







Mục lục

[I. Bài thực hành số 2 3](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.gjdgxs)

[1. Xây dựng một phân lớp dựa trên histogram 3](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.30j0zll)

[2. Xây dựng bộ phân lớp trên tập dữ liệu ‘person\_data.txt’ 5](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.1fob9te)

[3. Tính mean và variance của các vector đặc trưng sau: 8](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.3znysh7)

[4. Tính covariance matrix của các vector đặc trưng sau: 10](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.2et92p0)

[5. Tạo hàm mật độ của phân bố Gauss với mean là 5 và variance là 3 11](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.tyjcwt)

[6. Tạo hàm mật độ của phân bố Gauss khác với mean là 2 và variance là 1.5 12](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.3dy6vkm)

[7. Tạo hàm mật độ phân bố Gauss 2 chiều với mean [1 3] và variance [2 2] 12](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.1t3h5sf)

[8. Xây dựng bộ classifier sử dụng 1 đặc trưng có sẳn 13](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.4d34og8)

[9. Xây dựng bộ classifier với 2 tập dữ liệu là cross.dat và twoclass.dat 17](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.2s8eyo1)

[II. Bài thực hành số 3 20](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.17dp8vu)

[1. Xây dựng bộ phân lớp và trực quan hóa biên phân lớp đối với tập dữ liệu iris 20](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.3rdcrjn)

[2. Xây dựng bộ classifier với 2 đặc trưng từ 2 tập dữ liệu classA.mat và classB.mat 23](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.26in1rg)

[3. Xây dựng bộ classifier với 2 lớp, 2 đặc trưng 26](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.lnxbz9)

[III. Bài thực hành số 4 29](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.35nkun2)

[1. Xây dựng bộ classifier dựa trên Parzen window 29](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.1ksv4uv)

[2. Lặp lại bài 1 với hàm cửa sổ Gauss 32](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.2jxsxqh)

[3. Xây dựng bộ classifier để nhận dạng chữ số 34](https://docs.google.com/document/d/1UGSaUHHHdh81BVh2y9wAPVKH1yOXoLFh/edit#heading=h.z337ya)

**Bài thực hành 1**

# **Bài tập 1: Tìm điểm phân lớp sao cho kết quả phân lớp chính xác nhất.**

**Mô tả bài toán: Cho 2 lớp dữ liệu 1 chiều, tìm điểm phân lớp để phân ra 2 lớp sao cho lỗi ít nhất.**

**Ý tưởng:**

- Với mỗi điểm tồn tại trong dữ liệu, ta tính độ lỗi tại điểm đó và lưu vào list, biên phân lớp sẽ là điểm có độ lỗi thấp nhất.

- Độ lỗi được tính bằng số điểm bị phân loại sai ở lớp 1 cộng số điểm bị phân loại sai ở lớp 2. Điểm bị phân lớp sai ở lớp 1 là những điểm lớn hơn hoặc bằng điểm đang xét và ở lớp 2 là những điểm nhỏ hơn điểm đang xét.

**Hướng giải quyết:**

B1: Với 2 lớp dữ liệu tương ứng với hai mảng, ta sẽ gộp chúng lại thành một mảng tên là xy, rồi xóa những điểm dữ liệu bị trùng trên mảng đó.

B2: Sau đó ta lặp qua các điểm trong mảng xy, ta sẽ tính lỗi tại mỗi điểm bằng cách:

- Với lớp 1, ta sẽ tìm những điểm lớn hơn hoặc bằng điểm đang xét bằng np.where() trên mảng chứa lớp 1, output ta sẽ được một list là list1

- Với lớp 2, ta sẽ tìm những điểm nhỏ hơn điểm đang xét bằng np.where() trên mảng chứa lớp 2, output ta sẽ được một list là list2

=> Độ lỗi tại điểm đó bằng tổng độ dài của list1 và list2

B3: Chọn phần tử có độ lỗi thấp nhất làm biên phân lớp

**Code**:

|  |
| --- |
| **# Tạo 2 tap du lieu**  **X = np.array([1, 3, 5, 6, 6, 8])**  **Y = np.array([7, 7, 8, 9, 10, 11])** |

|  |
| --- |
| **def classify(X, Y):**  **# Nối 2 tập dữ liệu lại**  **xy = np.concatenate((X, Y))**  **# lọc bỏ những phần tử trùng**  **xy = np.unique(xy)**  **err = []**  **for i in xy:**  **# Tính loss (độ lỗi của mỗi phần tử và luư lại)**  **loss = len(np.where(X >= i)[0]) + len(np.where(Y < i)[0])**  **err.append(loss)**  **# Lấy phần tử có loss nhỏ nhất**  **point\_classify = xy[np.argmin(err)]**  **# Lấy loss nhỏ nhất**  **loss\_min = min(err)**  **return point\_classify, loss\_min** |

|  |
| --- |
| **point\_classify, loss = classify(X, Y)**  **print("Điểm phân lớp là", point\_classify, "với loss là", loss)**  **----OUTPUT----**  Điểm phân lớp là 7 với loss là 1 |

## **Bài tập 2: Tạo dữ liệu và phân lớp cho dữ liệu đã tạo**

#### **Tạo dữ liệu.**

**Ý tưởng:**

- Ta đưa ra một phương trình đường thẳng có dạng *𝑦*=*𝑎𝑥*+*𝑏* , ta random giá trị x và dựa vào đường thẳng đưa ra giá trị y, giá trị y đó sẽ cộng thêm noise và đánh label cho các dữ liệu đó theo 2 lớp để được một tập dữ liệu.

**Hướng giải quyết:**

- Đầu tiên ta xây dựng hàm là một đường thẳng có dạng: y = 5x + 3, với input là x và output là y.

- Đưa ra số lượng điểm dữ liệu cần

- Mỗi điểm dữ liệu cần có x, y, và label nên ta sẽ thực hiện các bước sau

+ Random giá trị x

+ Random noise

+ Từ giá trị random x ta sinh ra giá trị y cộng với giá trị noise

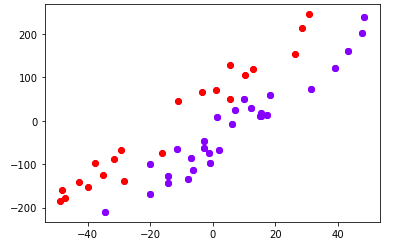
+ Với giá trị noise dương, tôi sẽ đánh label cho điểm đó là 0, ngược lại là 1

- Visualize dữ liệu

**Code**

|  |
| --- |
| **# Hám số dùng để tạo data**  **def yy(x):**  **return 5\*x + 3**  **# Tạo data từ random và hàm số**  **X, Y, label =[], [], []**  **num = 50**  **for i in range(num):**  **x = rd.randrange(-50,50) + rd.random() # sinh x**  **noise = rd.randint(-100, 100) + rd.random() #sinh noise**  **y = yy(x) + noise # sinh y bằng hàm số có thêm noise**  **X.append(x)**  **Y.append(y)**  **if noise >= 0: # đánh label**  **label.append(0)**  **else:**  **label.append(1)** |

|  |
| --- |
| **X1, Y1, X2, Y2 = [], [], [], []**  **# lọc data để visualize**  **for i in range(len(label)):**  **if label[i] == 0:**  **X1.append(X[i])**  **Y1.append(Y[i])**  **else:**  **X2.append(X[i])**  **Y2.append(Y[i])**    **plt.scatter(X1, Y1, color='r')**  **plt.scatter(X2, Y2, color='b')** |

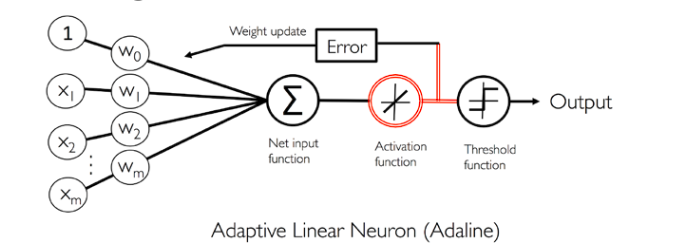
****

**Phân lớp dữ liệu**

#### **Ý tưởng:** Bài này sử dụng thuật toán Adaline để phân loại dữ liệu trên

**Cơ sở lí thuyết:**

- Adaptive linear neurons (Adaline ) là một loại single-layer neural network (NN)

****

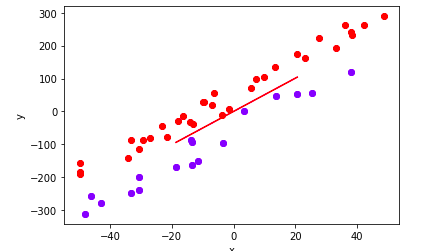
- Hàm loss function: được định nghĩa là tổng bình phương lỗi giữa kết quả tính toán và nhãn thực tế - **sum of squared errors (SSE)**

- Ở thuật toán này, ta dùng một phương pháp phổ biến để tối ưu là **Gradient Descent (GD)**

- Áp dụng GD ta có biểu thức cập nhất của w là: . Trong đó được định nghĩa là độ dốc âm nhân với learning rate ().

- Để cập nhật, ta cần tính đạo hàm riêng của hàm chi phí theo từng trọng số , vậy công thức cập nhật của ta là:

|  |
| --- |
| **X = np.array(X).reshape(-1,1)**  **y = np.array(Y).reshape(-1,1)**  **x = np.hstack((np.ones((X.shape[0], 1)), X))**  **w = np.array([0.,1.]).reshape(-1,1)**  **numOfIteration = 100**  **cost = np.zeros((numOfIteration,1))**  **learning\_rate = 0.000001**  **for i in range(1, numOfIteration):**  **# Tính độ lỗi với label thật**  **r = np.dot(x, w) - y**  **cost[i] = 0.5\*np.sum(r\*r)**  **w[0] -= learning\_rate\*np.sum(r)**  **# correct the shape dimension**  **w[1] -= learning\_rate\*np.sum(np.multiply(r, x[:,1].reshape(-1,1)))**  **plt.scatter(X1, Y1, color='r')**  **plt.scatter(X2, Y2, color='b')**  **plt.xlabel('x')**  **plt.ylabel('y')**  **N = X.shape[0]**  **predict = np.dot(x, w) #y hat**  **plt.plot((x[0][1], x[N-1][1]),(predict[0], predict[N-1]), 'r')# ve duong thang**  **plt.show()** |

****

**Bài thực hành 2**

# **Câu 1: Xây dựng một phân lớp dựa trên histogram, với tập dữ liệu (tham khảo bài toán từ Lecturer 3):**

# **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)]**

**Với dữ liệu ở trên, số đầu tiên thể hiện số phút sau 6h30, số thứ 2 thể hiện cố lần xảy ra sự việc này.**

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

**Ý tưởng:**

- Gọi số lần đi đúng giờ là in, số lần đi trễ là late, ta tính xác suất đi đúng giờ bằng, xác suất đi trễ là 1 - xác suất đi đúng giờ.

- Xây dựng hàm tính xác suất với input là thời gian cần dự đoán sẽ đi đúng hay trễ giờ, output là xác suất đi đúng giờ

**Hướng giải quyết:**

- Đầu tiên, ta vẽ histogram cho bộ dữ liệu

- Mỗi điểm dữ liệu gồm 2 thành phần là thời gian và số lần đi đúng giờ đối với lớp đúng giờ, tương tự với lớp trể giờ.

- Ta sẽ chuyển 2 phần từ này thành dạng dictionary với key là thời gian và value là số lần để khi ta cần tính xác suất tại thời gian nào thì dễ dàng lấy được tần suất của thời gian đó xảy ra.

- Xây dựng hàm với input là thời gian cần tính xác suất đi đúng giờ, với thời gian đó, ta lấy ra tần suất của thời gian đó với lớp in\_time và đặt tên biến là s, và với lớp cls\_late thì đặt tên là m, và output của hàm là xác suất đi đúng giờ được tính bằng **.**

|  |
| --- |
| **# Load data vào list**  **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)]**  **# Tách mỗi phần tử dữ liệu ra để thực hiện visualize data**  **X1, Y1 = zip(\*in\_time)**  **X2, Y2 = zip(\*cls\_late)**  **# Visualize dữ liệu in\_time**  **plt.bar(X1, Y1, color="green", alpha=0.75, label="In Time")**  **# Visualize dữ liệu late**  **plt.bar(X2,Y2, color="orange", alpha=0.75, label="Late")**  **# Show chú thích**  **plt.legend(loc='upper right')**  **plt.show()** |

# 

|  |
| --- |
| **# Chuyển list in\_time thành dangj dictionary**  **in\_time\_dict = dict(in\_time)**  **too\_late\_dict = dict(cls\_late)**  **# Xây dựng hàm xác suất**  **def train(min):**  **# Lấy số lần xảy ra lúc giấy thứ min ở dữ liệu in\_time**  **s = in\_time\_dict.get(min,0)**  **if s == 0:**  **return 0**  **else:**  **# Lấy số lần xảy ra lúc giấy thứ min ở dữ liệu late**  **m = too\_late\_dict.get(min,0)**  **# Trả về xác suất đi đúng giờ**  **return s/(s+m)** |

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

|  |
| --- |
| **for minutes in range(4,9):**  **if train(minutes) > 0.5:**  **print('Lúc 6h3' + str(minutes) + ' đi đúng giờ với xác suất ' + str(train(minutes)))**  **else:**  **print('Lúc 6h3' + str(minutes) + ' đi trể với xác suất ' + str(train(minutes)))**  **----OUTPUT----**  **Lúc 6h34 đi đúng giờ với xác suất 1.0**  **Lúc 6h35 đi đúng giờ với xác suất 0.8695652173913043**  **Lúc 6h36 đi đúng giờ với xác suất 0.7916666666666666**  **Lúc 6h37 đi đúng giờ với xác suất 0.68**  **Lúc 6h38 đi trể với xác suất 0.4** |

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

**Cở sở lý thuyết:** Tương tự nhu bài trên, để phân lớp cho bộ dữ liệu trên, ta sẽ tính xác suất của 2 lớp dựa trên đặc trưng nào đó (ở bài này là chiều cao), để xem xác suất lớp nào lớn hơn thì điểm đó thuộc lớp đó.

**Ý tưởng:** Với mỗi chiều cao, ta sẽ đếm xem có bao nhiêu male - c\_male, bao nhiêu female - c\_female và lưu lại, khi nhận một chiều cao để dự đoán, ta tính xác suất là male P\_male = và xác suất là female P\_female = và nếu P\_male > P\_female thì với chiều cao đó ta sẽ phân lớp nó vào lớp male và ngược lại.

**Hướng giải quyết**

- Show dữ liệu và xử lý dữ liệu để dễ dàng thực hiện các bước sau:

+ Mỗi dòng dữ liệu là một chuỗi gồm họ và tên người, chiều cao, cân nặng và giới tính.

+ Ta sẽ đưa tên vào một list gọi là name, đưa chiều cao vào list height, đưa cân nặng vào list weight, đưa giới tính vào list gender.

- Ta tạo 2 dictionary để lưu số lượng male và female tương ứng với mỗi chiều cao. Với dict về male, ta sẽ lưu chiều cao cũng với số lượng male trong chiều cao đó, tương tự với dict female. Key là height, value là tán suất của male hay female.

- Sau khi có bộ dictionary đó, ta tiến hành xây dựng hàm predict giá trị, với input là chiều cao, khi có chiều cao ta sẽ tính được ta tính xác suất là male P\_male = và xác suất là female P\_female = và nếu P\_male > P\_female thì ta trả về kết quả predict là male, ngược lại là female.

**Code:**

|  |
| --- |
| **# Load dữ liệu**  **data = open("person\_data.txt", "r", encoding='UTF-8')**  **data = data.readlines()** |

|  |
| --- |
| **data[:5]**  **----OUTPUT----**  ['Randy Carter 184 73.0 male\n',  'Stephanie Smith 149 52 female\n',  'Cynthia Watson 174 63 female\n',  'Jessie Morgan 175 67.0 male\n',  'Katherine Carter 183 81 female\n'] |

|  |
| --- |
| **name, height, weight, gender = [], [], [], []**  **count\_male = {}**  **count\_female = {}**  **for d in data:**  **# Tách dữ liệu chuổi thành list theo dấu khoảng trắng**  **tmp = d.split()**  **# tmp[0] và tmp[1] là họ và tên, ta ghép lại và lưu vào list name**  **name.append(tmp[0] + ' ' + tmp[1])**  **# lưu chiều cao vào list height**  **height.append(int(tmp[2]))**  **# lưu cân nặng vào list weight**  **weight.append(tmp[3])**  **# lưu giới tính vào list gender**  **gender.append(tmp[4])**  **if str(tmp[4]) == "male":**  **# đếm xem có bao nhiêu male trong mỗi chiều cao**  **if tmp[2] in count\_male.keys():**  **count\_male[tmp[2]] = count\_male[tmp[2]] + 1**  **else:**  **count\_male[tmp[2]] = 1**  **else:**  **# đếm xem có bao nhiêu female trong mỗi chiều cao**  **if tmp[2] in count\_female.keys():**  **count\_female[tmp[2]] = count\_female[tmp[2]] + 1**  **else:**  **count\_female[tmp[2]] = 1** |

|  |
| --- |
| **def bayes\_classify\_male\_female(height):**  **# Lấy số lượng male có chiều cao height**  **c\_male = count\_male.get(str(height), 0)**  **# Lấy số lượng female có chiều cao height**  **c\_female = count\_female.get(str(height), 0)**  **if c\_male == 0 and c\_female == 0:**  **return "error"**  **# Tính xác suất male**  **p\_male = c\_male/(c\_male+c\_female)**  **# Tính xác suất female**  **p\_female = c\_female/(c\_male+c\_female)**  **if p\_male > p\_female:**  **return "male"**  **else:**  **return "female"** |

|  |
| --- |
| **height\_test = 163**  **result = bayes\_classify\_male\_female(height\_test)**  **print("Với height là " + str(height\_test) + " thì predict là " + result)**  **----OUTPUT----**  **Với height là 163 thì predict là female** |

# **Câu 3: Tính mean và Variance của các vector đặc trưng sau**

**Cơ sở lý thuyết:**

Cho N giá trị x1, x2, ... xN

Kỳ vọng (mean) : Là giá trị mong muốn, nó đơn giản là trung bình cộng của toàn bộ các giá trị

Phương sai (variance): là trung bình cộng của bình phương khoảng cách từ mỗi điểm tới kỳ vọng

|  |
| --- |
| **# Hàm tính mean, lấy tổng giá trị các phần tử chia cho số lượng phần tử**  **def mean(vector):**  **return sum(vector)/len(vector)**  **# Hàm tính variance bằng công thức sau**  **def variance(vector):**  **rel = 0**  **m = mean(vector)**  **n = len(vector)**  **for i in vector:**  **rel += (i - m)\*(i - m)/n**  **return rel** |

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

|  |
| --- |
| **a = [1, 2, 4, 6, 9, 10, 20, 7]**  **print('mean: ', mean(a))**  **print('variance: ', variance(a))**  **----OUTPUT----**  **mean: 7.375**  **variance: 31.484375** |

**b. [0 2 4 6 8 … 100]; tất cả các số chẳn từ 0 đến 100.**

|  |
| --- |
| **b = list(range(0, 101, 2))**  **print('mean: ', mean(b))**  **print('variance: ', variance(b))**  **----OUTPUT----**  **mean: 50.0**  **variance: 866.6666666666666** |

**c. [1 3 25 … 9801]; tất cả bình phương các số lẻ từ 1 đến 100**

|  |
| --- |
| **c = np.power(list(range(1, 100, 2)), 2)**  **print('mean: ', mean(c))**  **print('variance: ', variance(c))**  **----OUTPUT----**  **mean: 3333.0**  **variance: 8884444.8** |

**d.** [[2,4], [3,7], [4,6], [5,5], [2,3]]

|  |
| --- |
| **def mean\_2D(lst):**  **d = np.array(lst)**  **# Tổng giá trị các phẩn tử**  **s1 = sum(d)[0]**  **s2 = sum(d)[1]**  **# Tổng số phần tử**  **c1 = d.shape[0]**  **c2 = d.shape[0]**  **return [s1/c1, s2/c2]**  **def variance\_2D(lst):**  **d = np.array(lst)**  **m1 = mean\_2D(lst)[0]**  **m2 = mean\_2D(lst)[1]**  **rel1, rel2 = 0, 0**  **for i in lst:**  **rel1 += (i[0] - m1)\*(i[0] - m1)**  **rel2 += (i[1] - m2)\*(i[1] - m2)**  **N = d.shape[0]**  **return [rel1/(N), rel2/(N)]** |

|  |
| --- |
| **d = [[2,4], [3,7], [4,6], [5,5], [2,3]]**  **print("mean: ",mean\_2D(d))**  **print("variance: ",variance\_2D(d))**  **----OUTPUT----**  **mean: [3.2, 5.0]**  **variance: [1.3599999999999999, 2.0]** |

# **Câu 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].**

**Cơ sở lý thuyết:**

**Ma trận hiệp phương sai :** Cho N điểm dữ liệu được biểu diễn bởi các vector cột x1, x2, ..., xN, khi đó, vector kỳ vọng và ma trận hiệp phương sai của toàn bộ dữ liệu được định nghĩa là:

Mean :

**Covariance Matrix:**



**Hướng giải quyết:**

- Từ 2 vector điểm dữ liệu, ta đưa thành ma trận 2 dòng

- Tính mean của matrix đó và xây dựng matrix mean với độ dài bằng ma trận ban đầu, mục đích để thực hiện trừ 2 matrix

- Sau khi trừ trừ xong ta ra được matrix , ta nhân nó với transpose của chính nó chia cho N-1 => covariance matrix

**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]**  **# Gộp hai vector thành matrix**  **matrix = [X, Y]**  **# Tính mean của matrix đó**  **mean\_matrix = [mean(matrix[0]), mean(matrix[1])]**  **M, tmp1, tmp2 = [], [], []**  **for i in range(len(X)):**  **tmp1.append(mean\_matrix[0])**  **tmp2.append(mean\_matrix[1])**  **# Tạo ma trận mean**  **M = np.array([tmp1, tmp2])**  **# Lấy matrix x - matrix mean**  **D = matrix - M**  **C = (1/(len(X)-1))\*D.dot(D.T)**  **C**  **----OUTPUT----**  **array([[6.16666667, 2.6 ],**  **[2.6 , 2. ]])** |

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

**Cơ sở lý thuyết:**

- Độ lệch chuẩn: tính bằng căn bậc 2 của variance.

- Hàm mật độ xác suất:

**Hướng giải quyết:**

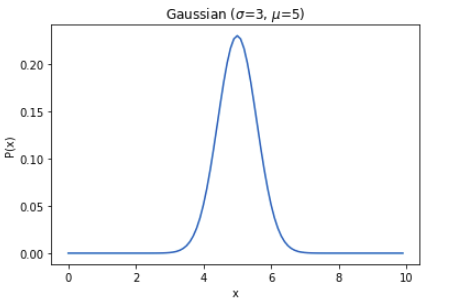
- Xây dựng hàm mật độ xác suất dựa trên công thức trên, với input đầu vào là điểm x, mean và variance.

- Ở bài này mean và variance được cho mặc định.

- Ta sinh một vector x có giá trị trong khoảng [-10 : 10 : 0.1]

- Với mỗi điểm trong vector x đưa vào hàm Gauss ta nhận được xác suất của nó, lưu lại và visualize.

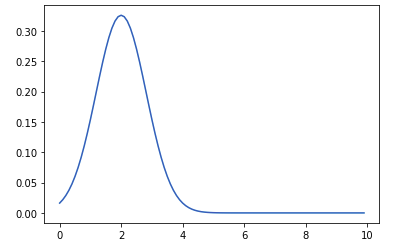
|  |
| --- |
| **# Xây dựng hàm phân phối gauss với 3 tham số truyền vào là x, meann, variancee**  **def gauss(X, meann, variancee):**  **std = np.sqrt(variancee)**  **return (1/(std\*np.sqrt(2\*math.pi)))\*math.exp(-np.power(X-meann, 2)/2\*variancee)**  **# Sinh dữ liệu: 0, 0.1, 0.2, ... 10**  **X = np.arange(0, 10, 0.1)**  **meann = 5**  **variancee = 3**  **G = []**  **# Tính gauss theo x**  **for i in X:**  **G.append(gauss(i, meann, variancee))**  **# Vẽ hàm mật độ xác suất với giá trị từ 0 -> 10**  **plt.plot(X, G)**  **plt.title(r"Gaussian ($\sigma$="+ str(variancee) + ", $\mu$=" + str(meann) + ")")**  **plt.ylabel("P(x)")**  **plt.xlabel("x")**  **plt.show()** |

****

# Câu 6: Tạo hàm mật độ của phân bố Gauss khác với mean là 2 và variance là 1.5.

Tương tự câu trên :

|  |
| --- |
| **X = np.arange(0, 10, 0.1)**  **meann = 2**  **variancee = 1.5**  **G = []**  **for i in X:**  **G.append(gauss(i, meann, variancee))**  **plt.plot(X, G)** |

****

# Câu 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].

Cơ sở lý thuyết:

- Hàm Gaussian 2 chiều:

- Khoảng cách Mahalanobis của vector so với một nhóm có trung bình là và và ma trận hiệp phương sai S được định nghĩa:

Hướng giải quyết :

- Viết hàm Gaussion 2 chiều theo công thức trên

- Tạo 2 mảng sinh giá trị tọa độ cách đều nhau trong khoảng (-10, 10) với độ dài mỗi mảng là 50

- Sau đố ta tạo 2 ma trận tọa độ từ 2 vector tọa độ đó

- Tính xác suất phân phối Gauss của 2 vector đó,

- Sau đó dùng hàm plt.contourf để vẽ các đường đồng mức với x, y là tọa độ 2 chiều, z là chiều cao mà đường bao được vẽ qua đó.

**def gauss\_multi\_dim(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.)))**

**# Tạo ra list các giá trị cách đều nhau, mặc định là 50 giá**

**x = np.linspace(-10, 10)**

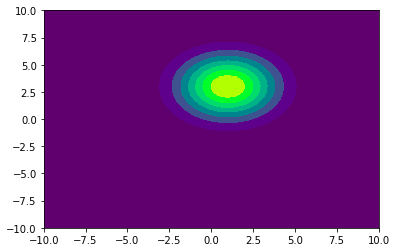
**y = np.linspace(-10, 10)**

**x, y = np.meshgrid(x, y)**

**z = gauss\_multi\_dim(x, y, 1, 3, 2, 2)**

**plt.contourf(x,y,z)**

**plt.show()**

****

**def mahalanobis\_distance(sample\_1):**

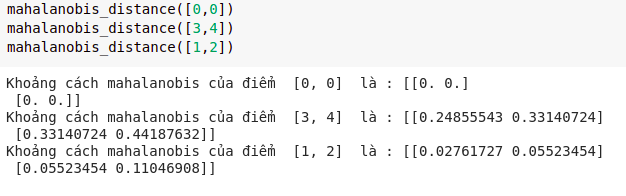
**list\_x1 = []**

**for i in range(50):**

**list\_x1.append(sample\_1)**

**m1 = np.asarray(list\_x1)**

**print("Khoảng cách mahalanobis của điểm ", sample\_1, " là :", np.dot(np.dot(m1.T,np.cov(z)), m1))**

****

# 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.**

**• 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.**

Cơ sở lý thuyết:

**Load data:**

* **Load 2 file tương ứng cho 2 class là: cross.dat và twoclass.dat**

**# Hàm hình mean, lấy tổng giá trị các phần tử chia cho số lượng phần tử**

**def mean(vector):**

**return sum(vector)/len(vector)**

**# Hàm tính variance bằng công thức sau**

**def variance(vector):**

**rel = 0**

**m = mean(vector)**

**n = len(vector)**

**for i in vector:**

**rel += (i - m)\*(i - m)/n**

**return rel**

**# function covariance**

**def covariance(x):**

**N = len(x)**

**TB = mean(x)**

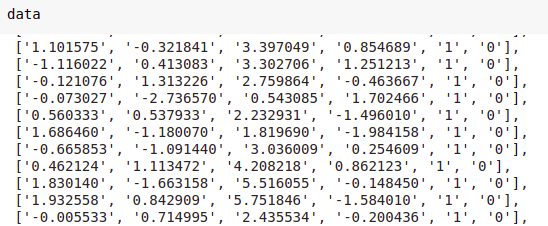
**M = []**

**for i in x:**

**M.append(i - TB)**

**D = np.asarray(M)**

**return (1/(N-1))\*D\*D.T**

****

* **Cho biết số mẫu và số đặc trưng của mỗi class.**
* **Tính mean, variance, và covariance của các vector đặc trưng.**

**data = [i.strip().split() for i in open("twoclass.dat", "r").readlines()]**

**print("Số mẫu là: " + str(len(data)))**

**print("Số đặc trưng của mỗi class là: " + str(len(data[0][:-2])))**

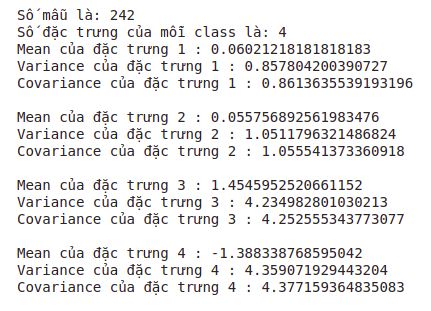
**for i in range(len(data[0][:-2])):**

**print("Mean của đặc trưng " + str(i+1) + " : " + str(mean([float(d[i]) for d in data])))**

**print("Variance của đặc trưng " + str(i+1) + " : " + str(variance([float(d[i]) for d in data])))**

**print("Covariance của đặc trưng " + str(i+1) + " : " + str(np.cov([float(d[i]) for d in data])))**

**print()**

****

* **Trích chọn 1 đặc trưng.**

**# Chọn đặc trưng 1**

**feature\_1 = [float(d[0]) for d in data]**

**feature\_2 = [float(d[1]) for d in data]**

**feature\_3 = [float(d[2]) for d in data]**

**feature\_4 = [float(d[3]) for d in data]**

**class\_1 = [int(d[4]) for d in data]**

**▪ Xây dựng classifier ứng với 1 đặc trưng được chọn:**

**- Chia tập dữ liệu thành 2 tập con, tập huấn luyện gồm 50%.**

**# 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]**

**- Tính histogram và plot:**

**import matplotlib.pyplot as plt**

**plt.hist(feature\_1\_train)**

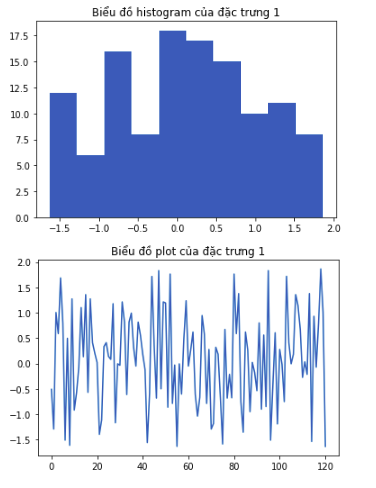
**plt.title("Biểu đồ histogram của đặc trưng 1")**

**plt.show()**

**plt.plot(feature\_1\_train)**

**plt.title("Biểu đồ plot của đặc trưng 1")**

**plt.show()**

****

**Xác định giá trị biệt số (ngưỡng). Ứng với histogram được tính, chọn biệt số**

**sao cho giá trị misclassification là nhỏ nhất.**

**def find\_thresh(x, label):**

**# Khởi tạo ngưỡng và độ lỗi**

**thresh = None**

**num\_errors = 9999999**

**# Lặp qua từng ngưỡng (giá trị) của x**

**for th in x:**

**errors = 0**

**for xi, lb in zip(x, label):**

**# Với mỗi giá trị và label của x, ta xem với x ở lớp 1 có thuộc ngưỡng th không, nếu không thì ta cộng lỗi lên 1**

**if xi > th and lb == 1:**

**errors += 1**

**# Với mỗi giá trị và label của x, ta xem với x ở lớp 2 có thuộc ngưỡng th không, nếu không thì ta cộng lỗi lên 1**

**elif xi <= th and lb == 0:**

**errors += 1**

**# Ta đang tìm lỗi nhỏ nhất, nên khi xuất hiện lỗi nào nhỏ hơn lỗi trước, ta sẽ lưu lại độ lỗi và ngưỡng**

**if errors < num\_errors:**

**num\_errors = errors**

**thresh = th**

**return (thresh, num\_errors)**

**def find\_error\_with\_thresh(x, label, thresh):**

**num\_errors = 0**

**for xi, lb in zip(x, label):**

**# Đếm lỗi của label 1 với ngưỡng xác định**

**if xi > thresh and lb == 1:**

**num\_errors += 1**

**# Đếm lỗi của label 0 với ngưỡng xác định**

**elif xi <= thresh and lb == 0:**

**num\_errors += 1**

**return num\_errors**

**# Đặc trưng 1**

**thresh\_f1 = find\_thresh(feature\_1\_train, class\_train)**

**print("Thresh of feature 1 from trainset is: " + str(thresh\_f1[0]))**

**print("Loss of feature 1 from trainset is: " + str(thresh\_f1[1]))**

**print("Loss of feature 1 from testset ís: " + str(find\_error\_with\_thresh(feature\_1\_test, class\_test, thresh\_f1[0])))**

**print()**

**# Đặc trưng 2**

**thresh\_f2 = find\_thresh(feature\_2\_train, class\_train)**

**print("Thresh of feature 2 from trainset is: " + str(thresh\_f2[0]))**

**print("Loss of feature 2 from trainset is: " + str(thresh\_f2[1]))**

**print("Loss of feature 3 from testset ís: " + str(find\_error\_with\_thresh(feature\_2\_test, class\_test, thresh\_f2[0])))**

**print()**

**# Đặc trưng 3**

**thresh\_f3 = find\_thresh(feature\_3\_train, class\_train)**

**print("Thresh of feature 3 from trainset is: " + str(thresh\_f3[0]))**

**print("Loss of feature 3 from trainset is: " + str(thresh\_f3[1]))**

**print("Loss of feature 3 from testset ís: " + str(find\_error\_with\_thresh(feature\_3\_test, class\_test, thresh\_f3[0])))**

**print()**

**# Đặc trưng 4**

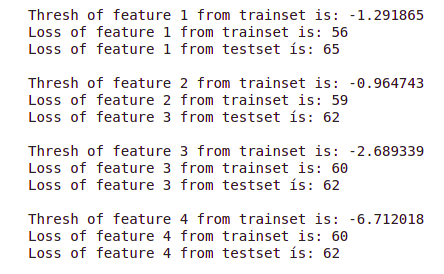
**thresh\_f4 = find\_thresh(feature\_4\_train, class\_train)**

**print("Thresh of feature 4 from trainset is: " + str(thresh\_f4[0]))**

**print("Loss of feature 4 from trainset is: " + str(thresh\_f4[1]))**

**print("Loss of feature 4 from testset ís: " + str(find\_error\_with\_thresh(feature\_4\_test, class\_test, thresh\_f4[0])))**

**print()**

****

**=> Từ kết quả trên ta có thể thấy là đặc trưng 3 và 4 là cho kết quả tốt nhất. Vì số điểm lỗi trên tập train với thực tế chỉ cách nhau thấp nhất.**

**Bài thực hành 3**

**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].**

**Load data: Load 2 file tương ứng cho 2 class là: classA.mat và classB.mat.**

**import scipy.io**

**classA = scipy.io.loadmat('classA.mat')**

**classB = scipy.io.loadmat('classB.mat')**

**Plot dữ liệu.**

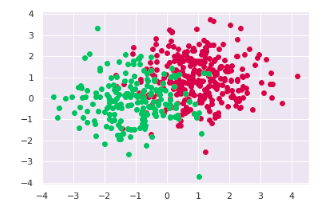
**X1, Y1 = zip(\*classA['classA'])**

**X2, Y2 = zip(\*classB['classB'])**

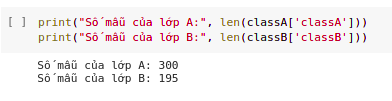
**plt.scatter(X1, Y1, color="r")**

**plt.scatter(X2, Y2, color="g")**

**plt.show()**

****

**Xác định số mẫu của mỗi class.**

****

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

**import random**

**import random**

**# Lấy số lượng 60% mẫu thứ nhất**

**clA = int((len(classA['classA'])\*60)/100)**

**# Lấy số lượng 60% mẫu thứ hai**

**clB = int((len(classB['classB'])\*60)/100)**

**# Tạo training set gồm 60% mẫu thứ nhất và 60% mẫu thứ hai**

**trainset = np.concatenate((classA['classA'][:clA],classB['classB'][:clB]), axis = 0)**

**# Tạo training set gồm 40% mẫu thứ nhất và 40% mẫu thứ hai**

**testset = np.concatenate((classA['classA'][clA:len(classA['classA'])],classB['classB'][clB:len(classB['classB'])]), axis = 0)**

**# convert array sang list**

**lst = trainset.tolist()**

**# Thêm class vào trong data**

**for i in range(len(lst)):**

**if i < clA:**

**lst[i] = lst[i] + [0]**

**else:**

**lst[i] = lst[i] + [1]**

**# convert list ngược lại array**

**trainset = np.array(lst)**

**# Xáo trộn dữ liệu**

**random.shuffle(trainset)**

**random.shuffle(testset)**

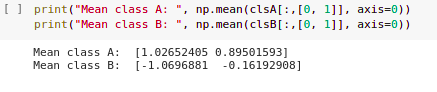
**Huấn luyện**

**Tính mean tương ứng cho từng class.**

**# Lấy dữ liệu lớp A và B từ training set**

**clsA = trainset[:clA]**

**clsB = trainset[clA:]**

****

**Xây dựng biệt thức (discriminant function).**

**# Tính mean của 2 lớp A và B**

**meanA = np.mean(clsA[:,[0, 1]], axis=0)**

**meanB = np.mean(clsB[:,[0, 1]], axis=0)**

**# Tính covariance của 2 lớp A và B**

**covA = np.cov(clsA[:,[0, 1]])**

**covB = np.cov(clsB[:,[0, 1]])**

**# Tính định thức của cov lớp A và B**

**detA = np.linalg.det(covA)**

**detB = np.linalg.det(covB)**

**# Tính ma trận nghich đảo của cov lớp A và B**

**invA = np.linalg.inv(covA)**

**invB = np.linalg.inv(covB)**

**# Tính giá trị biệt hàm**

**def g(x, meann, covv, invv):**

**return 0.5\*(x-meann).T\*invv\*(x-meann)**

**# Test thử với một dữ liệu test**

**g\_1 = g(testset[0:1,[0, 1]], meanA, invA)**

**g\_2 = g(testset[0:1,[0, 1]], meanB, invB)**

**if g\_1 > g\_2:**

**print("x thuộc lớp A")**

**else:**

**print("x thuộc lớp B")**

**Xây dựng hàm predict**

**def predict(x):**

**g\_1 = g(x, meanA, invA)**

**g\_2 = g(x, meanB, invB)**

**if g\_1 > g\_2:**

**return 0**

**else:**

**return 1**

**Predict trên tập test**

**Bài thực hành 4**

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

**- Load 2 file classA.txt và classB.txt.**

**# Load data class A**

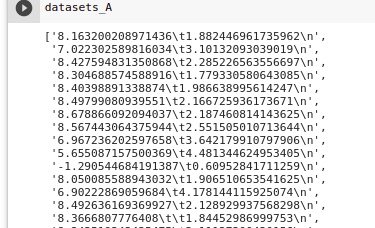
**f = open("classA.txt", "r")**

**datasets\_A = f.readlines()**

**# Load data class B**

**f = open("classB.txt", "r")**

**datasets\_B = f.readlines()**

****

**classA = []**

**for data in datasets\_A:**

**try:**

**classA.append(list(map(float, data.replace("\n", "").split("\t"))))**

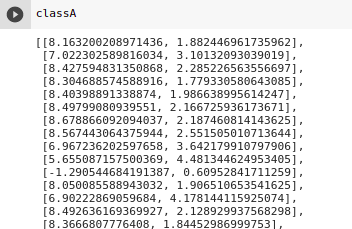
**except:**

**classA.append(list(map(float, data.replace("\n", "").split("\t\t"))))**

**classB = []**

**for data in datasets\_B:**

**classB.append(list(map(float, data.replace("\n", "").split("\t"))))**

****

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

**import numpy as np**

**# Tạo list index**

**idx\_a = np.arange(0, len(classA))**

**# Ranđom lại list index**

**np.random.shuffle(idx\_a)**

**# Lấy ra dữ liệu train 70% và test 30%**

**idx\_train\_a = idx\_a[:(len(classA)\*7)//10]**

**idx\_test\_a = idx\_a[(len(classA)\*7)//10:]**

**# Tạo list index**

**idx\_b = np.arange(0, len(classB))**

**# Ranđom lại list index**

**np.random.shuffle(idx\_b)**

**# Lấy ra dữ liệu train 70% và test 30%**

**idx\_train\_b = idx\_b[:(len(classB)\*70)//100]**

**idx\_test\_b = idx\_b[(len(classB)\*70)//100:]**

**class\_train\_a = [classA[i] for i in idx\_train\_a]**

**class\_train\_b = [classB[i] for i in idx\_train\_b]**

**class\_test\_a = [classA[i] for i in idx\_test\_a]**

**class\_test\_b = [classB[i] for i in idx\_test\_b]**

**- Huấn luyện bộ classifier sử dụng dữ liệu 2 chiều, 2 lớp từ tập training. Áp**

**dụng Parzen window với hàm cửa sổ hypercube bên dưới và h=1**

**import math**

**def distance\_between\_two\_points(point\_A, point\_B):**

**return math.sqrt((point\_A[0] - point\_B[0])\*\*2 + (point\_A[1] - point\_B[1])\*\*2)**

**def parzen(h, d, n\_a, n\_b, h\_d, class\_test, class\_train\_a, class\_train\_b, flag="B"):**

**# Dự đoán dựa vào bộ train class A**

**test\_1 = []**

**for x\_test in class\_test:**

**total = 0**

**for x\_train in class\_train\_a:**

**res = 0**

**if distance\_between\_two\_points(x\_test, x\_train) <= h/2:**

**res = 1**

**total += res**

**p\_1 = (1/n\_a)\*(1/h\_d)\*(total)**

**test\_1.append(p\_1)**

**# Dự đoán dựa vào bộ train class B**

**test\_2 = []**

**for x\_test in class\_test:**

**total = 0**

**for x\_train in class\_train\_b:**

**res = 0**

**if distance\_between\_two\_points(x\_test, x\_train) <= h/2:**

**res = 1**

**total += res**

**p\_2 = (1/n\_b)\*(1/h\_d)\*(total)**

**test\_2.append(p\_2)**

**if flag == "A":**

**ans\_1 = []**

**for p1, p2 in zip(test\_1, test\_2):**

**if p1>p2: # Dự đoán đúng**

**ans\_1.append(True)**

**elif p1<p2: # Dự đoán sai**

**ans\_1.append(False)**

**else: # Không thể dự đoán**

**ans\_1.append("Unknown")**

**return ans\_1.count(True)/len(test\_1)\*100**

**else:**

**ans\_2 = []**

**for p1, p2 in zip(test\_1, test\_2):**

**if p1<p2: # Dự đoán đúng**

**ans\_2.append(True)**

**elif p1>p2: # Dự đoán sai**

**ans\_2.append(False)**

**else: # Không thể dự đoán**

**ans\_2.append("Unknown")**

**return ans\_2.count(True)/len(test\_2)\*100**

**Với h = 1**

**# Liệt kê các tham số cần thiết**

**h = 1**

**d = 2**

**n\_a = len(class\_train\_a)**

**n\_b = len(class\_train\_b)**

**h\_d = h\*\*d**

**result\_1 = parzen\_1(h,d,n\_a,n\_b,h\_d, class\_test\_a, class\_train\_a, class\_train\_b, "A")**

**print("Độ chính xác trên tập test class A: " + str(result\_1) + "%")**

**result\_2 = parzen\_1(h,d,n\_a,n\_b,h\_d, class\_test\_b, class\_train\_a, class\_train\_b, "B")**

**print("Độ chính xác trên tập test class B: " + str(result\_2) + "%")**