**BÀI THỰC HÀNH SỐ 2**

1. **Mục đích:**

Để sử dụng và hiện thực các hàm cơ bản ứng dụng trong thống kê và các classifier đơn giản với 1 đặc trưng và 2 lớp. Histogram được sử dụng để xác định biệt số (discriminant) sao cho tối thiểu misclassification.

1. **Nội dung:**

**Bài tập 1***:* Xây dựng một phân lớp dựa trên histogram, với tập dữ liệu sau:

* + in\_time = [(0, 27), (1, 25), (2, 16), (3, 19), (4, 26), (5, 20), (6, 19), (7, 17), (8, 10), (9, 5), (10, 4), (11, 4), (12,2)]
  + cls\_late = [(5,3), (6, 5), (7, 8), (8, 15), (9, 17), (10, 18), (11, 19), (12,16), (13, 9), (14, 8), (15, 8)]

Kiểm tra kết quả nếu rời nhà lúc 6:34, 6:35, 6:36, 6:37, 6:38.

*Bước 1*: Import thư viện cần sử dụng:

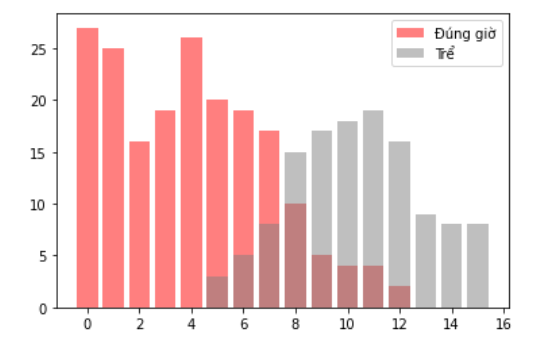
|  |
| --- |
| import **numpy** as np  import **matplotlib**.pyplot as plt  from **collections** import Counter |

*Bước 2:* Khai báo 2 danh sách các điểm dữ liệu in\_time và cls\_late theo yêu cầu đề bài:

|  |
| --- |
| **in\_time**=[(0,27),(1,25),(2,16),(3,19),(4,26),(5,20),(6,19),(7,17),  (8,10),(9,5),(10,4),(11,4),(12,2)]  **cls\_late** = [(5,3), (6, 5), (7, 8), (8, 15), (9, 17), (10, 18),  (11, 19), (12,16), (13, 9),(14, 8), (15, 8)] |

*Bước 3:* Sử dụng thư viện **matplotlib** biểu diễn dữ liệu bằng biểu đồ histogram nhằm trực quan hóa dữ liệu:

|  |
| --- |
| X, Y = **zip**(\*in\_time)  X2, Y2 = **zip**(\*cls\_late)  bar\_width = 0.8  plt.bar(X, Y, bar\_width, **color**="red", **alpha**=0.5, **label**="Đúng giờ")  bar\_width = 0.8  plt.bar(X2, Y2, bar\_width, **color**="gray", **alpha**=0.5, **label**="Trể")  plt.legend(loc='upper right')  plt.show() |



*Bước 4:* Sau khi quan sát biểu đồ histogram, dễ thấy ngưỡng để phân loại dữ liệu vào 2 tập “đúng giờ” và “trễ giờ” là 1 điểm.

|  |
| --- |
| def **pre\_sum**(A):  cur=0  res=np.zeros((len(A)))  for i in range(len(A)):  cur+=A[i]  res[i]=cur  return res  def **back\_sum**(A):  cur=0  n=len(A)  res=np.zeros((n))  for i in range(n):  cur+=A[n-i-1]  res[n-i-1]=cur  return res |

|  |
| --- |
| **pre\_sum**(vi\_dat) |
| Out:array([1. , 2. , 3. , 4. , 5. ,  5.86956522, 6.66123188, 7.34123188, 7.74123188, 7.96850461,  8.15032279, 8.32423584, 8.43534695, 8.43534695, 8.43534695]) |

|  |
| --- |
| **back\_sum**(vi\_dat) |
| array([8.43534695, 7.43534695, 6.43534695, 5.43534695, 4.43534695,  3.43534695, 2.56578173, 1.77411506, 1.09411506, 0.69411506,  0.46684234, 0.28502415, 0.11111111, 0. , 0. ]) |

|  |
| --- |
| vi\_dat = np.array(vi\_dat)  def **find\_thresh\_holding**(A,B):  sumA = np.sum(A)  sumB = np.sum(B)  AA = back\_sum(A)  BB = pre\_sum(B)  C = np.**zeros**((len(A)))  print(C)  for i in range(len(A)-1):  C[i] = (sumA-AA[i+1]+sumB-BB[i])  for i in range(len(A)-1):  if(C[i+1]<C[i]):  **return** [i,C[i]] |

|  |
| --- |
| thresh,\_\_ = **find\_thresh\_holding**(vi\_dat,1-vi\_dat) |

|  |
| --- |
| def **testt**(a):  if a>thresh:  **print**('Ket qua luc 6:{} la tre'.**format**(30+a))  else:  **print**('Ket qua luc 6:{} la dung gio'.**format**(30+a)) |

|  |
| --- |
| test = [4,5,6,7,8]  for i in test:  **testt**(i) |
| Ket qua luc 6:34 la dung gio  Ket qua luc 6:35 la dung gio  Ket qua luc 6:36 la dung gio  Ket qua luc 6:37 la dung gio  Ket qua luc 6:38 la tre |

**Bài 2:** Xây dựng bộ phân lớp trên tập dữ liệu ‘person\_data.txt’ cho các trường hợp (1) đặc trưng chiều cao, (2) kết hợp chiều cao và tên.

*Bước 1:* Load data:

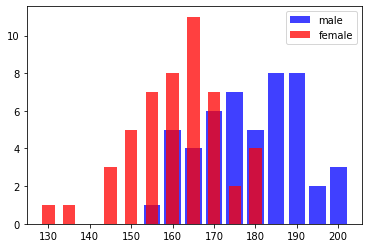
|  |
| --- |
| persons = []  with open('person\_data.txt', 'r') as fh:  for line in fh:  persons.append(line.strip().split()) |

|  |
| --- |
| firstnames = {}  heights = {}  genders = ['male','female']  for gender in genders:  firstnames[gender] = [x[0] for x in persons if x[4]==gender]  heights[gender] = [x[2] for x in persons if x[4]==gender]  heights[gender] = np.array(heights[gender],np.int)    for gender in ('female', 'male'):  print(gender + ':')  print(firstnames[gender][:10])  print(heights[gender][:10]) |
| female:  ['Stephanie', 'Cynthia', 'Katherine', 'Elizabeth', 'Carol', 'Christina', 'Beverly', 'Sharon', 'Denise', 'Rebecca']  [149 174 183 138 145 161 179 162 148 196]  male:  ['Randy', 'Jessie', 'David', 'Stephen', 'Jerry', 'Billy', 'Earl', 'Todd', 'Martin', 'Kenneth']  [184 175 187 192 204 180 184 174 177 200] |

|  |
| --- |
| class Feature:  def \_\_init\_\_(self, data, name=None, bin\_width=None):  self.name = name  self.bin\_width = bin\_width  if bin\_width:  self.min, self.max = min(data), max(data)  bins = np.arange((self.min // bin\_width) \* bin\_width, (self.max // bin\_width) \* bin\_width, bin\_width)  freq, bins = np.histogram(data, bins)  self.freq\_dict = dict(zip(bins, freq))  self.freq\_sum = sum(freq)  else:  self.freq\_dict = dict(Counter(data))  self.freq\_sum = sum(self.freq\_dict.values())    def frequency(self, value):  if self .bin\_width:  value = (value // self.bin\_width) \* self.bin\_width  if value in self.freq\_dict:  return self.freq\_dict[value]  else:  return 0 |

|  |
| --- |
| fts = {}  for gender in genders:  fts[gender] = Feature(heights[gender], name=gender, bin\_width=5)  print(gender, fts[gender].freq\_dict) |
| male {155: 1, 160: 5, 165: 4, 170: 6, 175: 7, 180: 5, 185: 8, 190: 8, 195: 2, 200: 3}  female {130: 1, 135: 1, 140: 0, 145: 3, 150: 5, 155: 7, 160: 8, 165: 11, 170: 7, 175: 2, 180: 4, 185: 0} |

|  |
| --- |
| for gender in genders:  frequencies = list(fts[gender].freq\_dict.items())  frequencies.sort(key=lambda x: x[1])  X, Y = zip(\*frequencies)  color = 'blue' if gender=='male' else 'red'  bar\_width = 4 if gender=='male' else 3  plt.bar(X,Y,bar\_width, color=color, alpha=0.75, label=gender)  plt.legend(loc='upper right')  plt.show() |



|  |
| --- |
| class NBclass:  def \_\_init\_\_(self, name, \*features):  self.features = features  self.name = name    def probability\_value\_given\_feature(self,feature\_value, feature):  if feature.freq\_sum==0:  return 0  else:  return feature.frequency(feature\_value)/feature.freq\_sum |

|  |
| --- |
| cls = {}  for gender in genders:  cls[gender] = NBclass(gender, fts[gender]) |

|  |
| --- |
| class Classifier:    def \_\_init\_\_(self, \*nbclasses):  self.nbclasses = nbclasses    def prob(self, \*d, best\_only=True):  nbclasses = self.nbclasses  probability\_list = []  for nbclass in nbclasses:  ftrs = nbclass.features  prob = 1  for i in range(len(ftrs)):  prob \*= nbclass.probability\_value\_given\_feature(d[i], ftrs[i])    probability\_list.append((prob, nbclass.name))  prob\_values = [f[0] for f in probability\_list]  prob\_sum = sum(prob\_values)  if prob\_sum == 0:  number\_classes = len(self.nbclasses)  p1 = []  for prob\_element in probability\_list:  p1.append(((1/number\_classes), prob\_element[1]))  probability\_list = p1  else:  probability\_list = [(p[0] / prob\_sum, p[1]) for p in probability\_list]  if best\_only:  return max(probability\_list)  else:  return probability\_list |

|  |
| --- |
| fts = {}  cls = {}  for gender in genders:  fts\_heights = Feature(heights[gender], name='heights', bin\_width=5)  fts\_names = Feature(firstnames[gender], name=gender)  cls[gender] = NBclass(gender, fts\_names, fts\_heights)    c = Classifier(cls["male"], cls["female"])  testnames = ['Edgar', 'Benjamin', 'Fred', 'Albert', 'Laura', 'Maria', 'Paula', 'Sharon', 'Jessie']  testSet = [('Maria', 140), ('Anthony', 200), ('Anthony', 153), ('Jessie', 188), ('Jessie', 159), ('Jessie', 160)]  for name in testSet:  print(name, c.prob(\*name, best\_only=False)) |
| ('Maria', 140) [(0.5, 'male'), (0.5, 'female')]  ('Anthony', 200) [(1.0, 'male'), (0.0, 'female')]  ('Anthony', 153) [(0.5, 'male'), (0.5, 'female')]  ('Jessie', 188) [(1.0, 'male'), (0.0, 'female')]  ('Jessie', 159) [(0.06666666666666667, 'male'), (0.9333333333333333, 'female')]  ('Jessie', 160) [(0.23809523809523817, 'male'), (0.761904761904762, 'female')] |

|  |
| --- |
| ambiguousJessie = [person for person in persons if person[0]=='Jessie']  for i in ambiguousJessie:  print(i) |
| ['Jessie', 'Morgan', '175', '67.0', 'male']  ['Jessie', 'Bell', '165', '65', 'female']  ['Jessie', 'Washington', '159', '56', 'female']  ['Jessie', 'Davis', '174', '45', 'female']  ['Jessie', 'Johnson', '165', '30.0', 'male']  ['Jessie', 'Thomas', '168', '69', 'female'] |

**Bài 3:** Tính mean và variance của các vector đặc trưng sau:

1. [1 2 4 6 9 10 20 7]
2. [0 2 4 6 8 … 100]; tất cả các số chẳn từ 0 đến 100.
3. [1 3 25 … 9801]; tất cả bình phương các số lẻ từ 1 đến 100.
4. 

|  |
| --- |
| def mean(x):    s = 0    for i in x:      s += i    return s/len(x)  def variance(x):    mu = mean(x)    s = 0    for i in x:      s += (i - mu)\*\*2    return s/len(x) |

|  |
| --- |
| import numpy as np  a = [1,2,3,4,6,9,10,20,7]  print("Trung bình của a là:")  print(mean(a))  print("Variance của a là:")  print(variance(a)) |
| Trung bình của a là:  6.888888888888889  Variance của a là:  29.876543209876544 |

|  |
| --- |
| b = list(range(101))  print("Trung bình của b là:")  print(mean(b))  print("Variance của b là:")  print(variance(b)) |
| Trung bình của b là:  50.0  Variance của b là:  850.0 |

|  |
| --- |
| c = list(range(1, 101, 2))  def power(my\_list):      return [ x\*\*2 for x in my\_list ]  c = power(c)  print("Trung bình của c là:")  print(mean(c))  print("Variance của c là:")  print(variance(c)) |
| Trung bình của c là:  3333.0  Variance của c là:  8884444.8 |

|  |
| --- |
| def mean\_2D(x):    s = 0    l = 0    for i in x:      for j in i:        s += j        l += 1    return s/l  def variance\_2D(x):    mu = mean\_2D(x)    s = 0    l = 0    for i in x:      for j in i:        s += (j - mu)\*\*2        l += 1    return s/l |

|  |
| --- |
| d = [[2,4], [3,7], [4,6], [5,5], [2,3]]  print("Trung bình của d là:")  print(mean\_2D(d))  print("Variance của d là:")  print(variance\_2D(d)) |
| Trung bình của d là:  4.1  Variance của d là:  2.4900000000000007 |

**Bài 4:** Tính covariance matrix của các vector đặc trưng sau:

X=[2 3 6 3 7 8] và Y=[5 7 9 6 7 8].

|  |
| --- |
| X = [2, 3, 6, 3, 7, 8]  Y = [5, 7, 9, 6, 7, 8]  matrix\_1 = [X, Y]  print("Gộp 2 ma trận:")  print(matrix\_1) |
| Gộp 2 ma trận:  [[2, 3, 6, 3, 7, 8], [5, 7, 9, 6, 7, 8]] |

|  |
| --- |
| print("Tính trung bình ma trận:")  avg\_matrix = [sum(matrix\_1[0])/len(matrix\_1[0]), sum(matrix\_1[1])/len(matrix\_1[1])]  print(avg\_matrix) |
| Tính trung bình ma trận:  [4.833333333333333, 7.0] |

|  |
| --- |
| print("Hiệu ma trận ban đầu với trung bình:")  matrix\_2 = []  for i in range(len(matrix\_1)):    line = []    for j in range(len(matrix\_1[i])):      line.append(matrix\_1[i][j] - avg\_matrix[i])    matrix\_2.append(line)  print(matrix\_2) |
| Hiệu ma trận ban đầu với trung bình:  [  [-2.833333333333333, -1.833333333333333, 1.166666666666667,  -1.833333333333333, 2.166666666666667, 3.166666666666667],  [-2.0, 0.0, 2.0, -1.0, 0.0, 1.0]  ] |

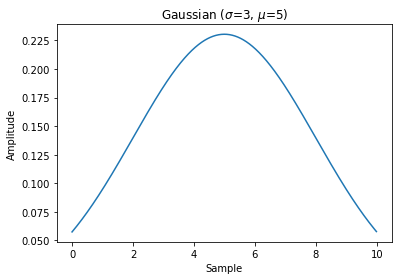
|  |
| --- |
| matrix\_2 = np.asarray(matrix\_2) |

|  |
| --- |
| N = 6  print("Ma trận hiệp phương sai với N = " + str(N))  print((1/(N-1))\*np.dot(matrix\_2, matrix\_2.T)) |
| Ma trận hiệp phương sai với N = 6  [[6.16666667 2.6 ]  [2.6 2. ]] |

**Bài 5:** Tạo hàm mật độ của phân bố Gauss với mean là 5 và variance là 3. Plot hàm kết quả.

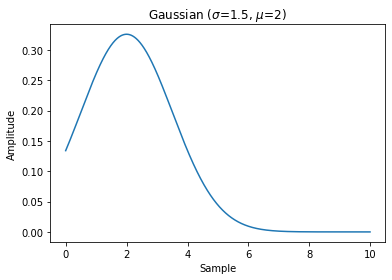
|  |
| --- |
| def gauss(x, mean, variance):    return (1/(math.sqrt(2\*math.pi\*variance)))\*math.exp((-1/2)\*((x-mean)/variance)\*\*2) |

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import math  mean = 5  variance = 3  list\_x = np.arange(0, 10, 0.01)  list\_y = []  for x in list\_x:    list\_y.append(gauss(x, mean, variance))  plt.plot(list\_x, list\_y)  plt.title(r"Gaussian ($\sigma$="+ str(variance) + ", $\mu$=" + str(mean) + ")")  plt.ylabel("Amplitude")  plt.xlabel("Sample")  plt.show() |



**Bài 6:** Tạo hàm mật độ của phân bố Gauss khác với mean là 2 và variance là 1.5. Plot hàm này trong cùng cửa sổ với hàm được tạo ra ở câu 3. Cho nhận xét.

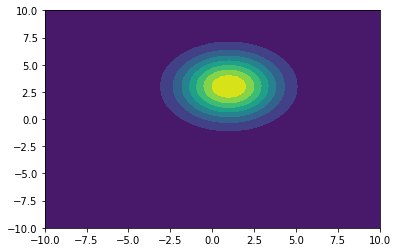
|  |
| --- |
| mean = 2  variance = 1.5  list\_x = np.arange(0, 10, 0.01)  list\_y = []  for x in list\_x:    list\_y.append(gauss(x, mean, variance))  plt.plot(list\_x, list\_y)  plt.title(r"Gaussian ($\sigma$="+ str(variance) + ", $\mu$=" + str(mean) + ")")  plt.ylabel("Amplitude")  plt.xlabel("Sample")  plt.show() |



Nhận xét: biểu đồ bị lệch sang trái so với biểu đồ có mean = 3 và variance = 5

**Bài 7:** Tạo hàm mật độ phân bố Gauss 2 chiều với mean [1 3] và variance [2 2]. Plot hàm này trên lưới [-10 10] x [-10 10] và tính khoảng cách Mahalanobis đối với các mẫu [0 0], [3 4], và [1 2].

|  |
| --- |
| import numpy as np  def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):      return 1. / (2. \* np.pi \* sx \* sy) \* np.exp(-((x - mx)\*\*2. / (2. \* sx\*\*2.) + (y - my)\*\*2. / (2. \* sy\*\*2.)))  x = np.linspace(-10, 10)  y = np.linspace(-10, 10)  x, y = np.meshgrid(x, y)  z = gaus2d(x, y, 1, 3, 2, 2)  plt.contourf(x,y,z)  plt.show() |



|  |
| --- |
| sample\_1 = [0,0]  list\_x1 = []  for i in range(50):    list\_x1.append(sample\_1)  m1 = np.asarray(list\_x1)  print("Khoảng cách mahalanobis của điểm [0,0]:")  print(np.dot(np.dot(m1.T,np.cov(z)), m1))  print()  sample\_1 = [3,4]  list\_x1 = []  for i in range(50):    list\_x1.append(sample\_1)  m1 = np.asarray(list\_x1)  print("Khoảng cách mahalanobis của điểm [3,4]:")  print(np.dot(np.dot(m1.T,np.cov(z)), m1))  print()  sample\_1 = [1,2]  list\_x1 = []  for i in range(50):    list\_x1.append(sample\_1)  m1 = np.asarray(list\_x1)  print("Khoảng cách mahalanobis của điểm [1,2]:")  print(np.dot(np.dot(m1.T,np.cov(z)), m1))  print() |
| Khoảng cách mahalanobis của điểm [0,0]:  [[0. 0.]  [0. 0.]]  Khoảng cách mahalanobis của điểm [3,4]:  [[0.24855543 0.33140724]  [0.33140724 0.44187632]]  Khoảng cách mahalanobis của điểm [1,2]:  [[0.02761727 0.05523454]  [0.05523454 0.11046908]] |

**Bài 8:** Xây dựng bộ classifier sử dụng 1 đặc trưng có sẳn.

* + - ***Load data:***
  + Load 2 file tương ứng cho 2 class là: *class1.txt* và *class2.txt*.
  + Cho biết số mẫu và số đặc trưng của mỗi class.
  + Tính mean, variance, và covariance của các vector đặc trưng.
  + Trích chọn 1 đặc trưng.

|  |
| --- |
| def covariance(x):    N = len(x)    TB = mean(x)    M = []    for i in x:      M.append(i - TB)    D = np.asarray(M)    return  (1/(N-1))\*D\*D.T |

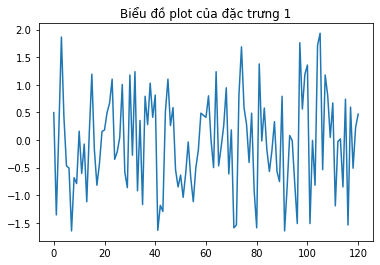
|  |
| --- |
| datContent = [i.strip().split() for i in open("twoclass.dat", "r").readlines()]  print("Số mẫu là: " + str(len(datContent)))  print("Số đặc trưng của mỗi class là: " + str(len(datContent[0][:-2])))  for i in range(len(datContent[0][:-2])):    print("Trung bình của đặc trưng " + str(i+1) + " là: " + str(mean([float(row[i]) for row in datContent])))    print("Phương sai của đặc trưng " + str(i+1) + " là: " + str(variance([float(row[i]) for row in datContent])))    print("Hiệp phương sai của đặc trưng " + str(i+1) + " là: " + str(np.cov([float(row[i]) for row in datContent])))    print()  # Chọn đặc trưng 1  feature\_1 = [float(row[0]) for row in datContent]  feature\_2 = [float(row[1]) for row in datContent]  feature\_3 = [float(row[2]) for row in datContent]  feature\_4 = [float(row[3]) for row in datContent]  class\_1 = [int(row[4]) for row in datContent]  # Tạo list index  idx = np.arange(0, len(feature\_1))  # Ranđom lại list index  np.random.shuffle(idx)  # Lấy ra phân nửa list index đã random  idx\_train = idx[:len(feature\_1)//2]  idx\_test = idx[len(feature\_1)//2:]  # Tách tập train  feature\_1\_train = [feature\_1[i] for i in idx\_train]  feature\_2\_train = [feature\_2[i] for i in idx\_train]  feature\_3\_train = [feature\_3[i] for i in idx\_train]  feature\_4\_train = [feature\_4[i] for i in idx\_train]  class\_train = [class\_1[i] for i in idx\_train]  # Tách tập test  feature\_1\_test = [feature\_1[i] for i in idx\_test]  feature\_2\_test = [feature\_2[i] for i in idx\_test]  feature\_3\_test = [feature\_3[i] for i in idx\_test]  feature\_4\_test = [feature\_4[i] for i in idx\_test]  class\_test = [class\_1[i] for i in idx\_test] |
| Số mẫu là: 242  Số đặc trưng của mỗi class là: 4  Trung bình của đặc trưng 1 là: 0.06021218181818183  Phương sai của đặc trưng 1 là: 0.8578042003907271  Hiệp phương sai của đặc trưng 1 là: 0.8613635539193196  Trung bình của đặc trưng 2 là: 0.055756892561983476  Phương sai của đặc trưng 2 là: 1.0511796321486822  Hiệp phương sai của đặc trưng 2 là: 1.055541373360918  Trung bình của đặc trưng 3 là: 1.4545952520661152  Phương sai của đặc trưng 3 là: 4.234982801030213  Hiệp phương sai của đặc trưng 3 là: 4.252555343773077  Trung bình của đặc trưng 4 là: -1.388338768595042  Phương sai của đặc trưng 4 là: 4.359071929443203  Hiệp phương sai của đặc trưng 4 là: 4.377159364835083 |

* + - ***Xây dựng classifier ứng với 1 đặc trưng được chọn:***
  + Chia tập dữ liệu thành 2 tập con, tập huấn luyện gồm 50%.
  + Tính histogram và plot:
* Xác định giá trị biệt số (ngưỡng). Ứng với histogram được tính, chọn biệt số sao cho giá trị misclassification là nhỏ nhất.
* Đánh giá trên tập dữ liệu test: ứng với giá trị ngưỡng được chọn (biệt số), chúng ta cần phải đánh giá trên 1 tập dữ liệu khác (dữ liệu không được dùng trong huấn luyện); tập dữ liệu test. Dựa vào các giá trị đặc trưng tưng ứng cho các mẫu test và giá trị ngưỡng được xác định, tính phần trăm lỗi.

Tương tự, thực nghiệm với các cách phân chia tập dữ liệu khác nhau (tập dữ liệu huấn luyện là 60%, 70%, và 80%) và các bin của histogram khác nhau.

Cho biết đặc trưng có khả năng phân loại tốt nhất.

|  |
| --- |
| import matplotlib.pyplot as plt  plt.hist(feature\_1\_train)  plt.title("Biểu đồ histogram của đặc trưng 1")  plt.show()  plt.plot(feature\_1\_train)  plt.title("Biểu đồ plot của đặc trưng 1")  plt.show() |

|  |
| --- |
| def find\_thresh(x, label):    thresh = None    num\_errors = 999999    for i in x:      errors = 0      for d, l in zip(x, label):        if d > i and l == 1:          errors += 1        elif d <= i and l == 0:          errors += 1      if errors < num\_errors:        num\_errors = errors        thresh = i    return (thresh, num\_errors)  def find\_error\_with\_thresh(x, label, thresh):    num\_errors = 0    for d, l in zip(x, label):      if d > thresh and l == 1:        num\_errors += 1      elif d <= thresh and l == 0:        num\_errors += 1    return num\_errors |

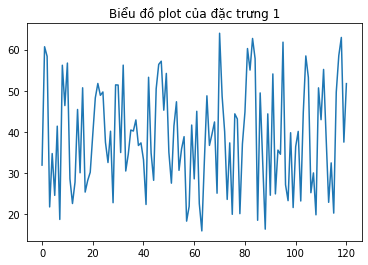
|  |
| --- |
| # Đặc trưng 1  fea\_1\_test = find\_thresh(feature\_1\_train, class\_train)  print("Ngưỡng tìm được từ tập train đặc trưng 1 là: " + str(fea\_1\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 1 là: " + str(fea\_1\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 1 là: "  + str(find\_error\_with\_thresh(feature\_1\_test, class\_test, fea\_1\_test[0])))  print()  # Đặc trưng 2  fea\_2\_test = find\_thresh(feature\_2\_train, class\_train)  print("Ngưỡng tìm được từ tập train đặc trưng 2 là: " + str(fea\_2\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 2 là: " + str(fea\_2\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 2 là: "  + str(find\_error\_with\_thresh(feature\_2\_test, class\_test, fea\_2\_test[0])))  print()  # Đặc trưng 3  fea\_3\_test = find\_thresh(feature\_3\_train, class\_train)  print("Ngưỡng tìm được từ tập train đặc trưng 3 là: " + str(fea\_3\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 3 là: " + str(fea\_3\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 3 là: "  + str(find\_error\_with\_thresh(feature\_3\_test, class\_test, fea\_3\_test[0])))  print()  # Đặc trưng 4  fea\_4\_test = find\_thresh(feature\_4\_train, class\_train)  print("Ngưỡng tìm được từ tập train đặc trưng 4 là: " + str(fea\_4\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 4 là: " + str(fea\_4\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 4 là: "  + str(find\_error\_with\_thresh(feature\_4\_test, class\_test, fea\_4\_test[0])))  print() |
| Ngưỡng tìm được từ tập train đặc trưng 1 là: 1.761705  Số lỗi tìm được từ tập train đặc trưng 1 là: 55  Số lỗi tìm được từ tập test đặc trưng 1 là: 66  Ngưỡng tìm được từ tập train đặc trưng 2 là: 2.284433  Số lỗi tìm được từ tập train đặc trưng 2 là: 57  Số lỗi tìm được từ tập test đặc trưng 2 là: 64  Ngưỡng tìm được từ tập train đặc trưng 3 là: 7.060067  Số lỗi tìm được từ tập train đặc trưng 3 là: 57  Số lỗi tìm được từ tập test đặc trưng 3 là: 66  Ngưỡng tìm được từ tập train đặc trưng 4 là: 2.397017  Số lỗi tìm được từ tập train đặc trưng 4 là: 57  Số lỗi tìm được từ tập test đặc trưng 4 là: 65 |

Từ kết quả trên ta có thể thấy là đặc trưng 1 và 4 là cho kết quả tốt nhất. Vì số điểm lỗi trên tập train với thực tế chỉ cách nhau thấp nhất.

**Bài 9:** Thực hiện tương tự như bài 8 với 2 tập dữ liệu là cross (cross.dat) và twoclass (twoclass.dat). Chú ý dữ liệu cho các class đều gom chung 1 file và 2 đặc trưng cuối để chỉ ra class, ví dụ [1 0] tương ứng cho class1 và [0 1] tương ứng cho class2.

|  |
| --- |
| datContent\_cross = [i.strip().split() for i in open("cross.dat", "r").readlines()]  print("Số mẫu là: " + str(len(datContent\_cross)))  print("Số đặc trưng của mỗi class là: "  + str(len(datContent\_cross[0][:-2])))  for i in range(len(datContent\_cross[0][:-2])):    print("Trung bình của đặc trưng " + str(i+1) + " là: "  + str(mean([float(row[i]) for row in datContent\_cross])))    print("Phương sai của đặc trưng " + str(i+1) + " là: "  + str(variance([float(row[i]) for row in datContent\_cross])))    print("Hiệp phương sai của đặc trưng " + str(i+1) + " là: "  + str(np.cov([float(row[i]) for row in datContent\_cross])))  # Chọn đặc trưng 1  feature\_1\_cross = [float(row[0]) for row in datContent\_cross]  feature\_2\_cross = [float(row[1]) for row in datContent\_cross]  class\_1\_cross = [int(row[2]) for row in datContent\_cross]  # Tạo list index  idx = np.arange(0, len(feature\_1\_cross))  # Ranđom lại list index  np.random.shuffle(idx)  # Lấy ra phân nửa list index đã random  idx\_train = idx[:len(feature\_1)//2]  idx\_test = idx[len(feature\_1)//2:]  # Tách tập train  feature\_1\_train\_cross = [feature\_1\_cross[i] for i in idx\_train]  feature\_2\_train\_cross = [feature\_2\_cross[i] for i in idx\_train]  class\_train\_cross = [class\_1\_cross[i] for i in idx\_train]  # Tách tập test  feature\_1\_test\_cross = [feature\_1\_cross[i] for i in idx\_test]  feature\_2\_test\_cross = [feature\_2\_cross[i] for i in idx\_test]  class\_test\_cross = [class\_1\_cross[i] for i in idx\_test] |
| Số mẫu là: 200  Số đặc trưng của mỗi class là: 2  Trung bình của đặc trưng 1 là: 40.38131579000002  Phương sai của đặc trưng 1 là: 157.35237854889428  Hiệp phương sai của đặc trưng 1 là: 158.14309401898927  Trung bình của đặc trưng 2 là: 40.393948170000016  Phương sai của đặc trưng 2 là: 171.44175718153042  Hiệp phương sai của đặc trưng 2 là: 172.30327354927695 |

|  |
| --- |
| import matplotlib.pyplot as plt  plt.hist(feature\_1\_train\_cross)  plt.title("Biểu đồ histogram của đặc trưng 1")  plt.show()  plt.plot(feature\_1\_train\_cross)  plt.title("Biểu đồ plot của đặc trưng 1")  plt.show() |

****

|  |
| --- |
| # Đặc trưng 1  fea\_1\_cross\_test = find\_thresh(feature\_1\_train\_cross, class\_train\_cross)  print("Ngưỡng tìm được từ tập train đặc trưng 4 là: " + str(fea\_1\_cross\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 4 là: " + str(fea\_1\_cross\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 4 là: "+str(find\_error\_with\_thresh(feature\_1\_test\_cross, class\_test\_cross, fea\_1\_cross\_test[0])))  print()  # Đặc trưng 2  fea\_2\_cross\_test = find\_thresh(feature\_2\_train\_cross, class\_train\_cross)  print("Ngưỡng tìm được từ tập train đặc trưng 4 là: " + str(fea\_2\_cross\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 4 là: " + str(fea\_2\_cross\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 4 là: " + str(find\_error\_with\_thresh(feature\_2\_test\_cross, class\_test\_cross, fea\_2\_cross\_test[0]))) |
| Ngưỡng tìm được từ tập train đặc trưng 4 là: 33.264324  Số lỗi tìm được từ tập train đặc trưng 4 là: 34  Số lỗi tìm được từ tập test đặc trưng 4 là: 32  Ngưỡng tìm được từ tập train đặc trưng 4 là: 32.043167  Số lỗi tìm được từ tập train đặc trưng 4 là: 35  Số lỗi tìm được từ tập test đặc trưng 4 là: 26 |

Từ kết quả trên ta có thể thấy là đặc trưng 2 là cho kết quả tốt nhất. Vì số điểm lỗi trên tập test thấp.

**BÀI THỰC HÀNH SỐ 3**

**I. Mục đích:**

Xây dựng classifier và biệt thức dựa trên luật Bayes.

**II. Báo cáo:**

**Bài 1:** Cho 2 tập dữ liệu class A (classA.mat) và class B (classB.mat). Xây dựng bộ classifier với 2 đặc trưng. Giả sử hai tập dữ liệu có dạng phân bố Gauss có cùng ma trận hiệp phương sai là SIGMA=[1 0; 0 1].

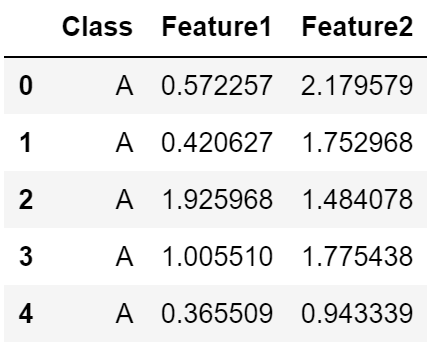
* *Bước 1:* Import thư viện:

|  |
| --- |
| import **numpy** as np  from **scipy.io** import loadmat  import **pandas** as pd  from **matplotlib** import pyplot as plt  import **matplotlib.colors** as colors  import **seaborn** as sns  import **itertools**  from **scipy.stats** import norm  import **scipy.stats**  from **sklearn.naive\_bayes** import GaussianNB  import **scipy.io**  from **sklearn.model\_selection** import train\_test\_split  from **sklearn.metrics** import confusion\_matrix |

*- Bước 2:* Load dữ liệu: Load 2 file tương ứng cho 2 class là: *classA.mat* và *classB.mat*.

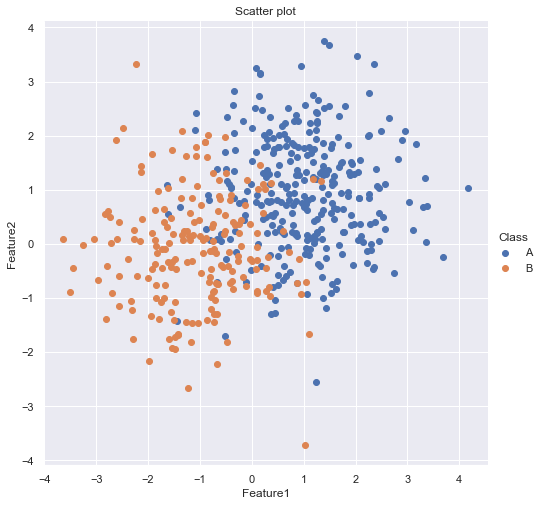
|  |
| --- |
| columnsName = ['Feature1','Feature2']  classA = pd.**DataFrame**(loadmat("classA.mat")['classA'],columns =  columnsName)  classB = pd.**DataFrame**(loadmat("classB.mat")['classB'],columns =  columnsName) |

|  |
| --- |
| #Merge data  classAB = pd.**concat**([classA,classB], keys=['A', 'B']).**reset\_index**().**drop**('level\_1', axis=1).**rename**(columns =  {'level\_0': '**Class**'})  classAB.**head**() |



- Plot dữ liệu.

|  |
| --- |
| sns.**FacetGrid**(**classAB**, hue="Class",  height=7).**map**(plt.scatter,"Feature1","Feature2",).add\_legend()  plt.**title**('Scatter plot')  plt.**show**() |



- Xác định số mẫu của mỗi class.

|  |
| --- |
| #Estimating the data  nA = len(**classA**)  nB = len(**classB**)  print("Số mẫu của class A là :",**nA**)  print("Số mẫu của class B là :",**nB**) |
| Số mẫu của class A là : 300  Số mẫu của class B là : 195 |

- Phân chia tập dữ liệu thành 2 tập con: tập huấn luyện (60%) và tập kiểm thử (40%).

|  |
| --- |
| train\_set, test\_set = **train\_test\_split**(**classAB**, train\_size=0.7)  print("Số lượng của tập train là :",len(**train\_set**))  print("Số lượng của tập train là :",len(**test\_set**)) |
| Số lượng của tập train là : **346**  Số lượng của tập train là : **149** |

**Huấn luyện**

- Tính mean tương ứng cho từng class.

|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(**train\_set**.groupby(**'Class'**).**mean**().**values**,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(**train\_set**.groupby(**'Class'**).**cov**().**values**,[2])  pi\_list = **train\_set**.iloc[:,0].value\_counts().values / len(train\_set)  print("Mean của từng class A trong train set là : ",mu\_list[0][0])  print("Mean của từng class B trong train set là : ",mu\_list[1][0]) |
| Mean của từng class A trong train set là : [1.02409221 0.89535824]  Mean của từng class B trong train set là : [-1.02802323 -0.11705038] |

- Xây dựng biệt thức (discriminant function). Tham khảo bài giảng.

|  |
| --- |
| #Build discriminant functions  def **DF**(X,mu\_list,cov\_list,pi\_list):  **scores\_list** = []  **classes** = len(mu\_list)  for p in range(**classes**):  Wi = (-1/2)\*np.linalg.inv(cov\_list[p])  wi = np.linalg.inv(cov\_list[p])@mu\_list[p][0]  wi0 = (-1/2)\*np.transpose(**mu\_list**[p][0])@np.linalg.inv(**cov\_list**[p])@**mu\_list**[p][0]  + (-1/2)\*np.log(np.linalg.norm(**cov\_list**[p]))  + np.log(**pi\_list**[p])  score = np.transpose(X)@**Wi**@**X** + np.transpose(**wi**)@**X** + **wi0**  # print(np.transpose(X)@**Wi**@**X**)  scores\_list.append(**score**)  # print(**scores\_list**)  return np.argmax(**scores\_list**) |

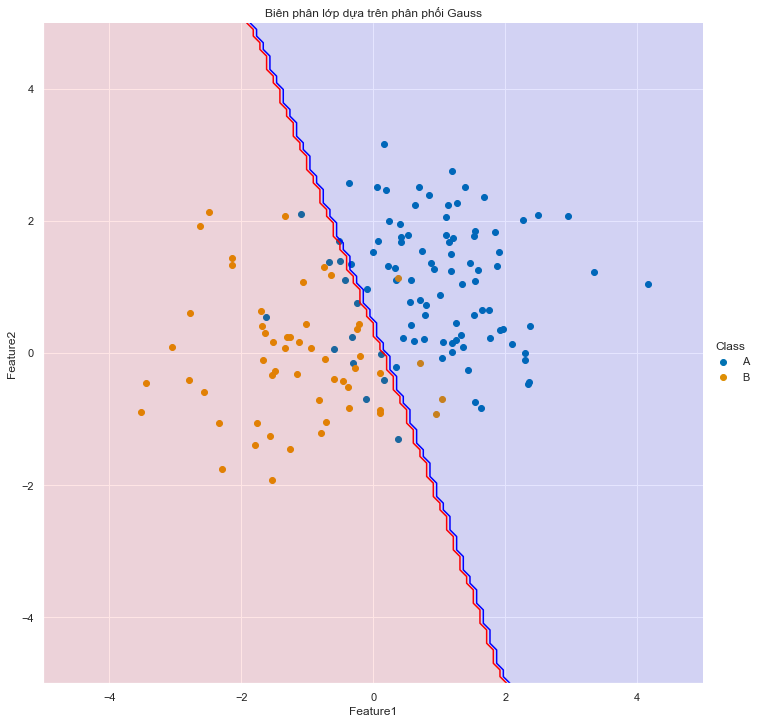
• Đánh giá

- Đánh giá trên tập dữ liệu test. Tính độ chính xác cho từng tập và xây dựng confusion matrix.

|  |
| --- |
| prediction = ["A" if **DF**(np.array([x,y]).reshape(-1,1),mu\_list, cov\_list, pi\_list)==0 else "B" for x, y in test\_set[["Feature1","Feature2"]].values]  label = list(test\_set['Class'].values)  print(pd.DataFrame(confusion\_matrix(label, prediction), index=['Class A', 'Class B'], columns=['Class A', 'Class B'])) |
| Class A Class B  Class A 82 13  Class B 4 50 |

- Plot dữ liệu testing của 2 lớp và đường biên phân lớp trên cùng một hình.

|  |
| --- |
| #Plot with boundary contours  N = 100  X = np.linspace(-5, 5, N)  Y = np.linspace(-5, 5, N)  X, Y = np.**meshgrid**(X, Y)  #Configure plot  color\_list = ['Blues','Reds']  g = sns.**FacetGrid**(test\_set, hue="Class", height=10, palette = 'colorblind', hue\_order=["A","B"]).map(plt.scatter,  "Feature1","Feature2",).add\_legend()  my\_ax = g.ax  #Computing the predicted class function for each value on the grid  zz = np.array( [**DF**( np.array([xx,yy]).reshape(-1,1),mu\_list, cov\_list, pi\_list) for xx, yy in zip(np.ravel(X),  np.**ravel**(Y)) ] )  # zz = np.array( [predict\_NB\_gaussian\_class( np.array([xx,yy]).reshape(-1,1),mu\_list, std\_list, pi\_list)  # for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )  #Reshaping the predicted class into the meshgrid shape  Z = zz.reshape(X.shape)  #Plot the filled and boundary contours  my\_ax.contourf( X, Y, Z, 1, alpha = .1, colors = ('blue','red'))  my\_ax.contour( X, Y, Z, 1, alpha = 1, colors = ('blue','red'))  # Addd axis and title  my\_ax.set\_xlabel(**'Feature1'**)  my\_ax.set\_ylabel(**'Feature2'**)  my\_ax.set\_title('Biên phân lớp dựa trên phân phối Gauss')  plt.show() |



**Trường hợp tập dữ liệu huấn luyện là 75%:**

* Chia tập dữ liệu vào train và test:

|  |
| --- |
| #Split train and test  train\_set, test\_set = **train\_test\_split**(classAB, train\_size=0.75)  print("Số lượng của tập train là :",len(**train\_set**))  print("Số lượng của tập train là :",len(**test\_set**)) |
| Số lượng của tập train là : 371  Số lượng của tập train là : 124 |

- Tính mean tương ứng cho từng class.

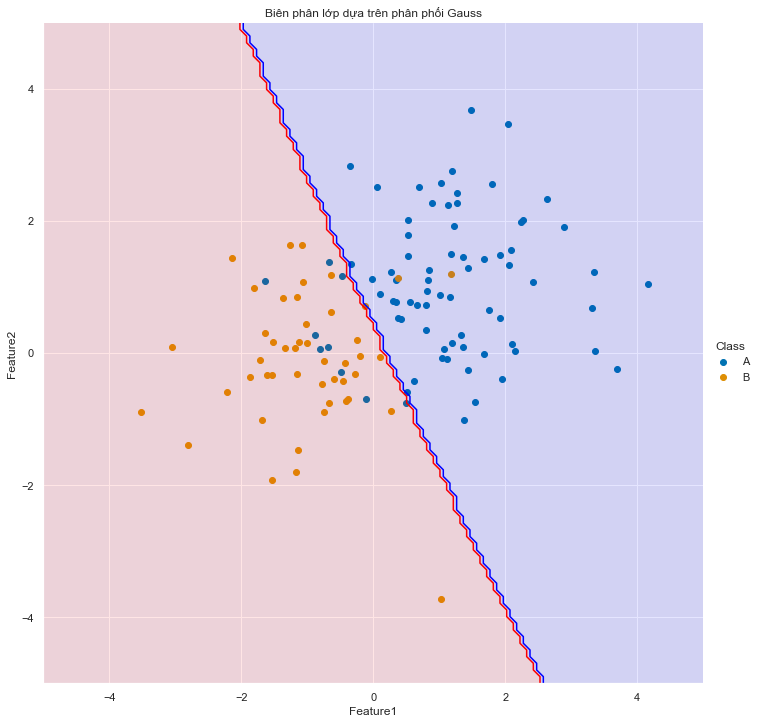
|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(**train\_set**.groupby('Class').mean().values,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(train\_set.groupby('Class').cov().values,[2])  pi\_list = train\_set.iloc[:,0].value\_counts().values / len(train\_set)  print("Mean của từng class A trong train set là : ",mu\_list[0][0])  print("Mean của từng class B trong train set là : ",mu\_list[1][0]) |
| Mean của từng class A trong train set là : [0.96513234 0.91431234]  Mean của từng class B trong train set là : [-1.09776943 -0.10862826] |

- Đánh giá trên tập dữ liệu test. Tính độ chính xác cho từng tập và xây dựng confusion matrix.

|  |
| --- |
| #Confusion matrix  prediction = ["A" if **DF**(np.array([x,y]).reshape(-1,1),mu\_list, cov\_list, pi\_list)==0 else "B"  for x, y in test\_set[["**Feature1**","**Feature2**"]].values]  label = list(test\_set[**'Class'**].values)  print(pd.**DataFrame**(confusion\_matrix(label, prediction), index=['Class A', 'Class B'],  columns=['Class A', 'Class B'])) |
| Class A Class B  Class A 68 9  Class B 3 44 |

- Plot dữ liệu testing của 2 lớp và đường biên phân lớp trên cùng một hình.

|  |
| --- |
| #Plot with boundary contours  N = 100  X = np.**linspace**(-5, 5, N)  Y = np.**linspace**(-5, 5, N)  X, Y = np.**meshgrid**(X, Y)  #Configure plot  color\_list = ['Blues','Reds']  g = sns.**FacetGrid**(test\_set, hue="Class", height=10, palette = 'colorblind', hue\_order=["A","B"]).map(plt.scatter,  "**Feature1**","**Feature2**",).add\_legend()  my\_ax = g.**ax**  #Computing the predicted class function for each value on the grid  zz = np.array( [**DF**( np.array([xx,yy]).reshape(-1,1),mu\_list, cov\_list, pi\_list) for xx, yy in zip(np.**ravel**(X),  np.**ravel**(Y)) ] )  # zz = np.array( [predict\_NB\_gaussian\_class( np.array([xx,yy]).reshape(-1,1),mu\_list, std\_list, pi\_list)  # for **xx, yy** in **zip**(np.ravel(X), np.ravel(Y)) ] )  #Reshaping the predicted class into the meshgrid shape  Z = zz.**reshape**(X.shape)  #Plot the filled and boundary contours  my\_ax.**contourf**( X, Y, Z, 1, **alpha** = .1, **colors** = ('blue','red'))  my\_ax.**contour**( X, Y, Z, 1, **alpha** = 1, **colors** = ('blue','red'))  # Addd axis and title  my\_ax.set\_xlabel(**'Feature1'**)  my\_ax.set\_ylabel(**'Feature2'**)  my\_ax.set\_title('Biên phân lớp dựa trên phân phối Gauss')  plt.show() |



Trường hợp tập dữ liệu huấn luyện là 80%:

Chia tập dữ liệu thành 2 tập train và test:

|  |
| --- |
| #Split train and test  **train\_set, test\_set** = **train\_test\_split**(**classAB**, train\_size=0.8)  print("Số lượng của tập train là :",len(**train\_set**))  print("Số lượng của tập train là :",len(**test\_set**)) |
| Số lượng của tập train là : 396  Số lượng của tập train là : 99 |

- Tính mean tương ứng cho từng class.

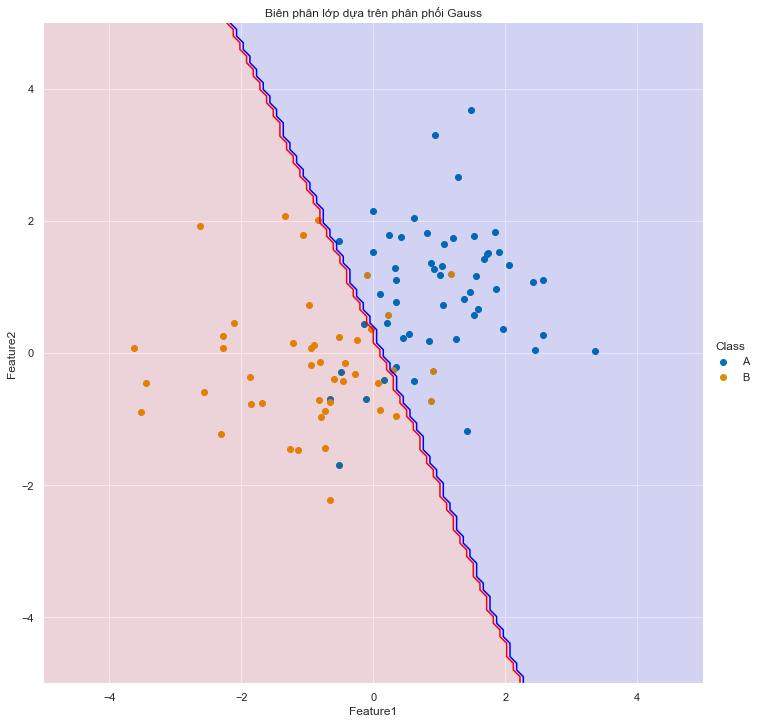
|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(train\_set.groupby('Class').mean().values,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(**train\_set**.groupby('Class').cov().values,[2])  pi\_list = **train\_set**.iloc[:,0].value\_counts().values / len(**train\_set**)  print("Mean của từng class A trong train set là : ",mu\_list[0][0])  print("Mean của từng class B trong train set là : ",mu\_list[1][0]) |
| Mean của từng class A trong train set là : [1.00020499 0.9389721 ]  Mean của từng class B trong train set là : [-1.10655257 -0.09811678] |

- Đánh giá trên tập dữ liệu test. Tính độ chính xác cho từng tập và xây dựng confusion matrix.

|  |
| --- |
| #Confusion matrix  prediction = ["A" if **DF**(np.array([x,y]).reshape(-1,1),mu\_list, cov\_list, pi\_list)==0 else "B"  for x, y in test\_set[["**Feature1**","**Feature2**"]].values]  label = list(test\_set[**'Class'**].values)  print(pd.**DataFrame**(**confusion\_matrix**(label, prediction), index=['Class A', 'Class B'],  columns=[**'Class A'**, **'Class B'**])) |
| Class A Class B  Class A 48 6  Class B 7 38 |

- Plot dữ liệu testing của 2 lớp và đường biên phân lớp trên cùng một hình.

|  |
| --- |
| #Plot with boundary contours  N = 100  X = np.**linspace**(-5, 5, N)  Y = np.**linspace**(-5, 5, N)  X, Y = np.**meshgrid**(X, Y)  #Configure plot  color\_list = ['Blues','Reds']  g = sns.**FacetGrid**(**test\_set**, hue="Class", height=10, palette = 'colorblind', hue\_order=["A","B"]).map(plt.scatter,  "**Feature1**","**Feature2**",).add\_legend()  my\_ax = g.ax  #Computing the predicted class function for each value on the grid  zz = np.array( [**DF**( np.array([xx,yy]).reshape(-1,1),mu\_list, cov\_list, pi\_list) for **xx, yy** in **zip**(np.ravel(X),  np.**ravel**(Y)) ] )  # zz = np.array( [**predict\_NB\_gaussian\_class**( np.array([xx,yy]).reshape(-1,1),mu\_list, std\_list, pi\_list)  # for xx, yy in **zip**(np.**ravel**(X), np.**ravel**(Y)) ] )  #Reshaping the predicted class into the meshgrid shape  Z = zz.reshape(X.shape)  #Plot the filled and boundary contours  my\_ax.**contourf**( X, Y, Z, 1, **alpha** = .1, colors = ('blue','red'))  my\_ax.**contour**( X, Y, Z, 1, **alpha** = 1, colors = ('blue','red'))  # Addd axis and title  my\_ax.**set\_xlabel**(**'Feature1'**)  my\_ax.**set\_ylabel**(**'Feature2'**)  my\_ax.**set\_title**('Biên phân lớp dựa trên phân phối Gauss')  plt.show() |



3. Xây dựng bộ classifier với 2 lớp, 2 đặc trưng. Giả sử tập dữ liệu có dạng phân bố Gauss. Tập dữ liệu huấn luyện là class1\_train.mat và class2\_train.mat. Tập dữ liệu đánh giá là class1\_test.mat và class2\_test.mat.

**• *Load data:***

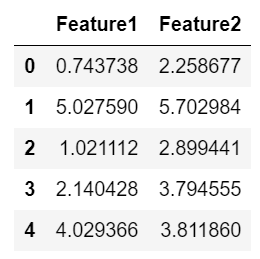
- Import thư viện:

|  |
| --- |
| import **numpy** as np  from **scipy**.io import loadmat  import **pandas** as pd  from **matplotlib** import pyplot as plt  import **matplotlib**.colors as colors  import **seaborn** as sns  import **itertools**  from **scipy**.stats import norm  import **scipy**.stats  from **sklearn**.naive\_bayes import GaussianNB  import **scipy**.io  from **sklearn**.model\_selection import train\_test\_split  from **sklearn**.metrics import confusion\_matrix |

- Load 2 file tương ứng cho 2 class là: *class1\_train.mat* và *class2\_train.mat*. Sử dụng lệnh:

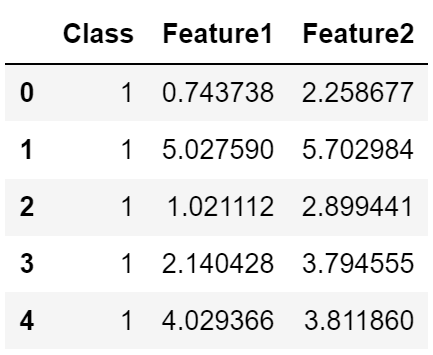
|  |
| --- |
| #Load data  columnsName = ['Feature1','Feature2']  class1\_train = pd.DataFrame(loadmat(**'class1\_train**.mat')[**'class1\_train'**],  columns = columnsName)  class2\_train = pd.DataFrame(scipy.io.loadmat('class2\_train.mat')['class2\_train'],  columns = columnsName)  class1\_test = pd.**DataFrame**(scipy.io.loadmat('class1\_test.mat')['class1\_test'],  columns = columnsName)  class2\_test = pd.DataFrame(scipy.io.loadmat('class2\_test.mat')['class2\_test'],  columns = columnsName) |

|  |
| --- |
| **class1\_train**.head() |



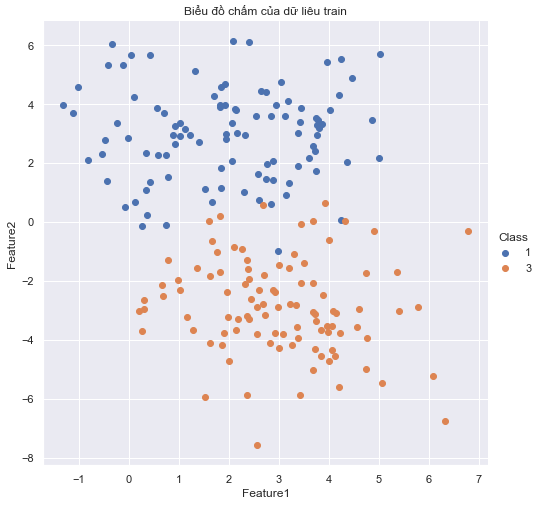
|  |
| --- |
| #Merge data  trainMerge = pd.concat([class1\_train,class2\_train], keys=['1', '3']).reset\_index().drop('level\_1',  axis=1).rename(columns = {'level\_0': 'Class'})  testMerge = pd.concat([class1\_test,class2\_test], keys=['1', '2']).reset\_index().drop('level\_1',  axis=1).rename(columns = {'level\_0': 'Class'}) |

|  |
| --- |
| trainMerge.head() |

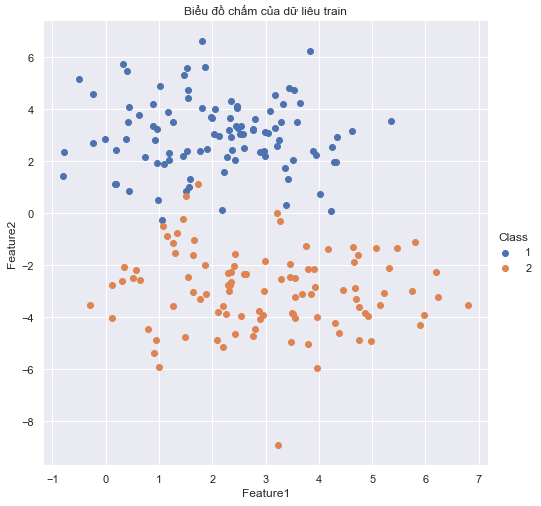


- Plot dữ liệu.

|  |
| --- |
| #Plot data  sns.set()  sns.**FacetGrid**(trainMerge, hue="Class", height=7).**map**(plt.scatter,"Feature1","Feature2",).add\_legend()  plt.title('Biểu đồ chấm của dữ liêu train')  plt.show() |



|  |
| --- |
| #Plot data  sns.set()  sns.FacetGrid(testMerge, hue="Class", height=7).map(plt.scatter,"Feature1","Feature2",).add\_legend()  plt.title('Biểu đồ chấm của dữ liêu train')  plt.show() |

****

• *Xây dựng classifier****:***

- Tính mean, và covariance tương ứng cho từng class.

|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(trainMerge.groupby('Class').mean().values,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(trainMerge.groupby('Class').cov().values,[2])  pi\_list = trainMerge.iloc[:,0].value\_counts().values / len(trainMerge)  print("Mean của từng class A trong train set là : ",mu\_list[0][0])  print("Mean của từng class B trong train set là : ",mu\_list[1][0]) |
| Mean của từng class A trong train set là : [1.97909027 2.97947776]  Mean của từng class B trong train set là : [ 3.01056198 -2.87167377] |

- Xây dựng biệt thức (discriminant function).

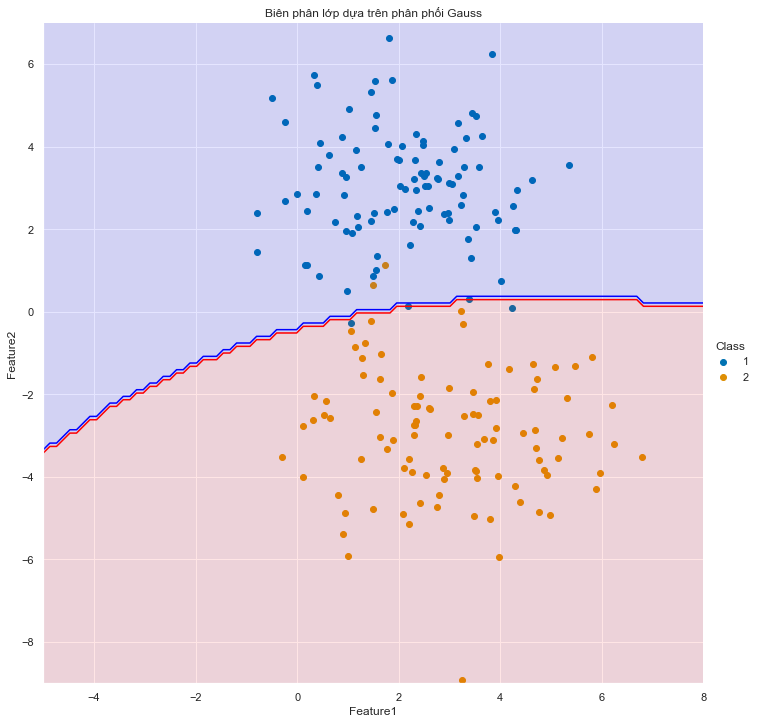
|  |
| --- |
| #Build discriminant functions  def DF(X,mu\_list,cov\_list,pi\_list):  scores\_list = []  classes = len(mu\_list)  for p in range(classes):  Wi = (-1/2)\*np.linalg.inv(cov\_list[p])  wi = np.linalg.inv(cov\_list[p])@mu\_list[p][0]  wi0 = (-1/2)\*np.transpose(mu\_list[p][0])@np.linalg.inv(cov\_list[p])@mu\_list[p][0]  - (-1/2)\*np.log(np.linalg.norm(cov\_list[p]))  + np.log(pi\_list[p])  score = np.transpose(X)@Wi@X + np.transpose(wi)@X + wi0  scores\_list.append(score)  return np.argmax(scores\_list) |

- Đánh giá trên tập dữ liệu test, *class1\_test.mat* và *class2\_test.mat*. Tính độ chính xác cho từng tập và xây dựng confusion matrix.

|  |
| --- |
| #Confusion matrix  prediction = ["1" if DF(np.array([x,y]).reshape(-1,1),mu\_list, cov\_list, pi\_list)==0 else "2"  for x, y in testMerge[["Feature1","Feature2"]].values]  label = list(testMerge['Class'].values)  print(pd.DataFrame(confusion\_matrix(label, prediction), index=['Class 1', 'Class 2'], columns=['Class 1', 'Class 2'])) |
| Class 1 Class 2  Class 1 98 2  Class 2 2 98 |

- Plot dữ liệu testing của 2 lớp và đường biên phân lớp trên cùng một hình.

|  |
| --- |
| #Plot with boundary contours  N = 100  X = np.linspace(-5, 8, N)  Y = np.linspace(-9, 7, N)  X, Y = np.meshgrid(X, Y)  #Configure plot  color\_list = ['Blues','Reds']  g = sns.FacetGrid(testMerge, hue="Class", height=10, palette = 'colorblind', hue\_order=["1","2"]).map(plt.scatter,  "Feature1","Feature2",).add\_legend()  my\_ax = g.ax  #Computing the predicted class function for each value on the grid  zz = np.array( [DF( np.array([xx,yy]).reshape(-1,1),mu\_list, cov\_list, pi\_list) for xx, yy in zip(np.ravel(X), np.ravel(Y))])  # zz = np.array( [predict\_NB\_gaussian\_class( np.array([xx,yy]).reshape(-1,1),mu\_list, std\_list, pi\_list)  # for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )  #Reshaping the predicted class into the meshgrid shape  Z = zz.reshape(X.shape)  #Plot the filled and boundary contours  my\_ax.contourf( X, Y, Z, 1, alpha = .1, colors = ('blue','red'))  my\_ax.contour( X, Y, Z, 1, alpha = 1, colors = ('blue','red'))  # Addd axis and title  my\_ax.set\_xlabel('Feature1')  my\_ax.set\_ylabel('Feature2')  my\_ax.set\_title('Biên phân lớp dựa trên phân phối Gauss')  plt.show() |



**BÀI THỰC HÀNH SỐ 4**

1. **Mục đích:**

Xây dựng bộ classifier dựa trên ước lượng Parzen Window và các tiếp cận khác.

1. **Nội dung:**

**Bài 1:** Xây dựng bộ classifier dựa trên Parzen window:

* + Load 2 file *classA.txt* và *classB.txt*.
  + Phân chia tập dữ liệu thành 2 tập con training (70%) và testing (30%).
  + Huấn luyện bộ classifier sử dụng dữ liệu 2 chiều, 2 lớp từ tập training. Áp dụng Parzen window với hàm cửa sổ hypercube



và *h*=1.

* + Sử dụng tập testing để đánh giá hiệu quả của bộ classifier.

Lặp lại với *h*=0.5 và 1.5.

**Bài 2:** Lặp lại bài 1 với hàm cửa sổ Gauss



**Bài 3:** Xây dựng bộ classifier để nhận dạng chữ số.

* + Tập dữ liệu: Class<số>.txt chứa các ảnh của số tương ứng. Mỗi mẫu chữ số có kích thước là 24x18.
  + Phân chia tập dữ liệu thành 2 tập con training (75%) và testing (25%) cho mỗi lớp.
  + Chọn tiếp cận/mô hình phân lớp.
  + Huấn luyện bộ phân lớp từ tập training.
  + Đánh giá kết quả từ tập testing.