

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA Công Nghệ Thông Tin

ĐỀ THI VÀ BÀI LÀM

Tên học phần: **Trí tuệ nhân tạo**

Mã học phần: Hình thức thi: *Tự luận có giám sát*

Đề số: **Đ0002** Thời gian làm bài: 75 phút (*không kể thời gian chép/phát đề*)

Được sử dụng tài liệu khi làm bài.

Họ tên:...Trần Phước Thành....**Lớp:**.....22T_Nhat2**MSSV:**.....102220339

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MSTeam:

Câu 1 (3 điểm): Trong các lâu đài cổ người ta thường xây dựng các đường hầm bí mật để thoát hiểm trong các trường hợp khẩn cấp. Các đường hầm chỉ có thể vào từ một cửa vào duy nhất tại phòng Trung tâm và thoát ra ở rất nhiều cửa ra. Các cửa ra đều nằm ở rìa lâu đài, do vậy, nếu thoát ra được rìa lâu đài thì coi như đã thoát hiểm. Để nguy trang, người ta cho đào nhiều nhánh hầm cụt và cửa vào giả. Ngoài ra, để tăng khả năng thoát hiểm, người ta còn xây dựng các đường hầm giao nhau tại một số vị trí. Để nghiệm thu công trình, chủ lâu đài cần kiểm tra xem từ phòng trung tâm có thể thoát hiểm qua hệ thống đường hầm hay không. Hãy sử dụng thuật toán A^* với hàm $f(x)$ là tổng chi phí được định nghĩa $f(x)=g(x)+h(x)$ (trong đó $g(x)$ -chi phí từ điểm xuất phát đến ô hiện tại, $h(x)$ -hàm ước lượng khoảng cách còn lại đến cửa ra, sử dụng khoảng cách Euclid) để giúp chủ lâu đài kiểm tra hệ thống trên.

Biết rằng lâu đài là một hình vuông được chia lưới ô vuông gồm n dòng, n cột. Trên đồ họa, ô ở dòng i cột j được ghi số 1 nếu có đường hầm, số 0 nếu không có (ô ở góc trên trái có tọa độ $(0,0)$). 2 ô chỉ có thể thông nhau nếu chúng có chung cạnh.

Dữ liệu nhập vào từ tập tin văn bản "[A_in.csv](#)" gồm:

- Dòng đầu chứa 3 số nguyên dương $n < 20$, D và C (trong đó D , C là dòng và cột của phòng trung tâm).
- n dòng tiếp theo, mỗi dòng chứa n số là các số ở các vị trí tương ứng trên họa đồ.

Kết quả tìm được ghi ra tập tin văn bản "**A_out.csv**". Dòng đầu chứa số m là số ô phải đi qua, nếu không thoát được thì $m = -1$. Trong trường hợp thoát được, m dòng tiếp theo: mỗi dòng chứa 2 số là số hiệu dòng cột của các ô phải đi qua theo đúng trình tự của một cách thoát hiểm.

Ví dụ: A_in.csv	Tệp A_out.csv
4 2 1	2
0 1 1 0	2 1
1 0 0 1	2 0
1 1 1 1	
0 1 1 0	

a) Xác định hàm $h(x)$

Trả lời: Minh hoạ giải thích hàm

Trong thuật toán A*, hàm h (hay heuristic) là hàm ước lượng chi phí còn lại từ một điểm hiện tại đến đích. Hàm này giúp thuật toán A* đánh giá và ưu tiên các bước đi tiếp theo dựa trên chi phí ước lượng này.

Mục đích:

+ Hàm này ước lượng khoảng cách từ vị trí hiện tại (x, y) đến rìa lâu đài. Trong ngữ cảnh của bài toán, rìa lâu đài là bất kỳ ô nào nằm trên biên của lưới (tức là các ô có tọa độ x = 0, x = n-1, y = 0, hoặc y = n-1).

Trả lời: Dán code hàm h(x)

```
def heuristic(x, y, n):  
    return min(x, n - 1 - x) + min(y, n - 1 - y)
```

b) Viết chương trình hoàn thiện cho bài toán trên

Trả lời: Dán code vào bên dưới

```
import csv  
import math  
from heapq import heappop, heappush  
  
def read_input(file_path):  
    with open(file_path, 'r') as file:  
        reader = csv.reader(file)  
        first_line = next(reader)  
        n = int(first_line[0])  
        start_row = int(first_line[1])  
        start_col = int(first_line[2])  
        grid = []  
        for line in reader:  
            grid.append(list(map(int, line)))  
    return n, start_row, start_col, grid  
  
def write_output(path, path_list):  
    with open(path, 'w', newline='') as file:  
        writer = csv.writer(file)  
        if path_list == -1:  
            writer.writerow([-1])  
        else:  
            writer.writerow([len(path_list)])  
            for position in path_list:  
                writer.writerow(position)  
  
def is_valid_move(x, y, n, grid, visited):  
    return 0 <= x < n and 0 <= y < n and grid[x][y] == 1 and (x, y) not in visited  
  
def heuristic(x, y, n):  
    return min(x, n - 1 - x) + min(y, n - 1 - y)  
  
def a_star(n, start_row, start_col, grid):  
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  
    open_set = [(heuristic(start_row, start_col, n), 0, start_row, start_col,  
[(start_row, start_col)])]
```

```

visited = set()

while open_set:
    _, g, x, y, path = heappop(open_set)

    if x in [0, n-1] or y in [0, n-1]:
        return path # Escape found

    visited.add((x, y))

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if is_valid_move(nx, ny, n, grid, visited):
            new_g = g + 1
            f = new_g + heuristic(nx, ny, n)
            heappush(open_set, (f, new_g, nx, ny, path + [(nx, ny)]))

    return -1 # No escape found

def main():
    input_file = "A_in.csv"
    n, start_row, start_col, grid = read_input(input_file)
    path = a_star(n, start_row, start_col, grid)
    output_file = 'A_out.csv'
    write_output(output_file, path)
    print(f"Output file has been written successfully to {output_file}!")

if __name__ == "__main__":
    main()

```

Trả lời: Giải thích chương trình

Chương trình này sử dụng thuật toán A* để tìm đường thoát hiểm từ phòng trung tâm của một lâu đài đến rìa lâu đài thông qua các đường hầm.

+ Đọc dữ liệu từ file CSV:

Hàm `read_input(file_path)` đọc dữ liệu từ file CSV `A_in.csv`. Dòng đầu tiên chứa kích thước lưới `n`, vị trí bắt đầu `start_row` và `start_col`. Các dòng tiếp theo chứa lưới `grid` với các giá trị 1 (có đường hầm) và 0 (không có đường hầm).

+ Ghi kết quả vào file CSV:

Hàm `write_output(path, path_list)` ghi kết quả đường đi vào file CSV `A_out.csv`. Nếu không tìm thấy đường đi, ghi -1, nếu có, ghi độ dài và các vị trí của đường đi.

+ Kiểm tra di chuyển hợp lệ:

Hàm `is_valid_move(x, y, n, grid, visited)` kiểm tra xem di chuyển đến vị trí `(x, y)` có hợp lệ không (trong lưới, có đường hầm, và chưa được thăm).

+ Hàm heuristic:


Hàm `heuristic(x, y, n)` tính khoảng cách ước lượng từ vị trí hiện tại `(x, y)` đến rìa lâu đài. Sử dụng khoảng cách Euclid để ước lượng.

+ Thuật toán A*:

Hàm `a_star(n, start_row, start_col, grid)` sử dụng thuật toán A* để tìm đường thoát từ vị trí bắt đầu. Sử dụng hàng đợi ưu tiên (priority queue) để chọn bước đi tiếp theo dựa trên chi phí ước lượng $f(x) = g(x) + h(x)$, trong đó $g(x)$ là chi phí từ điểm xuất phát đến ô hiện tại, và $h(x)$ là hàm ước lượng khoảng cách còn lại đến cửa ra.

c) Kết quả thực thi trên tệp "A_out.csv"

Trả lời: Dán kết quả vào bên dưới

 A_out.csv

1	8
2	5,5
3	5,6
4	4,6
5	4,7
6	3,7
7	3,8
8	2,8
9	2,9
10	

Trả lời: Nộp kết quả A_out.csv cùng với tệp bài làm

8	
5	5
5	6
4	6
4	7
3	7
3	8
2	8
2	9



cau1.ipynb

Câu 2 (4 điểm): Cho tập dữ liệu [input.csv](#) với 75 mẫu dữ liệu, mỗi mẫu có 4 đặc trưng (chiều dài đài hoa, chiều rộng đài hoa, chiều dài cánh hoa, chiều rộng cánh hoa) và tên loài hoa tương ứng.

a) (1 điểm) Xây dựng hàm mục tiêu (hàm mất mát) cho bài toán

Trả lời: Dán hàm mất mát vào đây:

```
def train(x_train, y_train, w, lr, num_of_iteration):
    losses = []
    for epoch in range(num_of_iteration):
        z = np.dot(x_train, w).astype(np.float32)
        y_predict = softmax(z)
        epsilon = 1e-7
        loss = -np.sum(y_train * np.log(y_predict + epsilon))
        losses.append(loss)
        dz = y_predict - y_train
        dw = np.dot(x_train.T, dz)
        w = w - lr*dw
        print(f"Epoch: {epoch}    Loss: {losses[-1]}")
    return w, losses
```

Trả lời: Dán code của hàm loss:

```

def train(x_train, y_train, w, lr, num_of_iteration):
    losses = []
    for epoch in range(num_of_iteration):
        z = np.dot(x_train, w).astype(np.float32)
        y_predict = softmax(z)
        epsilon = 1e-7
        loss = -np.sum(y_train * np.log(y_predict + epsilon))
        losses.append(loss)
        dz = y_predict - y_train
        dw = np.dot(x_train.T, dz)
        w = w - lr*dw
        print(f"Epoch: {epoch}      Loss: {losses[-1]}")
    return w, losses

```

b) (2 điểm) Hãy viết chương trình phân loại hoa trên cơ sở dùng Logistic Regression kết hợp với lớp softmax.

```

# Trả lời: Dán code vào đây
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

# Load data from CSV file
def get_data(file_name, labels=True, header=None):
    try:
        # Read data from CSV file
        data = pd.read_csv(file_name, header=header).values
        N, d = data.shape
        # Split data into input x and output y
        if labels:
            x = data[:, 0:d-1].reshape(-1, d-1)
            y = data[:, d - 1].reshape(-1, 1)
        else:
            x = data[:, 0:d].reshape(-1, d)
            y = data[:, 0].reshape(-1, 1)
        return x, y
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
        raise

# Add bias and modify labels
def prepare_data(x, y, train=True):
    N = x.shape[0]
    y_train = np.array([])
    x_train = np.array([])
    if train:
        for d in y:
            # Convert labels
            if d == "Iris-setosa":
                y_train = np.append(y_train, 0)
            elif d == "Iris-versicolor":

```

```

        y_train = np.append(y_train, 1)
    else:
        y_train = np.append(y_train, 2)
    y_train = y_train.reshape(-1, 1)
    # Add bias column
    x_train = np.hstack((np.ones((N, 1)), x))
    return x_train, y_train

# Save data to CSV file
def set_data(input_filename, y_val_predict, output_filename):
    try:
        df = pd.read_csv(input_filename, header=None)
        class_names = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
        df['predict'] = y_val_predict
        df['predict'] = df['predict'].map(lambda x: class_names[x])
        df.to_csv(output_filename, header=False, index=False)
    except Exception as e:
        print(f"Error saving file {output_filename}: {e}")
        raise

# Convert y to one-hot encoding
def one_hot_encoding(y):
    N = y.shape[0]
    K = len(np.unique(y))
    Y = np.zeros((N, K))
    for i in range(N):
        Y[i, y[i]] = 1
    return Y

def softmax(z):
    e_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return e_z / np.sum(e_z, axis=1, keepdims=True)

def train(x_train, y_train, w, lr, num_of_iteration):
    losses = []
    for epoch in range(num_of_iteration):
        z = np.dot(x_train, w).astype(np.float32)
        y_predict = softmax(z)
        epsilon = 1e-7
        loss = -np.sum(y_train * np.log(y_predict + epsilon))
        losses.append(loss)
        dz = y_predict - y_train
        dw = np.dot(x_train.T, dz)
        w = w - lr*dw
        print(f"Epoch: {epoch}    Loss: {losses[-1]}")
    return w, losses

if __name__ == "__main__":
    X, Y = get_data('input.csv')
    x_train, y_train = prepare_data(X, Y)
    y_train = y_train.astype('uint8')
    y_train = one_hot_encoding(y_train)

```

```

w = np.multiply(0.01, np.random.randn(x_train.shape[1], y_train.shape[1]))
lr = 0.001
num_of_iteration = 500
print(x_train.shape, y_train.shape, w.shape)

w, losses = train(x_train, y_train, w, lr, num_of_iteration)

x_axis = np.arange(len(losses))
plt.plot(x_axis, losses, color='r')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.show()

y_predict1 = softmax(np.dot(x_train, w).astype(np.float32))
y_predict1 = np.argmax(y_predict1, axis=1)

y_true = np.argmax(y_train, axis=1)

accuracy = np.mean(y_predict1 == y_true)
print("Training accuracy:", accuracy * 100, "%")

x_val, y_val = get_data('output.csv', False)
x_val, y_val = prepare_data(x_val, y_val, False)

y_val_predict = softmax(np.dot(x_val, w).astype(np.float32))
y_val_predict = np.argmax(y_val_predict, axis=1)
print(y_val_predict)
set_data('output.csv', y_val_predict, 'result.csv')

```

Trả lời: Gián kiến trúc mạng và giải thích làm thế nào để phân loại ?

Kiến trúc mạng:

Lớp input (4 neurons): Đầu vào của mạng là các đặc trưng của hoa Iris, bao gồm độ dài và độ rộng của lá đài và cánh hoa (tổng cộng 4 đặc trưng).

Lớp ẩn (1 hidden layer, 10 neurons): Mạng có một lớp ẩn với 10 nơ-ron. Các nơ-ron trong lớp ẩn này sẽ học các biểu diễn phức tạp của dữ liệu đầu vào và trích xuất các đặc trưng quan trọng để phân loại.

Hàm kích hoạt: Hàm kích hoạt ReLU được sử dụng sau lớp ẩn để tạo ra không gian phi tuyến tính và giúp mạng học được các đặc trưng phức tạp.

Lớp output (3 neurons): Lớp đầu ra có 3 nơ-ron tương ứng với 3 loài hoa Iris. Mỗi nơ-ron ở lớp này đại diện cho xác suất của một loài hoa cụ thể.

Phương pháp phân loại:

Huấn luyện mạng nơ-ron: Dữ liệu huấn luyện gồm các điểm dữ liệu có các đặc trưng của hoa Iris và nhãn tương ứng (loài hoa).

Chuyển đổi dữ liệu thành tensor: Dữ liệu đầu vào được chuyển đổi thành tensor và được

chia thành các batch để đưa vào mạng nơ-ron.

Feedforward (lan truyền tiến): Dữ liệu được đưa qua mạng theo chiều tiến để tính toán đầu ra của mạng.

Tính toán hàm mất mát (loss): So sánh đầu ra dự đoán với nhãn thực tế để tính toán độ lỗi của mạng, thông qua hàm CrossEntropyLoss.

Backpropagation (lan truyền ngược): Sử dụng giải thuật lan truyền ngược để tính gradient của hàm mất mát theo các tham số của mạng.

Cập nhật tham số: Các tham số của mạng được cập nhật theo hướng giảm gradient để giảm thiểu độ lỗi.

Lặp lại quá trình huấn luyện: Quá trình từ bước 3 đến bước 6 được lặp lại nhiều lần (trong ví dụ này là 100 epochs) để mạng học được càng nhiều từ dữ liệu huấn luyện.

Đánh giá mô hình: Mô hình được đánh giá bằng cách tính độ chính xác trên tập kiểm tra, để biết mức độ mà mô hình có thể phân loại đúng loài hoa Iris từ dữ liệu mới.

Trong quá trình huấn luyện và đánh giá, mạng nơ-ron cố gắng học và trích xuất các mối quan hệ phức tạp giữa các đặc trưng của hoa Iris để phân loại chúng vào các loài khác nhau dựa trên dữ liệu huấn luyện có sẵn.

c) (1 điểm) Hãy thực thi chương trình và cho biết nhãn của 30 mẫu dữ liệu trong [output.csv](#)

Trả lời: Dán code thực thi thành công

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

# Load data from CSV file
def get_data(file_name, labels=True, header=None):
    try:
        # Read data from CSV file
        data = pd.read_csv(file_name, header=header).values
        N, d = data.shape
        # Split data into input x and output y
        if labels:
            x = data[:, 0:d-1].reshape(-1, d-1)
            y = data[:, d - 1].reshape(-1, 1)
        else:
            x = data[:, 0:d].reshape(-1, d)
            y = data[:, 0].reshape(-1, 1)
        return x, y
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
        raise

# Add bias and modify labels
def prepare_data(x, y, train=True):
    N = x.shape[0]
    y_train = np.array([])
```



```

x_train = np.array([])
if train:
    for d in y:
        # Convert labels
        if d == "Iris-setosa":
            y_train = np.append(y_train, 0)
        elif d == "Iris-versicolor":
            y_train = np.append(y_train, 1)
        else:
            y_train = np.append(y_train, 2)
    y_train = y_train.reshape(-1, 1)
# Add bias column
x_train = np.hstack((np.ones((N, 1)), x))
return x_train, y_train

# Save data to CSV file
def set_data(input_filename, y_val_predict, output_filename):
    try:
        df = pd.read_csv(input_filename, header=None)
        class_names = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
        df['predict'] = y_val_predict
        df['predict'] = df['predict'].map(lambda x: class_names[x])
        df.to_csv(output_filename, header=False, index=False)
    except Exception as e:
        print(f"Error saving file {output_filename}: {e}")
        raise

# Convert y to one-hot encoding
def one_hot_encoding(y):
    N = y.shape[0]
    K = len(np.unique(y))
    Y = np.zeros((N, K))
    for i in range(N):
        Y[i, y[i]] = 1
    return Y

def softmax(z):
    e_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return e_z / np.sum(e_z, axis=1, keepdims=True)

def train(x_train, y_train, w, lr, num_of_iteration):
    losses = []
    for epoch in range(num_of_iteration):
        z = np.dot(x_train, w).astype(np.float32)
        y_predict = softmax(z)
        epsilon = 1e-7
        loss = -np.sum(y_train * np.log(y_predict + epsilon))
        losses.append(loss)
        dz = y_predict - y_train
        dw = np.dot(x_train.T, dz)
        w = w - lr*dw
        print(f"Epoch: {epoch}      Loss: {losses[-1]}")

```

```

    return w, losses

if __name__ == "__main__":
    X, Y = get_data('input.csv')
    x_train, y_train = prepare_data(X, Y)
    y_train = y_train.astype('uint8')
    y_train = one_hot_encoding(y_train)

    w = np.multiply(0.01, np.random.randn(x_train.shape[1], y_train.shape[1]))
    lr = 0.001
    num_of_iteration = 500
    print(x_train.shape, y_train.shape, w.shape)

    w, losses = train(x_train, y_train, w, lr, num_of_iteration)

    x_axis = np.arange(len(losses))
    plt.plot(x_axis, losses, color='r')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss Over Time')
    plt.show()

    y_predict1 = softmax(np.dot(x_train, w).astype(np.float32))
    y_predict1 = np.argmax(y_predict1, axis=1)

    y_true = np.argmax(y_train, axis=1)

    accuracy = np.mean(y_predict1 == y_true)
    print("Training accuracy:", accuracy * 100, "%")

    x_val, y_val = get_data('output.csv', False)
    x_val, y_val = prepare_data(x_val, y_val, False)

    y_val_predict = softmax(np.dot(x_val, w).astype(np.float32))
    y_val_predict = np.argmax(y_val_predict, axis=1)
    print(y_val_predict)
    set_data('output.csv', y_val_predict, 'result.csv')

```

Trả lời: Dán kết quả nhận ứng với 30 mẫu dữ liệu

```
[0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2]
```

result.csv

```
1 5.4,3.7,1.5,0.2,Iris-setosa
2 4.8,3.4,1.6,0.2,Iris-setosa
3 4.8,3.0,1.4,0.1,Iris-setosa
4 4.3,3.0,1.1,0.1,Iris-setosa
5 5.8,4.0,1.2,0.2,Iris-setosa
6 5.7,4.4,1.5,0.4,Iris-setosa
7 5.4,3.9,1.3,0.4,Iris-setosa
8 5.1,3.5,1.4,0.3,Iris-setosa
9 5.7,3.8,1.7,0.3,Iris-setosa
10 5.1,3.8,1.5,0.3,Iris-setosa
11 7.0,3.2,4.7,1.4,Iris-versicolor
12 6.4,3.2,4.5,1.5,Iris-versicolor
13 6.9,3.1,4.9,1.5,Iris-versicolor
14 5.5,2.3,4.0,1.3,Iris-versicolor
15 6.5,2.8,4.6,1.5,Iris-versicolor
16 5.7,2.8,4.5,1.3,Iris-versicolor
17 6.3,3.3,4.7,1.6,Iris-versicolor
18 4.9,2.4,3.3,1.0,Iris-versicolor
19 6.6,2.9,4.6,1.3,Iris-versicolor
20 5.2,2.7,3.9,1.4,Iris-versicolor
21 5.0,2.0,3.5,1.0,Iris-versicolor
22 5.9,3.0,4.2,1.5,Iris-versicolor
23 6.0,2.2,4.0,1.0,Iris-versicolor
24 5.8,2.7,5.1,1.9,Iris-virginica
25 7.1,3.0,5.9,2.1,Iris-virginica
26 6.3,2.9,5.6,1.8,Iris-virginica
27 6.5,3.0,5.8,2.2,Iris-virginica
28 7.6,3.0,6.6,2.1,Iris-virginica
29 4.9,2.5,4.5,1.7,Iris-virginica
30 7.3,2.9,6.3,1.8,Iris-virginica
```

Câu 3 (3 điểm): Cho tập dữ liệu [Countries.csv](#). Hãy viết chương trình phân cụm bằng thuật toán k -means

a) (1 điểm) Xây dựng hàm đo khoảng cách sử dụng độ đo Euclid

Trả lời: Minh họa tính khoảng cách:

```
point1 = [1, 2]
```

```
point2 = [4, 6]
```

```
distance = euclidean_distance(point1, point2)
```

```
print(f'Khoảng cách Euclid giữa {point1} và {point2} là {distance}')
```

Trả lời: Dán code hàm tính khoảng cách:

```
import numpy as np

def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((np.array(point1) - np.array(point2)) ** 2))
```

b) (1 điểm) Xây dựng hàm chứa thuật toán k -means để phân cụm

```
# Trả lời: Dán code về hàm
def k_means_clustering(data, n_clusters):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(scaled_data)
    return kmeans.labels_, kmeans.cluster_centers_
```

c) (1 điểm) Xây dựng hàm để khảo sát việc lựa chọn k

```
# Trả lời: Dán code về hàm và giải thích cách lựa chọn k phù hợp
def find_optimal_k(data, max_k=10):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)
    wcss = [] # Within-cluster sums of squares

    for k in range(1, max_k + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_data)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize=(8, 4))
    plt.plot(range(1, max_k + 1), wcss, marker='o')
    plt.title('Elbow Method for Finding Optimal k')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()

    optimal_k = 0
    for i in range(1, len(wcss) - 1): # Bỏ qua điểm đầu và cuối
        drop1 = wcss[i - 1] - wcss[i]
        drop2 = wcss[i] - wcss[i + 1]
        if drop1 > 2 * drop2: # Logic "góc khuỷu tay" đơn giản
            optimal_k = i + 1 # Vì index bắt đầu từ 0
            break

    print(f'Giá trị k tối ưu được đề xuất: {optimal_k}')
    return optimal_k
```

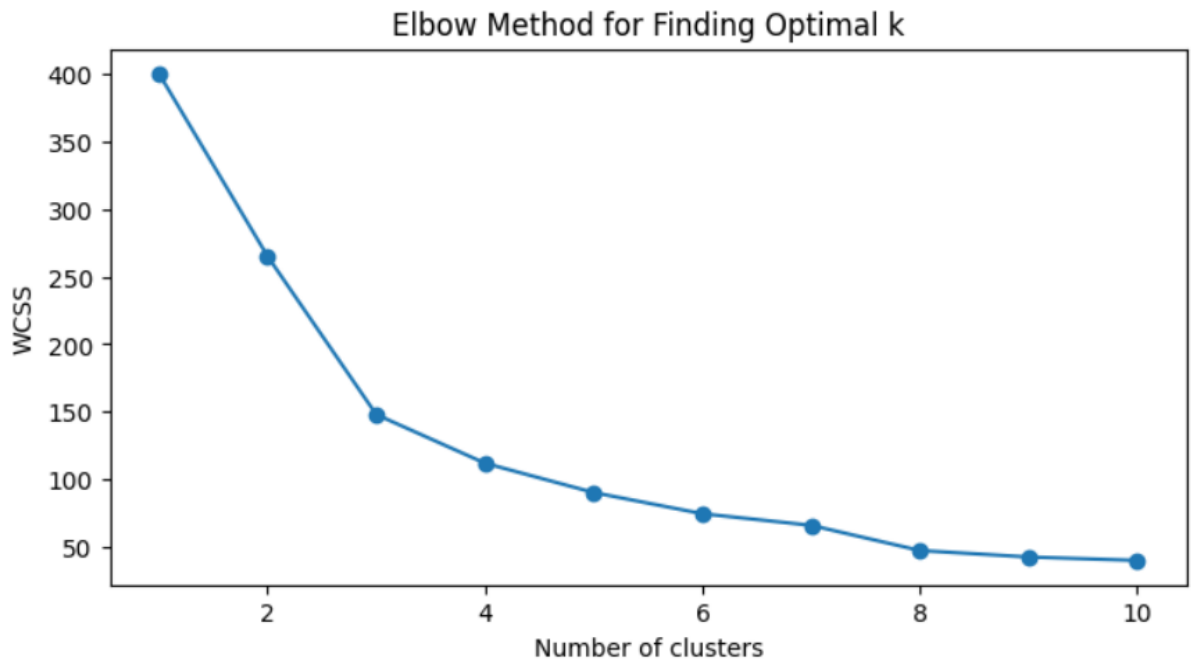
Giải thích việc chọn k :

+ Chuẩn hoá dữ liệu: chuyển đổi dữ liệu về dạng chuẩn hoá để đảm bảo rằng các đơn vị có đặc tính đo khác nhau sẽ không bị ảnh hưởng đến quá trình phân cụm.

+ Tính toán WCSS cho các giá trị từ $k = 1$ đến max_k : WCSS là tổng bình phương khoảng cách từ các điểm dữ liệu đến tâm cụm của chúng.

+ Vẽ biểu đồ Elbow: Biểu đồ hiển thị giá trị WCSS cho mỗi giá trị k . Điểm gấp khúc trên biểu đồ (elbow point) là nơi mà WCSS giảm dần chậm hơn khi số cụm tăng lên điều này chỉ ra số cụm tối ưu

Trả lời: Dán kết quả thi với k (lưu ý có giải thích và bình luận):



Giá trị k tối ưu được đề xuất: 3

```
[[ 0.88448091 -0.77575245]
 [-1.1277947 -0.48910321]
 [ 0.17359932  0.97046952]]
```

Phân cụm hoàn tất. Kết quả đã được lưu vào `Countries_with_clusters.csv`

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

Đà Nẵng, ngày 20 tháng 11 năm 2024

TRƯỞNG BỘ MÔN
(đã duyệt)