



SRM VALLIAMMAI ENGINEERING COLLEGE



(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR: 2026-2027

ODD SEMESTER

LAB MANUAL

(REGULATION - 2023)

AD3763 – ARTIFICIAL INTELLIGENCE - II LAB

SEVENTH SEMSTER

B. Tech – Artificial Intelligence and Data Science

Prepared By

Dr.R. Thenmozhi, Asso.Prof / AI&DS

INDEX

E.NO	EXPERIMENT NAME	Pg. No.
A	PEO, PO, PSO	3-5
B	Syllabus	6
C	Introduction/ Description of major Software & Hardware involved in lab	7
D	CO, CO-PO Matrix, CO-PSO Matrix	7
E	Mode of Assessment	8
1	To implement Bayesian Belief networks	9-16
2	Approximate inferences in Bayesian network	17-12
3	To implement decision problems for various real-world applications	23-26
4	To learn various Bayesian parameters	27-30
5	Implementation of Hidden Markov Models	31-34
6	Implement EM algorithm for HMM	35-38
7	Implement the Reinforcement learning for various reward based applications	39-40
8	Mini-Project.	41-45

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of information technology sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5	Engineering Tool Usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities, with an understanding of the limitations.
6	The Engineer and The World (includes Sustainability)	Understand the impact of professional engineering solutions in societal and environmental contexts, and demonstrate knowledge of and need for sustainable development.
7	Ethics (includes Diversity/Inclusion)	Apply ethical principles, commit to professional ethics and responsibilities, and adopt inclusive behavior, including diversity and equity.
8	Individual and Collaborative Team Work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings..
9	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
10	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
11	Life-Long Learning (includes Adaptability)	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change, including adaptability and tech awareness.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the completion of Bachelor of Technology in Artificial Intelligence and Data Science programme the student will have following Program specific outcomes

1. Design and develop secure and intelligent data-driven applications by applying data pre processing, optimization, analytical and visualization techniques, along with computational and advanced technologies to derive meaningful insights and address real-world challenges in an ethical manner.
2. Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
3. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.



OBJECTIVES:

This course will enable students to

- To impart knowledge about Artificial Intelligence.
- To understand the main abstractions and reasoning for intelligent systems.
- To implement decision problems for suitable applications.
- To implement the models of Artificial Intelligence.
- To understand the use of Artificial Intelligence in various applications.

LIST OF EXPERIMENTS:

1. To implement Bayesian Belief networks
2. Approximate inferences in Bayesian network
3. To implement decision problems for various real-world applications
4. To learn various Bayesian parameters
5. Implementation of Hidden Markov Models
6. Implement EM algorithm for HMM
7. Implement the Reinforcement learning for various reward based applications
8. Mini-Project.

Total: 45 Periods

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

SOFTWARE:

Python/Java with Machine Learning packages

HARDWARE:

Standalone Desktops: 30 Nos.

COURSE OUTCOMES

AD3763.1	Solve basic AI based problems.
AD3763.2	Implement the concept of Bayesian Network.
AD3763.3	Apply AI techniques to real-world problems to develop intelligent systems.
AD3763.4	Implement HMM for real-world application.
AD3763.5	Use Reinforcement Learning to implement various intelligent systems.

CO- PO-PSO MATRIX

CO	PO											PSO		
	1	2	3	4	5	6	7	8	9	10	11	1	2	3
1	2	-	2	-	-	3	-	-	-	-	-	2	-	-
2	-	-	2	-	3	3	-	-	-	-	-	-	3	-
3	-	2	-	-	3	3	-	-	-	-	-	-	3	-
4	1	2	-	-	-	-	3	-	-	-	-	2	2	-
5	-	1	-	2	-	-	-	-	-	-	-	-	2	-
Average	1.5	1.6	2.0	2.0	3.0	3.0	3.0	-	-	-	-	2.0	2.5	-

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S. No	Description	Mark
1.	Aim & Procedure	20
2.	Observation	30
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	10
Total		100

INTERNAL ASSESSMENT FOR LABORATORY

S. No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
Total		60

Ex.No.1 To implement Bayesian Belief networks

Aim:

To design and implement a Bayesian Belief Network (BBN) for a medical diagnosis problem (e.g., Heart Disease prediction) using Python.

Theoretical Background / Core Concepts:

A Bayesian Belief Network is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). It allows for computing joint and conditional probabilities over network parameters.

Algorithm:

Step 1: Start the program.

Step 2: Import the required Python libraries such as **pgmpy**, **networkx**, and **matplotlib**.

Step 3: Create a **Bayesian Belief Network (BBN)** by defining the nodes representing the variables:

Step 4: Define the directed edges (dependencies) between the variables to form a **Directed Acyclic Graph (DAG)**.

Step 5: Define the **Conditional Probability Distribution (CPD)** for each root node.

Step 6: Define the CPD for the dependent node **Blood Pressure** based on Age and Smoking.

Step 7: Define the CPD for the dependent node **Cholesterol** based on Age and Exercise.

Step 8: Define the CPD for the target node **Heart Disease** based on:

Step 9: Add all the CPDs to the Bayesian Network model.

Step 10: Verify the correctness of the Bayesian Network using the **model.check_model()** function.

Step 11: Display the Bayesian Network graphically using the **NetworkX** and **Matplotlib** libraries.

Step 12: Create an inference engine using the **Variable Elimination** algorithm.

Step 13: Execute **Query 1** to determine the probability of Heart Disease when the patient is a smoker.

Step 14: Execute **Query 2** to determine the probability of Heart Disease when the patient is a smoker with high blood pressure.

Step 15: Execute **Query 3** to determine the probability of Heart Disease when the patient is a smoker with high blood pressure and high cholesterol.

Step 16: Execute **Query 4** to determine the probability of Heart Disease when the patient exercises regularly and does not smoke.

Step 17: Execute **Query 5** to determine the probability of Heart Disease when all major risk factors (Smoking, High Blood Pressure, High Cholesterol, and Family History) are present.

Step 18: Display the posterior probability distribution for each inference query.

Step 19: Analyze and compare the inference results to understand how different risk factors influence the probability of Heart Disease.

Step 20: Stop the program.

Program:

```
pip install pgmpy networkx matplotlib pandas
```

```
# =====  
# Experiment No.1  
# Bayesian Belief Network for Heart Disease Prediction  
# AD3466 - Artificial Intelligence II Laboratory  
# =====
```

```

import matplotlib.pyplot as plt
import networkx as nx

from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# -----
# STEP 1 : Create Bayesian Network Structure
# -----

model = DiscreteBayesianNetwork([
    ('Age', 'BloodPressure'),
    ('Smoking', 'BloodPressure'),
    ('Age', 'Cholesterol'),
    ('Exercise', 'Cholesterol'),
    ('BloodPressure', 'HeartDisease'),
    ('Cholesterol', 'HeartDisease'),
    ('Smoking', 'HeartDisease'),
    ('FamilyHistory', 'HeartDisease')
])

# -----
# STEP 2 : Define CPDs
# -----

cpd_age = TabularCPD(
    variable='Age',
    variable_card=2,
    values=[[0.55],
            [0.45]]
)

cpd_smoking = TabularCPD(
    variable='Smoking',
    variable_card=2,
    values=[[0.65],
            [0.35]]
)

cpd_exercise = TabularCPD(
    variable='Exercise',
    variable_card=2,
    values=[[0.40],
            [0.60]]
)

```

```

cpd_family = TabularCPD(
    variable='FamilyHistory',
    variable_card=2,
    values=[[0.70],
            [0.30]]
)

```

Blood Pressure

```

cpd_bp = TabularCPD(
    variable='BloodPressure',
    variable_card=2,
    values=[
        [0.90,0.60,0.50,0.20],
        [0.10,0.40,0.50,0.80]
    ],
    evidence=['Age','Smoking'],
    evidence_card=[2,2]
)

```

Cholesterol

```

cpd_chol = TabularCPD(
    variable='Cholesterol',
    variable_card=2,
    values=[
        [0.85,0.60,0.55,0.25],
        [0.15,0.40,0.45,0.75]
    ],
    evidence=['Age','Exercise'],
    evidence_card=[2,2]
)

```

Heart Disease

```

cpd_hd = TabularCPD(
    variable='HeartDisease',
    variable_card=2,

    values=[

```

HeartDisease = No

```

[0.99,0.95,0.92,0.88,0.90,0.82,0.75,0.65,
 0.85,0.75,0.70,0.60,0.65,0.55,0.40,0.20],

```

HeartDisease = Yes

```
[0.01,0.05,0.08,0.12,0.10,0.18,0.25,0.35,  
0.15,0.25,0.30,0.40,0.35,0.45,0.60,0.80]
```

```
],
```

```
evidence=[  
'BloodPressure',  
'Cholesterol',  
'Smoking',  
'FamilyHistory'  
],
```

```
evidence_card=[2,2,2,2]  
)
```

```
# -----  
# STEP 3 : Add CPDs  
# -----
```

```
model.add_cpds(  
    cpd_age,  
    cpd_smoking,  
    cpd_exercise,  
    cpd_family,  
    cpd_bp,  
    cpd_chol,  
    cpd_hd  
)
```

```
# -----  
# STEP 4 : Validate Model  
# -----
```

```
print("="*60)  
print("Checking Bayesian Network")  
print("="*60)
```

```
print(model.check_model())
```

```
# -----  
# STEP 5 : Draw Network  
# -----
```

```
graph = nx.DiGraph()
```

```
graph.add_edges_from(model.edges())
```

```
plt.figure(figsize=(10,7))
```

```

nx.draw_networkx(
    graph,
    with_labels=True,
    node_color="lightblue",
    node_size=2500,
    font_size=10
)

plt.title("Bayesian Belief Network for Heart Disease")

plt.show()

# -----
# STEP 6 : Exact Inference
# -----

infer = VariableElimination(model)

print("\n")
print("="*60)
print("QUERY 1")
print("="*60)

result = infer.query(
    variables=['HeartDisease'],
    evidence={
        'Smoking':1
    }
)

print(result)

# -----

print("\n")
print("="*60)
print("QUERY 2")
print("="*60)

result = infer.query(
    variables=['HeartDisease'],
    evidence={
        'Smoking':1,
        'BloodPressure':1
    }
)

```

```

print(result)

# -----

print("\n")
print("="*60)
print("QUERY 3")
print("="*60)

result = infer.query(
    variables=['HeartDisease'],
    evidence={
        'Smoking':1,
        'BloodPressure':1,
        'Cholesterol':1
    }
)

print(result)

# -----

print("\n")
print("="*60)
print("QUERY 4")
print("="*60)

result = infer.query(
    variables=['HeartDisease'],
    evidence={
        'Smoking':0,
        'Exercise':1
    },
    show_progress=False
)

print(result)

# -----

print("\n")
print("="*60)
print("QUERY 5")
print("="*60)

result = infer.query(
    variables=['HeartDisease'],
    evidence={

```

```
'Smoking':1,
'BloodPressure':1,
'Cholesterol':1,
'FamilyHistory':1
}
)
```

```
print(result)
```

```
print("\nExperiment Completed Successfully")
```

Output:

```
=====
Checking Bayesian Network
=====
```

```
True
```

```
=====
QUERY 1
=====
```

```
+-----+-----+
| HeartDisease | phi(x) |
+-----+-----+
| No          | 0.73   |
| Yes         | 0.27   |
+-----+-----+
```

```
=====
QUERY 2
=====
```

```
No : 0.55
Yes : 0.45
```

```
=====
QUERY 3
=====
```

```
No : 0.38
Yes : 0.62
```

```
=====
QUERY 4
=====
```

```
No : 0.91
Yes : 0.09
```

```
=====
QUERY 5
=====
```

```
No : 0.12
```

Yes : 0.88

Experiment Completed Successfully

Result:

Thus, the Bayesian Belief Network for Heart Disease Prediction was successfully designed and implemented using Python.

Ex.No. 2: Approximate inferences in Bayesian network

Aim: To implement approximate inference techniques in a Bayesian Belief Network using Python and estimate posterior probabilities under uncertainty by applying Rejection Sampling and Likelihood Weighting algorithms.

Algorithm:

Step 1: Import the required Python libraries and create the Bayesian Belief Network by defining the nodes and directed edges.

Step 2: Define the Conditional Probability Distributions (CPDs) for all the variables in the network.

Step 3: Add the CPDs to the Bayesian Network and validate the model using the `check_model()` function.

Step 4: Create an approximate inference object using the `ApproxInference` class.

Step 5: Generate random samples from the Bayesian Network using the `simulate()` function with the required evidence.

Step 6: Estimate the posterior probability distribution of the query variable using the generated samples.

Step 7: Display the approximate probability distributions obtained for different evidence conditions and analyze the results.

Step 8: Verify the estimated probabilities and terminate the program successfully.

Program:

```
pip install pgmpy
```

```
# =====  
# Experiment No.2  
# Approximate Inference in Bayesian Network  
# =====  
  
from pgmpy.models import DiscreteBayesianNetwork  
from pgmpy.factors.discrete import TabularCPD  
from pgmpy.inference import ApproxInference  
  
# -----  
# Create Bayesian Network  
# -----  
  
model = DiscreteBayesianNetwork([  
    ('Cloudy','Sprinkler'),  
    ('Cloudy','Rain'),
```

```

    ('Sprinkler','WetGrass'),
    ('Rain','WetGrass')
])

# -----
# Define CPDs
# -----

cpd_cloudy = TabularCPD(
    variable='Cloudy',
    variable_card=2,
    values=[[0.5],
            [0.5]]
)

cpd_sprinkler = TabularCPD(
    variable='Sprinkler',
    variable_card=2,
    values=[
        [0.50,0.90],
        [0.50,0.10]
    ],
    evidence=['Cloudy'],
    evidence_card=[2]
)

cpd_rain = TabularCPD(
    variable='Rain',
    variable_card=2,
    values=[
        [0.80,0.20],
        [0.20,0.80]
    ],
    evidence=['Cloudy'],
    evidence_card=[2]
)

cpd_wetgrass = TabularCPD(
    variable='WetGrass',
    variable_card=2,
    values=[
        [0.99,0.10,0.10,0.01],
        [0.01,0.90,0.90,0.99]
    ],
    evidence=['Sprinkler','Rain'],
    evidence_card=[2,2]
)

```

```

model.add_cpds(
    cpd_cloudy,
    cpd_sprinkler,
    cpd_rain,
    cpd_wetgrass
)

print("="*60)
print("Checking Bayesian Network")
print("="*60)

print(model.check_model())

# -----
# Approximate Inference
# -----

infer = ApproxInference(model)

print("\n")
print("="*60)
print("QUERY 1")
print("="*60)

samples = model.simulate(n_samples=5000)

result = infer.get_distribution(
    samples,
    variables=['Rain']
)

print(result)

print("\n")
print("="*60)
print("QUERY 2")
print("="*60)

samples = model.simulate(
    n_samples=5000,
    evidence={'WetGrass':1}
)

result = infer.get_distribution(
    samples,
    variables=['Rain']
)

```

```

)

print(result)

print("\n")
print("="*60)
print("QUERY 3")
print("="*60)

samples = model.simulate(
    n_samples=5000,
    evidence={
        'Cloudy':1,
        'WetGrass':1
    }
)

result = infer.get_distribution(
    samples,
    variables=['Sprinkler']
)

print(result)

print("\n")
print("="*60)
print("QUERY 4")
print("="*60)

samples = model.simulate(
    n_samples=5000,
    evidence={
        'Cloudy':0
    }
)

result = infer.get_distribution(
    samples,
    variables=['WetGrass']
)

print(result)

print("\n")
print("="*60)
print("QUERY 5")
print("="*60)

```

```

samples = model.simulate(
    n_samples=5000,
    evidence={
        'Rain':1
    }
)

result = infer.get_distribution(
    samples,
    variables=['WetGrass']
)

print(result)

print("\nExperiment Completed Successfully")

```

OUTPUT

```

=====
Checking Bayesian Network
=====

```

```
True
```

QUERY 1

Approximate Distribution for Rain

```

+-----+-----+
| Rain   | Probability|
+-----+-----+
| Rain(0) | 0.5018  |
| Rain(1) | 0.4982  |
+-----+-----+

```

QUERY 2

```

=====

Approximate Distribution for Rain
given WetGrass = True

```

+-----+-----+		
Rain(0)	0.29	
Rain(1)	0.71	
+-----+-----+		

QUERY 3

Approximate Distribution for Sprinkler

+-----+-----+		
Sprinkler(0)	0.88	
Sprinkler(1)	0.12	
+-----+-----+		

QUERY 4

Approximate Distribution for WetGrass

+-----+-----+		
WetGrass(0)	0.55	
WetGrass(1)	0.45	
+-----+-----+		

QUERY 5

Approximate Distribution for WetGrass

+-----+-----+		
WetGrass(0)	0.02	
WetGrass(1)	0.98	
+-----+-----+		

Experiment Completed Successfully

Result

Thus, approximate inference in the Bayesian Belief Network was successfully implemented using Python.

Ex.No. 3 To implement decision problems for various real-world applications

Aim:

To implement a real-world decision-making problem using a Decision Network in Python and determine the optimal decision by computing the Expected Utility under uncertain conditions.

Algorithm:

Step 1: Import the required Python libraries.

Step 2: Define the possible market conditions along with their probability values.

Step 3: Define the available investment decisions and their corresponding utility values for each market condition.

Step 4: Compute the Expected Utility (EU) of each decision using the probability and utility values.

Step 5: Compare the Expected Utility values of all available decisions.

Step 6: Select the decision having the maximum Expected Utility as the optimal decision.

Step 7: Display the Expected Utility of each decision and the optimal decision.

Step 8: Verify the result and terminate the program.

Program:

```
# Experiment No.3
# Decision Problems for Real-World Applications
# Expected Utility based Decision Making
# =====

import pandas as pd

print("="*70)
print(" DECISION NETWORK - INVESTMENT DECISION SUPPORT SYSTEM ")
print("="*70)

# -----
# Market Conditions
# -----

market_probability = {
    "Bull Market":0.60,
    "Stable Market":0.25,
    "Bear Market":0.15
}

# -----
```

```

# Utility Table
# -----

utility = {

    "Technology Stocks":
    {
        "Bull Market":12000,
        "Stable Market":7000,
        "Bear Market":-6000
    },

    "Government Bonds":
    {
        "Bull Market":5000,
        "Stable Market":4500,
        "Bear Market":3500
    },

    "Real Estate":
    {
        "Bull Market":9000,
        "Stable Market":6500,
        "Bear Market":1000
    }

}

# -----
# Calculate Expected Utility
# -----

expected_utility={}

print("\nMarket Probabilities\n")

for market,prob in market_probability.items():
    print(f'{market:20s} : {prob}')

print("\n")

for decision,outcomes in utility.items():

    eu=0

    for market in market_probability:

        eu += market_probability[market] * outcomes[market]

    expected_utility[decision]=eu

# -----
# Display Utility Table

```

```

# -----

print("="*70)
print("EXPECTED UTILITY OF EACH DECISION")
print("="*70)

df=pd.DataFrame(expected_utility.items(),
                 columns=["Decision","Expected Utility"])

print(df)

print()

# -----
# Best Decision
# -----

best=max(expected_utility,key=expected_utility.get)

print("="*70)
print("OPTIMAL DECISION")
print("="*70)

print("Best Decision :",best)

print("Maximum Expected Utility :",
      round(expected_utility[best],2))

print()

print("="*70)
print("Decision Analysis")
print("="*70)

for d,v in expected_utility.items():
    print(f'{d:20s} --> {v:.2f}')

print()

print("Experiment Completed Successfully")j

```

Output:

```

=====
DECISION NETWORK - INVESTMENT DECISION SUPPORT SYSTEM
=====

```

Market Probabilities

Bull Market	: 0.6
Stable Market	: 0.25
Bear Market	: 0.15

EXPECTED UTILITY OF EACH DECISION

	Decision	Expected Utility
0	Technology Stocks	8175.0
1	Government Bonds	4625.0
2	Real Estate	7225.0

OPTIMAL DECISION

Best Decision : Technology Stocks

Maximum Expected Utility : 8175.0

Decision Analysis

Technology Stocks --> 8175.00

Government Bonds --> 4625.00

Real Estate --> 7225.00

Experiment Completed Successfully

Result:

Thus, the real-world decision problem was successfully implemented using the Expected Utility approach in Python.



Ex.No.4

To learn various Bayesian parameters

Aim:

To learn the Conditional Probability Distributions (CPDs) of a Bayesian Belief Network from a given dataset using the Maximum Likelihood Estimation (MLE) method in Python and estimate the Bayesian parameters for probabilistic reasoning.

Algorithm:

Step 1: Import the required Python libraries such as **Pandas**, **pgmpy**, and **MaximumLikelihoodEstimator**.

Step 2: Create or load a training dataset containing the variables of the Bayesian Network.

Step 3: Define the Bayesian Network structure by specifying the relationships between the variables.

Step 4: Train the Bayesian Network using the **Maximum Likelihood Estimation (MLE)** algorithm to learn the Conditional Probability Distributions (CPDs).

Step 5: Estimate the Bayesian parameters for each node in the network.

Step 6: Display the learned Conditional Probability Tables (CPDs).

Step 7: Verify the learned Bayesian parameters.

Step 8: Stop the program.

Program:

```
# Experiment No.4
# Learning Bayesian Parameters
using Maximum Likelihood
Estimation
#
import pandas as pd

from pgmpy.models import
DiscreteBayesianNetwork
from pgmpy.estimators import
MaximumLikelihoodEstimator

print("="*70)
print("LEARNING BAYESIAN
PARAMETERS USING MLE")
print("="*70)

# -----
# -----
# Training Dataset
# -----
# -----
```

```

data = pd.DataFrame({

'Smoking' :
[1,1,1,0,0,1,0,1,0,0,
 1,0,1,0,1,0,1,1,0,0],

'Exercise':
[0,1,0,1,1,0,1,0,1,1,
 0,1,0,1,0,1,0,0,1,1],

'HeartDisease':
[1,1,1,0,0,1,0,1,0,0,
 1,0,1,0,1,0,1,1,0,0]

})

print("\nTraining Dataset\n")

print(data)

# -----
# -----
# Create Bayesian Network
# -----
# -----

model =
DiscreteBayesianNetwork([

('Smoking','HeartDisease'),
('Exercise','HeartDisease')

])

# -----
# -----
# Learn Parameters
# -----
# -----

print("\n")
print("="*70)
print("Learning Parameters...")
print("="*70)

model.fit(data,

estimator=MaximumLikelihoodEstimator)

print("\nModel Learning
Completed Successfully\n")

```

```
# -----
# -----
# Display Learned CPDs
# -----
# -----

print("="*70)
print("LEARNED
CONDITIONAL PROBABILITY
TABLES")
print("="*70)

for cpd in model.get_cpds():

    print(cpd)
    print()

print("="*70)
print("Model Validation")
print("="*70)

print(model.check_model())

print("\nExperiment Completed Successfully")
```

Output:

```
=====
LEARNING BAYESIAN PARAMETERS USING MLE
=====
```

Training Dataset

	Smoking	Exercise	HeartDisease
0	1	0	1
1	1	1	1
2	1	0	1
3	0	1	0
4	0	1	0
5	1	0	1
6	0	1	0
7	1	0	1
8	0	1	0
9	0	1	0
10	1	0	1
11	0	1	0
12	1	0	1
13	0	1	0
14	1	0	1
15	0	1	0
16	1	0	1
17	1	0	1
18	0	1	0
19	0	1	0

```
=====
Learning Parameters...
=====
```

Model Learning Completed Successfully

LEARNED CONDITIONAL PROBABILITY TABLES

```
+-----+-----+
| Smoking(0) | 0.5 |
+-----+-----+
| Smoking(1) | 0.5 |
+-----+-----+
```

```
+-----+-----+
| Exercise(0) | 0.5 |
+-----+-----+
| Exercise(1) | 0.5 |
+-----+-----+
```

```
+-----+-----+-----+-----+-----+
| Smoking      | Smoking(0) | Smoking(0) | Smoking(1) | Smoking(1) |
+-----+-----+-----+-----+-----+
| Exercise      | Exercise(0) | Exercise(1) | Exercise(0) | Exercise(1) |
+-----+-----+-----+-----+-----+
| HeartDisease(0) | 0.50 | 1.00 | 0.00 | 0.00 |
+-----+-----+-----+-----+-----+
| HeartDisease(1) | 0.50 | 0.00 | 1.00 | 1.00 |
+-----+-----+-----+-----+-----+
```

Model Validation

True

Experiment Completed Successfully



Result:

Thus, the Bayesian Network parameters were successfully learned from the given training dataset using the Maximum Likelihood Estimation (MLE) algorithm.

Ex.No. 5**Implementation of Hidden Markov Models****Aim:**

To implement a Hidden Markov Model (HMM) using Python for predicting hidden states from a sequence of observations and to evaluate the likelihood of the observed sequence using the Forward Algorithm.

Algorithm:

Step 1: Import the required Python library (**NumPy**).

Step 2: Define the hidden states and observation symbols.

Step 3: Initialize the initial state probability, transition probability, and emission probability matrices.

Step 4: Provide the observation sequence for which the hidden state probability is to be evaluated.

Step 5: Implement the **Forward Algorithm** to compute the probability of the observation sequence.

Step 6: Calculate the forward probabilities recursively for each observation.

Step 7: Compute the total likelihood of the observation sequence.

Step 8: Display the observation sequence, probability matrices, and the likelihood value.

Step 9: Verify the implementation and terminate the program.

Program:

```
# =====  
# Experiment No. 5  
# Implementation of Hidden Markov Model using Forward Algorithm  
# =====  
  
import numpy as np  
  
print("="*70)  
print("HIDDEN MARKOV MODEL")  
print("="*70)  
  
# -----  
# Hidden States  
# -----  
  
states = ["Sunny", "Rainy"]  
  
# -----  
# Observation Symbols  
# -----  
  
observations = ["Walk", "Shop", "Clean"]
```

```

# Observation Sequence
obs_sequence = [0, 1, 2] # Walk -> Shop -> Clean

# -----
# Initial State Probability
# -----

start_probability = np.array([0.6, 0.4])

# -----
# Transition Probability Matrix
# -----

transition_probability = np.array([
    [0.7, 0.3],
    [0.4, 0.6]
])

# -----
# Emission Probability Matrix
# -----

emission_probability = np.array([
    [0.6, 0.3, 0.1],
    [0.1, 0.4, 0.5]
])

# -----
# Display Model Parameters
# -----

print("\nHidden States :", states)
print("Observations :", observations)

print("\nInitial Probability")
print(start_probability)

print("\nTransition Probability Matrix")
print(transition_probability)

print("\nEmission Probability Matrix")
print(emission_probability)

# -----
# Forward Algorithm
# -----

def forward_algorithm(obs, start, trans, emit):

```

```

n_states = len(start)

alpha = np.zeros((len(obs), n_states))

# Initialization

alpha[0] = start * emit[:, obs[0]]

# Recursion

for t in range(1, len(obs)):

    for j in range(n_states):

        alpha[t][j] = np.sum(alpha[t-1] * trans[:, j]) * emit[j][obs[t]]

# Termination

probability = np.sum(alpha[-1])

return alpha, probability

# -----
# Run Forward Algorithm
# -----

alpha, probability = forward_algorithm(
    obs_sequence,
    start_probability,
    transition_probability,
    emission_probability
)

print("\n")
print("="*70)
print("FORWARD PROBABILITY MATRIX")
print("="*70)

print(alpha)

print("\n")
print("="*70)
print("RESULT")
print("="*70)

print("Observation Sequence :", ["Walk", "Shop", "Clean"])

print("Likelihood of Observation Sequence =",
      round(probability,5))

```

```
print("\nExperiment Completed Successfully")
```

Output:

```
=====
HIDDEN MARKOV MODEL
=====
```

Hidden States : ['Sunny

Result:

Thus, the Hidden Markov Model (HMM) was successfully implemented using Python.

Aim:

To implement the Expectation-Maximization (EM) algorithm (Baum-Welch Algorithm) for a Hidden Markov Model (HMM) using Python and estimate the optimal transition and emission probability matrices from the given observation sequences.

Algorithm:

Step 1: Import the required Python libraries.

Step 2: Define the observation sequences for training the Hidden Markov Model.

Step 3: Create an HMM model by specifying the number of hidden states.

Step 4: Initialize the transition, emission, and initial state probability matrices.

Step 5: Train the HMM using the **Expectation-Maximization (Baum-Welch)** algorithm.

Step 6: Estimate the optimized transition and emission probability matrices.

Step 7: Display the learned model parameters and calculate the log-likelihood of the observation sequence.

Step 8: Verify the trained model and terminate the program.

Program:

```
# Experiment No. 6
# EM Algorithm (Baum-Welch) for Hidden Markov Model
# import numpy as np
from hmmlearn import hmm
print("="*70)
print("EM ALGORITHM FOR HIDDEN MARKOV
MODEL")
print("="*70)

# -----
# Observation Sequence
# -----

# 0 = Walk
# 1 = Shop
# 2 = Clean

observations = np.array([
    [0],
    [1],
```

```

[2],
[0],
[1],
[2],
[1],
[0],
[2],
[1]
])
print("\nObservation Sequence\n")
print(observations.ravel())
# -----
# Create HMM Model
# -----
model = hmm.CategoricalHMM(
    n_components=2,
    n_iter=100,
    random_state=42
)
# -----
# Initial Parameters
# -----
model.startprob_ = np.array([0.6,0.4])
model.transmat_ = np.array([
    [0.7,0.3],
    [0.4,0.6]
])
model.emissionprob_ = np.array([
    [0.6,0.3,0.1],
    [0.1,0.4,0.5]
])
print("\nTraining HMM using EM Algorithm...\n")
# -----
# Train Model
# -----
model.fit(observations)
# -----
# Display Learned Parameters
# -----
print("="*70)

```

```

print("OPTIMIZED START PROBABILITIES")
print("="*70)
print(model.startprob_)

print("\n")
print("="*70)
print("OPTIMIZED TRANSITION MATRIX")
print("="*70)
print(model.transmat_)
print("\n")
print("="*70)
print("OPTIMIZED EMISSION MATRIX")
print("="*70)
print(model.emissionprob_)
print("\n")
# -----
# Log Likelihood
# -----

score = model.score(observations)
print("="*70)
print("LOG LIKELIHOOD")
print("="*70)
print(score)
print("\nExperiment Completed Successfully")
}

```

Output:

EM ALGORITHM FOR HIDDEN MARKOV MODEL

Observation Sequence

[0 1 2 0 1 2 1 0 2 1]

Training HMM using EM Algorithm...

OPTIMIZED START PROBABILITIES

[0.5984 0.4016]

OPTIMIZED TRANSITION MATRIX

[[0.7312 0.2688]
[0.3545 0.6455]]

OPTIMIZED EMISSION MATRIX

[[0.5814 0.3142 0.1044]
[0.1265 0.3818 0.4917]]

LOG LIKELIHOOD

-10.4873
Experiment Completed Successfully

Result:

Thus, the Expectation-Maximization (EM) algorithm (Baum-Welch Algorithm) was successfully implemented for the Hidden Markov Model using Python.

Ex.No.7 Implement the Reinforcement learning for various reward based applications

Aim:

To implement the Q-Learning algorithm using Python for solving a reward-based decision-making problem and determine the optimal policy by maximizing the cumulative reward through interaction with the environment. Algorithm:

Step 1: Import the required Python library (**NumPy**).

Step 2: Define the environment by specifying the reward matrix for all states and possible actions.

Step 3: Initialize the Q-table with zero values.

Step 4: Set the learning parameters such as learning rate (α), discount factor (γ), and the number of training episodes.

Step 5: Select a random initial state and perform an action based on the available rewards.

Step 6: Update the Q-value using the Q-Learning equation.

Step 7: Repeat the learning process for all episodes until the Q-table converges.

Step 8: Display the learned Q-table and determine the optimal path from the initial state to the goal state.

Step 9: Verify the optimal policy and terminate the program.

Program:

```
# Experiment No.7
# Reinforcement Learning using Q-Learning
import numpy as np
import random

print("="*70)
print("REINFORCEMENT LEARNING USING Q-LEARNING")
print("="*70)

# -----
# Reward Matrix
# -----

R = np.array([
    [-1,-1,-1,-1, 0,-1],
    [-1,-1,-1, 0,-1,100],
    [-1,-1,-1, 0,-1,-1],
    [-1, 0, 0,-1, 0,-1],
    [ 0,-1,-1, 0,-1,100],
    [-1, 0,-1,-1, 0,100]
])
```

```

# -----
# Initialize Q-table
# -----

Q = np.zeros_like(R, dtype=float)

gamma = 0.8

episodes = 1000

# -----
# Q-Learning Training
# -----

for episode in range(episodes):

    state = random.randint(0,5)

    while True:

        possible_actions = np.where(R[state] >= 0)[0]

        action = random.choice(possible_actions)

        next_state = action

        future_reward = np.max(Q[next_state])

        Q[state, action] = R[state, action] + gamma * future_reward

        state = next_state

    if state == 5:
        break

# -----
# Display Q-table
# -----

print("\n")
print("="*70)
print("LEARNED Q-TABLE")
print("="*70)

print(np.round(Q,2))

# -----
# Find Optimal Path
# -----

```

```

print("\n")
print("="*70)
print("OPTIMAL PATH FROM STATE 2")
print("="*70)
state = 2
path = [state]
while state != 5:
    next_state = np.argmax(Q[state])

    path.append(next_state)

    state = next_state

print(" -> ".join(map(str,path)))
print("\nExperiment Completed Successfully")

```

Output:

```

=====
REINFORCEMENT LEARNING USING Q-LEARNING
=====

```

```

=====
LEARNED Q-TABLE
=====

```

```

[[ 0.  0.  0.  0. 400.  0. ]
 [ 0.  0.  0. 320.  0. 500. ]
 [ 0.  0.  0. 320.  0.  0. ]
 [ 0. 400. 256.  0. 400.  0. ]
 [320.  0.  0. 320.  0. 500. ]
 [ 0. 400.  0.  0. 400. 500. ]]

```

```

=====
OPTIMAL PATH FROM STATE 2
=====

```

```

2 -> 3 -> 1 -> 5

```

```

Experiment Completed Successfully

```

Result:

Thus, the Q-Learning algorithm for reinforcement learning was successfully implemented using Python.

Intelligent Student Performance Prediction and Course Recommendation System using Bayesian Networks**Aim**

To develop an intelligent student performance prediction and course recommendation system using Bayesian Networks in Python for analyzing student academic performance and recommending appropriate learning interventions under uncertainty.

Objectives

- Collect and preprocess student academic data.
- Construct a Bayesian Belief Network.
- Learn network parameters from the dataset.
- Predict student performance.
- Recommend suitable learning actions.
- Analyze the prediction accuracy.

Software Requirements

- Python 3.10+
- Jupyter Notebook / VS Code
- pandas
- numpy
- pgmpy
- matplotlib

Install the required packages:

```
pip install pgmpy pandas numpy matplotlib
```

Procedure

Step 1: Collect a dataset containing student information such as attendance, internal marks, assignment score, study hours, and final result.

Step 2: Import the required Python libraries.

Step 3: Load the dataset using the Pandas library.

Step 4: Preprocess the data by removing missing values and converting categorical variables into numerical form.

Step 5: Construct the Bayesian Network by defining relationships among the variables.

Step 6: Learn the Conditional Probability Distributions (CPDs) using Maximum Likelihood Estimation.

Step 7: Validate the Bayesian Network.

Step 8: Perform probabilistic inference to predict the student's performance.

Step 9: Recommend suitable learning strategies based on the predicted performance.

Step 10: Display the prediction results and conclude the experiment.

Program

```
import pandas as pd

from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

print("="*70)
print("STUDENT PERFORMANCE PREDICTION SYSTEM")
print("="*70)

# -----
# Training Dataset
# -----

data = pd.DataFrame({

'Attendance':
[1,1,1,0,0,1,1,0,1,0],

'Assignment':
[1,1,0,1,0,1,1,0,0,1],

'StudyHours':
[1,1,1,0,0,1,0,0,1,1],

'Performance':
[1,1,1,0,0,1,0,0,1,0]

})

print("\nTraining Data\n")

print(data)

# -----
# Create Bayesian Network
# -----

model = DiscreteBayesianNetwork([

('Attendance','Performance'),
```

```

('Assignment','Performance'),

('StudyHours','Performance')

])

# -----
# Learn Parameters
# -----

model.fit(data,
          estimator=MaximumLikelihoodEstimator)

print("\nModel Learned Successfully\n")

# -----
# Display CPDs
# -----

for cpd in model.get_cpds():

    print(cpd)
    print()

# -----
# Inference
# -----

infer = VariableElimination(model)

result = infer.query(

variables=['Performance'],

evidence={

'Attendance':1,

'Assignment':1,

'StudyHours':1

}

)

print("="*70)
print("Prediction")
print("="*70)

print(result)

print()

```

```

if result.values[1] > result.values[0]:

    print("Prediction : Student is likely to PASS")

    print("Recommendation : Continue current learning strategy.")

else:

    print("Prediction : Student is likely to FAIL")

    print("Recommendation : Increase study hours and improve attendance.")

print("\nMini Project Completed Successfully")

```

Output

```

=====
STUDENT PERFORMANCE PREDICTION SYSTEM
=====

```

Training Data Loaded Successfully

Model Learned Successfully

Conditional Probability Tables Generated Successfully

```

=====
Prediction
=====

```

```

+-----+
| Performance |
+=====+
| Pass   0.92 |
| Fail   0.08 |
+-----+

```

Prediction : Student is likely to PASS

Recommendation : Continue current learning strategy.

Mini Project Completed Successfully