Lab Report -3

Name: Md Yeasin Arafat

ID: 211-35-687

Section: A

Course name: Artificial Intelligence Lab

Course Code: SE334

Date: 18/11/2023

Problem Statement

In our lab class, we have talk about Bayesian Network. Where we were discuss about a person's has an appointment and that agent must attended the appointment. Whether that agent would be able to present at the appointment or not.

The agent will face issues of rain, on time of train, delayed of train.

We have to find possible result from all of the data given. We have to use Bayesian Network for presenting the probability of the agents, weather that agent would be able to attended or not.

interface.py

```
from model import model

# Calculate predictions
predictions = model.predict_proba({
    "train": "delayed"
})

# Print predictions for each node
for node, prediction in zip(model.states, predictions):
    if isinstance(prediction, str):
        print(f"{node.name}: {prediction}")
    else:
        print(f"{node.name}")
        for value, probability in prediction.parameters[0].items():
            print(f"    {value}: {probability:.4f}")
```

model.py

```
from lib2to3.pytree import Node

import BayesianNetwork
from pomegranate import *

# Rain node has no parents
rain = Node(DiscreteDistribution({
    "none": 0.7,
    "light": 0.2,
    "heavy": 0.1
}), name="rain")

# Track maintenance node is conditional on rain
maintenance = Node(ConditionalProbabilityTable([
    ["none", "yes", 0.4],
    ["none", "no", 0.6],
    ["light", "yes", 0.2],
    ["light", "no", 0.8],
     ["heavy", "yes", 0.1],
    ["heavy", "no", 0.9]
], [rain.distribution]), name="maintenance")

# Train node is conditional on rain and maintenance
train = Node(ConditionalProbabilityTable([
    ["none", "yes", "on time", 0.8],
    ["none", "yes", "delayed", 0.2],
    ["none", "no", "on time", 0.9],
      ["none", "no", "delayed", 0.1],
    ["light", "yes", "on time", 0.6],
    ["light", "yes", "delayed", 0.4],
    ["light", "no", "on time", 0.7],
    ["light", "no", "delayed", 0.3],
    ["heavy", "yes", "on time", 0.4],

    ["heavy", "yes", "delayed", 0.6],

     ["heavy", "no", "on time", 0.5],
    ["heavy", "no", "delayed", 0.5],
], [rain.distribution,
 maintenance.distribution]), name="train")

# Appointment node is conditional on train
appointment = Node(ConditionalProbabilityTable([
    ["on time", "attend", 0.9],
    ["on time", "miss", 0.1],
    ["delayed", "attend", 0.6],
    ["delayed", "miss", 0.4]
], [train.distribution]), name="appointment")

# Create a Bayesian Network and add states
model = BayesianNetwork()
model.add_states(rain, maintenance, train, appointment)

# Add edges connecting nodes
model.add_edge(rain, maintenance)
model.add_edge(rain, train)
model.add_edge(maintenance, train)
model.add_edge(train, appointment)

# Finalize model
model.bake()
```

likelihood.py

```
from model import model

# Calculate probability for a given observation
probability = model.probability([["none", "no", "on time", "attend"]])

print(probability)
```

sample.py

```
import pomegranate

from collections import Counter

from model import model

def generate_sample():

    # Mapping of random variable name to sample generated
    sample = {}

    # Mapping of distribution to sample generated
    parents = {}

    # Loop over all states, assuming topological order
    for state in model.states:

        # If we have a non-root node, sample conditional on parents
        if isinstance(state.distribution,
pomegranate.ConditionalProbabilityTable):
            sample[state.name] =
state.distribution.sample(parent_values=parents)
```

```
        # Otherwise, just sample from the distribution alone
        else:
            sample[state.name] = state.distribution.sample()

        # Keep track of the sampled value in the parents mapping
        parents[state.distribution] = sample[state.name]

    # Return generated sample
    return sample

# Rejection sampling
# Compute distribution of Appointment given that train is delayed
N = 10000
data = []
for i in range(N):
    sample = generate_sample()
    if sample["train"] == "delayed":
        data.append(sample["appointment"])
print(Counter(data))
```

Explaination of Code.

interface.py

The code is using a machine learning model (model) to make probalistic predictions. It takes an input data point related to a features named 'train', with the value 'delayed'. The script prints the pedictions for each node in the model. If the prediction is string, it prints the node's name and the predicted value else it implies probilistic predictions, and it prints the the node's name along with the probilisties for different values.

model.py

Nodes Creation:

Nodes are created for different events: 'rain', 'maintence', 'train' and 'appointment'. Each node is associated with a probability distribution describing the likelihood of its possible outcomes.

Conditional Probabilities

Conditional Probability Tables (CPTs) are defined for nodes that depend on others. For example, the probability ~~for~~ of 'maintence' depends on the state of 'rain'.

Bayesian Network Creation:

A Bayesian Network model is instantiated using 'BayesianNetwork()' from 'pomegranate'. Nodes are added to the model, representing different events.

Finalization:

Edges and nodes are finalized using the 'bake()' method, making the network ready for inference.

likelihood.py

Here the model is imported from 'model.py' where we used the 'pomegranate' library.

Then where 'probility' variable is assigned the result of calling the 'probality' method. on the model. The argument provided to 'probality' is a specific observation represented as a list of states for each node in the network: [none, no, ontime, attend]

Lastly calculated probability for the given observation is printed.

sample.py

Sample Generation Function:

Iterates over the nodes in the Bayesian Network model in a topological order. For each node, it generates a sample based on the distribution of that node conditional on its parent. The sampled values are stored in the 'sample' dictionary.

Rejection Sampling:

Repeats the sampling process ('generate_sample') a specified number of times (N).
Collects the 'Appointment' node value for each sample.

Result Printing:

Prints the counts of different 'Appointment' values given that the 'Train' is 'delayed' using the 'Counter' class.

```
        # Otherwise, just sample from the distribution alone
        else:
            sample[state.name] = state.distribution.sample()

        # Keep track of the sampled value in the parents mapping
        parents[state.distribution] = sample[state.name]

    # Return generated sample
    return sample

# Rejection sampling
# Compute distribution of Appointment given that train is delayed
N = 10000
data = []
for i in range(N):
    sample = generate_sample()
    if sample["train"] == "delayed":
        data.append(sample["appointment"])
print(Counter(data))
```

The output of this above code is unavailable due to the rewritten version of pomegranate library.