Байесовские сети

Датасет: Zoo

Что такое Bayesian Network

Байесовская сеть — это направленный ациклический граф (DAG), где узлы = переменные, а рёбра = вероятностные зависимости.

Модель хранит **условные вероятности** (CPT/CPD) и позволяет делать **inference**: считать P(интересующая переменная | наблюдения)

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

$$P(X_1, \dots, X_n) = \Pi_{i=1}^{n}\, p(X_i \,|\text{РОДИТЕЛЬСКАЯ}(X_i))$$

Загрузка и обработка датасета

Size: (101, 18)

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	class_type	
0	1	0	0	1	0	0	1	1	1	1	0	0	2	0	0	1	0	
1	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0	1	0	
2	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3	

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "zoo.csv"

# Load the latest version
data = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "uciml/zoo-animal-classification",
  file_path
)

print("First 5 records:", data.head())
```

```
RangeIndex: 101 entries, 0 to 100
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   hair        101 non-null    int64
 1   feathers    101 non-null    int64
 2   eggs        101 non-null    int64
 3   milk        101 non-null    int64
 4   airborne    101 non-null    int64
 5   aquatic     101 non-null    int64
 6   predator    101 non-null    int64
 7   toothed     101 non-null    int64
 8   backbone    101 non-null    int64
 9   breathes    101 non-null    int64
 10  venomous    101 non-null    int64
 11  fins        101 non-null    int64
 12  legs        101 non-null    int64
 13  tail        101 non-null    int64
 14  domestic    101 non-null    int64
 15  catsize     101 non-null    int64
 16  class_type  101 non-null    int64
dtypes: int64(17)
```

Загрузка и обработка датасета

Size: (101, 18)

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	class_type
0	1	0	0	1	0	0	1	1	1	1	0	0	2	0	0	1	0
1	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0	1	0
2	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
3	1	0	0	1	0	0	1	1	1	1	0	0	2	0	0	1	0
4	1	0	0	1	0	0	1	1	1	1	0	0	2	1	0	1	0

```
data = data.copy()

# id животного не используем в модели
if "animal_name" in data.columns:
    data = data.drop(columns=["animal_name"])

target = "class_type"
features = [c for c in data.columns if c != target]
data[features + [target]] = data[features + [target]].astype(int)

data.head()
```

```
from sklearn.preprocessing import LabelEncoder

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

data.head()
```

Построение Bayesian Network

```
from pgmpy.estimators import HillClimbSearch, BIC

hc = HillClimbSearch(data)
best_model = hc.estimate(
    scoring_method=BIC(data),
    max_indegree=2, # ограничение числа родителей
    tabu_length=20, # чтобы шире искать в пространстве графов
    epsilon=1e-4
)
model = DiscreteBayesianNetwork(best_model.edges())
model.edges()  # Автоматическая структура
```

```
OutEdgeView([('milk', 'eggs'), ('milk', 'hair'), ('milk', 'catsize'), ('milk', 'venomous'),
('aquatic', 'predator'), ('aquatic', 'hair'), ('predator', 'domestic'), ('toothed',
'class_type'), ('toothed', 'eggs'), ('class_type', 'milk'), ('class_type', 'feathers'),
('class_type', 'backbone'), ('class_type', 'breathes'), ('class_type', 'legs'), ('class_type',
'fins'), ('backbone', 'tail'), ('breathes', 'aquatic'), ('fins', 'aquatic'), ('legs',
'airborne')])
```

Оценка параметров и CPT

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

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

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

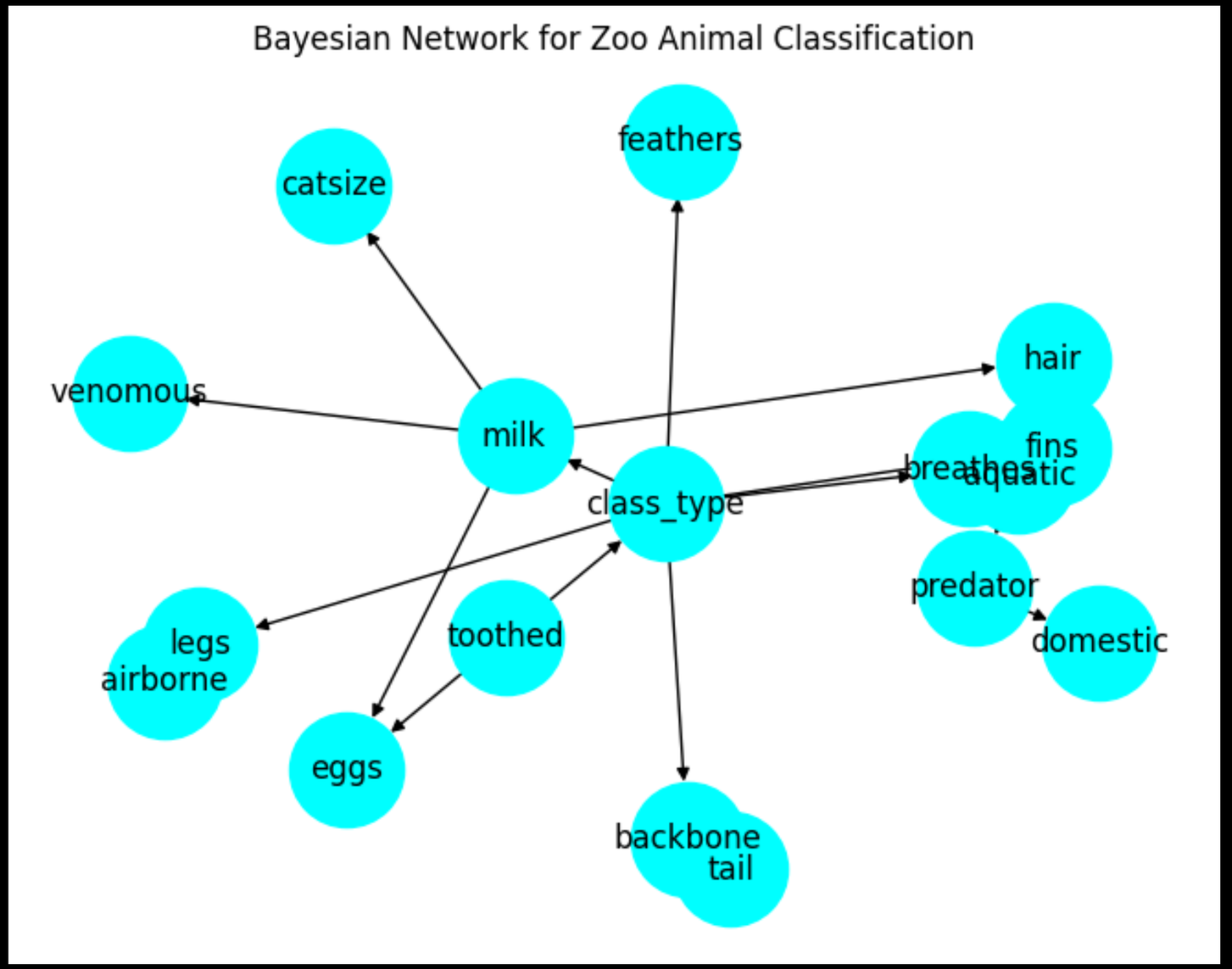
```
for node in ['hair', 'milk']:
    cpt = model.get_cpds(node)
    print(f"CPT for {node}:\n{cpt}")
```

```
CPT for hair:
+---------+---------------------+-----+---------------------+
| aquatic | aquatic(0)          | ... | aquatic(1)          |
+---------+---------------------+-----+---------------------+
| milk    | milk(0)             | ... | milk(1)             |
+---------+---------------------+-----+---------------------+
| hair(0) | 0.8384615384615385  | ... | 0.38235294117647056 |
+---------+---------------------+-----+---------------------+
| hair(1) | 0.16153846153846155 | ... | 0.6176470588235294  |
+---------+---------------------+-----+---------------------+
CPT for milk:
+------------+---------------------+-----+---------------------+
| class_type | class_type(0)       | ... | class_type(6)       |
+------------+---------------------+-----+---------------------+
| milk(0)    | 0.016835016835016835 | ... | 0.9375000000000001  |
+------------+---------------------+-----+---------------------+
| milk(1)    | 0.9831649831649831  | ... | 0.0625000000000001 |
+------------+---------------------+-----+---------------------+
```

```
cpt_class = model.get_cpds(target)
print(cpt_class)
```

```
+---------------+----------------------+----------------------+
| toothed       | toothed(0)           | toothed(1)           |
+---------------+----------------------+----------------------+
| class_type(0) | 0.03809523809523809  | 0.616883116883117    |
+---------------+----------------------+----------------------+
| class_type(1) | 0.4603174603174603   | 0.010822510822510826 |
+---------------+----------------------+----------------------+
| class_type(2) | 0.03809523809523809  | 0.07142857142857145  |
+---------------+----------------------+----------------------+
| class_type(3) | 0.015873015873015872 | 0.20779220779220783  |
+---------------+----------------------+----------------------+
| class_type(4) | 0.015873015873015872 | 0.07142857142857145  |
+---------------+----------------------+----------------------+
| class_type(5) | 0.1936507936507936   | 0.010822510822510826 |
+---------------+----------------------+----------------------+
| class_type(6) | 0.23809523809523805  | 0.010822510822510826 |
+---------------+----------------------+----------------------+
```

Визуализация сети



Inference и результаты

```python
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)
print(infer.query([target], evidence={"milk": 1}))
```

```
+--------------+--------------------+
| class_type   |   phi(class_type) |
+==============+====================+
| class_type(0) |             0.9068 |
+--------------+--------------------+
| class_type(1) |             0.0155 |
+--------------+--------------------+
| class_type(2) |             0.0155 |
+--------------+--------------------+
| class_type(3) |             0.0155 |
+--------------+--------------------+
| class_type(4) |             0.0155 |
+--------------+--------------------+
| class_type(5) |             0.0155 |
+--------------+--------------------+
| class_type(6) |             0.0155 |
+--------------+--------------------+
```

```
+--------------+--------------------+
| class_type   |   phi(class_type) |
+==============+====================+
| class_type(0) |             0.3517 |
+--------------+--------------------+
| class_type(1) |             0.1729 |
+--------------+--------------------+
| class_type(2) |             0.0584 |
+--------------+--------------------+
| class_type(3) |             0.1732 |
+--------------+--------------------+
| class_type(4) |             0.0473 |
+--------------+--------------------+
| class_type(5) |             0.0788 |
+--------------+--------------------+
| class_type(6) |             0.1177 |
+--------------+--------------------+
```

Сравнение результатов с baseline-моделью

Naive Bayes Accuracy: 0.9355

Bayesian Network Accuracy: 0.7097

```python
# --- Наивный байесовский классификатор---
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

y_pred_nb = nb_model.predict(X_test)
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb:.4f}")

# --- Байесовская сеть ---
bn_network = [(feat, target) for feat in features]
bn_model = DiscreteBayesianNetwork(bn_network)
DiscreteBayesianNetwork: bn_model
DiscreteBayesianNetwork with 17 items
ain], axis=1)
bn_model.fit(train_data_bn, estimator=BayesianEstimator, prior_type='BDeu', equivalent_sample_

infer_bn = VariableElimination(bn_model)

y_pred_bn = []
for index, row in X_test.iterrows():
    evidence = {col: row[col] for col in features}

    evidence_int = {k: int(v) for k, v in evidence.items()}

    query_result = infer_bn.query(variables=[target], evidence=evidence_int)

    pred = np.argmax(query_result.values)

    y_pred_bn.append(pred)

accuracy_bn = accuracy_score(y_test, y_pred_bn)
```