|  |
| --- |
| 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:** Đỗ Nhật Hoàng .**Lớp**: 21TCLC\_DT1 **MSSV: 102210057**

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, matrix):      stack = [(start\_row, start\_col, [(start\_row, start\_col)])]      visited = [[False]\*n for \_ in range(n)]      visited[start\_row][start\_col] = True      while stack:          current\_row, current\_col, path = stack.pop()          if current\_row == 0 or current\_row == n-1 or current\_col == 0 or current\_col == n-1:              return [(len(path))] + path          directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]            for dr, dc in directions:              new\_row, new\_col = current\_row + dr, current\_col + dc              if is\_valid(new\_row, new\_col, n, visited, matrix):                  visited[new\_row][new\_col] = True                  stack.append((new\_row, new\_col, path + [(new\_row, new\_col)]))        return [[-1]] |

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

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  import csv  # Đọc dữ liệu từ file CSV  def read\_data(file\_name):      with open(file\_name, 'r') as file:          reader = csv.reader(file)          data = list(reader)        n = int(data[0][0])      D = int(data[0][1])      C = int(data[0][2])        matrix = []      for row in data[1:]:          matrix.append(list(map(int, row)))        return n, D, C, matrix  # Ghi kết quả ra file CSV  def write\_data(file\_name, result):      with open(file\_name, 'w', newline='') as file:          writer = csv.writer(file)          for row in result:              if isinstance(row, int):  # Kiểm tra nếu row là số nguyên                  writer.writerow([row])              else:                  writer.writerow(row)  # Kiểm tra tính hợp lệ của ô (row, col)  def is\_valid(row, col, n, visited, matrix):      return 0 <= row < n and 0 <= col < n and not visited[row][col] and matrix[row][col] == 1  # Sử dụng thuật toán DFS để tìm đường thoát  def dfs(n, start\_row, start\_col, matrix):      stack = [(start\_row, start\_col, [(start\_row, start\_col)])]      visited = [[False]\*n for \_ in range(n)]      visited[start\_row][start\_col] = True      while stack:          current\_row, current\_col, path = stack.pop()          if current\_row == 0 or current\_row == n-1 or current\_col == 0 or current\_col == n-1:              return [(len(path))] + path          directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]            for dr, dc in directions:              new\_row, new\_col = current\_row + dr, current\_col + dc              if is\_valid(new\_row, new\_col, n, visited, matrix):                  visited[new\_row][new\_col] = True                  stack.append((new\_row, new\_col, path + [(new\_row, new\_col)]))        return [[-1]]  if \_\_name\_\_ == "\_\_main\_\_":      n, D, C, matrix = read\_data('bfs\_dfs.csv')      result = dfs(n, D, C, matrix)      write\_data('bfs\_dfs\_out.csv', result) |

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

|  |
| --- |
| 3  2,1  2,2  2,3  - 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? |

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  5.4,3.7,1.5,0.2,Iris-setosa  4.8,3.4,1.6,0.2,Iris-setosa  4.8,3.0,1.4,0.1,Iris-setosa  4.3,3.0,1.1,0.1,Iris-setosa  5.8,4.0,1.2,0.2,Iris-setosa  5.7,4.4,1.5,0.4,Iris-setosa  5.4,3.9,1.3,0.4,Iris-setosa  5.1,3.5,1.4,0.3,Iris-setosa  5.7,3.8,1.7,0.3,Iris-setosa  5.1,3.8,1.5,0.3,Iris-setosa  7.0,3.2,4.7,1.4,Iris-versicolor  6.4,3.2,4.5,1.5,Iris-versicolor  6.9,3.1,4.9,1.5,Iris-versicolor  5.5,2.3,4.0,1.3,Iris-versicolor  6.5,2.8,4.6,1.5,Iris-versicolor  5.7,2.8,4.5,1.3,Iris-versicolor  6.3,3.3,4.7,1.6,Iris-versicolor  4.9,2.4,3.3,1.0,Iris-versicolor  6.6,2.9,4.6,1.3,Iris-versicolor  5.2,2.7,3.9,1.4,Iris-versicolor  5.0,2.0,3.5,1.0,Iris-versicolor  5.9,3.0,4.2,1.5,Iris-versicolor  6.0,2.2,4.0,1.0,Iris-versicolor  5.8,2.7,5.1,1.9,Iris-virginica  7.1,3.0,5.9,2.1,Iris-virginica  6.3,2.9,5.6,1.8,Iris-virginica  6.5,3.0,5.8,2.2,Iris-virginica  7.6,3.0,6.6,2.1,Iris-virginica  4.9,2.5,4.5,1.7,Iris-virginica  7.3,2.9,6.3,1.8,Iris-virginica |

***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  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  data = pd.read\_csv('Countries.csv')  X = data.iloc[:, 1:3]  X  plt.scatter(X['Longitude'], X['Latitude'])  def initialize\_K\_centroids(X, K):      m,n = X.shape      k\_rand = np.ones((K, n))      k\_rand = X[np.random.choice(range(len(X)), K, replace=False),:]      return k\_rand  def find\_closest\_centroids(X, centroids):      m = len(X)      c = np.zeros(m)      for i in range(m):          # compute distances          distances = np.linalg.norm(X[i] - centroids, axis=1)          c[i] = np.argmin(distances)      return c  def compute\_means(X, idx, K):      m, n = X.shape      centroids = np.zeros((K, n))      for k in range(K):          points\_belong\_k = X[np.where(idx == k)]          centroids[k] = np.mean(points\_belong\_k, axis=0,)      return centroids  def compute\_inertia(X, idx, centroids):      total\_distance = 0      for i in range(len(X)):          centroid = centroids[int(idx[i])]          total\_distance += np.sum((X[i] - centroid) \*\* 2)      return total\_distance  def find\_k\_means(X, K, max\_iters=100):      \_, n = X.shape      centroids = initialize\_K\_centroids(X, K)      inertia = 0      for i in range(max\_iters):          idx = find\_closest\_centroids(X, centroids)          centroids = compute\_means(X, idx, K)          inertia = compute\_inertia(X, idx, centroids)      return centroids, idx, inertia  X = np.array(X)  K = 4  centroids, idx, inertial= find\_k\_means(X, K,max\_iters=10 )  print(inertial)  for i in range(4):      print("Toạ độ tâm" , str(i+1) , ":" , centroids[i])  **# Trả lời:** Dán kết quả thực thi với **k = 4** ( chỉ cần đưa ra số tâm, toạ độ của các tâm)  Trọng tâm 1 : [-83.7257277 5.08220159]  Trọng tâm 2 : [ 5.29275154 38.64682883]  Trọng tâm 3 : [43.11749148 10.97292168]  Trọng tâm 4 : [130.31436822 9.59308518] |

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  l = []  for i in range(1,10):      X = np.array(X)      K= i      \_, \_, inertia = find\_k\_means(X, K, max\_iters = 10)      l.append(inertia)  print(l)  plt.plot(range(1,10), l)  [1184443.0641129618, 592797.3834090307, 264682.9232003739, 206067.69611027115, 176870.74999492048, 123424.75235323446, 146350.29256168765, 128680.1052121379, 77216.75531085633]  Lựa chọn k = 5 , vì tổng bình phương khoảng cách của các điểm đến các điểm tâm nó đang nhỏ dẫn, và khi vẽ đồ thị ta thấy được k=5 là đoạn thoải, không có gấp khúc.  **# Trả lời:** Dán kết quả thi với k vừa khảo sát được ( chỉ cần đưa ra số tâm, toạ độ của các tâm)  Số tâm = 5  Toạ độ tâm 1 : [85.55511097 24.1663486 ]  Toạ độ tâm 2 : [26.93223587 -2.92891051]  Toạ độ tâm 3 : [-83.7257277 5.08220159]  Toạ độ tâm 4 : [146.91952438 2.39542366]  Toạ độ tâm 5 : [12.10694628 41.2877857 ] |

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

|  |
| --- |
| **# Trả lời**: Dán code hoàn thiện  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  data = pd.read\_csv('Countries.csv')  X = data.iloc[:, 1:3]  plt.scatter(X['Longitude'], X['Latitude'])  def initialize\_K\_centroids(X, K):      m,n = X.shape      k\_rand = np.ones((K, n))      k\_rand = X[np.random.choice(range(len(X)), K, replace=False),:]      return k\_rand  def find\_closest\_centroids(X, centroids):      m = len(X)      c = np.zeros(m)      for i in range(m):          # compute distances          distances = np.linalg.norm(X[i] - centroids, axis=1)          c[i] = np.argmin(distances)      return c  def compute\_means(X, idx, K):      m, n = X.shape      centroids = np.zeros((K, n))      for k in range(K):          points\_belong\_k = X[np.where(idx == k)]          centroids[k] = np.mean(points\_belong\_k, axis=0,)      return centroids  def compute\_inertia(X, idx, centroids):      total\_distance = 0      for i in range(len(X)):          centroid = centroids[int(idx[i])]          total\_distance += np.sum((X[i] - centroid) \*\* 2)      return total\_distance  def find\_k\_means(X, K, max\_iters=10):      \_, n = X.shape      centroids = initialize\_K\_centroids(X, K)      inertia = 0      for i in range(max\_iters):          idx = find\_closest\_centroids(X, centroids)          centroids = compute\_means(X, idx, K)          inertia = compute\_inertia(X, idx, centroids)      return centroids, idx, inertia  # how to choose K  l = []  for i in range(1,10):      X = np.array(X)      K= i      \_, \_, inertia = find\_k\_means(X, K, max\_iters = 10)      l.append(inertia)  print(l)  plt.plot(range(1,10), l)  X = np.array(X)  K = 5  centroids, idx, inertial= find\_k\_means(X, K,max\_iters=10 )  print(inertial)  for i in range(5):      print("Toạ độ tâm" , str(i+1) , ":" , centroids[i]) |

Đà Nẵng, ngày 14 tháng 05 năm 2023

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