**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**

[**Mục lục** 3](#_Toc57164844)

[**BÀI THỰC HÀNH SỐ 1** 4](#_Toc57164845)

[**Bài tập 1**: Tìm biên phân lớp cho các tập dữ liệu 4](#_Toc57164846)

[**Bài tập 2**: Xây dựng một model đơn giản để predict một phương trình đường thẳng 7](#_Toc57164847)

[**BÀI THỰC HÀNH SỐ 2** 9](#_Toc57164848)

[**Bài tập 1**: Xây dựng một phân lớp dựa trên histogram, với tập dữ liệu cho trước. 9](#_Toc57164849)

[**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](#_Toc57164850)

[**Bài tập 3**:Tính mean và variance của các vector đặc trưng sau. 12](#_Toc57164851)

[**Bài tập 4**:Tính covariance matrix của các vector đặc trưng sau. 13](#_Toc57164852)

[**Bài tập 5**: Tạo hàm mật độ phân bố Gauss với mean là 5, variance là 3. Plot hàm kết quả. 14](#_Toc57164853)

[**Bài tập 6**: Tạo hàm mật độ phân bố Gauss với mean là 2, variance là 1.5. Plot hàm kết quả. 15](#_Toc57164854)

[**Bài tập 7**: Tạo hàm mật độ phân bố Gauss 2 chiều với mean là [1 3], variance là [2, 2]. Plot hàm trên lưới [-10 10] x [-10 10] và tính khoảng cách Mahalanobis đối với các mẫu [0 0], [3 4], [1 2] 16](#_Toc57164855)

[**Bài tập 9**: Xây dựng bộ classifier sử dụng một đặc trưng có sẵn 17](#_Toc57164856)

[**BÀI THỰC HÀNH SỐ 3** 23](#_Toc57164857)

[**Bài tập 1:** Xây dựng bộ phân lớp và trực quan hóa biên phân lớp đối với tập dữ liệu iris. 23](#_Toc57164858)

[**Bài tập 2:** Cho 2 tập dữ liệu class A và class B. 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] 24](#_Toc57164859)

[**Bài tập 3:** Xây dựng bộ classifier với 2 lớp, 2 đặc trưng. Giả sử tập dữ liệu có dạng phân bố gauss. 30](#_Toc57164860)

[**Bài thực hành số 4** 33](#_Toc57164861)

[**Bài tập 1:** Xây dựng bộ classifier dựa trên Parzen windows 33](#_Toc57164862)

[**Bài tập 2:** Xây dựng bộ classifier sử dụng hàm cửa sổ Gauss 36](#_Toc57164863)

# **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.

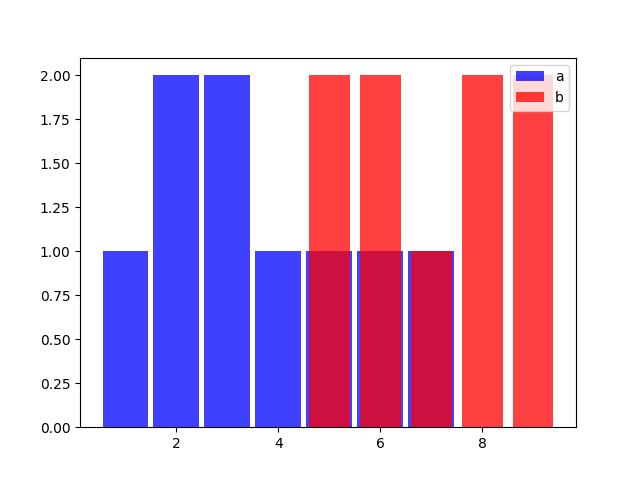
* Cài đặt hàm:

|  |
| --- |
| 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

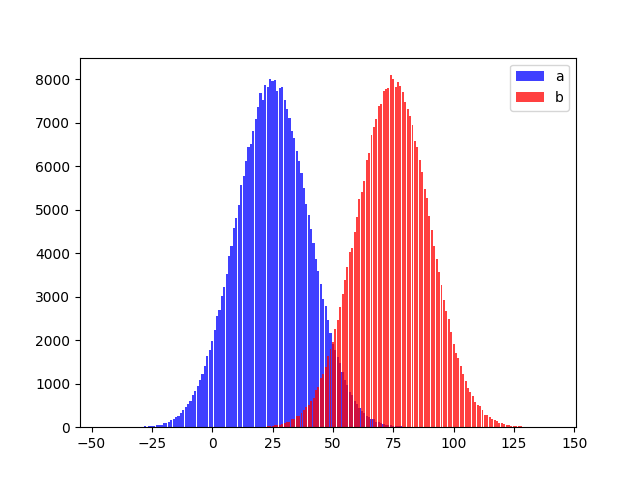
* Lần này ta sẽ sử dụng hàm phân bố chuẩn để sinh tập dữ liệu ngẫu nhiên

|  |
| --- |
| 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

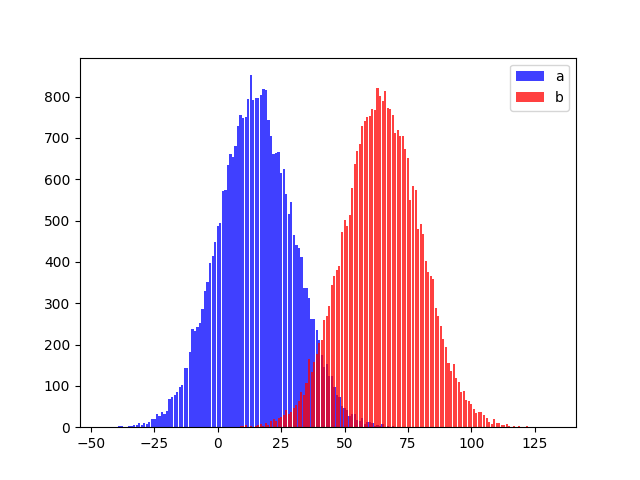
+ Sinh dữ liệu như bước b, và lưu xuống file.

+ 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) |

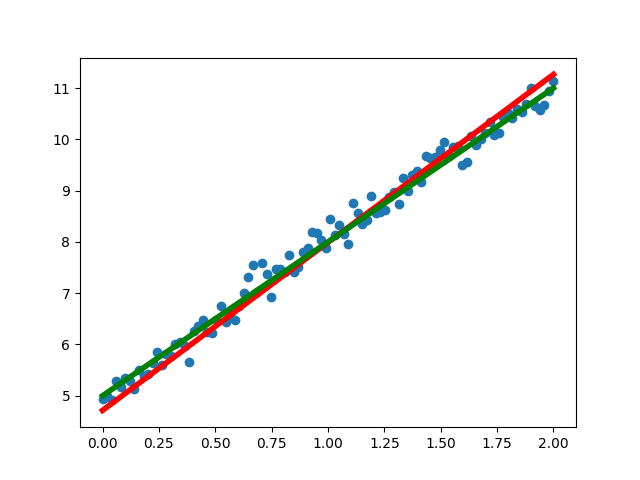
Sau mỗi đợt train ta có thể thấy được hàm *loss* của chúng ta đang tiệm cận về không.

|  |
| --- |
| 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)]

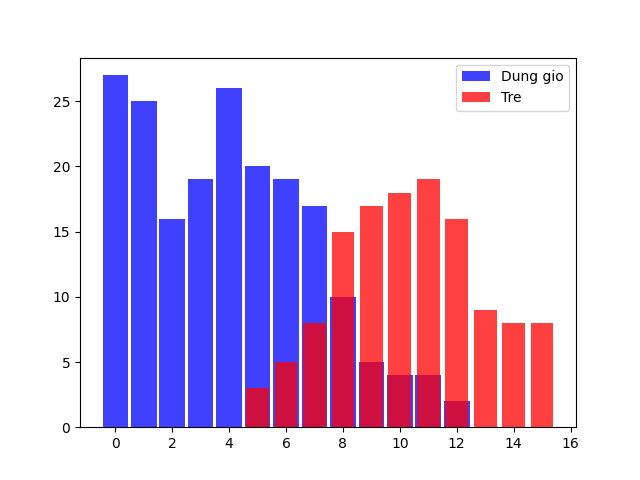
Ta sử dụng lại biệt hàm ở Bài tập 1 – Tuần 1, ta tìm được giá trị biên để phân lớp là vào 6:37.

|  |
| --- |
| 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]

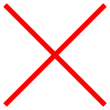
Để tìm được mean của một vector đặc trưng, ta chỉ cần cộng tổng tất cả các phần tử có trong vector đặc trưng rồi đem kết quả chia cho số lượng phần tử có trong vector đặc trưng đó. Trong một số trường hợp vector đặc trưng là một ma trận 2 chiều, ta sẽ thực hiện lần lượt, lấy từng hàng ra tính trung bình, 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 xây dựng một hàm tính mean, vậy nên ta sẽ sử dụng lại hàm *mean* của bài tập số 3.

Các bước để tính covariance matrix.

- Tính *mean* của từng hàng trong ma trận đặc trưng

|  |  |
| --- | --- |
| |  | | --- | | for r in range(row):  mean\_ar.append(mean(dataset[r])) | |

- Ta chạy tạo ma trận size RxR với R là độ dài, ta gọi giá trị ở cell dòng i cột j là , ta có công thức sau: với:

+: Giá trị trong vector đặc trưng ở hàng i\_1, cột j.

+ : Giá trị trong vector đặc trưng ở hàng i\_2, cột j.

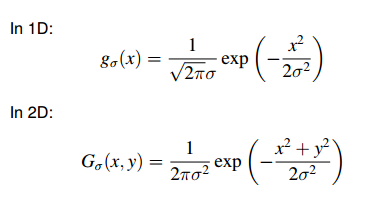
+ : Số lượng cột ở trong vector đặc trưng

- Kết quả chạy thử:

|  |
| --- |
| Correct output:  [[6.16666667 2.6 ]  [2.6 2. ]]  Implementation output:  [[6.16666667 2.6 ]  [2.6 2. ]] |

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

Công thức tính phân bố Gauss như sau:



Thực hiện cái đặt thuật toán trên trong python:

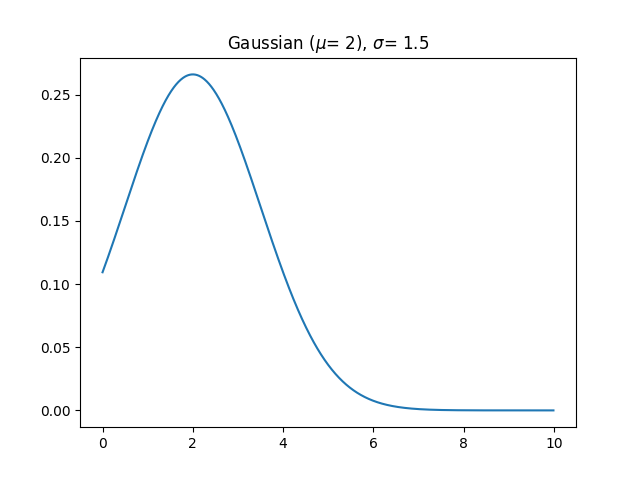
|  |
| --- |
| def gauss(x, mean, var):  return (1 / (var \* math.sqrt(2\*math.pi)) \* math.exp(-0.5 \* ((x-mean)/var)\*\*2)) |

Plot hàm kết quả:

|  |
| --- |
|  |

## **Bài tập 6**: Tạo hàm mật độ phân bố Gauss với mean là 2, variance là 1.5. Plot hàm kết quả.

Sử dụng hàm Gauss ở bài trước ta plot được hàm kết quả sau:

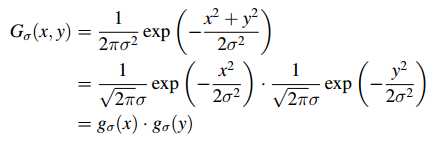


Nhận xét:

- Phân bố gauss ở bài 6 bị lệch sang trái so với phân bố gauss ở bài 5 do mean của bài 6 bé hơn bài 5, đồng thời độ dốc của phân bố cũng dốc hơn so với bài 5 do var của bài 6 bé hơn bài 5.

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

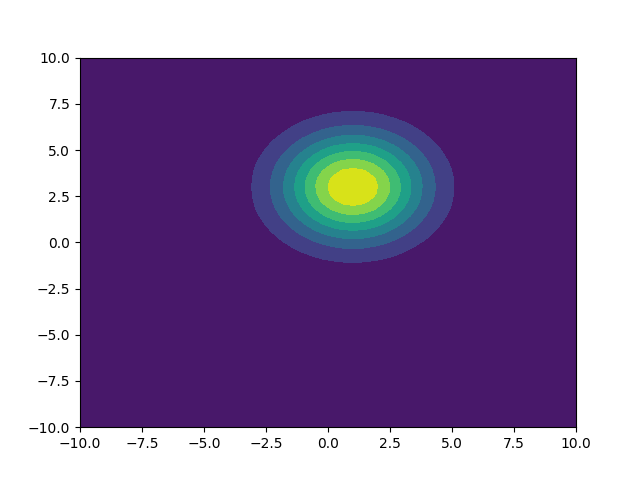
Công thức tính gauss 2 chiều như sau:



Dựa trên hàm gauss ở bài trước ta xây dựng một hàm gauss hai chiều như sau:

|  |  |
| --- | --- |
| |  | | --- | | def gauss2d(x, y, mean, var):  return gauss(x, mean[0], var[0]) \* gauss(y, mean[1], var[1]) | |

Plot hàm kết quả:



Công thức tính khoảng cách:



Cài đặt hàm tính:

|  |  |
| --- | --- |
| |  | | --- | | def mahalanobis\_dist(x, mean, cov):  return np.sqrt(np.transpose(x - mean)@np.linalg.inv(cov)@(x - mean)) | |

## **Bài tập 9**: Xây dựng bộ classifier sử dụng một đặc trưng có sẵn

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

Số mẫu đặc trưng, mean và covariance của class1:

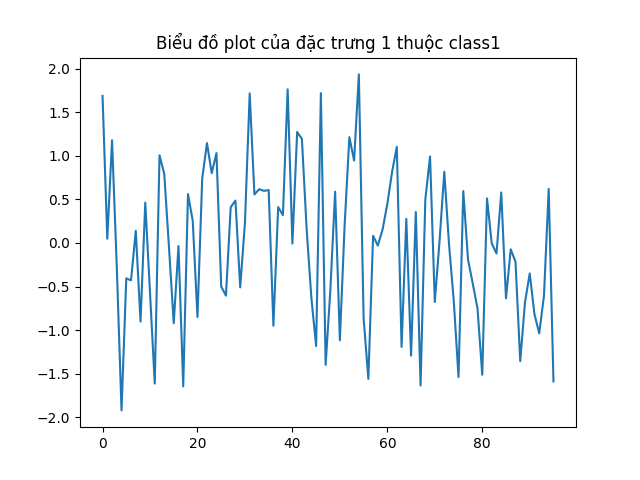
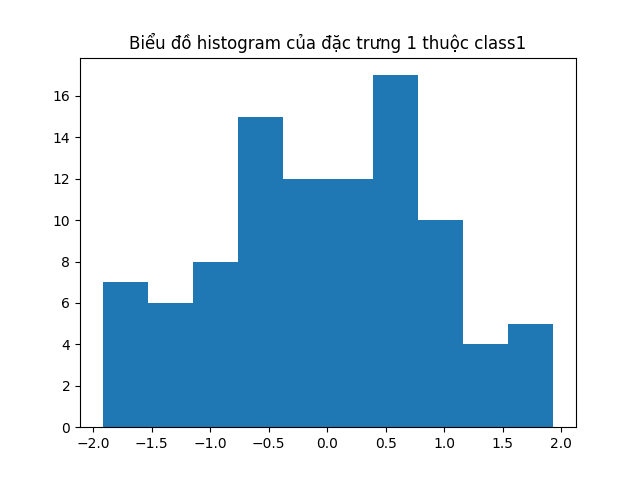
|  |
| --- |
| \*CLASS1:  Feature1:  +Mean: 0.06218687500000001  +Var : 0.8717452837660599  Feature2:  +Mean: 0.05867279999999998  +Var : 1.0685284890658253  Feature3:  +Mean: 2.96126355  +Var : 3.039673866930216  Feature4:  +Mean: 0.08500643333333335  +Var : 0.9921645930659284  Covariance giữa Feature1 với Feature2 là:  [[0.87174528 0.04184359]  [0.04184359 1.06852849]]  Covariance giữa Feature1 với Feature3 là:  [[ 0.87174528 -0.0425842 ]  [-0.0425842 3.03967387]]  Covariance giữa Feature1 với Feature4 là:  [[ 0.87174528 -0.07247153]  [-0.07247153 0.99216459]]  Covariance giữa Feature2 với Feature3 là:  [[1.06852849 0.05295708]  [0.05295708 3.03967387]]  Covariance giữa Feature2 với Feature4 là:  [[ 1.06852849 -0.07206057]  [-0.07206057 0.99216459]]  Covariance giữa Feature3 với Feature4 là:  [[ 3.03967387 -0.03565163]  [-0.03565163 0.99216459]] |

Số mẫu đặc trưng, mean và covariance của class2:

|  |
| --- |
| \*CLASS2:  Feature1:  +Mean: 0.06021218181818183  +Var : 0.8649525687273167  Feature2:  +Mean: 0.0535757603305785  +Var : 1.0601799186312004  Feature3:  +Mean: -0.03323852066115702  +Var : 1.019089111137485  Feature4:  +Mean: -2.8515889752066115  +Var : 3.4766801685859914  Covariance giữa Feature1 với Feature2 là:  [[0.86495257 0.04081435]  [0.04081435 1.06017992]]  Covariance giữa Feature1 với Feature3 là:  [[ 0.86495257 -0.0217829 ]  [-0.0217829 1.01908911]]  Covariance giữa Feature1 với Feature4 là:  [[ 0.86495257 -0.12831742]  [-0.12831742 3.47668017]]  Covariance giữa Feature2 với Feature3 là:  [[1.06017992 0.02422017]  [0.02422017 1.01908911]]  Covariance giữa Feature2 với Feature4 là:  [[ 1.06017992 -0.12980585]  [-0.12980585 3.47668017]]  Covariance giữa Feature3 với Feature4 là:  [[1.01908911e+00 5.03602239e-05]  [5.03602239e-05 3.47668017e+00]] |

Ta sẽ chia tập dữ liệu thành 2 phần, với 50% dữ liệu train và 50% dữ liệu test

Plot tập dữ liệu, sét đặc trưng Feature thứ 1



Sử dụng hàm tìm biên phân lớp ở tuần 1, ta cho ra kết quả:

|  |
| --- |
| Biệt số tìm được là: -0.665853  Tổng số lỗi trong tập train là: 91  Tổng số lỗi trong tập test là: 19  misclassification class1: 16  misclassification class2: 3 |

# **BÀI THỰC HÀNH SỐ 3**

## **Bài tập 1:** Xây dựng bộ phân lớp và trực quan hóa biên phân lớp đối với tập dữ liệu iris.

Plot tập dữ liệu:

|  |  |
| --- | --- |
| Biệt hàm:   |  | | --- | | def predict\_NB\_gaussian\_class(X, mu\_list, std\_list, pi\_list):  scores\_list = list()  classes = len(mu\_list)  for p in range(classes):  # score = g(x\_1) \* g(x\_2)  score = (scipy.stats.norm.pdf(x = X[0], loc = mu\_list[p][0], scale = std\_list[p][0] ) \* scipy.stats.norm.pdf(x = X[1], loc = mu\_list[p][1], scale = std\_list[p][1] ) \* pi\_list[p])  scores\_list.append(score)  return np.argmax(scores\_list) | |

Plot dữ liệu cùng biên phân lớp:

## **Bài tập 2:** Cho 2 tập dữ liệu class A và class B. 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]

\* Load data và đặt tên cho các cột:

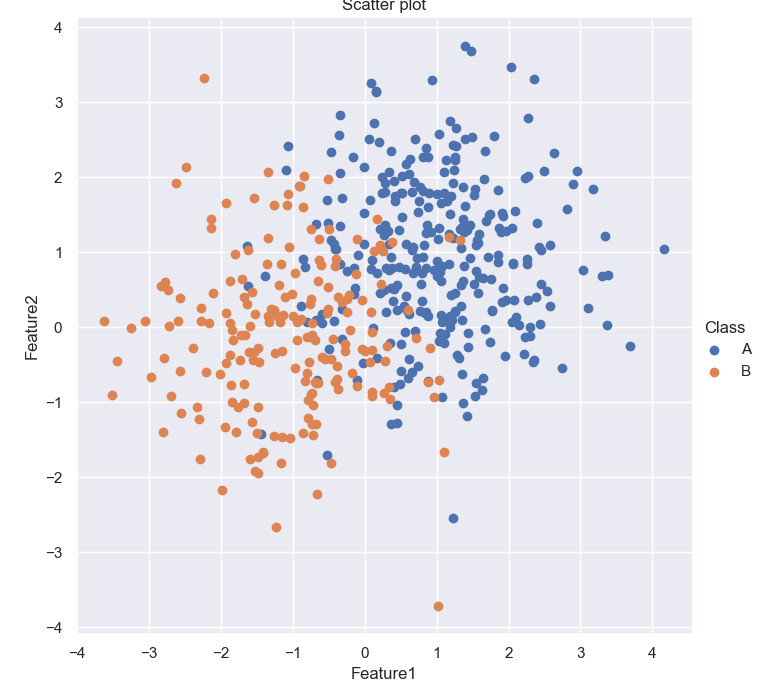
|  |  |
| --- | --- |
| |  | | --- | | columnsName = ['Feature1','Feature2']  classA = pd.DataFrame(scipy.io.loadmat('./Week3/Data/classA.mat')['classA'],columns = columnsName)  classB = pd.DataFrame(scipy.io.loadmat('./Week3/Data/classB.mat')['classB'],columns = columnsName) | |

Ta thực hiện lần lượt các bước:

- Nối 2 tập A và B thành một tập dữ liệu lớn.

|  |
| --- |
| classAB = 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(classAB, hue="Class", height=7).map(plt.scatter,"Feature1","Feature2",).add\_legend()  plt.title('Scatter plot')  plt.savefig("Week3/2a.png")  plt.show() |

- Plot tập dữ liệu đó trên seaborn.



Số lượng mẫu của ClassA là: 300

Số lượng mẫu của ClassB là: 195

Tách tập dữ liệu thành 2 phần, 70% dữ liệu để train và 30% dữ liệu để test:

|  |  |
| --- | --- |
| |  | | --- | | # Split train data set and test set  train, test = train\_test\_split(classAB, train\_size=0.7) | |

Ta được số mẫu trong tập dữ liệu train là 396, tập dữ liệu test là 99

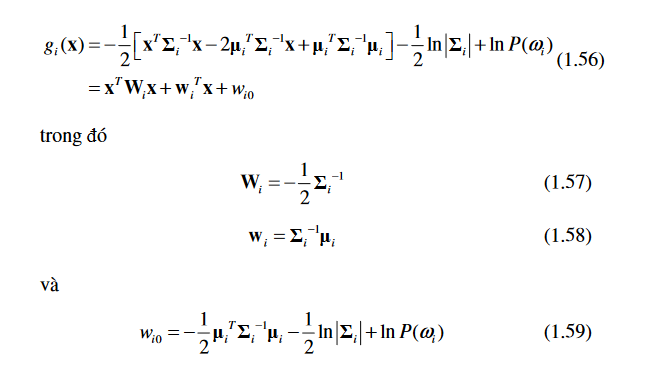
Ta lần lượt tính mean, covariance và cho từng đặc trưng trong mỗi class

|  |  |
| --- | --- |
| |  | | --- | | mu\_list = [np.ravel(x) for x in np.split(train.groupby('Class').mean().values,[1])]  cov\_list = np.split(train.groupby('Class').cov().values,[2])  pi\_list = train.iloc[:,0].value\_counts().values / len(train) | |

Kết quả:

|  |
| --- |
| Mean của từng đặc trưng trong Class A là: [1.0201986 0.9430574]  Mean của từng đặc trưng trong Class B là: [-1.06096171 -0.12991483] |

Để tăng tính đa dụng, ta sẽ dùng biệt hàm sau để có thể sử dụng cho tất cả mọi loại dữ liệu, công thức là:



Cài đặt công thức trên bằng python

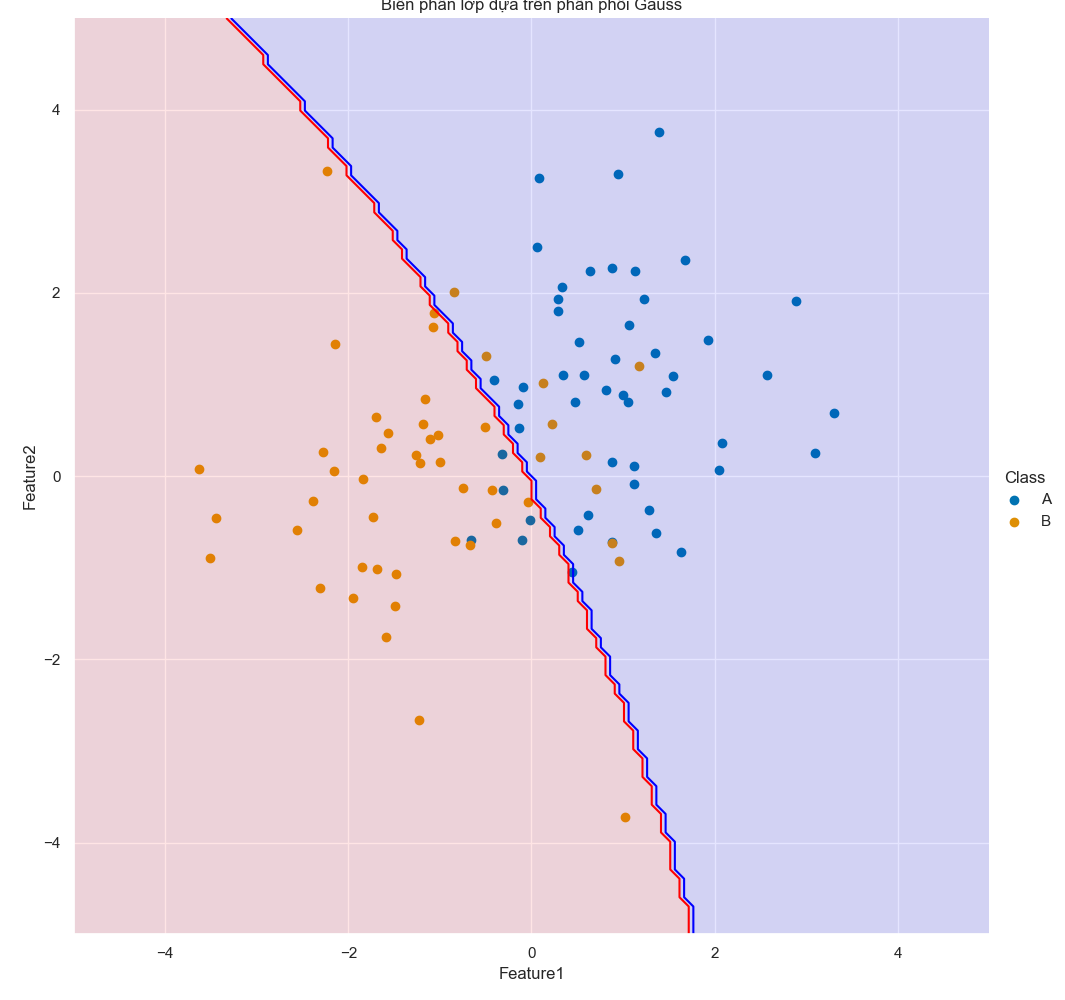
|  |  |
| --- | --- |
| |  | | --- | | def df(X, mu\_list, cov\_list, pi\_list):  scores\_list = list()  classes = len(mu\_list)  for p in range(classes):  Wi = (-1/2)\*np.linalg.inv(cov\_list[p])  wi = np.linalg.inv(cov\_list[p])@mu\_list[p]  wi0 = (-1/2)\*np.transpose(mu\_list[p])@np.linalg.inv(cov\_list[p])@mu\_list[p]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

Ta sẽ đáng giá biệt hàm bằng cách tính toán confusion matrix và precision+recal

|  |
| --- |
| # confusion matrix to precision + recall  def cm2pr\_binary(cm):  p = cm[0,0]/np.sum(cm[:,0])  r = cm[0,0]/np.sum(cm[0])  return (p, r)  # Calculate 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[["Feature1","Feature2"]].values]  label = list(test["Class"].values) |

Kết quả:

|  |
| --- |
| Class A Predict Class B Predict  Class A 45 5  Class B 10 39  Precition = 0.82, Recall = 0.90 |

Plot tập dữ liệu trên với biệt hàm:

## **Bài tập 3:** Xây dựng bộ classifier với 2 lớp, 2 đặc trưng. Giả sử tập dữ liệu có dạng phân bố gauss.

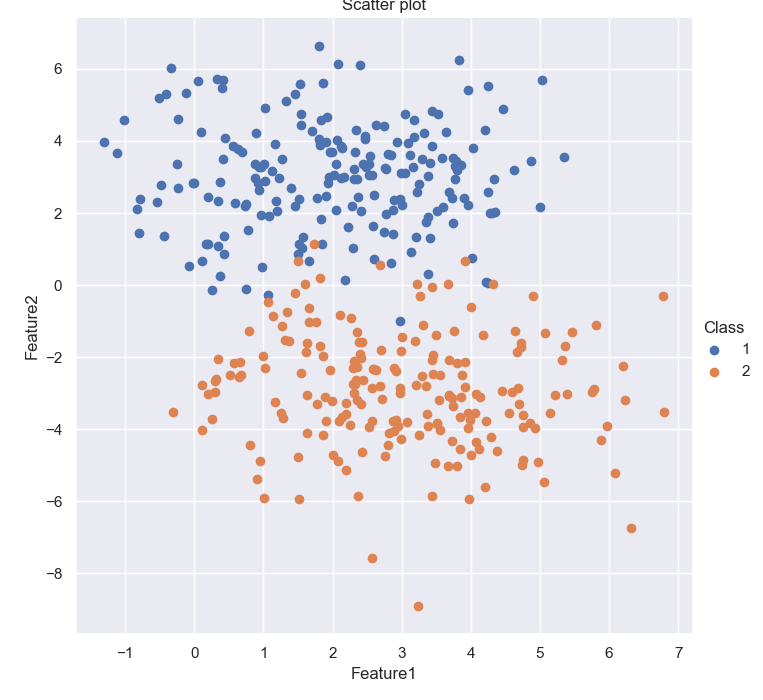
Load dữ liệu train/test

|  |  |
| --- | --- |
| |  | | --- | | # Load data  columnsName = ['Feature1','Feature2']  train1 = pd.DataFrame(scipy.io.loadmat('./Week3/Data/class1\_train.mat')['class1\_train'],columns = columnsName)  test1 = pd.DataFrame(scipy.io.loadmat('./Week3/Data/class1\_test.mat')['class1\_test'],columns = columnsName)  train2 = pd.DataFrame(scipy.io.loadmat('./Week3/Data/class2\_train.mat')['class2\_train'],columns = columnsName)  test2 = pd.DataFrame(scipy.io.loadmat('./Week3/Data/class2\_test.mat')['class2\_test'],columns = columnsName) | |

Tạo tập dữ liệu tổng để plot

|  |
| --- |
| # Merge data frame  train\_set = pd.concat([train1, train2], keys=['1', '2']).reset\_index().drop('level\_1', axis=1).rename(columns = {'level\_0': 'Class'})  test\_set = pd.concat([test1, test2], keys=['1', '2']).reset\_index().drop('level\_1', axis=1).rename(columns = {'level\_0': 'Class'})  # Concat into one large dataset to plot  data\_set = pd.concat([train\_set, test\_set]).reset\_index().drop("index", axis=1)  # Plot data  sns.set()  sns.FacetGrid(data\_set, hue="Class", height=7).map(plt.scatter,"Feature1","Feature2",).add\_legend()  plt.title('Scatter plot')  plt.savefig("Week3/3a.png")  plt.show() |

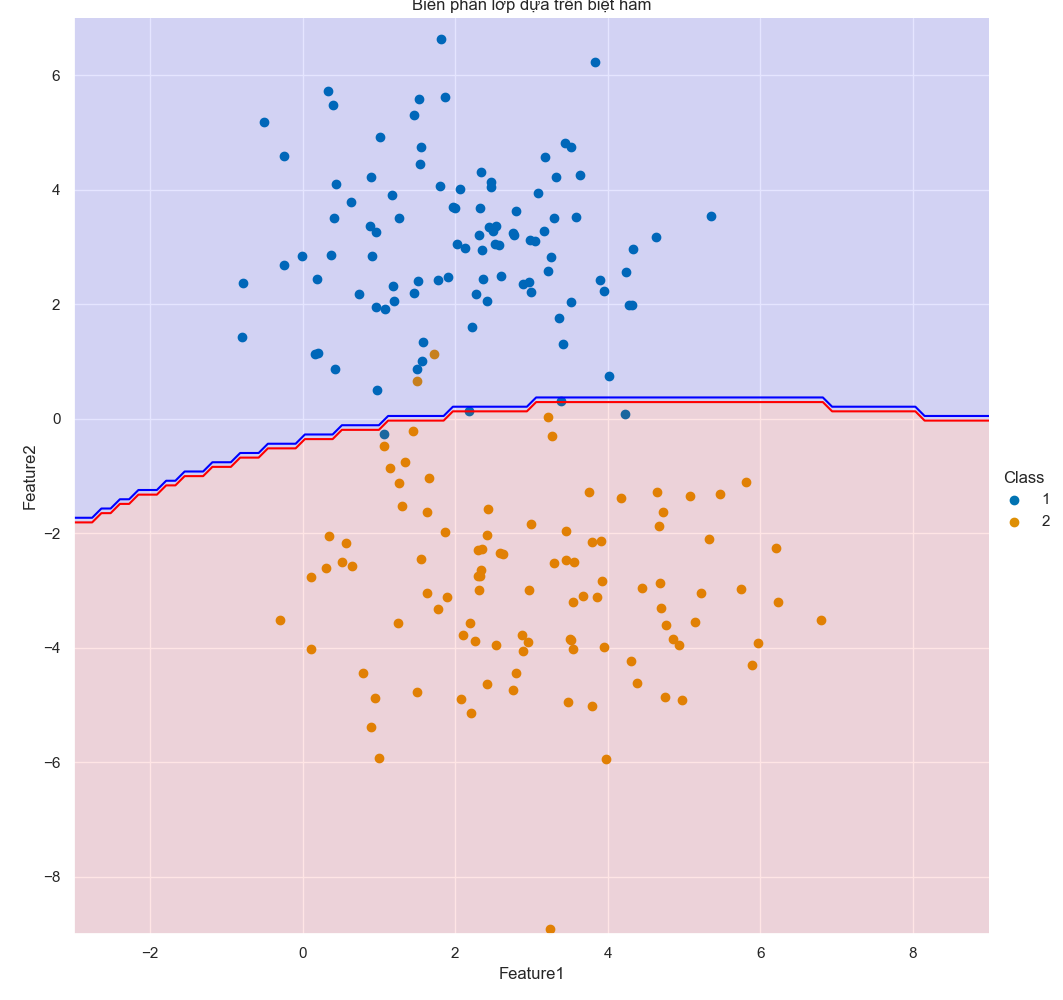
Kết quả:

Tính toán mean và cov giữa các đặc trưng trong từng class:

|  |
| --- |
| Mean của từng đặc trưng trong Class 1 là: [1.97909027 2.97947776]  Mean của từng đặc trưng trong Class 2 là: [ 3.01056198 -2.87167377]  Cov của Class 1 là:  [[2.4802316 0.08304637]  [0.08304637 2.44708868]]  Cov của Class 2 là:  [[ 1.89496114 -0.31990908]  [-0.31990908 2.64843169]] |

Tiếp tục sử dụng biệt hàm ở bài số 2, ta đánh giá được kết quả predict thông qua confusion matrix và precision+recal trên tập test.

|  |
| --- |
| Class 1 predict Class 2 predict  Class 1 98 2  Class 2 4 96  Precition = 0.96, Recall = 0.98 |

Plot biên phân lớp:

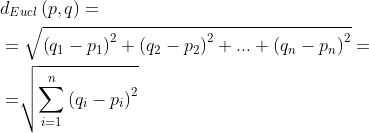
# **Bài thực hành số 4**

## **Bài tập 1:** Xây dựng bộ classifier dựa trên Parzen windows

Dựa vào những bước trước ta có thể sử dụng lại các hàm load dữ liệu, phân chia tập.

Để có thể tính được số lượng điểm có trong một hypercube trước hết ta cần tính khoảng cách từ điểm tâm X tới điểm cần kiểm tra .

Để có thể tính được khoảng cách đó ta có thể áp dụng công thức sau:



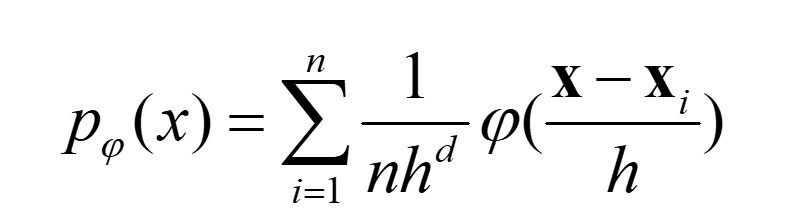
Thực hiện cài đặt bằng python:

|  |
| --- |
| def dist(x, y):  return np.sqrt(np.sum(np.power(x-y, 2))) |

Từ đó ta xây hàm phi như sau:

|  |
| --- |
| def phi(x, y, h):  if(dist(x, y)/h > 0.5):  return False  return True |

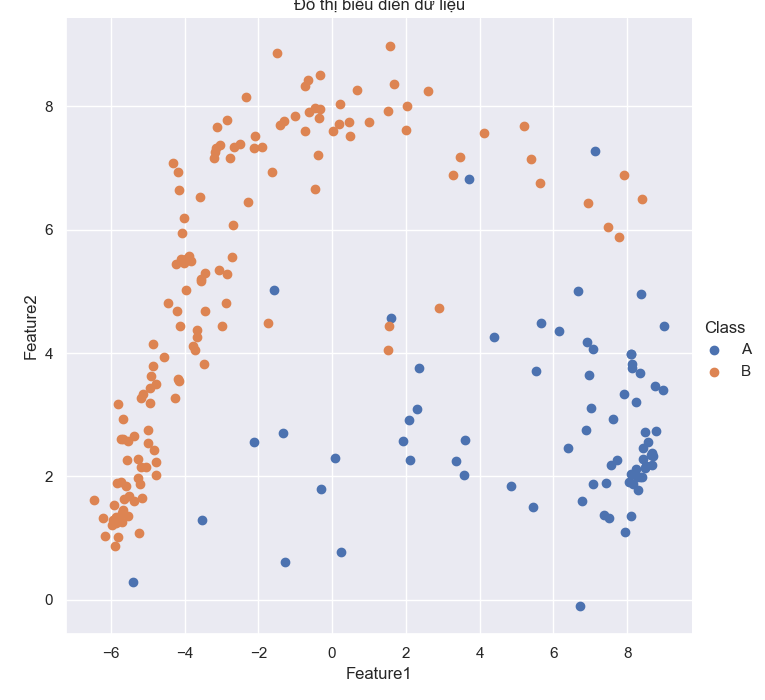
Để tính tổng density ta dùng công thức sau:



Cài đặt:

|  |
| --- |
| def pw(X, data\_set, h):  score\_list = list()  for p in data\_set.groupby("Class"):  k = 0  n = p[1].shape[0] # Get number of feature  d = len(p[1].shape) # Get dimenstino  for x in p[1][['Feature1','Feature2']].to\_numpy():  k += phi(X, x, h)  score\_list.append(k/(n \* (h\*\*d)))  return np.argmax(score\_list) |

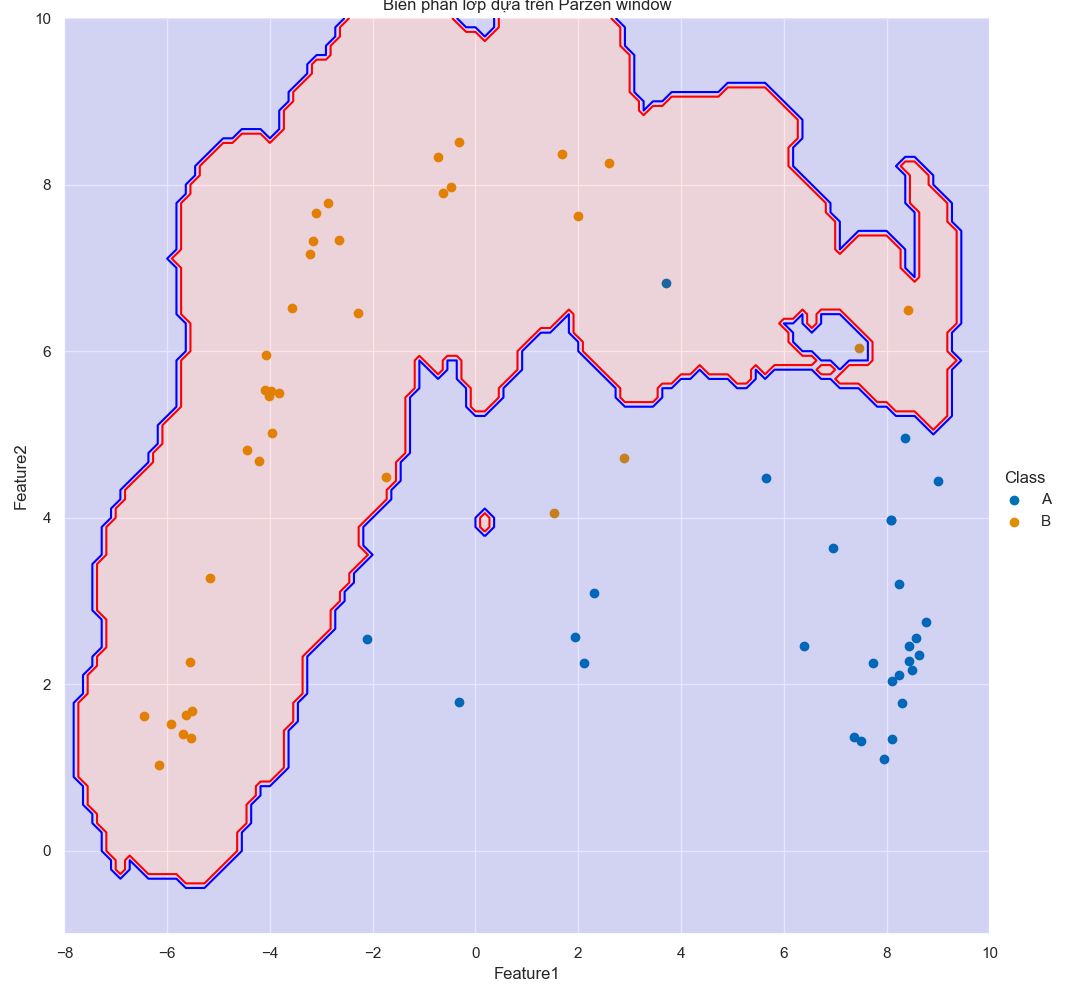
\*Plot tập dữ liệu tổng



Ta thu được kết quả output như sau:

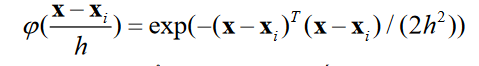
|  |
| --- |
| Số lượng tập train: 151  Số lượng tập test : 65  Class A predict Class B predict  Class A 28 1  Class B 3 33  Precition = 0.90, Recall = 0.97 |

Plot tập dữ liệu tổng cùng với biên phân lớp:



## **Bài tập 2:** Xây dựng bộ classifier sử dụng hàm cửa sổ Gauss

Tiếp tục làm theo các bước của bài 1, tuy nhiên ta cần phải sửa lại hàm phi của mình thành hàm Gauss có công thức sau:



Cài đặt công thức:

|  |
| --- |
| def phi(x, y, h):  return np.exp((-np.transpose(x-y)@(x-y))/(2\*np.power(h, 2))) |

Kết quả output:

|  |
| --- |
| Số lượng tập train: 151  Số lượng tập test : 65  Class A predict Class B predict  Class A 23 2  Class B 2 38  Precition = 0.92, Recall = 0.92 |

Plot tập dữ liệu cùng biên phân lớp:

