ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

AICP309
ARTIFICIAL INTELLIGENCE
LAB MANUAL
III SEMESTER

Lab In-Charge: Dr. M. KALAISELVI GEETHA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. COMPUTER SCIENCE AND ENGINEERING (Artificial Intelligence and Machine Learning)

VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

MISSION

M1: Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.

M2: Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.

M3: Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs.

M4: Expose the students to the emerging technological advancements for meeting the demands of the industry.

B.E. COMPUTER SCIENCE AND ENGINEERING (Artificial Intelligence and Machine Learning)

PROGRAMME EDUCATIONAL OBJECTIVES (PEO)

PEO	PEO Statements
PEO1	To prepare graduates with potential to get employed in the right role and/or become entrepreneurs to contribute to the society.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science and Engineering.
PEO3	To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career.
PEO4	To train the graduates to communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

AICP309	ARTIFICIAL INTELLIGENCE LAB	L	T	P	C
		0	0	3	1.5

COURSE OBJECTIVES:

1. To learn Python Programming and Key Python Libraries relate to AI.
2. To formulate Real World Problems for AI.
3. To study specific algorithm design methods related to game playing.
4. To understand the process involved in computing with natural language specifically: Texts and Words.

LIST OF EXERCISES

1. Write a program to implement Blind Search Techniques like Breadth First and Depth First SearchTraversal.
2. Write a program to implement Tic-Tac-Toe game using A* Algorithm
3. .Write a program to implement Constraint Satisfaction Problem
4. Write a program to implement Logical Programming using Scipy
5. Write a program to implement Resolution by Refutation using Resolution Theorem Prover
6. Write a program to implement K-Nearest Neighbour (KNN) Algorithm .
7. Write a program to implement Naïve Bayes Classifier Algorithm
8. Write a program to implement Game Playing Algorithm like Mini-Max Algorithm and Alpha – Beta Pruning
9. Write a program to implement Natural Language Processing
    - Tokenization , Stemming using NLTK
    - Spell Checking using NLTK
10. Write a program to implement Medical Expert System

Lab Incharge Head Of the Department

COURSE OUTCOMES:

At the end of this course, the students will be able to

1. Understand the problem as a state space, design heuristics and select amongst differentsearch based techniques to solve them.

2.Analyze the design heuristics and apply different game based techniques to solve gameplaying problems.

3.Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.

Mapping of Course Outcomes with Programme Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	2	2	3	2	-	-	-	-	-	-	-	-
CO2	1	2	-	2	-	-	-	-	-	-	-	-
CO3	2	2	-	1	-	-	-	-	-	2	-	2

Rubric for CO3

Rubric for CO3 in Laboratory Courses				
Rubric	**Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks**			
	Up To 2.5 Marks	**Up To 5 Marks**	**Up To 7.5 Marks**	**Up To 10 marks**
Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem.	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors.	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description.

1. BLIND SEARCH TECHIQUE (UNINFORMED SEARCH ALGORITHMS)

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day. Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines.The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
  - **A Start State** - The state from where the search begins.
  - **A State Space** - Set of all possible states where you can be.
  - **A Goal Test** - A function that looks at the current state returns whether or not it is the goal state.
- The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.

The following uninformed search algorithms are discussed in this section. - **Breadth First Search** - **Depth First Search**

Each of these algorithms will have:

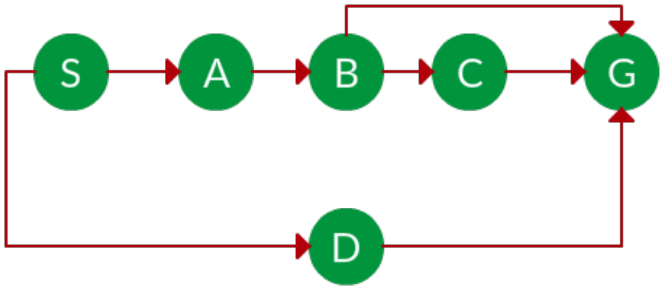
- A problem **graph**, containing the start node S and the goal node G.
- A **strategy**, describing the manner in which the graph will be traversed to get to G.
- A **fringe**, which is a data structure used to store all the possible states (nodes) that you can go from the current states.
- A **tree**, which results while traversing to the goal node.
- A solution **plan**, which the sequence of nodes from S to G.

Breadth First Search:

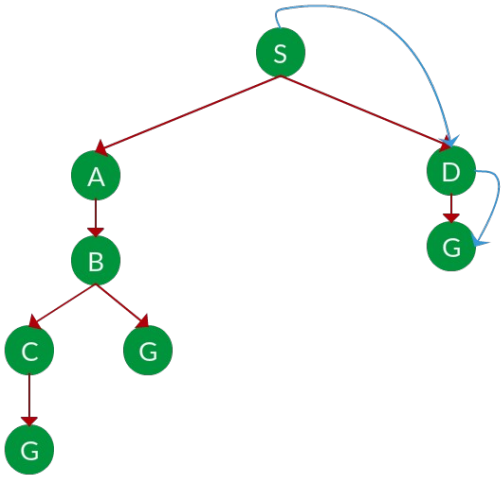
Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Example:

Which solution would BFS find to move from node S to node G if run on the graph below?



The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> D -> G

Let S = the depth of the shallowest solution. n^i = number of nodes in level i .

Time complexity: Equivalent to the number of nodes traversed in BFS until the shallowest solution.

$$T(n)=1+n^2+n^3+\ldots+n^s=O(n^s)$$

Space complexity: Equivalent to how large can the fringe get.

$$S(n)=O(n^s)$$

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

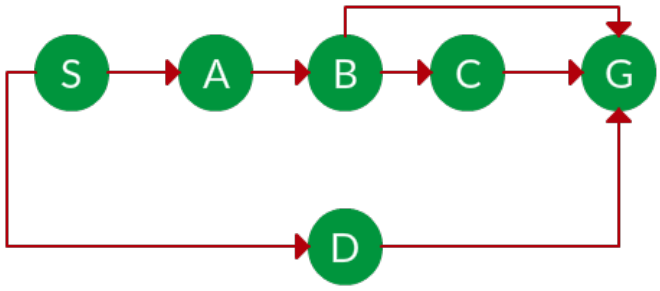
Optimality: BFS is optimal as long as the costs of all edges are equal.

Depth First Search:

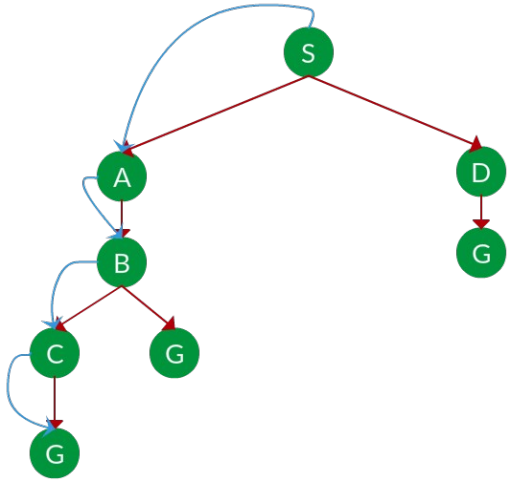
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Example:

Which solution would DFS find to move from node S to node G if run on the graph below?



The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> A -> B -> C -> G

Let **d** = the depth of the search tree = number of levels of the search tree. $\mathbf{n^i}$ = number of nodes in level **i**

Time complexity: Equivalent to the number of nodes traversed in DFS.

$T(n) = 1+n^2+n^3+\ldots+n^d=O(n^d)$

Space complexity: Equivalent to how large can the fringe get.

S (n) = O (n* x *d)

Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Optimality: DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

Exercise 1 - Blind Search Techniques

a) Breadth First Search (BFS)

AIM:

To write a python program to implement Breadth First Search algorithm in a graph.

ALGORITHM:

```
Algorithm BFS(g, start, goal)
    Input : g - A Graph
            start - starting node for the BFS
            goal - target node to be found
   Output : a path from start to goal in the graph g

    fringe <- Queue([start])
    predecessor <- Dict({start:NULL})
    visited <- Set([start])
    while fringe != Queue([]) //fringe is not empty
        current_node <- fringe.dequeue()
        if current_node = goal
            return traceback_path(goal,predecessor)
        for neighbour in g.neighbours(current)
            if neighbour not in visited:
                visited.add(neighbour)
                fringe.enqueue(neighbour)
                predecessor[neighbour] <- current
    return []
end Algorithm

Algorithm traceback_path(goal, predecessor)
    Input : goal - the goal node
            predecessor - a dictionary with each node as key
                        and its predecessor as value
   Output : a path from start node to goal node as List

   current <- goal
   path <- deque()
   do
       path.appendleft(goal)
       current <- predecessor[current]
    while current
    return path
end Algorithm
```

SOURCE CODE:

```
from collections import deque
class Graph:
    def __init__(self, directed):
        self.edges={}
        self.directed = directed
```

```
    def add_edge(self, node1, node2,_reversed=False):
        try:
            self.edges[node1].add(node2)
        except KeyError:
            self.edges[node1] = set()
            self.edges[node1].add(node2)
        if not self.directed and not _reversed:
            self.add_edge(node2,node1,True)

    def add_edges(self, edges):
        for edge in edges:
            self.add_edge(edge[0],edge[1])

    def neighbours(self, node):
        try:
            return self.edges[node]
        except KeyError:
            return []

    @staticmethod
    def traceback_path(goal, predecessor):
        current,path = goal,deque()
        while True:
            path.appendleft(current)
            current = predecessor[current]
            if current is None: break
        return path

    def bfs(self, start, goal):
        fringe = deque(start)
        visited = {start}
        predecessor = {start: None}
        current = '-'
        print(f"{'Current Node':15} | {'Fringe'}")
        while fringe:
            print(f"{current:15} |", *fringe)
            current = fringe.pop()
            if current == goal:
                path =  Graph.traceback_path(goal,predecessor)
                print(f"Path: {' => '.join(path)}")
                return path
            for x in self.neighbours(current):
                if x not in visited:
                    fringe.appendleft(x)
                    visited.add(x)
                    predecessor[x] = current

if __name__ == "__main__":
    g = Graph(directed = False)
    g.add_edges([
        ("A","B"),("A","S"),("S","G"),("S","C"),("C","F"),
        ("G","F"),("C","D"),("C","E"),("E","H"),("G","H")
    ])
    start,goal = "A","H"
```

```
    g.bfs(start,goal) or print("No paths Found!")
```

OUTPUT:

```
Current Node    | Fringe
-               | A
A               | S B
B               | S
S               | C G
G               | F H C
C               | D E F H
Path: A => S => G => H
```

b) Depth First Search (DFS)

AIM:

To write a python program to implement Depth First Search algorithm in a graph.

ALGORITHM:

```
Algorithm DFS(g, start, goal)
    Input : g - A Graph
            start - starting node for the DFS
            goal - target node to be found
   Output : a path from start to goal in the graph g

    fringe <- Stack([start])
    predecessor <- Dict({start:Null})
    visited <- Set([start])
    while fringe != Stack([]) //fringe is not empty
        current_node <- fringe.pop()
        if current_node = goal
            return traceback_path(goal,predecessor)
        for neighbour in g.neighbours(current)
            if neighbour not in visited:
                visited.add(neighbour)
                fringe.push(neighbour)
                predecessor[neighbour] <- current
    return []
end Algorithm

Algorithm traceback_path(goal, predecessor)
    Input : goal - the goal node
            predecessor - a dictionary with each node as key
                        and its predecessor as value
   Output : a path from start node to goal node as List

   current <- goal
   path <- deque()
   do
       path.appendleft(goal)
       current <- predecessor[current]
    while current
    return path
end Algorithm
```

SOURCE CODE:

```
# The below program is same as the previous one
# except the change of bfs to dfs in which
# the only change is using append function
# instead of appendleft on the fringe to change
# the behaviour of the fringe from queue to stack
from collections import deque

class Graph:
    def __init__(self, directed):
```

```
        self.edges={}
        self.directed = directed

    def add_edge(self, node1, node2,__reversed=False):
        try:
            self.edges[node1].add(node2)
        except KeyError:
            self.edges[node1] = set()
            self.edges[node1].add(node2)
        if not self.directed and not __reversed:
            self.add_edge(node2,node1,True)

    def add_edges(self, edges):
        for edge in edges:
            self.add_edge(edge[0],edge[1])

    def neighbours(self, node):
        try:
            return self.edges[node]
        except KeyError:
            return []

    @staticmethod
    def traceback_path(goal, predecessor):
        current,path = goal,deque()
        while True:
            path.appendleft(current)
            current = predecessor[current]
            if current is None: break
        return path

    def dfs(self, start, goal):
        fringe = deque(start)
        visited = {start}
        predecessor = {start: None}
        current = '-'
        print(f"{'Current Node':15} | {'Fringe'}")
        while fringe:
            print(f"{current:15} |", *fringe)
            current = fringe.pop()
            if current == goal:
                path =  Graph.traceback_path(goal,predecessor)
                print(f"Path: {' => '.join(path)}")
                return path
            for x in self.neighbours(current):
                if x not in visited:
                    fringe.append(x)
                    visited.add(x)
                    predecessor[x] = current

if __name__ == "__main__":
    g = Graph(directed = False)
    g.add_edges([
        ("A","B"),("A","S"),("S","G"),("S","C"),("C","F"),
```

```
        ("G","F"),("C","D"),("C","E"),("E","H"),("G","H")
    ])
    start,goal = "A","H"
    g.dfs(start,goal) or print("No paths Found!")
```

OUTPUT:

```
Current Node    | Fringe
-               | A
A               | B S
S               | B G C
C               | B G E D F
F               | B G E D
D               | B G E
E               | B G H
Path: A => S => C => E => H
```

2. HEURISTIC SEARCH TECHNIQUE

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed.

A Heuristic (or a heuristic function) takes a look at search algorithms. At each branching step, it evaluates the available information and makes a decision on which branch to follow. It does so by ranking alternatives. The Heuristic is any device that is often effective but will not guarantee work in every case.

So **why do we need heuristics**? One reason is to produce, in a reasonable amount of time, a solution that is good enough for the problem in question. It doesn't have to be the best- an approximate solution will do since this is fast enough. Most problems are exponential. Heuristic Search let us reduce this to a rather polynomial number.

We use this in AI because we can put it to use in situations where we can't find known algorithms. We can say Heuristic Techniques are weak methods because they are vulnerable to combinatorial explosion. Briefly, we can taxonomize such techniques of Heuristic into two categories:

Direct Heuristic Search Techniques in AI:

Other names for these are Blind Search, Uninformed Search, and Blind Control Strategy. These aren't always possible since they demand much time or memory. They search the entire state space for a solution and use an arbitrary ordering of operations. Examples of these are Breadth First Search (BFS) and Depth First Search (DFS).

Weak Heuristic Search Techniques in AI:

Other names for these are Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information. We need this extra information to compute preference among child nodes to explore and expand. Each node has a heuristic function associated with it. Examples are Best First Search (BFS) and A*. Before we move on to describe certain techniques, let's first take a look at the ones we generally observe. Below, we name a few.

- Best-First Search
- A* Search
- Bidirectional Search
- Tabu Search
- Beam Search
- Simulated Annealing
- Hill Climbing
- Constraint Satisfaction Problems

Exercise 2 - Heuristic Search Techniques

a) Solving Tic Tac Toe using A*algorithm and Gamestate class

AIM:

To write a python program to solve Tic Tac Toe problem using A*algorithm and GameState class.

ALGORITHM:

```
Class GameState
    state - List() of each slot in the game board with values X,O or N
    player - The current player who made the last move in the game board
    opponent - The other player who has to make the next move
    nextStates - The next possible states in the game board  as GameState
                  with the opponent as player and player as opponent
    score(player) - return the score of a player in the game state

Algorithm bestMove(g)
    Input : g - current GameState
   Output : best move for opponent as GameState

   (bestState, _) <- findBestMove(g,depth)
   return bestState
end Algorithm

Algorithm findBestMove(g,depth)
    Input : g - current GameState
            depth - depth of the game ply
   Output : best move for the opponent in the current state as
            Tuple(GamesState,Int)

    if depth = 0 or g.isFinal()
        return Tuple(g,g.score(g.player))
    (bestState,bestScore) <- Tuple(NULL,-INFINITY)
    for state in nextStates
        (_, score) <- findBestMove(state,depth-1)
        if score > bestScore
            (bestState,bestScore) <- (state,score)
    return (bestState,-bestScore)
end Algorithm
```

SOURCE CODE:

```
from copy import deepcopy

X,O,N = "X","O"," "

class GameState:
    def __init__(self,state=None,current_player=O):
        self.state = state or [N for _ in range(9)]
        self.player = current_player
```

```
    @property
    def opponent(self):
        return X if self.player == O else O

    def next_states(self):
        for i in range(9):
            if self.state[i] == N:
                next_state = deepcopy(self)
                next_state.state[i] = next_state.player = self.opponent
                yield next_state
        return

    def win_position(self):
        state = self.state
        win_pos = [
            (0,1,2),(3,4,5),(6,7,8),(0,3,6),
            (1,4,7),(2,5,8),(0,4,8),(2,4,6)
        ]
        for pos in win_pos:
            if (state[pos[0]] == state[pos[1]] == state[pos[2]] != N):
                return pos

    def winner(self):
        pos = self.win_position()
        if pos:
            return self.state[pos[0]]

    def is_filled(self):
        return N not in self.state

    def is_final(self):
        return True if self.winner() or self.is_filled() else False

    def has_won(self,player):
        return self.winner() == player

    def is_draw(self):
        return not self.winner() and self.is_final()

    def has_lost(self,player):
        winner = self.winner()
        return winner and winner != player

    def score(self,player):
        return (
            10 if self.has_won(player) else
            -10 if self.has_lost(player) else
            0
        )

    def is_valid_move(self,move):
        return self.state[move]==N

    def best_move(self,ply_depth):
```

```
        (best_state,_) = self.find_best_move(ply_depth)
        return best_state

    def find_best_move(self,depth):
        if depth == 0 or self.is_final():
            return (None, self.score(self.player))
        best_state,best_score = (None, None)
        for next_state in self.next_states():
            (_,score) =  next_state.find_best_move(depth-1)
            if best_score == None or (score >best_score):
                best_state,best_score = (next_state,score)
        return (best_state,best_score/-2)

    def __str__(self):
        return  '''\
Player: {} Opponent: {}
-------------
| {} | {} | {} |
-------------
| {} | {} | {} |
-------------
| {} | {} | {} |
-------------
'''.format(self.player,self.opponent,*self.state)

    def ai_move(self,ply_depth=8):
        best_state = self.best_move(ply_depth)
        if best_state:
            self.__dict__ = best_state.__dict__

    def player_move(self,i):
        if self.is_valid_move(i):
            self.state[i] = self.player = self.opponent
if __name__ == "__main__":
    g = GameState()
    print(g)
    while True:
        pos = int(input("Enter a position to play [0-8]:"))
        while not g.is_valid_move(pos):
            print("\nCell already filled!\n")
            pos = int(input("Please, Enter another position to play [0-8]:"))
        g.player_move(pos)
        print(g)
        if g.is_final(): break
        print("The AI plays:")
        g.ai_move()
        print(g)
        if g.is_final(): break
    winner = g.winner()
    print(
        "Player Wins!"if winner == X else
        "AI Wins!" if winner == O else
        "Match Draw!"
    )
```

OUTPUT:

```
Player: O Opponent: X
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------

Enter a position to play [0-8]: 4
Player: X Opponent: O
-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

The AI plays:
Player: O Opponent: X
-------------
| O |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

Enter a position to play [0-8]: 4

Cell already filled!

Please, Enter another position to play [0-8]: 7
Player: X Opponent: O
-------------
| O |   |   |
-------------
|   | X |   |
-------------
|   | X |   |
-------------

The AI plays:
Player: O Opponent: X
-------------
| O | O |   |
-------------
|   | X |   |
-------------
|   | X |   |
-------------
```

```
Enter a position to play [0-8]: 2
Player: X Opponent: O
-------------
| O | O | X |
-------------
|   | X |   |
-------------
|   | X |   |
-------------

The AI plays:
Player: O Opponent: X
-------------
| O | O | X |
-------------
|   | X |   |
-------------
| O | X |   |
-------------

Enter a position to play [0-8]: 3
Player: X Opponent: O
-------------
| O | O | X |
-------------
| X | X |   |
-------------
| O | X |   |
-------------

The AI plays:
Player: O Opponent: X
-------------
| O | O | X |
-------------
| X | X | O |
-------------
| O | X |   |
-------------

Enter a position to play [0-8]: 8
Player: X Opponent: O
-------------
| O | O | X |
-------------
| X | X | O |
-------------
| O | X | X |
-------------

Match Draw!
```

b) Solving Tic Tac Toe using A*algorithm with Tkinter GUI Toolkit

AIM:

To write a python program to solve Tic Tac Toe problem using A*algorithm with Tkinter GUI Toolkit.

ALGORITHM:

```
Class GameState
    state - List() of each slot in the game board with values X,O or N
    player - The current player who made the last move in the game board
    opponent - The other player who has to make the next move
    nextStates - The next possible states in the game board  as GameState
                 with the opponent as player and player as opponent
    score(player) - return the score of a player in the game state
class Game - the Tkinter app class for Tic Tac Toe

Algorithm bestMove(g)
    Input : g - current GameState
   Output : best move for opponent as GameState

   (bestState, _) <- findBestMove(g,depth)
   return bestState
end Algorithm

Algorithm findBestMove(g,depth)
    Input : g - current GameState
            depth - depth of the game ply
   Output : best move for the opponent in the current state as
            Tuple(GamesState,Int)

    if depth = 0 or g.isFinal()
        return Tuple(g,g.score(g.player))
    (bestState,bestScore) <- Tuple(NULL,-INFINITY)
    for state in nextStates
        (_, score) <- findBestMove(state,depth-1)
        if score > bestScore
            (bestState,bestScore) <- (state,score)
    return (bestState,-bestScore)
end Algorithm
```

SOURCE CODE:

```
from tkinter import Tk, Button, messagebox

# below is the same program from the previous part
# without the __str__ function in GameState class
# as it is not required for the gui
from copy import deepcopy

X,O,N = "X","O"," "

class GameState:
    def __init__(self,state=None,current_player=O):
```

```
        self.state = state or [N for _ in range(9)]
        self.player = current_player

    @property
    def opponent(self):
        return X if self.player == O else O

    def next_states(self):
        for i in range(9):
            if self.state[i] == N:
                next_state = deepcopy(self)
                next_state.state[i] = next_state.player = self.opponent
                yield next_state
        return
    def win_position(self):
        state = self.state
        win_pos = [
            (0,1,2),(3,4,5),(6,7,8),(0,3,6),
            (1,4,7),(2,5,8),(0,4,8),(2,4,6)
        ]
        for pos in win_pos:
            if (state[pos[0]] == state[pos[1]] == state[pos[2]] != N):
                return pos

    def winner(self):
        pos = self.win_position()
        if pos: return self.state[pos[0]]

    def is_filled(self):
        return N not in self.state

    def is_final(self):
        return True if self.winner() or self.is_filled() else False

    def has_won(self,player):
        return self.winner() == player

    def is_draw(self):
        return not self.winner() and self.is_final()

    def has_lost(self,player):
        winner = self.winner()
        return winner and winner != player

    def score(self,player):
        return (
            10 if self.has_won(player) else
            -10 if self.has_lost(player) else
            0
        )

    def is_valid_move(self,move):
        return self.state[move]==N
```

```
    def best_move(self,ply_depth):
        (best_state,_) = self.find_best_move(ply_depth)
        return best_state

    def find_best_move(self,depth):
        if depth == 0 or self.is_final():
            return (None, self.score(self.player))
        best_state,best_score = (None, None)
        for next_state in self.next_states():
            (_,score) =  next_state.find_best_move(depth-1)
            if best_score == None or (score >best_score):
                best_state,best_score = (next_state,score)
        return (best_state,best_score/-2)

    def ai_move(self,ply_depth=8):
        best_state = self.best_move(ply_depth)
        if best_state:
            self.__dict__ = best_state.__dict__

    def player_move(self,i):
        if self.is_valid_move(i):
            self.state[i] = self.player = self.opponent

class Game():
    '''The class for the gui'''
    def __init__(self):
        self.app = Tk()
        self.app.title("Tic Tac Toe")
        self.app.resizable(width=False, height=False)
        self.board = GameState()
        self.size = 3
        self.buttons = [
            Button(self.app,text=x)
            for x in self.board.state
        ]
        for i in range(9):
            x,y = i//self.size,i%self.size
            self.buttons[i].grid(row = x,column = y)
            self.buttons[i]["command"] = lambda x = i: self.move(x)
        reset = Button(self.app,text = "reset",command = self.reset)
        reset.grid(row =self.size+1,column=0,columnspan = 4)
        self.reset()
        self.update()

    def update(self):
        for i in range(0,9):
            self.buttons[i]["text"] = self.board.state[i]
            if self.board.state[i]!= N:
                self.buttons[i]["disabledforeground"] = "black"
                self.buttons[i]["state"]="disabled"
        win_pos = self.board.win_position()
        if win_pos:
            for i in range(0,9):
                self.buttons[i]["state"]="disabled"
```

```
            for i in win_pos:
                self.buttons[i]["disabledforeground"]="red"
            self.retry_popup(self.board.winner()+" Wins!\n")
            return
        if self.board.is_draw():
            self.retry_popup("Match Draw!\n")
            return

    def reset(self):
        self.board = GameState()
        for i in range(0,9):
            self.board.state[i] = N
            self.buttons[i]["state"]="normal"
        self.update()

    def move(self,i):
        self.board.player_move(i)
        self.update()

        self.app.config(cursor="watch")
        self.app.update()
        self.board.ai_move()
        self.app.config(cursor="")
        self.app.update()
        self.update()

    def retry_popup(self,message):
        msg = messagebox.askretrycancel(message = message+"Do you want to
retry ?",icon = "question")
        if not msg:
            self.app.destroy()
        else:
            self.reset()

    def mainloop(self):

        self.app.mainloop()
if __name__ == "__main__":
    Game().mainloop()
```

OUTPUT:

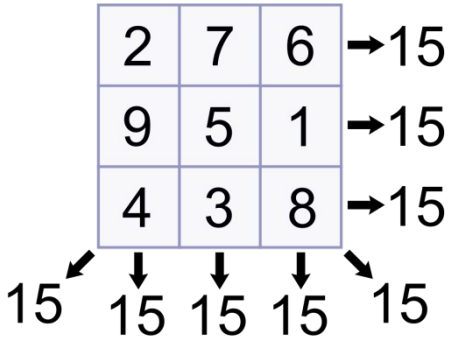
3. CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint is nothing but a limitation or a restriction. **Working with AI**, we may need to satisfy some constraints to solve problems. A constraint satisfaction problem (CSP) consists of

- a set of variables,
- a domain for each variable, and
- a set of constraints.

The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints.

Let's try solving a problem this way. Let's talk of a magic square. This is a sequence of numbers- usually integers- arranged in a square grid. The numbers in each row, each column, and each diagonal all add up to a constant which we call the *Magic Constant*. For example consider the below picture:



Magic square with **magic constant** 15

Since the values add up to the constant 15 in all directions, surely, this is a magic square!

Let's implement this with Python.

Exercise 3 - Constraint Satisfaction Problems

Magic Square

AIM:

To write a python program to implement a Constraint Satisfaction Problem.

ALGORITHM:

```
Algorithm MagicSquare(m)
    Input : m - a square matrix of order n
   Output : Bool - wheather m is a magic square

    sums = Set([])
    diag_b <- 0
    diag_f <- 0
    for i <- 0 to n
        row <- 0
        col <- 0
        for j <- 0 to n
            row <- row + m[i][j]
            col <- col + m[j][i]
        sums.add(row)
        sums.add(col)
        diag_b <- diag_b + m[i][i]
        diag_f <- diag_f + m[i][n-i-1]
    sums.add(diag_b)
    sums.add(diag_f)
    return (not sums.length > 1)
end Algorithm
```

SOURCE CODE:

```
def columns(array): return zip(*array)
def magic_square(m):
    n = len(m)
    row_sums = {sum(row) for row in m}
    col_sums = {sum(col) for col in columns(m)}
    diagonal_sums = {
        sum([m[i][i] for i in range(n)]),
        sum([m[i][-i-1] for i in range(n)])
    }
    sums = row_sums|col_sums|diagonal_sums
    return len(sums)==1

if __name__ == "__main__":
    square1 = [
        [1,2,3],
        [4,5,6],
        [7,8,9]
    ]
    square2 =[
        [2,7,6],
```

```
        [9,5,1],
        [4,3,8]
    ]
    print(magic_square(square1))
    print(magic_square(square2))
```

OUTPUT:

```
False
True
```

ALTERNATE METHOD USING NUMPY:

```
import numpy as np

def magic_square(m):
    m = np.array(m)
    n = m.shape[0]
    sums = {
        *m.sum(1), # row sums
        *m.sum(0), # col sums
        m.diagonal().sum(), # backward diagonal sum
        np.fliplr(m).diagonal().sum() # forward diagonal sum
    }
    return len(sums)  == 1

if __name__ == "__main__":
    square1 = [
        [1,2,3],
        [4,5,6],
        [7,8,9]
    ]
    square2 =[
        [2,7,6],
        [9,5,1],
        [4,3,8]
    ]
    print(magic_square2(square1))
    print(magic_square2(square2))
```

OUTPUT:

```
False
True
```

4. LOGIC PROGRAMMING USING SYMPY

Artificial Intelligence (AI) is the ability for an artificial machine to act intelligently. Logic Programming is a method that computer scientists are using to try to allow machines to reason because it is useful for knowledge representation. In logic programming, logic is used to represent knowledge and inference is used to manipulate it. The logic used to represent knowledge in logic programming is clausal form which is a subset of first-order predicate logic. It is used because first-order logic is well understood and able to represent all computational problems. Knowledge is manipulated using the resolution inference system which is required for proving theorems in clausal-form logic. The diagram below shows the essence of logic programming.

SymPy

- SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.
- SymPy only depends on mpmath, a pure Python library for arbitrary floating point arithmetic, making it easy to use.

INSTALLING SYMPY MODULE:

pip install sympy

SymPy as a calculator:

SymPy defines following numerical types: *Rational* and *Integer*. The Rational class represents a rational number as a pair of two Integers, numerator and denominator, so Rational (1, 2) represents 1/2,Rational(5, 2) 5/2 and so on. The Integer class represents Integer number.

Example 1:

```
# import everything from sympy module
from sympy import *
a = Rational(5, 8)
print("value of a is :" + str(a))
b = Integer(3.579)
print("value of b is :" + str(b))
```

Output:

```
value of a is :1/2
value of b is :3
```

SymPy uses mpmath in the background, which makes it possible to perform computations using arbitrary-precision arithmetic. That way, some special constants, like exp, pi, oo (Infinity), are treated as symbols and can be evaluated with arbitrary precision.

Example 2:

```
# import everything from sympy module
from sympy import *
# you can\'t get any numerical value
p = pi**3
```

```
print("value of p is :" + str(p))
# evalf method evaluates the expression to a floating-point number
q = pi.evalf()
print("value of q is :" + str(q))
# equivalent to e \^ 1 or e \*\* 1
r = exp(1).evalf()
print("value of r is :" + str(r))
s = (pi + exp(1)).evalf()
print("value of s is :" + str(s))
rslt = oo + 10000
print("value of rslt is :" + str(rslt))
if oo > 9999999 :
    print("True")
else:
    print("False")
```

Output:

```
value of p is :pi**3
value of q is :3.14159265358979
value of r is :2.71828182845905
value of s is :5.85987448204884
value of rslt is :oo
True
```

In contrast to other Computer Algebra Systems, in SymPy you have to declare symbolic variables explicitly using Symbol() method.

Example 3:

```
# import everything from sympy module
from sympy import * x = Symbol('x')
y = Symbol('y')
z = (x + y) + (x-y)
print("value of z is :" + str(z))
```

Output:

```
value of z is :2*x
```

Calculus:

The real power of a symbolic computation system such as SymPy is the ability to do all sorts of computations symbolically. SymPy can simplify expressions, compute derivatives, integrals, and limits, solve equations, work with matrices, and much, much more, and do it all symbolically. Here is a small sampling of the sort of symbolic power SymPy is capable of, to whet your appetite.

Exercise 4 - Logic Programming Using SymPy

a) Checking for Prime numbers

AIM:

To write a python program for checking prime numbers using SymPy.

FUNCTIONS USED:

```
sympy.prime(n)
    Input : n - A number
   Output : nth prime number

sympy.isprime(n)
    Input : n - A number
   Output : Bool - whether n is a prime

sympy.primerange(start, end)
    Input : start - start of the prime range
            end - end of the prime range
   Output : Generator yeilding prime numbers in
            the range [start,end)

sympy.sieve.primerange(n)
    Same as sympy.primerange
    But uses a Sieve instance to store the
    previously computed values which is
    more efficient and consumes more space

sympy.randprime(start, end)
    Input : start - start of the  range
            end - end of the  range
   Output : A random prime number within
            the range [start,end)

sympy.prevprime(n)
    Input : n - A number
   Output : largest prime number less than n

sympy.nextprime(n)
    Input : n - A number
   Output : smallest prime number greater than n
```

SOURCE CODE:

```
import sympy
if __name__ == "__main__":
    print("The 5th prime is:",sympy.prime(5))
    print("The 13th prime is:",sympy.prime(13))
    print("Is 5 a prime number:",sympy.isprime(5))
    print("Is 6 a prime number:",sympy.isprime(6))
    print(
        "The prime numbers between 0 & 100 are:",
```

```
        list(sympy.primerange(0, 100)),sep="\n"
    )
    print(
        "The prime numbers between 0 & 100 are(Using Sieve):",
        list(sympy.sieve.primerange(0, 100)),sep="\n"
    )
    print("A random prime between 0 & 100:", sympy.randprime(0, 100))
    print("A random prime between 0 & 100:", sympy.randprime(0, 100))
    print("The largest prime less than 50 is:", sympy.prevprime(50))
    print("The smallest prime greater than 50 is:", sympy.nextprime(50))
```

OUTPUT:

```
The 5th prime is: 11
The 13th prime is: 41
Is 5 a prime number: True
Is 6 a prime number: False
The prime numbers between 0 & 100 are:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97]
The prime numbers between 0 & 100 are(Using Sieve):
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97]
A random prime between 0 & 100: 37
A random prime between 0 & 100: 11
The largest prime less than 50 is: 47
The smallest prime greater than 50 is: 53
```

b) Matching Mathematical Exprressions

AIM:

To write a python program for matching mathematical exprressions using SymPy.

FUNCTIONS USED:

```
sympy.diff(f, *variables)
    Input : f - A mathematical function
            variables - Mathematical variables
   Output : The deriviative of the function f
            with respect to the variables

sympy.integrate(f, *variables)
    Input : f - A mathematical function
            variables - Mathematical variables
   Output : The anti-deriviative or integral of the function f
            with respect to the variables

sympy.sin(x)
    Input : x - A number
   Output : Mathematical sine value of the number x

sympy.cos(x)
    Input : x - A number
   Output : Mathematical cosine value of the number x

sympy.exp(x)
    Input : x - A number
   Output : Mathematical exponential  value = e^x

sympy.limit(e, x, x0, dir)
    Input : e - expression, the limit of which is to be taken
            x -  variable in the limit.
            x0 - the value toward which x tends.
            dir - direction of approach of the limit
                  accepts the values ["+", "-", "+-"]
   Output : The mathematical limit of the expression e as
            the variable x tends to x0

sympy.solve(f,*variables)
    Input : f   - polynomial
                - transcendental
                - piecewise combinations of the above
                - systems of linear and polynomial equations
                - systems containing relational expressions
            variables - Mathematical variables
   Output : The required mathematical solution
            for the input(varies according to the input)
```

SOURCE CODE:

```
from sympy import (
    diff, integrate, sin, cos,
```

```
    exp, limit, solve, oo
)
from sympy.abc import x
if __name__ == "__main__":
    print("The derivative of sin(x)*e^x is :", diff(sin(x)*exp(x), x))
    print(
        "The indefinite integration of e^x*sin(x)+e^x*cos(x) is :",
        integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
    )
    print(
        "The definite integration of sin(x^2) from -oo to oo is :",
        integrate(sin(x**2), (x, -oo, oo))
    )
    print(
        "The limit of sin(x)/x as x tends to 0 is :",
        limit(sin(x)/x, x, 0)
    )
    print(
        "The roots of the equation x^2-2 = 0 are :",
        solve(x**2 - 2, x)
    )
```

OUTPUT:

```
The derivative of sin(x)*e^x is : exp(x)*sin(x) + exp(x)*cos(x)
The indefinite integration of e^x*sin(x)+e^x*cos(x) is : exp(x)*sin(x)
The definite integration of sin(x^2) from -oo to oo is : sqrt(2)*sqrt(pi)/2
The limit of sin(x)/x as x tends to 0 is : 1
The roots of the equation x^2-2 = 0 are : [-sqrt(2), sqrt(2)]
```

5. RESOLUTION BY REFUTATION

Resolution is one kind of proof technique that works this way - 1. select two clauses that contain conflicting terms 2. combine those two clauses and
3. cancel out the conflicting terms.

For example we have following statements, 1. If it is a pleasant day you will do strawberry picking 2. If you are doing strawberry picking you are happy.

Above statements can be written in propositional logic like this 1. $\text{strawberryPicking} \leftarrow \text{pleasant}$ 2. $\text{happy} \leftarrow \text{strawberryPicking}$

And again these statements can be written in CNF like this 1. $(\text{strawberryPicking} \vee \neg \text{pleasant}) \wedge ¿¿$ 2. $(\text{happy} \vee \neg \text{strawberryPicking})$

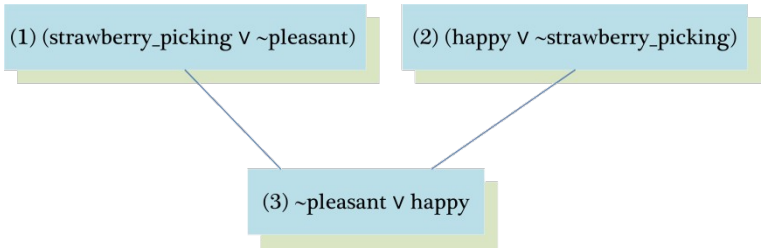
By resolving these two clauses and cancelling out the conflicting terms strawberryPicking and $\neg \text{strawberryPicking}$, we can have one new clause,

3. $\neg \text{pleasant} \vee \text{happy}$

When we write above new clause in infer or implies form, we have

$\text{pleasant} \rightarrow \text{happy}$ or $\text{happy} \leftarrow \text{pleasant}$

i.e. If it is a pleasant day you are happy.



But sometimes from the collection of the statements we have, we want to know the answer of this question - "Is it possible to prove some other statements from what we actually know?" In order to prove this we need to make some inferences and those other statements can be shown true using Refutation proof method i.e. proof by contradiction using Resolution. So for the asked goal we will negate the goal and will add it to the given statements to prove the contradiction.

Let's see an example to understand how Resolution and Refutation work. In below example

Part (I): English Sentences: represents the English meanings for the clauses

1. If it is sunny and warm day you will enjoy.
2. If it is warm and pleasant day you will do strawberry picking
3. If it is raining then no strawberry picking.
4. If it is raining you will get wet.
5. It is warm day
6. It is raining
7. It is sunny

Part (II): Propositional Statements: represents the propositional logic statements for given English sentences,

1. $\text{enjoy} \leftarrow \text{sunny} \wedge \text{warm}$
2. $\text{strawberryPicking} \leftarrow \text{warm} \wedge \text{pleasant}$

3. $\neg strawberryPicking \leftarrow raining$
4. $wet \leftarrow raining$
5. $warm$
6. $raining$
7. $sunny$

Part (III): CNF of Part (II) :) represents the Conjunctive Normal Form (CNF) of Part(II)

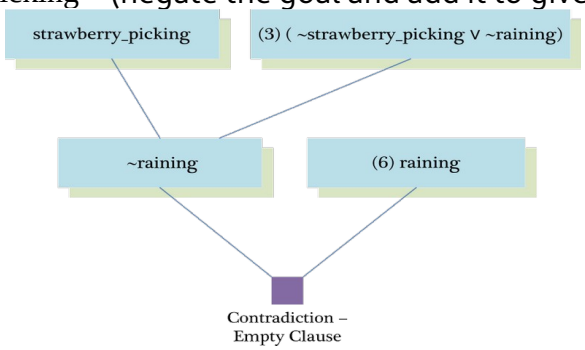
1. $(enjoy \vee \neg sunny \vee \neg warm) \wedge$
2. $(strawberryPicking \vee \neg warm \vee \neg pleasant) \wedge$
3. $(\neg strawberryPicking \vee \neg raining) \wedge$
4. $(wet \vee \neg raining) \wedge$
5. $(warm) \wedge$
6. $(raining) \wedge$
7. $(sunny)$

Part (IV): Other statements we want to prove by Refutation: shows some other statements we want to prove using Refutation proof method.

- (Goal 1) You are not doing strawberry picking.
- (Goal 2) You will enjoy.
- (Goal 3) Try it yourself: You will get wet.

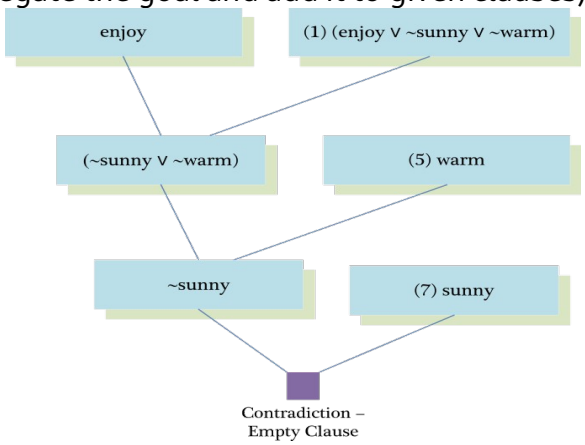
Goal 1: You are not doing strawberry picking.

1. Prove: $\neg strawberryPicking$
2. Assume: $strawberryPicking$ (negate the goal and add it to given clauses).



Goal 2: You will enjoy.

1. Prove: $enjoy$
2. Assume: $\neg enjoy$ (negate the goal and add it to given clauses)



Exercise 5 - Rusolution by Refutation

Resolution Theorem Prover

MODULE USED:

Module - pyprover

Installation - pip install pyprover

Usage

```
props(string) - for creating Prop()s and Pred()s
    Input : string containing names delimeterd by space
   Output : map object containing Prop() objects

    Usage - Likes, Boy, Womwn = props("Likes Boy Women")

terms(string)- for creating Const()s and Vars()s

    Input : string containing names delimeterd by space
   Oupput : map object containing Const() objects

    Usage - John, Mary, Rudo = props("John, Mary, Rudo")


Operator & - logical AND
Operator | - logical OR
Operator ~ - logical NOT
Operator >> - logical IMPLIES

FA or ForALL - Universal Quantifier
    Usage - FA(x,FA(y, A(x,y)>> B(x,y)))

TE or Exists - Existential Quantifier
    Usage - FA(x,TE(y, A(x,y) & B(x,y)))

expr(string) - convert mathematical notaion to logic expression
    Usage - expr("A x. Likes(x,carl)"

solve(expr) - Converts the expression to CNF and performs all possible resolutions
    Input : Expression
   Output : CNF expression of all resolutions

top - tautalogy
    assert solve(A | ~ A) == top
    assert simplify(A | ~ A) == top

bot - contradiction
    assert solve(A & ~ A) == bot
    assert simplify(A & ~ A) == bot
```

Exercise 5 - Rusolution by Refutation

Resolution Theorem Prover

AIM:

To write a python program for checking prime numbers using SymPy.

MODULE USED:

Module - pyprover

Installation - pip install pyprover

Usage

```
props(string) - for creating Prop()s and Pred()s
    Input : string containing names delimeterd by space
   Output : map object containing Prop() objects

    Usage - Likes, Boy, Womwn = props("Likes Boy Women")

terms(string)- for creating Const()s and Vars()s

    Input : string containing names delimeterd by space
   Oupput : map object containing Const() objects

    Usage - John, Mary, Rudo = props("John, Mary, Rudo")


Operator & - logical AND
Operator | - logical OR
Operator ~ - logical NOT
Operator >> - logical IMPLIES

FA or ForALL - Universal Quantifier
    Usage - FA(x,FA(y, A(x,y)>> B(x,y)))

TE or Exists - Existential Quantifier
    Usage - FA(x,TE(y, A(x,y) & B(x,y)))

expr(string) - convert mathematical notaion to logic expression
    Usage - expr("A x. Likes(x,carl)"

solve(expr) - Converts the expression to CNF and performs all possible resolutions
    Input : Expression
   Output : CNF expression of all resolutions

top - tautalogy
    assert solve(A | ~ A) == top
    assert simplify(A | ~ A) == top

bot - contradiction
```

```
    assert solve(A & ~ A) == bot
    assert simplify(A & ~ A) == bot
```

SOURCE CODE:

```
from pyprover import *
from pyprover.logic import Var

def vars (string) : return map(Var,string.split(" "))

def prove_by_refutation(givens,conclusion):
    if isinstance(givens, (list)):
        givens = And(*givens)
    # (permises & ~ conclusion) is contradiction
    return solve(givens & ~conclusion) == bot

def prove_theorems(givens, theorems, from_exprs= False):
    if from_exprs:
        givens = list(map(expr,givens))
        theorems = map(expr,theorems)
    for theorem in theorems:
        result = prove_by_refutation(givens,theorem)
        print(
            "The Given Facts prove the theorem,",
            theorem,"as",result
        )

def find_matching_solutions(givens,query,from_exprs = False):
    if from_exprs:
        givens = list(map(expr,givens))
    expr_ = solve(And(*givens))
    while isinstance(expr_,(FA,TE)):
        expr_ = expr_.elem
    matches = [
        elem
        for elem in expr_.elems
        if query.find_unification(elem)
    ]
    return matches


if __name__ == "__main__":
    (
        Child,Loves,Raindeer,HasRedNose,Wierd,Clown
    ) =props(
        "Child Loves Raindeer HasRedNose Wierd Clown"
    )
    Santa,Rudo,Carl= terms("Santa Rudo Carl")

    givens = [
        FA(x,Child(x)>>Loves(x,Santa)),
        FA(x,FA(y,(Loves(x,Santa) & Raindeer(y))>>Loves(x,y))),
        Raindeer(Rudo) & HasRedNose(Rudo),
        FA(x, HasRedNose(x) >> (Wierd(x) | Clown(x))),
        FA(x, Raindeer(x)>> ~Clown(x)),
```

```
        FA(x, Wierd(x) >> ~Loves(Carl,x))
    ]
    theorems=[
        ~Child(Carl),
        Loves(Carl,Santa),
    ]
    print("THEOREM PROVING:\n")
    prove_theorems(givens,theorems)

    givens = [
        "A x. A y. Child(x) & Candy(y) -> Loves(x,y)",
        "A x. E y. Candy(y) & Loves(x,y) -> ~NutritionFan(x)",
        "A x. A y. Pumpkin(y) & Eats(x,y) -> NutritionFan(x)",
        "A x. A y. Pumpkin(y) & Buys(x,y) -> Eats(x,y)",
        "E y. Buys(john,y) & Pumpkin(y)",
    ]

    theorems = [
        "E y. Eats(john,y) & Pumpkin(y)",
        "NutritionFan(john)",
        "Child(john)",

    ]

    print("\nTHEOREM PROVING USING THEOREM EXPRESSIONS:\n")
    prove_theorems(givens, theorems,True)

    SimpleSentence, = props("SimpleSentence")
    x1,z1,u1,v1  = vars("x1 z1 u1 v1")
    dog, = terms("dog")
    query = SimpleSentence(x1,dog,z1,u1,v1)
    givens = [
        '''\
A x. A y. A z. A u. A v. (
    Article(x)& Noun(y) & Verb(z) & Article(u) & Noun(v) -> SimpleSentence(x,y,z,u,v)
)''',
        "Article(a) & Article(the)",
        "Noun(man) & Noun(dog) ",
        "Verb(likes) & Verb(bites)",
    ]
    print("\nFINDING SOLUTIONS:\n")
    print("GIVENS:\n")
    for permise in givens: print(permise)
    print("\nQUERY:\n")
    print(query)
    solutions = find_matching_solutions(givens, query,True)
    print("\nSOLUTIONS:\n")
    for solution in solutions:print(*solution.args)
```

OUTPUT:

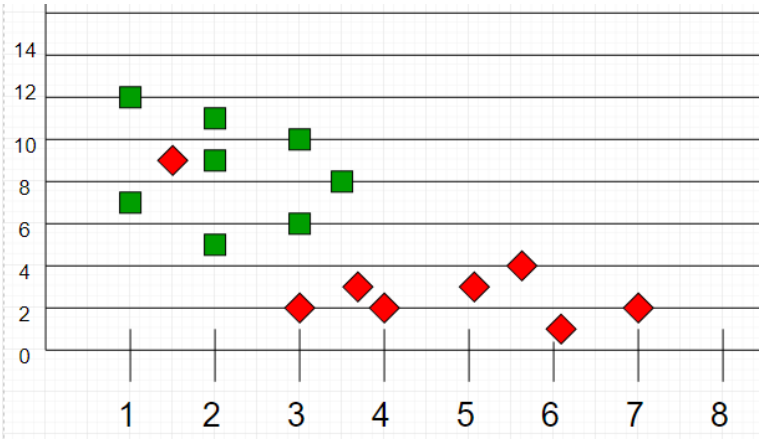
```
THEOREM PROVING:

The Given Facts prove the theorem, ~Child(Carl) as True
The Given Facts prove the theorem, Loves(Carl, Santa) as False
```

```
THEOREM PROVING USING THEOREM EXPRESSIONS:

The Given Facts prove the theorem, TE y, (Eats(john, y) & Pumpkin(y)) as True
The Given Facts prove the theorem, NutritionFan(john) as True
The Given Facts prove the theorem, Child(john) as False

FINDING SOLUTIONS:

GIVENS:

A x. A y. A z. A u. A v. (
    Article(x)& Noun(y) & Verb(z) & Article(u) & Noun(v) -> SimpleSentence(x,y,z,u,v)
)
Article(a) & Article(the)
Noun(man) & Noun(dog)
Verb(likes) & Verb(bites)

QUERY:

SimpleSentence(x1, dog, z1, u1, v1)

SOLUTIONS:

a dog likes the dog
a dog bites the dog
a dog likes a dog
a dog bites a dog
the dog likes the dog
the dog bites the dog
the dog likes a dog
the dog bites a dog
a dog likes the man
a dog bites the man
a dog likes a man
a dog bites a man
the dog likes the man
the dog bites the man
the dog likes a man
the dog bites a man
```

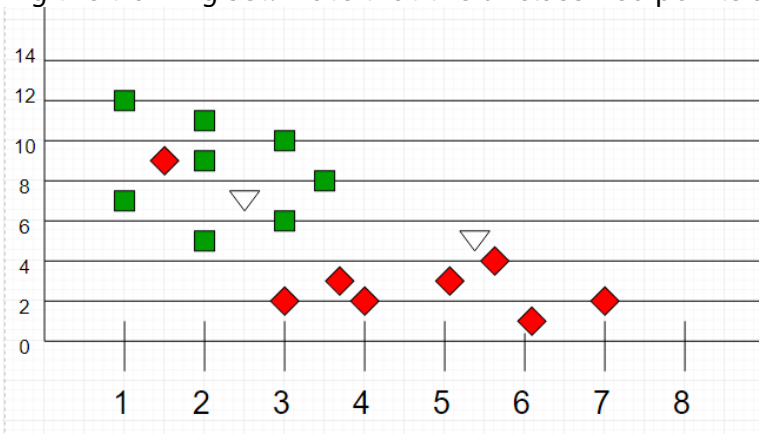
6. STATISTICAL REASONING - ‘k’ NEAREST NEIGHBOR

Statistical reasoning is the way people reason with statistical ideas and make sense of statistical information. Statistical reasoning may involve connecting one concept to another (e.g., center and spread) or may combine ideas about data and chance.

- *k*-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
- It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).
- We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.
- As an example, consider the following table of data points containing two features:



- Now, given another set of data points (also called testing data), allocate these points a group by analyzing the training set. Note that the unclassified points are marked as ‘White’.



Intuition:

- If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as ‘Red’ has a higher probability of getting classified as ‘Red’.

- Intuitively, we can see that the first point (2.5, 7) should be classified as 'Green' and the second point (5.5, 4.5) should be classified as 'Red'.

Algorithm:

Let m be the number of training data samples. Let p be an unknown point.

1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m:
3. Calculate Euclidean distance d(arr[i], p).
4. Make set S of K smallest distances obtained. Each of these distances corresponds to an already classified data point.
5. Return the majority label among S.

Exercise 6 - Statistical Reasoning - ‘k’ Nearest Neighbour

AIM:

To write a python program to implement the ‘k’ Nearest Neighbour algorithm.

ALGORITHM :

```
Algorithm euclidian_dist(p1,p2)
    Input : p1,p2 - points as Tuple()s
   Output : euclidian distance between the two points

    return sqrt(
        sum(
            List([(p1[i]-p2[i])^2 for i <- 0 to p1.length])
        )
    )
end Algorithm

Algorithm KNN_classify(dataset,k,p)
    Input : dataset – Dict() with class labels as keys
                      and data_points for the class as values.
            p - test point p(x,y),
            k - number of nearest neighbour.
   Output : predicted class of the test point

    dist=List([
        Tuple(euclidian_dist(test_point,data_point),class)
        for class in dataset
        for data_point in class
    ])
     dist = first k elements of sorted(dist,ascending)
     freqs = Dict(class:(freqency of class in dist) for class in data_set)
     return (class with max value in freqs)
end Algorithm
```

SOURCE CODE :

```
from math import sqrt
def euclidian_dist(p1,p2):
    return sqrt(
        sum([(x1-x2)**2 for (x1,x2) in zip(p1,p2)])
    )

class KNNClassifier:
    def __init__(self,data_set,k=3,dist=euclidian_dist):
        self.data_set = data_set
        self.k = k
        self.dist = dist

    def classify(self,test_point):
        distances = sorted([
```

```
            (self.dist(data_point,test_point),data_class)
            for data_class in self.data_set
            for data_point in self.data_set[data_class]
        ])[:k]
        freqs={data_class:0 for data_class in self.data_set}
        for (_,data_class) in distances:
            freqs[data_class]+=1
        return max(freqs,key = freqs.get)

if __name__ == "__main__":
    data_set = {
        "Class 1":{(1,12),(2,5),(3,6),(3,10),(3.5,8),(2,11),(2,9),(1,7)},
        "Class 2":{(5,3),(3,2),(1.5,9),(7,2),(6,1),(3.8,1),(5.6,4),(4,2),(2,5)}
    }
    test_points= [(2.5,7),(7,2.5)]
    classifier = KNNClassifier(data_set,3)
    for test_point in test_points:
        print(
            f"The given test point {test_point} is classified to:",
            classifier.classify(test_point)
        )
```

OUTPUT:

```
The given test point (2.5, 7) is classified to: Class 1
The given test point (7, 2.5) is classified to: Class 2
```

ALTERNATIVE METHOD USING NUMPY:

```
import numpy as np

def euclidian_dist_np(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2,axis=-1))

class KNNClassifier:
    def __init__(self,train_x,train_y,k=3,dist=euclidian_dist_np):
        self.train_x = train_x
        assert train_y.dtype == np.int, "Class labels should be integers"
        self.train_y = train_y
        self.k = k
        self.dist = dist

    def classify(self,test_point):
        k_nearest_classes = self.train_y[
            # indexes of k nearest neignbours
            np.argsort(self.dist(self.train_x,test_point))[:k]
        ]
        # maximum occuring class
        return np.bincount(k_nearest_classes).argmax()
if __name__ == "__main__":
    dataset = np.loadtxt("knn_dataset.csv",dtype=np.float,delimiter=",")
    train_x,train_y = dataset[:,:-1], dataset[:,-1].astype(np.int)
    test_x= np.array([[2.5,7],[7,2.5]])
    k = 3
    classifier = KNNClassifier(train_x,train_y,k=k)
    for test_vector in test_x:
```

```
        print(
            f"The given test point {test_vector} is classified to Class :",
            classifier.classify(test_vector)
        )
```

OUTPUT:

```
The given test point [2.5 7. ] is classified to Class : 1
The given test point [7.  2.5] is classified to Class : 2
```

7. UNCERTAINTY - NAIVE BAYES CLASSIFIER

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI: - When there are unpredictable outcomes. - When specifications or possibilities of predicates becomes too large to handle. - When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge: - Bayes' rule - Bayesian Statistics - Here we discuss the theory behind the Naive Bayes classifiers and their implementation. - Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. - To start with, let us consider a dataset. - Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for plaing golf. - Here is a tabular representation of our dataset.

S.No	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

- The dataset is divided into two parts, namely, feature matrix and the response vector.
- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of dependent features. In above dataset, features are 'Outlook', 'Temperature', 'Humidity' and 'Windy'.
- Response vector contains the value of class variable(prediction or output) for each row of feature matrix. In above dataset, the class variable name is 'Play golf'.

Assumption:

The fundamental Naive Bayes assumption is that each feature makes an: - independent - equal

Contribution to the outcome.

With relation to our dataset, this concept can be understood as:

- We assume that no pair of features are dependent. For example, the temperature being 'Hot' has nothing to do with the humidity or the outlook being 'Rainy' has no effect on the winds. Hence, the features are assumed to be **independent**.
- Secondly, each feature is given the same weight (or importance). For example, knowing only temperature and humidity alone can't predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing **equally** to the outcome.

Note: The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Now, before moving to the formula for Naive Bayes, it is important to know about Bayes' theorem.

Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

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

where A and B are events and P(B) ? 0.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X)=\frac{P(X|y)P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector (of size *n*) where:

$$X=(x_1,x_2,x_3,\ldots,x_n)$$

Just to clear, an example of a feature vector and corresponding class variable can be: (refer 1st row of dataset)

- X = (Rainy, Hot, High, False)

- y = No

So basically, $P(y|X)$ here means, the probability of "Not playing golf" given that the weather conditions are "Rainy outlook", "Temperature is hot", "high humidity" and "no wind".

Naive assumption:

Now, its time to put a naive assumption to the Bayes' theorem, which is, **independence** among the features. So now, we split **evidence** into the independent parts.

Now, if any two events A and B are independent, then,

$$P(A,B)=P(A)P(B)$$

Hence, we reach to the result:

$$P(y|x_1,\dots,x_n)=\frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

$$P(y|x_1,\dots,x_n)=\frac{P(y)\prod_{i=1}^{n}P(x_i|y)}{P(x_1)P(x_2)\dots P(x_n)}$$

which can be expressed as:

Now, as the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1,\dots,x_n)\propto P(y)\prod_{i=1}^{n}P(x_i|y)$$

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable *y* and pick up the output with maximum probability. This can be expressed mathematically as:

$$y=argmax_y(P(y)\prod_{i=1}^{n}P(x_i|y))$$

So, finally, we are left with the task of calculating $P(y)$ and $P(x_i|y)$.

- Please note that $P(y)$ is also called **class probability** and $P(x_i|y)$ is called **conditional probability**.
- The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i|y)$.
- Let us try to apply the above formula manually on our weather dataset. For this, we need to do some pre computations on our dataset.

- We need to find $P(x_i|y_j)$ for each x_i in X and y_j in y. All these calculations have been demonstrated in the tables below:

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Dataset

So, in the figure above, we have calculated $P(x_i|y_j)$ for each x_i in X and y_j in y manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e $P(\text{temp}=\text{cool}|\text{playGolf}=\text{Yes})=3/9$.

Also, we need to find class probabilities $P(y)$ which has been calculated in the table 5. For example, $P(\text{playGolf}=\text{Yes})=9/14$.

So now, we are done with our pre-computations and the classifier is ready!

Let us test it on a new set of features (let us call it today):

- today = (Sunny, Hot, Normal, False)

So, probability of playing golf is given by:

$$P(\text{Yes}|\text{today})=\frac{P(\text{Sunny}|\text{Yes})P(\text{Hot}|\text{Yes})P(\text{NormalHumid}|\text{Yes})P(\text{NoWind}|\text{Yes})}{P(\text{today})}$$

and probability to not play golf is given by:

$$P(\text{No}|\text{today})=\frac{P(\text{Sunny}|\text{No})P(\text{Hot}|\text{No})P(\text{NormalHumid}|\text{No})P(\text{NoWind}|\text{No})}{P(\text{today})}$$

Since, $P(\text{today})$ is common in both probabilities, we can ignore $P(\text{today})$ and find proportional probabilities as:

$$P(\text{Yes}|\text{today})\propto\frac{2}{9}.\frac{2}{9}.\frac{6}{9}.\frac{6}{9}.\frac{9}{14}\approx 0.0141;P(\text{No}|\text{today})\propto\frac{3}{5}.\frac{2}{5}.\frac{1}{5}.\frac{2}{5}.\frac{5}{14}\approx 0.0068$$

Now, since

$$P(\text{Yes}|\text{today})+P(\text{No}|\text{today})=1$$

These numbers can be converted into a probability by making the sum equal to 1 (normalization):

$$P(\text{Yes}|\text{today})=\frac{0.0141}{0.0141+0.0068}=0.67;P(\text{No}|\text{today})=\frac{0.0068}{0.0141+0.0068}=0.33$$

Since

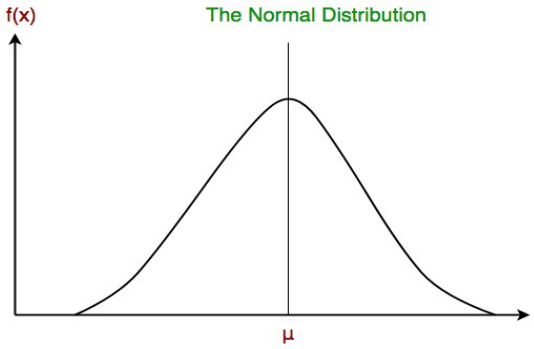
$$P(\text{Yes}|\text{today}>P(\text{No}|\text{today})$$

So, prediction that golf would be played is 'Yes'.

The method that we discussed above is applicable for discrete data. In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i|y)$. Now, we discuss one of such classifiers here.

Gaussian Naive Bayes classifier:

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:



Gaussian Distribution

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is

$$P(x_i|y)=\frac{1}{\sqrt{2\pi\sigma_y^2}}exp\left(\frac{-(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

given by:

Let's implement Gaussian naive bayes classifier in python.

Exercise 7 - Naive Bayes Classifier

Gaussian Naive Bayes Classifier

AIM

To write a python program to implement Naive Bayes Classifier.

ALGORITHM:

Function GaussianProb(x, μ , σ)

$$\text{GaussianProb}(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}*e^{-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

```
Algorithm MakeModel(train_set)
    Input : train_set - a 2D list as table containing containing
                        response vector as the last column and
                        other columns as feature matrix
   Output : model - a Dict() with classes as keys and
                    List(Tuple()s) containing Tuples of (mean,stdev) for
                    each feature of the records in the train_set
                    belonging to that class as values

    classes = Set(record[-1] for record in train_set)
    class_separated_feature_matrices = Dict(
        class : List(
            record[:-1]
            for record in train_set
            if record[-1] == class
        )
        for class in classes
    )
    model = Dict(
        class : List(
            Tuple(Mean(feature_col),StDev(feature_col))
            for feature_col in columns(feature_matrix)
        )
        for (class,feature_matrix) in class_separated_feature_matrix
    )
    return model
End Algorithm

Algorithm classify(model,test_vector)
    Input : model - the gaussian Naive Bayes model
            test_vector - the feature vector for the test case as Tuple()
   Output : class - The predicted class of the test_vector

    classProb = Dict(
        class : Product(
            List(
                gaussianProb(test_feature,mean,stdev)
```

```
                for test_feature,(mean,stdev) in zip(test_vector,model[class])
            )
        )
        for class in model
    )
    return argmax(classProb)
End Algorithm
```

DATASET USED:

pima-indian-diabetes.csv - contains a table with each column representing features collected from the test subjects and the last column is the response vector consists of the class as 0(diabetes Negative) or 1(diabetes Positive) The file consists of 769 rows(including header row) and 9 columns. Each row consists of the features extracted from each test subjects

Each column represent a feature as follows

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

The 9th or the last column represents the response vector as having diabetes(1) or not(0).

SOURCE CODE:

```
import statistics as st
import csv
import math

def load_csv(filename,header = False):
    table = csv.reader(open(filename, "r"));
    if header :
        table.__next__()
    dataset = [
        [float(item) for item in record]
        for record in table
    ]
    return dataset

def gaussian_prob(x, mean, stdev):
    return math.exp(
        -(
            math.pow(x-mean,2)
            /(2*math.pow(stdev,2))
         )
    )/(math.sqrt(2*math.pi)*stdev)

def columns(array): return zip(*array)
def argmax(dict_): return  max(dict_,key= dict_.get)
```

```
class GNBClassifier:
    def __init__(self,training_data):
        self.model = GNBClassifier.make_model(training_data)

    @staticmethod
    def make_model(training_data):
        classes = {record[-1] for record in training_data}
        class_separated_feature_matrices = {
            class_:[
                record[:-1]
                for record in training_data
                if record[-1] == class_
            ]
            for class_ in classes
        }
        class_seperated_feature_mean_and_stdevs = {
            class_:[
                (st.mean(feature_col),st.stdev(feature_col))
                for feature_col in columns(feature_matrix)
            ]
            for class_,feature_matrix in class_separated_feature_matrices.items()
        }
        return class_seperated_feature_mean_and_stdevs

    def classify(self,test_vector):
        class_probs = {
            class_:math.prod([
                gaussian_prob(test_feature,mean,stdev)
                for test_feature,(mean,stdev) in zip(test_vector,self.model[class_])
            ])
            for class_ in self.model
        }
        return argmax(class_probs)

    def compute_accuracy(self,testing_data):
        result = [
            original_class==self.classify(test_vector)
            for *test_vector,original_class in testing_data
        ]
        return result.count(True)/len(result)

if __name__ == "__main__":
    dataset = load_csv("pima-indians-diabetes.csv",True)
    split_ratio = .75
    split_length = int(len(dataset)*split_ratio)
    train = dataset[:split_length]
    test = dataset[split_length:]
    print('The length of the training set',len(train))
    print('The length of the testing set',len(test))
    diabetes_predictor = GNBClassifier(train)
    print(
        "\nThe accuracy of predictions for this classifier is:",
        diabetes_predictor.compute_accuracy(test)
    )
```

```
    print("\nPredicting diabetes for few patients from the testing set :")
    for *test_vector,_ in test[:5]:
        print(
            "\nThe patitent with features : ",
            test_vector,
            "is predicted for diabetes as:",
            "POSITIVE" if diabetes_predictor.classify(test_vector) else "NEGATIVE"
            ,sep="\n"
        )
```

OUTPUT:

```
The length of the training set 576
The length of the testing set 192

The accuracy of predictions for this classifier is: 0.7760416666666666

Predicting diabetes for few patients from the testing set :

The patitent with features :
[6.0, 108.0, 44.0, 20.0, 130.0, 24.0, 0.813, 35.0]
is predicted for diabetes as:
NEGATIVE

The patitent with features :
[2.0, 118.0, 80.0, 0.0, 0.0, 42.9, 0.693, 21.0]
is predicted for diabetes as:
NEGATIVE

The patitent with features :
[10.0, 133.0, 68.0, 0.0, 0.0, 27.0, 0.245, 36.0]
is predicted for diabetes as:
POSITIVE

The patitent with features :
[2.0, 197.0, 70.0, 99.0, 0.0, 34.7, 0.575, 62.0]
is predicted for diabetes as:
POSITIVE

The patitent with features :
[0.0, 151.0, 90.0, 46.0, 0.0, 42.1, 0.371, 21.0]
is predicted for diabetes as:
POSITIVE
```

ALTERNATIVE METHOD USING NUMPY

```
import numpy as np

def gaussian_prob(x,mean,stdev):
    x = x.reshape(-1,1,x.shape[-1])
    return np.exp(
        -(np.power(x-mean,2)/(2*np.power(stdev,2)))
    )/(np.sqrt(2*np.pi)*stdev)

class GNBClassifier:
    def __init__(self,train_x,train_y):
```

```
        classes = np.unique(train_y)
        class_seperated_train_x =[train_x[train_y==class_] for class_ in classes]
        self.means = np.array([d.mean(axis=0) for d in class_seperated_train_x])
        self.stdevs = np.array([d.std(axis=0) for d in class_seperated_train_x])

    def classify(self,test_x):
        return np.argmax(
            np.prod(gaussian_prob(test_x,self.means,self.stdevs),axis = -1),axis=-1
        )

    def compute_accuracy(self,test_x,test_y):
        return np.count_nonzero(self.classify(test_x) == test_y)/test_y.shape[0]

if __name__ == "__main__":
    dataset = np.loadtxt("pima-indians-diabetes.csv",delimiter=",",skiprows=1)
    split_ratio = .75
    split_length = int(len(dataset)*split_ratio)
    train = dataset[:split_length]
    train_x,train_y = train[:,:-1],train[:,-1]
    test = dataset[split_length:]
    test_x,test_y = test[:,:-1],test[:,-1]

    print('The length of the training set',len(train))
    print('The length of the testing set',len(test))
    diabetes_predictor = GNBClassifier(train_x,train_y)
    print(
        "\nThe accuracy of predictions for this classifier is:",
        diabetes_predictor.compute_accuracy(test_x,test_y)
    )

    print("\nPredicting diabetes for few patients from the testing set :")
    for  test_vector in test_x[:5]:
        print(
            "\nThe patitent with features : ",test_vector.,
            "is predicted for diabetes as:",
            "POSITIVE" if diabetes_predictor.classify(test_vector) else "NEGATIVE"
            ,sep="\n"
        )
```

OUTPUT:

```
The length of the training set 576
The length of the testing set 192

The accuracy of predictions for this classifier is: 0.7760416666666666

Predicting diabetes for few patients from the testing set :

The patitent with features :
[  6.    108.     44.     20.    130.     24.      0.813  35.   ]
is predicted for diabetes as:
NEGATIVE

The patitent with features :
[  2.    118.     80.      0.      0.     42.9     0.693  21.   ]
```

```
is predicted for diabetes as:
NEGATIVE

The patitent with features :
[ 10.    133.     68.      0.      0.     27.      0.245  36.   ]
is predicted for diabetes as:
POSITIVE

The patitent with features :
[  2.    197.     70.     99.      0.     34.7     0.575  62.   ]
is predicted for diabetes as:
POSITIVE

The patitent with features :
[  0.    151.     90.     46.      0.     42.1     0.371  21.   ]
is predicted for diabetes as:
POSITIVE
```

8. GAME PLAYING

GAME PLAYING IN ARTIFICIAL INTELLIGENCE

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS (Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need other search procedures that improve

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

The most common search technique in game playing is Minimax search procedure. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Minimax algorithm uses two functions:

- **MOVEGEN:** It generates all the possible moves that can be generated from the current position.
- **STATICEVALUATION:** It returns a value depending upon the goodness from the viewpoint of two-player

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

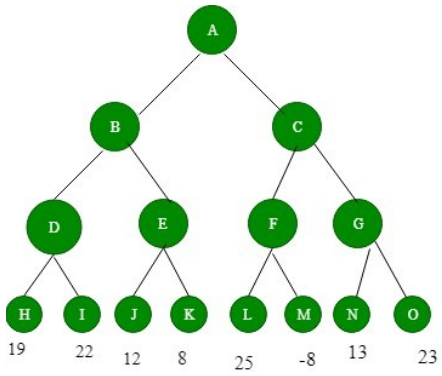


Figure 1: Before backing-up of values

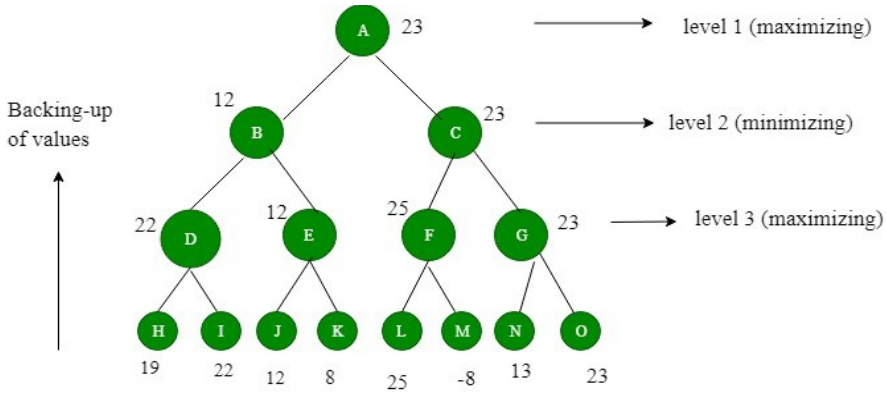


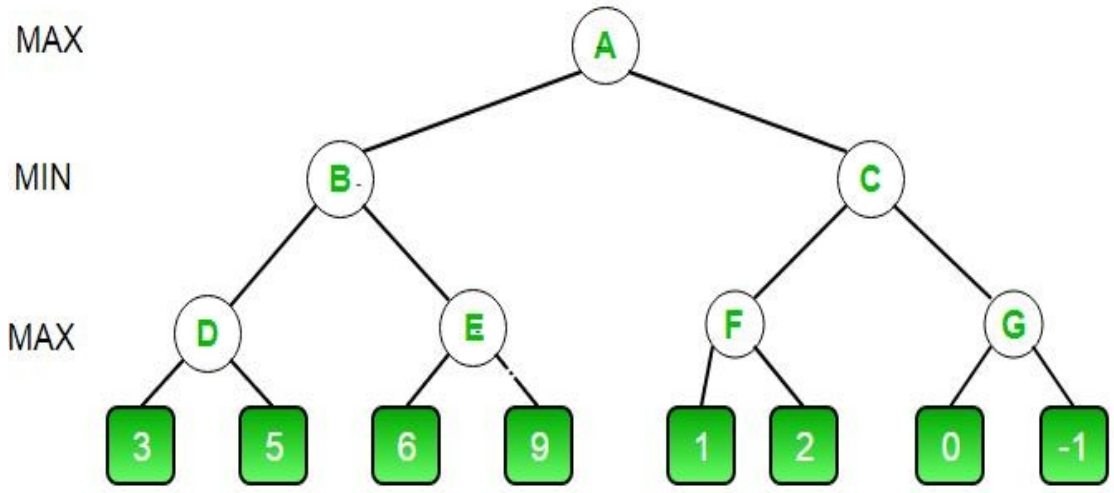
Figure 2: After backing-up of values

We assume that Maximizer will start the game. 4 levels are generated. The value to nodes H, I, J, K, L, M, N, O is provided by STATICEVALUATION function. Level 3 is maximizing level, so all nodes of level 3 will take maximum values of their children. Level 2 is minimizing level, so all its nodes will take minimum values of their children. This process continues. The value of A is 23. That means A should choose C move to win.

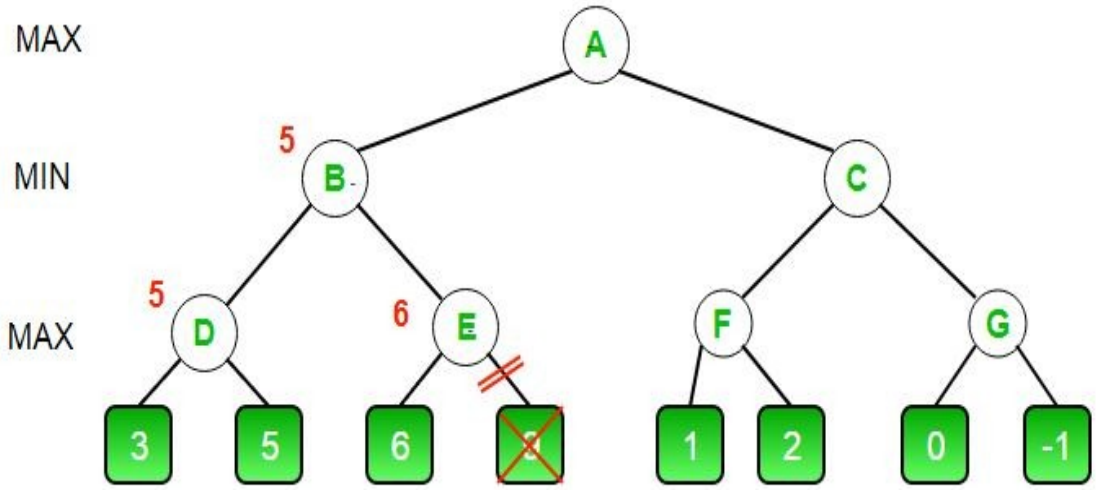
Alpha - Beta pruning

Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

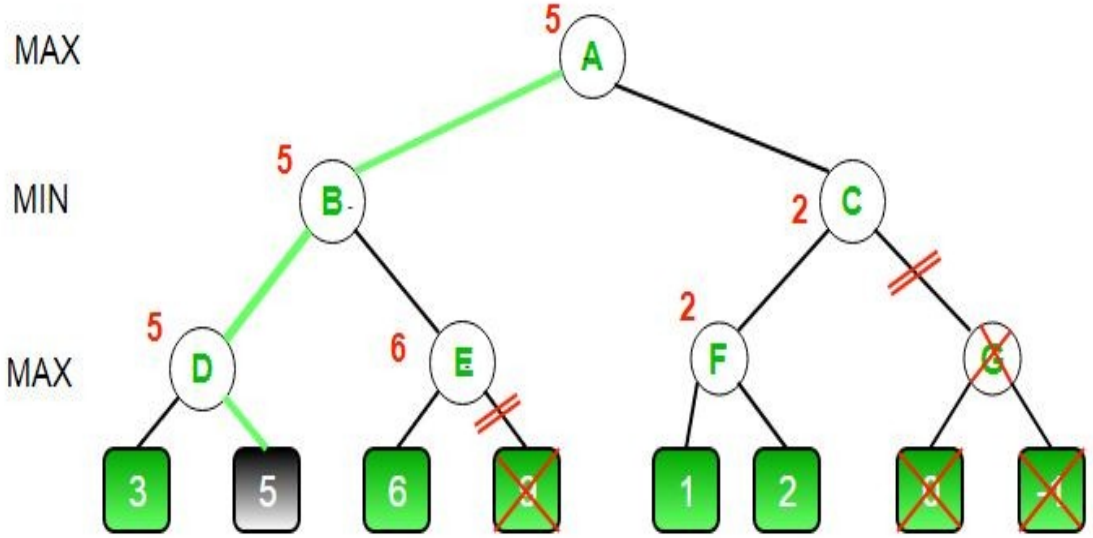
Let's define the parameters alpha and beta. **Alpha** is the best value that the **maximizer** currently can guarantee at that level or above. **Beta** is the best value that the **minimizer** currently can guarantee at that level or above.



Initial game tree



From Minimizer level cut off occur at Maximizer level



From Maximizer level cut off occur at Minimizer level

Exercise 8 - Game Playing

a) MIN-MAX ALGORITHM

AIM:

To write a python program to implement minimax algorithm.

ALGORITHM:

```
Algorithm MiniMax(node, depth ,maxTurn ,scores ,treeDepth)
    Input : node - current node
            depth - depth of the current ply
            maxTurn - Bool() whether it is maximizer's ply
            scores - List() containing scores of the last ply
            treeDepth - The depth of the game tree
   Output : best value for the player

    if depth = treeDepth
        return scores[node]
    if maxTurn
        return max(
            minimax(child_node, depth +1, False, scores,treeDepth)
            for child_node in childeren(node)
        )
    else
        return min(
            minimax(child_node, depth +1, True, scores,treeDepth)
            for child_node in childeren(node)
        )
end Algorithm
```

SOURCE CODE:

```
import math

def minimax (node,depth, max_turn, scores, target_depth):
    if (depth == target_depth):
        return scores[node]
    if max_turn:
        return max(
            minimax(node * 2,depth + 1,False, scores, target_depth),
            minimax(node * 2 + 1, depth + 1,False, scores, target_depth)
        )
    else:
        return min(
            minimax(node * 2, depth + 1,True, scores, target_depth),
            minimax(node * 2 + 1, depth + 1,True, scores, target_depth)
        )

if __name__ == "__main__":
    scores = [3, 5, 2, 9, 12, 5, 23, 23]
    tree_depth = math.log(len(scores), 2)
    print("The optimal value is :",
```

```
        minimax(0, 0, True, scores, tree_depth)
    )
```

OUTPUT:

```
The optimal value is : 12
```

b) ALPHA-BETA PRUNING

AIM:

To write a python program to implement minimax algorithm with alpha-beta pruning.

ALGORITHM:

```
Algorithm MiniMax(node, depth ,maxTurn ,scores ,treeDepth, alpha, beta)
    Input : node - current node
            depth - depth of the current ply
            maxTurn - Bool() whether it is maximizer's ply
            scores - List() containing scores of the last ply
            treeDepth - The depth of the game tree
            alpha - best value along the path of the maximizer
            beta - best value along the path of the minimizer

   Output : best value for the player

    if depth = treeDepth
        return scores[node]
    if maxTurn
        best <- -INFINITY
        for child_node in children(node)
            best <- max(
                minimax(child_node, depth +1, False, scores,treeDepth,alpha,beta),
                best
            )
            alpha <- max( alpha, best)
            if beta <= alpha
                break
        return best
    else
        best <- INFINITY
        for child_node in children(node)
            best <- min(
                minimax(child_node, depth+1, True, scores,treeDepth,alpha,beta),
                best
            )
            beta <- min( beta , best)
            if beta <= alpha:
                break
        return best
end Algorithm
```

SOURCE CODE:

```
import math
MAX, MIN = 1000, -1000

def minimax(node, depth, max_turn ,scores,tree_depth, alpha, beta):
    if depth == tree_depth:
        return scores[node]
    if max_turn:
        best = MIN
```

```
        for i in range(0, 2):
            val = minimax(node * 2 + i, depth + 1, False, scores, tree_depth, alpha,
beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = MAX
        for i in range(0, 2):
            val = minimax(node * 2 + i, depth + 1, True, scores, tree_depth, alpha,
beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                    break
        return best
if __name__ == "__main__":
    scores = [3, 5, 6, 9, 1, 2, 0, -1]
    tree_depth = math.log(len(scores), 2)
    print("The optimal value (using alpha-beta pruning) is :",
        minimax(0, 0, True, scores, tree_depth,MIN,MAX)
    )
```

OUTPUT:

```
The optimal value (using alpha-beta pruning) is : 5
```

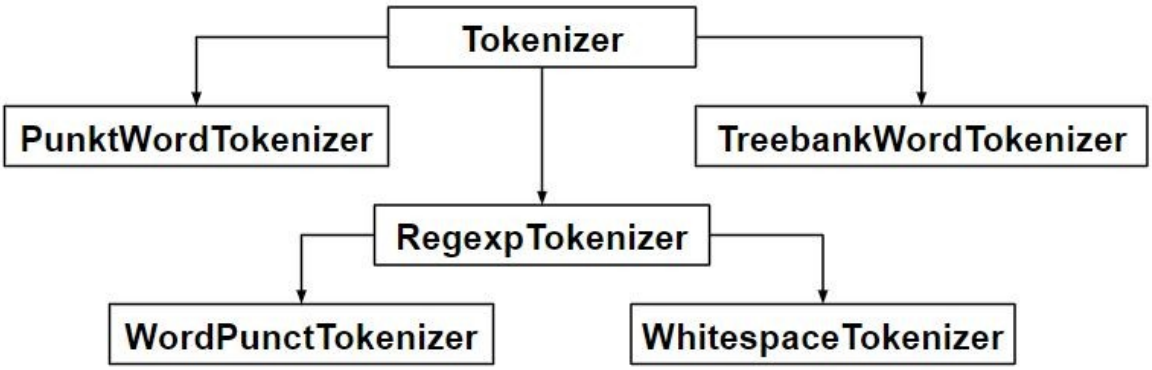
9. NATURAL LANGUAGE PROCESSING

NLP : TOKENIZATION, STEMMING USING NLTK :

Natural Language Processing (NLP) is a subfield of computer science, artificial intelligence, information engineering, and human-computer interaction. This field focuses on how to program computers to process and analyze large amounts of natural language data. It is difficult to perform as the process of reading and understanding languages is far more complex than it seems at first glance.

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

- Text into sentences tokenization
- Sentences into words tokenization
- Sentences using regular expressions tokenization



Sentence Tokenization - Splitting sentences in the paragraph

Example 1

```
from nltk.tokenize import sent_tokenize
text = "Hello everyone. Welcome to GeeksforGeeks. You are studying NLP article"
sent_tokenize(text)
```

Output:

```
['Hello everyone.',
'Welcome to GeeksforGeeks.',
'You are studying NLP article']
```

Punkt Sentence Tokenizer

When we have huge chunks of data then it is efficient to use it.

Example 2

```
import nltk.data
# Loading PunktSentenceTokenizer using English pickle file
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
tokenizer.tokenize(text)
```

Output:

```
['Hello everyone.',
'Welcome to GeeksforGeeks.',
'You are studying NLP article']
```

Tokenize sentence of different language

One can also tokenize sentence from different languages using different pickle file other than English.

Example 3

```
import nltk.data
spanish_tokenizer =
nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
text = 'Hola amigo. Estoy bien.'
spanish_tokenizer.tokenize(text)
```

Output

```
['Hola amigo.',
'Estoy bien.']
```

Word Tokenization - Splitting words in a sentence.

Example 4

```
from nltk.tokenize import word_tokenize
text = "Hello everyone. Welcome to GeeksforGeeks."
word_tokenize(text)
```

Output:

```
['Hello', 'everyone', '.', 'Welcome', 'to',
'GeeksforGeeks', '.']
```

Treebank Word Tokenizer

Example 5

```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

Output:

```
['Hello', 'everyone.', 'Welcome', 'to', 'GeeksforGeeks','.']
```

Punkt Word Tokenizer

It doesn't separates the punctuation from the words.

Example 6

```
from nltk.tokenize import PunktWordTokenizer
tokenizer = PunktWordTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

Output:

```
['Let', "'s", 'see', 'how', 'it', "'s", 'working', '.']
```

Word Punct Tokenizer

It separates the punctuation from the words.

Example 7

```
from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

Output:

```
['Let', "'", 's', 'see', 'how', 'it', "'", 's', 'working', '.']
```

Using Regular Expression

Example 8

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer("[w']+")
text = "Let's see how it's working."
tokenizer.tokenize(text)
```

Output:

```
["Let's", 'see', 'how', "it's", 'working']
```

Example 9

```
from nltk.tokenize import regexp_tokenize
text = "Let's see how it's working."
regexp_tokenize(text, "[w']+")
```

Output:

```
["Let's", 'see', 'how', "it's", 'working']
```

Stemming Words with NLTK

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

Some more example of stemming for root word “like” include: - likes - liked - likely - liking

Errors in Stemming:

There are mainly two errors in stemming **Overstemming** and **Understemming**. Overstemming occurs when two words are stemmed to same root that are of different stems. Under-stemming occurs when two words are stemmed to same root that are not of different stems.

Applications of stemming are:

- Stemming is used in information retrieval systems like search engines.
- It is used to determine domain vocabularies in domain analysis.

Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.

SPELL CHECKING USING pyspellchecker:

This library is based on Peter Norvig's implementation.

Installation - pip install pyspellchecker

Exercise 9 - Natural Language Processing

a)Tokenization and stemming using NLTK

AIM :

To write a python program to Tokenize and stem words or sentences using NLTK

ABOUT THE MODULE:

Module - nltk

Installation - pip install nltk

Stemming is the process of producing morphological variants of a root/base word

Class PorterStemmer

PorterStemmer is an abstract base class from which the porter stemmer function is derived. Porter stemmer is one of the most gentle stemmers.

Usage:

```
porter = PorterStemmer()
porter.stem("string")
```

Class LancasterStemmer

LancasterStemmer is an abstract base class from which the lancaster stemmer function is derived. Lancaster stemmer is an aggresive stemmer.

Usage:

```
lancaster =LancasterStemmer()
lancaster.stem("string")
```

Functions:

word_tokenize(string) - List() of word in the given string

wordpunct_tokenize(string) - Same as word_tokenize but also splits punctuations marks like " ' , : etc..

sent_tokenize(string) - List() of sencences in the given string

SOURCE CODE :

```
from nltk import (
    PorterStemmer, LancasterStemmer,
    word_tokenize, wordpunct_tokenize, sent_tokenize
)
import yaml

porter = PorterStemmer()
lancaster = LancasterStemmer()
```

```
line = "\n"+"="*80+"\n"

def stem_passage(passage,punct_tokenize = False):
    return " ".join(
        stem_sentence(sentence,punct_tokenize)
        for sentence in sent_tokenize(passage)
    )

def stem_sentence(sentence,punct_tokenize=False):
    tokenizer_func = wordpunct_tokenize if punct_tokenize else word_tokenize
    return " ".join(
        porter.stem(word)
        for word in tokenizer_func(sentence)
    )

if __name__ == "__main__":
    with open("stemming.yaml") as f:
        stemming_words, stemming_passage = yaml.full_load(f).values()
    print(
        "STEMMING WORDS:\n",
        "STEMMER COMPARISION:\n",sep = "\n"
    )
    print("-"*64)
    print(f"|{'WORD':^20}|{'PORTER STEMMER':^20}|{'LANCASTER STEMMER':^20}|")
    print("-"*64)
    for word in stemming_words:
        print(f"|{word:^20}|{porter.stem(word):^20}|{lancaster.stem(word):^20}|")
    print("-"*64)
    print(line)

    print("STEMMING PASSAGES:\n")

    print(
        "ORIGINAL PASSAGE :\n",
        stemming_passage,
        "\nSTEMMED PASSAGE :\n",
        stem_passage(stemming_passage),
        "\nSTEMMED PASSAGE (using wordpunct_tokenizer) :\n",
        stem_passage(stemming_passage,True),sep="\n"
    )
    print(line)
```

OUTPUT:

```
STEMMING WORDS:

STEMMER COMPARISION:

----------------------------------------------------------------
|        WORD        |   PORTER STEMMER   | LANCASTER STEMMER  |
----------------------------------------------------------------
|        cats        |        cat         |        cat         |
|      trouble       |       troubl       |       troubl       |
|     troubling      |       troubl       |       troubl       |
```

```
|      troubled      |       troubl       |       troubl       |
|       friend       |       friend       |       friend       |
|      friends       |       friend       |       friend       |
|     friendship     |     friendship     |       friend       |
|    friendships     |     friendship     |       friend       |
|       stabil       |       stabil       |       stabl        |
|    destabilize     |      destabil      |        dest        |
|  misunderstanding  |   misunderstand    |   misunderstand    |
|      railroad      |      railroad      |      railroad      |
|     moonlight      |     moonlight      |     moonlight      |
|      football      |      footbal       |      footbal       |
----------------------------------------------------------------


================================================================================


STEMMING PASSAGES:


ORIGINAL PASSAGE :


Pythoners are very intelligent and work very pythonly and now they are pythoning their
way to success. 'cause python's a very intelligent language. The Internet of things
(IoT) is a system of interrelated computing devices, mechanical and digital machines
provided with unique identifiers (UIDs) and  the ability to transfer data over a
network without requiring human-to-human  or human-to-computer interaction.


STEMMED PASSAGE :


python are veri intellig and work veri pythonli and now they are python their way to
success . 'caus python 's a veri intellig languag . the internet of thing ( iot ) is a
system of interrel comput devic , mechan and digit machin provid with uniqu identifi (
uid ) and the abil to transfer data over a network without requir human-to-human or
human-to-comput interact .


STEMMED PASSAGE (using wordpunct_tokenizer) :


python are veri intellig and work veri pythonli and now they are python their way to
success . ' caus python ' s a veri intellig languag . the internet of thing ( iot ) is
a system of interrel comput devic , mechan and digit machin provid with uniqu identifi
( uid ) and the abil to transfer data over a network without requir human - to - human
or human - to - comput interact .


================================================================================
```

b) Spell Checking using Pyspellchecker

AIM :

To write a python program to check the spelling of a word using pyspellchecker

ABOUT THE MODULE :

Module - pyspellchecker ***Installation*** - pip intall pyspellchecker

class SpellChecker

The object of SpellChecker class has various function like correcting and finding similar words from a String.

Usage:

```
spell= SpellChecker()
spell.correction(word)
```

Functions

SpellChecker().correction(word) - The grammatically corrected string for the given word.

SpellChecker().candidates(word) - The set of words similar to the given word.

SpellChecker().unknown(words) - The subset of misspelled words from the given words.

SpellChecker().word_probability(word) - The probability of the word being the desired, correct word

SOURCE CODE :

```
from spellchecker import SpellChecker
from nltk import word_tokenize
import yaml
spell = SpellChecker()

line = "\n"+"="*80+"\n"

if __name__ == "__main__":
    with open("spell_checking.yaml") as f:
        sentence,spell_check_words = yaml.full_load(f).values()
    words = word_tokenize(sentence)
    misspelled_words = spell.unknown(words)
    correct_sentence = " ".join(spell.correction(word) for word in words)
    print(
        "SPELL CORRECTION:",
        "\nORIGINAL SENTENCE:\n",
        sentence,
        "\nWORDS:\n",
        ", ".join(words),
        "\nMISSPELLED WORDS:\n",
        ", ".join(misspelled_words),
        "\nCORRECTED SENTENCE:\n",
        correct_sentence,sep="\n"
```

```
    )
    print(line)
    print("SPELL CHECKER TESTS:")

    print("-"*80)
    print(f"|{'WORD':^13}|{'PROBABILITY':^15}|{'CANDIDATES':^32}|{'CORRECTION':^15}|")
    print("-"*80)
    for word in spell_check_words:
        print(
            f"|{word:^13}"
            f"|{spell.word_probability(word):^15.3}"
            f"|{', '.join(spell.candidates(word)):^32}"
            f"|{spell.correction(word):^15}|"
        )
    print("-"*80)
    print(line)
```

OUTPUT:

```
SPELL CORRECTION:

ORIGINAL SENTENCE:

someting is happenning hete. Do yuo knw wht?

WORDS:

someting, is, happenning, hete, ., Do, yuo, knw, wht, ?

MISSPELLED WORDS:

someting, hete, happenning, knw, wht, yuo

CORRECTED SENTENCE:

something is happening here . Do you know what ?

================================================================================

SPELL CHECKER TESTS:
--------------------------------------------------------------------------------
|    WORD     |  PROBABILITY  |           CANDIDATES           |  CORRECTION   |
--------------------------------------------------------------------------------
|  calandar   |      0.0      |            calendar            |   calendar    |
| lightening  |   8.28e-07    |           lightening           |  lightening   |
|   misspel   |      0.0      |            misspelt            |   misspelt    |
|  necessary  |   0.000187    |           necessary            |   necessary   |
|  bussiness  |      0.0      | bossiness, fussiness, business |   business    |
|   recieve   |      0.0      |        relieve, receive        |    receive    |
|   adress    |      0.0      |    andress, dress, address     |    address    |
--------------------------------------------------------------------------------

================================================================================
```

10. EXPERT SYSTEM

Expert Systems

Artificial Intelligence is a piece of software that simulates the behavior and judgment of a human or an organization that has experts in a particular domain is known as an expert system. It does by acquiring relevant knowledge from its knowledge base and interpreting it according to the user's problem. The data in the knowledge base is added by humans that are expert in a particular domain and this software is used by a non-expert user to acquire some information. It is widely used in many areas such as medical diagnosis, accounting, coding, games etc.
An expert system is an AI software that uses knowledge stored in a knowledge base to solve problems that would usually require a human expert thus preserving a human expert's knowledge in its knowledge base. They can advise users as well as provide explanations to them about how they reached a particular conclusion or advice.

Examples: There are many examples of expert system. Some of them are given below:

- **MYCIN:** One of the earliest expert systems based on backward chaining. It can identify various bacteria that can cause severe infections and can also recommend drugs based on the person's weight.
- **DENDRAL:** It was an artificial intelligence based expert system used for chemical analysis. It used a substance's spectrographic data to predict it's molecular structure.
- **R1/XCON:** It could select specific software to generate a computer system wished by the user.
- **PXDES:** It could easily determine the type and the degree of lung cancer in a patient based on the data.
- **CaDet:** It is a clinical support system that could identify cancer in its early stages in patients.
- **DXplain:** It was also a clinical support system that could suggest a variety of diseases based on the findings of the doctor.

Components of an expert system:

- **Knowledge base:** The knowledge base represents facts and rules. It consists of knowledge in a particular domain as well as rules to solve a problem, procedures and intrinsic data relevant to the domain.
- **Inference engine:** The function of the inference engine is to fetch the relevant knowledge from the knowledge base, interpret it and to find a solution relevant to the user's problem. The inference engine acquires the rules from its knowledge base and applies them to the known facts to infer new facts. Inference engines can also include an explanation and debugging abilities.
- **Knowledge acquisition and learning module:** The function of this component is to allow the expert system to acquire more and more knowledge from various sources and store it in the knowledge base.
- **User interface:** This module makes it possible for a non-expert user to interact with the expert system and find a solution to the problem.

- **Explanation module:** This module helps the expert system to give the user an explanation about how the expert system reached a particular conclusion.

Characteristics of an expert system:

- Human experts are perishable but an expert system is permanent.
- It helps to distribute the expertise of a human.
- One expert system may contain knowledge from more than one human experts thus making the solutions more efficient.
- It decreases the cost of consulting an expert for various domains such as medical diagnosis.
- They use a knowledge base and inference engine.
- Expert systems can solve complex problems by deducing new facts through existing facts of knowledge, represented mostly as if-then rules rather than through conventional procedural code.
- Expert systems were among the first truly successful forms of artificial intelligence (AI) software.

Limitations:

- Don't have human-like decision making power.
- Can't possess human capabilities.
- Can't produce correct result from less amount of knowledge.
- Requires excessive training.

Advantages:

- Low accessibility cost.
- Fast response.
- Not affected by emotions unlike humans.
- Low error rate.
- *Capable of explaining how they reached a solution.*

Disadvantages:

- Expert system has no emotions.
- Common sense is the main issue of the expert system.
- It is developed for a specific domain.
- It needs to be updated manually. It does not learn itself.
- Not capable to explain the logic behind the decision.

Exercise 10 - Developing a Simple Medical Expert System

Medical Expert System using Experta module

ABOUT THE MODULE:

Module - experta

Installation - pip install experta

Class KnowledgeEngine

The base clas from which the expert system class is derived

Class Fact

Used to define facts for the ES.

Example :

```
Fact(disease = "Alzheimers")
```

Decorator @DefFacts()

Decorator used to provide initial conditions for the Expert system to proceed. It shoud be used with a **generator** which yields the initial facts required for the ES. The method decorated with @DefFacts is called whenever the reset method of the KnowledgeEngine is called.

Decorator @Rule()

Used to declare the LHS of the expert system, accepting the preconditions as Facts and Patterns. The function decorated with this decorator is fired whenever the LHS matches. Thus acting as the RHS in the expert system. The salience keyword argument specifies the precendence for the Rule

Example:

```
@Rule(
    Fact(find_disease = "yes"),
    Fact(couch = "yes"),
    Fact(fever = "no"), salience = 100)
def rhs_fucnction(self):
    pass
```

W() - wildCard

This when used within Facts matches any value for the variable in the Facts

Example:

```
@Rule (Fact(disease=W())
```

Exercise 10 - Developing a Simple Medical Expert System

Medical Expert System using Experta module

AIM:

To write a python program to develop a simple Medical Expert System.

ABOUT THE MODULE:

Module - experta

Installation - pip install experta

Class KnowledgeEngine

The base clas from which the expert system class is derived

Class Fact

Used to define facts for the ES.

Example :

```
Fact(disease = "Alzheimers")
```

Decorator @DefFacts()

Decorator used to provide initial conditions for the Expert system to proceed. It shoud be used with a **generator** which yields the initial facts required for the ES. The method decorated with @DefFacts is called whenever the reset method of the KnowledgeEngine is called.

Decorator @Rule()

Used to declare the LHS of the expert system, accepting the preconditions as Facts and Patterns. The function decorated with this decorator is fired whenever the LHS matches. Thus acting as the RHS in the expert system. The salience keyword argument specifies the precendence for the Rule

Example:

```
@Rule(
    Fact(find_disease = "yes"),
    Fact(couch = "yes"),
    Fact(fever = "no"), salience = 100)
def rhs_fucnction(self):
    pass
```

W() - wildCard

This when used within Facts matches any value for the variable in the Facts

Example:

```
@Rule (Fact(disease=W())
```

This mathches Fact(disease= “Alzheimers”),Fact(disease= “Tuberculosis”),etc. This will not match if no Facts with ‘disease’ as keyword argument is declared.

NOT() - not

This will match is the given pattern does not match.

Example:

```
@Rule (NOT(Fact(disease=W()))
```

This will match if no Facts with “disease” are declared.

MATCH - object

This will match the all Facts with given arguments and bind its value to the variable wich can be passed to the RHS

Example

```
@Rule (Fact(disease=MATCH.disease))
def rhs_function(self,disease):
    pass
```

This will match if Facts with “disease” are declared and the value can be passed to the RHS.

ENGINE EXECUTION PROCEDURE

This is the usual process to execute a KnowledgeEngine.

1. The class must be instantiated, of course.
2. The reset method must be called:

- This declares the special fact InitialFact. Necessary for some rules to work properly.
- Declare all facts yielded by the methods decorated with @DefFacts.

3. The run method must be called. This starts the cycle of execution.

SOURCE CODE:

```
from experta import *
import yaml

with open("disease/disease_symptoms.yaml", "r") as f:
    (
        SYMPTOMS,
        SYMPTOM_QUERY,
        DISEASE_SYMPTOMS
    ) = yaml.full_load(f).values()

DISEASE_FACTS = {
    disease: [
        Fact(**{symptom: "yes" if symptom in disease_symptoms else "no"})
        for symptom in SYMPTOMS
    ]
```

```
    for disease, disease_symptoms in DISEASE_SYMPTOMS.items()
}
hash_line = "\n"+"# "*50+"\n"
# %%
class MedicalExpert(KnowledgeEngine):
    username = "MR.TESTER"
    response = {}

    def declare_symptom_response(self,symptom):
        self.response[symptom]=input(
            f"\nDo you {SYMPTOM_QUERY[symptom]}?\nPlease type
Yes/No :").strip().lower()
        self.declare(Fact(**{symptom:self.response[symptom]}))

    def declare_disease(self,disease):
        self.declare(Fact(disease=disease))

    @DefFacts()
    def initialization(self):
        response = input(
            "Hi! I am Mr.Expert.\n\n"
            "You can get yourself diagnosed here free of cost!\n"
            "I will ask you 10 questions.\n\n"
            "Do you want to get diagonised?\n"
            "Please type Yes/No :"
        ).strip().lower()
        yield Fact(findDisease= response)

    @Rule(Fact(findDisease="yes"), NOT(Fact(name=W())), salience=1000)
    def ask_name(self):
        self.username = input("\nWhat's your name?") or self.username
        self.declare(Fact(name=self.username))


    @Rule(Fact(findDisease="yes"), NOT(Fact(chest_pain=W())), salience=995)
    def hasChestPain(self):
        self.declare_symptom_response('chest_pain')

    @Rule(Fact(findDisease="yes"), NOT(Fact(cough=W())), salience=985)
    def hasCough(self):
        self.declare_symptom_response('cough')

    @Rule(Fact(findDisease="yes"), NOT(Fact(fainting=W())), salience=975)
    def hasFainting(self):
        self.declare_symptom_response('fainting')

    @Rule(Fact(findDisease="yes"), NOT(Fact(fatigue=W())), salience=970)
    def hasFatigue(self):
        self.declare_symptom_response('fatigue')

    @Rule(Fact(findDisease="yes"), NOT(Fact(headache=W())), salience=965)
    def hasHeadache(self):
        self.declare_symptom_response('headache')
```

```
    @Rule(Fact(findDisease="yes"), NOT(Fact(back_pain=W())), salience=955)
    def hasBackPain(self):
        self.declare_symptom_response('back_pain')

    @Rule(Fact(findDisease="yes"), NOT(Fact(sunken_eyes=W())), salience=950)
    def hasSunkenEyes(self):
        self.declare_symptom_response('sunken_eyes')

    @Rule(Fact(findDisease="yes"), NOT(Fact(fever=W())), salience=945)
    def hasFever(self):
        self.declare_symptom_response('fever')

    @Rule(Fact(findDisease="yes"), NOT(Fact(sore_throat=W())), salience=940)
    def hasSoreThroat(self):
        self.declare_symptom_response('sore_throat')

    @Rule(Fact(findDisease="yes"), NOT(Fact(restlessness=W())), salience=935)
    def hasRestlessness(self):
        self.declare_symptom_response('restlessness')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Covid"])
    def covid(self):
        self.declare_disease('Covid')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Alzheimers"])
    def alzheimers(self):
        self.declare_disease('Alzheimers')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Asthma"])
    def asthma(self):
        self.declare_disease('Asthma')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Diabetes"])
    def diabetes(self):
        self.declare_disease('Diabetes')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Epilepsy"])
    def epilepsy(self):
        self.declare_disease('Epilepsy')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Glaucoma"])
    def glaucoma(self):
        self.declare_disease('Glaucoma')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Heart Disease"])
    def heartDisease(self):
        self.declare_disease('Heart Disease')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Heat Stroke"])
    def heatStroke(self):
        self.declare_disease('Heat Stroke')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Hyperthyroidism"])
    def hyperthyroidism(self):
```

```
        self.declare_disease('Hyperthyroidism')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Hypothermia"])
    def hypothermia(self):
        self.declare_disease('Hypothermia')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Jaundice"])
    def jaundice(self):
        self.declare_disease('Jaundice')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Sinusitis"])
    def sinusitis(self):
        self.declare_disease('Sinusitis')

    @Rule(Fact(findDisease="yes"), *DISEASE_FACTS["Tuberculosis"])
    def tuberculosis(self):
        self.declare_disease('Tuberculosis')

    @Rule(Fact(findDisease="yes"), NOT(Fact(disease=W())), salience=-1)
    def unmatched(self):
        self.declare_disease('unknown')

    @Rule(Fact(findDisease="yes"), Fact(disease=MATCH.disease), salience=1)
    def getDisease(self, disease):
        if(disease == 'unknown'):
            yes_symptoms = {
                symptom
                for symptom,response in self.response.items()
                if response == "yes"
            }
            disease = max(
                DISEASE_SYMPTOMS,
                key=lambda x: len(DISEASE_SYMPTOMS.get(x) & yes_symptoms)
            )
            print('\nWe checked the following symptoms:', *SYMPTOMS, sep="\n")
            print('\nSymptoms found in the patient are:',
                  *yes_symptoms or [None], sep="\n")
            if len(DISEASE_SYMPTOMS[disease] & yes_symptoms) == 0:
                print("\nNo diseases found.You are healthy!")
                return
            else:
                print(
                    "\nWe are unable to tell you the "
                    "exact disease with confidence."
                    "But we believe that you suffer from :",
                    disease
                )
        else:
            print('\nThe most probable illness you are suffering from is:', disease)
        self.print_disease_info(disease)

    def print_disease_info(self, disease):
        print(hash_line)
        print(f'Some info about {disease}:\n')
```

```
        with open("disease/disease_descriptions/" + disease + ".txt", "r") as f:
            print(f.read().strip())
        print(hash_line)
        print(
            f'No need to worry {self.username}. '
            'We even have some preventive measures for you!\n'
        )
        with open("disease/disease_treatments/" + disease + ".txt", "r")as f:
            print(f.read().strip())
        print(hash_line)

if __name__ == "__main__":
    engine = MedicalExpert()
    engine.reset()
    engine.run()
```

OUTPUT:

```
Hi! I am Mr.Expert.

You can get yourself diagnosed here free of cost!
I will ask you 10 questions.

Do you want to get diagonised?
Please type Yes/No : YES

What's your name? MR.TESTER

Do you have chest pain?
Please type Yes/No : NO

Do you have cough?
Please type Yes/No : YES

Do you faint occasionally?
Please type Yes/No : NO

Do you experience fatigue occasionally?
Please type Yes/No : YES

Do you experience headaches?
Please type Yes/No : YES

Do you have back pains?
Please type Yes/No : YES

Do you experience sunken eyes?
Please type Yes/No : NO

Do you experience fever?
Please type Yes/No : NO

Do you experience sore throat?
Please type Yes/No : YES
```

```
Do you experience restlessness?
Please type Yes/No : NO

We checked the following symptoms:
sore_throat
back_pain
chest_pain
sunken_eyes
restlessness
fainting
cough
headache
fatigue
fever

Symptoms found in the patient are:
sore_throat
back_pain
cough
headache
fatigue

We are unable to tell you the exact disease with confidence.But we believe that you
suffer from : Hypothermia

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # #

Some info about Hypothermia:

Hypothermia is a medical emergency that occurs when your body loses heat faster than
it can produce heat, causing a dangerously low body temperature. Normal body
temperature is around 98.6 F (37 C). Hypothermia (hi-poe-THUR-me-uh) occurs as your
body temperature falls below 95 F (35 C).

When your body temperature drops, your heart, nervous system and other organs can't
work normally. Left untreated, hypothermia can eventually lead to complete failure of
your heart and respiratory system and eventually to death.

Hypothermia is often caused by exposure to cold weather or immersion in cold water.
Primary treatments for hypothermia are methods to warm the body back to a normal
temperature.

Hypothermia has two main types of causes. It classically occurs from exposure to
extreme cold.

It may also occur from any condition that decreases heat production or increases heat
loss.
Commonly this includes alcohol intoxication but may also include low blood sugar,
anorexia, and advanced age.

Body temperature is usually maintained near a constant level of 36.5–37.5 °C (97.7–
99.5 °F) through thermoregulation.

Efforts to increase body temperature involve shivering, increased voluntary activity,
and putting on warmer clothing.
```

Diagnosis:

The diagnosis of hypothermia is usually apparent based on a person's physical signs and the conditions in which the person with hypothermia became ill or was found. Blood tests also can help confirm hypothermia and its severity.

A diagnosis may not be readily apparent, however, if the symptoms are mild, as when an older person who is indoors has symptoms of confusion, lack of coordination and speech problems.

# #

No need to worry MR.TESTER. We even have some preventive measures for you!

Depending on the severity of hypothermia, emergency medical care for hypothermia may include one of the following interventions to raise the body temperature:

Passive rewarming: For someone with mild hypothermia, it is enough to cover them with heated blankets and offer warm fluids to drink.

Blood rewarming : Blood may be drawn, warmed and recirculated in the body. A common method of warming blood is the use of a hemodialysis machine, which is normally used to filter blood in people with poor kidney function. Heart bypass machines also may need to be used.

Warm intravenous fluids: A warmed intravenous solution of salt water may be put into a vein to help warm the blood.

Airway rewarming: The use of humidified oxygen administered with a mask or nasal tube can warm the airways and help raise the temperature of the body.

Irrigation: A warm saltwater solution may be used to warm certain areas of the body, such as the area around the lungs (pleura) or the abdominal cavity (peritoneal cavity). The warm liquid is introduced into the affected area with catheters.

# #