



R.M.K. ENGINEERING COLLEGE

[An Autonomous Institution]

R.S.M Nagar, Kavaraipettai, Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu- 601 206
(Affiliated to Anna University, Chennai / Approved by AICTE, New Delhi/ Accredited by NAAC with A+ Grade
An ISO 9001:2015 Certified Institution / All the Eligible UG Programs are accredited by NBA, New Delhi.)



Department of Artificial Intelligence and Data Science

20AI611

Knowledge Engineering Laboratory

LAB MANUAL

2022 - 23 EVEN SEMESTER

VI Semester

B.Tech. Artificial Intelligence and Data Science



R.M.K. ENGINEERING COLLEGE

[An Autonomous Institution]

R.S.M Nagar, Kavaraipettai, Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu- 601 206
(Affiliated to Anna University, Chennai / Approved by AICTE, New Delhi/ Accredited by NAAC with A+ Grade
An ISO 9001:2015 Certified Institution / All the Eligible UG Programs are accredited by NBA, New Delhi.)



Department of Artificial Intelligence and Data Science

20AI611

Knowledge Engineering Lab

LAB MANUAL

2022 - 23 EVEN SEMESTER

VI Semester

B.Tech. Artificial Intelligence and Data Science

Manual Prepared by:

Approved by

Signature of Faculty

Signature of HOD

Dr. M Hemalatha
Associate Professor / ADS

Dr. Sandra Johnson
Professor & Head /ADS

PRINCIPAL
(Dr. K. A. Mohamed Junaid)



R.M.K. ENGINEERING COLLEGE

[An Autonomous Institution]

R.S.M Nagar, Kavaraipttai, Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu- 601 206
(Affiliated to Anna University, Chennai / Approved by AICTE, New Delhi/ Accredited by NAAC with A+ Grade
An ISO 9001:2015 Certified Institution / All the Eligible UG Programs are accredited by NBA, New Delhi.)



Department of Artificial Intelligence and Data Science

VISION OF THE DEPARTMENT

- To become the most preferred destination in the country to learn 'Artificial Intelligence & Data Science' at the undergraduate level and develop professionals of international relevance to meet the societal needs.

MISSION OF THE DEPARTMENT

- M1: To collaborate with industry and provide the state-of-the-art infrastructural facilities, with a conducive teaching-learning ambience.
- M2: To instil in the students the knowledge for world class technical competence, entrepreneurial skill and a spirit of innovation in the area of Artificial Intelligence and Data Science to solve real world problems.
- M3: To encourage students to pursue higher education and research.
- M4: To inculcate right attitude and discipline and develop industry ready professionals for serving the society.

PROGRAM EDUCATIONAL OBJECTIVES

The B.Tech. (Artificial Intelligence & Data Science) Graduates of R.M.K. Engineering College will

PEO 1. Apply the fundamental knowledge of basic sciences, mathematics, computer science, artificial intelligence, and data science to solve socially relevant problems.

PEO 2. Work in multi-disciplinary teams, communicate effectively, and develop world class solutions with the professional knowledge gained by lifelong learning.

PEO 3. Demonstrate ethics, values, integrity, and provide novel innovative solutions during their profession.

PEO 4. Pursue higher studies and research in the areas of Artificial Intelligence and Data Science.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the successful completion of the program, the graduates will be able to:

- Apply the fundamental principles of basic sciences and the knowledge gained from the core concepts of Artificial Intelligence and Data Science to analyze, design and solve complex real world problems.
- Apply strong analytical skills using cutting edge technologies to solve business and engineering problems.
- Excel in Artificial Intelligence and Data Science technologies for career enhancement as an entrepreneur and pursue higher education.



R.M.K. ENGINEERING COLLEGE

[An Autonomous Institution]

R.S.M Nagar, Kavaraipettai, Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu- 601 206
(Affiliated to Anna University, Chennai / Approved by AICTE, New Delhi/ Accredited by NAAC with A+ Grade
An ISO 9001:2015 Certified Institution / All the Eligible UG Programs are accredited by NBA, New Delhi.)



Department of Artificial Intelligence and Data Science

20AI611 KNOWLEDGE ENGINEERING LABORATORY

OBJECTIVES:

- To represent Knowledge for various domains.
- To implement an Expert System.
- To develop Ontologies for a given domain.
- To develop Fuzzy Rule based Systems.
- To apply classification algorithms.

OUTCOMES:

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

- CO1: Represent Knowledge for various domains.
- CO2: Implement an Expert System.
- CO3: Develop Ontologies for a given domain.
- CO4: Develop Fuzzy Rule based Systems.
- CO5: Apply classification algorithms.

Syllabus:

Lab Exercises:

1. Implementation of Missionaries and Cannibals Problem using rule-based approach.
2. Implementation of First Order Logic
3. Implementation of Bayesian networks.
4. Implementation of Semantic Networks.
5. Developing a Fuzzy Inference system
6. Construction of Ontology for a given domain.
7. Implementation of Frames.
8. Develop an expert system for classification of Animals with Property Inheritance
9. Mini Project using Fuzzy Rules and Machine Learning

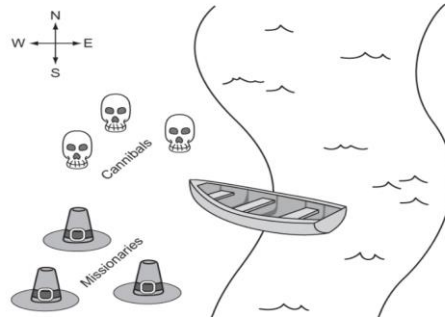
INDEX

SI. NO	EXERCISES	PAGE NO
1.	Implementation of missionaries and cannibals' problem using Rule-based approach	6
2.	Implementation of first order logic	10
3.	Implementation of Bayesian networks	13
4.	Spam filter implementation using bayes theorem	16
5.	Implementation of semantic networks	21
6.	Developing a fuzzy inference system	24
7.	Construction of ontology for a given domain.	27
8.	Implementation of frames.	30
9.	Develop an expert system for classification of animals with property inheritance	33
10.	Forward chaining	37
11.	Backward chaining and goal tree	39

EX.NO.1: MISSIONARIES AND CANNIBALS PROBLEM USING RULE BASED APPROACH

AIM:

In the missionaries and cannibals' problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.



PROCEDURE:

There are 3 missionaries, 3 cannibals, and 1 boat.

The boat can carry up to two people on one side of a river.

- Goal: Move all the missionaries and cannibals across the river.
- Constraint: Missionaries can never be outnumbered by cannibals on either side of river, or else the missionaries are killed.
- State: configuration of missionaries and cannibals and boat on each side of river.
- Operators: Move boat containing some set of occupants across the river (in either direction) to the other side.

State of Left bank of river	Move	State of Right bank of river
3M 3C	1M1C →	1M 1C
2M 2C	1M ←	0M 1C
3M 2C	2C →	0M 3C
3M 0C	1C ←	0M 2C
3M 1C	2M →	2M 2C
1M 1C	1M 1C ←	1M 1C
2M 2C	2M →	3M 1C
0M 3C	1C ←	3M 0C
0M 1C	2C →	3M 2C
1M 1C	1M ←	2M 2C
0M 0C	1M 1C →	3M 3C

ALGORITHM:

1. Define a goal state where all three missionaries and three cannibals are on the opposite side of the river, and the boat is also on that side.
2. Define the state representation as a tuple of four integers: (M, C, B, D), where M is the number of missionaries on the starting side, C is the number of cannibals on the starting side, B is the location of the boat (0 for the starting side, 1 for the destination side), and D is the number of missionaries and cannibals on the destination side.
3. Define the set of valid moves as all possible combinations of 1 or 2 missionaries and/or cannibals crossing the river in the boat, provided that they do not violate the constraint that there cannot be more cannibals than missionaries on either side.
4. Implement a search algorithm such as Breadth-First Search, Depth-First Search, or A* Search to find a path from the initial state to the goal state, using the valid moves to generate new states and checking if each new state is valid and has not been visited before.
5. Output the sequence of states that represents the solution path, along with the corresponding boat movements and the number of moves required to reach the goal state.

PROGRAM:

```
lM = 3          #lM = Left side Missionaries number
lC = 3          #lC = Left side Cannibals number
rM=0           #rM = Right side Missionaries number
rC=0           #rC = Right side cannibals number
userM = 0       #userM = User input for number of missionaries for right to left side travel
userC = 0       #userC = User input for number of cannibals for right to left travel
k = 0
print("\nlM lM lM lC lC lC |      --- | \n")
try:
    while(True):
        while(True):
            print("Left side -> right side river travel")
            #uM = user input for number of missionaries for left to right travel
            #uC = user input for number of cannibals for left to right travel
            uM = int(input("Enter number of Missionaries travel => "))
            uC = int(input("Enter number of Cannibals travel => "))

            if((uM==0)and(uC==0)):
                print("Empty travel not possible")
                print("Re-enter : ")
            elif(((uM+uC) <= 2)and((lM-uM)>=0)and((lC-uC)>=0)):
                break
            else:
                print("Wrong input re-enter : ")

            lM = (lM-uM)
            lC = (lC-uC)
            rM += uM
            rC += uC
```

```

print("\n")
for i in range(0,IM):
    print("M ",end="")
for i in range(0,IC):
    print("C ",end="")
print("| --> | ",end="")
for i in range(0,rM):
    print("M ",end="")
for i in range(0,rC):
    print("C ",end="")
print("\n")

k +=1

if(((IC==3)and (IM == 1))or((IC==3)and(IM==2))or((IC==2)and(IM==1))or((rC==3)and
(rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):
    print("Cannibals eat missionaries:\nYou lost the game")

    break

if((rM+rC) == 6):
    print("You won the game : \n\tCongrats")
    print("Total attempt")
    print(k)
    break
while(True):
    print("Right side -> Left side river travel")
    userM = int(input("Enter number of Missionaries travel => "))
    userC = int(input("Enter number of Cannibals travel => "))

    if((userM==0)and(userC==0)):
        print("Empty travel not possible")
        print("Re-enter : ")
    elif(((userM+userC) <= 2)and((rM-userM)>=0)and((rC-userC)>=0)):
        break
    else:
        print("Wrong input re-enter : ")

    IM += userM
    IC += userC
    rM -= userM
    rC -= userC

    k +=1
    print("\n")
    for i in range(0,IM):
        print("M ",end="")
    for i in range(0,IC):
        print("C ",end="")
    print("| <-- | ",end="")
    for i in range(0,rM):
        print("M ",end="")

```



```

        for i in range(0,rC):
            print("C ",end="")
        print("\n")

        if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and
(rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1)))):
            print("Cannibals eat missionaries:\nYou lost the game")
            break
except EOFError as e:
    print("\nInvalid input please retry !!")

```

OUTPUT:

```

Game Start
Now the task is to move all of them to right side of the river
rules:
1. The boat can carry at most two people
2. If cannibals num greater than missionaries then the cannibals would eat the missionaries
3. The boat cannot cross the river by itself with no people on board

M M C C C | --- |

Left side -> right side river travel
Enter number of Missionaries travel => 1
Enter number of Cannibals travel => 1

M M C C | --> | M C

Right side -> Left side river travel
Enter number of Missionaries travel => 1
Enter number of Cannibals travel => 0

M M C C | <-- | C

Left side -> right side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 2

M M M | --> | C C C

Right side -> Left side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 1

M M M C | <-- | C C

Left side -> right side river travel
Enter number of Missionaries travel => 2
Enter number of Cannibals travel => 0

M C | --> | M M C C

Right side -> Left side river travel
Enter number of Missionaries travel => 1
Enter number of Cannibals travel => 1

M M C C | <-- | M C

```

RESULT:

Thus, implementation of Missionaries and Cannibals Problem using rule-based approach is successful.

EX.NO.2: IMPLEMENTATION OF FIRST ORDER LOGIC (FOL)

AIM:

To implement First Order Logic (FOL) using python for the below stated problem statements

Statement 1: The law says that it is a crime to sell weapons to hostile nations.

Statement 2: The country Nano, an enemy of America, has some missiles.

Statement 3: All of its missiles were sold to it by colonel West, who is an American.

Statement 4: Prove that Colonel West is a criminal.

PROCEDURE:

Step1 : It is crime for Americans to sell the weapon to the enemy of America

FOL: $\text{American}(x) \text{ Weapon}(y) \text{ Enemy}(z, \text{America}) \text{ Sell}(x, y, z) \text{ Criminal}(x)$

Step2 : Country Nono is enemy of America

FOL : $\text{Enemy}(\text{Nono}, \text{America})$

Step3 : Nono has some Missile

FOL : $\text{Owns}(\text{Nono}, x)$

$\text{Missile}(x)$

Step4 : All missiles were sold to Nono by Colonel

FOL : $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sell}(\text{Colonel}, x, \text{Nono})$

Step5 : Colonel is American

FOL : $\text{American}(\text{Colonel})$

Step6 : Missile is weapon

FOL : $\text{Missile}(x) \rightarrow \text{Weapon}(x)$

ALGORITHM:

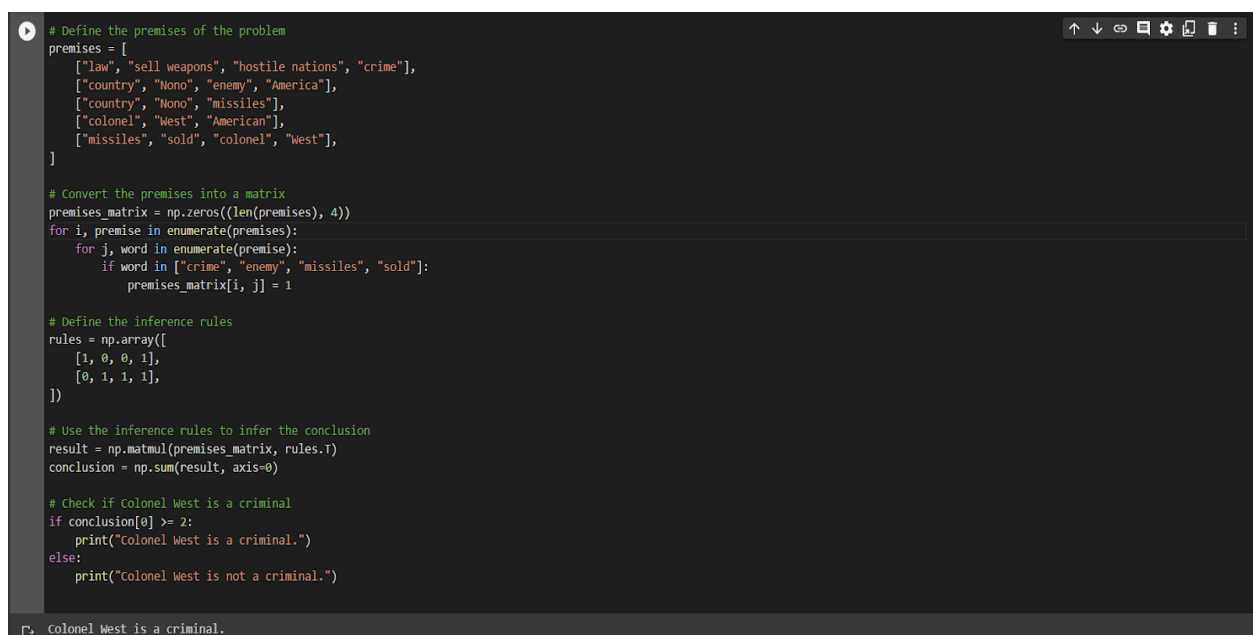
1. Import the NumPy library.
2. Define the premises of the problem as a list of lists. Each inner list represents a premise and contains words that describe the premise.
3. Convert the premises list into a NumPy matrix, where each row represents a premise and each column represents a word.
4. Define two inference rules as a NumPy matrix, where each row represents a rule and each column represents a word.
5. Multiply the premises matrix with the transpose of the rules matrix to get a result matrix, where each row represents the conclusion of a rule.
6. Calculate the sum of each column of the result matrix.
7. If the sum of a column is greater than or equal to 2, Colonel West is considered a criminal.
8. Print "Colonel West is a criminal" if Colonel West is a criminal, or "Colonel West is not a criminal."; if Colonel West is not a criminal.

PROGRAM 1:

```
import numpy as np
# Define the premises of the problem
premises = [
    ["law", "sell weapons", "hostile nations", "crime"],
    ["country", "Nono", "enemy", "America"],
    ["country", "Nono", "missiles"],
    ["colonel", "West", "American"],
    ["missiles", "sold", "colonel", "West"],
]
# Convert the premises into a matrix
premises_matrix = np.zeros((len(premises), 4))
for i, premise in enumerate(premises):
    for j, word in enumerate(premise):
        if word in ["crime", "enemy", "missiles", "sold"]:
            premises_matrix[i, j] = 1
# Define the inference rules
rules = np.array([
    [1, 0, 0, 1],
    [0, 1, 1, 1],
])
# Use the inference rules to infer the conclusion
result = np.matmul(premises_matrix, rules.T)
conclusion = np.sum(result, axis=0)

# Check if Colonel West is a criminal
if conclusion[0] >= 2:
    print("Colonel West is a criminal.")
else:
    print("Colonel West is not a criminal.")
```

OUTPUT

A screenshot of a code editor with a dark theme. The editor displays the same Python code as in the 'PROGRAM 1' section. At the bottom of the editor, a console window shows the output: 'Colonel West is a criminal.' The code is color-coded with green for comments, blue for keywords, and white for identifiers and literals. The console output is in white text on a dark background.

```
# Define the premises of the problem
premises = [
    ["law", "sell weapons", "hostile nations", "crime"],
    ["country", "Nono", "enemy", "America"],
    ["country", "Nono", "missiles"],
    ["colonel", "West", "American"],
    ["missiles", "sold", "colonel", "West"],
]

# Convert the premises into a matrix
premises_matrix = np.zeros((len(premises), 4))
for i, premise in enumerate(premises):
    for j, word in enumerate(premise):
        if word in ["crime", "enemy", "missiles", "sold"]:
            premises_matrix[i, j] = 1

# Define the inference rules
rules = np.array([
    [1, 0, 0, 1],
    [0, 1, 1, 1],
])

# Use the inference rules to infer the conclusion
result = np.matmul(premises_matrix, rules.T)
conclusion = np.sum(result, axis=0)

# Check if Colonel West is a criminal
if conclusion[0] >= 2:
    print("Colonel West is a criminal.")
else:
    print("Colonel West is not a criminal.")
```

Colonel West is a criminal.

PROGRAM 2:

```
!pip install aim3
import aim3.utils
import aim3.logic
def main():
    clauses = []
    clauses.append(aim3.utils.expr("(American(x) & Weapon(y) & Sells(x
, y, z) & Hostile(z)) ==> Criminal(x)"))
    clauses.append(aim3.utils.expr("Enemy(Nono, America)"))
    clauses.append(aim3.utils.expr("Owns(Nono, M1)"))
    clauses.append(aim3.utils.expr("Missile(M1)"))
    clauses.append(aim3.utils.expr("(Missile(x) & Owns(Nono, x)) ==> S
ells(West, x, Nono)"))

    clauses.append(aim3.utils.expr("American(West)"))
    clauses.append(aim3.utils.expr("Missile(x) ==> Weapon(x)"))
    KB = aim3.logic.FolKB(clauses)
    KB.tell(aim3.utils.expr("Enemy(x, America) ==> Hostile(x)"))
    hostile = aim3.logic.fol_fc_ask(KB, aim3.utils.expr('Hostile(x)')
)
    criminal = aim3.logic.fol_fc_ask(KB, aim3.utils.expr('Criminal(x)
'))
    print('Hostile?')
    print(list(hostile))
    print('\nCriminal?')
    print(list(criminal))
    print()
    if __name__ == "__main__": main()
```

OUTPUT

```
Hostile?
[{x: Nono}]
Criminal?
[{x: West}]
```

RESULT

Thus, the implementation of First Order Logic for the given problem statement has been executed successfully.

EX.NO.3:**IMPLEMENTATION OF BAYESIAN NETWORKS****AIM:**

To write a python program to implement Bayesian networks.

PROCEDURE:

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

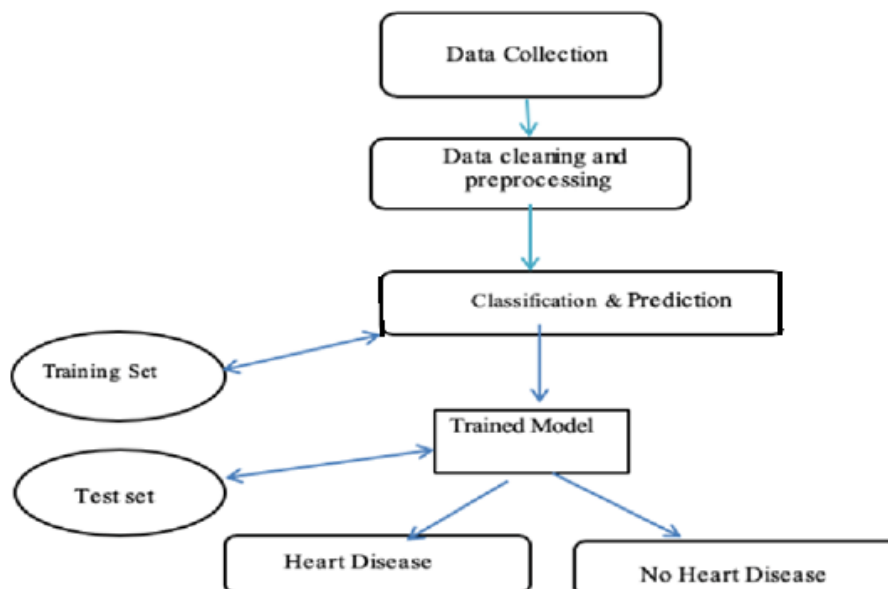
Bayesian network consists of two major parts:

- A directed acyclic graph

- A set of conditional probability distributions

The directed acyclic graph is a set of random variables represented by nodes.

The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).



PROGRAM:

```
pip install pgmpy
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('/content/7-dataset.csv')
heartDisease = heartDisease.replace('?',np.nan)
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
model=
BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','heartdi
sease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDisetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDisetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDisetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

OUTPUT:

Sample instances from the dataset are given below

	age	gender	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak
0	63	1	1	145	233	1	2	150	0	2.3
1	67	1	4	160	286	0	2	108	1	1.5
2	67	1	4	120	229	0	2	129	1	2.6
3	37	1	3	130	250	0	0	187	0	3.5
4	41	0	2	130	204	0	2	172	0	1.4

	slope	ca	thal	heartdisease
0	3	0	6	0
1	2	3	3	2
2	2	2	7	1
3	3	0	3	0
4	1	0	3	0

```

Attributes and datatypes
age                int64
gender            int64
cp               int64
trestbps         int64
chol             int64
fbs             int64
restecg         int64
thalach         int64
exang           int64
oldpeak        float64
slope           int64
ca              object
thal           object
heartdisease    int64
dtype: object

```

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

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

2. Probability of HeartDisease given evidence= cp

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

RESULT:

Thus, the program to implement Bayesian networks is implemented successfully

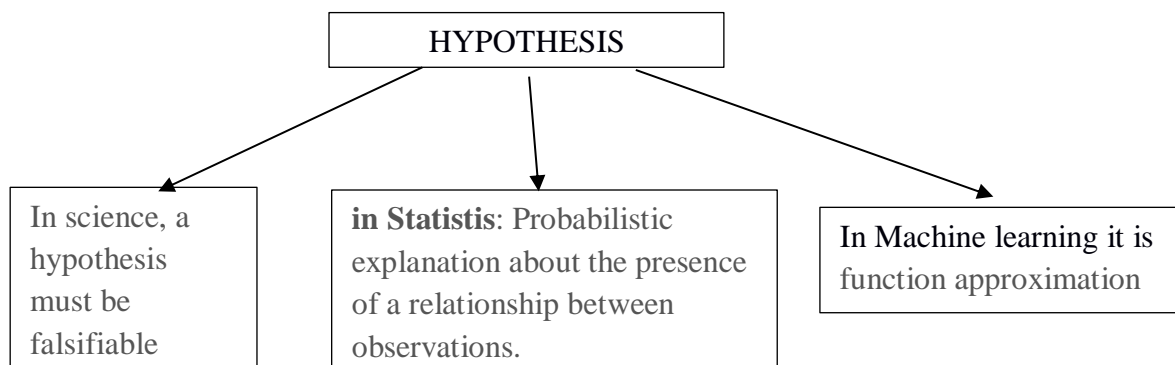
EX.NO.4: SPAM FILTER IMPLEMENTATION USING BAYES THEOREM

AIM:

To build a spam filter using Python and the multinomial Naive Bayes algorithm that classifies messages with higher accuracy.

PROCEDURE:

A hypothesis is an explanation for something. A good hypothesis is testable; it can be either true or false.



ALGORITHM:

1. Download the dataset of 5,572 SMS messages. Tiago A. Almeida and José María Gómez Hidalgo put together the dataset, you can download it from the UCI Machine Learning Repository.
2. Import the Pandas library and read in a dataset of SMS messages, label them as spam or ham.
3. Randomize the dataset and split it into training and test sets.
4. Clean the training set by removing punctuation, converting all letters to lowercase, and creating a vocabulary of unique words.
5. Calculate the parameters needed to apply Naive Bayes to classify messages as spam or ham. These include the prior probabilities of spam and ham, the total number of words in spam and ham messages, and the conditional probabilities of each word given spam or ham.
6. Define a function to classify a message as spam or ham based on the parameters calculated in step four.
7. Apply the classify function to the test set to make predictions, and store the predicted labels in a new column.
8. Evaluate the accuracy of the predictions by comparing them to the actual labels in the test set.

PROGRAM:

```
import pandas as pd
sms_spam = pd.read_csv('SMSSpamCollection', sep='\t',
header=None, names=['Label', 'SMS'])
print(sms_spam.shape)
sms_spam.head()
sms_spam['Label'].value_counts(normalize=True)
# Randomize the dataset
data_randomized = sms_spam.sample(frac=1, random_state=1)
# Calculate index for split
training_test_index = round(len(data_randomized) * 0.8)
# Split into training and test sets
training_set = data_randomized[:training_test_index].reset_index(drop=True)
test_set = data_randomized[training_test_index:].reset_index(drop=True)
print(training_set.shape)
print(test_set.shape)
training_set['Label'].value_counts(normalize=True)
test_set['Label'].value_counts(normalize=True)
# Before cleaning
training_set.head(3)
# After cleaning
training_set['SMS'] = training_set['SMS'].str.replace(
'\W', ' ') # Removes punctuation
training_set['SMS'] = training_set['SMS'].str.lower()
training_set.head(3)
training_set['SMS'] = training_set['SMS'].str.split()
vocabulary = []
for sms in training_set['SMS']:
for word in sms:
vocabulary.append(word)
vocabulary = list(set(vocabulary))
len(vocabulary)
word_counts_per_sms = {'secret': [2,1,1],

'prize': [2,0,1],
'claim': [1,0,1],
'now': [1,0,1],
'coming': [0,1,0],
'to': [0,1,0],
'my': [0,1,0],
'party': [0,1,0],
'winner': [0,0,1]
}

word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
word_counts_per_sms = {unique_word: [0] * len(training_set['SMS']) for unique_word in
vocabulary}
for index, sms in enumerate(training_set['SMS']):
for word in sms:
```

```

word_counts_per_sms[word][index] += 1
word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
training_set_clean = pd.concat([training_set, word_counts], axis=1)
training_set_clean.head()
# Isolating spam and ham messages first
spam_messages = training_set_clean[training_set_clean['Label'] == 'spam']
ham_messages = training_set_clean[training_set_clean['Label'] == 'ham']
# P(Spam) and P(Ham)
p_spam = len(spam_messages) / len(training_set_clean)
p_ham = len(ham_messages) / len(training_set_clean)
# N_Spam
n_words_per_spam_message = spam_messages['SMS'].apply(len)
n_spam = n_words_per_spam_message.sum()
# N_Ham
n_words_per_ham_message = ham_messages['SMS'].apply(len)
n_ham = n_words_per_ham_message.sum()
# N_Vocabulary
n_vocabulary = len(vocabulary)
# Laplace smoothing
alpha = 1
# Initiate parameters
parameters_spam = {unique_word:0 for unique_word in vocabulary}
parameters_ham = {unique_word:0 for unique_word in vocabulary}
# Calculate parameters
for word in vocabulary:
    n_word_given_spam = spam_messages[word].sum() # spam_messages already
    defined
    p_word_given_spam = (n_word_given_spam + alpha) / (n_spam +
    alpha*n_vocabulary)
    parameters_spam[word] = p_word_given_spam
    n_word_given_ham = ham_messages[word].sum() # ham_messages already defined
    p_word_given_ham = (n_word_given_ham + alpha) / (n_ham + alpha*n_vocabulary)
    parameters_ham[word] = p_word_given_ham
import re
def classify(message):
    """
    message: a string
    """

    message = re.sub('\W', ' ', message)
    message = message.lower().split()
    p_spam_given_message = p_spam
    p_ham_given_message = p_ham
    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]
        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]
    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)
    if p_ham_given_message > p_spam_given_message:

```

```

print('Label: Ham')
elif p_ham_given_message < p_spam_given_message:
print('Label: Spam')
else:
print('Equal probabilities, have a human classify this!')
classify('WINNER!! This is the secret code to unlock the money: C3421.')
classify("Sounds good, Tom, then see u there")
def classify_test_set(message):
'''
message: a string
'''

message = re.sub('\W', ' ', message)
message = message.lower().split()
p_spam_given_message = p_spam
p_ham_given_message = p_ham
for word in message:
if word in parameters_spam:
p_spam_given_message *= parameters_spam[word]
if word in parameters_ham:
p_ham_given_message *= parameters_ham[word]
if p_ham_given_message > p_spam_given_message:
return 'ham'
elif p_spam_given_message > p_ham_given_message:
return 'spam'
else:
return 'needs human classification'
test_set['predicted'] = test_set['SMS'].apply(classify_test_set)
test_set.head()
correct = 0
total = test_set.shape[0]
for row in test_set.iterrows():
row = row[1]
if row['Label'] == row['predicted']:
correct += 1
print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', correct/total)

```

OUTPUT:

```
p_ham_given_message = p_ham
for word in message:
    if word in parameters_spam:
        p_spam_given_message *= parameters_spam[word]
    if word in parameters_ham:
        p_ham_given_message *= parameters_ham[word]
    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'
test_set['predicted'] = test_set['SMS'].apply(classify_test_set)
test_set.head()
correct = 0
total = test_set.shape[0]
for row in test_set.iterrows():
    row = row[1]
    if row['label'] == row['predicted']:
        correct += 1
print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', correct/total)
```

Correct: 1100
Incorrect: 14
Accuracy: 0.9874326750448833

RESULT:

Thus, a Spam Filter has been implemented using Bayes Theorem successfully.

EX.NO.5: IMPLEMENTATION OF SEMANTIC NETWORKS

AIM:

To write python code that performs semantic network implementation.

PROCEDURE:

- Natural Language Processing, which is responsible for interaction between users and machines using natural language.
- Machine Learning algorithms are mostly statistically based and work on crunching numbers and digits
- To let the machine, understand our motive using Natural Language, we need to convert our textual data to the machine-understandable form, i.e., numbers

Five Natural Language Processing Phases

Natural Language rules are messy and loose, and transferring our ideas to a machine using natural language processing follows the below set of rules or steps

What Is Semantic Analysis?

This step is responsible for generating the possible meaning of the sentence by joining word-level meanings together. It helps to identify the text elements and find their logical meanings.

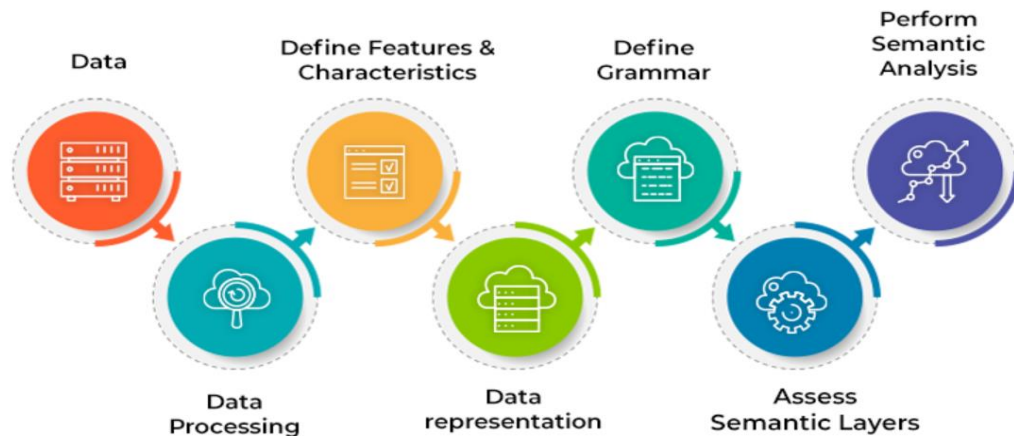


e.g., “colorful red” may seem grammatically correct, but logically it is irrelevant, so that the Semantic Analyzer will discard it.

Semantic analysis refers to a process of understanding natural language (text) by extracting insightful information such as context, emotions, and sentiments from unstructured data. It gives computers and systems the ability to understand, interpret, and derive meanings from sentences, paragraphs, reports, registers, files, or any document of a similar kind.

How Does Semantic Analysis Work?

MEANING OF INDIVIDUAL WORDS + COMBINATION OF WORDS => SEMANTIC ANALYSIS



Semantic Analysis Techniques:

Semantic analysis methodology can be broadly classified in two ways as shown in below figure



Applications of Semantic Analysis

1. Conversational chatbots
2. Automated ticketing support
3. Search engine results
4. Language translation

ALGORITHM:

1. Name entity recognition is implemented using Natural Language Processing.
2. Download the necessary NLTK packages.
3. Import the required packages.
4. Extract the sentences from the text given as input.
5. Perform tokenization and extract entities from the text.
6. Classify them into different categories accordingly.
7. Use display function and display the categories of a particular word.
8. Evaluate results.

PROGRAM:

```
#Type here the complete program
import networkx as nx
# Create an empty directed graph
G = nx.DiGraph()
# Add nodes to the graph
G.add_nodes_from(['food', 'fruit', 'vegetable', 'apple', 'carrot', 'banana'])
# Add edges to the graph
G.add_edge('food', 'fruit')
G.add_edge('food', 'vegetable')
G.add_edge('fruit', 'apple')
G.add_edge('fruit', 'banana')
G.add_edge('vegetable', 'carrot')
# Compute degree centrality
dc = nx.degree_centrality(G)
# Print degree centrality
for node, deg in dc.items():
    print(f"{node}: {deg}")
import matplotlib.pyplot as plt
# Compute layout
pos = nx.spring_layout(G)
# Draw the graph
nx.draw(G, pos, with_labels=True)
# Show the plot
plt.show()
```

OUTPUT:

```
food:0.4
fruit:0.60000000000000001
vegetable:0.4
apple:0.2
carrot:0.2
banana:0.2
```

RESULT:

Thus the python code that performs semantic network implementation was executed successfully.

EX.NO.06:

FUZZY INFERENCE SYSTEM

AIM:

To develop python code that implements fuzzy inference system.

PROCEDURE:

What is Fuzzy Inference System?

Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It uses the “IF...THEN” rules along with connectors “OR” or “AND” for drawing essential decision rules.

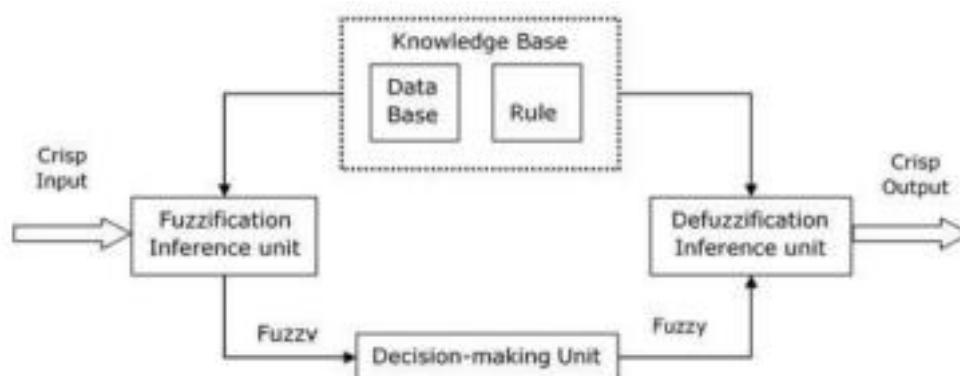
Characteristics of Fuzzy Inference System (FIS)

- The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.
- It is necessary to have fuzzy output when it is used as a controller.
- A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.
-

Functional Blocks of FIS

- **Rule Base** – It contains fuzzy IF-THEN rules.
- **Database** – It defines the membership functions of fuzzy sets used in fuzzy rules.
- **Decision-making Unit** – It performs operation on rules.
- **Fuzzification Interface Unit** – It converts the crisp quantities into fuzzy quantities.
- **Defuzzification Interface Unit** – It converts the fuzzy quantities into crisp quantities.

Block Diagram of Fuzzy Interference System:



Working of FIS

- A Fuzzification unit supports the application of numerous fuzzification methods, and converts the crisp input into fuzzy input.
- A knowledge base - collection of rule base and database is formed upon the conversion of crisp input into fuzzy input.
- The defuzzification unit fuzzy input is finally converted into crisp output.

ALGORITHM:

1. Import all the necessary libraries.
2. Install the fuzzy package using pip install from sci-kit.
3. Import the dataset needed and perform fuzzy logic using the rules.
4. Read crisp value from the process
5. Maps the crisp value into fuzzy value using the fuzzy membership function
6. Apply IF-THEN rules from the fuzzy rule base and compute fuzzy output
7. Convert fuzzy output into crisp by applying some defuzzification methods.

PROGRAM:

```
pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Create the input variables
temperature = ctrl.Antecedent(np.arange(0, 101, 1), '&#39;temperature&#39;)
humidity = ctrl.Antecedent(np.arange(0, 101, 1), '&#39;humidity&#39;)

# Create the output variable
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), '&#39;fan_speed&#39;)
# Create the membership functions for the input variables
temperature[&#39;low&#39;] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature[&#39;medium&#39;] = fuzz.trimf(temperature.universe, [0, 50, 100])
temperature[&#39;high&#39;] = fuzz.trimf(temperature.universe, [50, 100, 100])
humidity[&#39;low&#39;] = fuzz.trimf(humidity.universe, [0, 0, 50])
humidity[&#39;medium&#39;] = fuzz.trimf(humidity.universe, [0, 50, 100])
humidity[&#39;high&#39;] = fuzz.trimf(humidity.universe, [50, 100, 100])

# Create the membership functions for the output variable
fan_speed[&#39;low&#39;] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed[&#39;medium&#39;] = fuzz.trimf(fan_speed.universe, [0, 50, 100])
fan_speed[&#39;high&#39;] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Create the rules for the fuzzy system
rule1 = ctrl.Rule(temperature[&#39;low&#39;] & humidity[&#39;low&#39;],
fan_speed[&#39;low&#39;]
)
rule2 = ctrl.Rule(temperature[&#39;low&#39;] & humidity[&#39;medium&#39;],
fan_speed[&#39;m
edium&#39;])
rule3 = ctrl.Rule(temperature[&#39;low&#39;] & humidity[&#39;high&#39;],
fan_speed[&#39;hig
h&#39;])
```

```

rule4 = ctrl.Rule(temperature[&#39;medium&#39;] & amp; humidity[&#39;low&#39;],
fan_speed[&#39;m
edium&#39;])
rule5 = ctrl.Rule(temperature[&#39;medium&#39;] & amp; humidity[&#39;medium&#39;],
fan_speed
[&#39;medium&#39;])
rule6 = ctrl.Rule(temperature[&#39;medium&#39;] & amp; humidity[&#39;high&#39;],
fan_speed[&#39;
high&#39;])
rule7 = ctrl.Rule(temperature[&#39;high&#39;] & amp; humidity[&#39;low&#39;],
fan_speed[&#39;hig
h&#39;])
rule8 = ctrl.Rule(temperature[&#39;high&#39;] & amp; humidity[&#39;medium&#39;],
fan_speed[&#39;
high&#39;])
rule9 = ctrl.Rule(temperature[&#39;high&#39;] & amp; humidity[&#39;high&#39;],
fan_speed[&#39;hi
gh&#39;])
# Create the control system
fan_speed_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5,
rule6, rule7, rule8, rule9])
fan_speed_system = ctrl.ControlSystemSimulation(fan_speed_ctrl)

# Pass inputs to the system and get the output
fan_speed_system.input[&#39;temperature&#39;] = 80
fan_speed_system.input[&#39;humidity&#39;] = 40
fan_speed_system.compute()

```

OUTPUT:

58.78048780487803

RESULT:

Thus the python code for implementation of fuzzy inference system was executed successfully.

EX.NO: 07 CONSTRUCTION OF ONTOLOGY FOR A GIVEN DOMAIN

AIM:

To construct an Ontology for a given domain.

PROCEDURE:

Definition of Ontology:

An ontology is an explicit formal specification of the terms that are used to represent an agent's world.

In an ontology,

- **Definitions** associate names of entities in the agent's world with human readable text and formal axioms.
- **Text** describes what a name means.
- **Axioms** constrain the interpretation and use of a term.

Idea of the ontological representation:

- To represent knowledge in the form of a graph
- Here nodes represent objects, situations, or events.
- The arcs represent the relationships between them.

How/why does ontology plays a crucial role in cognitive assistants?

- Ontology is the basis of knowledge representation, user-agent communication, problem solving, knowledge acquisition, and learning in cognitive assistants.
- First, the ontology provides the basic representational constituents for all the elements of the knowledge base.
- Second, the agent's ontology enables the agent to communicate with the user and with other agents by declaring the terms that the agent understands through ontological commitments.
- Third, the problem-solving methods or rules of the agent are applied by matching them against the current state of the agent's world, which is represented in the ontology.
- Fourth, a main focus of knowledge acquisition is the elicitation of the domain concepts and of their hierarchical organization.
- Fifth, the ontology represents the generalization hierarchy for learning, in which specific problem-solving episodes are generalized into rules by replacing instances with concepts from the ontology.

ALGORITHM:

1. Define the domain: Decide on the scope and boundaries of the domain that the ontology will represent. This includes identifying the key concepts, entities, and relationships that are relevant to the domain.
2. Create a list of terms: Compile a list of terms and concepts that are specific to the domain. This includes nouns, verbs, adjectives, and other descriptive terms.
3. Categorize the terms: Group the terms into categories and subcategories based on their meaning and relationships to other terms. This helps to organize the ontology and provide a structure for the concepts within the domain.

4. Define the relationships: Identify the relationships between the concepts and entities within the domain. This includes defining hierarchies, part-whole relationships, causal relationships, and other types of relationships.
5. Choose an ontology language: Decide on an ontology language, such as OWL, RDF, or RDFS, that will be used to represent the ontology.
6. Create the ontology: Use an ontology editor, such as Protégé, to create the ontology. Define the classes, properties, and relationships within the ontology using the chosen ontology language.
7. Test and refine the ontology: Test the ontology with sample data and refine it based on feedback and evaluation. This includes verifying that the ontology accurately represents the concepts and relationships within the domain and that it can be used effectively in the intended applications.
8. Publish and maintain the ontology: Publish the ontology in a machine-readable format and make it available to users. Maintain the ontology by updating it as new concepts and relationships are discovered within the domain.

PROGRAM:

```
from owlready2 import *

# Create an ontology
onto = Ontology("http://example.com/population")

# Define classes
class Population(Thing):
    pass

class Location(Thing):
    pass

class Capital(Thing):
    pass

class Part(Thing):
    pass

# Define properties
class hasPartOf(ObjectProperty):
    domain = [Population]
    range = [Part]

class isLocatedIn(ObjectProperty):
    domain = [Population]
    range = [Location]

class isCapitalOf(ObjectProperty):
    domain = [UrbanArea]
    range = [Capital]

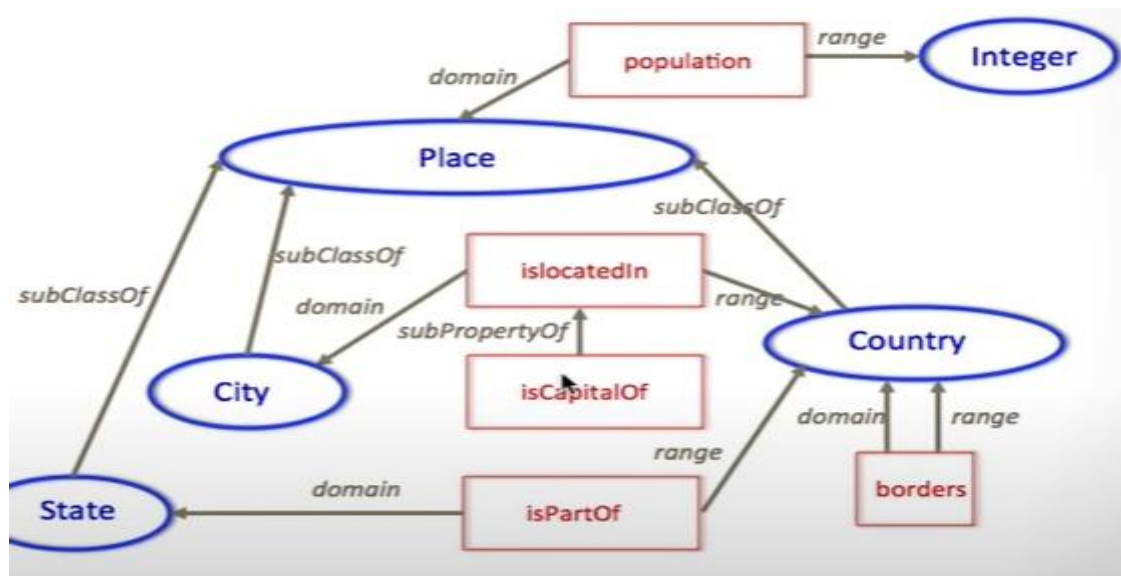
# Create instances
pop = Population("pop")
```

```
loc = Location("loc")
cap = Capital("cap")
part = Part("part")
```

```
# Add properties
pop.isLocatedIn.append(loc)
pop.isCapitalOf.append(cap)
urb.isPartOf.append(part)
```

```
# Save ontology to file
onto.save(file="population.owl", format="rdfxml")
```

OUTPUT:



RESULT:

Thus, an ontology of car domain is constructed successfully.

EX.NO.8: IMPLEMENTATION OF FRAMES

AIM:

To develop python code that performs implementation of frames.

PROCEDURE:

Frames in Knowledge Representation:

- Frames are more structured form of packaging knowledge,
- Frames are used for representing objects, concepts etc.
- Frames are organized into hierarchies or network of frames.
- Lower-level frames can inherit information from upper-level frames in network.
- Nodes are connected using links viz.,

Description of Frames

- Each frame represents either a class or an instance.
- Class frame represents a general concept whereas instance frame represents a specific occurrence of the class instance.
- Class frame generally have default values which can be redefined at lower levels.
- If class frame has actual value facet then decedent frames can not modify that value.
- Value remains unchanged for subclasses and instances.

Advantages of frame representation:

- The frame knowledge representation makes the programming easier by grouping the related data.
- The frame representation is comparably flexible and used by many applications in AI.
- It is very easy to add slots for new attribute and relations.
- It is easy to include default data and to search for missing values.
- Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

- In frame system inference mechanism is not be easily processed.
- Inference mechanism cannot be smoothly proceeded by frame representation.
- Frame representation has a much generalized approach.

ALGORITHM:

1. Define the frames to be used. Each frame should have attributes that describe the properties of the objects being represented.
2. Define the knowledge base, which should contain information about different objects in the form of instances of the frames.
3. Implement a user interface that prompts the user to input the type of object and its name.

4. If the object is in the knowledge base, use the attributes of the objects frame to provide a response to the user query.
5. If the object is not in the knowledge base, inform the user that the object is not recognized.

PROGRAM:

Define the Animal and Bird frames

```
Animal = {  
    "name": "",  
    "habitat": "",  
    "sound": ""  
}
```

```
Bird = {  
    "beak": ""  
}
```

Define the bird knowledge base

```
parrot = {  
    "name": "parrot",  
    "habitat": "forest",  
    "sound": "squawk",  
    "beak": "hooked"  
}
```

```
penguin = {  
    "name": "penguin",  
    "habitat": "arctic",  
    "sound": "honk",  
    "beak": "sharp"  
}
```

```
ostrich = {  
    "name": "ostrich",  
    "habitat": "savanna",  
    "sound": "boom",  
    "beak": "long"  
}
```

Define the mammal knowledge base

```
dog = {  
    "name": "dog",  
    "habitat": "home",  
    "sound": "bark",  
    "legs": 4  
}
```

```
cat = {  
    "name": "cat",  
    "habitat": "home",  
    "sound": "meow",  
    "legs": 4  
}
```

```

}
elephant = {
    "name": "elephant",
    "habitat": "jungle",
    "sound": "trumpet",
    "legs": 4
}
# Define the user interface
animal_type = input("Enter the type of animal (mammal or bird): ")
animal = input("Enter the name of the animal: ")
if animal_type.lower() == "bird":
    if animal.lower() == "parrot":
        print(f"The {parrot['name']} is a {animal_type} that lives in the {parrot['habitat']} and
makes a {parrot['sound']} sound. Its beak is {parrot['beak']}".)
    elif animal.lower() == "penguin":
        print(f"The {penguin['name']} is a {animal_type} that lives in the {penguin['habitat']} and
makes a {penguin['sound']} sound. Its beak is {penguin['beak']}".)
    elif animal.lower() == "ostrich":
        print(f"The {ostrich['name']} is a {animal_type} that lives in the {ostrich['habitat']} and
makes a {ostrich['sound']} sound. Its beak is {ostrich['beak']}".)
    else:
        print("Sorry, the animal is not in the knowledge base.")
elif animal_type.lower() == "mammal":
    if animal.lower() == "dog":
        print(f"The {dog['name']} is a {animal_type} that lives in the {dog['habitat']} and makes a
{dog['sound']} sound. It has {dog['legs']} legs.")
    elif animal.lower() == "cat":
        print(f"The {cat['name']} is a {animal_type} that lives in the {cat['habitat']} and makes a
{cat['sound']} sound. It has {cat['legs']} legs.")
    elif animal.lower() == "elephant":
        print(f"The {elephant['name']} is a {animal_type} that lives in the {elephant['habitat']} and
makes a {elephant['sound']} sound. It has {elephant['legs']} legs.")
    else:
        print("Sorry, the animal is not in the knowledge base.")

```

OUTPUT:

Enter the type of animal(mammal or bird):bird
Enter the name of animal:ostrich
the ostrich is a bird that lives in the savanna and makes a boom sound.Its beak is long.

RESULT:

Thus the python code for implementation of frames was executed successfully.

EX.NO.9: DEVELOP AN EXPERT SYSTEM FOR ANIMAL CLASSIFICATION WITH PROPERTY INHERITANCE

AIM:

To develop an expert system for animal classification with property inheritance.

PROCEDURE:

What is an expert system (ES)?

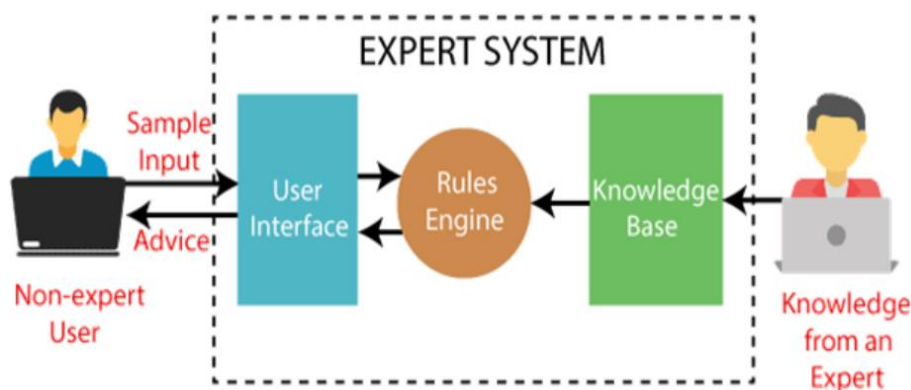
An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI,

First ES was developed in the year 1970.

Performance of an expert system:

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.



Characteristics of Expert System

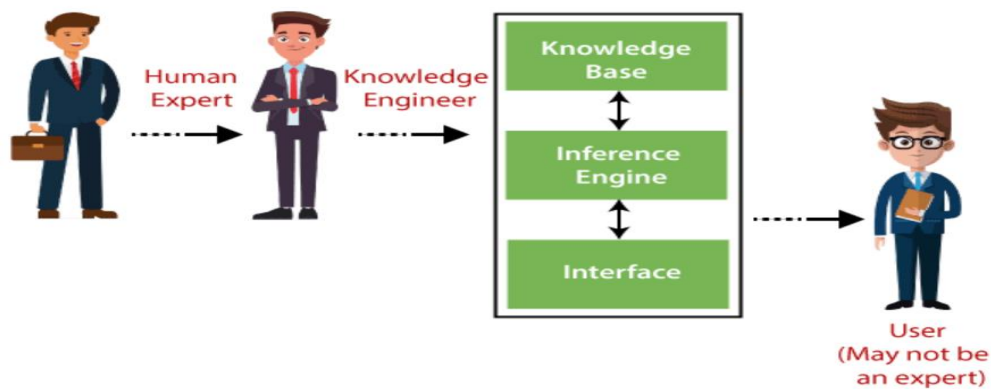
- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- User Interface
- Inference Engine

- Knowledge Base



ALGORITHM:

1. Define the knowledge base that includes the properties and characteristics of different animals.
2. Create a database to store the relevant data.
3. Define a set of rules that match the input properties with the corresponding animal type and name.
4. Develop a user interface that allows users to input data and receive the classification results.
5. Test the system thoroughly and refine it as needed to improve its accuracy.

PROGRAM:

Define the knowledge base

```

knowledge_base = {
    'mammal': {
        'dog': {
            'size': 'medium',
            'coat': 'short',
            'feathers': 'no',
            'beak': 'no',
            'tail': 'yes',
            'legs': 4
        },
        'cat': {
            'size': 'small',
            'coat': 'short',
            'feathers': 'no',
            'beak': 'no',
            'tail': 'yes',
            'legs': 4
        },
        'elephant': {
            'size': 'large',
            'coat': 'rough',

```

```

        'feathers': 'no',
        'beak': 'no',
        'tail': 'yes',
        'legs': 4
    }
},
'bird': {
    'parrot': {
        'size': 'small',
        'coat': 'no',
        'feathers': 'colorful',
        'beak': 'hooked',
        'tail': 'no',
        'legs': 2
    },
    'penguin': {
        'size': 'medium',
        'coat': 'no',
        'feathers': 'black and white',
        'beak': 'sharp',
        'tail': 'no',
        'legs': 2
    },
    'ostrich': {
        'size': 'large',
        'coat': 'no',
        'feathers': 'brown',
        'beak': 'long',
        'tail': 'no',
        'legs': 2
    }
}
}

```

```

# Define the rules for classification
def classify_animal(animal_properties):
    for animal_type, animals in knowledge_base.items():
        for animal, properties in animals.items():
            match = True
            for prop, val in animal_properties.items():
                if properties.get(prop) != val:
                    match = False
                    break
            if match:
                return animal_type, animal
    return None, None

```

```

def main():
    animal_type = input("Enter the type of animal (mammal or bird): ")
    size = input("Size: ")
    coat = input("Coat: ")
    feathers = input("Feathers: ")
    beak = input("Beak: ")
    tail = input("Tail: ")
    legs = int(input("Legs: "))
    animal_properties = {
        'size': size,
        'coat': coat,
        'feathers' : feathers,
        'beak': beak,
        'tail' : tail,
        'legs': legs
    }
    animal_type, animal = classify_animal(animal_properties)
    if animal_type:
        print(f"The animal is a {animal} and belongs to the {animal_type} group.")
    else:
        print("Sorry, the animal could not be classified.")

if __name__ == "__main__":
    main()

```

OUTPUT:

```

Enter the type of animal(mammal or bird):bird
size:large
coat:no
feathers:brown
Beak:long
tail:no
legs:2
the animal is ostrich and belongs to brid group.

```

```

Enter the name of animal:ostrich
the ostrich is a bird that lives in the savanna and makes a boom sound.Its beak is long.

```

RESULT:

Thus an expert system for animal classification with property inheritance was implemented successfully.

EX.NO.10:

FORWARD CHAINING

Production Rule System:

A production rule system is given a list of rules and a list of data.

The rules look for certain things in the data -- these things are the antecedents of the rules -- and usually produce a new piece of data, called the consequent. Rules can also delete existing data. Importantly, rules can contain variables. This allows a rule to match more than one possible datum. The consequent can contain variables that were bound in the antecedent.

A rule is an expression that contains certain keywords, like IF, THEN, AND, OR, and NOT. An example of a rule looks like this:

```
IF( AND( 'parent (?x) (?y)',  
        'parent (?x) (?z)' ),  
    THEN( 'sibling (?y) (?z)' ))
```

This could be taken to mean:

If x is the parent of y, and x is the parent of z, then y is the sibling of z.

Given data that look like 'parent marge bart' and 'parent marge lisa', then, it will produce further data like 'sibling bart lisa'. (It will also produce 'sibling bart bart', which is something that will need to be dealt with.)

Of course, the rule system doesn't know what these arbitrary words "parent" and "sibling" mean! It doesn't even care that they're at the beginning of the expression. The rule could also be written like this:

```
IF (AND( '(?x) is a parent of (?y)',  
        '(?x) is a parent of (?z)' ),  
    THEN( '(?y) is a sibling of (?z)' ))
```

Then it will expect its data to look like 'marge is a parent of lisa'.

This gets wordy and involves some unnecessary matching of symbols like 'is' and 'a', and it doesn't help anything for this problem, but we'll write some later rule systems in this English-like way for clarity.

Rule Expressions:

Here's a more complete description of how the system works.

The rules are given in a specified order, and the system will check each rule in turn: for each rule, it will go through all the data searching for matches to that rule's antecedent, before moving on to the next rule.

A rule is an expression that can have an IF antecedent and a THEN consequent. Both of these parts are required. Optionally, a rule can also have a DELETE clause, which specifies some data to delete. The IF antecedent can contain AND, OR, and NOT expressions. AND requires that multiple statements are matched in the dataset, OR requires that one of multiple statements are matched in the dataset, and

NOT requires that a statement is not matched in the dataset. AND, OR, and NOT expressions can be nested within each other. When nested like this, these expressions form an AND/OR tree (or really an

AND/OR/NOT tree). At the bottom of this tree are strings, possibly with variables in them.

The data are searched for items that match the requirements of the antecedent. Data items that appear

earlier in the data take precedence. Each pattern in an AND clause will match the data in order, so that

later ones have the variables of the earlier ones.

If there is a NOT clause in the antecedent, the data are searched to make sure that no items in the data

match the pattern. A NOT clause should not introduce new variables - the matcher won't know what to

do with them. Generally, NOT clauses will appear inside an AND clause, and earlier parts of the AND

clause will introduce the variables. For example, this clause will match objects that are asserted to be

birds, but are not asserted to be penguins:

```
AND( '?x) is a bird',
```

```
NOT( '?x) is a penguin' ) )
```

The other way around won't work:

```
AND( NOT( '?x) is a penguin' ), # don't do this!
```

```
'?x) is a bird' )
```

The terms match and fire are important. A rule matches if its antecedent matches the existing data.

A

rule that matches can fire if its THEN or DELETE clauses change the data. (Otherwise, it fails to fire.)

Only one rule can fire at a time. When a rule successfully fires, the system changes the data appropriately, and then starts again from the first rule. This lets earlier rules take precedence over later

ones. (In other systems, the precedence order of rules can be defined differently.)

EX.NO.11: BACKWARD CHAINING AND GOAL TREES

Goal Trees:

For the next problem, we're going to need a representation of goal trees. Specifically, we want to make trees out of AND and OR nodes, much like the ones that can be in the antecedents of rules. (There won't be any NOT nodes.) They will be represented as AND() and OR() objects. Note that both 'AND' and

'OR' inherit from the built-in Python type 'list', so you can treat them just like lists.

Strings will be the leaves of the goal tree. For this problem, the leaf goals will simply be arbitrary symbols or numbers like g1 or 3.

An AND node represents a list of subgoals that are required to complete a particular goal.

If all the branches of an AND node succeed, the AND node succeeds. AND(g1, g2, g3) describes a goal that is completed by completing g1, g2, and g3 in order.

An OR node is a list of options for how to complete a goal. If any one of the branches of an OR node

succeeds, the OR node succeeds. OR(g1, g2, g3) is a goal that you complete by first trying g1, then g2, then g3.

Unconditional success is represented by an AND node with no requirements: AND().
Unconditional

failure is represented by an OR node with no options: OR().

A problem with goal trees is that you can end up with trees that are described differently but mean exactly the same thing. For example, AND(g1, AND(g2, AND(AND(), g3, g4))) is more reasonably expressed as AND(g1, g2, g3, g4). So, we've provided you a function that reduces some of these cases to the same tree. We won't change the order of any nodes, but we will prune some

nodes that it is fruitless to check.

We have provided this code for you. You should still understand what it's doing, because you can benefit

from its effects. You may want to write code that produces "messy", unsimplified goal trees, because it's

easier, and then simplify them with the simplify function.

Simplification Of Goal Trees:

This is how we simplify goal trees:

1. If a node contains another node of the same type, absorb it into the parent node. So OR(g1, OR(g2, g3), g4) becomes OR(g1 g2 g3 g4).

Any AND node that contains an unconditional failure (OR) has no way to succeed, so replace it with unconditional failure.

3. Any OR node that contains an unconditional success (AND) will always succeed, so replace it with unconditional success.

4. If a node has only one branch, replace it with that branch. AND(g1), OR(g1), and g1 all represent the same goal.

5. If a node has multiple instances of a variable, replace these with only one instance. AND(g1, g1, g2) is the same as AND(g1, g2).

We've provided an abstraction for AND and OR nodes, and a function that simplifies them, in

production.py. There is nothing for you to code in this section, but please make sure to understand this representation, because you're going to be building goal trees in the next section. Some examples:

```
simplify(OR(1, 2, AND())) =>
AND()
simplify(OR(1, 2, AND(3, AND(4)), AND(5))) =>
OR(1, 2, AND(3, 4), 5)
simplify(AND('g1', AND('g2', AND('g3', AND('g4', AND()))))) =>
AND('g1', 'g2', 'g3', 'g4')
simplify(AND('g')) => 'g'
simplify(AND('g1', 'g1', 'g2')) =>
AND('g1', 'g2')
```

Backward Chaining:

Backward chaining is running a production rule system in reverse. You start with a conclusion, and then you see what statements would lead to it, and test to see if those statements are true.

In this problem, we will do backward chaining by starting from a conclusion, and generating a goal tree of all the statements we may need to test. The leaves of the goal tree will be statements like 'opus

swims', meaning that at that point we would need to find out whether we know that Opus swims or not.

We'll run this backward chainer on the ZOOKEEPER system of rules, a simple set of production rules

for classifying animals, which you will find in zookeeper.py. As an example, here is the goal tree generated for the hypothesis 'opus is a penguin':

```
OR(
  'opus is a penguin',
  AND(
    OR('opus is a bird', 'opus has feathers', AND('opus flies', 'opus
lays eggs'))
    'opus does not fly',
    'opus swims',
    'opus has black and white color' ))
```

You will write a procedure, `backchain_to_goal_tree(rules, hypothesis)`, which outputs the goal tree.

The rules you work with will be limited in scope, because general-purpose backward chainers are difficult to write. In particular:

- ☐ You will never have to test a hypothesis with unknown variables. All variables that appear in the antecedent will also appear in the consequent.
- ☐ All assertions are positive: no rules will have DELETE parts or NOT clauses.
- ☐ Antecedents are not nested. Something like `(OR (AND x y) (AND z w))` will not appear in the antecedent parts of rules.

Note that an antecedent can be a single hypothesis (a string) or a RuleExpression.

The Backward Chaining Process:

Here's the general idea of backward chaining:

- Given a hypothesis, you want to see what rules can produce it, by matching the consequents of those rules against your hypothesis. All the consequents that match are possible options, so you'll collect their results together in an OR node. If there are no matches, this statement is a leaf, so output it as a leaf of the goal tree.
- If a consequent matches, keep track of the variables that are bound. Look up the antecedent of that rule, and instantiate those same variables in the antecedent (that is, replace the variables with their values). This instantiated antecedent is a new hypothesis.
- The antecedent may have AND or OR expressions. This means that the goal tree for the antecedent is already partially formed. But you need to check the leaves of that AND-OR tree, and recursively backward chain on them.

Other requirements:

- The branches of the goal tree should be in order: the goal trees for earlier rules should appear before (to the left of) the goal trees for later rules. Intermediate nodes should appear before their expansions.
- The output should be simplified as in the previous problem (you can use the simplify function). This way, you can create the goal trees using an unnecessary number of OR nodes, and they will be conglomerated together nicely in the end.
- If two different rules tell you to check the same hypothesis, the goal tree for that hypothesis should be included both times, even though it seems a bit redundant.