Sử dụng Machine Learning

Trí tuệ tính toán và ứng dụng

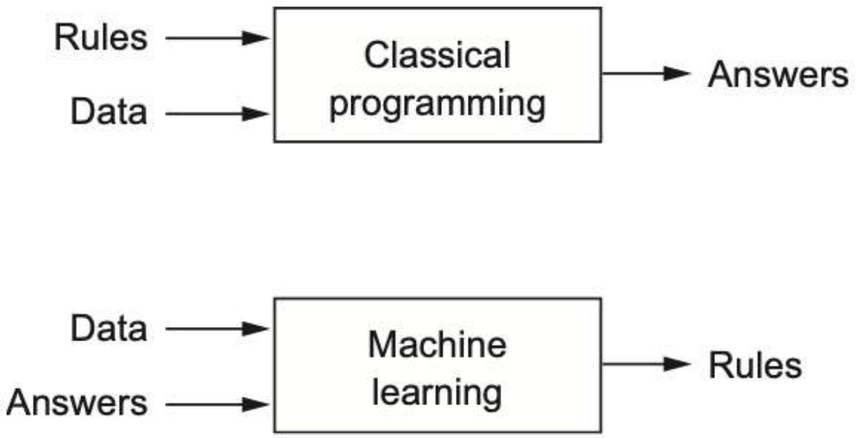
A.PROF. TRAN VAN LANG, PHD - VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY

Giới thiệu chung

- Ngày nay, nhìn ở góc độ lý luận thì lĩnh vực Công nghệ thông tin có xu hướng chuyển dần từ Khoa học máy tính (*Computer Science*) sang Khoa học dữ liệu (*Data Science*)
- Điều đó cho thấy chiếc máy tính (*Computer*) đã trở thành công cụ tất yếu trong cuộc sống, không còn là đối tượng nghiên cứu.
- Mà Dữ liệu (*Data*) mới chính là đối tượng nghiên cứu.
- Như vậy, có sự chuyển dần từ những nghiên cứu về sự lưu trữ và xử lý của máy tính, sang những nghiên cứu về nội hàm của dữ liệu.
- Bởi bản thân trong dữ liệu đang tồn tại đã hàm chứa thông tin từ đó có thể rút ra quy tắc dùng trong việc xử lý.

- Quy tắc này có thể tường minh dưới dạng của một hàm dự đoán hoặc được rút ra một cách âm thầm để máy tính xử lý dưới dạng một mô hình (Model).
- Hình ảnh đơn giản của bài toán đặt ra trong **Khoa học dữ liệu** đó là:
  - Từ input là 2 tập dữ liệu *X*, *Y* quan hệ với nhau dạng *X* là tiền đề (*Antecedent*), còn *Y* là kết quả (*Consequent*)
  - Hãy suy ra output *f* một cách tự nhiên được ẩn chứa dạng $f : X \rightarrow Y$
  - Đó chính là quy tắc mà trong học máy thường hay gọi là mô hình
- Sau khi đã có được *f*, quay trở về nhờ máy tính để tiếp tục xử lý (**Khoa học máy tính**) với input là *X'*, cùng *với f* đã tìm được cần tìm output là *Y'*.

- Từ đây cho ta thấy để giải quyết vấn đề của khoa học dữ liệu, vai trò của Machine Learning (ML) vô cùng quan trọng, qua đó cho thấy sự khác biệt với cách tiếp cận trước đây về lập trình.
- Đó là ML giúp để tìm ra quy tắc hay mô hình đã tồn tại trong dữ liệu
- Francois Chollet – tác giả của thư viện Keras và là người có đóng góp lớn cho TensorFlow cũng đã có những đúc kết hình ảnh bên dưới mà trong đó Rules chính là quy tắc hay mô hình trong quyển sách "*Deep Learning with Python*" của mình.



"ML: a new programming paradigm"

- Nhiệm vụ của Machine Learning (ML) là làm sao giúp máy tính tìm ra mô hình này. Chính vì vậy mà người ta coi ML là một mẫu (hay rộng hơn là *một luận thuyết*) lập trình mới.
- ML phân thành 2 nhóm thuật toán chính. Sự khác nhau giữa 2 nhóm *này là việc cung cấp tập dữ liệu huấn luyện*; *cách thuật toán sử dụng dữ liệu* và *loại vấn đề mà thuật toán giải quyết*.
  - Học có giám sát (Supervised Learning)
  - Học không giám sát (Unsupervised Learning)

- Học có giám sát là việc có input là một tập nguồn và một tập đích tương ứng (gọi là đã được gán nhãn) để làm cơ sở tìm ra output là hàm hay mô hình (model) mong muốn.
- Tập hợp kết hợp bởi hai tập này được gọi là tập huấn luyện (Training Set)
- Khi thực hiện, dữ liệu mới lại được sử dụng để huấn luyện. Việc huấn luyện lại nhằm cải tiến mô hình này hơn nữa

Supervised Learning

- Trong học có giám sát, căn cứ vào dữ liệu của tập đích; có thể phân thành 2 dạng bài toán:
  - Phân lớp (Classification): Khi bài toán mà dữ liệu của tập đích là hữu hạn và có gán nhãn. Chẳng hạn,
    - Phân nhóm học sinh vào trường dựa vào bảng tên
  - Hồi quy hay tiên lượng (Regression): Khi dữ liệu tập đích có tính liên tục và nối tiếp nhau. Chẳng hạn,
    - Dự đoán biến động của chứng khoán dựa trên dữ liệu của những năm trước
    - Tuyển dụng nhân sự dựa vào số liệu đã tuyển dụng

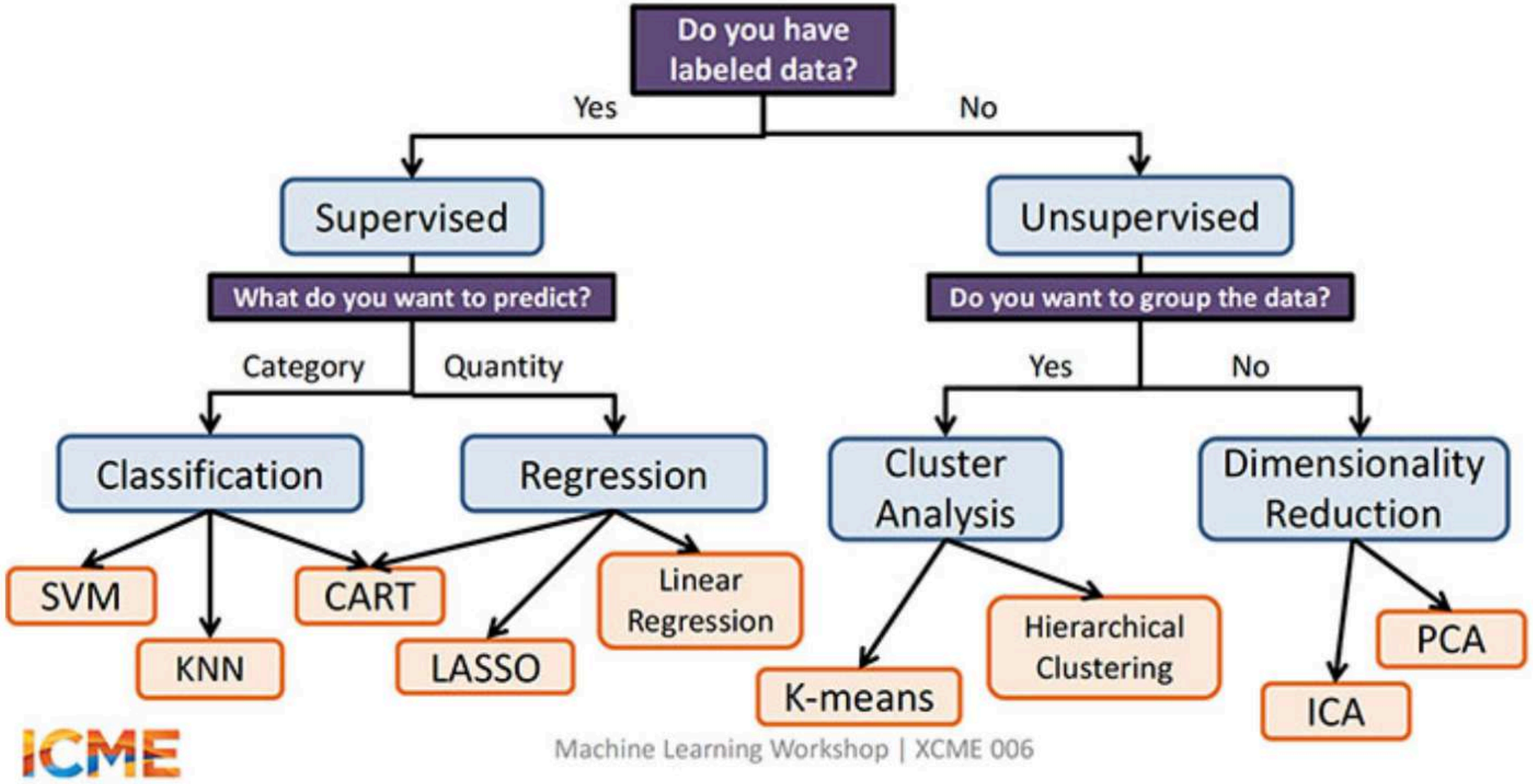
Unsupervised Learning

- Học không giám sát phải sử dụng khi không có tập đích mà chỉ có tập nguồn và các đặc trưng của tập nguồn này.
- Nhằm khám phá cấu trúc và mối quan hệ của dữ liệu
- Học không giám sát còn gọi là học không được sự chỉ bảo.
- Khi thực hiện, dữ liệu mới được suy diễn để thêm vào.
- Từ đó đưa về việc học có giám sát.
- Chính vì vậy ranh giới giữa học giám sát và không giám sát đôi khi cũng không được phân chia rõ ràng.
  - Có thể gọi trong trường hợp này là học bán giám sát (Semi-Supervised Learning)

- Các bài toán học không giám sát cũng được chia làm hai loại dựa vào dữ liệu tập nguồn:
  - Gom cụm (Clustering): Khi tập nguồn chứa một số đặc tính có thể nhận dạng được (gọi là các đặc trưng - feature hay các category). Chẳng hạn,
    - Gom email vào thư mục spam dựa trên cách đặt tên, nơi gửi, tần suất xuất hiện của nơi gửi, một số nội dung
  - Rút gọn hay ước lượng (Reduction, Estimation): nhằm để giảm bớt độ phức tạp của bài toán

Thuật toán trong Machine Learning

- Phân cấp các thuật toán trong Machine Learning



Lựa chọn thuật toán machine learning phù hợp

Các phương pháp hồi quy

- Hồi quy (Regression) hay tiên lượng là phương pháp mà mối quan hệ giữa tập nguồn và tập đích là quan hệ thể hạng dưới dạng hàm.
- Trong phương pháp tính, thực chất là phép tính nội suy, phương pháp bình phương tối thiểu (Mean Least Square)
- Bao gồm hồi quy có thể phân loại theo góc nhìn.
- Chẳng hạn, khi quan tâm tới hệ số hàm xấp xỉ xuất hiện trong biểu thức, có dạng
  - Hồi quy tuyến tính
  - Hồi quy phi tuyến
- Hoặc quan tâm đến cách giải, có: hồi quy đa thức, hồi quy logistic, hồi quy Bayesian, ...

Bài toán hồi quy tổng quát

- Bài toán:
  - Cho bảng gồm N bộ giá trị $(X_i, y_i), \forall i = \overline{1,N}$ và một hàm $y = \varphi(X, w_0, w_1, \ldots, w_M)$, trong đó X có thể 1 hoặc nhiều biến.
  - Hãy tìm các hệ số $w_0, w_1, \ldots, w_M$ sao cho hàm φ xấp xỉ tốt nhất các giá trị y_i theo nghĩa trung bình phương.
- Hồi quy tuyến tính hay phi tuyến liên quan đến trường hợp φ là hàm tuyến tính hay phi tuyến theo X.

Phương pháp giải

- Bài toán được đưa về dạng:
  - Gọi L là hàm mất mát

$$L(w_0, w_1, \ldots, w_M) = \sum_{i=1}^{N} \left[y_i - \varphi(X_i, w_0, w_1, \ldots, w_M) \right]^2$$

  - Hãy tìm

$$\arg \min_{(w_0, w_1, \ldots, w_M)} L(w_0, w_1, \ldots, w_M)$$

- Bài toán được đưa về tìm các $w_0, w_1, \ldots, w_M$ thoả hệ M+1 phương trình:

$$\begin{cases} \frac{\partial L}{\partial w_0} = 0 \\ \ldots \\ \frac{\partial L}{\partial w_M} = 0 \end{cases}$$

- Hay phương trình

$$\begin{cases} \sum_{i=1}^{N} [y_i - \varphi(X_i, w_0, w_1, \ldots, w_M)] \frac{\partial \varphi}{\partial w_0} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{N} [y_i - \varphi(X_i, w_0, w_1, \ldots, w_M)] \frac{\partial \varphi}{\partial w_M} = 0 \end{cases}$$

Hồi quy tuyến tính (Linear Regression)

- Hồi quy tuyến tính (hay *Ordinary Least Squares*) là phương pháp mà mối quan hệ giữa input và output là quan hệ tuyến tính.
- Trong Toán học gọi đó là phương pháp bình phương tối thiểu tuyến tính (*Linear Least Square*), cón trong Thống kê gọi đó là phương pháp hiệu chỉnh tuyến tính (*Linear Fitting*)
- Giả sử hàm xấp xỉ có dạng tuyến tính theo biến X

$$\varphi(X, w_0, w_1) = w_0 + w_1 X$$

- Ở đây có thể $X = [x]^T$ chỉ là vector có 1 thành phần, và hàm xấp xỉ là $\varphi(x, w_0, w_1) = w_0 + w_1 x$

- Trong trường hợp này hệ phương trình để tìm w_0, w_1 là

$$\begin{cases} \sum_{i=1}^{N} (y_i - w_0 - w_1 x)i & = 0 \\ \sum_{i=1}^{N} (y_i - w_0 - w_1 x_i)x_i & = 0 \end{cases}$$

- Hay

$$\begin{cases} \sum_{i=1}^{N} w_0 & + \sum_{i=1}^{N} x_i w_1 = \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i w_0 & + \sum_{i=1}^{N} x_i^2 w_1 = \sum_{i=1}^{N} x_i y_i \end{cases}$$

Hồi quy tuyến tính

- Viết lại dưới dạng ma trận

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix}$$

- Đặt

$$A = \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}, b = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix}$$

- Hệ phương trình viết lại thành

$$A \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = b$$

- Suy ra nghiệm cần tìm w_0, w_1 là

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = A^{-1} b$$

- $\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}$

- Để xây dựng thuật toán, ta đặt

$$X = \begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_N \end{bmatrix}$$

- Suy ra

$$XX^T = \begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_N \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}$$

- Hay

$$A = XX^T$$

- Đồng thời

$$\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix} = X \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

- Nên

$$b = X \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Dùng Python

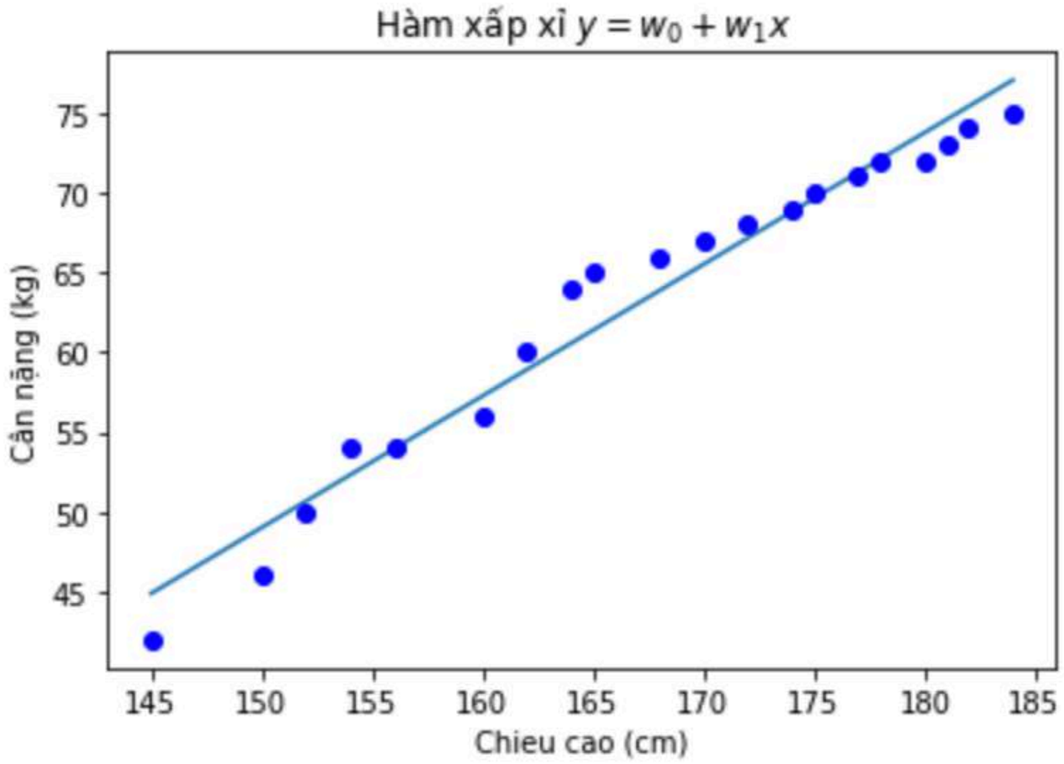
- Ví dụ chiều cao và cân nặng của 20 người cho trong bảng, hãy xây dựng hàm dự đoán để tính cân nặng của một người dựa trên chiều cao của họ.

STT	Cao	Nặng	STT	Cao	Nặng	STT	Cao	Nặng
1	145	42	8	164	64	15	177	71
2	150	46	9	165	65	16	178	72
3	152	50	10	168	66	17	180	72
4	154	54	11	170	67	18	181	73
5	156	54	12	172	68	19	182	74
6	160	56	13	174	69	20	184	75
7	162	60	14	175	70			

- Để làm điều này, giả sử quan hệ giữa chiều cao x và cân nặng y là quan hệ tuyến tính $y = w_0 + w_1 x$.

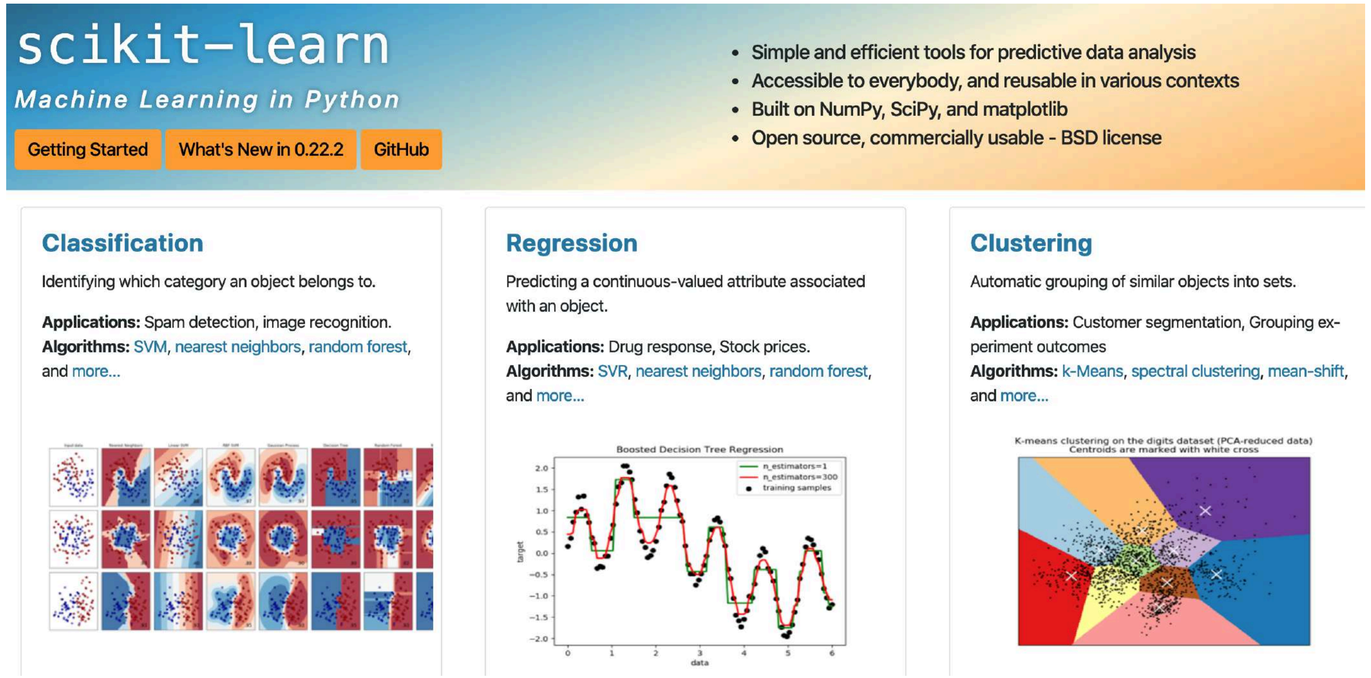
```
import numpy as np

### Bảng dữ liệu gồm x và y
x = np.array([[145,150,152,154,156,160,162,164,
165,168,170,172,174,175,177,178,180,181,182,184]]).T
y =[42,46,50,54,54,56,60,64,65,66,67,68,69,70,71,
72, 72,73,74,75]

##### Tạo ma trận X
N = len(x)
onesMat = np.ones( (1,N) )
X = np.concatenate((onesMat, x.T), axis = 0)

### Gom lại để thành ma trận A và vector b
A = np.dot( X, X.T )
b = np.dot( X, y )

### Giải hệ phương trình để tìm w
w = np.dot( np.linalg.pinv(A), b )
w0 = w[0][0]
w1 = w[1][0]
```

```
### Vẽ lại đồ thị của y = w0 + w1x

import matplotlib.pyplot as plt
### Tạo ra 2 dữ liệu trong đoạn [145,184] để
lưu vào mảng x0
x0 = np.linspace( 145, 184, 2 )
### Tính toán để có mảng y0 tương ứng với x0
dựa vào hàm tuyến tính đã tìm ra
y0 = w0 + w1*x0
plt.plot( x0, y0 )
plt.plot( x, y, 'bo' )
plt.title( "Hàm xấp xỉ " + "$y=w_0+w_1x$" )
plt.xlabel( "Chieu cao (cm)" )
plt.ylabel( "Cân nặng (kg)" )
plt.show()
```



File: LinearRegression-origin.py

- Python có rất nhiều thư viện cho việc này, cụ thể ở đây là scikit-learn khá phổ biến



- Trong trường hợp này, chương trình viết như sau (cần install thêm sklearn):

```
from sklearn import linear_model
w = linear_model.LinearRegression(
fit_intercept = False)
w.fit( X.T,y )
w0 = w.coef_.T[0]
w1 = w.coef_.T[1]
```

- File: LinearRegression-sklearn.py

Áp dụng

- Sau khi đã có hàm xấp xỉ tiên lượng (có w_0, w_1), ta có thể dùng để dự báo.
- Chẳng hạn,

```
### Tính toán thử nghiệm
x = float(input( "Du doan trong luong cua nguoi co chieu cao: " ))
ypred = w0 + w1*x
print( "Cân nặng cua nguoi co chieu cao %.0f cm la %.2f kg"  %(x, ypred) )
```

Áp dụng

- Trong thực tế, X ở dạng vector với nhiều thành phần
- Chẳng hạn, trọng lượng không chỉ phụ thuộc vào chiều cao mà có thể phụ thuộc cả việc ăn uống thời trẻ như thế nào, thời gian ngủ trong một ngày, ...
- Chẳng hạn, khi dữ liệu làm tiền đề có K tham số $X = [x_1, x_2, \ldots, x_K]^T$ thuật toán hồi quy tuyến tính cần tìm hàm xấp xỉ có dạng $\varphi(X, w) = w_0 + w_1 x_1 + \ldots + w_K x_K$

$$= w_0 + \sum_{j=1}^{K} w_j x_j = \sum_{j=0}^{K} w_j x_j$$

- Với $w = [w_0, w_1, \ldots, w_K]^T$, và $x_0 = 1$
- Khi đó bảng dữ liệu $(X_i, y_i), \forall i = \overline{1,N}$ sẽ là $\left([x_{1i}, x_{2i}, \ldots, x_{Ki}]^T, y_i\right)$ và

$$\varphi(X_i, w) = w_0 + \sum_{j=1}^{K} w_j x_{ji}, \ \forall i = \overline{1,N}$$

- Đặt

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1N} \\ \vdots & \vdots & \cdots & \vdots \\ x_{K1} & x_{K2} & \cdots & x_{KN} \end{bmatrix}$$

- Và $Y = [y_1, y_2, \ldots, y_N]^T$
- Suy ra

$$\mathbf{X}^T w = \begin{bmatrix} w_0 + w_1 x_{11} + w_2 x_{21} + \ldots + w_K x_{K1} \\ w_0 + w_1 x_{12} + w_2 x_{22} + \ldots + w_K x_{K2} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ w_0 + w_1 x_{1N} + w_2 x_{2N} + \ldots + w_K x_{KN} \end{bmatrix}$$

- Hay

$$\mathbf{X}^T w = \begin{bmatrix} w_0 + \sum_{j=1}^{K} w_j x_{j1} \\ w_0 + \sum_{j=1}^{K} w_j x_{j2} \\ \cdots\cdots\cdots\cdots \\ w_0 + \sum_{j=1}^{K} w_j x_{jN} \end{bmatrix}$$

-

- Bài toán đưa về tìm cực tiểu của hàm $L(w)$ như trên được viết lại là

$$L(w) = \sum_{i=1}^{N} \left[y_i - \left(w_0 + \sum_{j=1}^{K} w_j x_{ji} \right) \right]^2$$

- Dưới dạng ma trận

$$L(w) = \| Y - \mathbf{X}^T w \|_2^2$$

- Nên

$$\frac{\partial L}{\partial w} = 0 \Longleftrightarrow \frac{\partial}{\partial w} \| Y - \mathbf{X}^T w \|_2^2 \Longleftrightarrow \mathbf{X}(Y - \mathbf{X}^T w) = 0$$

- Từ đây

$$w = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}Y$$

Ví dụ

```
print( "Dự đoán chiều cao của một người có" )
Wi = float(input( "- cân nặng là (kg): " ))
Fe = float(input( "- số lượng kg thịt ăn trong 1 ngày: " ) )
Sl = float(input( "- số giờ ngủ trong 1 ngày: " ) )
Hi = w[0] + w[1]*Wi + w[2]*Fe + w[3]*Sl
print( "Người này có chieu cao là %.0fcm" %Hi )
```

- Cần dự báo chiều cao của một đứa trẻ căn cứ vào trọng lượng, số thịt đã ăn và số giờ đã ngủ trong một ngày.

```
import numpy as np
x1 = np.array([[42,46,50,54,55,56,60,64,65,
66,67,68,69,70,71,72,72,73,74,75]]).T
x2 = np.array([[0.2,0.2,0.3,0.4,0.8,0.2,0.5,0.7,
0.2,0.1,0.3,0.5,0.6,0.7,0.3,0.5,0.4,0.5,0.8,0.5]
]).T
x3 = np.array([[8,10,9,12,12,8,8,11,10,9,9,8,
9,10,9,8,8,8,7,7]]).T
y = [145,150,152,154,156,160,162,164,165,
168,170,172,174,175,177,178,180,181,182,184]
```

```
N = len(x1)
onesMat = np.ones( (1,N) )
X = np.concatenate((onesMat, x1.T))
X = np.concatenate((X, x2.T))
X = np.concatenate((X, x3.T))
A = np.dot( X, X.T )
b = np.dot( X, y )
w = np.dot( np.linalg.pinv(A), b )
```

File: LinearRegression-linearPolynomial.py

Ví dụ

File: LinearRegression-linearPolynomial-sklearn.py

- Sử dụng phương thức LinearRegression() có trong thư viện sklearn

```
onesMat = np.ones( (1,N) )
X = np.concatenate((onesMat, x1.T))
X = np.concatenate((X, x2.T))
X = np.concatenate((X, x3.T))

w = linear_model.LinearRegression( fit_intercept = False )
w.fit( X.T, y )

print( "Dự đoán chiều cao của một người có" )
Wi = float(input( "- cân nặng là (kg): " ))
Fe = float(input( "- số lượng kg thịt ăn trong 1 ngày: " ) )
Sl = float(input( "- số giờ ngủ trong 1 ngày: " ) )
Hi = w.coef_.T[0] + w.coef_.T[1]*Wi + w.coef_.T[2]*Fe + w.coef_.T[3]*Sl
print( "Người này có chieu cao là %.0fcm" %Hi )
```

Tở chức lại dữ liệu huấn luyện

- Chúng ta có thể tổ chức lại dữ liệu bằng cách thay vì mảng, ta đưa vào các DataFrame - là kiểu dữ liệu bảng có trong gói pandas (để phân tích dữ liệu)
- Ví dụ trên có thể viết

```
import pandas as pd
data_dictionary = {
    "Weight":[42,46,50,54,55,56,60,64,65,66,67,68,69,70,71,72,72,73,74,75],
    "Meat":[0.2,0.2,0.3,0.4,0.8,0.2,0.5,0.7,0.2,0.1,0.3,0.5,0.6,0.7,0.3,0.5,0.4,0.5,0.8,0.5],
    "Sleep":[8,10,9,12,12,8,8,11,10,9,9,8,9,10,9,8,8,8,7,7],
    "Height":[145,150,152,154,156,160,162,164,165,168,170,172,174,175,177,178,180,181,182,184],
}
df = pd.DataFrame( data_dictionary,columns= ["Weight","Meat","Sleep","Height"] )
```

- Lưu ý: cần install thêm gói pandas này

Tổ chức dữ liệu

- Khi đó chuyển dữ liệu về dạng ma trận $\mathbf{X}^T$ để thêm cột có giá trị là 1 (là hệ số của w_0)

```
X = df[["Weight","Meat","Sleep"]]
X["Ones"] = 1
X = X[["Ones","Weight","Meat","Sleep"]]
y = df["Height"]
```

- Tiếp theo dùng phương thức LinearRegression() như trước, nhưng lưu ý là X ở đây đã là $\mathbf{X}^T$

```
w = linear_model.LinearRegression(fit_intercept=False)
w.fit( X, y )
```

File: LinearRegress-linearPolynomial-sklearn-pandas.py

- Một lưu ý nữa là: thông thường N rất lớn (các giá trị $(X_i, y_i), \forall i = \overline{1,N}$) và hay lưu trữ trong một tập tin.
- Chẳng hạn tập tin CSV (Comma-Separated Values).
- Khi đó ta có thể tạo ra DataFrame từ tập tin này bằng lệnh sau thay vì tạo ra một DataFrame mới từ một Dictionary như trên:

```
df = pd.read_csv( "filename.csv" )
```

- Lưu ý, để ghi DataFrame thành file CSV, dùng lệnh

```
df.to_csv( "filename.cvs",index=False )
```

File: height.csv

Hồi quy đa thức (Polynomial Regression)

- Hàm xấp xỉ trong trường hợp này có dạng một đa thức
$\varphi(x, w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_K x^K$,
với $x \in \mathbb{R}$
- Tại các giá trị rời rạc x_i, ta có:

$$\varphi(x_i, w) = w_0 + \sum_{j=1}^{K} w_j x_i^j, \forall i = \overline{1,N}$$

- Tương tự như trường hợp trước, ta viết lại ma trận $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \\ \vdots & \vdots & \cdots & \vdots \\ x_1^K & x_2^K & \cdots & x_N^K \end{bmatrix}$$

- Để có chương trình, cũng tương tự trên chúng ta chỉ cần đưa vào ma trận $\mathbf{X}$ được tính như bên cạnh.
- Giả sử với bảng số liệu về chiều cao và cân nặng, hàm xấp xỉ là $\varphi(x, w) = w_0 + w_1 x + w_2 x^2$
- Chương trình được viết

```
N = len(x)
onesMat = np.ones( (1,N) )
X = np.concatenate((onesMat, x.T))
X = np.concatenate((X, x.T*x.T))
w = linear_model.LinearRegression(fit_intercept=False)
w.fit( X.T,y )
w0 = w.coef_.T[0]
w1 = w.coef_.T[1]
w2 = w.coef_.T[2]
```

File: PolynomialRegression-sklearn.py, PolynomialRegression-sklearn-pandas.py và PolynomialRegression-sklearn-pandas-csv.py với dữ liệu heightweight.csv

Tổng quát hoá hồi quy đa thức

- Xét hàm xấp xỉ dạng
  $\varphi(w,x) = w_0 + w_1 g_1(x) + w_2 g_2(x) + \ldots + w_K g_K(x)$
  trong đó $g_j(x), \forall j = \overline{1,K}$ là hàm bất kỳ.
- Từ đây ma trận tính toán là

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ g_1(x_1) & g_1(x_2) & \cdots & g_1(x_N) \\ \vdots & \vdots & \cdots & \vdots \\ g_K(x_1) & g_K(x_2) & \cdots & g_K(x_N) \end{bmatrix}$$

- Ví dụ: Tìm hàm xấp xỉ dạng
  $\varphi(x,w) = w_0 + w_1 \sin x + w_2 \cos x$
- Để thực nghiệm, ta tạo ra dữ liệu ngẫu nhiên

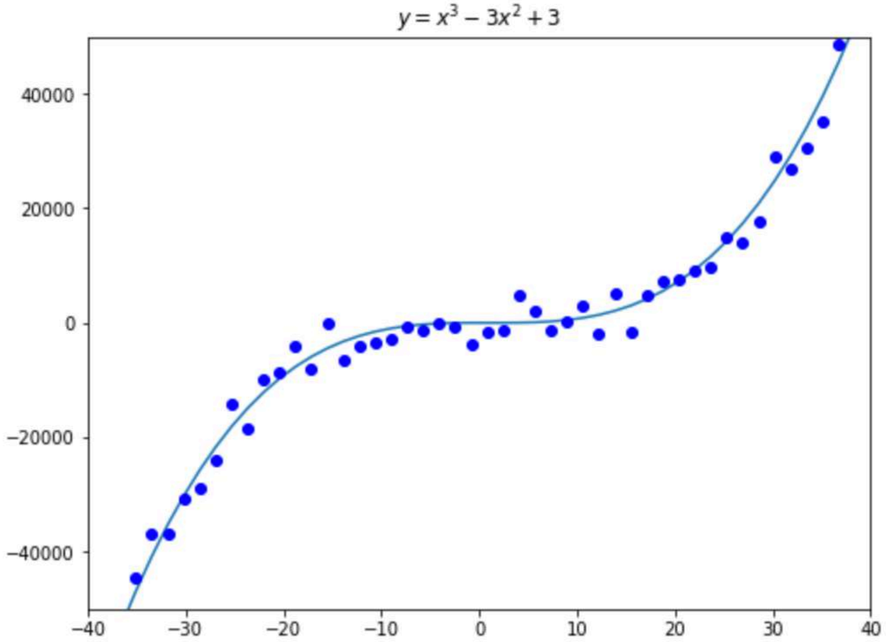
```
import random
import pandas as pd

def myrand(a,b):
    r = random.random()
    return a + (b-a)*r

x0 = np.linspace( -10,10 )
y0 = 1 + 2*np.sin(x0)-np.cos(x0)+myrand(-1,1)

data = {"X":x0,"Y":y0}
df = pd.DataFrame( data )
df.to_csv( "xy.csv")
```

- Đọc dữ liệu để đưa vào DataFrame

```
df = pd.read_csv( "xy.csv" )
X = pd.DataFrame({"Ones":1,"Sin":np.sin(df["X"]),"Cos":np.cos(df["X"])})
y = df["Y"]
X.head(), y.head()
```

Bài tập

1. Cho tập tin dữ liệu dataset.csv bao gồm 2 cột là data và label. Hãy tìm hàm xấp xỉ dạng $\varphi(x, w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$, chẳng hạn như hình bên cạnh là một kết quả với $w_0 = 3, w_1 = 0, w_2 = -3, w_3 = 1$
2. Viết chương trình tìm hàm xấp xỉ dạng $\varphi(x, w) = w_0 + w_1 \sin x + w_2 \cos x$ như trong ví dụ trước
3. Dùng phương pháp bình phương tối thiểu để xây dựng thuật toán tìm các hàm xấp xỉ dạng:



$$\varphi(x, a, b) = ae^{bx}$$

$$\varphi(x, a, b) = ax^b$$

$$\varphi(x, a, b) = ab^x$$

Đưa hàm xấp xỉ dạng tích thành tổng

- Giả sử hàm xấp xỉ cần tìm là $\varphi(x, a, b) = ae^{bx}$
- Bằng cách lấy logarithm hai vế, ta được

$$\begin{aligned}\ln \varphi(x, a, b) &= \ln ae^{bx} \\ &= \ln a + \ln e^{bx} \\ &= \ln a + bx\end{aligned}$$

- Đặt

$$\Phi = \ln \varphi(x, a, b), A = \ln a$$

- Phương trình được viết lại

$$\Phi(x, A, b) = A + bx$$

- Hay

$$\Phi(x, w_0, w_1) = w_0 + w_1 x$$

- Với

$$w_0 = A, w_1 = b$$

- Sau khi giải xong, suy ngược trở lại với

$$\varphi = e^{\Phi}$$

- **Với hàm** $\varphi(x, a, b) = ax^b$
- Cách tương tự, ta có:

$$\begin{aligned}\ln \varphi(x, a, b) &= \ln a + b \ln x \\ &= A + bX \\ \Phi(X, w_0, w_1) &= w_0 + w_1 X\end{aligned}$$

- Với

$$\begin{aligned}\Phi &= \ln \varphi \\ w_0 &= \ln a \\ w_1 &= b \\ X &= \ln x\end{aligned}$$

- Để tính toán, cần thay các data point x bởi X
- Sau khi tính xong cũng quay trở về $\varphi = e^{\Phi}$

- **Hoặc với hàm** $\varphi(x, a, b) = ab^x$
- Ta có:

$$\begin{aligned}\ln \varphi(x, a, b) &= \ln a + x \ln b \\ &= A + Bx \\ \Phi(x, w_0, w_1) &= w_0 + w_1 x\end{aligned}$$

- Trong đó

$$\begin{aligned}\Phi &= \ln \varphi \\ w_0 &= \ln a \\ w_1 &= \ln b\end{aligned}$$

- Cũng tương tự, sau khi tính xong cũng quay trở về $\varphi = e^{\Phi}$

Hồi quy Logistic (Logistic Regression)

- Với các dữ liệu dự đoán mang tính xác suất (kết quả $\in [0,1]$), người ta sử dụng hàm Sigmoid có dạng $\varphi(x, w) = \dfrac{1}{1 + e^{-(a+bx)}}$
- Trên cơ sở hàm này Ông **David R. Cox** (chuyên gia về thống kê học) đã phát triển mô hình Logistic Regression vào đầu những năm 1970 để phân tích các biến nhị phân.
- Hàm này cũng còn có tên là hàm Logistic, nên phương pháp hồi quy tìm hàm xấp xỉ có dạng này gọi là phương pháp hồi quy Logistic
- Do kết quả giá trị trả về của hàm xấp xỉ thuộc (0,1] nên những ứng dụng có nhãn thuộc khoảng này được sử dụng khi cần dự báo.
- Ngoài ra, do hàm Sigmoid có ngưỡng rõ ràng nên thường hay được sử dụng để phân lớp.

Minh hoạ

- Trong gói sklearn có phương thức LogisticRegression chứa trong sklearn.linear_model giúp chúng ta sử dụng nhanh chóng.
- **Ví dụ**: Trong hệ thống BI (Bussiness Intelligence) người ta hay có những ví dụ về việc này.
- Chẳng hạn, quan hệ giữa tình trạng phơi nhiễm chất độc da cam (Agent Orange – AO) và khả năng ung thư tuyến tiền liệt như ở trang http://bis.net.vn/forums/t/484.aspx

06-16-2011 06:12 PM

chucnv

Business Intelligence Solution

Tham gia 12-05-2008
Điểm 10,155

Mô hình hồi qui logistic (Logistic Regression Model)

Trả lời | Liên hệ

Mô hình hồi qui logistic (Logistic Regression Model)

1. Giới thiệu

Mục tiêu của hồi qui Logistic là nghiên cứu mối tương quan giữa một (hay nhiều) yếu tố nguy cơ (*risk factor*) và đối tượng phân tích (*outcome*). Chẳng hạn như đối với nghiên cứu mối tương quan giữa thói quen hút thuốc lá và nguy cơ mắc ung thư phổi thì yếu tố nguy cơ ở đây là thói quen hút thuốc lá và đối tượng phân tích ở đây là nguy cơ mắc ung thư phổi. Trong hồi qui logistic thì các đối tượng nghiên cứu thường được thể hiện qua các biến số nhị phân (binary) như *xảy ra/ không xảy ra ; chết/sống ; có/không,...* còn các yếu tố nguy cơ có thể được thể hiện qua các biến số liên tục (tuổi, huyết áp,...) hoặc các biến nhị phân (giới tính) hay các biến thứ bậc (thu nhập : Cao, trung bình, thấp). Vấn đề đặt ra cho nghiên cứu dạng này là là sao để ước tính độ tương quan của các yếu tố nguy cơ và đối tượng phân tích. Các phương pháp phân tích như hồi qui tuyến tích không áp dụng được vì biến phụ thuộc không phải là biến liên tục mà là biến nhị phân. Nhà thống kê học **David R. Cox** đã phát triển mô hình có tên Logistic Regression Model (1970s) để phân tích các biến nhị phân.

Ví dụ : Bảng dữ liệu dưới đây thu thập để nghiên cứ mối tương quan giữa tình trạng phơi nhiễm chất độc gia cam (Agent Orange – AO) và ung thư tuyến tiền liệt.

- Nghiên cứu mối tương quan giữa thói quen hút thuốc lá và nguy cơ mắc ung thư phổi
- Yếu tố xảy ra ở đây là nguy cơ mắc ung thư phổi.
- Ở đây ta xét khi yếu tố xảy ra được thể hiện qua các biến số nhị phân có hay không xảy ra.

Ví dụ

- Trong trường hợp tổng quát hơn, ta có thể xem xét yếu tố xảy ra được thể hiện qua thuộc tính định lượng (như tuổi, huyết áp, …) hoặc thuộc tính nhị phân (như giới tính) hay thuộc tính hạng mục (cao, trung bình, thấp, ...) như kiểu thực hiện trong điều khiển mờ.
- Xét tỷ số nguy cơ (Odds Ratio – OR) đây là tỷ số của 2 giá trị của một biến nhị phân.
- Chẳng hạn, với bảng như ở trang web trên

	Ung thư (47)	Đối chứng (144)
Phơi nhiễm AO	11	17
Không phơi nhiễm AO	36	127

- Khả năng (odd) mắc ung thư trong nhóm phơi nhiễm AO là 11/17 = 0.647
- Và trong nhóm không phơi nhiễm AO là 36/127 = 0.283
- Suy ra OR = 0.647/0.283= 2.28
- Điều nay cho thấy nguy cơ mắc ung thư tuyến tiền liệt của những người từng phơi nhiễm AO cao hơn gấp 2.28 lần so với những người không bị phơi nhiễm AO

- Từ ví dụ minh họa trên, ta có thể xây dựng công thức chung của mô hình hồi qui logistic như sau
  - Gọi p là xác suất của một sự kiện (chẳng hạn sự kiện mắc ung thư tuyến tiền liệt).
  - Khi đó odd được định nghĩa là $odd = \frac{p}{1-p}$
  - Gọi yếu tố nguy cơ là x (là tình trạng phơi nhiễm AO)
  - x có 2 giá trị là 0 và 1 (0: không phơi nhiễm AO, và 1 bị phơi nhiễm AO)
  - Khả năng (mắc ung thư) có giá trị thuộc (0,1], nên dùng hàm ln() để biểu diễn khả năng này, khi đó thay vì odd ta dùng $\ln(odd)$
  - $\ln(odd)$ phụ thuộc vào giá trị của x qua một hàm số tuyến tính dạng $a + bx$, nghĩa là $\ln(odd) = a + bx$
  - Từ đây suy ra $odd = e^{a+bx}$
  - Như vậy khả năng mắc ung thư của nhóm bị phơi nhiễm là e^a, của nhóm không bị là e^{a+b}
  - Khi đó tỷ số nguy cơ $OR = \frac{e^{a+b}}{e^a} = e^b$
  - Thực tế ta không thể biết 2 giá trị này mà phải ước tính qua bảng số liệu.
  - Trong bảng trên tham số b ước lượng là $e^b = 2.28$, nên $b = 0.824$

Mô hình hồi quy tổng quát

- Mô hình hồi qui logistic tổng quát với K yếu tố nguy cơ $x_1, x_2, \ldots, x_K$ được định nghĩa qua hàm xấp xỉ

$$\varphi(X, w) = \frac{e^X}{1 + e^X} = \frac{1}{1 + e^{-X}} \text{ với}$$

$$X = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_K x_K$$

- Người ta hay gọi
  - w_0 là yếu tố bị chặn (intercept): có nghĩa là khi các yếu tố nguy cơ (hay biến độc lập) $x_1, x_2, \ldots, x_K$ đều bằng 0 thì $\varphi(X, w) = w_0$
  - các $w_1, w_2, \ldots, w_K$ là các hệ số hồi quy (regression cofficient)

- Để hiểu rõ hơn, ta đoán khả năng (xác suất) tử vong do bệnh tim của người trên 50 tuổi.
- Trong trường hợp đơn giản ta sử dụng 3 yếu tố nguy cơ (risk factor) là tuổi (age), giới tính (sex) và nồng độ cholesterol trong máu (blood cholesterol level) để dự đoán nguy cơ chết do bệnh tim trong thời gian tới.

- Các biến độc lập (các yếu tố nguy cơ): x_1 : age (trên 50 tuổi), x_2 : giới tính, 0: nam, 1: nữ, x_3 : cholesterol level mmol/L (trên 5.0)
- Sau khi thu thập dữ liệu và tính toán các tham số cho mô hình (hệ số chặn, các hệ số hồi qui) giả sử tìm được $w_0 = -5, w_1 = 2.0, w_2 = -1$ và $w_3 = 1.2$
- Như vậy khả năng tử vong là $\varphi(X, w) = \frac{1}{1 + e^{-X}}$ với $X = -5 + 2x_1 - 1x_2 + 1.2x_3$
- Trong mô hình,
  - Hệ số của yếu tố nguy cơ x_1 (age) là +2.0, nghĩa là khi tuổi tăng sẽ làm tăng nguy cơ chết vì bệnh tim (X tăng lên 2.0 sau mỗi năm đối với người trên 50 tuổi).

- Hệ số hồi qui của yếu tố nguy cơ giới tính x_2 (sex) có dấu âm và có độ lớn là 1.0 điều này có nghĩa là nữ giới có nguy cơ chết vì bệnh tim thấp hơn nam giới (X giảm 1 nếu giới tính là nữ).
- Hệ số hồi qui của yếu tố nguy cơ x_3 (cholesterol level mmol/L) có dấu dương và có độ lớn là 1.2 có nghĩa là việc tăng 1 mmol/L đối với các trường hợp trên 5 mmol/L sẽ làm tăng nguy cơ chết vì bệnh tim với độ lớn 1.2.

- Sau khi có kết quả, dùng mô hình này để dự đoán nguy cơ chết vì bệnh tim của một người là: nam giới, 50 tuổi và đo được nồng độ Cholesterol trong máu là 7.0 mmol/L.
- Ta có $\varphi(X, w) = \dfrac{1}{1 + e^{-X}}$ với
$X = -5 + 2 \times (50 - 50) - 1 \times 0 + 1.2 \times (7.0 - 5.0)$
- Nên $X = -2.6$, từ đây suy ra
$$\varphi(X, w) = \frac{1}{1 + e^{-(2.6)}} = 0.0691$$
- Ta kết luận được là người này khó có khả năng tử vong vì xác suất chỉ xấp xỉ 7%

- Để viết chương trình tìm hàm xấp xỉ, nói cách khác là tìm hệ số chắn (intercept) và các hệ số hồi quy ta dùng một dataset gồm 4 cột x 100 dòng như trong file riskofdeath.csv.

Age (>50)	Sex	Cholesterol Level mmol/L	Risk of Death
50	0	9	1
85	0	6	1
60	1	9	0
79	0	7	0

```
import numpy as np
from sklearn import linear_model
import pandas as pd

df = pd.read_csv( "riskofdeath.csv" )

X = df[["Age (> 50)","Sex","Cholesterol Level mmol/L"]]
y = df["Risk of Death"]

w = LogisticRegression()
w.fit( X, y )

w0 = w.intercept_[0]
w1 = w.coef_.T[0]
w2 = w.coef_.T[1]
w3 = w.coef_.T[2]

y_pred_proba = w.predict_proba( X )
y_pred = w.predict( X )
```

- Từ đây có thể dùng để dự báo với người có tuổi A, giới tính S là 0 hoặc 1, cholesterol là C

```
XX  = w0 + w1*A + w2*S + w3*C
phi = 1.0/(1.0 + np.exp(-XX))
```

File: LogisticRegression-sklearn.py

Phương pháp phân lớp

- Các phương pháp hồi quy ngoài việc dùng để dự báo kết quả của tương lai dựa trên dataset đã có và đã gán nhãn, i.e. gồm cả X_i và y_i, $\forall i = \overline{1,N}$, chúng ta có thể dùng để phân lớp.
- Chẳng hạn, với phương pháp hồi quy Logistic, do kết quả rời rạc dạng nhị phân nên có thể dùng để phân lớp.
- Thuật toán cũng không thay đổi, tuy nhiên trong dataset cần chia thành 2: một phần để huấn luyện (**training data**) và phần còn để dùng làm dữ liệu kiểm tra (**test data**).

- Do ta không có dữ liệu để kiểm tra chương trình nên có thể viết một đoạn chương trình nhỏ để tạo dữ liệu như sau:

```
NUM_DATA = 1000
df = pd.DataFrame()
df["Sex"] = np.random.randint(2,size=NUM_DATA)
df["Age (> 50)"] = np.random.randint(50,
size=NUM_DATA) + 50
df["Cholesterol Level (>5 mmol/L)"] =
np.random.uniform( 5,size=NUM_DATA) + 5
df["Risk of Death"] =  1
for i in range(NUM_DATA):
    if df.at[i,"Cholesterol Level (>5 mmol/L)"] > 7.5:
        df.at[i,"Risk of Death"] =  1
    else:
        df.at[i,"Risk of Death"] =  0
df = df[["Age (> 50)","Sex","Cholesterol Level (>5
mmol/L)","Risk of Death"]]

df.to_csv( "riskofdeath.csv",index=False)
```

Dùng Logistic Regression

- Chương trình này tạo ra NUM_DATA có dạng như sau được lưu trữ vào file mydataset.csv

```
1 df.head()
```

	Age (> 50)	Sex	Choleterol Level mmol/L	Risk of Death
0	69	0	9	0
1	80	0	9	0
2	80	1	6	0
3	77	1	8	1
4	65	1	7	1

- Sau khi nạp dữ liệu vào 2 DataFrame là X và y từ tập tin CSV

```
df = pd.read_csv( "riskofdeath.csv" )
X = df[["Age (> 50)","Sex","Choleterol Level mmol/L"]]
y = df["Risk of Death"]
```

- Tách để dành 25% dùng làm dữ liệu kiểm tra sau đó dùng mô hình hồi quy Logistic với dữ liệu huấn luyện:

```
X_train, X_test, y_train, y_test = model_selection.train_test_split( X,y,test_size=0.25,random_state=42 )
w = linear_model.LogisticRegression()
w.fit( X_train, y_train )
```

- Để đánh giá về độ chính xác (Accuracy):

```
y_pred = w.predict( X_test )
print( "Độ chính xác là: ", metrics.accuracy_score(y_test, y_pred) )
```

- Lưu ý rằng dữ liệu kiểm tra như cách ta làm ở trên thực chất đó là dữ liệu *kiểm định* (**validation test)** để xem xét tính đúng đắn của mô hình.
- Còn dữ liệu kiểm tra (**test data**) chính là dữ liệu của tương lai mà chúng ta cần kiểm chứng coi có đúng với thực tế xảy ra hay không. Dữ liệu này chưa có nhãn (chưa có giá trị y_i).
- Chúng ta có thể tính toán để biết bộ dữ liệu kiểm tra có
  - Xác suất mà dữ liệu này là tử vong (giá trị 0) và xác suất mà cũng bộ dữ liệu này không tử vong (giá trị 1) bằng lệnh:

```
print( w.predict_proba(X_test)[:,0] )
print( w.predict_proba(X_test)[:,1] )
```

Về độ chính xác trong dự đoán

- Ngoài phương thức accuracy_score(validation, prediction), còn có ma trận confusing (ma trận "bối rối") thông qua phương thức confusion_matrix(validation, prediction)

```
len(X_test)
metrics.accuracy_score(y_test,y_pred)
metrics.confusion_matrix(y_test,y_pred)
```

- Ma trận này có ý nghĩa như bên cạnh

```
Number of Validation Data
 250
Accuracy
 0.496
Confusion Matrix
 [[31 96]
 [30 93]]
```

	Đoán là không	Đoán là có
Thực tế không	31	96
Thực tế có	30	93

	Đoán là không (Negative)	Đoán là có (Positive)
Thực tế không (Negative)	True Negative (31)	False Positive (96)
Thực tế có (Positive)	False Negative (30)	True Positive (93)

- Với kết quả trên cho thấy: trong 250 trường hợp
  - Thực tế có 123 trường hợp ***tử vong***, nhưng hệ thống chỉ đoán đúng 93 trường hợp, còn 30 cho là ***không tử vong***
  - Thực tế có 127 trường hợp là ***không tử vong***, hệ thống đoán chỉ đúng 31 trường hợp, còn lại cho rằng ***tử vong*** là 96 trường hợp.

- Từ ma trận này ta suy ra:
  - **Precision**: đây là tỷ lệ giữa dự đoán tử vong đúng so với tất cả các trường hợp được đoán là tử vong. Nói cách khác, có bao nhiêu dự đoán là "positive" mà thực sự là "true" trong thực tế

$$Precision = \frac{TP}{TP + FP}$$

$$= \frac{93}{93 + 96} = 49.2\,\%$$

Như vậy 49.2% trường hợp dự đoán tử vong là chính xác.

- **Recall** hay còn gọi là **Sensitivity**: chỉ trong những trường hợp thực tế tử vong, thì bao nhiêu trong số đó được dự đoán đúng. Nói cách khác, có bao nhiêu dự đoán "positive" đúng là do mô hình đưa ra.

$$Recall = \frac{TP}{TP + FN}$$

$$= \frac{93}{93 + 30} = 75.6\,\%$$

	Đoán là không (Negative)	Đoán là có (Positive)
Thực tế không (Negative)	True Negative (31)	False Positive (96)
Thực tế có (Positive)	False Negative (30)	True Positive (93)

- Như vậy, chỉ dự đoán 93 trường hợp tử vong trong khi thực tế có tới 30+93 = 123 trường hợp tử vong. Nên mô hình có thể dự đoán được 75,6% số trường hợp tử vong trong thực tế.

- ***F_1-score***: Do khó giải thích trong trường hợp tổng quát thì Precision quan trọng hay Recall quan trọng, nên trong trường hợp này ta có độ độ $F_\beta - score$, mà trường hợp riêng đó là $F_1 - score$

  $$F_\beta - score = (1 + \beta^2)\frac{Precision \times Recall}{\beta^2 \times Precision + Recall}$$

  - Nếu coi trọng Precision hơn Recall, chọn $\beta < 1$ (thông thường là 0.5)
  - Ngược lại chọn $\beta > 1$ (thường là 2)
  - Với trường hợp trên, ta có $F_1 - score = 2\frac{75.6 \times 49.2}{75.6 + 49.2} = 59.6\,\%$

- **Accuracy**: cho biết trong tất cả các dự đoán, tỷ lệ dự đoán đúng là bao nhiêu. Nếu dataset là cân đối (số trường hợp tử vong và không tử vong là ngang nhau), khi đó chỉ cần dùng Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Trong trường hợp với kết quả trên,

$$Accuracy = \frac{31 + 93}{31 + 93 + 96 + 30} = 49.6\,\%$$

File: LogisticRegression-sklearn-classification.py

```
from sklearn import metrics
metrics.precision_score(y_test,y_pred)
metrics.recall_score(y_test,y_pred)
metrics.f1_score(y_test,y_pred)
metrics.accuracy_score(y_test,y_pred)
```

Ngoài sai số khi dự đoán, còn có

- Trong scikit-learn có cá phương thức tính sai số như sau có trong lớp metrics của gói sklearn
  - Sai số bình quân tuyệt đối (Mean Absolute Error - MAE)

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_{pred} - y_{test}|$$

```
metrics.mean_absolute_error(y_pred,y_test)
```

  - Sai số toàn phương trung bình (Mean Squared Error - MSE)

$$MSE = \frac{1}{N}\sum_{i=0}^{N} [y_{pred} - y_{test}]^2$$

```
metrics.mean_squared_error(y_pred,y_test)
```

  - Sai số chuẩn hay sai số quân phương (Root MSE)

$$RMSE = RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N} [y_{pred} - y_{test}]^2}$$

```
numpy.sqrt(metrics.mean_squared_error(y_pred,y_test))
```

Đánh giá mô hình

- **Overfitting** chỉ hiện tượng mô hình đạt kết quả rất tốt trên tập dữ liệu huấn luyện nhưng lại kém trên tập dữ liệu kiểm tra.
- **Underfitting**: thuật toán đạt kết quả xấu trên cả tập huấn luyện và tập kiểm tra.
- Để đánh giá mô hình, chúng ta có thể căn cứ theo các tiêu chí sau trên tập huấn luyện (training data), tập kiểm tra (test data) và tập kiểm chứng (unseen data, hay validation data) với hàm sai số là E.
  - E_{train} nhỏ, E_{test}, $E_{validate}$ nhỏ: Good Fitting: chọn mô hình này
  - E_{train} nhỏ, E_{test}, $E_{validate}$ lớn: Overfitting: nên hiệu chỉnh
  - E_{train} lớn, E_{test}, $E_{validate}$ lớn: Underfitting: loại bỏ

Machine Learning với TensorFlow

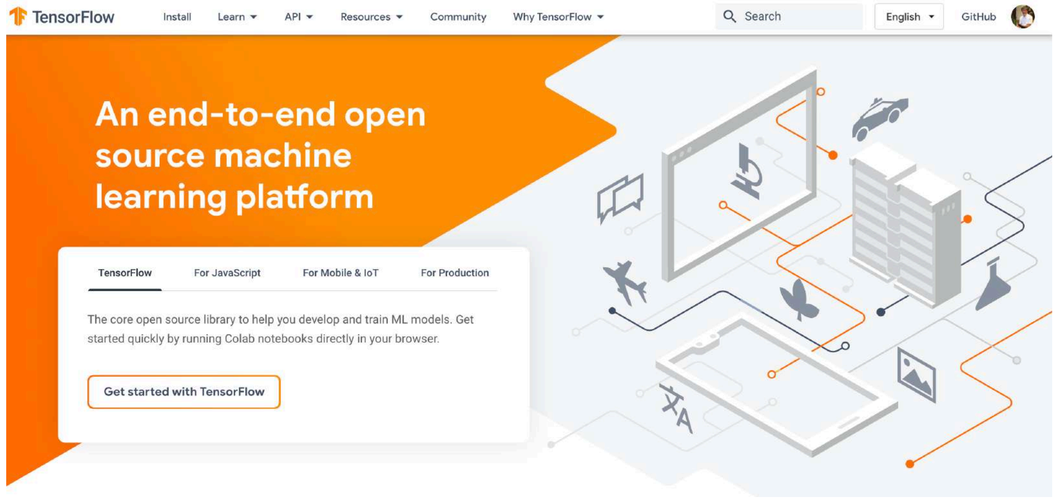
- Những phương pháp hồi quy nêu trên là những cách thức cơ bản đã có trong Toán học từ lâu.
- Về sau này, những phương pháp của Toán học, đặc biệt là Phương pháp tính (Computational Mathematics) cũng như Toán học tính toán (Numerical Analysis) đã được vận dụng vào trong Machine Learning khá nhiều.
- TensorFlow là một công nghệ sử dụng triệt để những vấn đề có trong các lĩnh vực kể trên như vector, ma trận, đạo hàm số, toán rời rạc, lý thuyết đồ thị, tối ưu rời rạc, ...

Về TensorFlow

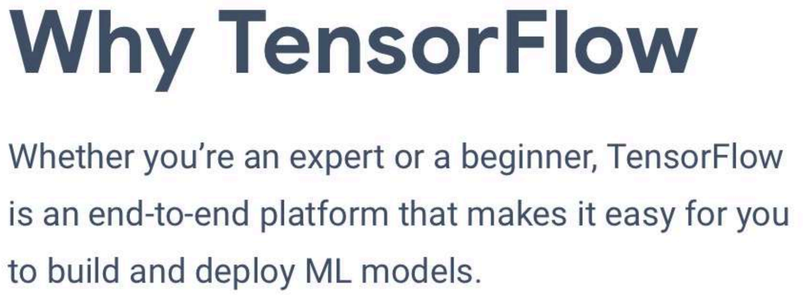
Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

- TensorFlow của Google là công cụ giúp tạo ứng dụng về thông minh nhân tạo - AI (Artifical Intelligence)
  - TensorFlow là thư viện phần mềm nguồn mở về tính toán số hiệu năng cao (high performance numerical computation) thực hiện trên các loại máy tính hiện nay (desktop, cluster, mobile)
  - Tensor là khái niệm mở rộng matrix trong toán học.
- Trong tin học là mảng nhiều chiều
  - Trường hợp 0 chiều là scalar (vô hướng),
  - 1 chiều là vector (véc tơ),
  - 2 chiều là matrix (ma trận),
  - và n-chiều là tensor



- Trong ML, việc tìm ra hàm hay model thể hiện mối quan hệ giữa tập nguồn và tập đích là nhiệm vụ chính.
- TensorFlow là công cụ để hiện thực ML, nên đã đưa ra khái niệm Flow như là đồ thị luồng dữ liệu (*data flow graphs*)
- Đó chính là luồng dữ liệu giữa các Tensor
- Chính vì vậy, trong TensorFlow có 2 khái niệm cơ bản đó là Node và TensorNode: là chỗ làm thay đổi dữ liệu trong luồng dữ liệu.
  - Nên **node** là các phép toán, các hằng giá trị, các biến.
    - Đó chính là điểm giao cắt trong Graph (đồ thị)



Ví dụ đơn giản

File: Example-tensorflow.py

- **Tensor**: đó chính là loại dữ liệu.
  - Ở góc độ Graph, Tensor nằm trên cạnh (Edge) của đồ thị còn Node là nút của đồ thị đó.
  - Tensor có 3 thuộc tính cơ bản là *rank*, *shape* và *type:*
    - rank: 0 là Scalar, 1 là vector, 2 là matrix, 3 là 3D-array, ...
    - shape: là hình dạng, hay kích thước của Tensor
    - type: kiểu dữ liệu của các phần tử trong Tensor

- Ta thử làm ví dụ đơn giản đó là xuất ra một dòng chữ thông qua TensorFlow

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

sess = tf.Session()
xinchao = tf.constant("Xin chào các bạn lớp KTM19B")
print( sess.run(xinchao).decode("utf-8") )
```

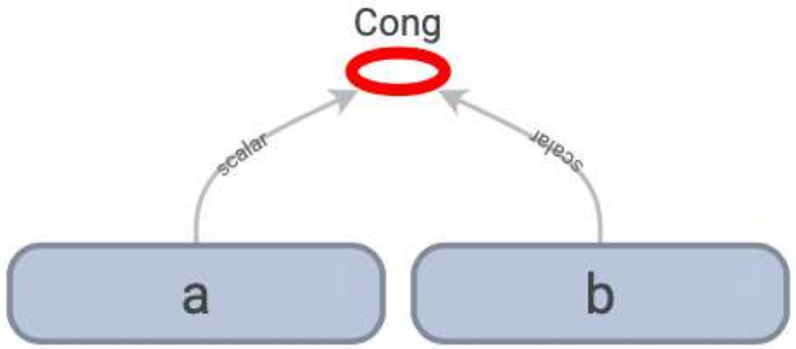
- Do dùng TensorFlow nên cần phải thông qua một Worker của nó.
- Ngoài ra, do dùng chữ 16 bit nên phải decode chữ 8 bit

```
1 import tensorflow.compat.v1 as tf
2 tf.disable_v2_behavior()
3 sess = tf.Session()
4 xinchao = tf.constant( "Xin chào các em lớp KTM19B" )
5 print( sess.run(xinchao).decode("utf-8") )
```

```
Xin chào các em lớp KTM19B
```

Graph và Session

- Trong TensorFlow có 2 nhiệm vụ rạch ròi, đó là:
  - Định nghĩa: nhằm xây dựng đồ thị (Graph)
  - Thực thi các tính toán: dùng Session để thực thi
- Hoặc để thực hiện phép toán cộng 2 số nguyên, trước hết cần phải định nghĩa một đồ thị gồm 3 nút:
  - Biến a
  - Biến b
  - Phép toán cộng




```
import tensorflow as tf
a = tf.Variable(2, name = "a")
b = tf.Variable(3, name = "b")
x = tf.add(a,b, name = "Cong" )
```

- Sau đó thực thi bằng một phiên Session
- Lưu ý: ngoại trừ các hằng giá trị, các biến phải được khởi tạo giá trị mới dùng được bằng hành vi initializer()
- Thay vì sess.run(), có thể viết:

```
with tf.Session() as sess:
    a.initializer.run()
    b.initializer.run()
    print( "Kết quả: ", x.eval() )
```

```
import tensorflow as tf

a = tf.Variable(2, name = "a")
b = tf.Variable(3, name = "b")
x = tf.add(a,b, name = "Cong" )

sess = tf.Session()
sess.run(a.initializer)
sess.run(b.initializer)
print( "Kết quả: ", sess.run(x))
sess.close()
```

Một ví dụ đơn giản khác

- Cần suy ra tuổi khi biết năm sinh, khai báo những gì cần thiết khi sử dụng TensorFlow 2

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

- Khai báo các biến để dùng

```
from datetime import date

birthy = int( input("Năm sinh: ") )
y = tf.Variable( birthy,name="NamSinh")
curry = date.today().year
c = tf.Variable( curry,name="NamHienTai" )
a = tf.add( c,-y, name="PhepToan" )
```

- Thực hiện để xuất kết quả

```
sess = tf.Session()
sess.run( y.initializer )
sess.run( c.initializer )
print( "Tuổi là: ", sess.run(a) )
```

- Như vậy, khi sử dụng TensorFlow ta phải khởi tạo Graph trước khi đưa giá trị của các đỉnh (là các tensor) vào trong Graph. Ở đây
  - Đỉnh(node) lưu biến, phép tính toán
  - Cạnh (edge) là dữ liệu truyền bên trong Graph.
- Chẳng hạn, lệnh:

```
tf.constant("Xin chào các em lớp KTM19B")
```

- Gọi phương thức constant() của đối tượng tf để tạo ra một đối tượng tf.Operation (là một thao tác) và thêm nó vào Graph, rồi trả về một đối tượng tf.Tensor có giá trị "Xin chào các ..."

- Hoặc như với biến với phép toán, khởi tạo Graph bằng các lệnh:

```
y = tf.Variable( birthy,name="NamSinh")
c = tf.Variable( curry,name="NamHienTai" )
a = tf.add( c,-y, name="PhepToan" )
```

- Sau đó lại chuẩn bị

```
init = tf.initialize_all_variables()
# Hoặc cho tất cả các biến trong toàn bộ chương trình
init = tf.global_variables_initializer()
# Hoặc
init_y = y.initializer
init_c = c.initializer
```

- Để từ đó mới thực thi

```
sess = tf.Session()
sess.run( init )
print( "Tuổi là: ", sess.run(a) )
```

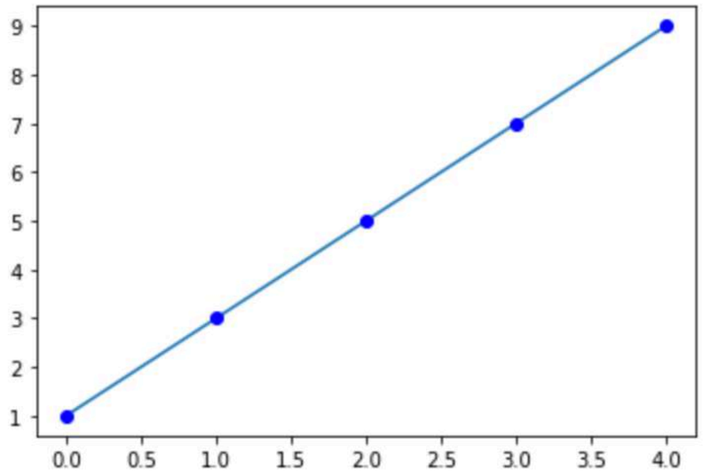
- Lưu ý rằng, TensorFlow không phải ra đời nhằm để giải quyết những công việc qua các vị dụ trên.
- Mà TensorFlow chủ yếu để dùng trong việc xây dựng các ứng dụng có tính trí tuệ (hộc)
- Cũng lưu ý thêm, đã có TensorFlow version 2; trong đó có nhiều thay đổi; chúng ta sẽ làm quen trong các phần sau.

Ứng dụng trong hồi quy tuyến tính

- Trong phần trước ta sử dụng phương pháp hời quy tuyến tính nhằm tìm ra hàm xấp xỉ thông qua việc ***tính toán trên dataset*** để từ đó tìm ra các hệ số $w_0, w_1, w_2, \ldots, w_K$.
- Tuy nhiên, chúng ta cũng có thể tiếp cận theo nghĩa ***quan sát các dữ liệu có trong dataset*** để từ đó rút ra các hệ số này.
- TensorFlow giúp ta hình thành mô hình theo lối suy nghĩ này.

```
1  import matplotlib.pyplot as plt
```

```
1  plt.plot( X_train,y_train )
2  plt.plot( X_train,y_train,"bo" )
3  plt.show()
```



- Giả sử chúng ta cần tìm ra một model (một quy tắc) tương ứng giữa
  - Tập nguồn {0,1,2,3,4}
  - Và tập đích {1,3,5,7,9}
- Quan sát, rõ ràng ta có thể rút ra quy tắc đó là $y = 2x + 1$. Trong đó x là phần tử thuộc tập nguồn, y là phần tử thuộc tập đích
- Nếu xem xét từng tọa độ trong mặt phẳng (x,y), đó là các điểm có toạ độ (0,1), (1,3), (2,5), (3,7), (4,9)

Quan sát từ người

- Dùng TensorFlow để kiểm chứng lại coi có đúng là $w_0 = 1, w_1 = 2$ trong trường hợp hàm xấp xỉ là $\varphi(x, w) = w_0 + w_1 x$.
- Ta biết cần phải cực tiểu hàm

$$L(w) = \sum_{i=1}^{N} [y_i - \varphi(x_i, w)]^2$$

- Với quán sát chính xác như trên thì khi tính toán với $w_0 = 1$, $w_1 = 2$, $L(w)$ phải bằng 0.

- Với dataset

```
X_train = [0,1,2,3,4]
y_train = [1,3,5,7,9]
```

- Khởi tạo Graph để lưu trữ với $w_0 = 1, w_1 = 2$ và thao tác là hàm $\varphi(x, w)$

```
a = 2
b = 1
w0  = tf.Variable( [b], dtype=tf.float32 )
w1  = tf.Variable( [a], dtype=tf.float32 )
x   = tf.placeholder( tf.float32 )
y = tf.placeholder( tf.float32 )
phi = w0 + w1*x
L = tf.reduce_sum( tf.square(y-phi) )
```

- Tinh toán

```
sess = tf.Session()
sess.run( tf.global_variables_initializer() )
Lw0w1 = sess.run(L,{x:X_train,y:y_train})
```

- Xuất

```
print( Lw0w1 ) # Xuất kết quả là 0.0
```


File: LinearRegression-tensorflow.py

Đến lượt máy

- Như ta biết, trong trường hợp tổng quát của phương pháp hồi quy tuyến tính thì tập đích và tập nguồn có quan hệ với nhau dạng ma trận $\varphi(X, w) = w_0 + w^T X$, trong đó $w = [w_1, w_2, \ldots, w_K], X^T = [x_1, x_2, \ldots, x_K]$
- Vấn đề là phải làm sao huấn luyện máy cách "suy nghĩ" để sai số trung bình phương $L(w)$ đạt cực tiểu, từ đó tìm ra được các hệ số $w_0, w_1, w_2, \ldots, w_K$ của hàm $\varphi(X, w)$
- Xét trường hợp hàm xấp xỉ chỉ có 1 tham số và x, khi đó $\varphi(x, w) = w_0 + w_1 x$
- Giả sử ban đầu cho $w_0 = 1, w_1 = 1$, rồi thử dần dần với điều kiện $\frac{\partial \varphi}{\partial w}$ tiến đến 0, từ đó tìm ra w_0, w_1 làm cho $L(w)$ là nhỏ nhất.

- Cũng như trên, đưa vào dataset, rồi sau đó tạo một graph với các biến, các thao tác tương ứng

```
X_train =[1.45,1.50,1.52,1.54,1.56,1.60,1.62,1.64,1.65,
1.68,1.70,1.72,1.74,1.75,1.77,1.78,1.80,1.81,1.82,1.84]
y_train =[42,46,50,54,54,56,60,64,65,66,67,68,
69,70,71,72, 72,73,74,75]

w0 = tf.Variable([1.], dtype=tf.float32)
w1 = tf.Variable([1.], dtype=tf.float32)
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
phi = w0 + w1*x
L   = tf.reduce_sum( tf.square(y - phi) )
```

- Dùng phương thức GradientDescentOptimizer() để tính đạo hàm, sau đó tìm giá trị nhỏ nhất của đạo hàm này

```
deriv = tf.train.GradientDescentOptimizer(0.01)
opt   = deriv.minimize(L)
```

- Sau khi chuẩn bị graph xong, khởi động giá trị ban đầu cho tất cả các biến

```
init = tf.global_variables_initializer()
```

- Dùng worker thông qua Session của TensorFlow để thực hiện

```
sess = tf.Session()
sess.run(init)
```

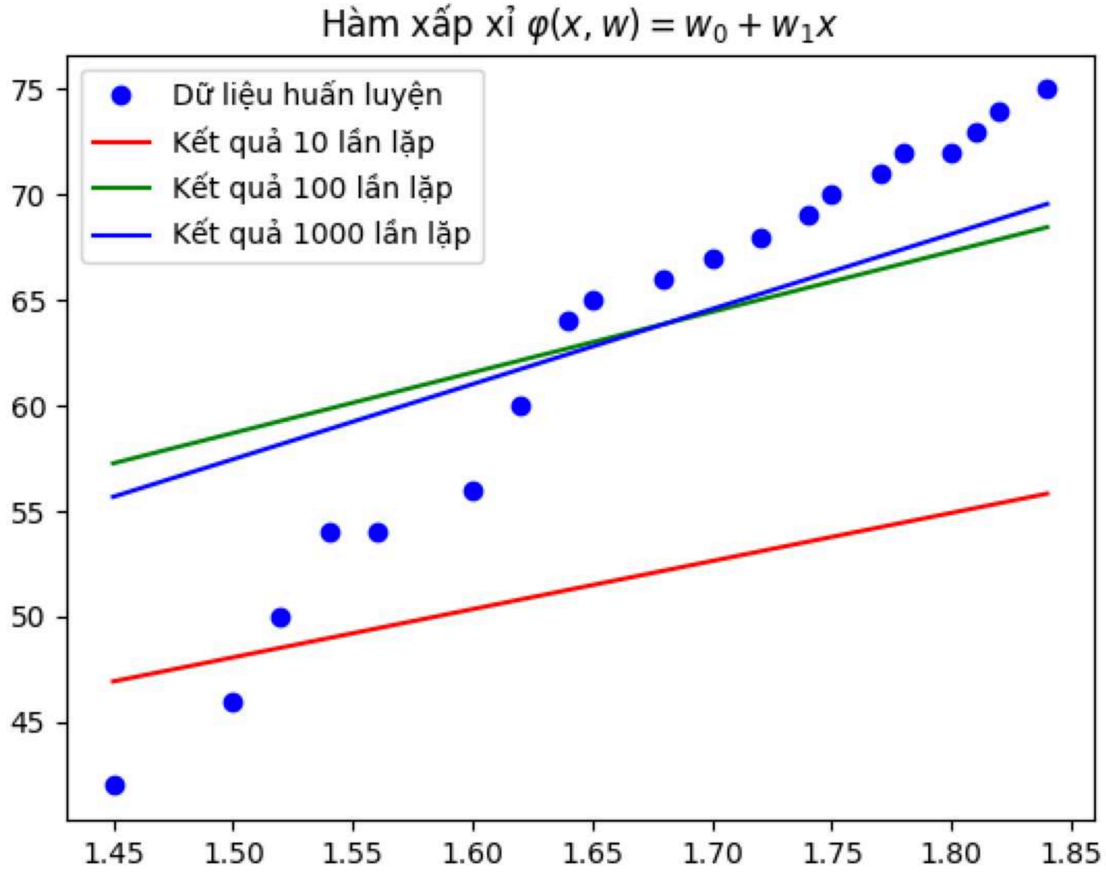
- Ở đây với NUM_ITER là số lần lặp

```
NUM_ITER = 1000
for i in range(NUM_ITER):
  sess.run( opt, {x:X_train, y:y_train} )
Lw0w1, w0p, w1p = sess.run( [L,w0,w1],{x:
X_train, y:y_train} )
y_pred = w0p + w1p*X_train
```

- Xuất ra sai số trung bình bình phương

```
from sklearn import metrics
print("Sai số trung bình phương (MSE): %.5f" % (metrics.mean_squared_error(y_train,y_pred)))
```

File: LinearRegression-tensorflow-1D.py



- Tương tự như vậy, ta có thể hiện thực với TensorFlow về những mô hình có nhiều loại dữ liệu đầu vào hơn.
- Chẳng hạn, có $X = [x_1, x_2, x_3]$, tìm $\varphi(X, w) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$
- Tìm w để

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} [y_i - \varphi(X_i, w)]^2 \to 0$$

- Mô tả

```
X = tf.placeholder(tf.float32, name="X")
y = tf.placeholder(tf.float32, name="Y")
w = tf.Variable(tf.zeros([3, 1]), name="W")
w0 = tf.Variable(tf.zeros([1]), name="b")
phi = tf.add( tf.matmul(X,w),w0 )
L = tf.reduce_mean( tf.square(y - phi) )
deriv = tf.train.GradientDescentOptimizer(0.001)
opt = deriv.minimize(L)
X_train,y_train = createDataset( LEN_DATASET )
```

- Thực thi

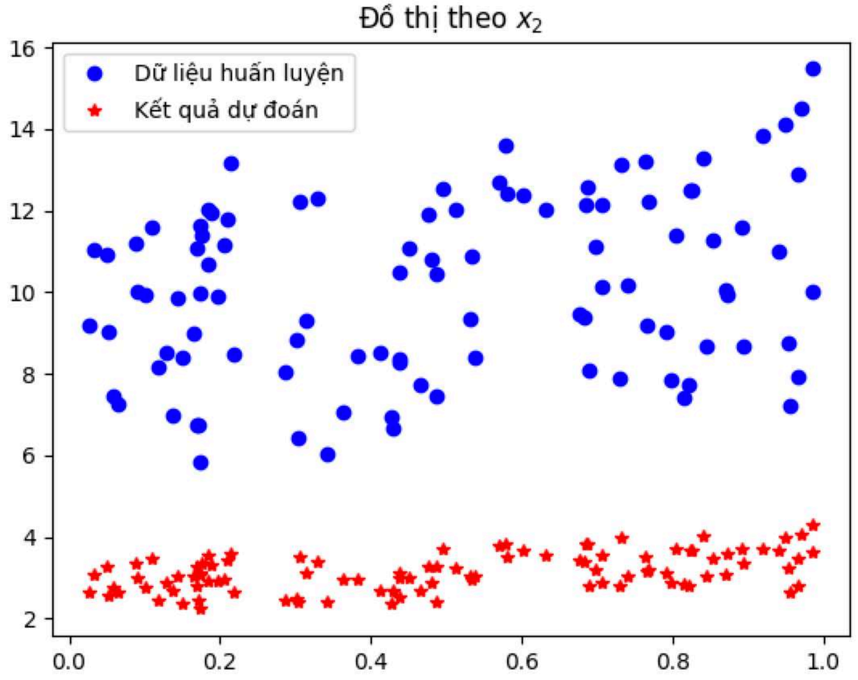
```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(NUM_ITER):
    sess.run( opt, {X:X_train,y:y_train} )
Lw,Wp,w0p = sess.run( [L,w,w0],{X:X_train,y:y_train} )

y_pred =
sess.run( tf.add(tf.matmul(tf.cast(X_train,tf.float32),W
p),w0p) )
```

- Xuất kết quả sai số

```
print("Root MSE: ",metrics.mean_absolute_error(y_pred,y_train))
print("MAE: ",np.sqrt(metrics.mean_squared_error(y_pred,y_train)))
```

- Vẽ hình

```
plt.title("Đồ thị theo " + r"$x_2$")
plt.plot(X_train[:,1],y_train,'bo',label="Dữ liệu huấn luyện")
plt.plot(X_train[:,1],y_pred, 'r*',label="Kết quả dự đoán" )
plt.legend()
plt.show()
```



File: LinearRegression-tensorflow-3D.py

TensorFlow ver. 2

- Mọi thứ không rườm ra như TensorFlow ver. 1
- Chúng ta quay trở lại với ví dụ ban đầu về hồi quy tuyến tính, chương trình được viết lại như sau:

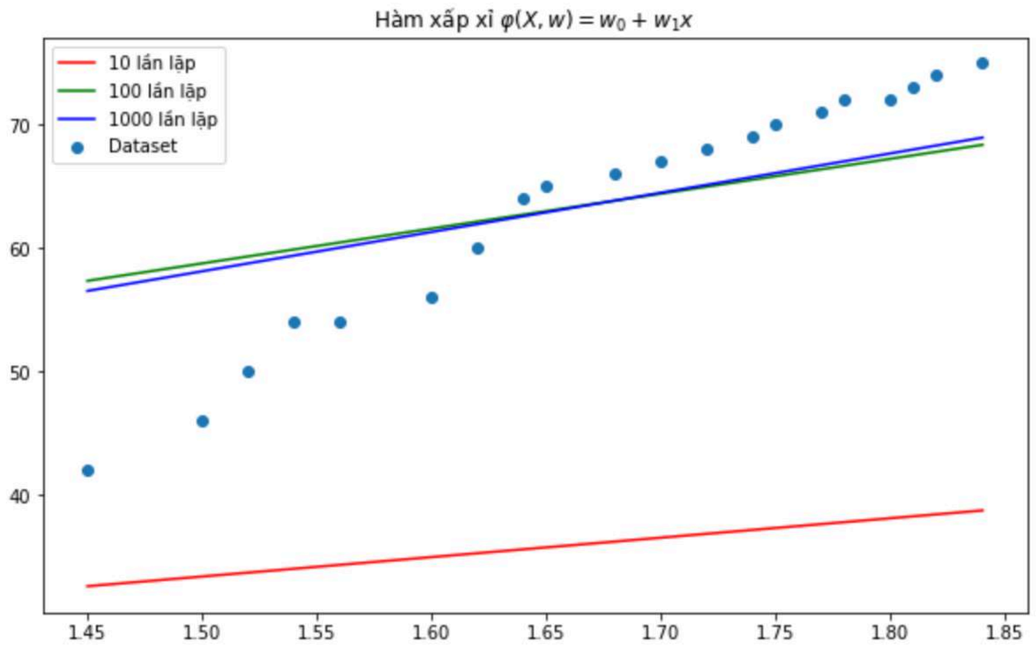
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

ERROR = 0.01

def phi(x):
    return w0 + w1*x
X_train = [...] # Như ví dụ trước
y_train = [...]
```

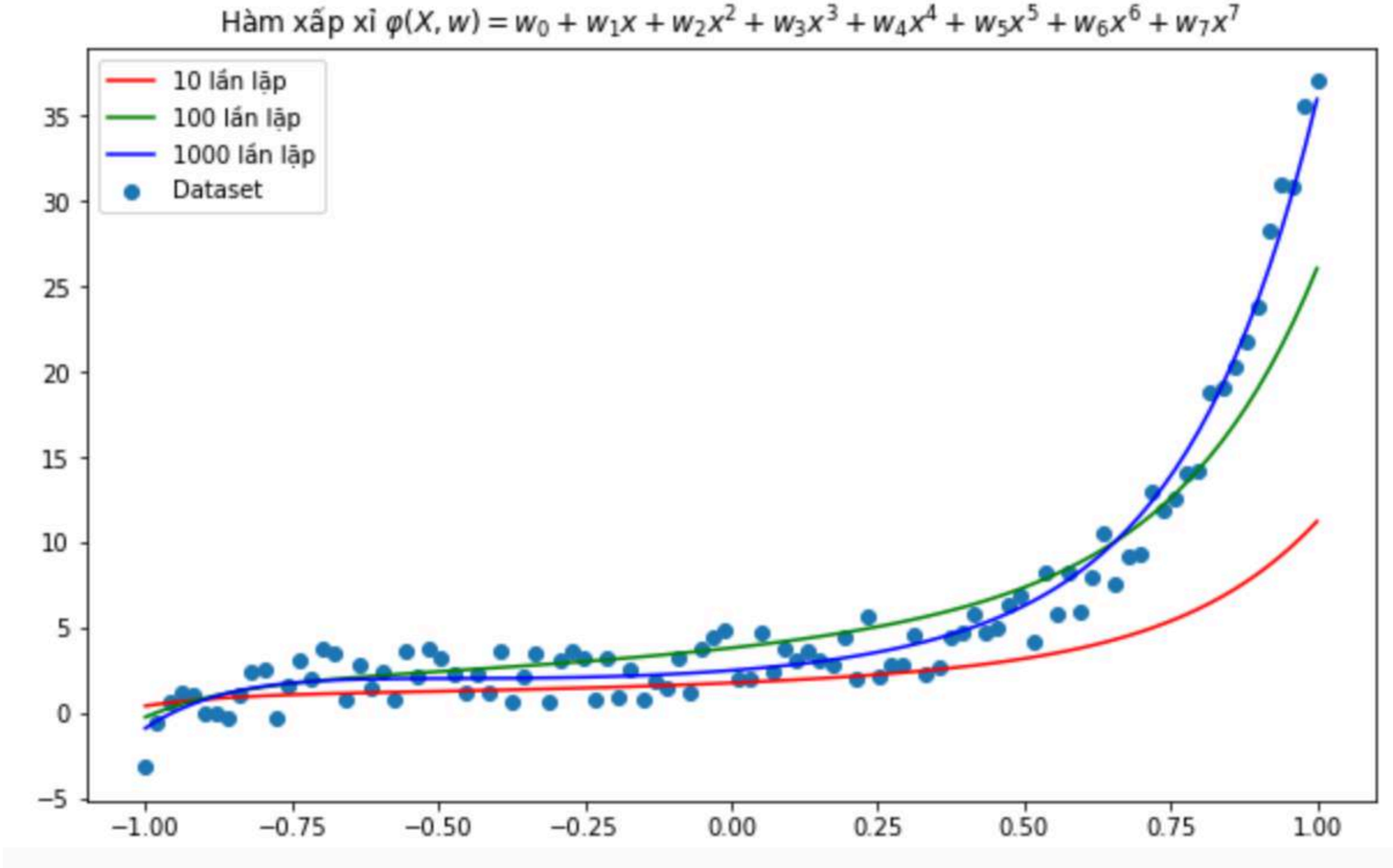
```
w0 = tf.Variable(1.)
w1 = tf.Variable(1.)

def manytimes( iter ):
  for _ in range(iter):
    with tf.GradientTape() as tape:
      y_cal = phi( X_train )
      L = tf.reduce_mean( tf.square(y_cal - y_train) )
    grads = tape.gradient(L, [w0,w1] )
    w0.assign_sub( grads[0]*ERROR ) # w0 -= grads[0]*ERROR
    w1.assign_sub( grads[1]*ERROR ) # w1 -= ...
  return phi(X_train)

y_pred1 = manytimes( 10 )
y_pred2 = manytimes( 100 )
y_pred3 = manytimes( 1000 )

plt.figure( figsize=(10,6))
plt.title( "Hàm xấp xỉ " + r"$\varphi(X,w) = w_0+w_1x$" )
plt.scatter( X_train, y_train, label="Dataset" )
plt.plot( X_train, y_pred1, 'r',label="10 lần lặp" )
plt.plot( X_train, y_pred2, 'g',label="100 lần lặp"  )
plt.plot( X_train, y_pred3, 'b',label="1000 lần lặp"  )
plt.legend()
plt.show()
```


File: LinearRegression-tensorflowver2-1D.py

Một ví dụ khác, giả sử ta muốn xấp xỉ dưới dạng đa thức bậc 7

$$\varphi(X, w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6 + w_7 x^7$$





- Với các thư viện như trên, tạo dataset ngẫn nhiên với kích thước là NUM_DATASET

```
ERROR = 0.01
NUM_DATASET = 100
LEVEL = 8

coeffs = []
for i in range(LEVEL):
    coeffs.append(i+1)

X_train = np.linspace( -1,1,NUM_DATASET )
y_train = 0
for i in range(LEVEL):
    y_train += coeffs[i] * np.power( X_train,i )
y_train += np.random.random(NUM_DATASET)*4
```

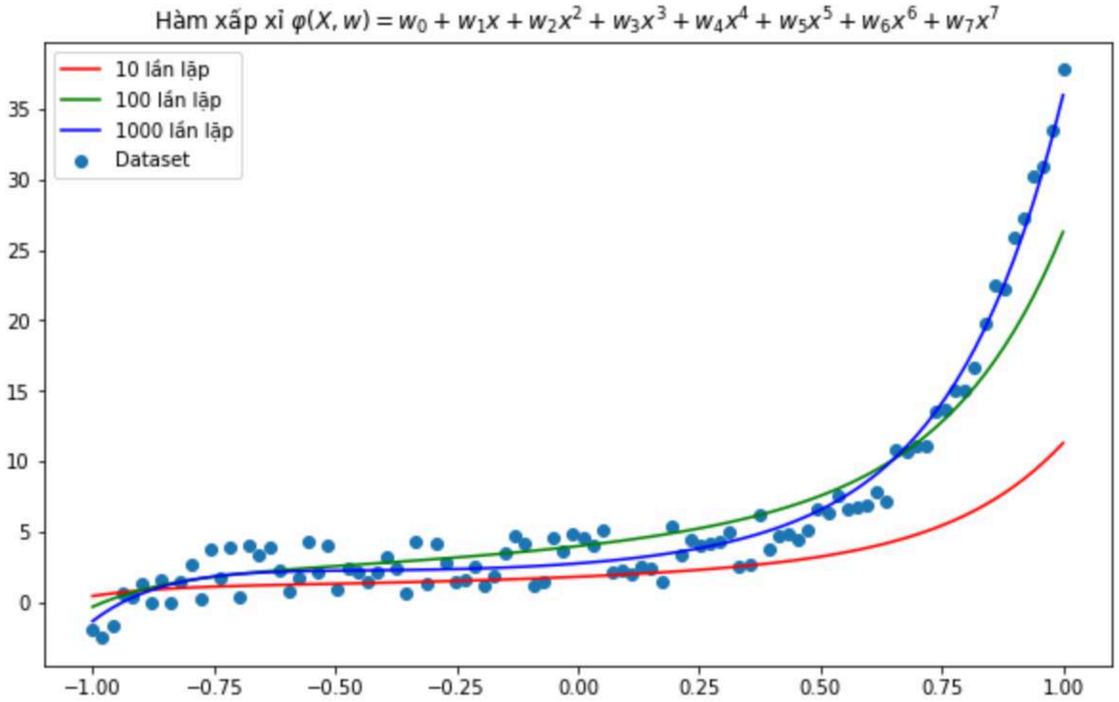
- Các hệ số $w_i = 1, \forall i = \overline{1,LEVEL}$

```
w = tf.Variable( [1.]*LEVEL )
def phi(x):
    y = 0
    for i in range(LEVEL):
        y += w[i]*pow(x,i)
    return y
```

- Huấn luyện để có mô hình

```
def manytimes(iter):
    for _ in range(iter):
        with tf.GradientTape() as tape:
            y_cal = phi( X_train )
            L = tf.reduce_mean( tf.square(y_cal -
y_train) )
        grads = tape.gradient( L,w )
        w.assign_sub( grads*ERROR )
    return phi( X_train )
```

File: PolynomialRegression-tensorflowver2.py



- Tính toán sau đó hiển thị kết quả như hình ở trước

```
y_pred1 = manytimes( 10 )
y_pred2 = manytimes( 100 )
y_pred3 = manytimes( 1000 )

plt.figure( figsize=(10,6))
plt.title( "Hàm xấp xỉ " + r"$\varphi(X,w) = w_0+w_1x+w_2x^2+w_3x^3+w_4x^4+w_5x^5+w_6x^6+w_7x^7$" )
plt.scatter( X_train, y_train, label="Dataset" )
plt.plot( X_train, y_pred1, 'r',label="10 lần lặp" )
plt.plot( X_train, y_pred2, 'g',label="100 lần lặp"  )
plt.plot( X_train, y_pred3, 'b',label="1000 lần lặp"  )
plt.legend()
plt.show()
```

Một ví dụ nữa

- Hàm xấp xỉ có dạng $\varphi(X, w) = w_0 + w_1 x_1 + w_2 x_2$
- Do để thuận lợi trong việc cài đặt chương trình, nên đặt $x_0 = 1$, khi đó $\varphi(X, w) = w_0 x_0 + w_1 x_1 + w_2 x_2$Vì vậy cần bổ sung thêm cột thứ nhất có giá trị là 1 vào X_train

```
ERROR = 0.01

X_train = np.array([[2.167,7.0], [3.1,6.0], ... ,[10.791,9.0]])
y_train = [1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366, 2.596, 2.53, 1.221, 2.827, 3.465, 1.65,
2.904, 2.42, 2.94, 1.3]

N = len(X_train)
onesMat = np.ones( (1,N) )
X = np.concatenate((onesMat, X_train.T), axis = 0)
x_train = X.T
```

- Các phần còn lại giả sử ban đầu $w_0 = 1, w_1 = 2, w_2 = 3$

```
w = tf.Variable( [1.0,2.0,3.0] )
```

- x ở đây là một ma trận

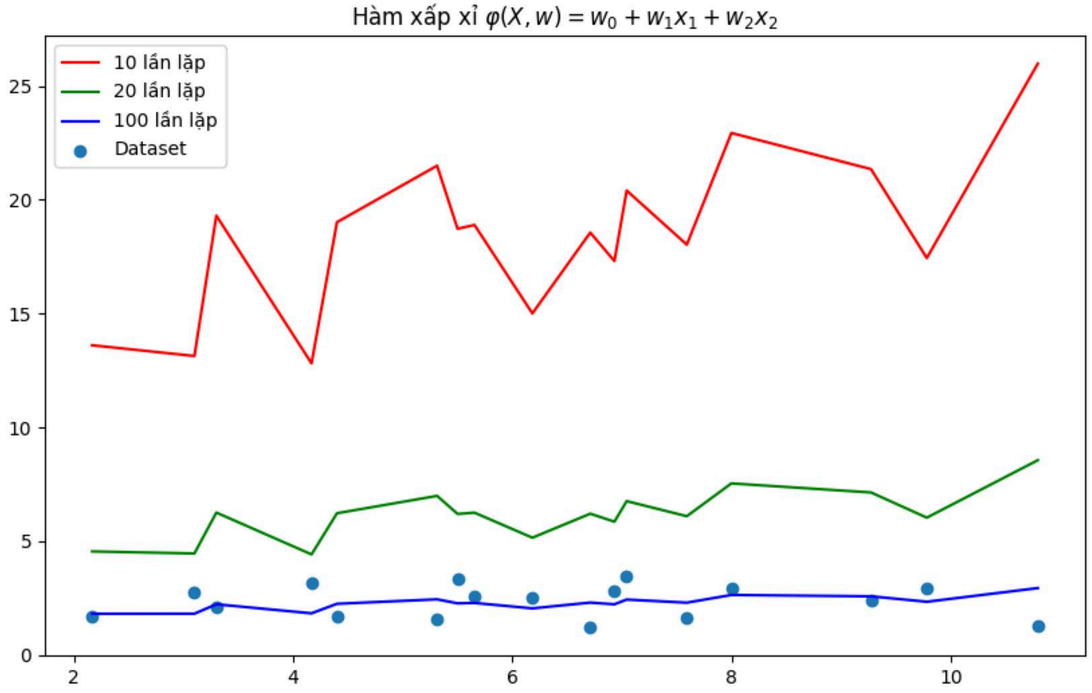
```
def phi(x):
    return w[0]*x[:,0] + w[1]*x[:,1] + w[2]*x[:,2]

def manytimes(iter):
    for _ in range(iter):
        with tf.GradientTape() as tape:
            y_cal = phi( x_train )
            L = tf.reduce_mean( tf.square(y_cal - y_train) )
        grads = tape.gradient( L,w )
        w.assign_sub( grads*ERROR )
    return phi( x_train )
```

- Tính toán thử 3 lần

```
y_pred1 = manytimes( 10 )
y_pred2 = manytimes( 20 )
y_pred3 = manytimes( 100 )
```

- Xuất ra dạng đồ thị để coi

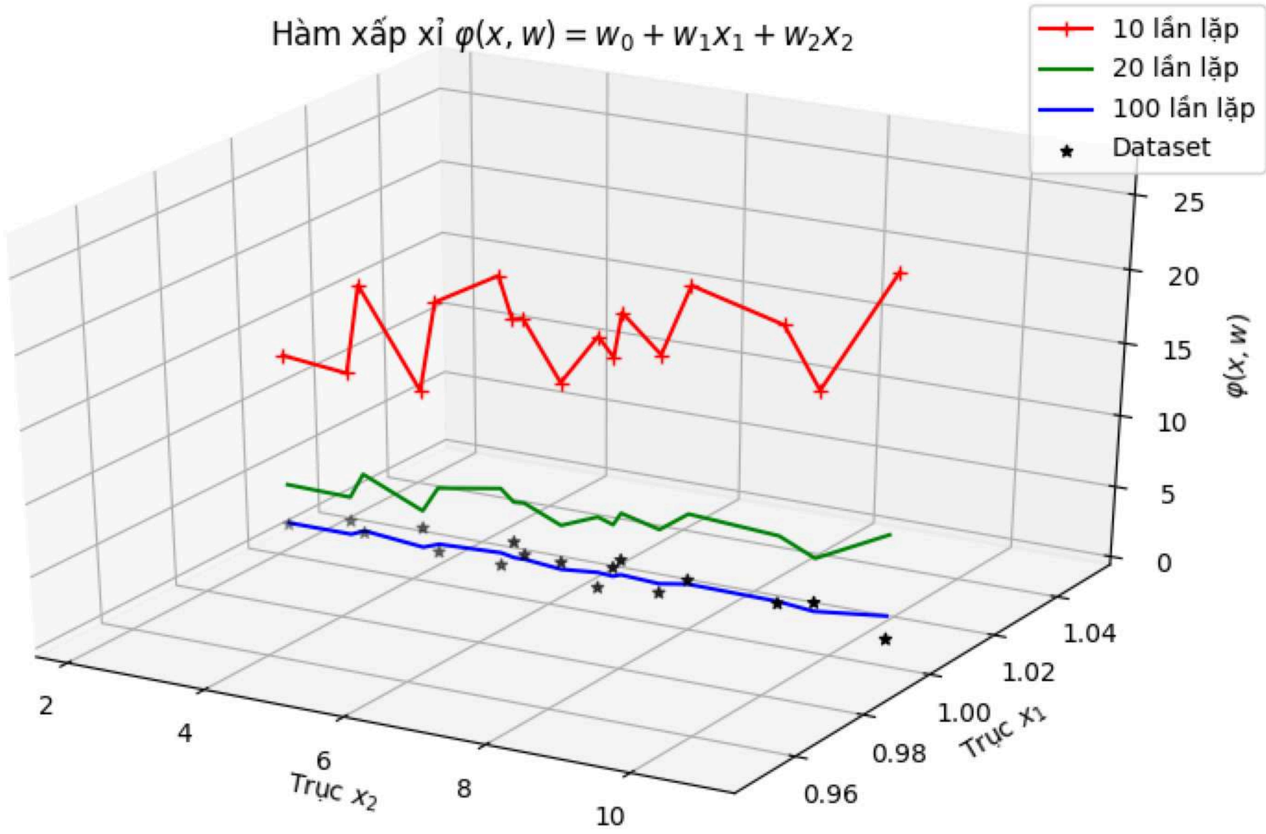
```
plt.figure( figsize=(10,6))
plt.title( "Hàm xấp xỉ " + r"$\varphi(X,w) =
w_0+w_1x_1+w_2x_2$" )
XX = x_train[:,1]
plt.scatter( XX, y_train, label="Dataset" )
plt.plot( XX, y_pred1, "red", label="10 lần lặp" )
plt.plot( XX, y_pred2, "green", label="20 lần lặp"  )
plt.plot( XX, y_pred3, "blue", label="100 lần lặp"  )
plt.legend()
plt.show()
```



File: LinearRegression-tensorflowver2-2D.py

- Thực ra phải ở dạng đồ thị 3 chiều

```
plt.figure( figsize=(10,6) )
ax = plt.axes( projection='3d' )
ax.set_title( "Hàm xấp xỉ " + r"$\varphi(x,w) =
w_0+w_1x_1+w_2x_2$" )
ax.set_xlabel( "Trục " + r"$x_2$" )
ax.set_ylabel( "Trục " + r"$x_1$" )
ax.set_zlabel( r"$\varphi(x,w)$" )
X0 = x_train[:,0]
X1 = x_train[:,1]
ax.plot(X1,X0,y_pred1,c='r',marker='+',label="10
lần lặp")
ax.plot(X1,X0,y_pred2,c='g',label="20 lần lặp")
ax.plot(X1,X0,y_pred3,c='b',label="100 lần lặp")
ax.scatter(X1,X0,y_train,color='black',marker='*
,label="Dataset")
ax.legend()
plt.show()
```



Bài tập

- Trên Kaggle tại https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset/download có tập tin covid_19_data.csv chứa nhiều thông tin liên quan đến Wuhan Coronavirus, hãy rút trích về một quốc gia nào đó chứa thông tin sau đây theo từng ngày mà có số liệu:
- Viết chương trình dự báo về số người bị nhiễm, người tử vong và đã âm tính

Confirmed	Deaths	Recovered
532	6	338
500	0	400
300	1	200
400	4	285