Лабораторная работа 4

Байесовские сети на примере датасета Mushroom Classification

Выполнял: Книга Тимофей
Группа: М80-309Б-23

Что такое байесовская сеть

Bayesian Network (байесовская сеть) — это вероятностная модель в виде **графа**:

- **узлы** = случайные переменные (признаки/события),
- **стрелки** = *условные зависимости* (что на что влияет),
- у каждого узла есть **таблица условных вероятностей (CPT)**: $P(X|Parents(X))$

Главная идея: байесовская сеть описывает **совместное распределение** всех переменных и позволяет делать выводы:

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(\, X_i \mid Parents(X_i)\,)$$

То есть мы разбиваем «большую» сложную вероятность на набор более простых.

Используется для**:**

- **Моделирования неопределённость**: когда данные шумные, неполные или неоднозначные.
- **Понимания зависимости**: сеть показывает, какие факторы связаны и как.
- **Inference (вывод)**: отвечать на вопросы вида «какова вероятность A, если известно B?»

В каких задачах применяются

- **Диагностика и медицина**: вероятность болезни по симптомам и анализам, поддержка решений врача.
- **Риск-аналитика и финансы**: кредитный скоринг, выявление мошенничества, оценка вероятности дефолта.
- **Надёжность и инженерия**: поиск причин отказа системы, анализ аварий (fault diagnosis).
- **Прогнозирование и поддержка решений**: “что будет, если…” при заданных условиях (сценарный анализ).
- **Рекомендательные системы**: вероятность интереса к товару/контенту при известных предпочтениях.
- **NLP и информационные системы**: извлечение сущностей/классификация при неполной информации (в классических подходах).

Загрузка датасета

```
import kagglehub
import pandas as pd
from pathlib import Path

path = Path(kagglehub.dataset_download("uciml/mushroom-classification"))
print("Path to dataset files:", path)

file_dir = path / 'mushrooms.csv'
raw = pd.read_csv(file_dir, header=0)
data = raw.copy()
```

[451] Python

```
Path to dataset files: /Users/nugget/.cache/kagglehub/datasets/uciml/mushroom-classification/versions/1
```

Обработка данных

можно закодировать метки с помощью Label Encoding

```
[454]
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for col in data.columns:
    data[col] = le.fit_transform(data[col])

data.head(3)
```

Python

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3	5
1	0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3	2	1
2	0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3	2	3

3 rows × 23 columns

Проверим, есть ли у нас дубликаты:

```
[455]
data.duplicated().sum()
```

Python

```
np.int64(0)
```

Для pgmpy признаки должны быть дискретными. Так как они категориальные, мы делаем Label Encoding.

Также полезно проверить на дубликаты

Построение байесовской сети

В работе использовали два варианта структуры: **ручную** и **автоматическую** (HillClimbSearch + BIC).

Ручная структура проще и понятнее: берём четыре сильных признака (запах, наличие потемнений, размер и цвет пластинок) и ведём стрелки на class:

```python
from pgmpy.models import DiscreteBayesianNetwork

# Задаем направление между вершинами
network = [
    ('odor', 'class'),
    ('gill-size', 'class'),
    ('gill-color', 'class'),
    ('bruises', 'class'),
]

# Строим Дискретную Байесовскую сеть
model = DiscreteBayesianNetwork(network)
model.edges()  # Просмотр ребер
```

Python

```
OutEdgeView([('odor', 'class'), ('gill-size', 'class'), ('gill-color', 'class'), ('bruises', 'class')])
```

Автоматическое построение байесовской сети

Этот код использует метод поиска с подъемом на холм (Hill Climb Search) для оценки структуры Байесовской сети из данных, используя Байесовский информационный критерий (BIC) в качестве функции оценки.

BIC (Bayesian Information Criterion) — это функция оценки, используемая для сравнения статистических моделей (в данном случае, структур Байесовских сетей).

Цель: Он наказывает модели за излишнюю сложность (большое количество параметров/ребер), одновременно вознаграждая за хорошее соответствие данным.

Принцип: Модели с меньшим значением BIC считаются лучшими.

Формула (концептуально): BIC≈L−k⋅ln(N), где L — логарифмическое правдоподобие модели (насколько хорошо модель объясняет данные), k — количество параметров модели (сложность), а N — количество точек данных.

❗ ЕСЛИ будете строить сеть автоматически, посмотрите на методы и разберитесь с ними☺️❗

```python
from pgmpy.estimators import HillClimbSearch, BIC

hc = HillClimbSearch(data)
best_model = hc.estimate(scoring_method=BIC(data))
hc_model = DiscreteBayesianNetwork(best_model.edges())
hc_model.edges()  # Автоматическая структура
```

```
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred from data:
 {'class': 'N', 'cap-shape': 'N', 'cap-surface': 'N', 'cap-color': 'N', 'bruises': 'N', 'odor': 'N', 'gill-attachment': 'N', 'gill-spacing': 'N', 'gill-size':
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred from data:
 {'class': 'N', 'cap-shape': 'N', 'cap-surface': 'N', 'cap-color': 'N', 'bruises': 'N', 'odor': 'N', 'gill-attachment': 'N', 'gill-spacing': 'N', 'gill-size':
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred from data:
 {'class': 'N', 'cap-shape': 'N', 'cap-surface': 'N', 'cap-color': 'N', 'bruises': 'N', 'odor': 'N', 'gill-attachment': 'N', 'gill-spacing': 'N', 'gill-size':
  0%|          | 53/1000000 [00:00<4:46:55, 58.08it/s]
```

```
OutEdgeView([('class', 'habitat'), ('class', 'stalk-surface-above-ring'), ('class', 'population'), ('class', 'bruises'), ('class', 'stalk-surface-below-ring'),
```

Оценка параметров

Параметры в Bayesian Network — это **таблицы условных вероятностей (CPT)** для каждого узла: $P\big(X \mid Parents(X)\big)$. Их можно оценивать двумя популярными способами

Maximum Likelihood Estimator (MLE)

Идея простая: берём **частоты в данных** и превращаем их в вероятности.

Плюсы:
- максимально “честно” и быстро отражает данные;

Минусы:
- если каких-то комбинаций в данных **не было**, получаются **нули** в CPT

Bayesian Estimator (BayesianEstimator)

Тоже опирается на частоты, но добавляет **априорное сглаживание** чтобы избежать нулевых вероятностей.

Плюсы:
- меньше нулей, модель стабильнее на редких случаях;
- лучше работает на небольших данных/редких комбинациях.

Минусы:
- результат чуть зависит от настроек prior и equivalent_sample_size.

Maximum Likelihood Estimator (MLE)

```
# # Метод Максимального Правдоподобия
from pgmpy.estimators import MaximumLikelihoodEstimator

model.fit(data, estimator=MaximumLikelihoodEstimator)
```

Bayesian Estimator

```
# Байесовский оценщик
from pgmpy.estimators import BayesianEstimator

model.fit(data, estimator=BayesianEstimator, prior_type='BDeu', equivalent_sample_size=10)
hc_model.fit(data, estimator=BayesianEstimator, prior_type='BDeu', equivalent_sample_size=10)
```

Просмотр CPT

Просмотр полной таблички

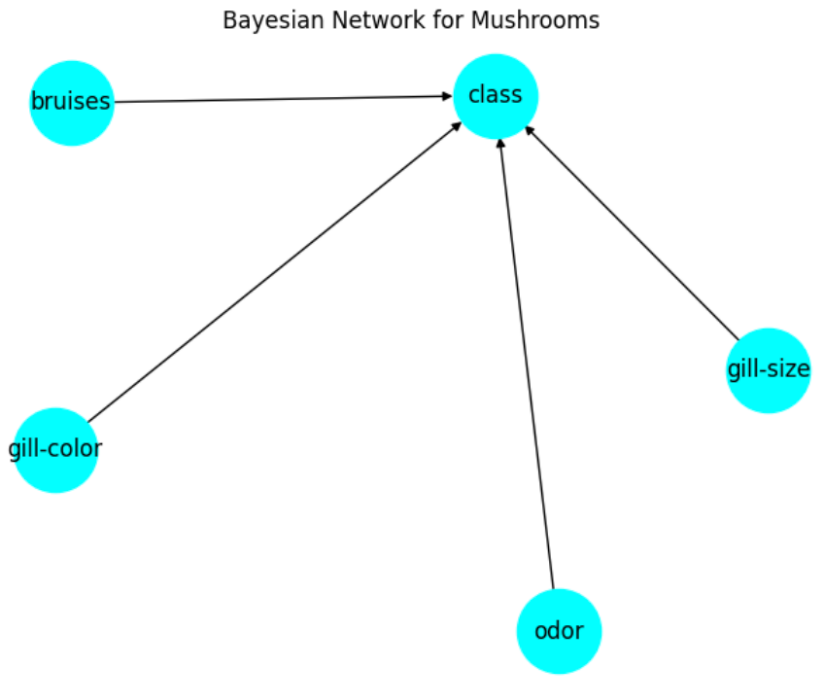
```
[460]
print(model.get_cpds('class'))
print(hc_model.get_cpds('class'))
```

```
...
+------------+---------------+-----+----------------+----------------+
| bruises    | bruises(0)    | ... | bruises(1)     | bruises(1)     |
+------------+---------------+-----+----------------+----------------+
| gill-color | gill-color(0) | ... | gill-color(11) | gill-color(11) |
+------------+---------------+-----+----------------+----------------+
| gill-size  | gill-size(0)  | ... | gill-size(1)   | gill-size(1)   |
+------------+---------------+-----+----------------+----------------+
| odor       | odor(0)       | ... | odor(7)        | odor(8)        |
+------------+---------------+-----+----------------+----------------+
| class(0)   | 0.5           | ... | 0.5            | 0.5            |
+------------+---------------+-----+----------------+----------------+
| class(1)   | 0.5           | ... | 0.5            | 0.5            |
+------------+---------------+-----+----------------+----------------+
+-------------+-----+----------------------+
| odor        | ... | odor(8)              |
+-------------+-----+----------------------+
| stalk-shape | ... | stalk-shape(1)       |
+-------------+-----+----------------------+
| class(0)    | ... | 0.0004817883985353632 |
+-------------+-----+----------------------+
| class(1)    | ... | 0.9995182116014646    |
+-------------+-----+----------------------+
```

Визуализация сети: ручная

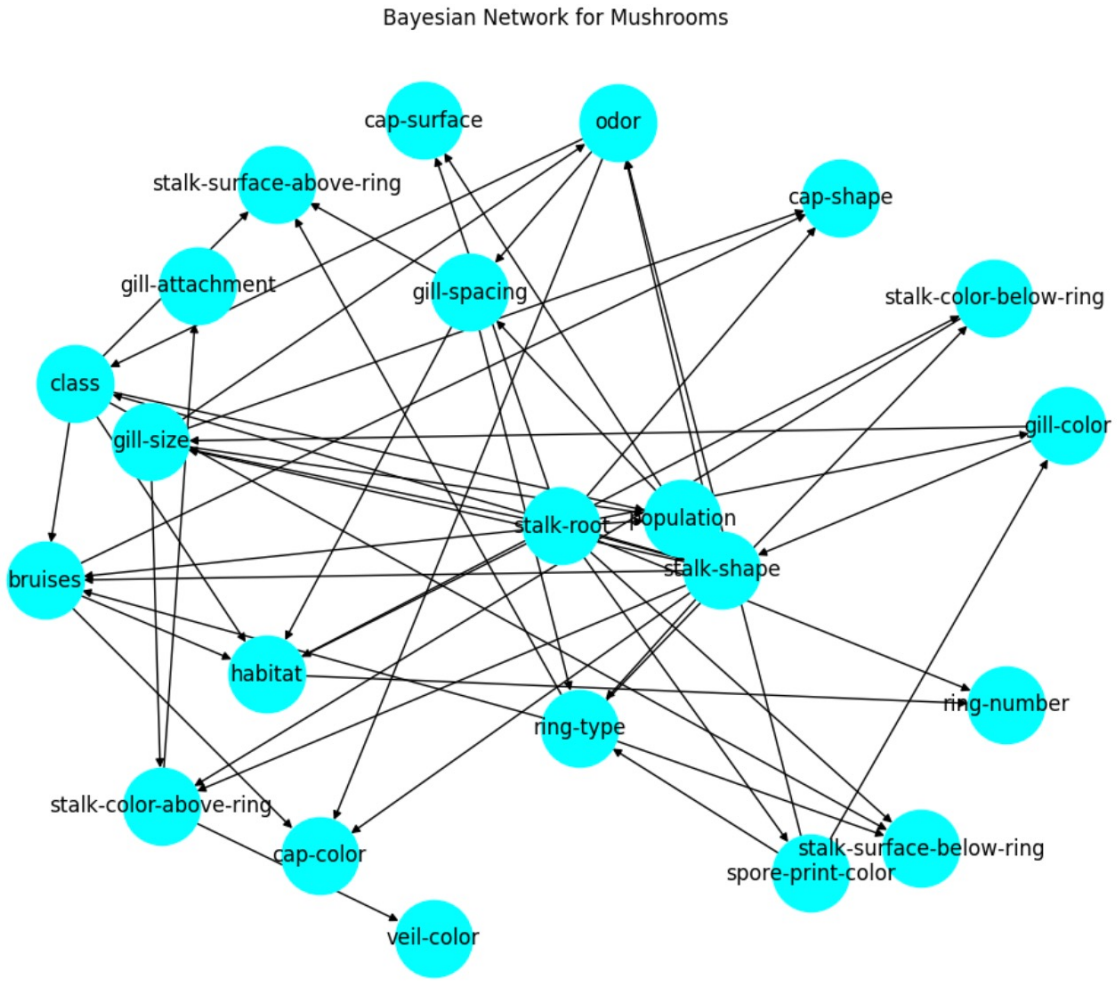
Ручная структура: 4 признака → `class`.

Стрелка читается как «признак помогает объяснить вероятность класса».



Визуализация сети: автоматическая

Хоть тут и очень много связей, стоит обратить внимание на то, что в class ведут всего ли два признака



Inference

Inference — это вычисление вероятностей интересующих переменных при заданных фактах (evidence).

В pgmpy использовали `VariableElimination`: задаём evidence и получаем распределение P(class | evidence).

```
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)
hc_infer = VariableElimination(hc_model)

print(infer.query(variables=['class']))
print(hc_infer.query(variables=['class']))

print(infer.query(variables=['class'], evidence={'odor': 2}))
print(hc_infer.query(variables=['class'], evidence={'odor': 2}))
```

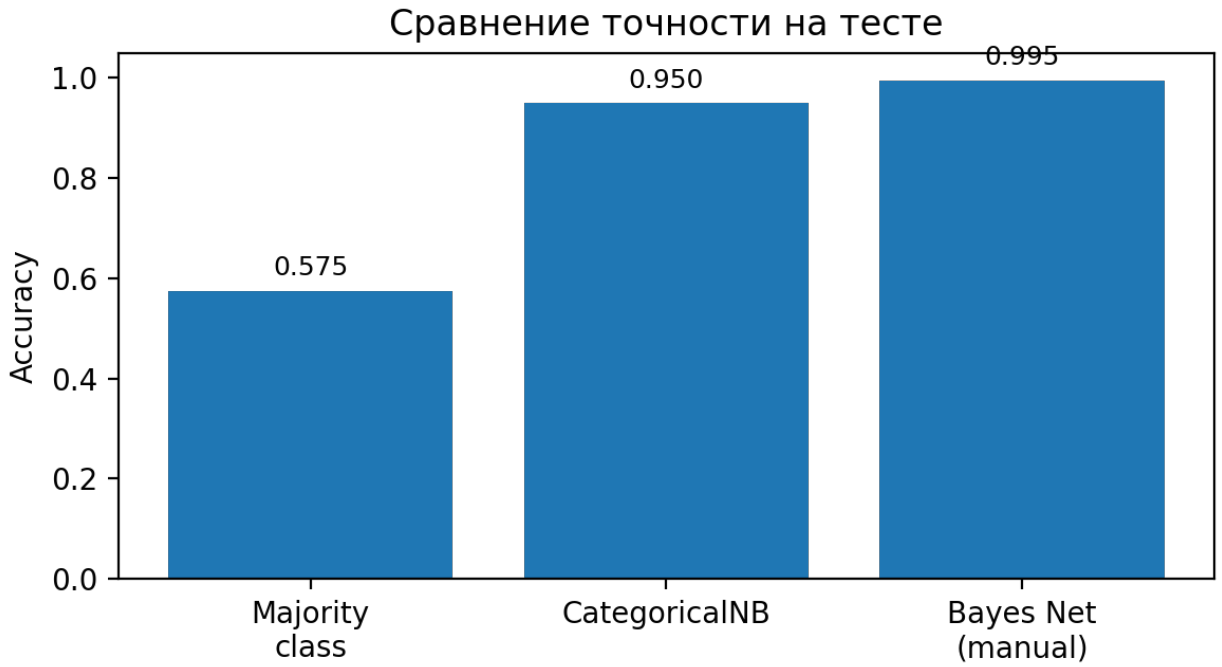
Inference: результаты

Сценарий	P(e) съед.	P(p) ядов.
Без условий (manual)	0.5752	0.4248
Без условий (auto)	0.5178	0.4822
При odor=2 (`f`) (manual)	0.3464	0.6536
При odor=2 (`f`) (auto)	0.0003	0.9997

Интерпретация: если `odor='f'`, модель резко повышает вероятность «ядовитый».

В «auto» структуре (HillClimbSearch) при этом признаке уверенность почти 100%.

Сравнение с baseline и выводы



Модель	Accuracy	Log-likelihood*
Majority class	0.5752	—
CategoricalNB (baseline)	0.9495	-0.1384
Bayes Net (manual)	0.9947	-0.0144

Bayesian Network показала **лучшую точность** на тесте (≈0.995) по сравнению с baseline CategoricalNB (≈0.950).

Сеть даёт не только предсказание, но и понятные вероятности: можно задавать evidence и смотреть, как меняется P(class).

*Log-likelihood здесь показан как «-log_loss» из ноутбука (чем ближе к 0, тем лучше).