|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA  **KHOA Công Nghệ Thông Tin**  BỘ MÔN: Công Nghệ Phần Mềm |

**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: **00004** Thời gian làm bài: 70 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên: Võ Văn Bản** …**Lớp**:…19TCLC\_DT4…**MSSV**:……102190152...

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*** (*4 điểm*): Cho tập dữ liệu [input.csv](https://drive.google.com/file/d/1zgHg7FMS654ImWQyihErR1Qu1Kn1KV6E/view?usp=sharing) với 80 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. *(3 điểm)* Hãy viết chương trình phân loại hoa sử dụng Logistic Regression. Nêu rõ mô hình thức phân loại trong chương trình như thế nào (Ví dụ: có bao nhiêu tế bào nơ-ron, mỗi nơ-ron phụ trách công việc gì, làm sao để phân loại,…)?

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt    # Importing Sklearn module and classes  from sklearn.linear\_model import LogisticRegression  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn import datasets  from sklearn.model\_selection import train\_test\_split  df = pd.read\_csv('input.csv')  test = pd.read\_csv('output.csv')  test  X\_train = df.drop('Labeling', axis=1)  X\_test = test.drop('Labeling', axis=1)  X\_train  Y\_train = df['Labeling']  Y\_test = test['Labeling']  Y\_test  sc = StandardScaler()  sc.fit(X\_train)  X\_train\_std = sc.transform(X\_train)  X\_test\_std = sc.transform(X\_test)  lr = LogisticRegression(C=100.0, random\_state=1, solver='lbfgs', multi\_class='ovr')    # Fit the model  #  lr.fit(X\_train\_std, Y\_train)  lr  # Create the predictions  #  Y\_predict = lr.predict(X\_test\_std)  Y\_predict  # Use metrics.accuracy\_score to measure the score  #print("LogisticRegression Accuracy %.3f" %metrics.accuracy\_score(Y\_test, Y\_predict))  **# Trả lời:** Mô tả mô hình phân loại bằng hình ảnh hoặc bằng lời.  Cách thức hoạt động của mô hình Logistic Regression with Softmax như sau:  - Tính tổng có trọng số của đặc trưng đầu vào với ma trận trọng số w.  - Áp dụng hàm Softmax để tính toán xác suất dự đoán cho mỗi lớp.  - Dựa vào xác suất dự đoán, lớp với xác suất cao nhất được chọn là lớp dự đoán cho mẫu dữ liệu  đó.  Mô hình Logistic Regression with Softmax sử dụng hàm Softmax để tính toán các xác suất cho các lớp  khác nhau. Hàm Softmax được sử dụng để chuyển đổi giá trị đầu ra của mô hình thành các xác suất  tổng quát mà tổng các xác suất đó bằng 1. Điều này cho phép chúng ta dễ dàng xác định lớp có xác suất  dự đoán cao nhất. |

1. *(1 điểm)* Hãy thực thi chương trình và cho biết nhãn của 10 mẫu dữ liệu trong [output4.csv](https://drive.google.com/file/d/1ubiVXlDfQ8DIJt8TqkmRY7_VEC1C8Usq/view?usp=sharing)

|  |
| --- |
| **# Trả lời:** Dán code thực thi thành công  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt    # Importing Sklearn module and classes  from sklearn.linear\_model import LogisticRegression  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn import datasets  from sklearn.model\_selection import train\_test\_split  df = pd.read\_csv('input\_04.csv')  test = pd.read\_csv('output\_04.csv')  test  X\_train = df.drop('Labeling', axis=1)  X\_test = test.drop('Labeling', axis=1)  X\_train  Y\_train = df['Labeling']  Y\_test = test['Labeling']  Y\_test  sc = StandardScaler()  sc.fit(X\_train)  X\_train\_std = sc.transform(X\_train)  X\_test\_std = sc.transform(X\_test)  lr = LogisticRegression(C=100.0, random\_state=1, solver='lbfgs', multi\_class='ovr')    # Fit the model  #  lr.fit(X\_train\_std, Y\_train)  lr  # Create the predictions  #  Y\_predict = lr.predict(X\_test\_std)  Y\_predict  # Use metrics.accuracy\_score to measure the score  #print("LogisticRegression Accuracy %.3f" %metrics.accuracy\_score(Y\_test, Y\_predict))  **# Trả lời:** Dán kết quả nhãn ứng với 10 mẫu dữ liệu |

***Câu 2*** (2 *điểm*): Cho không gian Oxyz với 6 điểm có tọa độ tương ứng (0,2,2), (2,2,0) (3,3,0),(3,2,2),(2,3,2) và (2,3,1).

1. *(1 điểm)* Viết hàm thực thi thuật toán *k*-means

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  import numpy as np  X = np.array([[0,2,2], [2,2,0],[3,3,0],[3,2,2],[2,3,2],[2,3,1]])  n\_cluster = 3  def init\_centers(X, n\_cluster):    # chọn ngẫu nhiên k tâm cụm từ X    return X[np.random.choice(X.shape[0], n\_cluster, replace=False)]  def group\_data(X, centers):      # gom cụm dữ liệu      n = X.shape[0]      y = np.zeros(n)      # tính khoảng cách từ điểm bất kỳ đến tất cả các tâm cụm      # xác định điểm đó thuộc cụp = min(dist(X[i] - centers))      for i in range(n):          dist = np.linalg.norm(X[i] - centers, axis=1)          y[i] = np.argmin(dist)      return y  def update\_centers(X, labels, n\_cluster):    centers = np.zeros((n\_cluster, X.shape[1]))      for k in range(n\_cluster):      # chọn tất cả các điểm thuộc cụm thứ k      Xk = X[labels == k, :]      # chia trung bình công      centers[k,:] = np.mean(Xk, axis = 0)    return centers    def kmeans(X,k):      centers = init\_centers(X, n\_cluster)      y = []      while True:          y\_old = y          y = group\_data(X, centers)          # kiểm tra đã hội tụ hay chưa          if np.array\_equal(y\_old, y):              break          # cập nhật tâm cụm          centerse = update\_centers(X, y, n\_cluster)      return (centers, y)  (centers, y) = kmeans(X,n\_cluster)  for k in range(n\_cluster):      # chọn tất cả các điểm thuộc cụm thứ k    Xk = X[y == k, :]    print('Tâm cụm thứ ',k + 1,': ', centers[k])    print('Cac diem thuoc cum thu ',k + 1)    print(str(Xk)) |

1. *(1 điểm)* Nếu sử dụng thuật toán *k*-means với k = 3 thì kết quả phân nhóm sẽ như thế nào? (các điểm thuộc mỗi nhóm, trọng tâm của mỗi nhóm).

|  |
| --- |
| **# Trả lời**: viết câu trả lời vào bên dưới  import numpy as np  X = np.array([[0,2,2], [2,2,0],[3,3,0],[3,2,2],[2,3,2],[2,3,1]])  n\_cluster = 3  def init\_centers(X, n\_cluster):    # chọn ngẫu nhiên k tâm cụm từ X    return X[np.random.choice(X.shape[0], n\_cluster, replace=False)]  def group\_data(X, centers):      # gom cụm dữ liệu      n = X.shape[0]      y = np.zeros(n)      # tính khoảng cách từ điểm bất kỳ đến tất cả các tâm cụm      # xác định điểm đó thuộc cụp = min(dist(X[i] - centers))      for i in range(n):          dist = np.linalg.norm(X[i] - centers, axis=1)          y[i] = np.argmin(dist)      return y  def update\_centers(X, labels, n\_cluster):    centers = np.zeros((n\_cluster, X.shape[1]))      for k in range(n\_cluster):      # chọn tất cả các điểm thuộc cụm thứ k      Xk = X[labels == k, :]      # chia trung bình công      centers[k,:] = np.mean(Xk, axis = 0)    return centers    def kmeans(X,k):      centers = init\_centers(X, n\_cluster)      y = []      while True:          y\_old = y          y = group\_data(X, centers)          # kiểm tra đã hội tụ hay chưa          if np.array\_equal(y\_old, y):              break          # cập nhật tâm cụm          centerse = update\_centers(X, y, n\_cluster)      return (centers, y)  (centers, y) = kmeans(X,n\_cluster)  for k in range(n\_cluster):      # chọn tất cả các điểm thuộc cụm thứ k    Xk = X[y == k, :]    print('Tâm cụm thứ ',k + 1,': ', centers[k])    print('Cac diem thuoc cum thu ',k + 1)    print(str(Xk)) |

***Câu 3*** (*4 điểm*): Cho cây G = (V,E) như hình vẽ với V là tập đỉnh và E là tập cạnh, trọng số tại đỉnh là hàm ước lượng khoảng cách từ đỉnh đó đến trạng thái đích( giá trị các nhỏ thì càng gần trạng thái đích), trọng số trên cạnh thể hiện chi phí phải trả khi đi qua cạnh.

Shape

Description automatically generated

1. *(2 điểm)* Hãy viết đoạn code biểu diễn đồ thị trên bằng cách khởi tạo tập đỉnh V, tập cạnh E, trọng số trên đỉnh và trọng số trên cạnh.

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  V = [('S', 12), ('A', 7), ('B', 8), ('C', 9), ('D', 6), ('E', 5), ('F', 4), ('G', 10), ('H', 10), ('K', 3), ('M', 9), ('N', 10), ('I', 6), ('J', 0), ('L', 0), ('Z', 8)]  E = [('S', 'A', 5), ('A', 'S', 5), ('S', 'B', 6), ('B', 'S', 6), ('S', 'C', 5), ('C', 'S', 5),('A', 'D', 6), ('D', 'A', 6), ('A', 'E', 7), ('E', 'A', 7), ('B', 'F', 3), ('F', 'B', 3), ('B', 'G', 4), ('G', 'B', 4), ('C', 'H', 6), ('H', 'C', 6), ('C', 'K', 4), ('K', 'C', 4), ('D', 'M', 5), ('M', 'D', 5), ('D', 'N', 8), ('N','D',8), ('E', 'I', 8), ('I', 'E', 8), ('F', 'J', 4), ('J', 'F', 4), ('F', 'L', 4), ('L', 'F', 4), ('K','Z', 2),('Z', 'K', 2)] |

1. *(2 điểm)* Hãy viết chương trình sử dụng thuật toán **A\*** để tìm đường đi từ đỉnh “S” đến các đỉnh có trọng số trên đỉnh bằng 0. Trong chương trình, hãy in ra thứ tự đỉnh khám phá trong quá trình tìm kiếm.

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới  class Node:      def \_\_init\_\_(self, name, parent=None, g = 0, h=0):          self.name = name          self.g = g          self.parent = parent          self.h = h      def display(self):          print(self.name, self.g, self.h)      def \_\_lt\_\_(self, other):          if other is None:              return False          return self.h + self.g < other.h + other.g      def checkInArray(self, Close):          for x in Close:              if self.equal(x):                  return True          return False      def \_\_eq\_\_(self, other):          if other is None:              return False          return self.name == other.name      def equal(self, other):          if self.name == other.name:              return True          return False      def checkInPriority(self, c):          if self is None:              return False          return self in c.queue      def getPath(self):          print(self.name)          if self.parent is not None:              self.parent.getPath()          else:              return  from collections import defaultdict  from queue import PriorityQueue  import sys  data = defaultdict(list)  data['S'] = ['A', 5, 'B', 6, 'C', 5, 12]  data['A'] = ['D', 6, 'E', 7, 7]  data['B'] = ['F', 3, 'G', 4, 8]  data['C'] = ['H', 6, 'K', 4, 9]  data['D'] = ['M', 5, 'N', 8, 6]  data['E'] = ['I', 8, 5]  data['F'] = ['J', 4, 'L', 4, 4]  data['G'] = [10]  data['H'] = [10]  data['K'] = ['Z', 2, 3]  data['M'] = [9]  data['N'] = [10]  data['I'] = [6]  data['J'] = [0]  data['L'] = [0]  data['Z'] = [4]  def AStar(S=Node('S'), G=Node('J')):      Open = PriorityQueue()      Closed = []      S.h = data[S.name][-1]      Open.put(S)      while True:          if Open.empty():              print("Tim kiem that bai")              return          O = Open.get()          Closed.append(O)          for i in Closed:              sys.stdout.write(i.name)          print()          # print('duyet: ', O.name, O.h)          if O.equal(G):              print("Tim kiem thanh cong")              O.getPath()              print('Cost: ', O.g)              return          i = 0          while i < len(data[O.name]) - 1:              name = data[O.name][i]              g = O.g + data[O.name][i + 1]              h = data[name][-1]              tmp = Node(name=name, h=h, g=g)              tmp.parent = O              ok1 = tmp.checkInPriority(Open)              ok2 = tmp.checkInArray(Closed)              if not ok1 and not ok2:                  Open.put(tmp)              i += 2  AStar(Node('S'), Node('L'))  **# Trả lời:** Dán kết quả thực thi vào bên dưới |

Đà Nẵng, ngày 5 tháng 06 năm 2022

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