|  |
| --- |
| 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: **00002** Thời gian làm bài: 70 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên:**……………………….**Lớp**:……………………………..**MSSV**:……………………………..

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*): Cho bài toán như sau: 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 ngụy 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 **DFS** 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 hoạ, 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 toạ độ (0,0)). 2 ô chỉ có thể thông nhau nếu chúng có chung cạnh.

Dữ liệu nhập vào từ tập tin văn bản “[bfs\_dfs.csv](https://drive.google.com/file/d/1kb9IWLQpH-KkkDiD6EhjgXr7gcxX8xei/view?usp=sharing)” gồm:

- Dòng đầu chứa 3 số nguyên dương n < 30, 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 “bfs\_dfs\_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: bfs\_dfs.csv  **4 2 1**  **0 1 1 0**  **1 0 0 1**  **1 1 1 1**  **0 1 1 0** | Tệp bfs\_dfs\_out.csv(lưu ý đây chỉ là một phương án)  **3**  **2 1**  **2 2**  **3 2** |

1. (*1 điểm*)Viết hàm **DFS** để giải quyết bài toán trên

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  def dfs(n, start\_row, start\_col, grid):      directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]      stack = [(start\_row, start\_col, [(start\_row, start\_col)])]      visited = set()      while stack:          x, y, path = stack.pop()          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):                  stack.append((nx, ny, path + [(nx, ny)]))      return -1 # No escape found |

1. (*1 điểm*)Viết chương trình hoàn thiện cho bài toán trên

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  import csv  from io import StringIO  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 dfs(n, start\_row, start\_col, grid):      directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]      stack = [(start\_row, start\_col, [(start\_row, start\_col)])]      visited = set()      while stack:          x, y, path = stack.pop()          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):                  stack.append((nx, ny, path + [(nx, ny)]))      return -1 # No escape found  def main():      input\_file = "bfs\_dfs.csv";      n, start\_row, start\_col, grid = read\_input(input\_file)      path = dfs(n, start\_row, start\_col, grid)      output\_file = 'bfs\_dfs\_out.csv'      write\_output(output\_file, path)      print(f"Output file has been written successfully to {output\_file}!")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1. (*1 điểm*)Kết quả thực thi trên tệp “bfs\_dfs\_out.csv

|  |
| --- |
| **# Trả lời:** Dán kết quả vào bên dưới và kèm lời giải thích    - Đầu vào:  Kích thước ma trận: 4x4.  Phòng trung tâm tại tọa độ (2, 1).  Ma trận biểu diễn hệ thống đường hầm.  - Thuật toán DFS:  Bắt đầu từ phòng trung tâm (2, 1).  Sử dụng ngăn xếp để duyệt các ô có thể đi đến.  - Các bước tìm kiếm: Bắt đầu từ (2, 1), tìm các ô lân cận hợp lệ: (3, 1), (2, 0), (2, 2).  Tiếp tục từ (2, 0), tìm thấy ô (3, 0) ở rìa, kết thúc.  - Kết quả:  Đường thoát ngắn nhất: (2, 1) -> (2, 0) -> (3, 0).  Đã thoát ra rìa lâu đài tại (3, 0). |

***Câu 2*** (*4 điểm*): Cho tập dữ liệu [input.csv](https://drive.google.com/file/d/11wcxZ4o8IcxInM1uAtCzK8Jlhc4L-QBU/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. *(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, "%")  **# Trả lời:** Dán kiến trúc mạng ( yêu cầu kiến trúc chứa ít nhất 1 lớp ẩn) và giải thích làm thể 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. |

1. *(2 điểm)* Hãy thực thi chương trình và cho biết nhãn của 30 mẫu dữ liệu trong [output.csv](https://drive.google.com/file/d/1VFOrij_lpUZWrDymB_UQcFq7TMZAS7ga/view?usp=sharing)

|  |
| --- |
| **# 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)     set\_data('output.csv', y\_val\_predict, 'result.csv')  **# Trả lời:** Dán kết quả nhãn ứng với 30 mẫu dữ liệu |

***Câu 3*** (3 *điểm*): Cho tập dữ liệu [Countries.csv](https://drive.google.com/file/d/1qYh4cdwoYPFHrhUetJ0uSoh-45PoWv7f/view?usp=sharing). Hãy viết chương trình phân cụm bằng thuật toán *k*-means

1. *(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\_  **# Trả lời:** Dán kết quả thực thi với **k = 4** ( chỉ cần đưa ra số tâm, toạ độ của các tâm) |

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  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 khủy 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 đổ Elbow: Biểu đồ hiển trị 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 tang lên điều này chỉ ra số cụm tối ưu |  |  | | --- | |  |   **# Trả lời:** Dán kết quả thi với k vừa khảo sát được ( chỉ cần đưa ra số tâm, toạ độ của các tâm) |

1. *(1 điểm)* Xây dựng chương trình hoàn thiện và thực thi với dữ liệu đã cho

|  |
| --- |
| **# Trả lời**: Dán code hoàn thiện  import pandas as pd  import numpy as np  from sklearn.cluster import KMeans  import matplotlib.pyplot as plt  from sklearn.preprocessing import StandardScaler  file\_path = 'Countries.csv'  data = pd.read\_csv(file\_path)  print(data.head())  # a. Xây dựng hàm chứa thuật toán K-Means để phân cụm  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\_  # b. Xây dựng hàm để khảo sát việc lựa chọn k  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()  # c. Xây dựng chương trình hoàn thiện và thực thi với dữ liệu đã cho  def main():      # Đọc dữ liệu      data = pd.read\_csv(file\_path)      # Lựa chọn cột dữ liệu để phân cụm      features = data[['Longitude', 'Latitude']]      # Khảo sát số lượng cụm k      find\_optimal\_k(features)      # Số cụm k tối ưu dựa trên phương pháp Elbow      optimal\_k = 4  # Giả sử chúng ta chọn 4 cụm từ biểu đồ Elbow      # Thực hiện phân cụm với k tối ưu      labels, cluster\_centers = k\_means\_clustering(features, optimal\_k)      print(cluster\_centers)      # Thêm nhãn cụm vào dữ liệu ban đầu      data['Cluster'] = labels      # Xuất dữ liệu đã phân cụm      output\_path = 'Countries\_with\_clusters.csv'      data.to\_csv(output\_path, index=False)      print(f'Phân cụm hoàn tất. Kết quả đã được lưu vào {output\_path}')  # Thực thi chương trình  main()  s |

Đà Nẵng, ngày 14 tháng 05 năm 2023

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