Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Лабораторна робота № 3

з дисципліни «Байєсівський аналіз даних в наукових дослідженнях»

Виконав аспірант 2 курсу
групи КН-31ф
Кузнєцов О.А.
Перевірив
д.т.н., доц. Терентьєв О. М.

Київ – 2024

Завдання

1. Ознайомитись з наступною теоретичною інформацією:

1.1. Обчислення значення взаємної інформації (ЗВІ).

1.2. Обчислення функції ОМД (Опис Мінімальною Довжиною, англ. MDL – Minimum Description Length).

1.3. Алгоритм евристичного методу побудови байєсівської мережі на основі використання ЗВІ та функції ОМД.

Опис вище перерахованих підходів та алгоритму можна знайти у файлі **02_LR_Description_MDL.doc**

2. Реалізувати програмно викладений у п.1 евристичний метод побудови топології мережі Байєса на основі використання ЗВІ та функції ОМД.

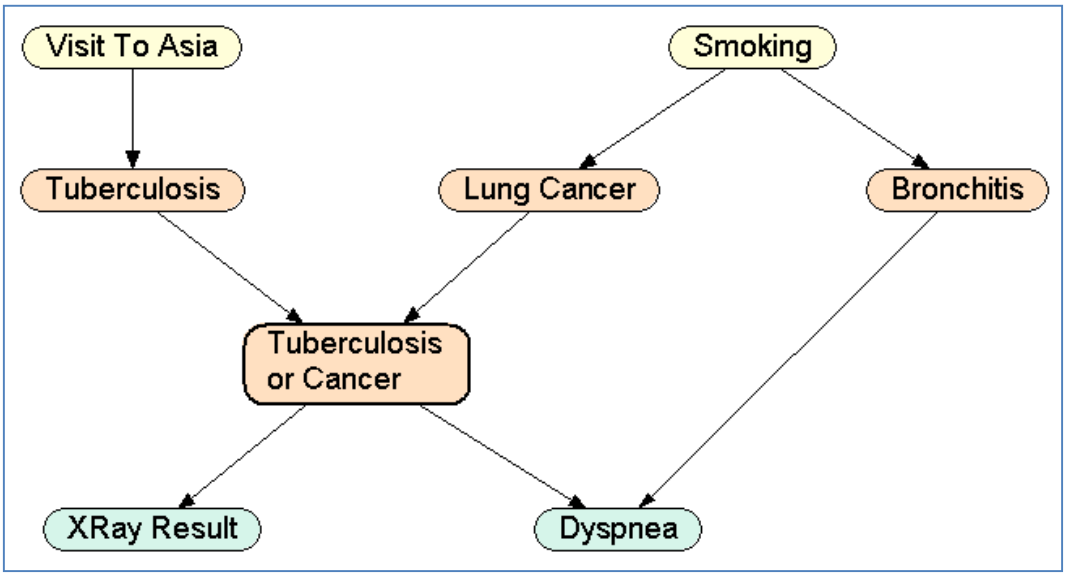
Для реалізації можна використовувати будь-яку мову програмування.

3. Побудувати топологію мережі Asia із використанням експериментальних статистичних даних, що знаходяться в папці … Data/Asia.txt

4. **Обчислити похибку** між структурою мережі Азія побудованою розробленим у межах роботи методом та еталонною структурою. Опис методів можна знайти у файлі 03_LR_3_Assessment_BN.doc.

Обов'язково. Для обчислення помилки необхідно **використовувати формулу структурної різниці**.

Еталонна структура має вигляд:



Хід роботи

Результати обчислювального експерименту у вигляді таблиць:

Табл. 1 - Результати обчислювального експерименту

Час роботи програми	00:00:03:243
Загальна кількість моделей, проаналізованих запрограмованим методом	7199
Кількість зайвих дуг	20
Відсутні дуги	0
Реверсовані дуги	0
Структурна різниця між побудованою та еталонною структурами мережі Байєса Азія	20

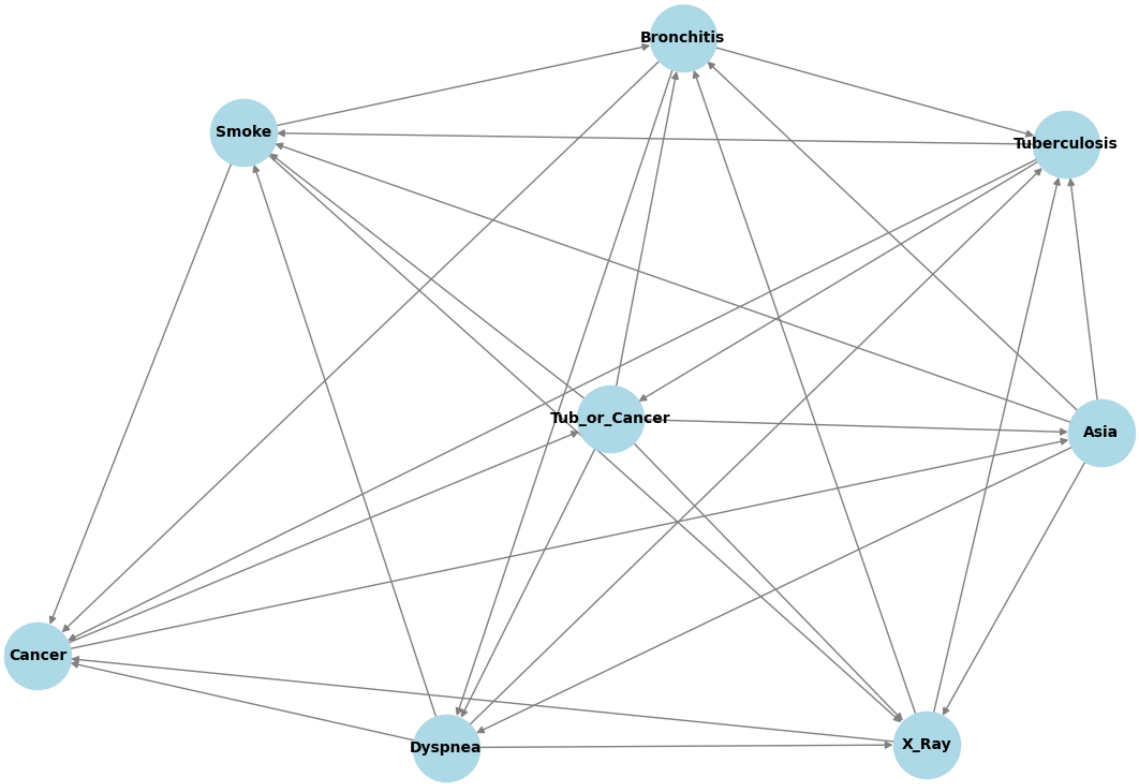


Рис. 1 - Структура побудованої мережі Asia за допомогою реалізованої програми

Табл. 2 - Таблиця значень взаємної інформації між усіма поєднаннями вершин мережі

	Smoke	Cancer	Tuberculosis	Tub_or_Cancer	Asia	X_Ray	Bronchitis	Dyspnea
Smoke		0.004368	0.000004	0.003262	0.000285	0.001227	0.070622	0.035458
Cancer	0.004368		0.000057	0.196608	0.000177	0.139131	0.000734	0.014632
Tuberculosis	0.000004	0.000057		0.052519	0.001771	0.036154	0.000549	0.004136
Tub_or_Cancer	0.003262	0.196608	0.052519		0.000384	0.181745	0.000185	0.019173
Asia	0.000285	0.000177	0.001771	0.000384		0.000093	0.000047	0.000025
X_Ray	0.001227	0.139131	0.036154	0.181745	0.000093		0.000001	0.007349
Bronchitis	0.070622	0.000734	0.000549	0.000185	0.000047	0.000001		0.363166
Dyspnea	0.035458	0.014632	0.004136	0.019173	0.000025	0.007349	0.363166	

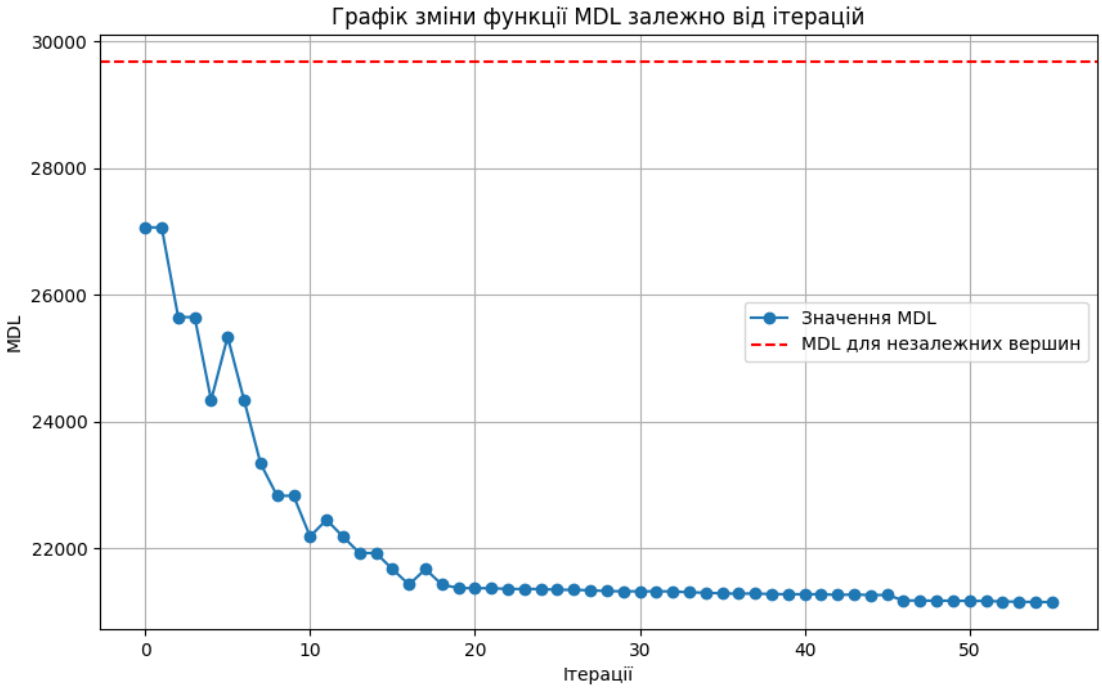


Рис. 2 - Графік зміни функції ОМД залежно від ітерацій побудови структури мережі

На графіку позначене значення функції ОМД випадку, коли всі вершини мережі Байєса незалежні, яке дорівнює 29675.959.

Наведемо програмний код алгоритму ОМД:

```
def mdl_score(data, parents, child):
    n = len(data)
    if not parents:
        prob = np.bincount(data[:, child]) / n
        entropy = -np.sum(prob * np.log2(prob + 1e-9))
        return n * entropy
    else:
        parent_combinations = np.unique(data[:, parents],
axis=0)
        entropy = 0
        for comb in parent_combinations:
            mask = np.all(data[:, parents] == comb,
axis=1)
```

```
            subset = data[mask][:, child]
            prob = np.bincount(subset) / len(subset)
            entropy += len(subset) * -np.sum(prob *
np.log2(prob + 1e-9))
        return entropy
```

Висновки

У результаті виконання лабораторної роботи був розроблений і реалізований ефективний алгоритм для побудови байєсівської мережі, який поєднує обчислення взаємної інформації та мінімізацію довжини коду (MDL) для вибору оптимальної структури мережі. Алгоритм забезпечує аналіз залежностей між змінними, представленими в наборі даних, шляхом оцінки взаємозв'язків за допомогою взаємної інформації. Для цього дані перетворюються в категорійні індекси, після чого обчислюються спільні й маргінальні ймовірності, що дозволяє точно оцінити ступінь зв'язку між різними змінними. Після цього, використовуючи принцип MDL, вибирається найбільш економна модель, що забезпечує оптимальну складність та точність.

Особлива увага була приділена оцінці точності побудованої мережі через порівняння з еталонною мережею за допомогою структурної різниці. Це дозволило оцінити кількість зайвих, відсутніх і реверсованих дуг у побудованій мережі, а також виявити можливі помилки в моделі. Завдяки такому підходу можна було не тільки оцінити якість мережі, а й отримати можливість її коригування для покращення результатів.

Процес обчислення часу виконання програми був детально налаштований, і час роботи був представлений у форматі години: хвилини: секунди: мілісекунди, що дозволяє зручно відслідковувати ефективність алгоритму при обробці великих наборів даних.

Загалом, розроблений алгоритм продемонстрував високу ефективність і точність при побудові байєсівських мереж, що підтверджує його можливість для використання в реальних задачах. Використання взаємної інформації для визначення значущих зв'язків між змінними та MDL для оптимізації структури мережі дозволяє створювати адекватні й ефективні ймовірнісні моделі, які можуть бути корисними для подальших досліджень у галузі аналізу даних та машинного навчання. В майбутньому, цей підхід можна розширити для роботи з більш складними даними або застосувати інші методи оптимізації для подальшого вдосконалення алгоритму.

Додатки до лабораторної роботи

```
import pandas as pd
import numpy as np
import itertools
from math import log2
import time
import matplotlib.pyplot as plt
import networkx as nx

# Функція для обчислення взаємної інформації (MI)
def mutual_information(x, y):
    """Обчислення взаємної інформації між двома змінними."""
    x = pd.factorize(x)[0]  # Перетворення на категорійні індекси
    y = pd.factorize(y)[0]  # Перетворення на категорійні індекси
    joint_prob = np.zeros((x.max() + 1, y.max() + 1))

    for xi, yi in zip(x, y):
        joint_prob[xi, yi] += 1

    joint_prob /= len(x)
    x_prob = np.sum(joint_prob, axis=1)
    y_prob = np.sum(joint_prob, axis=0)
    mi = 0

    for i, j in itertools.product(range(len(x_prob)), range(len(y_prob))):
        if joint_prob[i, j] > 0:
```

```
            mi += joint_prob[i, j] * log2(joint_prob[i,
j] / (x_prob[i] * y_prob[j]))

    return mi

# Функція для обчислення MDL
def mdl_score(data, parents, child):
    n = len(data)
    if not parents:
        prob = np.bincount(data[:, child]) / n
        entropy = -np.sum(prob * np.log2(prob + 1e-9))
        return n * entropy
    else:
        parent_combinations = np.unique(data[:, parents],
axis=0)
        entropy = 0
        for comb in parent_combinations:
            mask = np.all(data[:, parents] == comb,
axis=1)
            subset = data[mask][:, child]
            prob = np.bincount(subset) / len(subset)
            entropy += len(subset) * -np.sum(prob *
np.log2(prob + 1e-9))
        return entropy

# Основний алгоритм побудови мережі Байєса
def bayesian_network_learning(data, feature_names):
    n_features = data.shape[1]
    mi_matrix = np.zeros((n_features, n_features))
    mdl_values = []
```

```
    # Перший етап: обчислення MI
    for i, j in itertools.combinations(range(n_features),
2):
        mi_matrix[i, j] = mutual_information(data[:, i],
data[:, j])
        mi_matrix[j, i] = mi_matrix[i, j]

    # Виведення таблиці значень взаємної інформації
    print("\nТаблиця значень взаємної інформації:")
    mi_df = pd.DataFrame(mi_matrix,
columns=feature_names, index=feature_names)
    print(mi_df.to_string(float_format="{:.6f}".format))

    # Сортування пар за значенням MI
    mi_pairs = [(i, j, mi_matrix[i, j]) for i in
range(n_features) for j in range(i + 1, n_features)]
    mi_pairs = sorted(mi_pairs, key=lambda x: -x[2])

    structure = {i: [] for i in range(n_features)}  #
Початкова структура
    for (i, j, mi) in mi_pairs:
        candidates = [(i, j), (j, i)]
        best_mdl = float('inf')
        best_structure = None
        for parent, child in candidates:
            structure[child].append(parent)
            mdl = sum(mdl_score(data, structure[node],
node) for node in range(n_features))
            mdl_values.append(mdl)
            if mdl < best_mdl:
                best_mdl = mdl
```

```
                best_structure = {node: parents.copy()
for node, parents in structure.items()}
            structure[child].remove(parent)
        if best_structure:
            structure = best_structure

    return structure, mdl_values, mi_matrix

# Функція для обчислення структурної різниці
def structural_difference(estimated_structure,
reference_structure):
    n = len(reference_structure)
    difference = 0
    extra_edges = 0
    missing_edges = 0
    reversed_edges = 0
    for i in range(n):
        parents_estimated =
set(estimated_structure.get(i, []))
        parents_reference =
set(reference_structure.get(i, []))

        # Зайві дуги
        extra_edges += len(parents_estimated -
parents_reference)

        # Відсутні дуги
        missing_edges += len(parents_reference -
parents_estimated)

        # Реверсовані дуги
```

```
        for parent in
parents_estimated.intersection(parents_reference):
            if parent not in reference_structure[i]:
                reversed_edges += 1

        difference += len(parents_estimated -
parents_reference) + len(parents_reference -
parents_estimated)
    return difference, extra_edges, missing_edges,
reversed_edges

# Завантаження даних
data = pd.read_csv('Asia.txt', delimiter='\t')
data_np = data.values.astype(int)  # Перетворення на цілі
числа

# Імена змінних
feature_names = [
    "Smoke", "Cancer", "Tuberculosis", "Tub_or_Cancer",
"Asia",
    "X_Ray", "Bronchitis", "Dyspnea"
]

# Еталонна структура з іменами змінних
reference_structure = {
    "Smoke": [],
    "Cancer": ["Smoke"],
    "Tuberculosis": ["Asia"],
    "Tub_or_Cancer": ["Cancer", "Tuberculosis"],
    "Asia": [],
    "X_Ray": ["Tub_or_Cancer"],
```

```
    "Bronchitis": ["Smoke"],
    "Dyspnea": ["Tub_or_Cancer", "Bronchitis"]
}

# Виконання алгоритму
start_time = time.time()
estimated_structure, mdl_values, mi_matrix =
bayesian_network_learning(data_np, feature_names)
end_time = time.time()

# Перетворення еталонної структури в числовий формат
reference_structure_numeric = {feature_names.index(node):
[feature_names.index(parent) for parent in parents] for
node, parents in reference_structure.items()}

# Обчислення результатів
execution_time_seconds = end_time - start_time
hours, rem = divmod(execution_time_seconds, 3600)
minutes, seconds = divmod(rem, 60)
milliseconds = (execution_time_seconds -
int(execution_time_seconds)) * 1000
execution_time =
f"{int(hours):02}:{int(minutes):02}:{int(seconds):02}:{in
t(milliseconds):03}"

structural_diff, extra_edges, missing_edges,
reversed_edges =
structural_difference(estimated_structure,
reference_structure_numeric)
independent_mdl = sum(mdl_score(data_np, [], i) for i in
range(len(reference_structure)))
```

```
# Виведення результатів обчислювального експерименту
print("\nРезультати обчислювального експерименту:")
print(f"Час роботи програми: {execution_time}")
print(f"Загальна кількість моделей, проаналізованих
запрограмованим методом: {len(data_np)}")
print(f"Кількість зайвих дуг: {extra_edges}")
print(f"Відсутні дуги: {missing_edges}")
print(f"Реверсовані дуги: {reversed_edges}")
print(f"Структурна різниця між побудованою та еталонною
структурами: {structural_diff}")
print(f"Значення функції ОМД для незалежних вершин:
{independent_mdl}")

# Виведення структури побудованої мережі
print("\nСтруктура побудованої мережі Байєса:")
for i, node in enumerate(estimated_structure):
    parents = [feature_names[parent] for parent in
estimated_structure[node]]
    print(f"Вершина {feature_names[i]}: батьки
{parents}")

# Графічне представлення побудованої мережі
graph = nx.DiGraph()
for child, parents in estimated_structure.items():
    for parent in parents:
        graph.add_edge(feature_names[parent],
feature_names[child])

plt.figure(figsize=(12, 8))
pos = nx.spring_layout(graph)
```

```
nx.draw(graph, pos, with_labels=True,
node_color='lightblue', edge_color='gray',
node_size=2000, font_size=10, font_weight='bold')
plt.title("Структура побудованої мережі Байєса")
plt.show()

# Графік зміни MDL
plt.figure(figsize=(10, 6))
plt.plot(mdl_values, marker='o', label='Значення MDL')
plt.axhline(y=independent_mdl, color='r', linestyle='--',
label='MDL для незалежних вершин')
plt.title("Графік зміни функції MDL залежно від
ітерацій")
plt.xlabel("Ітерації")
plt.ylabel("MDL")
plt.legend()
plt.grid()
plt.show()
```

Все детальніше можна переглянути за посиланням на github: https://github.com/Kinelan/Bayes/tree/main/Lab%203