(y_min,y_max,h))
# predict class using data and kNN classifier
Z = clf.predict(np.c_[xx.ravel(),yy.ravel()])
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx,yy,Z,cmap=cmap_light)
# Plot also the training points
plt.scatter(X[:,0],X[:,1],c=y,cmap=cmap_bold)
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
plt.title("3−Class classification (k = %i)" % (n_neighbors))
plt.xlabel(names[0])
plt.ylabel(names[1])
plt.show()
```

Output



CONCLUSION

Here we implemented the KNN Algorithm,to classify the iris dataset,to predict the species of the flowers based on given data.

EXERCISE

Apply KNN on suitable datasets for finding clusters

LESSON - 10

APPLY THE TECHNIQUE OF PRUNING FOR A NOISY DATA MONK2 DATA,AND DERIVE THE DECISION TREE FROM THIS DATA. ANALYZE THE RESULTS BY COMPARING THE STRUCTURE OF PRUNED AND UNPRUNED TREE

CONTENTS

DECISION TREES

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. Decision trees can handle both categorical and numerical data.

It involves entropy,information gain,GINI index to split the tree. GINI and entropy are measures of impurity of a node. A node having multiple classes is impure whereas a node having only one class is pure.

- Entropy – Entropy of a probability distribution function p tells us how much extra information we get from knowing the value of the feature,i.e,F=fi for a given data point

$$Entropy = \sum_{i=1}^{x'} -p(c_i)\log_2(p(c_i)) \qquad (12)$$

- Gini index – Probability that the selected element is incorrectly classified. It should be low

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i) \qquad (13)$$

where p(ci) is the probability/percentage of class ci in a node.

- Information gain(IG) - Measure of change in entropy,i.e.,the entropy of parent node minus sum of weighted entropies of child nodes

$$IG(T, A) = Entropy(T) - \sum_{v \in A} \frac{|T_v|}{T} * Entropy(T_v) \qquad (14)$$

PRUNING

Pruning is done to reduce the error in decision tree model. If the test-set of the data performs as well as the training set,after pruning,it's considered as the right choice.

In pruning,a node is removed along with its subtree,and is replaced by a leaf node,to which the majority classification of the removed node is assigned.

DECISION TREE BASIC STEPS

1. Find the best attribute to add as the root node
2. Split the training dataset into subsets,such that each subset of the training data has the same value for an attribute
3. Repeat steps 1 and 2 for each leaf node and corresponding subsets in the dataset,till no more attributes are left

DATASET DESCRIPTION

The Monk's Problems dataset was a standard dataset used to compare different learning algorithms. There are three Monks Problems,and we will be using the Monk2 Dataset here.

ATTRIBUTES

A simple set of 6 attributes with 2 classes is what it comprises of:

•class: 0,1

• a1: 1,2,3

• a2: 1,2,3

• a3: 1,2

• a4: 1,2,3

• a5: 1,2,3,4

• a6: 1,2

•Id: (A unique symbol for each instance)

DECISION TREE IN PYTHON

Listing 15: Decision Tree

```
# −*− coding: utf−8 −*−
import pandas as pd
import numpy as np
from sklearn.metrics
import confusion_matrix
from sklearn.model_selection
import train_test_split from sklearn.tree
import DecisionTreeClassifier
from sklearn import metrics
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
import os
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
#input data is read and printed
traindatainput = pd.read_csv('monks2_train.csv',delimiter=",")
print("Training Dataset")
print(traindatainput)
testdatainput = pd.read_csv('monks2_test.csv',delimiter=",")
print("Test Dataset")
print(testdatainput)
X_train = traindatainput[['a1','a2','a3','a4','a5','a6']].values
#from sklearn import preprocessing
y_train = traindatainput["label"] print(y_train[0:6])
X_test = testdatainput[['a1','a2','a3','a4','a5','a6']].values
#from sklearn import preprocessing
y_test = testdatainput["label"] print(y_test[0:6])
```

```
#split into training and testing set and print them to see their dimensions
#X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_
state=3)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
#using Entropy on the input data,create the decision tree
monksTree = DecisionTreeClassifier(criterion="entropy",max_depth=4)
monksTree.fit(X_train,y_train) predicted = monksTree.predict(X_test)
print(predicted)
#find and print the accuracy of the decision tree model
print("\nDecisionTrees's Accuracy: ",metrics.accuracy_score(y_test,predicted))
#Visualize the decision tree dot_data = StringIO()
export_graphviz(monksTree,out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue()) graph.write_
png('tree.png')
Image(graph.create_png())
#since the tree is long,we can change some parameters to see if the accuracy can
be improved,using the criteria of "gini" or "Entropy"
monksTree = DecisionTreeClassifier(criterion='gini') monksTree.fit(X_train,y_
train)
pred = monksTree.predict(X_test)
print('Criterion=gini',accuracy_score(y_test,pred))
monksTree = DecisionTreeClassifier(criterion='entropy')
monksTree.fit(X_train,y_train)
pred = monksTree.predict(X_test)
print('Criterion=entropy',accuracy_score(y_test,pred))
#Check if pruning can improve the results
#Pruning is done by specifying Max_Depth to reduce the depth of the tree
max_depth = []
acc_gini = []
acc_entropy = []
```

```
 for i in range(1,30):
monksTree = DecisionTreeClassifier(criterion='gini',max_depth=i)
monksTree.fit(X_train,y_train)
pred = monksTree.predict(X_test)
acc_gini.append(accuracy_score(y_test,pred))
####
monksTree  = DecisionTreeClassifier(criterion='entropy',max_depth=i)
monksTree.fit(X_train,y_train)
pred = monksTree.predict(X_test)
acc_entropy.append(accuracy_score(y_test,pred))
####
max_depth.append(i)
d = pd.DataFrame({'acc_gini':pd.Series(acc_gini)
,'acc_entropy':pd.Series(acc_entropy),
'max_depth':pd.Series(max_depth)})
# visualizing changes in parameters
plt.plot('max_depth','acc_gini',data=d,label='gini')
plt.plot('max_depth','acc_entropy',data=d,label='entropy')
plt.xlabel('max_depth')
plt.ylabel('accuracy') plt.legend()
#check accuracy with a shorter tree,with maxdepth of 7 and  criterion  of
entropy monksTree = DecisionTreeClassifier(criterion='entropy',max_depth=7)
monksTree.fit(X_train,y_train)
pred = monksTree.predict(X_test)
accuracy_score(y_test,pred)
print("Accuracy after reducing depth and using Entropy",accuracy_score(y_
test,pred))
```

Training Dataset:

label a1 a2 a3 a4 a5 a6 id

0 0 1 1 1 1 2 2 data_4

1 0 1 1 1 1 4 1 data_7

2 0 1 1 1 2 1 1 data_9

3 0 1 1 1 2 1 2 data_10

4 0 1 1 1 2 2 1 data_11

.. ..

164 0 3 3 2 2 3 1 data_421

165 0 3 3 2 2 3 2 data_422

166 1 3 3 2 3 1 1 data_425

167 0 3 3 2 3 2 1 data_427

168 0 3 3 2 3 4 2 data_432

[169 rows x 8 columns] Test Dataset:

label a1 a2 a3 a4 a5 a6 id 0 0 1 1 1 1 1 1 data_1

1 0 1 1 1 1 1 2 data_2

2 0 1 1 1 1 2 1 data_3

3 0 1 1 1 1 2 2 data_4

4 0 1 1 1 1 3 1 data_5

.. ..

427 0 3 3 2 3 2 2 data_428

428 0 3 3 2 3 3 1 data_429

429 0 3 3 2 3 3 2 data_430

430 0 3 3 2 3 4 1 data_431

431 0 3 3 2 3 4 2 data_432

[432 rows x 8 columns]

0 0

1 0

2 0

3 0

4 0

5 0

Name: label,dtype: int64 0 0

1 0

2 0

3 0

4 0

5 0

Name: label,dtype: int64 (169,6)

(432,6)

(169,)

(432,)

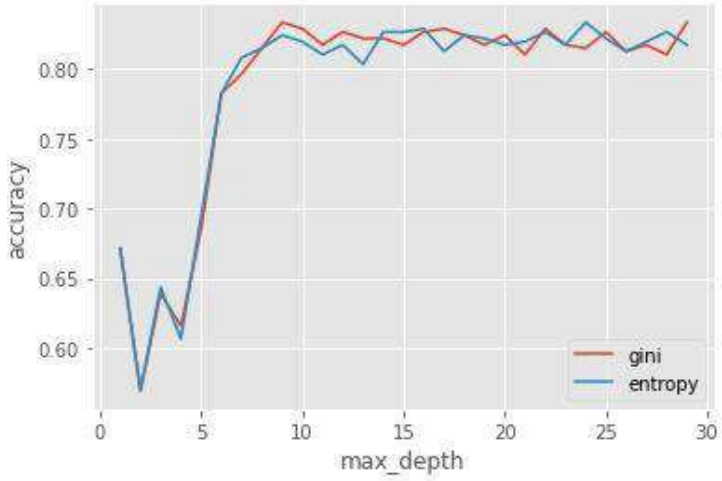
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1
0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1
1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1
0 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1
1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 1 0
0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1
1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0]

DecisionTrees's Accuracy: 0.6064814814814815

Criterion=gini 0.8240740740740741

Criterion=entropy 0.8217592592592593

Accuracy after reducing depth and using Entropy: 0.8078703703703703



Output

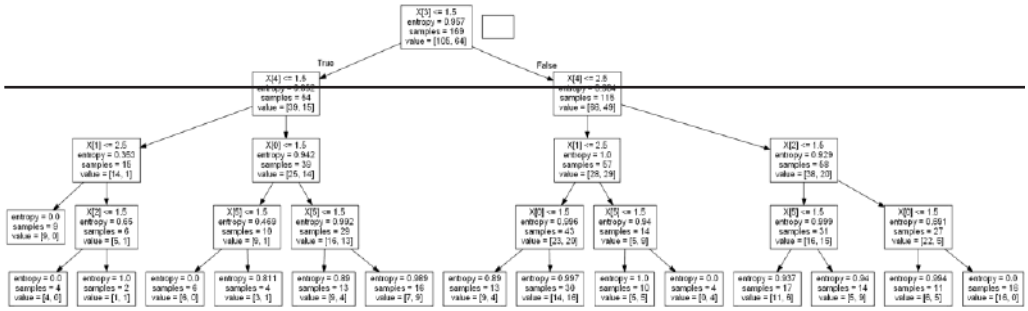


Figure 9: Unpruned Decision tree modelled on Monks 2 dataset

CONCLUSION

Here we modeled a Decision tree on the Monk-2 dataset,and analysed it's accuracy for pruned and unpruned trees. The pruned tree gave better accuracy w.r.t. Entropy and GINI,while the unpruned tree gave low accuracy,before applying any criteria.

EXERCISE

Apply Decision Tree Models on suitable datasets.

LESSON - 11

BUILD AN ARTIFICIAL NEURAL NETWORK BY IMPLEMENT ING THE BACKPROPAGATION ALGORITHM AND TEST THE SAME USING APPROPRIATE DATA SETS

CONTENTS

MULTI LAYER PERCEPTRON - MLP

A multilayer perceptron is a class of feedforward artificial neural networks.An MLP consists of at least three layers of nodes:

- Input layer
- Hidden layer
- Output layer

The hidden and output nodes use a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.

DATASET 1 - Wheat Seeds

Data Set Information:

The examined group comprised kernels belonging to three different varieties of wheat:

Kama,Rosa and Canadian,70 elements each,randomly selected for the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology.

The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combined harvested wheat grain originating from experimental fields,explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

No. of instances: 210

Attribute Information:

To construct the data,seven geometric parameters of wheat kernels were measured:

1. Area A
2. Perimeter P
3. Compactness C = 4 * Pi * A/P2
4. Length of kernel
5. Width of kernel
6. Asymmetry coefficient
7. Length of kernel groove

All of these parameters were real-valued continuous.

Link: https://archive.ics.uci.edu/ml/datasets/seeds or seed-data.csv

MLP IN PYTHON ON WHEAT-SEEDS DATASET

Listing 16: MLP python code on seed dataset

```
# −*− coding: utf−8 −*−
import numpy as np
import pandas as pd
data = pd.read_csv("seed−data.csv")
data = data.sample(frac=1).reset_index(drop=True)
data.head()
X = np.array(data)[:,1:−1] print(X[0])
data.shape
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(sparse=False)
Y = np.array(data)[:,−1]
Y = one_hot_encoder.fit_transform(np.array(Y).reshape(−1,1))
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25)
X_train.shape
def NeuralNetwork(X_train,Y_train,X_val=None,Y_
val=None,epochs=10,nodes=[],lr=0.07):
hidden_layers = len(nodes) − 1
weights = InitializeWeights(nodes)
```

```
for epoch in range(1,epochs+1):
weights = Train(X_train,Y_train,lr,weights)
if(epoch % 50 == 0):
print("Epoch {}".format(epoch))
print("Training Accuracy:{}".format(Accuracy(X_train,Y_train,weights))) if
X_val is not None:
print("Validation Accuracy:{}".format(Accuracy(X_val,Y_val,weights)))
return weights
def InitializeWeights(nodes):
layers = len(nodes)
weights = []
for i in range(1,layers):
w = [[np.random.uniform(−1,1)
for k in range(nodes[i−1] + 1)] for j in range(nodes[i])]
weights.append(np.matrix(w)) return weights
def ForwardPropagation(x,weights,layers):
activations,layer_input = [x],x
for j in range(layers):
activation = Sigmoid(np.dot(layer_input,weights[j].T))
activations.append(activation)
layer_input = np.append(1,activation)
return activations
def BackPropagation(y,activations,weights,layers):
outputFinal = activations[−1]
error = np.matrix(y − outputFinal)
for j in range(layers,0,−1):
currActivation = activations[j]
if(j > 1):
prevActivation = np.append(1,activations[j−1]) else:
prevActivation = activations[0]
delta = np.multiply(error,SigmoidDerivative(currActivation))
weights[j−1] += lr * np.multiply(delta.T,prevActivation)
```

```
w = np.delete(weights[j−1],[0],axis=1)
error = np.dot(delta,w)
return weights
def Train(X,Y,lr,weights):
layers = len(weights)
for i in range(len(X)):
x,y = X[i],Y[i]
x = np.matrix(np.append(1,x))
activations = ForwardPropagation(x,weights,layers)
weights = BackPropagation(y,activations,weights,layers)
return weights
def Sigmoid(x):
return 1/(1 + np.exp(−x))
def SigmoidDerivative(x):
return np.multiply(x,1−x)
def Predict(item,weights):
layers = len(weights)
item = np.append(1,item)

activations = ForwardPropagation(item,weights,layers)
outputFinal = activations[−1].A1
m,index = outputFinal[0],0
for i in range(1,len(outputFinal)):
if(outputFinal[i] > m):
m,index = outputFinal[i],i
y = [0 for i in range(len(outputFinal))]
y[index] = 1
return y
def Accuracy(X,Y,weights,display=False):
 correct = 0
for i in range(len(X)):
x,y = X[i],list(Y[i])
```

```
guess = Predict(x,weights)
if display == True:
print("\n\nInput:\n",x,"\nPredicted:\n",guess,"\nActual:\n",y)
if(y == guess):
correct += 1
elif display == True:
print("mispredicted")
return correct / len(X)
layers = [len(X[0]),11,8,5,5,len(Y[0])]
 lr,epochs = 0.1,500
weights = NeuralNetwork(X_train,Y_train,epochs=epochs,nodes=layers,lr=lr)
print("Final weights:\n",weights)
print("Testing Accuracy: {}".format(Accuracy(X_test,Y_test,weights,display =
True)))
import sklearn Y_result = []
for x in X_test:
guess = Predict(x,weights)
Y_result.append(guess)
print("R2 score : %f" % sklearn.metrics.r2_score(Y_test,Y_result))
print(sklearn.metrics.classification_report(Y_test,Y_result))
ytest,yresult = [],[]
for i in range(len(Y_test)):
ytest.append(Y_test[i][0])
yresult.append(Y_result[i][0])
from sklearn.metrics import roc_curve,roc_auc_score
fpr ,tpr ,thresholds = roc_curve ( ytest ,yresult)
import matplotlib.pyplot as plt
def plot_roc_curve(fpr,tpr):
plt.plot(fpr,tpr)
plt.axis([0,1,0,1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

plt.show()

plot_roc_curve (fpr,tpr)

Output:

Input: [13.2 0.8783 5.137 2.981 3.631 4.87]

Predicted: [0,1,0]

Actual: [0.0,0.0,1.0] mispredicted

Testing Accuracy: 0.6037735849056604 R2 score : -0.246756 precision recall f1-score support

0 0.41 0.69 0.51 13

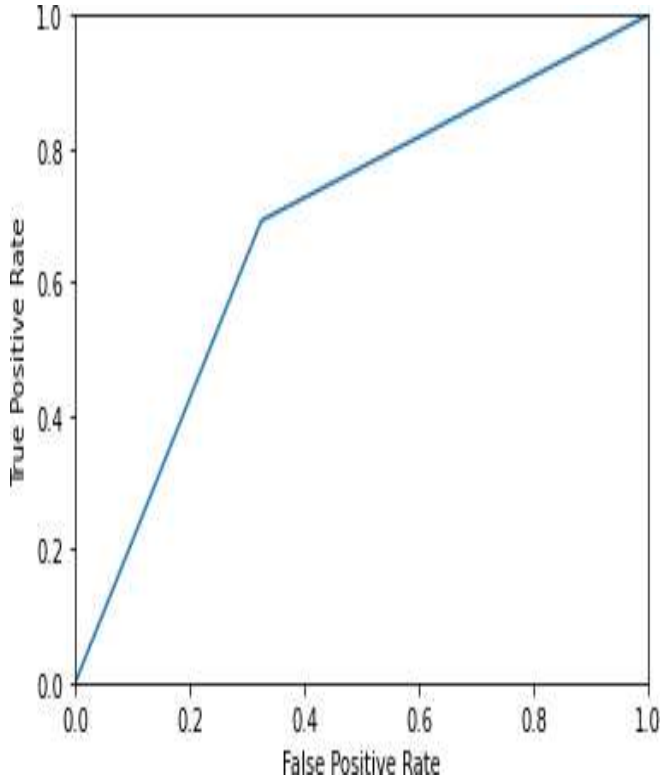
1 0.58 0.35 0.44 20

2 0.84 0.80 0.82 20

micro avg 0.60 0.60 0.60 53

macro avg 0.61 0.61 0.59 53

weighted avg 0.64 0.60 0.60 53



samples avg 0.60 0.60 0.60 53

DATASET 2 - Blood Transfusion Service Center

Attribute Information

Given is the variable name,variable type,the measurement unit and a brief description.The "Blood Transfusion Service Center" is a classification problem. The order of this listing corresponds to the order of numerals along the rows of the database.

1. R (Recency - months since last donation),
2. F (Frequency - total number of donation),
3. M (Monetary - total blood donated in c.c.),
4. T (Time - months since first donation),and
5. A binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

MLP IN PYTHON ON BLOOD TRANSFUSION DATASETListing

17: Multilayer Perceptron on Blood Transfusion dataset classification

```
import numpy as np
import pandas as pd
data = pd.read_csv("blood−transf.csv")
data = data.sample(frac=1).reset_index(drop=True)
data.head()
X = np.array(data)[:,1:−1] print(X[0])
data.shape
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(sparse=False)
Y = np.array(data)[:,−1]
Y = one_hot_encoder.fit_transform(np.array(Y).reshape(−1,1))
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25)
X_train.shape
def NeuralNetwork(X_train,Y_train,X_val=None,Y_val=None,epochs=10,nodes=[],
lr=0.07):
hidden_layers = len(nodes) − 1
```

```
weights = InitializeWeights(nodes)
for epoch in range(1,epochs+1):
weights = Train(X_train,Y_train,lr,weights)
if(epoch % 50 == 0):
print("Epoch {}".format(epoch))
print("Training Accuracy:{}".format(Accuracy(X_train,Y_train,weights))) if
X_val is not None:
print("Validation Accuracy:{}".format(Accuracy(X_val,Y_val,weights)))
return weights
def InitializeWeights(nodes):
layers = len(nodes)
weights = []
for i in range(1,layers):
w = [[np.random.uniform(−1,1)
for k in range(nodes[i−1] + 1)] for j in range(nodes[i])]
weights.append(np.matrix(w))
return weights
def ForwardPropagation(x,weights,layers):
activations,layer_input = [x],x
for j in range(layers):
activation = Sigmoid(np.dot(layer_input,weights[j].T))
activations.append(activation)
layer_input = np.append(1,activation)
return activations
def BackPropagation(y,activations,weights,layers):
outputFinal = activations[−1]
error = np.matrix(y − outputFinal)
for j in range(layers,0,−1):
currActivation = activations[j]
if(j > 1):
prevActivation = np.append(1,activations[j−1]) else:
prevActivation = activations[0]
```

```
delta = np.multiply(error,SigmoidDerivative(currActivation))
weights[j−1] += lr * np.multiply(delta.T,prevActivation)
w = np.delete(weights[j−1],[0],axis=1)
error = np.dot(delta,w)
return weights
def Train(X,Y,lr,weights):
layers = len(weights)
for i in range(len(X)):
x,y = X[i],Y[i]
x = np.matrix(np.append(1,x))
activations = ForwardPropagation(x,weights,layers)
weights = BackPropagation(y,activations,weights,layers)
return weights
def Sigmoid(x):
return 1/(1 + np.exp(−x))
def SigmoidDerivative(x):
return np.multiply(x,1−x)
def Predict(item,weights):
layers = len(weights)
item = np.append(1,item)
activations = ForwardPropagation(item,weights,layers)
outputFinal = activations[−1].A1
m,index = outputFinal[0],0
for i in range(1,len(outputFinal)):
if(outputFinal[i] > m):
m,index = outputFinal[i],i
y = [0 for i in range(len(outputFinal))]
y[index] = 1
return y
def Accuracy(X,Y,weights,display=False):
correct = 0
for i in range(len(X)):
```

```
x,y = X[i],list(Y[i])
guess = Predict(x,weights)
if display == True:
print("\n\nInput:\n",x,"\nPredicted:\n",guess,"\nActual:\n",y)
if(y == guess):
correct += 1
elif display == True:
print("mispredicted")
return correct / len(X)
layers = [len(X[0]),11,8,5,5,len(Y[0])]
lr,epochs = 0.1,500
weights = NeuralNetwork(X_train,Y_train,epochs=epochs,nodes=layers,lr=lr)
print("Final weights:\n",weights)
print("Testing Accuracy: {}".format(Accuracy(X_test,Y_test,weights,display =
True)))
import sklearn
Y_result = []
for x in X_test:
guess = Predict(x,weights)
Y_result.append(guess)
print("R2 score : %f" % sklearn.metrics.r2_score(Y_test,Y_result))
print(sklearn.metrics.classification_report(Y_test,Y_result))
ytest,yresult = [],[]
for i in range(len(Y_test)):
ytest.append(Y_test[i][0])
yresult.append(Y_result[i][0])
from sklearn.metrics import roc_curve,roc_auc_score
fpr ,tpr ,thresholds = roc_curve ( ytest ,yresult)
import matplotlib.pyplot as plt
def plot_roc_curve(fpr,tpr):
plt.plot(fpr,tpr) plt.axis([0,1,0,1])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.show()
plot_roc_curve (fpr,tpr)
\lstset{language=Python}
\lstset{basicstyle=\footnotesize}
\lstset{columns=fullflexible}
\lstset{frame=lines}
```

Output:

Testing Accuracy: 0.7593582887700535

R2 score : −0.316901

precision recall f1−score support

0 0.76 1.00 0.86 142

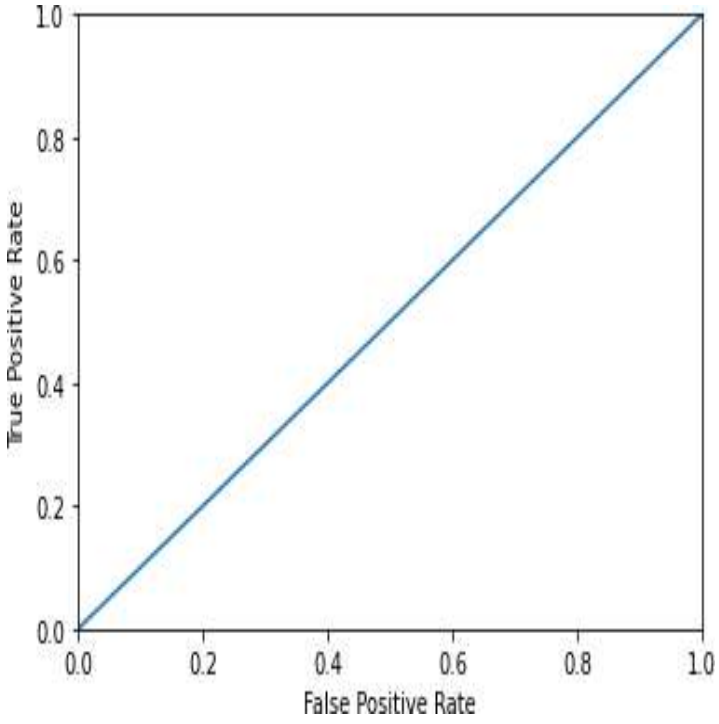
1 0.00 0.00 0.00 45

micro avg 0.76 0.76 0.76 187

macro avg 0.38 0.50 0.43 187

weighted avg 0.58 0.76 0.66 187

samples avg 0.76 0.76 0.76 187



CONCLUSION

The Multilayer Perceptron was executed on two different datasets,the “wheat-seeds dataset” and the “Blood Transfusion Center” dataset. Accuracy does not always improve if we increase the number of hidden layers. Overfitting leads to lesser accuracy. The number of hidden layers and neurons depends on the complexity of the problem. The number of epochs,learning rate and weights affect the result.

EXERCISE

Apply MLP on other datasets.

LESSON - 12

BUILD AN ARTIFICIAL NEURAL NETWORK BY IMPLEMENT-ING THE BACKPROPAGATION ALGORITHM AND TEST THE SAME USING APPROPRIATE DATA SETS

CONTENTS

SVM

It is a supervised learning algorithm. It is based on the idea of finding a hyper-plane that best separates the features into different domains. It is used for linearly separable dataset. There are 3 kernels in this which are radial basis function,polynomial and sigmoidal.

- Radial basis function : It is used in support vector machine classification.
- Polynomial kernel : It is used where there are similarity of vectors (training samples) in a feature space over polynomials of the original variables. Degree is given to the polynomial.
- Sigmoidal kernel : The Sigmoid Kernel comes from the Neural Networks field,where the bipolar sigmoid function is often used as an activation function for artificial neurons.

For different datasets ,any of the above kernels will provide high accuracy.

DATASET DESCRIPTION

Here we use a dataset with attributes related to Breast Cancer. This is used to detect breast cancer,using SVM classification.

Attributes in Dataset

The attributes in the dataset include :

1. **id**: ID number

2. **diagnosis**: The diagnosis of breast tissues (M = malignant,B = benign)
3. **radius_mean**: mean of distances from center to points on the perimeter
4. **texture_mean** standard deviation of gray-scale values
5. **perimeter_mean** mean size of the core tumor
6. **area_mean**
7. **smoothness_mean mean** of local variation in radius lengths
8. **compactness_mean** $\frac{meanofperimeter^2}{area} - 1.0$
9. **concavity_mean** mean of severity of concave portions of the contour
10. **concave points_mean** mean for number of concave portions of the contour
11. **symmetry_mean**
12. **fractal_dimension_mean** mean for "coastline approximation" - 1
13. **radius_se** standard error for the mean of distances from center to points on the perimeter
14. **texture_se** standard error for standard deviation of gray-scale values
15. **perimeter_se**
16. **area_se**
17. **smoothness_se** standard error for local variation in radius lengths
18. **compactness_se** standard error for perimeter^2/area − 1.0
19. **concavity_se** standard error for severity of concave portions of the contour
20. **concave points_se** standard error for number of concave portions of the contour
21. **symmetry_se**
22. **fractal_dimension_se** standard error for "coastline approximation" - 1
23. **radius_worst "worst" or largest mean value for mean** of distances from center to points on the perimeter
24. **texture_worst "worst" or largest mean** value for standard deviation of gray- scale values
25. **perimeter_worst**
26. **area_worst**
27. **smoothness_worst "worst" or largest mean** value for local variation in radius lengths
28. **compactness_worst "worst"**orlargest mean value for perimeter^2/area − 1.0
29. **concavity_worst "worst"** or largest mean value for severity of concave

portions of the contour

30. **concave points_worst "worst"** or largest mean value for number of concave portions of the contour
31. **symmetry_worst**
32. **fractal_dimension_worst** ”worst”orlargestmeanvaluefor”coastline approximation”− 1

SVM IN PYTHON

Listing 18: SVM in Python

```
# −*− coding: utf−8 −*−
import pandas as pd
df = pd.read_csv('breast−cancer−data.csv')
target = df['diagnosis']
s = set()
for val in target: s.add(val)
s = list(s)
import matplotlib.pyplot as plt
x = df['radius_mean']
y = df['concavity_mean']
open_x = x[:50]
open_y = y[:50]
high_x = x[50:100]
high_y = y[50:100]
plt.figure(figsize=(10,8))
plt.scatter(open_x,open_y,marker='+',color='green')
plt.scatter(high_x,high_y,marker='_',color='red')
plt.show()
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import numpy as np
## Drop rest of the features and extract the target values
df = df.drop(['radius_mean','concavity_mean'],axis=1)
Y = []
```

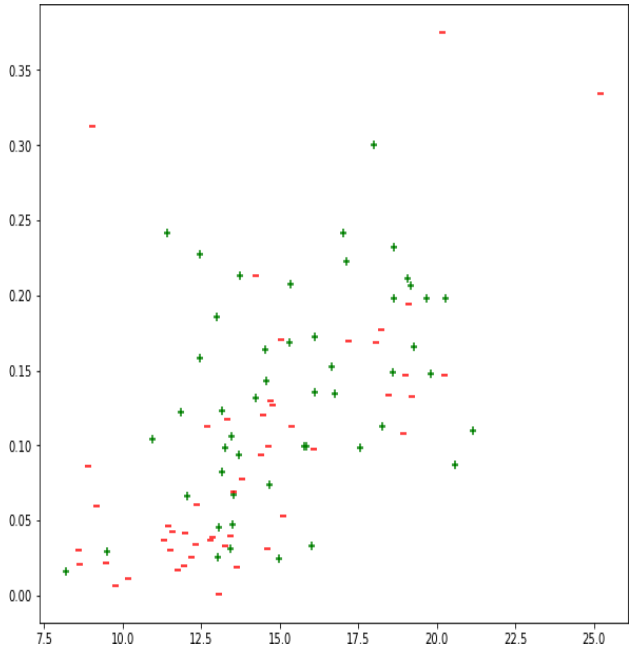
```
target = df['diagnosis']
for val in target:
if(val == 1):
Y.append(−1)
else:
Y.append(1)
df = df.drop(['diagnosis'],axis=1)
X = df.values.tolist()
## Shuffle and split the data into training and test set
X,Y = shuffle(X,Y)
x_train = []
y_train = []
x_test = []
y_test = []
x_train,x_test,y_train,y_test = train_test_split(X,Y,train_size=0.9)
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test  =  np.array(x_test)
y_test = np.array(y_test)
y_train = y_train.reshape(90,1)
y_test = y_test.reshape(10,1)
print(X[0])
import numpy as np
train_f1 = x_train[:,0]
train_f2 = x_train[:,1]
train_f1 = train_f1.reshape(90,1)
train_f2 = train_f2.reshape(90,1)
w1 = np.zeros((90,1))
w2 = np.zeros((90,1))
epochs = 1
alpha = 0.0001
while(epochs < 10000):
```

```
y = w1 * train_f1 + w2 * train_f2 prod = y * y_train
print(epochs)
count = 0
for val in prod:
if(val >= 1):
cost = 0
w1 = w1 − alpha * (2 * 1/epochs * w1)
w2 = w2 − alpha * (2 * 1/epochs * w2)
else:
cost = 1 − val
w1 = w1 + alpha * (train_f1[count] * y_train[count] − 2 * 1/epochs *
w1)
w2 = w2 + alpha * (train_f2[count] * y_train[count] − 2 * 1/epochs *
w2)
count += 1
epochs += 1
from sklearn.metrics import accuracy_score
## Clip the weights index = list(range(10,90))
w1 = np.delete(w1,index)
w2 = np.delete(w2,index)
w1 = w1.reshape(10,1)
w2 = w2.reshape(10,1)
## Extract the test data features test_f1 = x_test[:,0]
test_f2 = x_test[:,1]
test_f1 = test_f1.reshape(10,1)
test_f2 = test_f2.reshape(10,1)
## Predict
y_pred = w1 * test_f1 + w2 * test_f2 predictions = []
for val in y_pred:
if(val > 1):
predictions.append(1)
else:
```

predictions.append(−1)

print(accuracy_score(y_test,predictions))

Output



CONCLUSION

Support Vector Machine has been implemented to predict breast cancer based on the given dataset,and its accuracy has been calculated.

EXERCISE

Apply SVM on other datasets involving predictions.

LESSON - 13

IMPLEMENT LOGISTIC REGRESSION TO CLASSIFY THE PROBLEMS SUCH AS SPAM DETECTION. DIABETES PREDICTION AND SO ON.

CONTENTS

LOGISTIC REGRESSION

Linear regression models the relationship between two variables by fitting a linear equation to observed data. One variable is considered as an explanatory variable,and the other is considered as a dependent variable.

Logistic regression is a supervised learning classification algorithm used to predictthe probability of a target variable. The nature of target or dependent variable is dichotomous,which means there would be only two possible classes. In simple words,the dependent variable is binary in nature having data coded as either 1 (stands for yes) or 0 (stands for no).

DIABETES DATASET

The dataset contains the list of people who may or may not have diabetes. So the main idea of this project is to predict whether a person is susceptible to diabetesor not. For that the attributes the dataset contains are:

- Pregnancies
- Glucose
- Blood Pressure
- Skin Thickness
- Insulin
- BMI
- Diabetes Pedigree function

- Age

The dataset contains 768 observations which is divided into train and test set based on the percentage required. The dataset is analyzed using the following LR program.

LOGISTIC REGRESSION IN PYTHON

Listing 19: Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv("diabetes.csv")
data.shape
data.head()
X = data.drop(['SkinThickness','Outcome'],axis=1)
y = data['Outcome']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.20)
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train,y_train)
#Predict the response for test dataset y_pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
# Model Accuracy,how often is the classifier correct?
print("Accuracy:",accuracy_score(y_test,y_pred))
```

Output

```
Accuracy: 0.7662337662337663
```

CONCLUSION

The Logistic Regression is thus studied from the above given dataset and can be applied to any such dataset.

EXERCISE

Compare SVM,Linear Regression and Decision Tree on the same dataset.

Appendix A Datasets Used

Diabetes Dataset

https://www.kaggle.com/kumargh/pimaindiansdiabetescsv?select=pima-indi-ans-diabetes. csv

Heart-disease Dataset

https://archive.ics.uci.edu/ml/datasets/heart+disease

Iris Dataset

https://archive.ics.uci.edu/ml/datasets/iris

Monks Problem Dataset

https://archive.ics.uci.edu/ml/datasets/MONK's+Problems#:~:text=Data%20Set%20Information%3A,Thrun%2C%20J.

Wheat Seeds Dataset

https://www.kaggle.com/jmcaro/wheat-seedsuci

Blood Transfusion Dataset

https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center

Breast Cancer Dataset

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Appendix B Python Packages Used

B.1PySAT

PySAT integrates a number of widely used state-of-the-art SAT (satisfiability) solvers. We used Glucose3 in our lab,which is a mini satisfiability solver. To install PySAT: pip install python-sat

Numpy

Numpy is a python package useful in scientific computing problems. To install numpy on a system with Python on it,use the command:

pip install numpy

For alternate methods of installation,check their website Numpy.org.

Matplotlib

This is a library used mainly for making interactive visualization of data in Python. We used it for making the graphs in the programs listed in this manual. To install,use the command:

pip install matplotlib

For Alternate Methods of Installation,and tutorials,check out their Website,Matplotlib.org

Scikit-learn

Sklearn is the most useful and robust library for machine learning in Python. It contains tools for machine learning and statistical modeling including classification,regression,clustering and dimensionality reduction. This library is built upon NumPy,SciPy and Matplotlib. Install command is:

pip install scikit-learn

For complete reference to the tools provided in sklearn,check out their web-site,Scikit-learn.org

Pgmpy

This is a collection of programs that is designed for implementation of Bayesian Networks. To install the package,use the command:

pip install pgmpy

For alternate methods and other details,use their website Pgmpy.org

Pydotplus

This provides an Interface to Graphviz's Dot Language. We used it's modules to construct the decision tree and visualize it graphically. To install,use the command:
pip install pydotplus