Ví dụ 1. Logistic với dữ liệu input là 1 chiều

Chúng ta bắt đầu với một dữ liệu vui, với đầu vào một biến, đầu ra phân loại 02 lớp (nguồn từ https://en.wikipedia.org/wiki/Logistic_regression):
Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi.
Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào?
Theo đề bài chúng ta có 02 lớp là vượt qua kỳ thi (class 1) và không qua kỳ thi (class 0).

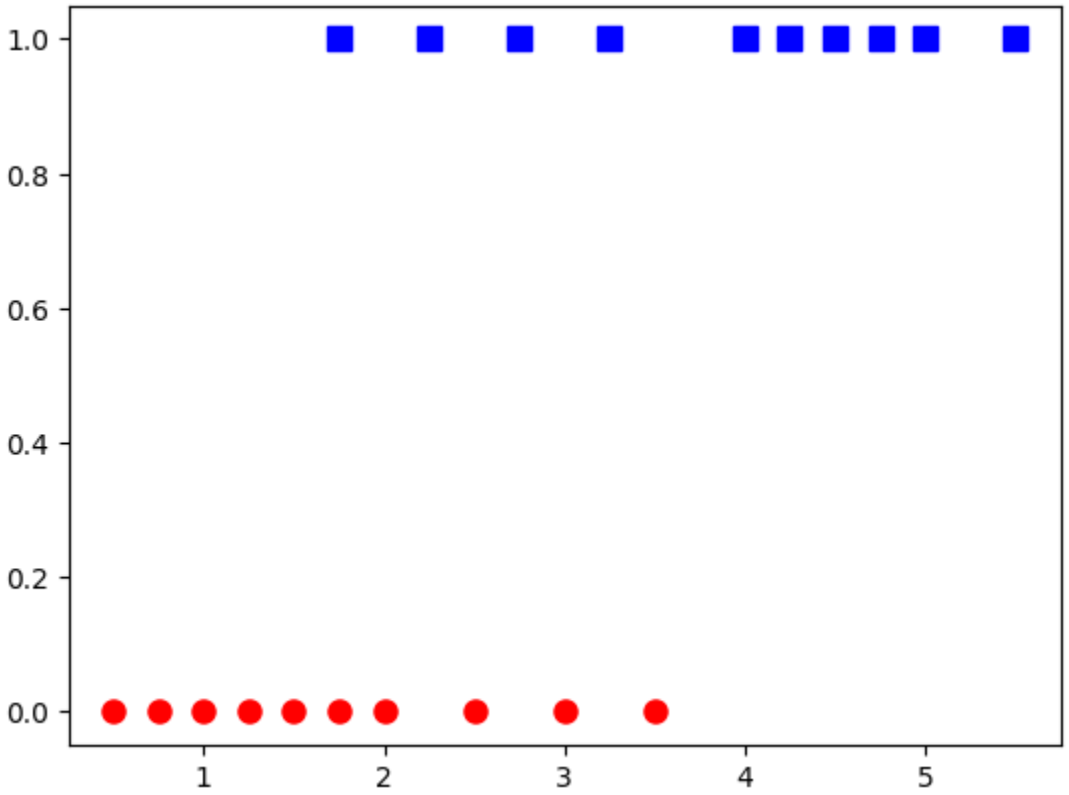
Nạp data train và test

In [ ]:
```
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)

### Training data
X_vd1_train = np.array([[0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50,
              2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50]])
y_vd1_train = np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1])

# extended data by adding a column of 1s (x_0 = 1)
X_vd1_train = np.concatenate((np.ones((1, X_vd1_train.shape[1])), X_vd1_train), axis = 0)


### Test data
X_vd1_test = np.array([[2.45, 1.85, 3.75, 3.21, 4.05]])
# extended data by adding a column of 1s (x_0 = 1)
X_vd1_test = np.concatenate((np.ones((1, X_vd1_test.shape[1])), X_vd1_test), axis = 0)
```

Cách 1. Tự xây dựng hàm thuật toán mô hình

Vẽ các điểm data trong tập training để có cảm quan

In [ ]:
```
X_vd1_0 = X_vd1_train[1, np.where(y_vd1_train == 0)][0]
y_vd1_0 = y_vd1_train[np.where(y_vd1_train == 0)]
X_vd1_1 = X_vd1_train[1, np.where(y_vd1_train == 1)][0]
y_vd1_1 = y_vd1_train[np.where(y_vd1_train == 1)]

plt.plot(X_vd1_0, y_vd1_0, 'ro', markersize = 8)
plt.plot(X_vd1_1, y_vd1_1, 'bs', markersize = 8)
plt.show()
```



Hàm tự xây dựng để training mô hình

In [ ]:
```
def sigmoid(s):
    return 1/(1 + np.exp(-s))

def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
    # method to calculate model logistic regression by Stochastic Gradient Descent method
```

```
    # eta: learning rate; tol: tolerance; max_count: maximum iterates
    w = [w_init]
    it = 0
    N = X.shape[1]
    d = X.shape[0]
    count = 0
    check_w_after = 20
    # loop of stochastic gradient descent
    while count < max_count:
        # shuffle the order of data (for stochastic gradient descent).
        # and  put into mix_id
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = y[i]
            zi = sigmoid(np.dot(w[-1].T, xi))
            w_new = w[-1] + eta*(yi - zi)*xi
            count += 1
            # stopping criteria
            if count%check_w_after == 0:
                if np.linalg.norm(w_new - w[-check_w_after]) < tol:
                    return w
            w.append(w_new)
    return w
```

Dùng hàm tự xây dựng train mô hinh với training data

In [ ]:
```
d = X_vd1_train.shape[0] # số chiều của dữ liệu input
w_init = np.random.randn(d, 1)
eta = .05

w = logistic_sigmoid_regression(X_vd1_train, y_vd1_train, w_init, eta)
```

In ra mô hình được xây dựng bằng hàm tự code

In [ ]:
```
print(w[-1])
```

```
[[-4.092695  ]
 [ 1.55277242]]
```

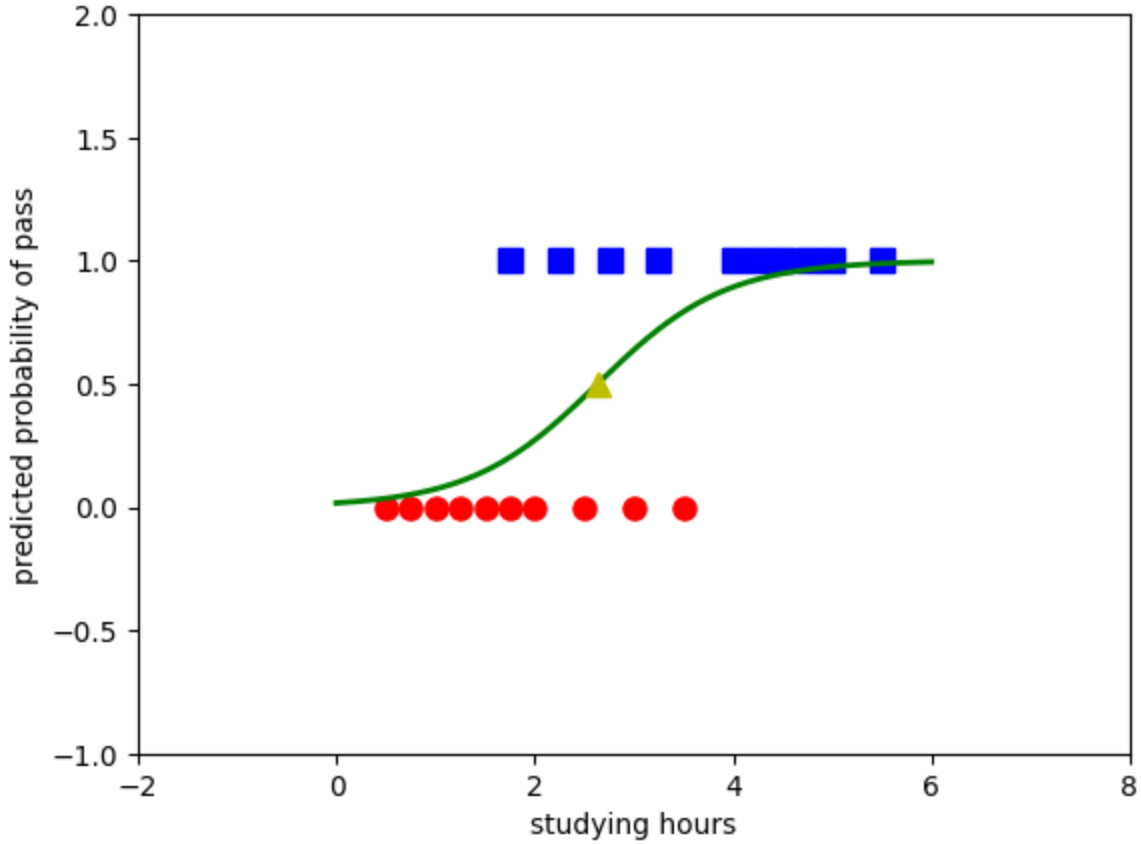
In ra xác suất rơi vào lớp 1 (pass) với dữ liệu training khi sử dụng mô hình bằng hàm tự xây dựng

In [ ]:
```
print(sigmoid(np.dot(w[-1].T, X_vd1_train)))
```

```
[[0.03501592 0.05078108 0.07310642 0.10416972 0.14634799 0.20175793
  0.20175793 0.27147534 0.35458234 0.4475032  0.54424128 0.63775242
  0.72188183 0.79282004 0.89267823 0.92460236 0.94758783 0.96384008
  0.97518471 0.9884283 ]]
```

Vẽ ra các điểm data training và hàm sigmod - mô hình xây dựng bằng hàm tự viết

In [ ]:
```
X_vd1_0 = X_vd1_train[1, np.where(y_vd1_train == 0)][0]
y_vd1_0 = y_vd1_train[np.where(y_vd1_train == 0)]
X_vd1_1 = X_vd1_train[1, np.where(y_vd1_train == 1)][0]
y_vd1_1 = y_vd1_train[np.where(y_vd1_train == 1)]

plt.plot(X_vd1_0, y_vd1_0, 'ro', markersize = 8)
plt.plot(X_vd1_1, y_vd1_1, 'bs', markersize = 8)

xx = np.linspace(0, 6, 1000)
w0 = w[-1][0][0]
w1 = w[-1][1][0]
threshold = -w0/w1
yy = sigmoid(w0 + w1*xx)
plt.axis([-2, 8, -1, 2])
plt.plot(xx, yy, 'g-', linewidth = 2)
plt.plot(threshold, .5, 'y^', markersize = 8)
plt.xlabel('studying hours')
plt.ylabel('predicted probability of pass')
plt.show()
```



So sánh giữa xác suất tính ra và đầu ra thực tế (có sự sai khác ở 1 vài trường hợp đặc biệt)

In [ ]:
```
prob_y_pred = sigmoid(np.dot(w[-1].T, X_vd1_train))

for i in range(len(X_vd1_train[0])):
    print('prob = ', round(prob_y_pred[0][i], 3), '| actual =', y_vd1_train[i])
```

```
prob =  0.035 | actual = 0
prob =  0.051 | actual = 0
prob =  0.073 | actual = 0
prob =  0.104 | actual = 0
prob =  0.146 | actual = 0
prob =  0.202 | actual = 0
prob =  0.202 | actual = 1
prob =  0.271 | actual = 0
prob =  0.355 | actual = 1
prob =  0.448 | actual = 0
prob =  0.544 | actual = 1
prob =  0.638 | actual = 0
prob =  0.722 | actual = 1
prob =  0.793 | actual = 0
prob =  0.893 | actual = 1
prob =  0.925 | actual = 1
prob =  0.948 | actual = 1
prob =  0.964 | actual = 1
prob =  0.975 | actual = 1
prob =  0.988 | actual = 1
```

Sử dụng mô hình với tập test

In [ ]:
```
# Thực hiện mô hình phân loại trên để test vs dữ liệu khác (data chưa có nhãn pass hay không)
print(sigmoid(np.dot(w[-1].T, X_vd1_test))) # p > 0.5 thì coi như vào lớp 1
```

```
[[0.42839499 0.22792483 0.8494382  0.70924179 0.89989273]]
```

Cách 2. Sử dụng thư viện

In [ ]:
```
from sklearn import linear_model

# Dinh nghia cac tham so cho ham khoi tao model
logReg = linear_model.LogisticRegression(penalty='none', max_iter=10000, fit_intercept=False)
# penalty = none vi o tren (cach 1) khong cong them ham hieu chinh R, boi vay de so sanh 2 cach thi lam nhu


# Training & compute weight
logReg.fit(X_vd1_train.T, y_vd1_train.T)

# Neu co data X_test, predict bang lenh: logReg.predict(X_test)
```

```
print(logReg.coef_)
print(logReg.predict(X_vd1_test.T))
```

```
[[-4.07771764  1.50464522]]
[0 0 1 1 1]
```

Như vậy cả 2 cách code tay và sử dụng thư viện đều cho kết quả giống nhau trên X_test

Ví dụ 2. Logistic với dữ liệu input là nhiều chiều (2 chiều)

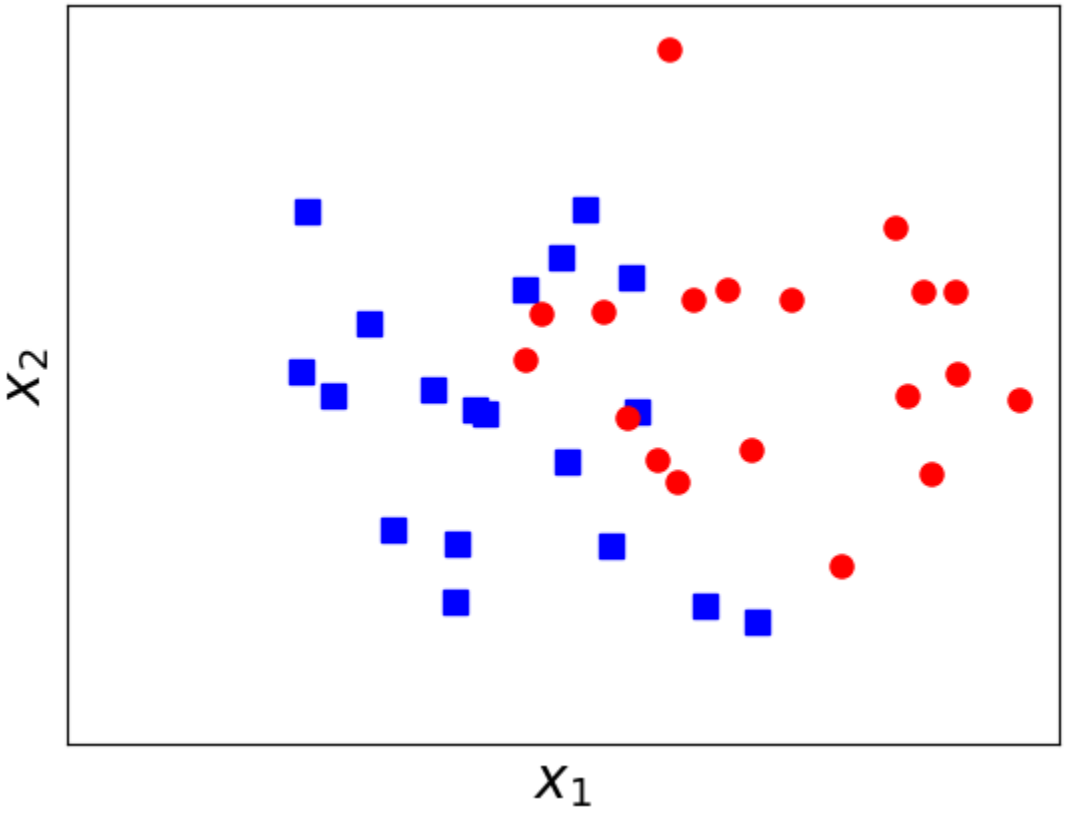
Ta tiếp tục với một dữ liệu tự tạo trong không gian hai chiều (d = 2). Số điểm dữ liệu trong ví dụ này cũng là 20. Chúng ta sẽ sử dụng độ đậm của màu để minh họa xác suất một điểm rơi vào phân lớp nào, từ đó có cái nhìn trực quan về cách phân lớp của phương pháp hồi quy Logistic.

In [ ]:
```
# Tạo dữ liệu, trong đó dòng lệnh cuối cùng ta đã tạo 2 cột ứng với tọa độ x1, x2 của dữ liệu, tức là mỗi d

from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist

# generate list of data points
np.random.seed(22)

# Khởi tạo các gía trị thống kê (ép dữ liệu sinh ra theo luật định trước)
means = [[2, 2], [4, 2]]
cov = [[.7, 0], [0, .7]]
N = 20
X1 = np.random.multivariate_normal(means[0], cov, N)
X2 = np.random.multivariate_normal(means[1], cov, N)
```

Vẽ dữ liệu minh họa

In [ ]:
```
plt.plot(X1[:, 0], X1[:, 1], 'bs', markersize = 8, alpha = 1)
plt.plot(X2[:, 0], X2[:, 1], 'ro', markersize = 8, alpha = 1)
plt.axis('equal')
plt.ylim(0, 4)
plt.xlim(0, 5)
```

```
# hide tikcs
cur_axes = plt.gca()
cur_axes.axes.get_xaxis().set_ticks([])
cur_axes.axes.get_yaxis().set_ticks([])

plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)

# save the figure to an image first
# plt.savefig('logistic_2d.png', bbox_inches='tight', dpi = 300)
plt.show()
```



Định nghĩa hàm hồi quy logistic (số chiều dữ liệu là bất kỳ) tương tự như của ví dụ 1.

In [ ]:
```
# Các hàm sigmoid và logistic này dùng tự định nghĩa như ở ví dụ 1.
```

```
# def sigmoid(s):
#     pass
# def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
#     pass
```

Bổ sung thêm cột X0 ≡1 vào bên trái để có dữ liệu Xbar = (1, X1, X2), khởi tạo xấp xỉ ban đầu cho bộ tham số w (chọn ngẫu nhiên bằng random), sau đó gọi hàm logistic_sigmoid_regression ở trên để thực hiện quá trình lặp gradient descent ngẫu nhiên tìm tham số tối ưu.

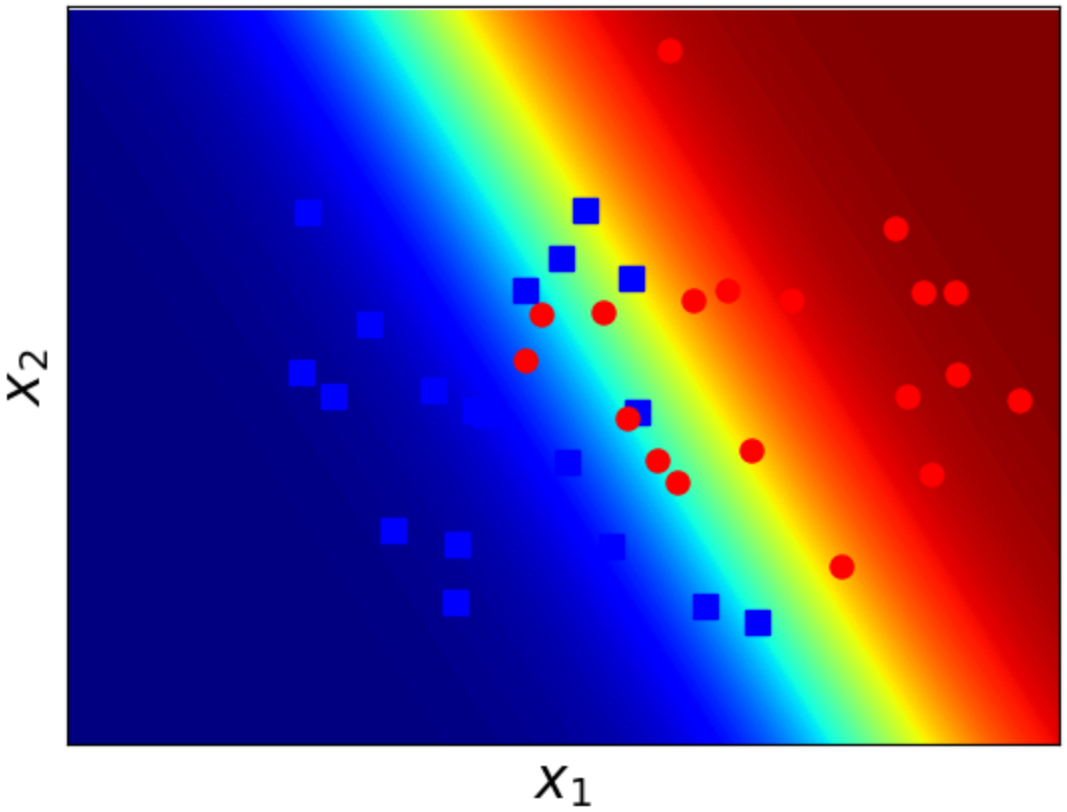
In [ ]:
```
X = np.concatenate((X1, X2), axis = 0).T
y = np.concatenate((np.zeros((1, N)), np.ones((1, N))), axis = 1).T
# Xbar
X = np.concatenate((np.ones((1, 2*N)), X), axis = 0)
eta = 0.05
d = X.shape[0]
w_init = np.random.randn(d, 1) # initialize parameters w = w_init
# call logistic_sigmoid_regression procedure
w = logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count= 10000)
# print out the parameter
print(w[-1])
```

```
[[-9.51043127]
 [ 2.25978315]
 [ 1.42588178]]
```

Trong đoạn code dưới đây, chúng ta sẽ dùng độ đậm của màu để minh họa xác suất một điểm sẽ thuộc class nào. Ở đây ta dùng 2 màu là xanh và đỏ, màu đỏ càng đậm nghĩa là xác suất điểm dữ liệu thuộc lớp đỏ càng cao; ngược lại màu xanh càng đậm nghĩa là xác suất điểm dữ liệu thuộc lớp xanh càng cao. Để thực hiện, trước hết ta sử dụng các hệ số tính được theo phương pháp hồi quy logistic với dữ liệu training ở trên. Sau đó ta tạo tập dữ liệu là toàn bộ các điểm (x1m, x2m) trong khoảng [-1, 6]×[0, 4] (thật ra ta lấy các điểm với bước lưới 0.025). Sau đó ta tính xác suất thuộc lớp 1 (đỏ) cho tất cả các điểm x = (x1m, x2m) trên lưới mà chúng ta vừa tạo ra: zm = P(c|x) = sigmoid(wTx) = sigmoid(w0 + w1*x1m + w2*x2m). Cuối cùng chúng ta dùng hàm contourf để kẻ các đường đồng mức (cùng giá trị) zm với các màu tương ứng.

In [ ]:
```
# Make data.
x1m = np.arange(-1, 6, 0.025) # generate data coord. X1
xlen = len(x1m)
x2m = np.arange(0, 4, 0.025)  # generate data coord. X2
x2en = len(x2m)
x1m, x2m = np.meshgrid(x1m, x2m) # create mesh grid X = (X1, X2)
```

```
# now assign the parameter w0, w1, w2 from array w which was computed above
w0 = w[-1][0][0]
w1 = w[-1][1][0]
w2 = w[-1][2][0]

# calculate probability zm=P(c|x)=sigmoid(w^Tx)=sigmoid(w0+w1*x1m+w2*x2m)
zm = sigmoid(w0 + w1*x1m + w2*x2m)

# plot contour of prob. zm by the saturation of blue and red
# more red <=> prob. that data point belong to red class is higher & vise versa
CS = plt.contourf(x1m, x2m, zm, 200, cmap='jet')

# finally, plot the data and take a look
plt.plot(X1[:, 0], X1[:, 1], 'bs', markersize = 8, alpha = 1)
plt.plot(X2[:, 0], X2[:, 1], 'ro', markersize = 8, alpha = 1)
plt.axis('equal')
plt.ylim(0, 4)
plt.xlim(0, 5)

# hide tikcs
cur_axes = plt.gca()
cur_axes.axes.get_xaxis().set_ticks([])
cur_axes.axes.get_yaxis().set_ticks([])

plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.savefig('logistic_2d_2.png', bbox_inches='tight', dpi = 300)
plt.show()
```



Ví dụ 3. (Bài tập 1)

Xét một ví dụ được lấy từ dữ liệu tuyển sinh sau đại học (master) của Ấn độ.
Link lấy dữ liệu ở đây: https://www.kaggle.com/mohansacharya/graduate-admissions hoặc từ tệp Admission_Predict.csv đính kèm.

Các trường dữ liệu như sau

- GRE (Graduate Record Exam) Scores (0..340): bảng điểm học tập đại học
- TOEFL Scores (0.. 120): Điểm tiếng Anh (toefl)
- University Rating (0.. 5): Điểm xếp loại đại học
- SOP (Statement of Purpose) Strength (0..5): Điểm bài viết tự giới thiệu
- LOR (Letter of Recommendation) Strength (0..5): Điểm cho thư giới thiệu

- Undergraduate GPA - CGPA (0..10): Điểm trung bình ĐH
- Research Experience (0 hoặc 1): kinh nghiệm nghiên cứu (chỉ 1 – có hoặc 0 – không)
- Chance of Admit (Số thực 0 .. 1): Khả năng được chọn

Mô tả chi tiết hơn về dữ liệu có thể tìm thấy trong link:
https://www.researchgate.net/publication/348433004_Graduate_Admission_Prediction_Using_Machine_Learning
Chú ý file online trên đường link có thể được update nên khác dữ liệu gửi kèm theo bài thực hành này. Trong tệp đính kèm, chúng ta có 400 bản ghi ứng với các hồ sơ.

Đọc tệp csv, sau đố lấy các cột dữ liệu vào mảng X1, X2, ...
Ở đây chúng ta bỏ qua trường đầu tiên ('Serial No').

In [ ]:
```
%pwd
```

Out[ ]:
```
'/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Lec_Ass/Week2_3/logistic_regression'
```

In [ ]:
```
# importing module
import numpy as np
from pandas import *

# reading CSV file
data = read_csv("data/Admission_Predict.csv")
# converting column data to list, then convert list to array
sn = data['Serial No.'].tolist()    # skip sn in bt1_X1

gre = data['GRE Score'].tolist()
bt1_X1 = np.asarray(gre)

tfl = data['TOEFL Score'].tolist()
bt1_X2 = np.asarray(tfl)

unirt = data['University Rating'].tolist()
bt1_X3 = np.asarray(unirt)

sop = data['SOP'].tolist()
bt1_X4 = np.asarray(sop)

lor1 = data['LOR '].tolist()
bt1_X5 = np.asarray(lor1)
```

```
cgpa1 = data['CGPA'].tolist()
bt1_X6 = np.asarray(cgpa1)

research_exp = data['Research'].tolist()
bt1_X7 = np.asarray(research_exp)

prob_Admit = data['Chance of Admit '].tolist() # vì trong tệp đoạn cuối là thừa 1 khoảng trắng
bt1_Y = np.asarray(prob_Admit)

# printing list data
print(bt1_X1[:5], bt1_X2[:5], bt1_X3[:5], bt1_X4[:5], bt1_X5[:5], bt1_X6[:5], bt1_X7[:5], bt1_Y[:5], sep='\
```

```
[337 324 316 322 314]
[118 107 104 110 103]
[4 4 3 3 2]
[4.5 4.  3.  3.5 2. ]
[4.5 4.5 3.5 2.5 3. ]
[9.65 8.87 8.   8.67 8.21]
[1 1 1 1 0]
[0.92 0.76 0.72 0.8  0.65]
```

Cách 1: Sử dụng hồi quy Logistic

Đọc dữ liệu, chọn ra 350 dòng đầu làm dữ liệu training, phần còn lại là dữ liệu test. Đổi các dòng dữ liệu sang dạng cột (thêm .T vào sau – đọc các đoạn code trong ví dụ trước).

Sắp xếp dữ liệu để có ma trận dữ liệu X = (X1, X2, ..., X7).

Bổ sung cột X0 ≡ 1 vào bên trái của ma trận X để được Xbar.

Và in ra hệ số đã tính ra.

In [ ]:
```
# N là số lượng bản ghi
N = len(bt1_X1)
bt1_X0 = np.ones(N)
bt1_X = np.vstack((bt1_X0, bt1_X1, bt1_X2, bt1_X3, bt1_X4, bt1_X5, bt1_X6, bt1_X7)).T
# bt1_Y = np.array([1 if num >= 0.75 else 0 for num in Yt])
# or
bt1_Y[np.where(bt1_Y >= 0.75)] = 1
```

```
bt1_Y[np.where(bt1_Y < 0.75)] = 0
print(bt1_X[:5], bt1_Y[:5], sep='\n')
```

```
[[  1.   337.   118.     4.     4.5    4.5    9.65   1.  ]
 [  1.   324.   107.     4.     4.     4.5    8.87   1.  ]
 [  1.   316.   104.     3.     3.     3.5    8.     1.  ]
 [  1.   322.   110.     3.     3.5    2.5    8.67   1.  ]
 [  1.   314.   103.     2.     2.     3.     8.21   0.  ]]
[1. 1. 0. 1. 0.]
```

In [ ]:
```
print(bt1_X.shape, bt1_Y.shape)
```

```
(400, 8) (400,)
```

In [ ]:
```
# Phân tách X_train, Y_train và X_test, Y_test
bt1_X_train = bt1_X[:350]
bt1_Y_train = bt1_Y[:350]
bt1_X_test = bt1_X[350:]
bt1_Y_test = bt1_Y[350:]
```

In [ ]:
```
print(bt1_X_train.shape, bt1_Y_train.shape)
```

```
(350, 8) (350,)
```

In [ ]:
```
eta = 0.05
d = bt1_X_train.shape[1]
w_init = np.random.randn(d, 1) # initialize parameters w = w_init
# call logistic_sigmoid_regression procedure
w = logistic_sigmoid_regression(bt1_X_train.T, bt1_Y_train, w_init, eta, tol = 1e-4, max_count= 10000)
# print out the parameter
print(w[-1])
```

```
/tmp/ipykernel_22951/211867840.py:2: RuntimeWarning: overflow encountered in exp
  return 1/(1 + np.exp(-s))
[[ -4.83725511]
 [-64.22077858]
 [213.42905896]
 [144.21116441]
 [111.1654168 ]
 [ 94.10831121]
 [ 44.56664283]
 [ 53.07172681]]
```

Sử dụng kết quả các hệ số này với dữ liệu test và so sánh với dữ liệu thực tế.

Tính toán các đại lượng: Accuracy, Precision, Recall

In [ ]:
```
bt1_y_pred = sigmoid(np.dot(w[-1].T, bt1_X_test.T)).T
bt1_y_pred[np.where(bt1_y_pred >= 0.75)] = 1
bt1_y_pred[np.where(bt1_y_pred < 0.75)] = 0
```

In [ ]:
```
print(bt1_y_pred.T, np.array(bt1_Y_test))
```

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1.]] [0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1.
 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1.
 0. 1.]
```

In [ ]:
```
# Chuyển về dạng số nguyên để tính toán
bt1_y_pred = np.round(bt1_y_pred).astype(int).T

# Tính các giá trị True Positive, False Positive, True Negative, False Negative
TP = np.sum((bt1_y_pred == 1) & (bt1_Y_test == 1))
FP = np.sum((bt1_y_pred == 1) & (bt1_Y_test == 0))
TN = np.sum((bt1_y_pred == 0) & (bt1_Y_test == 0))
FN = np.sum((bt1_y_pred == 0) & (bt1_Y_test == 1))

# Tính Accuracy, Precision và Recall
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.44
Precision: 0.44
Recall: 1.0
```

Cách 2: Sử dụng hồi quy tuyến tính

In [ ]:
```
prob_Admit = data['Chance of Admit '].tolist() # vì trong tệp đoạn cuối là thừa 1 khoảng trắng
```

```
# Không chuyển Yt sang 0 và 1 nữa
bt1_Y_train_linear = np.asarray(prob_Admit)

# Phân tách X_train, Y_train và X_test, Y_test
linear_bt1_X_train = bt1_X[:350]
linear_bt1_Y_train = bt1_Y_train_linear[:350]
linear_bt1_X_test = bt1_X[350:]
linear_bt1_Y_test = bt1_Y_train_linear[350:]
```

In [ ]:
```
print(linear_bt1_X_train.shape, linear_bt1_Y_train.shape)
```

```
(350, 8) (350,)
```

In [ ]:
```
from sklearn import linear_model
# fit the model by Linear Regression
regr = linear_model.LinearRegression(fit_intercept=False)
# fit_intercept = False for calculating the bias
# Building Xbar
# regr.fit(x_train.reshape(-1, 1), y_train.reshape(-1, 1))
regr.fit(linear_bt1_X_train, linear_bt1_Y_train)
print('Coef:', regr.coef_)
print('Intercept:', regr.intercept_)
```

```
Coef: [-1.19503649  0.00154891  0.00324646  0.00931544 -0.00438297  0.02493287
  0.11235144  0.02194172]
Intercept: 0.0
```

In [ ]:
```
linear_bt1_y_pred = regr.predict(linear_bt1_X_test)
```

In [ ]:
```
print(linear_bt1_y_pred[:5], linear_bt1_Y_test[:5])
```

```
[0.69804036 0.78315311 0.61917215 0.59362637 0.52735201] [0.74 0.73 0.64 0.63 0.59]
```

In [ ]:
```
error = sum([(linear_bt1_y_pred[i] - linear_bt1_Y_test[i]) ** 2 for i in range(len(linear_bt1_y_pred))]) /
print(error)
```

```
0.0046416784403671306
```

Có vẻ sử dụng mô hình hồi quy tuyến tính sẽ cho kết quả dự đoán chính xác hơn.

Cách 3: Sử dụng phương pháp Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
import time

# Fit model with GaussianNB
start_time = time.time()
naivebayes_model = GaussianNB(priors = None)
naivebayes_model.fit(bt1_X_train, bt1_Y_train)
end_time = time.time()
execution_time = end_time - start_time
print('Time run by Naive Bayes:', execution_time)

# Fit model with Logistic
start_time = time.time()
logistic_model = linear_model.LogisticRegression(penalty="none")     # không có hàm phạt hiệu chỉnh.
logistic_model.fit(bt1_X_train, bt1_Y_train)
end_time = time.time()
execution_time = end_time - start_time
print('Time run by Logistic:', execution_time)

# Đánh giá mô hình sử dụng thư viện
naivebayes_y_pred = naivebayes_model.predict(bt1_X_test)
logistic_y_pred = logistic_model.predict(bt1_X_test)

naivebayes_report = classification_report(bt1_Y_test, naivebayes_y_pred)
logistic_report = classification_report(bt1_Y_test, logistic_y_pred)
print("NaiveBayes Report:")
print(naivebayes_report)
print("Logistic Report:")
print(logistic_report)
```

```
Time run by Naive Bayes: 0.0016677379608154297
Time run by Logistic: 0.04525160789489746
NaiveBayes Report:
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.92        28
         1.0       0.95      0.82      0.88        22

    accuracy                           0.90        50
   macro avg       0.91      0.89      0.90        50
weighted avg       0.90      0.90      0.90        50

Logistic Report:
              precision    recall  f1-score   support

         0.0       0.84      0.96      0.90        28
         1.0       0.94      0.77      0.85        22

    accuracy                           0.88        50
   macro avg       0.89      0.87      0.88        50
weighted avg       0.89      0.88      0.88        50


/home/harito/venv/py/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning
: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

So sánh qua về thời gian và các chỉ số đánh giá thì Naive Bayes tỏ ra tốt hơn.

Làm lại ví dụ 1 và 2 nhưng Sử dụng thư viện Scikit-Learn cho mô hình Logistic Regression và so sánh với cách viết hàm tự lập trình xây dựng.

In [ ]:
```
# Sử dụng thư viện scikit-learn
from sklearn import linear_model
```

```
logReg = linear_model.LogisticRegression(penalty="none")     # không có hàm phạt hiệu chỉnh.

# Training & compute weight
logReg.fit(X_vd1_train.T, y_vd1_train.T)

# Neu co data X_test, predict bang lenh: logReg.predict(X_test)
print(logReg.coef_)
print(logReg.predict(X_vd1_test.T))
```

```
[[-2.03885791  1.50464797]]
[0 0 1 1 1]
```

Trọng số mô hình khác nhau so với ở ví dụ trên. Cụ thể là hệ số tự do. Lý do là sử dụng thư viện như này có quan tâm đến hệ số tự do.

Ví dụ 4. (Bài tập 2)

Ngân hàng Bồ Đào Nha đã có sự sụt giảm doanh thu và họ muốn biết cần thực hiện những cải cách gì.
Sau khi điều tra, họ phát hiện ra rằng nguyên nhân sâu xa là do khách hàng của họ không đầu tư đủ cho các khoản tiền gửi dài hạn. Vì vậy, ngân hàng muốn xác định những khách hàng hiện tại có khả năng đăng ký tiền gửi dài hạn cao hơn và tập trung nỗ lực tiếp thị vào những khách hàng đó.
Trong tệp banking.csv đính kèm có 41188 bản ghi liên quan đến thông tin khách hàng, gồm 20 trường dữ liệu quan sát và 01 trường y là đầu ra (dự báo) tương ứng: y = 1 nếu khách hàng CÓ đăng ký tiền gửi dài hạn; y = 0 nếu ngược lại.

Xử lý data

Trước hết cần chuyển các trường dữ liệu kiểu text-categories/nomial (dữ liệu dạng phân loại/danh nghĩa) sang dạng danh sách các số định danh.

Ví dụ:

- yes/no chuyển thành 1/0;
- 'jan', 'feb', 'mar'... chuyển thành 1, 2, 3... Chú ý có 2 loại dữ liệu Categories/Nomial.

Loại thứ nhất như yes/no; month; dayofweek ta có thể chuyển thành các số 0, 1, 2...

Loại thứ 2 như education; marital... chúng ta cần chuyển thành dạng vector kiểu như trong bag of words, tức là lập vector có số chiều = tổng số danh nghĩa, sau đó ứng với danh nghĩa nào, ta đặt vị trí thành phần đó = 1, còn lại đặt là 0 (one host coding)

In [ ]:
```
import pandas as pd
data = pd.read_csv("data/banking.csv")
data.head()

#job :
dict_job = {'admin.': 1, 'blue-collar': 2, 'entrepreneur': 3, 'housemaid': 4, 'management': 5, 'retired': 6
data['job'] = data['job'].map(dict_job)

#marital :
dict_marital = {'divorced': 1, 'married': 2, 'single': 3, 'unknown': -1}
data['marital'] = data['marital'].map(dict_marital)

#education :
dict_education = {'basic.4y': 1, 'basic.6y': 2, 'basic.9y': 3, 'high.school': 4, 'illiterate': 5, 'professi
data['education'] = data['education'].map(dict_education)

#default :
dict_default = {'no' : 0, 'yes' : 1, 'unknown': -1}
data['default'] = data['default'].map(dict_default)
#housing :
dict_housing = {'no' : 0, 'yes' : 1, 'unknown': -1}
data['housing'] = data['housing'].map(dict_housing)
#loan :
dict_loan = {'no' : 0, 'yes' : 1, 'unknown': -1}
data['loan'] = data['loan'].map(dict_loan)

#contact :
dict_contact = {'cellular': 1,'telephone': 2}
data['contact'] = data['contact'].map(dict_contact)
```

```
#month :
dict_month = {'jan' : 1, 'feb' : 2, 'mar' : 3, 'apr' : 4, 'may' : 5, 'jun' : 6, 'jul' : 7, 'aug' : 8, 'sep'
data['month'] = data['month'].map(dict_month)
#day_of_week :
dict_day = {'sun' : 1, 'mon' : 2, 'tue' : 3, 'wed' : 4, 'thu' : 5, 'fri' : 6, 'sat' : 7}
data['day_of_week'] = data['day_of_week'].map(dict_day)

#poutcome :
dict_poutcome = {'failure': 1, 'nonexistent': 2, 'success': 3}
data['poutcome'] = data['poutcome'].map(dict_poutcome)
```

In [ ]:
```
print(data.columns)
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
       'cons_conf_idx', 'euribor3m', 'nr_employed', 'y'],
      dtype='object')
```

In [ ]:
```
print(data[:5])
```

```
   age  job  marital  education  default  housing  loan  contact  month  \
0   44    2        2          1       -1        1     0        1      8
1   53   10        2         -1        0        0     0        1     11
2   28    5        3          7        0        1     0        1      6
3   39    8        2          4        0        0     0        1      4
4   55    6        2          1        0        1     0        1      8

   day_of_week  ...  campaign  pdays  previous  poutcome  emp_var_rate  \
0            5  ...         1    999         0         2           1.4
1            6  ...         1    999         0         2          -0.1
2            5  ...         3      6         2         3          -1.7
3            6  ...         2    999         0         2          -1.8
4            6  ...         1      3         1         3          -2.9

   cons_price_idx  cons_conf_idx  euribor3m  nr_employed  y
0          93.444          -36.1      4.963       5228.1  0
1          93.200          -42.0      4.021       5195.8  0
2          94.055          -39.8      0.729       4991.6  1
3          93.075          -47.1      1.405       5099.1  0
4          92.201          -31.4      0.869       5076.2  1

[5 rows x 21 columns]
```

Phân chia Train Test

In [ ]:
```
# Phân chia dữ liệu thành Training và Test
from sklearn.model_selection import train_test_split

X = data.drop('y', axis=1)  # Features
y = data['y']  # Target variable

# Phân chia dữ liệu thành Training và Test theo tỷ lệ 8:2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train model và so sánh time, accuracy sau test

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
import time

# a) Sử dụng mô hình hồi quy logistic
start_time = time.time()
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)
end_time = time.time()

acc_logreg = accuracy_score(y_test, y_pred_logreg)
precision_logreg = precision_score(y_test, y_pred_logreg)
recall_logreg = recall_score(y_test, y_pred_logreg)
f1_logreg = f1_score(y_test, y_pred_logreg)
time_logreg = end_time - start_time

print("Logistic Regression:")
print("Accuracy:", acc_logreg)
print("Precision:", precision_logreg)
print("Recall:", recall_logreg)
print("F1-score:", f1_logreg)
print("Time:", time_logreg)

# b) Sử dụng mô hình Naïve Bayes
start_time = time.time()
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
end_time = time.time()

acc_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb)
recall_nb = recall_score(y_test, y_pred_nb)
f1_nb = f1_score(y_test, y_pred_nb)
time_nb = end_time - start_time

print("\nNaive Bayes:")
print("Accuracy:", acc_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("F1-score:", f1_nb)
print("Time:", time_nb)
```

```
# c) So sánh thời gian chạy và độ chính xác của hai phương pháp
print("\nComparison:")
print("Logistic Regression Time:", time_logreg)
print("Naive Bayes Time:", time_nb)
print("Logistic Regression Accuracy:", acc_logreg)
print("Naive Bayes Accuracy:", acc_nb)
```

```
Logistic Regression:
Accuracy: 0.9058023792182569
Precision: 0.6412859560067682
Recall: 0.4019088016967126
F1-score: 0.49413298565840935
Time: 1.7324955463409424

Naive Bayes:
Accuracy: 0.8498421947074533
Precision: 0.40134228187919463
Recall: 0.6341463414634146
F1-score: 0.4915741882449651
Time: 0.10058188438415527

Comparison:
Logistic Regression Time: 1.7324955463409424
Naive Bayes Time: 0.10058188438415527
Logistic Regression Accuracy: 0.9058023792182569
Naive Bayes Accuracy: 0.8498421947074533
/home/harito/venv/py/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning
: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Như vậy cách dùng Logistic tốn nhiều thời gian hơn nhưng mặt khác lại có Accuracy cao hơn.

Ví dụ 5. (Bài tập 3)

Giới thiệu bài toán

Chúng ta sử dụng dữ liệu chứa các thông tin về nhân khẩu, hành vi thói quen và lịch sử bệnh lý để xác định khả năng bị trụy tim ở một bệnh nhân. Bộ dữ liệu được cung cấp công khai trên trang web Kaggle và nó là từ một nghiên cứu tim mạch đang diễn ra trên cư dân của thị trấn Framingham, Massachusetts. Mục tiêu phân loại là dự đoán liệu bệnh nhân có nguy cơ mắc bệnh tim mạch vành (CHD) trong tương lai (10 năm) hay không. Bộ dữ liệu cung cấp thông tin của bệnh nhân, có thể tìm thấy tại link https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset/data hoặc lấy từ tệp đính kèm framingham.csv. Nó bao gồm hơn 4.000 bản ghi và 15 thuộc tính.

Giới thiệu data

Mỗi thuộc tính là một yếu tố rủi ro tiềm ẩn. Có cả các yếu tố rủi ro về nhân khẩu học, hành vi và y tế.

Nhân khẩu học:
• Giới tính: male - nam hoặc nữ (nomial)
• Tuổi: age - Tuổi của bệnh nhân;(Liên tục - Mặc dù tuổi được ghi đã bị cắt bớt thành số nguyên nhưng khái niệm tuổi là liên tục)

Hành vi
• Người hút thuốc hiện tại: currentSmoke - bệnh nhân có hút thuốc hay không (Nomial)
• Cigs Per Day: số điếu thuốc mà một người hút trung bình trong một ngày. (có thể coi là liên tục vì một người có thể hút bao nhiêu điếu, thậm chí nửa điếu.)

Tiền sử y tế/bệnh
• Thuốc BP: BPMed - bệnh nhân có dùng thuốc huyết áp hay không (yes/no)
• Đột quỵ trước đó: Prevalent Stroke - trước đó bệnh nhân có bị đột quỵ hay không (yes/no)
• Prevalent Hyp: bệnh nhân có tăng huyết áp hay không (yes/no)
• Đái tháo đường: Diabetes - bệnh nhân có bị đái tháo đường hay không (Danh định)

Tình trạng y tế (hiện tại)
• Tot Chol: mức cholesterol toàn phần (Liên tục)
• Sys HA: huyết áp tâm thu (Liên tục)
• Dia BP: huyết áp tâm trương (Liên tục)

• BMI: Chỉ số khối cơ thể (Liên tục)
• Heart Rate: nhịp tim (Liên tục - Trong nghiên cứu y học, các biến số như nhịp tim mặc dù trên thực tế là rời rạc, nhưng vẫn được coi là liên tục vì số lượng lớn các giá trị có thể có.)
• Glucose: mức glucose (Liên tục)

Biến dự đoán (đầu ra y)
• Nguy cơ mắc bệnh mạch vành CHD trong 10 năm (nhị phân: "1", nghĩa là "Có", "0" nghĩa là "Không")

Xử lý data

Một số trường dữ liệu chứa giá trị lỗi N/A. Giả sử dữ liệu đã được đọc vào đối tượng df (pandas data). Câu lệnh sau thống kê các trường có N/A trong dữ liệu:

In [ ]:
```
import pandas as pd
df = pd.read_csv("data/framingham.csv")
df.head()
df.isnull().sum()
```

Out[ ]:
```
male               0
age                0
education        105
currentSmoker      0
cigsPerDay        29
BPMeds            53
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol           50
sysBP              0
diaBP              0
BMI               19
heartRate          1
glucose          388
TenYearCHD         0
dtype: int64
```

Đoạn lệnh dưới đây loại bỏ các thành phần N/A trong dữ liệu:

In [ ]:
```
df = df.dropna(how="any", axis=0)
```

Chia dữ liệu thành các tập Training – Validation theo tỉ lệ 7:3

In [ ]:
```
print(df[:3])
```

```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39        4.0              0         0.0     0.0                0
1     0   46        2.0              0         0.0     0.0                0
2     1   48        1.0              1        20.0     0.0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0

   TenYearCHD
0           0
1           0
2           0
```

In [ ]:
```
from sklearn.model_selection import train_test_split

# Chia dataframe thành features (X) và target variable (y)
bt3_X = df.drop('TenYearCHD', axis=1)
bt3_y = df['TenYearCHD']

# Chia dữ liệu thành tập train và tập validation theo tỷ lệ 7:3
bt3_X_train, bt3_X_valid, bt3_y_train, bt3_y_valid = train_test_split(bt3_X, bt3_y, test_size=0.3, random_s

# Xem số lượng bản ghi trong mỗi tập
print("Số lượng bản ghi trong tập train:", len(bt3_X_train))
print("Số lượng bản ghi trong tập validation:", len(bt3_X_valid))
```

```
Số lượng bản ghi trong tập train: 2559
Số lượng bản ghi trong tập validation: 1097
```

Sử dụng mô hình hồi quy logistic đã huấn luyện trên tập Training để dự đoán cho tập Validation.

Training model

In [ ]:
```
from sklearn.linear_model import LogisticRegression

# Khởi tạo mô hình hồi quy logistic
bt3_logreg_model = LogisticRegression()

# Huấn luyện mô hình trên tập train
bt3_logreg_model.fit(bt3_X_train, bt3_y_train)
```

```
/home/harito/venv/py/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning
: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]:
```
▾ LogisticRegression

LogisticRegression()
```

Dự đoán trên tập Valid

In [ ]:
```
# Dự đoán trên tập validation
bt3_y_pred_valid = bt3_logreg_model.predict(bt3_X_valid)
print(bt3_y_pred_valid[:5])
```

```
[0 0 0 0 0]
```

Độ chính xác của mô hình theo các độ đo Accuracy, Recall và Precision.

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Tính toán accuracy
bt3_accuracy = accuracy_score(bt3_y_valid, bt3_y_pred_valid)
```

```
# Tính toán precision
bt3_precision = precision_score(bt3_y_valid, bt3_y_pred_valid)

# Tính toán recall
bt3_recall = recall_score(bt3_y_valid, bt3_y_pred_valid)

print("Accuracy:", bt3_accuracy)
print("Precision:", bt3_precision)
print("Recall:", bt3_recall)
```

```
Accuracy: 0.8413855970829535
Precision: 0.5
Recall: 0.034482758620689655
```