Лабораторная работа №4
Байесовские сети

Dataset: Mushroom Classification

Введение

Что такое Байесовские сети?

- Графические вероятностные модели, представляющие зависимости между случайными переменными
- Состоят из узлов (переменные) и ребер (вероятностные зависимости)
- Позволяют выполнять логические выводы при наличии неполной информации

Формула Байеса:

$$P(A \mid B) = P(B \mid A) \cdot P(A) P(B)$$

Введение

Этот набор данных содержит описания гипотетических образцов, соответствующих 23 видам пластинчатых грибов семейств Agaricus и Lepiota, взятым из Полевого справочника по североамериканским грибам Общества Одюбона (1981). Каждый вид идентифицирован как определенно съедобный, определенно ядовитый или с неизвестной съедобностью и не рекомендуется к использованию. Этот последний класс был объединен с ядовитым. В справочнике четко указано, что не существует простого правила определения съедобности гриба; нет правила типа «три листочка, пусть будет» для ядовитого дуба и ядовитого плюща.

Загрузка и обработка данных

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

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

# Загрузка Mushroom Classification
file_dir = path / 'mushrooms.csv'
data = pd.read_csv(file_dir)

print()   # Размер датасета
```

6.3s Python

```
c:\Users\tvm15\AppData\Local\Programs\Python\Python313\Lib\site-packages\tqdm\auto.py:21:
  from .autonotebook import tqdm as notebook_tqdm
Path to dataset files: C:\Users\tvm15\.cache\kagglehub\datasets\uciml\mushroom-classificat
```

```python
if 'stalk-root' in data.columns:
    if data['stalk-root'].dtype == 'object':
        data['stalk-root'] = data['stalk-root'].replace('?', 'missing')
        print(f"Обработаны пропущенные значения в 'stalk-root'. Новое распределение:")
        print(data['stalk-root'].value_counts().head())
```

0.0s Python

```
Обработаны пропущенные значения в 'stalk-root'. Новое распределение:
stalk-root
b          3776
missing    2480
e          1120
c           556
r           192
Name: count, dtype: int64
```

Обработка данных

Label Encoding

```
from sklearn.preprocessing import LabelEncoder


df = data.copy()
le = LabelEncoder()
for column in df.columns:
    if df[column].dtype == 'object':
        try:
            df[column] = le.fit_transform(df[column])
        except Exception as e:
            if df[column].isnull().any():
                df[column] = df[column].fillna('missing')
                df[column] = le.fit_transform(df[column])


print(df.head())
print(df.dtypes)
```

✓ 0.0s

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	\
0	1	5	2	4	1	6	1	
1	0	5	2	9	1	0	1	
2	0	0	2	8	1	3	1	
3	1	5	3	8	1	6	1	
4	0	5	2	3	0	5	1	

Создание байесовской сети

```python
import networkx as nx
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator, HillClimbSearch
from pgmpy.models import DiscreteBayesianNetwork


train_data = df.copy()


estimator = HillClimbSearch(train_data)
best_model = estimator.estimate(
    scoring_method="bic-d",
    max_indegree=2,
    max_iter=100,
    show_progress=True
)

print(f"Найдено ребер: {len(best_model.edges())}")
print(f"Ребра до фильтрации: {list(best_model.edges())[:5]}...")


edges = [(u, v) for (u, v) in best_model.edges() if not (u == 'class' and v != 'class')]
edges = [(u, v) for (u, v) in edges if u in train_data.columns and v in train_data.columns]


mandatory_connections = [('odor', 'class')]
for edge in mandatory_connections:
    if edge not in edges and edge[0] in train_data.columns and edge[1] in train_data.columns:
        edges.append(edge)
        print(f"Добавлена обязательная связь: {edge[0]} → {edge[1]}")
```

```python
G = nx.DiGraph(edges)
while not nx.is_directed_acyclic_graph(G):
    cycles = list(nx.simple_cycles(G))
    if not cycles:
        break
    cycle = cycles[0]
    edge_to_remove = (cycle[-1], cycle[0])
    if edge_to_remove in edges:
        edges.remove(edge_to_remove)
        print(f"Удалено ребро для устранения цикла: {edge_to_remove}")
    G = nx.DiGraph(edges)

if len(edges) > 10:
    print(f"Слишком сложная структура ({len(edges)} ребер). Упрощаем до 10 ключевых связей...")
    class_edges = [(u, v) for (u, v) in edges if v == 'class']
    other_edges = [(u, v) for (u, v) in edges if v != 'class']
    edges = class_edges[:3] + other_edges[:5]

print(f"ФИНАЛЬНАЯ СТРУКТУРА ({len(edges)} ребер):")
for i, (u, v) in enumerate(edges, 1):
    print(f"   {i}. {u} → {v}")

print("Создаем и обучаем байесовскую сеть...")
model = DiscreteBayesianNetwork(edges)

try:
    from pgmpy.estimators import BayesianEstimator
    model.fit(
        train_data,
        estimator=BayesianEstimator,
        prior_type='BDeu',
        equivalent_sample_size=10
    )
    print("Успешно обучено с BDeu prior (equivalent_sample_size=10)")
except Exception as e:
    print(f"Ошибка BDeu: {e}. Используем MLE...")
    from pgmpy.estimators import MaximumLikelihoodEstimator
    model.fit(train_data, estimator=MaximumLikelihoodEstimator)
    print("Успешно обучено с Maximum Likelihood Estimation")
```

✓ 2.3s

Оценка параметров

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

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

✓ 0.2s

```
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred from data:
 {'class': 'C', 'cap-shape': 'C', 'cap-surface': 'C', 'cap-color': 'C', 'bruises': 'C', 'odor': 'C', 'gill-attachment': 'C', 'gill-spacing': 'C', 'gill-size': 'C',
WARNING:pgmpy:Replacing existing CPD for odor
WARNING:pgmpy:Replacing existing CPD for class
WARNING:pgmpy:Replacing existing CPD for stalk-shape
WARNING:pgmpy:Replacing existing CPD for cap-color
WARNING:pgmpy:Replacing existing CPD for gill-spacing
WARNING:pgmpy:Replacing existing CPD for gill-size
WARNING:pgmpy:Replacing existing CPD for ring-type
WARNING:pgmpy:Replacing existing CPD for stalk-surface-above-ring
```

```
<pgmpy.models.DiscreteBayesianNetwork.DiscreteBayesianNetwork at 0x1bf581b6d50>
```

Вместо того чтобы просто считать частоты (как MLE), Байесовский оценщик вычисляет апостериорное распределение вероятностей (CPT)

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

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

✓ 0.0s

```
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred from data:
 {'class': 'C', 'cap-shape': 'C', 'cap-surface': 'C', 'cap-color': 'C', 'bruises': 'C', 'odor': 'C', 'gill-attachment': 'C', 'gill-spacing': 'C', 'gill-size': 'C',
WARNING:pgmpy:Replacing existing CPD for odor
WARNING:pgmpy:Replacing existing CPD for class
WARNING:pgmpy:Replacing existing CPD for cap-color
WARNING:pgmpy:Replacing existing CPD for gill-spacing
WARNING:pgmpy:Replacing existing CPD for gill-size
WARNING:pgmpy:Replacing existing CPD for stalk-shape
WARNING:pgmpy:Replacing existing CPD for ring-type
WARNING:pgmpy:Replacing existing CPD for stalk-surface-above-ring
```

CPT

```
cpt_class = model.get_cpds('class')
print(cpt_class)
```

1] ✓ 0.0s

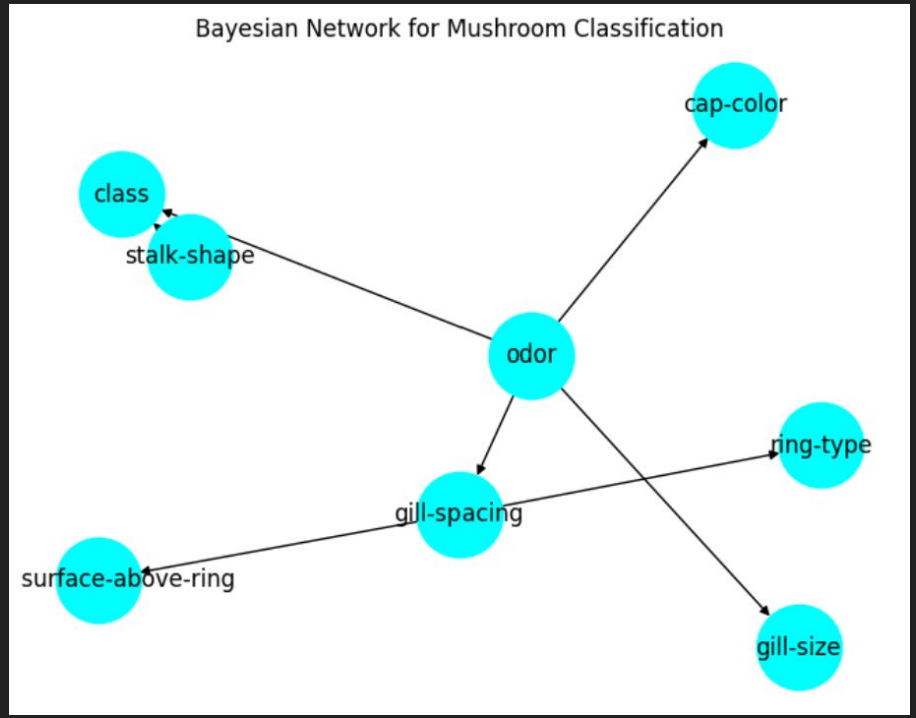
```
+-------------+-----+-----------------------+
| odor        | ... | odor(y)               |
+-------------+-----+-----------------------+
| stalk-shape | ... | stalk-shape(t)        |
+-------------+-----+-----------------------+
| class(e)    | ... | 0.00048178839853536 32 |
+-------------+-----+-----------------------+
| class(p)    | ... | 0.9995182116014646    |
+-------------+-----+-----------------------+
```

Визуализация сети

```
import networkx as nx
import matplotlib.pyplot as plt

nx_graph = nx.DiGraph(model.edges())
pos = nx.spring_layout(nx_graph)
nx.draw(nx_graph, pos, with_labels=True, node_size=2000, node_color='cyan', arrows=True)
plt.title("Bayesian Network for Mushroom Classification")
plt.show()
```

✓ 0.3s



Random Search

```python
# Создаем inference engine перед использованием
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)
print("Inference engine создан успешно")
print(f"Доступные переменные для запросов: {model.nodes()}")
print()

# Пример 1: Маргинальное распределение класса
print("\n1. Маргинальное распределение класса (без evidence):")
query_class = infer.query(variables=['class'])
print(query_class)
print(f"Базовые вероятности: съедобный={query_class.values[0]:.2%}, ядовитый={query_class.values[1]:.2%}")

# Пример 2: Запрос с одним свидетельством
print("\n2. Запрос с evidence odor=2 (неприятный запах):")
query_odor = infer.query(variables=['class'], evidence={'odor': 'y'})
print(query_odor)
print(f"При неприятном запахе: съедобный={query_odor.values[0]:.2%}, ядовитый={query_odor.values[1]:.2%}")
```

✓ 0.0s

```
1. Маргинальное распределение класса (без evidence):
+----------+--------------+
| class    |   phi(class) |
+==========+==============+
| class(e) |       0.5581 |
+----------+--------------+
| class(p) |       0.4419 |
+----------+--------------+
Базовые вероятности: съедобный=55.81%, ядовитый=44.19%

2. Запрос с evidence odor=2 (неприятный запах):
+----------+--------------+
| class    |   phi(class) |
+==========+==============+
| class(e) |       0.2167 |
+----------+--------------+
| class(p) |       0.7833 |
+----------+--------------+
При неприятном запахе: съедобный=21.67%, ядовитый=78.33%
```

Сравнение с baseline-моделью

```python
print("СРАВНЕНИЕ МОДЕЛЕЙ")

comparison_df = pd.DataFrame({
    'Модель': ['Naive Bayes', 'Bayesian Network'],
    'Accuracy': [nb_accuracy, bn_accuracy],
    'Точность (Precision)': [
        classification_report(y_test, nb_predictions, output_dict=True)['weighted avg']['precision'],
        classification_report(y_test_numeric, bayesian_predictions, output_dict=True)['weighted avg']['precision']
    ],
    'Полнота (Recall)': [
        classification_report(y_test, nb_predictions, output_dict=True)['weighted avg']['recall'],
        classification_report(y_test_numeric, bayesian_predictions, output_dict=True)['weighted avg']['recall']
    ],
    'F1-score': [
        classification_report(y_test, nb_predictions, output_dict=True)['weighted avg']['f1-score'],
        classification_report(y_test_numeric, bayesian_predictions, output_dict=True)['weighted avg']['f1-score']
    ]
})

print("Таблица сравнения моделей:")
print(comparison_df.to_string(index=False))

print("5.ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
Размер обучающей выборки: (6499, 22)
Размер тестовой выборки: (1625, 22)
Распределение классов в тестовой выборке:
Съедобные (e): 842
Ядовитые (p): 783
ccuracy наивного байеса: 0.9458 (94.58%)
                    precision    recall  f1-score   support

  edible (съедобный)       0.91      0.99      0.95       842
poisonous (ядовитый)       0.99      0.90      0.94       783

            accuracy                           0.95      1625
           macro avg       0.95      0.94      0.95      1625
        weighted avg       0.95      0.95      0.95      1625

Успешно выполнено предсказаний: 0/1625
Пример предсказаний (первые 5 строк):
Истинный класс | Предсказание NB | Предсказание BN
--------------------------------------------------
  Ядовитый     |    Ядовитый     |    Съедобный
  Ядовитый     |    Съедобный    |    Ядовитый
  Съедобный    |    Съедобный    |    Ядовитый
  Ядовитый     |    Ядовитый     |    Ядовитый
  Ядовитый     |    Ядовитый     |    Ядовитый
...
          Модель  Accuracy  Точность (Precision)  Полнота (Recall)  F1-score
     Naive Bayes  0.945846              0.949423          0.945846  0.945643
Bayesian Network  0.493538              0.494406          0.493538  0.493688
```

Сравнение с baseline-моделью