DiscreteBayesianNetwork – Detailed Explanation

The **DiscreteBayesianNetwork** class is part of the **pgmpy** library in Python, which is used to model **Bayesian Networks** where all the variables are **discrete**. It is an updated version of the older `BayesianNetwork` class, which is now deprecated. The main purpose of this class is to create probabilistic models that can perform **reasoning under uncertainty** using **categorical variables** like "Yes/No", "True/False", or levels such as "Low/Medium/High".

What is a Bayesian Network?

A **Bayesian Network** (also known as a **Belief Network**) is a **graphical model** that represents a set of variables and their conditional dependencies through a **Directed Acyclic Graph (DAG)**.

- Each **node** in the graph represents a **random variable**.
- Each **directed edge** shows a **causal or probabilistic relationship** between two variables.
- The network is "acyclic", meaning it does not contain any loops or cycles.

The Bayesian Network allows us to compute the **joint probability distribution** over all variables and to make **inferences** such as the probability of an event given some evidence.

Key Components of DiscreteBayesianNetwork

1. **Nodes (Variables):**
   These are the random variables involved in the model. They can take on values from a discrete set.
   Example: Burglary (Yes/No), Alarm (On/Off), Rain (True/False)
2. **Edges (Dependencies):**
   Directed links between variables that represent how one variable probabilistically depends on another.
   For example, "Burglary → Alarm" shows that the alarm depends on whether a burglary has happened.
3. **CPDs (Conditional Probability Distributions):**
   These are tables that define the probability of a variable, given the states of its parent variables.
   Example: P(Alarm = On | Burglary = Yes, Earthquake = No)

Functions and Features in DiscreteBayesianNetwork

- `add_nodes_from()` – Adds variables to the network.
- `add_edges_from()` – Defines relationships between variables.
- `add_cpds()` – Attaches CPD tables to variables.
- `check_model()` – Checks whether the model is valid and complete.
- `get_cpds()` – Retrieves CPDs for inspection.
- `get_parents()` / `get_children()` – Shows the dependencies for each variable.

Example: Burglary Alarm System

Consider a real-world situation where an alarm can go off because of either a burglary or an earthquake. If the alarm sounds, people like David or Sophia might call the police.

Variables:

- B = Burglary
- E = Earthquake
- A = Alarm
- D = David calls

- S = Sophia calls

Structure:
B → A ← E
A → D
A → S

Each variable has a probability distribution, and we can ask questions like:
"What is the probability that the alarm has sounded if there is no burglary and no earthquake?"

This kind of reasoning is made possible by **DiscreteBayesianNetwork**.

Applications of Discrete Bayesian Networks

- **Medical Diagnosis:**
  Predicting diseases based on symptoms (e.g., fever, cough, headache).
- **Fault Diagnosis:**
  Used in machinery, aircraft, and nuclear plants to detect errors based on observed signals.
- **Decision Support Systems:**
  In finance and marketing, Bayesian networks help assess risk and support strategic decisions.
- **Gene Expression Analysis:**
  In bioinformatics, to model and analyze relationships between genes and traits.
- **Spam Detection:**
  Determine whether an email is spam based on keywords and metadata.

Conclusion

DiscreteBayesianNetwork is a powerful tool for modeling uncertainty in real-world systems where variables are discrete. It helps in making predictions, understanding causes, and making informed decisions based on evidence. It is widely used in AI, machine learning, diagnostics, and decision-making systems.

By learning to implement and use this network, we gain practical knowledge of **probabilistic reasoning**, which is a core concept in artificial intelligence.

DiscreteBayesianNetwork – Detailed Explanation

The **DiscreteBayesianNetwork** class is part of the **pgmpy** library in Python, which is used to model **Bayesian Networks** where all the variables are **discrete**. It is an updated version of the older `BayesianNetwork` class, which is now deprecated. The main purpose of this class is to create probabilistic models that can perform **reasoning under uncertainty** using **categorical variables** like "Yes/No", "True/False", or levels such as "Low/Medium/High".

7/30/25, 11:42 PM AI-DS Exp1 - Colab

HARSHIT SHARMA/51

```
!pip install pgmpy
```

```
     24.6/24.6 MB 53.6 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 31.2 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.8 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.3 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 12.8 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.5 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.3 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 32.5 MB/s eta 0:00:00
Downloading pyro_api-0.1.2-py3-none-any.whl (11 kB)
Installing collected packages: pyro-api, nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, n
```

```
Attempting uninstall: nvidia-nvjitlink-cu12
 Found existing installation: nvidia-nvjitlink-cu12 12.5.82
 Uninstalling nvidia-nvjitlink-cu12-12.5.82:
 Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
 Attempting uninstall: nvidia-curand-cu12
 Found existing installation: nvidia-curand-cu12 10.3.6.82
 Uninstalling nvidia-curand-cu12-10.3.6.82:
 Successfully uninstalled nvidia-curand-cu12-10.3.6.82
 Attempting uninstall: nvidia-cufft-cu12
 Found existing installation: nvidia-cufft-cu12 11.2.3.61
 Uninstalling nvidia-cufft-cu12-11.2.3.61:
 Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
 Attempting uninstall: nvidia-cuda-runtime-cu12
 Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
 Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
 Attempting uninstall: nvidia-cuda-nvrtc-cu12
 Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
 Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
 Attempting uninstall: nvidia-cuda-cupti-cu12
 Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
 Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
 Attempting uninstall: nvidia-cublas-cu12
 Found existing installation: nvidia-cublas-cu12 12.5.3.2
 Uninstalling nvidia-cublas-cu12-12.5.3.2:
 Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
 Attempting uninstall: nvidia-cusparse-cu12
 Found existing installation: nvidia-cusparse-cu12 12.5.1.3
 Uninstalling nvidia-cusparse-cu12-12.5.1.3:
 Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
 Attempting uninstall: nvidia-cudnn-cu12
 Found existing installation: nvidia-cudnn-cu12 9.3.0.75
 Uninstalling nvidia-cudnn-cu12-9.3.0.75:
 Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
 Attempting uninstall: nvidia-cusolver-cu12
 Found existing installation: nvidia-cusolver-cu12 11.6.3.83
 Uninstalling nvidia-cusolver-cu12-11.6.3.83:
 Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
 Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-run
```

```
import numpy as np
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Define the structure
model = DiscreteBayesianNetwork([('B', 'A'), ('E', 'A'), ('A', 'D'), ('A', 'S')])

# Define CPDs
cpd_b = TabularCPD(variable='B', variable_card=2, values=[[0.002], [0.998]])
cpd_e = TabularCPD(variable='E', variable_card=2, values=[[0.001], [0.999]])
cpd_a = TabularCPD(variable='A', variable_card=2,
                   values=[[0.94, 0.95, 0.31, 0.001],
                           [0.06, 0.04, 0.69, 0.999]],
                   evidence=['B', 'E'], evidence_card=[2, 2])
cpd_d = TabularCPD(variable='D', variable_card=2,
                   values=[[0.91, 0.05],
                           [0.09, 0.95]],
                   evidence=['A'], evidence_card=[2])
cpd_s = TabularCPD(variable='S', variable_card=2,
                   values=[[0.75, 0.02],
                           [0.25, 0.98]],
                   evidence=['A'], evidence_card=[2])

# Add CPDs to model
model.add_cpds(cpd_b, cpd_e, cpd_a, cpd_d, cpd_s)
assert model.check_model()

# Inference
inference = VariableElimination(model)
query_result = inference.query(variables=['D', 'S', 'A'], evidence={'B': 0, 'E': 0}, joint=True)
print(query_result)

# Extracting probabilities
probability = query_result.values[1, 1, 1]
probability1 = query_result.values[1, 0, 1]
probability2 = query_result.values[0, 1, 0]

print(f"\nProbability S=True, D=True, A=True (given B=False, E=False): {probability:.8f}")
print("Probability1:", probability1)
print("Probability2:", probability2)
```



```
+------+------+------+--------------+
| D | S | A | phi(D,S,A) |
+======+======+======+==============+
| D(0) | S(0) | A(0) | 0.6416 |
+------+------+------+--------------+
| D(0) | S(0) | A(1) | 0.0001 |
```

```
+------+------+------+--------------+
| D(0) | S(1) | A(0) | 0.2139 |
+------+------+------+--------------+
| D(0) | S(1) | A(1) | 0.0029 |
+------+------+------+--------------+
| D(1) | S(0) | A(0) | 0.0635 |
+------+------+------+--------------+
| D(1) | S(0) | A(1) | 0.0011 |
+------+------+------+--------------+
| D(1) | S(1) | A(0) | 0.0212 |
+------+------+------+--------------+
| D(1) | S(1) | A(1) | 0.0559 |
+------+------+------+--------------+

Probability S=True, D=True, A=True (given B=False, E=False): 0.05586000
Probability1: 0.0011400000000000002
Probability2: 0.21385000000000004
```

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

# Define structure
model = DiscreteBayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])

# CPDs
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])
cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])
cpd_host = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],
                                  [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],
                                  [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
                      evidence=['Guest', 'Price'], evidence_card=[3, 3])

# Add CPDs
model.add_cpds(cpd_guest, cpd_price, cpd_host)
assert model.check_model()

# Inference
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price': 2})

print(posterior_p)

# Extract a probability
print("Probability for Host = 0:", posterior_p.values[0])
```

```
+---------+-------------+
| Host | phi(Host) |
+=========+=============+
| Host(0) | 0.5000 |
+---------+-------------+
| Host(1) | 0.5000 |
+---------+-------------+
| Host(2) | 0.0000 |
+---------+-------------+
```



```
Probability for Host = 0: 0.5
```