**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\*\*\*\*\*\*🙣🟋🙡\*\*\*\*\***



***BÁO CÁO THỰC HÀNH***

***MÔN NHẬN DẠNG MẪU***

**Giáo viên hướng dẫn: Hồ Đắc Quán**

**Sinh viên thực hiện: Đinh Hoàng Hải Đăng**

**Mã số sinh viên: 17100261**

**Lớp: DHKHMT13A**

**Tp Hồ Chí Minh – 11/2020**

**Mục lục**

[**BÀI THỰC HÀNH SỐ 1** 4](#_Toc56584849)

[**Bài tập 1**: Tìm biên phân lớp cho các tập dữ liệu 4](#_Toc56584850)

[**Bài tập 2**: Xây dựng một model đơn giản để predict một phương trình đường thẳng 7](#_Toc56584851)

[**BÀI THỰC HÀNH SỐ 2** 9](#_Toc56584852)

[**Bài tập 1**: Xây dựng một phân lớp dựa trên histogram, với tập dữ liệu cho trước. 9](#_Toc56584853)

[**Bài tập 2**:Xây dựng một bộ phân lớp trên tập dữ liệu *person\_data.txt* cho các trường hợp (1) đặc tính chiều cao, (2) kết hợp chiều cao và tên. 10](#_Toc56584854)

[**Bài tập 3**:Tính mean và variance của các vector đặc trưng sau. 12](#_Toc56584855)

[**Bài tập 4**:Tính mean và variance của vector đặc trưng sau. 13](#_Toc56584856)

# **BÀI THỰC HÀNH SỐ 1**

## **Bài tập 1**: Tìm biên phân lớp cho các tập dữ liệu

1. Từ 2 tập dữ liệu cho trước:

|  |
| --- |
| a = np.array([1,2,3,2,3,4,5,6,7])  b = np.array([5,5,6,6,7,8,9,9,8]) |

* Đặt *v* là biên phân lớp của hai tập dữ liệu, ta có phát biểu sau:

+ Nếu *v\_i* <= *v*: *v\_i* thuộc lớp thứ 1, ngược lại

+ Nếu *v\_i > v: v\_i* thuộc lớp thứ 2.

* Từ đó ta cài đặt một thuật toán đơn giản sau:

+ Liệt kê tất cả các phần từ của cả 2 lớp đưa vào 1 cái *set*

+ Chọn thử từng phần tử ở trong *set* làm biên phân lớp, ta gọi là biên phân lớp tạm.

+ Chạy hai lớp, kiểm tra tổng số lượng phần tử được phân sai lớp, nếu số lượng phân tách sai bé hơn số lượng đợt trước, update giá trị biên phân lớp.

+ Kết quả cuối cùng cho ta một giá trị biên có số phần tử bị phân lớp sai ít nhất.

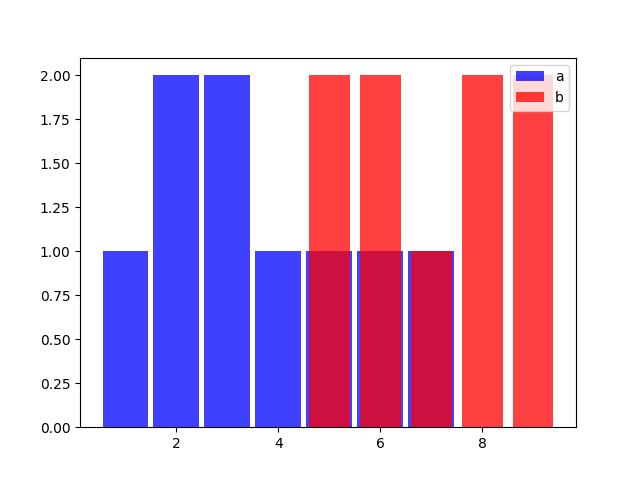
* Cài đặt hàm:

|  |
| --- |
| def find\_TH(arr1, arr2):  data\_set = set(arr1).union(arr2)  # v <= tp -> class1  # v > tp -> class2  num\_of\_errors = arr1.size + arr2.size  thread\_hold = None  for tp in data\_set:  noe = 0  for v in class1:  if (v > tp):  noe += class1[v]  for v in class2:  if (v <= tp):  noe += class2[v]  if noe < num\_of\_errors:  num\_of\_errors = noe  thread\_hold = tp  return (thread\_hold, num\_of\_errors) |

* Kết quả thu được:

|  |
| --- |
| First array : [1 2 2 3 3 4 5 6 7]  Second array: [5 5 6 6 7 8 8 9 9]  Thread Hold : 4  Number of errors: 3  New first array : [1, 2, 2, 3, 3, 4]  New second array: [5, 5, 6, 6, 7, 8, 8, 9, 9] |

* Plot:



1. Từ 2 tập dữ liệu ngẫu nhiên

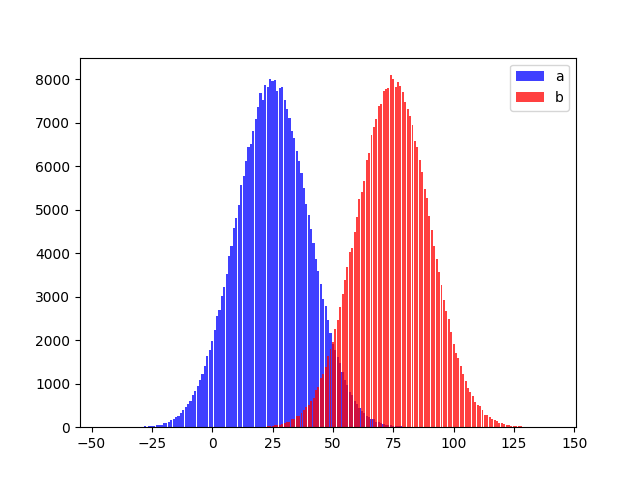
* Lần này ta sẽ sử dụng hàm phân bố chuẩn để sinh tập dữ liệu ngẫu nhiên

|  |
| --- |
| a = np.random.normal(25, 15, size=300000).round(0).astype(np.int)  b = np.random.normal(75, 15, size=300000).round(0).astype(np.int) |

* Ở đây ta tạo hai bộ dữ liệu chỉ chứa số nguyên, có 300000 phần tử.
* Kết quả thu được:

|  |
| --- |
| First array : [-45 -43 -41 ... 88 91 92]  Second array: [ 3 8 11 ... 138 140 141]  Thread Hold : 49  Number of errors: 29002  New first array : [-45 -43 -41 ... 49 49 49]  New second array: [ 50 50 50 ... 138 140 141] |

* Plot dữ liệu:



1. Đọc hai tập dữ liệu từ file.

* Ta thực hiện đồng thời hai việc

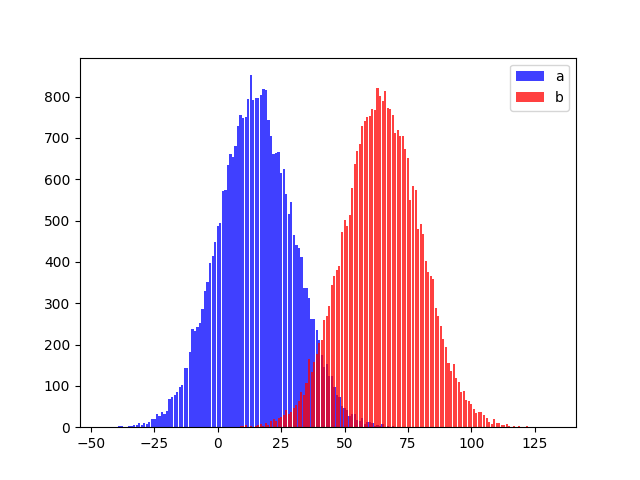
+ Sinh dữ liệu như bước b, và lưu xuống file.

+ Đọc file dữ liệu ta vừa sinh.

* Các việc tiếp theo thì như các bước trước.
* Kết quả thu được:

|  |
| --- |
| First array : [-45 -40 -39 ... 73 75 76]  Second array: [ 1 9 9 ... 122 128 132]  Thread Hold : 40  Number of error: 2884  New first array : [-45 -40 -39 ... 40 40 40]  New second array: [ 41 41 41 ... 122 128 132] |

* Plot dữ liệu:



## **Bài tập 2**: Xây dựng một model đơn giản để predict một phương trình đường thẳng

Ta có phương trình gốc ban đầu là: y = 3x + 5

Các bước cần thực hiện:

+ Sinh dữ liệu huấn luyện.

+ Sử dụng tensorflow để hỗ trợ xây model.

+ Đánh giá model.

Thực hiện bước sinh dữ liệu và khởi tạo các node của model:

|  |
| --- |
| data\_x, data\_y = generate\_dataset(100)  x, y, y\_pred, loss = linear\_regression()  epochs = 50 # train times |

Ở đây ta sẽ train model này 50 lần, nếu kết quả bị sai lệch quá nhiều thì ta sẽ thực hiện đồng thời tăng size huấn luyện và số lần train.

Ta có thêm hai biến mới là *y\_ pred* và *loss*, *y­\_ pred* là giá trị y là model dự đoán, ta sẽ dựa theo kết quả thu được mà đánh giá model bằng hàm *loss*, là bằng , hàm *loss* của chúng ta càng nhỏ có nghĩa là kết quả dự đoán của chúng ta càng tiệm cận tới kết quả thực. Vì thế ta cần tối thiểu hàm *loss.*

|  |
| --- |
| optimizer = tf.train.GradientDescentOptimizer(0.1)  train\_op = optimizer.**minimize**(loss) |

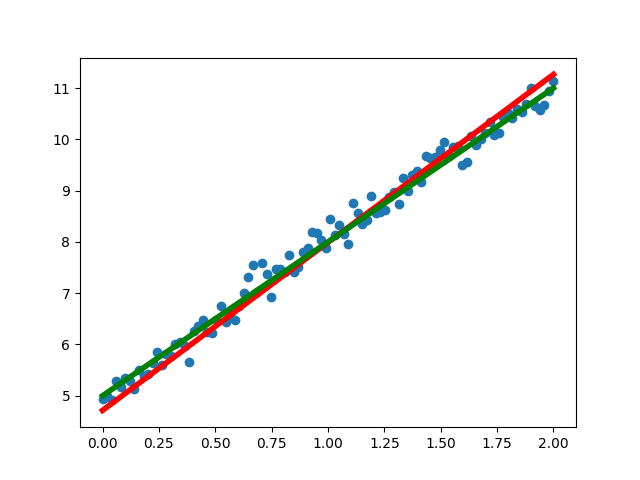
Sau mỗi đợt train ta có thể thấy được hàm *loss* của chúng ta đang tiệm cận về không.

|  |
| --- |
| 0 Loss val: 12.274088  1 Loss val: 4.302384  2 Loss val: 1.7507209  3 Loss val: 0.91924936  4 Loss val: 0.63463134  5 Loss val: 0.5247086  …  45 Loss val: 0.06886044  46 Loss val: 0.06664056  47 Loss val: 0.06455681  48 Loss val: 0.06260076  49 Loss val: 0.060764626 |

Ta đáng giá lại model:

|  |
| --- |
| Input value: [1. 7. 3.5 3.141592]  Expected result: [ 8. 26. 15.5 14.424776]  Predict result: [ 7.932784 27.683392 16.162205 14.982409] |

Plot dữ liệu, với đường đỏ là đường do model phán đoán và đường xanh lá cây là đường thẳng của phương trình tuyến tính ban đầu.



# **BÀI THỰC HÀNH SỐ 2**

## **Bài tập 1**: Xây dựng một phân lớp dựa trên histogram, với tập dữ liệu cho trước.

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)]

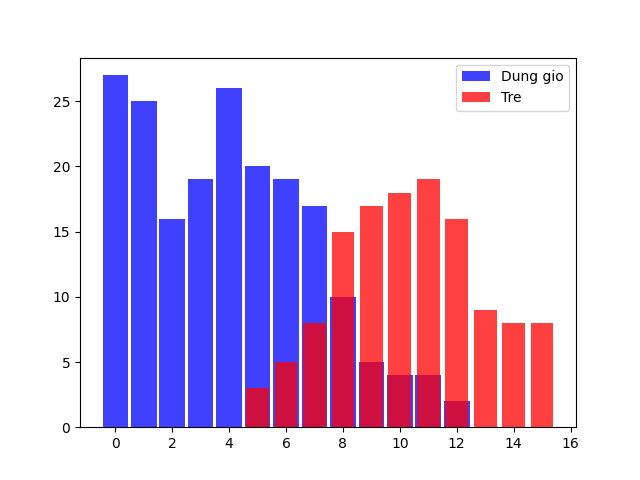
Ta sử dụng lại biệt hàm ở Bài tập 1 – Tuần 1, ta tìm được giá trị biên để phân lớp là vào 6:37.

|  |
| --- |
| Found threadhold at: 7  Number of error is: 41 |

Nếu ta rời nhà vào các giờ *6:34, 6:35, 6:36, 6:37* và *6:38* thì tỉ lệ trễ học của chúng ta lần lượt là:

|  |
| --- |
| Probability of going late to school when start at 6:34 is 0 %  Probability of going late to school when start at 6:35 is 13 %  Probability of going late to school when start at 6:36 is 21 %  Probability of going late to school when start at 6:37 is 32 %  Probability of going late to school when start at 6:38 is 60 % |

Plot dữ liệu:



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

Ta sẽ cài đặt bài toán như sau:

* Đọc và chia dữ liệu thành các đặc trưng cụ thể bằng thư viện pandas.
* Xây dựng một lớp đặc trưng cho mỗi đặc trưng riêng biệt.
* Xây dựng lớp bayes để tính xác suất nằm trong một lớp khi biết được giá trị thuộc tính.
* Xây dựng một lớp dùng để tính toán xác suất bayes cao nhất khi xét các giá trị đặc trưng.

Cài đặt lớp đặc trưng:

|  |
| --- |
| class Feature:  def \_\_init\_\_(self, data, name=None, bin\_w=None):  self.name = name  self.bin\_w = bin\_w  if bin\_w:  self.min, self.max = min(data), max(data)  bins = np.arange((self.min // bin\_w) \* bin\_w,  (self.max // bin\_w) \* bin\_w,  bin\_w)  self.freq\_dict = dict(zip(\*np.histogram(data, bins)[::-1]))  else:  self.freq\_dict = Counter(data)  self.freq\_sum = sum(self.freq\_dict.values())  def get\_freq(self, value):  # P(wi)  if self.bin\_w:  value = (value // self.bin\_w) \* self.bin\_w  return self.freq\_dict.get(value, 0) |

Cài đặt lớp Bayes:

|  |
| --- |
| class NaiveBayes:  def \_\_init\_\_(self, name, \*features):  self.features = features  self.name = name  def prob\_value\_giving\_feature(self, \*feature\_value):  '''  can be know as giving a feature how probability that value is in the giving feature  P(x | wi)  '''  result = 1  for f, fv in zip(self.features, feature\_value):  if f.freq\_sum == 0:  return 0  else:  result \*= f.get\_freq(fv) / f.freq\_sum  return result |

Cài đặt biệt hàm phân lớp

|  |
| --- |
| class Classifier:  def \_\_init\_\_(self, \*nbClass):  self.nbClass = nbClass  def prob\_feature\_giving\_value(self, data, best\_only=True):  # P(wi | x)  prob\_list = list()  for nbc in self.nbClass:  prob\_list.append( (nbc.prob\_value\_giving\_feature(\*data), nbc.name) )  prob\_sum = sum( [v[0] for v in prob\_list] )  if prob\_sum == 0:  # In case of not found  # Distribute evenly between every class  # This make sure the final result <= 1  number\_class = len(self.nbClass)  prob\_list = [ (1/number\_class, name) for \_, name in prob\_list ]  else:  prob\_list = [ (value/prob\_sum, name) for value, name in prob\_list ]  if best\_only:  return max(prob\_list)  else:  return prob\_list |

Kết quả thu được:

|  |
| --- |
| Sử dụng đặc trưng chiều cao:  data test: [(140,), (200,), (153,), (188,), (159,), (160,), (180,), (150,), (170,)]  (140,) (0.5, 'male')  (200,) (1.0, 'male')  (153,) (1.0, 'female')  (188,) (1.0, 'male')  (159,) (0.875, 'female')  (160,) (0.6153846153846154, 'female')  (180,) (0.5555555555555556, 'male')  (150,) (1.0, 'female')  (170,) (0.5384615384615385, 'female')  Sử dụng đặc trưng tên:  data test: [('Edgar',), ('Benjamin',), ('Fred',), ('Albert',), ('Laura',), ('Maria',), ('Paula',), ('Sharon',), ('Jessie',)]  ('Edgar',) (0.5, 'male')  ('Benjamin',) (1.0, 'male')  ('Fred',) (1.0, 'male')  ('Albert',) (1.0, 'male')  ('Laura',) (1.0, 'female')  ('Maria',) (1.0, 'female')  ('Paula',) (1.0, 'female')  ('Sharon',) (1.0, 'female')  ('Jessie',) (0.6666666666666667, 'female')  Sử dụng đặc trưng tên + chiều cao:  data test: [('Maria', 140), ('Anthony', 200), ('Anthony', 153), ('Jessie', 188), ('Jessie', 159), ('Jessie', 160)]  ('Maria', 140) (0.5, 'male')  ('Anthony', 200) (1.0, 'male')  ('Anthony', 153) (0.5, 'male')  ('Jessie', 188) (1.0, 'male')  ('Jessie', 159) (0.9333333333333333, 'female')  ('Jessie', 160) (0.761904761904762, 'female') |

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

1. [1 2 4 6 9 10 20 7]

Để tìm được mean của một vector đặc trưng, ta chỉ cần cộng tổng tất cả các phần tử có trong vector đặc trưng rồi đem kết quả chia cho số lượng phần tử có trong vector đặc trưng đó. Trong một số trường hợp vector đặc trưng là một ma trận 2 chiều, ta sẽ thực hiện lần lượt, lấy từng hàng ra tính trung bình, kết quả cuối cùng ta thu được là một mảng trung bình.

Cài đặt hàm:

|  |
| --- |
| def mean(arr):  if len(arr.shape) != 1:  return np.array([mean(np.asarray(i)) for i in zip(\*arr)])  return sum(arr)/len(arr) |

Còn variance, ta sẽ áp dụng công thức sau:



Với *X* là các giá trị trong vector đặc trưng của chúng ta, là trung bình, và *N* là số lượng phần tử có trong vector đặc trưng.

Cài đặt hàm:

|  |
| --- |
| def var(arr):  if len(arr.shape) != 1:  return np.array([var(np.asarray(i)) for i in zip(\*arr)])  m = mean(arr)  return sum((xi - m)\*\*2 for xi in arr)/len(arr) |

Sử dụng hai hàm trên sẽ cho ta kết quả như sau (đã đối chiếu kết quả với hàm chuẩn của thư viện numpy):

|  |
| --- |
| mean(a): 7.375 True  var(a) : 31.484375 True |

1. [0, 2, 4, 6, 8,…100]

Sử dụng lại những hàm ở trên đã khai báo ta thu được kết quả như sau:

|  |
| --- |
| mean(b): 50.0 True  var(b) : 866.6666666666666 True |

1. [1, 3, 25, … 9801]

Kết quả thu được:

|  |
| --- |
| mean(c): 3333.0 True  var(c) : 8884444.8 True |

1. [[2, 4], [3, 7], [4, 6], [5, 5], [2, 3]]

Tương tự như ở trên

|  |
| --- |
| mean(d): [3.2 5. ] [ True True]  var(d) : [1.36 2. ] [ True True] |

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

X = [2 3 6 3 7 8], Y = [5 7 9 6 7 8]

Để tính được covariance matrix của các vector đặc trưng ta cần tính

* + 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:

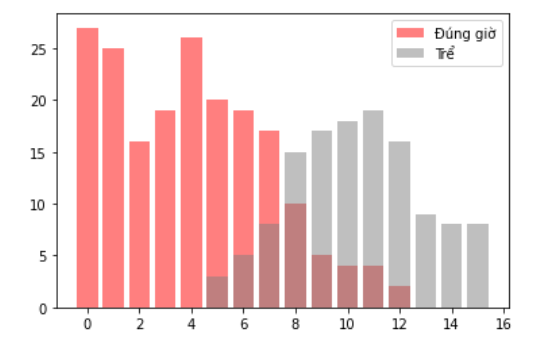
|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from collections import Counter |

*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")  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 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(2,101,2))  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:  51.0  Variance của c là:  833.0 |

|  |
| --- |
| 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]  ] |

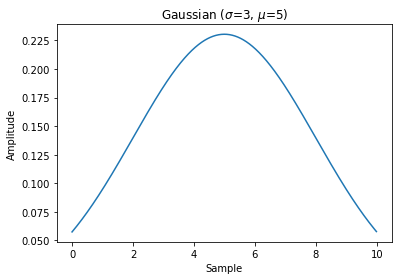
|  |
| --- |
| matrix\_2 = np.asarray(**matrix\_2**) |

|  |
| --- |
| 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.

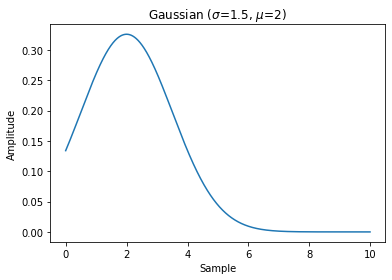
|  |
| --- |
| def gauss(x, mean, variance):    return (1/(math.sqrt(2\*math.pi\*variance)))\*math.exp((-1/2)\*((x-mean)/variance)\*\*2) |

|  |
| --- |
| 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.

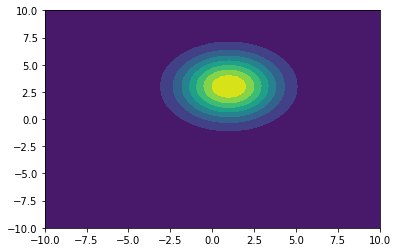
|  |
| --- |
| mean = 2  variance = 1.5  list\_x = np.arange(0, 10, 0.01)  list\_y = []  for x in list\_x:    list\_y.append(gauss(x, mean, variance))  plt.plot(list\_x, list\_y)  plt.title(r"Gaussian ($\sigma$="+ str(variance) + ", $\mu$=" + str(mean) + ")")  plt.ylabel("Amplitude")  plt.xlabel("Sample")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

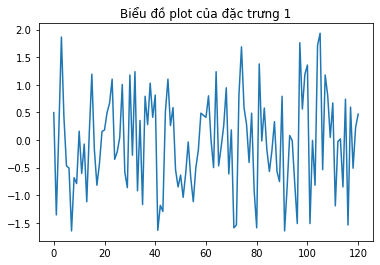
|  |
| --- |
| datContent = [i.strip().split() for i in open("twoclass.dat", "r").readlines()]  print("Số mẫu là: " + str(len(datContent)))  print("Số đặc trưng của mỗi class là: " + str(len(datContent[0][:-2])))  for i in range(len(datContent[0][:-2])):    print("Trung bình của đặc trưng " + str(i+1) + " 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 |

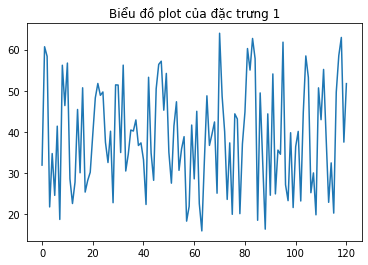
|  |
| --- |
| # Đặc trưng 1  fea\_1\_test = find\_thresh(feature\_1\_train, class\_train)  print("Ngưỡng tìm được từ tập train đặc trưng 1 là: " + str(fea\_1\_test[0]))  print("Số lỗi tìm được từ tập train đặc trưng 1 là: " + str(fea\_1\_test[1]))  print("Số lỗi tìm được từ tập test đặc trưng 1 là: "  + 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].

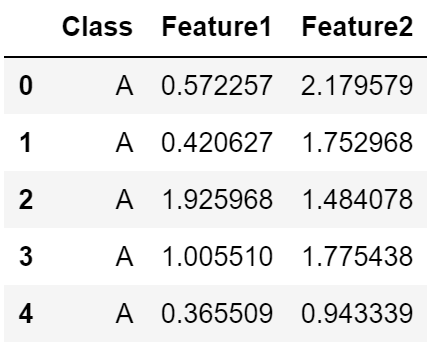
* *Bước 1:* Import thư viện:

|  |
| --- |
| 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*.

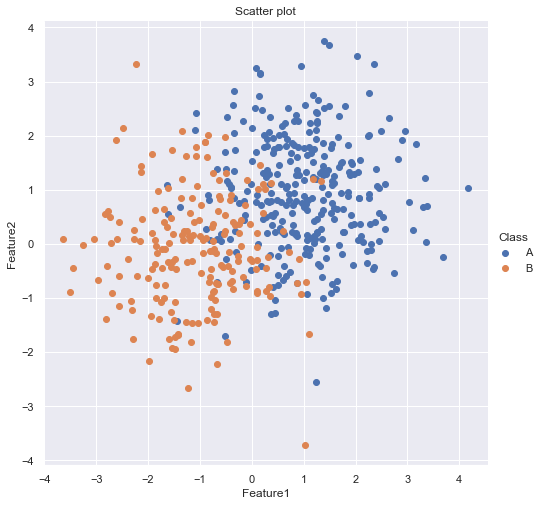
|  |
| --- |
| columnsName = ['Feature1','Feature2']  classA = pd.**DataFrame**(loadmat("classA.mat")['classA'],columns =  columnsName)  classB = pd.**DataFrame**(loadmat("classB.mat")['classB'],columns =  columnsName) |

|  |
| --- |
| #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**) |

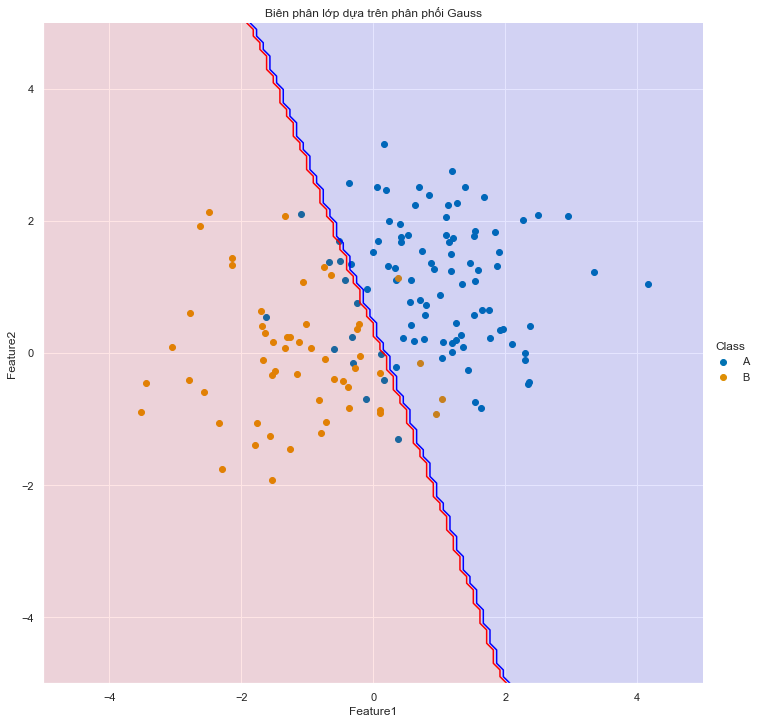
• Đánh giá

- 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.

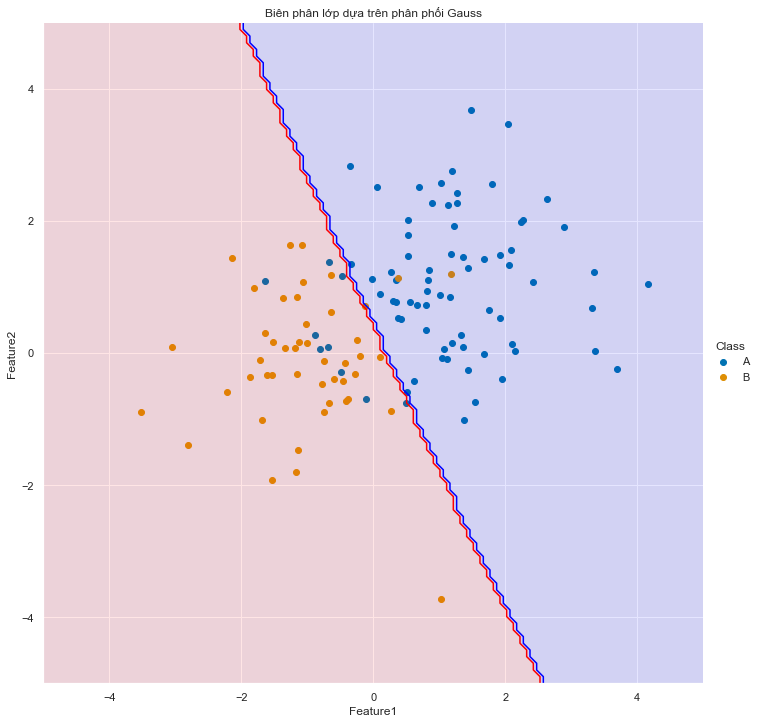
|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(**train\_set**.groupby('Class').mean().values,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(train\_set.groupby('Class').cov().values,[2])  pi\_list = train\_set.iloc[:,0].value\_counts().values / 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.

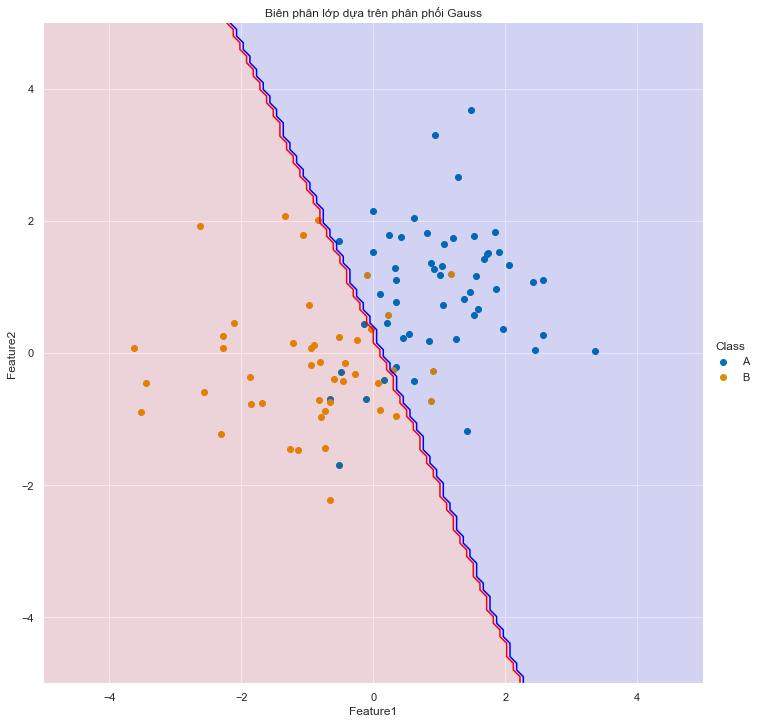
|  |
| --- |
| #Estimating the parameters  mu\_list = np.split(train\_set.groupby('Class').mean().values,[1])  # cov\_list = np.split(np.array([1,0,0,1]\*2).reshape(-1,2),[2])  cov\_list = np.split(**train\_set**.groupby('Class').cov().values,[2])  pi\_list = **train\_set**.iloc[:,0].value\_counts().values / 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:***

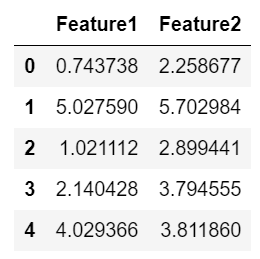
- Import thư viện:

|  |
| --- |
| 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:

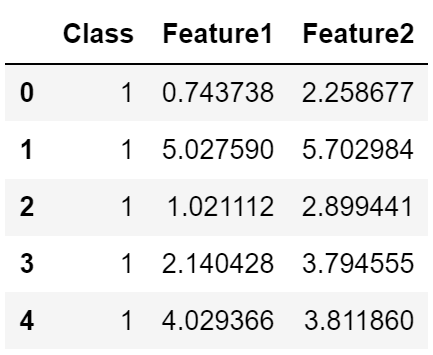
|  |
| --- |
| #Load data  columnsName = ['Feature1','Feature2']  class1\_train = pd.DataFrame(loadmat(**'class1\_train**.mat')[**'class1\_train'**],  columns = columnsName)  class2\_train = pd.DataFrame(scipy.io.loadmat('class2\_train.mat')['class2\_train'],  columns = columnsName)  class1\_test = pd.**DataFrame**(scipy.io.loadmat('class1\_test.mat')['class1\_t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

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



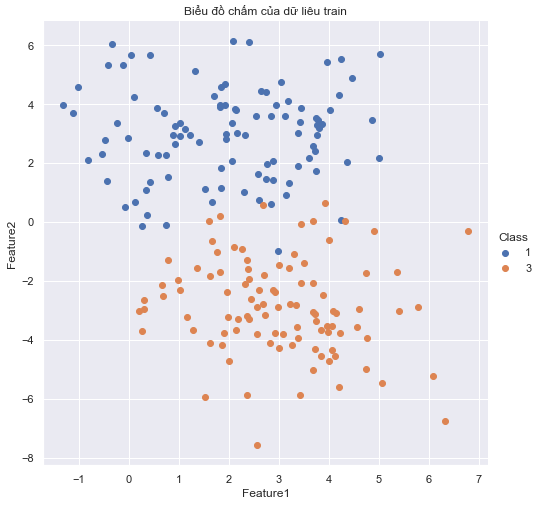
|  |
| --- |
| #Merge data  trainMerge = pd.concat([class1\_train,class2\_train], keys=['1', '3']).reset\_index().drop('level\_1',  axis=1).rename(columns = {'level\_0': 'Class'})  testMerge = pd.concat([class1\_test,class2\_test], keys=['1', '2']).reset\_index().drop('level\_1',  axis=1).rename(columns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

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

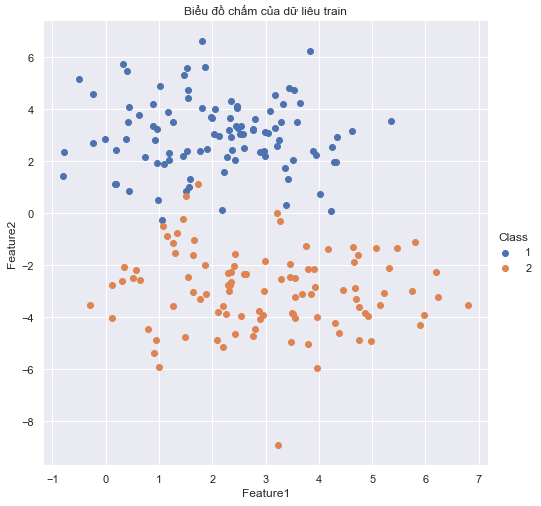


- Plot dữ liệu.

|  |
| --- |
| #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).

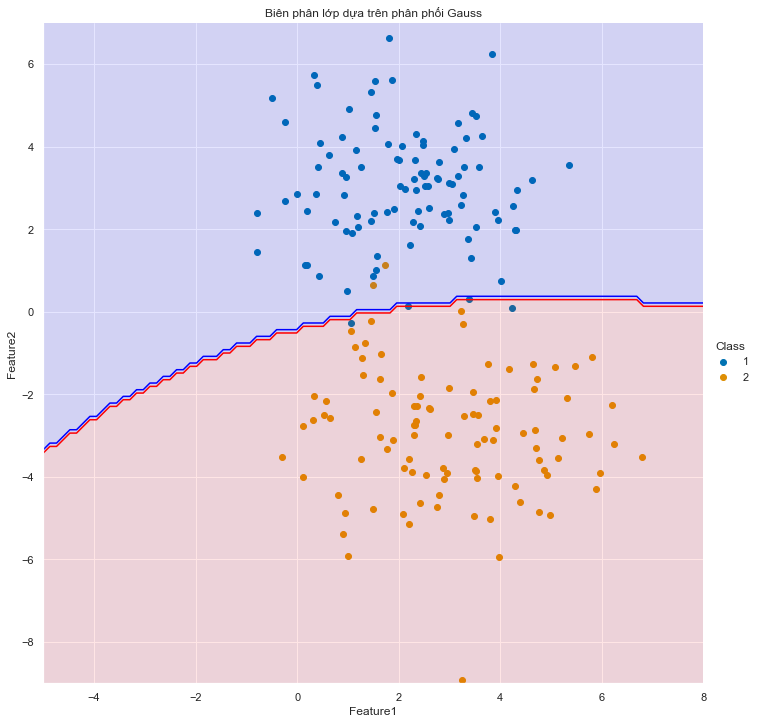
|  |
| --- |
| #Build discriminant functions  def DF(X,mu\_list,cov\_list,pi\_list):  scores\_list = []  classes = len(mu\_list)  for p in range(classes):  Wi = (-1/2)\*np.linalg.inv(cov\_list[p])  wi = np.linalg.inv(cov\_list[p])@mu\_list[p][0]  wi0 = (-1/2)\*np.transpose(mu\_list[p][0])@np.linalg.inv(cov\_list[p])@mu\_list[p][0]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

**Giải:**

|  |
| --- |
| import numpy as np  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 |

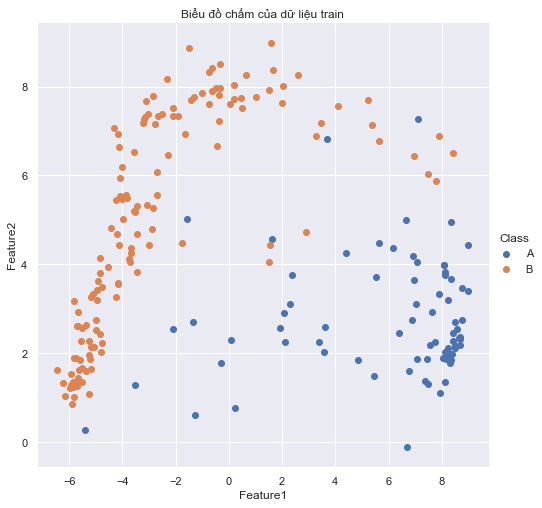
* *Bước 1:* Import thư viện cần sử dụng
* *Bước 2:* Load 2 file *classA.txt* và *classB.txt*.

|  |
| --- |
| #Load data  columnsName = ['Feature1','Feature2']  f = open("classA.txt", "r")  classA = pd.DataFrame(np.array(f.read().split()).astype('float64')  .reshape(-1,2),columns = columnsName)  f = open("classB.txt", "r")  classB = pd.DataFrame(np.array(f.read().split()).astype('float64')  .reshape(-1,2),columns = columnsName) |

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

* *Bước 3*: Biểu diễn đồ thị bằng biểu đồ 2 chiều nhằm trực quan hóa các điểm dữ liệu.

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



* *Bước 4:* Phân chia tập dữ liệu thành 2 tập con training (70%) và testing (30%).

|  |
| --- |
| #Split train and test  train\_set, test\_set = train\_test\_split(classMerge, train\_size=0.7)  print("Số lượng của tập train là :",len(train\_set.index))  print("Số lượng của tập train là :",len(test\_set.index)) |
| Số lượng của tập train là : 151  Số lượng của tập train là : 65 |

* *Bước 5:* 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.

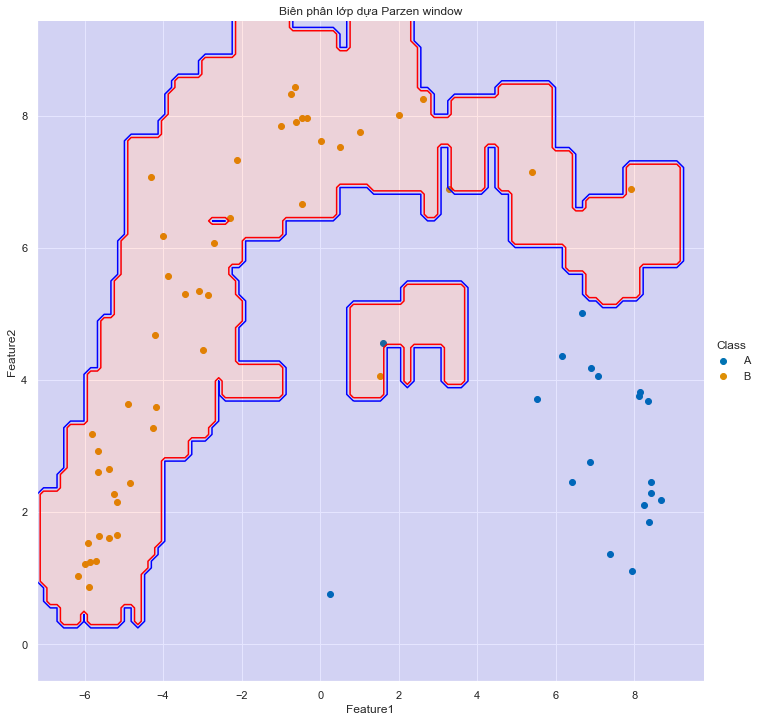
|  |
| --- |
| #Build classifier functions  h = 1.5  def **phi**(x):  if ((np.**absolute**(x)>1/2).any()):  return 0  else:  return 1  def **PW**(X,data\_train):  scores\_list = []  for **p** in **train\_set**.groupby('Class'):  **score** = 0  for x in **p**[1][[**'Feature1'**,**'Feature2'**]].to\_numpy():  **score**+=phi((X-x.reshape(-1,1))/h)  **scores\_list**.append(**score**)  return np.argmax(**scores\_list**) |

* *Bước 6:* Sử dụng tập testing để đánh giá hiệu quả của bộ classifier.

|  |
| --- |
| #Confusion matrix  **prediction** = ["A" if **PW**(np.array([x,y]).reshape(-1,1),**train\_set**)==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 18 1  Class B 1 45 |

* *Bước 7:* Vẽ biểu đồ thể hiện biên phân lớp của dữ liệu.

|  |
| --- |
| #Plot with boundary contours  N = 100  maxv = **classMerge**[["Feature1","Feature2"]].max().values  minv = **classMerge**[["Feature1","Feature2"]].min().values  len = maxv - minv  X = np.**linspace**(minv[0]-len[0]\*0.05, maxv[0]+len[0]\*0.05, N)  Y = np.**linspace**(minv[1]-len[1]\*0.05, maxv[1]+len[1]\*0.05, 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( [**PW**(np.array([**xx,yy**]).reshape(-1,1),**train\_set**)  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 Parzen window')  plt.**show**() |



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

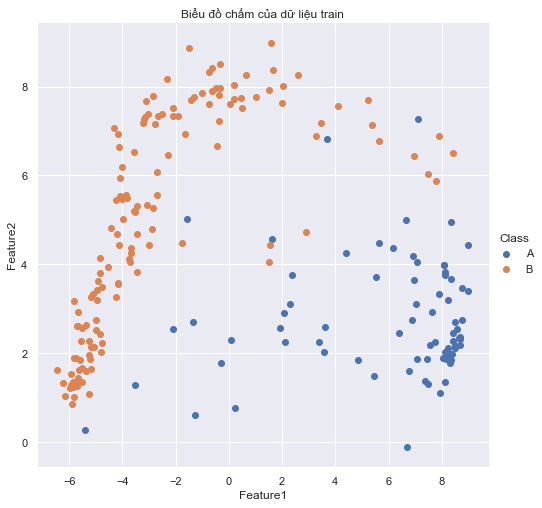


|  |
| --- |
| import **numpy** as np  import **pandas** as pd  from **matplotlib** import pyplot as plt  import **matplotlib**.colors as colors  import **seaborn** as sns  import **itertools**  from **scipy**.stats import **norm**  import **scipy**.stats  from **sklearn**.naive\_bayes import **GaussianNB**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

|  |
| --- |
| #Load data  columnsName = [**'Feature1'**,**'Feature2'**]  f = open("**classA**.**txt**", "r")  classA = pd.**DataFrame**(np.array(f.read().split()).astype('float64').reshape(-1,2),columns = columnsName)  f = open("**classB**.**txt**", "r")  classB = pd.**DataFrame**(np.array(f.read().split()).astype('float64').reshape(-1,2),columns = columnsName) |

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

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

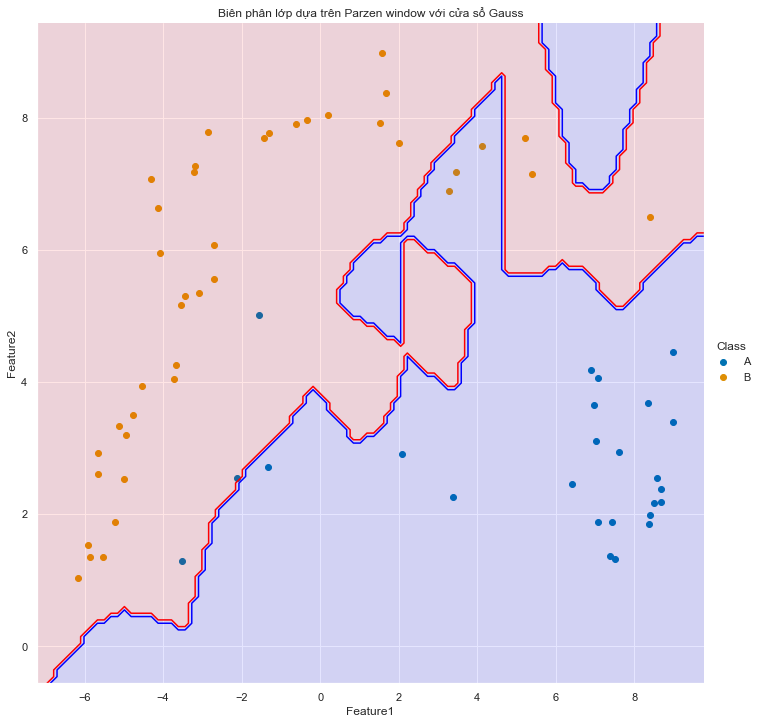


|  |
| --- |
| #Split train and test  **train**\_**set,** **test**\_**set** = **train**\_**test**\_**split**(classMerge, train\_size=0.7)  print("Số lượng của tập train là :",len(**train\_set**.index))  print("Số lượng của tập train là :",len(**test\_set**.index)) |
| Số lượng của tập train là : 151  Số lượng của tập train là : 65 |

|  |
| --- |
| #Build classifier functions  h = 0.5  def **phi**(x):  return np.exp(-np.transpose(**x**)@**x**)/2  def **PW**(X,data\_train):  scores\_list = []  for **p** in **train\_set**.groupby(**'Class'**):  **score** = 0  for x in **p**[1][[**'Feature1'**,**'Feature2'**]].to\_numpy():  **score**+=**phi**((X-x.reshape(-1,1))/h)  **scores**\_**list**.append(**score**)  return np.argmax(**scores**\_**list**) |

|  |
| --- |
| #Confusion matrix  **prediction** = [**"A**" if **PW**(np.array([x,y]).reshape(-1,1),**train\_set**)==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 22 3  Class B 3 37 |

|  |
| --- |
| #Plot with boundary contours  N = 100  **maxv** = classMerge[["Feature1","Feature2"]].max().values  **minv** = classMerge[["Feature1","Feature2"]].min().values  len = maxv - minv  X = np.linspace(minv[0]-len[0]\*0.05, **maxv**[0]+len[0]\*0.05, N)  Y = np.linspace(minv[1]-len[1]\*0.05, **maxv**[1]+len[1]\*0.05, 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( [**PW**(np.array([**xx,yy**]).reshape(-1,1),**train\_set**)  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 Parzen window với cửa sổ Gauss')  plt.**show**() |



**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.