|  |
| --- |
| 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: **Đ0001** Thời gian làm bài: 75 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên:Nguyễn Phan Thanh Lớp**: 22T\_DT2……………………………..**MSSV**:102220126

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*): Một lâu đài cổ có hệ thống đường hầm bí mật, với một cửa vào duy nhất tại phòng trung tâm và nhiều cửa ra ở rìa lâu đài. Để đánh lạc hướng, hệ thống có thêm các nhánh hầm cụt và cửa giả. Hai ô hầm chỉ nối với nhau nếu có chung cạnh. Hãy giúp chủ lâu đài kiểm tra khả năng thoát hiểm từ phòng trung tâm đến rìa lâu đài bằng thuật toán A\*, với hàm chi phí:

* f(x) = g(x) + h(x), trong đó:
  + g(x): chi phí từ điểm bắt đầu đến ô hiện tại.
  + h(x): khoảng cách Manhattan đến rìa lâu đài.

#### Dữ liệu đầu vào (file “A\_in.csv”):

* Dòng 1: ba số nguyên dương n , D, C — kích thước lâu đài và tọa độ phòng trung tâm.
* Dòng 2 đến n+1: ma trận n x n, mỗi ô là:
  + 1: có đường hầm (đi được),
  + 0: không có (không đi được).

#### Kết quả đầu ra (file “A\_out.csv”):

* Nếu không thoát được: ghi -1.
* Nếu thoát được:
  + Dòng đầu: số ô phải đi qua (m).
  + Tiếp theo m dòng: tọa độ các ô (dòng, cột) theo thứ tự đường đi từ phòng trung tâm đến một cửa ra.

Dữ liệu minh hoạ

|  |  |
| --- | --- |
| A\_in.csv 5,2,2  0,0,1,0,0  0,1,1,1,0  0,0,1,0,0  1,1,1,0,0  0,0,1,1,1 | A\_out.csv 6  2,2  1,2  1,1  2,1  3,1  3,0 |

1. Xác định hàm h(x)

|  |
| --- |
| **# Trả lời:** Minh hoạ giải thích hàm  Hàm h(x) là khoảng cách Manhattan từ ô hiện tại (i,j) đến rìa gần nhất của lưới n×n. Rìa là các hàng  i = 0, i = n-1, hợc cột j =0, j =n-1  Công thức:    i: khoảng cách đến rìa trên  n-1-i: khoảng cách đến rìa dưới  j: khoảng cách đến rìa trái  n-1-j: khoảng cách đến rìa phải  **# Trả lời:** Dán code hàm h(x)  def manhattan\_to\_exit(i, j, n):      """Tính hàm h(x) - khoảng cách Manhattan đến cửa ra gần nhất"""      return min(i, n-1-i, j, n-1-j) |

1. Viết chương trình hoàn thiện cho bài toán trên

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  **Code:**  import heapq  def manhattan\_to\_exit(i, j, n):      """Tính hàm h(x) - khoảng cách Manhattan đến cửa ra gần nhất"""      return min(i, n-1-i, j, n-1-j)  def is\_exit(i, j, n):      """Kiểm tra xem ô (i, j) có phải là cửa ra (rìa lâu đài)"""      return i == 0 or i == n-1 or j == 0 or j == n-1  def a\_star(grid, start, n):      """Thuật toán A\* để tìm đường thoát hiểm"""      # Hướng di chuyển: lên, xuống, trái, phải      directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]        # Hàng đợi ưu tiên cho A\*      open\_list = []      heapq.heappush(open\_list, (0, start, [start]))  # (f(x), (i, j), path)        # Lưu chi phí g(x) và đường đi      g = {start: 0}      visited = set()        while open\_list:          f, (i, j), path = heapq.heappop(open\_list)            if (i, j) in visited:              continue            visited.add((i, j))            # Kiểm tra nếu ô hiện tại là cửa ra          if is\_exit(i, j, n):              return path            # Duyệt các ô kề          for di, dj in directions:              ni, nj = i + di, j + dj              if 0 <= ni < n and 0 <= nj < n and grid[ni][nj] == 1 and (ni, nj) not in visited:                  new\_g = g[(i, j)] + 1  # Chi phí từ điểm xuất phát đến ô kề                  if (ni, nj) not in g or new\_g < g[(ni, nj)]:                      g[(ni, nj)] = new\_g                      h = manhattan\_to\_exit(ni, nj, n)                      f = new\_g + h                      new\_path = path + [(ni, nj)]                      heapq.heappush(open\_list, (f, (ni, nj), new\_path))        return None  # Không tìm được đường thoát  def solve\_escape\_tunnel():      # Đọc dữ liệu từ file      with open("A\_in.csv", "r") as f:          n, D, C = map(int, f.readline().strip().split(','))          grid = [list(map(int, f.readline().strip().split(','))) for \_ in range(n)]        # Tìm đường thoát bằng A\*      start = (D, C)      path = a\_star(grid, start, n)        # Ghi kết quả ra file A\_out.csv      with open("A\_out.csv", "w") as f:          if path is None:              f.write("-1\n")          else:              f.write(f"{len(path)}\n")              for i, j in path:                  f.write(f"{i} {j}\n")  # Thực thi chương trình  solve\_escape\_tunnel()  **# Trả lời:** Giải thích chương trình  **Các hàm chính:**   1. manhattan\_to\_exit(i, j, n): Tính heuristic - khoảng cách Manhattan từ vị trí hiện tại đến cửa ra gần nhất (rìa mê cung) 2. is\_exit(i, j, n): Kiểm tra xem vị trí có phải là cửa ra không (ở rìa lưới) 3. a\_star(grid, start, n): Thuật toán A\* chính:    * Sử dụng priority queue (heap) để chọn ô có f(x) = g(x) + h(x) nhỏ nhất    * g(x): chi phí thực tế từ điểm bắt đầu    * h(x): ước lượng chi phí đến đích (Manhattan distance)    * Duyệt 4 hướng: lên, xuống, trái, phải    * Chỉ đi qua ô có giá trị = 1 (đường đi) 4. solve\_escape\_tunnel(): Hàm chính:    * Đọc input từ [A\_in.csv](vscode-file://vscode-app/c:/Users/nptha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): kích thước n, vị trí bắt đầu (D,C), ma trận mê cung    * Gọi A\* để tìm đường    * Ghi kết quả vào [A\_out.csv](vscode-file://vscode-app/c:/Users/nptha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): số bước + tọa độ từng bước   **Input/Output:**   * **Input**: File CSV với dòng đầu là [n,D,C](vscode-file://vscode-app/c:/Users/nptha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) và n dòng tiếp theo là ma trận 0/1 * **Output**: Số bước + danh sách tọa độ đường đi (hoặc -1 nếu không tìm được) |

1. Kết quả thực thi trên tệp “[A\_in.csv](https://drive.google.com/file/d/1uv4FzTY8qbCrupKO7fTdbXvk0aj9IRgd/view?usp=sharing)”

|  |
| --- |
| **# Trả lời:** Dán kết quả kết quả A\_out.csv vào bên dưới  7  5 5  5 6  5 7  5 8  6 8  7 8  7 9 |

***Câu 2*** (*4 điểm*): Cho tập dữ liệu [input.csv](https://drive.google.com/file/d/1BXZywZ_Bxfq-g2rErlXOr5HxyznwqaCe/view?usp=sharing) 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.

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

|  |
| --- |
| **# Trả lời:** Dán hàm mất mát vào đây:  **Softmax:**  Công thức:    Trong đó:  Z: Vecto đầu vào  J: Chỉ số lớp  C: Số lớp  Softmax: Xác suất dự đoán cho lớp j  **Cross-Entropy Loss (phân loại đa lớp)**  **Công thức:**  Ảnh có chứa Phông chữ, văn bản, chữ viết tay, thuật in máy  Nội dung do AI tạo ra có thể không chính xác.  Trong đó:  N: số mẫu  C: Số lớp  y(i,c): Nhãn thực tế  y^(I,c): Xác suất dự đoán từ softmax  **# Trả lời:** Dán code của hàm loss:  import numpy as np  def cross\_entropy\_loss(y\_true, y\_pred):  """Tính mất mát cross-entropy (slide Lesson 13, trang 20)"""  n\_samples = len(y\_true)  y\_pred = np.clip(y\_pred, 1e-15, 1 - 1e-15) # Tránh log(0)  loss = -np.sum(y\_true \* np.log(y\_pred)) / n\_samples  return loss   def softmax(self, z):          exp\_z = np.exp(z - np.max(z, axis=1, keepdims=True))  # Tránh tràn số          return exp\_z / np.sum(exp\_z, axis=1, keepdims=True) |

1. (2 điểm) Hãy viết chương trình phân loại hoa trên.

|  |
| --- |
| **# Trả lời:** Dán code vào đây  **Code:**  import numpy as np  import pandas as pd  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  class SoftmaxRegression:      def \_\_init\_\_(self, lr=0.01, n\_iters=1000):          self.lr = lr          self.n\_iters = n\_iters          self.weights = None          self.bias = None        def softmax(self, z):          exp\_z = np.exp(z - np.max(z, axis=1, keepdims=True))  # Tránh tràn số          return exp\_z / np.sum(exp\_z, axis=1, keepdims=True)        def fit(self, X, y):          n\_samples, n\_features = X.shape          # Encode string labels to integer indices          self.classes\_, y\_indices = np.unique(y, return\_inverse=True)          n\_classes = len(self.classes\_)            # Khởi tạo trọng số và bias          self.weights = np.zeros((n\_features, n\_classes))          self.bias = np.zeros(n\_classes)            # Mã hóa one-hot cho nhãn          y\_one\_hot = np.zeros((n\_samples, n\_classes))          for i in range(n\_samples):              y\_one\_hot[i, y\_indices[i]] = 1            # Gradient Descent          for \_ in range(self.n\_iters):              # Tính xác suất Softmax              linear\_model = np.dot(X, self.weights) + self.bias              y\_pred = self.softmax(linear\_model)                # Tính gradient              grad\_w = (1 / n\_samples) \* np.dot(X.T, (y\_pred - y\_one\_hot))              grad\_b = (1 / n\_samples) \* np.sum(y\_pred - y\_one\_hot, axis=0)                # Cập nhật tham số              self.weights -= self.lr \* grad\_w              self.bias -= self.lr \* grad\_b        def predict(self, X):          linear\_model = np.dot(X, self.weights) + self.bias          y\_pred = self.softmax(linear\_model)          class\_indices = np.argmax(y\_pred, axis=1)          # Convert indices back to original class labels          return self.classes\_[class\_indices]  # Tải và tiền xử lý dữ liệu Iris  iris = pd.read\_csv("input\_2.csv")  X, y = iris.iloc[:, :-1].values, iris.iloc[:, -1].values  # Chuẩn hóa dữ liệu  scaler = StandardScaler()  X = scaler.fit\_transform(X)  # Đọc dữ liệu test từ file csv, đảm bảo đọc đúng format  # Read first few lines to check if there's a header  with open("output\_2.csv", "r") as f:      first\_line = f.readline().strip()      # Check if the first line looks like a header or data      if ',' in first\_line and all(c.isdigit() or c in '.,+-' for c in first\_line.replace(',', '')):          # Looks like data, no header          X\_test = pd.read\_csv("output\_2.csv", header=None)      else:          # Has a header          X\_test = pd.read\_csv("output\_2.csv")  print(f"Number of test samples: {X\_test.shape[0]}")  print(f"X\_test data shape: {X\_test.shape}")  # Huấn luyện mô hình  model = SoftmaxRegression(lr=0.1, n\_iters=1000)  model.fit(X, y)  # Dự đoán trên tất cả mẫu test  # Chuẩn hóa dữ liệu test giống như dữ liệu train  X\_test\_scaled = scaler.transform(X\_test.values)  # Dự đoán nhãn cho dữ liệu test  predictions = model.predict(X\_test\_scaled)  print(f"Number of predictions: {len(predictions)}")  # Ghi kết quả ra file  with open("iris\_predictions.csv", "w") as f:      f.write("Sample,Predicted\_Label\n")      for i, pred in enumerate(predictions):          f.write(f"{i+1},{pred}\n")  print("Done writing predictions. Last sample number:", len(predictions))  **# Trả lời:** Gián kiến trúc mạng và giải thích làm thế nào để phân loại ?  **Kiến trúc mạng Softmax Regression**  **1. Tổng quan kiến trúc:**  - **\*\*Input Layer\*\***: n\_features đặc trưng đầu vào (4 đặc trưng cho dataset Iris)  - **\*\*Linear Layer\*\***: Ma trận trọng số W (4 x 3) + bias b (3)  - **\*\*Softmax Activation\*\***: Chuyển đổi logits thành xác suất  - **\*\*Output Layer\*\***: 3 lớp tương ứng với 3 loài hoa Iris  **2. Công thức toán học:**  ```  z = X @ W + b  :Linear transformation  y\_pred = softmax(z)  : Activation function  Softmax function  Cross\_entropy\_loss  ```  **3. Kiến trúc chi tiết:**  ```  Input (75 samples x 4 features)      ↓  Linear Layer: W(4x3) + b(3)      ↓  Logits (75 x 3)      ↓  Softmax Activation      ↓  Probabilities (75 x 3)      ↓  Argmax → Predicted Class  ```  **Các bước để phân loại:**  **Bước 1: Chuẩn bị dữ liệu**  ```python  Input: x = [5,3.2,1.2,0.2]  ```  **Bước 2: Linear Transformation (Tính toán tuyến tính)**  z = X @ W + b  - **X**: Ma trận input (n\_samples × 4 features)  - **W**: Ma trận trọng số (4 × 3 classes)  - **b**: Vector bias (3,)  - **z**: Vector logits (3,) - điểm số thô cho mỗi class  **Bước 3: Softmax Activation (Chuyển đổi thành xác suất)**  ```  y\_pred = softmax(z) = exp(z\_i) / Σ(exp(z\_j))  ```  - Chuyển đổi logits thành xác suất [0,1]  - Tổng các xác suất = 1  - Class nào có xác suất cao nhất → Dự đoán  **Bước 4: Prediction (Đưa ra dự đoán)**  ```  predicted\_class = argmax(y\_pred)  ```  - Chọn class có xác suất cao nhất  - Trả về tên class (setosa, versicolor, virginica)  **Ví dụ cụ thể:**  ```  Input: [5.1, 3.5, 1.4, 0.2]  ↓  Logits: [-2.1, 0.8, -1.3]  ↓  Softmax: [0.05, 0.89, 0.06]  ↓  Prediction: versicolor (index=1, confidence=89%)  ``` |

1. *(1 điểm)* Hãy thực thi chương trình và cho biết nhãn của 30 mẫu dữ liệu trong [output.csv](https://drive.google.com/file/d/1ok6hOkd2YSpnCzxmXdPsX0qAT8g7O-Vp/view?usp=sharing)

|  |
| --- |
| **# Trả lời:** Dán code thực thi thành công  Number of test samples: 30  X\_test data shape: (30, 4)  Number of predictions: 30  Done writing predictions. Last sample number: 30  **# Trả lời:** Dán kết quả nhãn ứng với 30 mẫu dữ liệu  Sample,Predicted\_Label  1,Iris-setosa  2,Iris-setosa  3,Iris-setosa  4,Iris-setosa  5,Iris-setosa  6,Iris-setosa  7,Iris-setosa  8,Iris-setosa  9,Iris-setosa  10,Iris-setosa  11,Iris-versicolor  12,Iris-versicolor  13,Iris-versicolor  14,Iris-versicolor  15,Iris-versicolor  16,Iris-versicolor  17,Iris-versicolor  18,Iris-versicolor  19,Iris-versicolor  20,Iris-versicolor  21,Iris-versicolor  22,Iris-versicolor  23,Iris-versicolor  24,Iris-virginica  25,Iris-virginica  26,Iris-virginica  27,Iris-virginica  28,Iris-virginica  29,Iris-versicolor  30,Iris-virginica |

***Câu 3*** (3 *điểm*): Cho tập dữ liệu [Countries.csv](https://drive.google.com/file/d/1TMMgszKfn3hBV_PFGXx8FSR6lnw1h1KN/view?usp=sharing). Hãy viết chương trình phân cụm bằng thuật toán *k*-means

1. *(1 điểm) Xây dựng hàm đo khoảng cách sử dụng độ đo* Manhattan

|  |
| --- |
| **# Trả lời:**Minh hoạ tính khoảng cách:  **Khoảng cách Manhattan :**  Công thức:  Ảnh có chứa văn bản, Phông chữ, đồng hồ, Đồ họa  Nội dung do AI tạo ra có thể không chính xác.  Trong đó   * + x,y: Hai điểm dữ liệu   + n: Số đặc trưng   Mô tả: Đo tổng chênh lệch tuyệt đối giữa các chiều. Thường hiệu quả khi dữ liệu có phân phối không chuẩn  **# Trả lời:** Dán code hàm tính khoảng cách:  def manhattan\_distance(x, y):  """Tính khoảng cách Manhattan (slide Lesson 11, trang 19)"""  return sum(abs(a - b) for a, b in zip(x, y)) |
|  |

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

|  |
| --- |
| **# Trả lời:** Dán code về hàm  def kmeans(X, k, max\_iters=100):      """Thuật toán K-means với khoảng cách Manhattan"""      # Chuyển đổi DataFrame thành numpy array nếu cần      if isinstance(X, pd.DataFrame):          # Chỉ lấy các cột số (bỏ qua cột tên nếu có)          numeric\_columns = X.select\_dtypes(include=[np.number]).columns          X\_array = X[numeric\_columns].values      else:          X\_array = X        n\_samples, n\_features = X\_array.shape        # Khởi tạo ngẫu nhiên các centroid      idx = np.random.choice(n\_samples, k, replace=False)      centroids = X\_array[idx]        for \_ in range(max\_iters):          # Gán nhãn cụm          labels = np.array([np.argmin([manhattan\_distance(x, c) for c in centroids]) for x in X\_array])            # Cập nhật centroid          new\_centroids = np.array([X\_array[labels == i].mean(axis=0) for i in range(k)])            # Kiểm tra hội tụ          if np.allclose(centroids, new\_centroids, atol=1e-6):              break          centroids = new\_centroids        return labels, centroids |

1. *(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  **Code:**  def elbow\_method(X, max\_k=10):      """Khảo sát số cụm k bằng phương pháp Elbow"""      # Chuyển đổi DataFrame thành numpy array nếu cần      if isinstance(X, pd.DataFrame):          # Chỉ lấy các cột số (bỏ qua cột tên nếu có)          numeric\_columns = X.select\_dtypes(include=[np.number]).columns          X\_array = X[numeric\_columns].values          print(f"Sử dụng các cột: {list(numeric\_columns)}")      else:          X\_array = X        print(f"Kích thước dữ liệu: {X\_array.shape}")        inertias = []      for k in range(1, max\_k + 1):          labels, centroids = kmeans(X\_array, k)          inertia = 0          for i in range(len(X\_array)):              inertia += manhattan\_distance(X\_array[i], centroids[labels[i]])          inertias.append(inertia)          print(f"k={k}, inertia={inertia:.2f}")        # Vẽ biểu đồ Elbow      plt.figure(figsize=(10, 6))      plt.plot(range(1, max\_k + 1), inertias, 'bo-')      plt.title('Elbow Method for Optimal k')      plt.xlabel('Number of Clusters (k)')      plt.ylabel('Inertia')      plt.grid(True)      plt.show()        return inertias  def visualize\_clusters(X, k=3):      """Trực quan hóa kết quả phân cụm"""      if isinstance(X, pd.DataFrame):          numeric\_columns = X.select\_dtypes(include=[np.number]).columns          X\_array = X[numeric\_columns].values      else:          X\_array = X        labels, centroids = kmeans(X\_array, k)        plt.figure(figsize=(12, 8))      colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown', 'pink', 'gray']        for i in range(k):          cluster\_points = X\_array[labels == i]          plt.scatter(cluster\_points[:, 0], cluster\_points[:, 1],                     c=colors[i % len(colors)], label=f'Cluster {i+1}', alpha=0.6)        # Vẽ centroids      plt.scatter(centroids[:, 0], centroids[:, 1],                 c='black', marker='x', s=200, linewidths=3, label='Centroids')        plt.title(f'K-means Clustering (k={k})')      plt.xlabel('Longitude')      plt.ylabel('Latitude')      plt.legend()      plt.grid(True)      plt.show()        return labels, centroids  # Thực thi trên tập dữ liệu Countries  print("Loading Countries.csv...")  X = pd.read\_csv('Countries.csv')  print(f"Dữ liệu gốc: {X.shape}")  print(f"Các cột: {list(X.columns)}")  print(f"Kiểu dữ liệu các cột:")  print(X.dtypes)  print("\nMột vài dòng đầu tiên:")  print(X.head())  print("\nThực hiện Elbow Method...")  inertias = elbow\_method(X, max\_k=8)  **Cách lựa chọn k phù hợp**  **Dấu hiệu nhận biết k tối ưu:**  - Điểm Elbow: Vị trí mà inertia giảm mạnh rồi chuyển sang giảm chậm  - Tỷ lệ giảm: Khi tốc độ giảm inertia chậm lại đáng kể  - Hình dạng đồ thị: Tìm điểm uốn cong rõ rệt nhất  **### Các yếu tố cần xem xét:**  **1. \*\*Độ giảm inertia\*\*:**     - k=1→2: Giảm nhiều     - k=2→3: Giảm vừa phải     - k=3→4: Giảm ít → k=3 có thể là lựa chọn tốt  **2. \*\*Ý nghĩa thực tế\*\*:**     - Số cụm phải có ý nghĩa trong bối cảnh bài toán  **3. \*\*Độ phức tạp mô hình\*\*:**     - Quá nhiều cụm → overfitting     - Quá ít cụm → underfitting  **# Trả lời:** Dán kết quả thi với k( lưu ý có giải thích và bình luận):    Loading Countries.csv...  Dữ liệu gốc: (200, 3)  Các cột: ['name', 'Longitude', 'Latitude']  Kiểu dữ liệu các cột:  name object  Longitude float64  Latitude float64  dtype: object  Một vài dòng đầu tiên:  name Longitude Latitude  0 China 103.819074 36.561765  1 CÃ´te d'Ivoire -5.569216 7.628426  2 Cameroon 12.739642 5.691098  3 Dem. Rep. Congo 23.643961 -2.877463  4 Congo 15.219658 -0.837875  Thực hiện Elbow Method...  Sử dụng các cột: ['Longitude', 'Latitude']  Kích thước dữ liệu: (200, 2)  k=1, inertia=14935.06  k=2, inertia=11384.34  k=3, inertia=8133.87  k=4, inertia=7262.69  k=5, inertia=6369.20  k=6, inertia=5629.24  k=7, inertia=4820.83  k=8, inertia=4546.99  **1. Điểm Elbow rõ ràng tại k=3→4:**   * **k=1→2**: Giảm 3550.72 (rất mạnh) * **k=2→3**: Giảm 3250.47 (vẫn mạnh) * **k=3→4**: Giảm 1183.42 (đáng kể) * **k=4→5**: Giảm 742.02 (chậm lại) * **k=5→6**: Giảm 474.93 (rất chậm)   **2. Kết luận:**  **K = 3 là lựa chọn tối ưu** cho dataset Countries vì:  -  **Điểm elbow rõ ràng**: Tại k=3→4, tốc độ giảm inertia chậm lại đáng kể - **Cân bằng**: Không quá đơn giản (k=2) cũng không quá phức tạp (k>4)  **3. Tại sao k=3 tốt:**   * **200 quốc gia** được phân thành **3 cụm địa lý** hợp lý * **Inertia giảm từ 14935 → 8134** (giảm 45%) với chỉ 3 cụm * **Từ k=4 trở đi** việc giảm inertia không đáng kể so với độ phức tạp tăng thêm |

Đà Nẵng, ngày 20 tháng 05 năm 2025

|  |  |
| --- | --- |
| **GIẢNG VIÊN BIÊN SOẠN ĐỀ THI** | **TRƯỞNG BỘ MÔN** |
|  | (đã duyệt) |