MACHINE LEARNING: BÀI THỰC HÀNH SỐ 2.

A. PHƯƠNG PHÁP K-NEAREST NEIGHBORS

Trong các ví dụ dưới đây, dữ liệu đầu vào và đầu ra đều có dạng số. Các bước của phương pháp K-NN có thể mô tả:

(i) Với mỗi **x** – unseen ở đầu vào (các phần tử trong tập Validation)

(ii) Tính khoảng cách $\mathbf{d_i}$ từ **x** đến các $\mathbf{x_i}$ trong tập Training => Lập thành 1 mảng.

(iii) Sắp xếp các khoảng cách này theo thứ tự tăng dần, nhưng cần có tham chiếu để biết chỉ số phần tử ban đầu trong mảng $\mathbf{d_i}$

(iv) Tìm ra K chỉ số của các phần tử $\mathbf{x_i}$ trong tập Training ứng với các $\mathbf{d_i}$ nhỏ nhất.

(v) Lấy đầu ra dự đoán y_pred = $(\Sigma\, y_i)/K$

Như vậy, chúng ta cần xây dựng hàm tính khoảng cách giữa **x** và các $\mathbf{x_i}$.

- Trong trường hợp biến đơn (không gian có d=1 chiều), ta sử dụng khoảng cách là $|\mathbf{x} - \mathbf{x_i}|$, trong python ta dùng **abs**($\mathbf{x} - \mathbf{x_i}$).
- Trong trường hợp d > 1, ta sử dụng khoảng cách Euclide.

Giả sử mảng các khoảng cách tương ứng là **Array_D**, lúc đó trong Python ta có thể dùng các công cụ sau để lấy chỉ số của K phần tử nhỏ nhất **Array_D** như:

- **np.argsort(array_D)** Phương thức này trả về chỉ số của các phần tử mảng array_D được sắp xếp tăng. Do đó ta có thể lấy **k** phần tử đầu tiên của dãy chỉ số trả về để thu được kết quả

```
indexes = np.argsort(array_D)[:k]
y_pred = 0
for i in range(k):
    y_pred = y_pred + y[indexes[i] ]
y_pred = y_pred/k
```

- **np.argpartition(array_D, k)** trả về mảng chỉ số, trong đó đảm bảo rằng **k** phần tử đầu tiên sẽ có giá trị nhỏ nhất và sắp xếp tăng. Chú ý phương thức này cũng như phương thức trên sẽ trả về tất cả chỉ số của toàn bộ mảng, nên khi lấy **k** phần tử nhỏ nhất, ta chỉ lấy các chỉ số từ **0:k.** Cách sử dụng nó để tính y_pred như sau

```
indexes = np.argpartition(array_D, k)[:k]
y_pred = 0
for i in range(k):
    y_pred = y_pred + y[indexes[i] ]
y_pred = y_pred/k
```

Code trên đây chỉ để giải thích tường minh các bước. Các bạn có thể dùng các phương thức có sẵn để tính toán.

Ví dụ A.1. Trong tệp dữ liệu SAT_GPA.csv đính kèm có 84 mẫu dữ liệu điểm thi của các sinh viên, mẫu có 02 trường dữ liệu, trong cột thứ nhất chứa trường điểm SAT (Reading + Mathematic + Writing) của các kỳ thi trong bậc phổ thông; cột thứ hai chứa điểm trung bình GPA của sinh viên tương ứng ở bậc học đại học/cao đẳng. Chúng ta sử dụng phương pháp **K-nearest neighbors** (K-NN) để dự đoán điểm GPA ở bậc đại học/cao đẳng của một sinh viên bất kỳ dựa vào điểm SAT ở bậc phổ thông, bằng cách lấy trung bình điểm SAT của K sinh viên gần nhất.

Dữ liệu đều có dạng số, với đầu vào X (điểm SAT) và đầu ra Y (điểm GPA).

Trong ví dụ này, chúng ta có thể sử dụng thư viện panda để đọc tệp csv. Đoạn đọc dữ liệu từ tệp dạng csv bằng thư viện panda như dưới đây, các bạn cần sửa lại để có đường dẫn dữ liệu phù hợp:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Change to data path on your computer
data = \
pd.read_csv("D:\\Teach_n_Train\\Machine_Learning\\exam_n_practice\\linear_reg\\SAT_GPA.csv")
# Show the description of data
data.describe()

# Set to training data (x, y)
y = data['GPA']
x = data['SAT']

# Remind that we need to put component x_0 = 1 to x
plt.scatter(x,y)
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.show()
```

04 dòng lệnh cuối là để hiển thị dữ liệu lên mặt phẳng tọa độ.

- Coi 64 mẫu dữ liệu đầu là tập training và các mẫu còn lại là tập validation.
- Lấy k = 8 = sqrt(64), thực hiện phương pháp K-NN để dự đoán đầu ra cho tập Validation.
- Đoạn chương trình tham khảo (nối tiếp vào đoạn đọc dữ liệu ở phần đầu bài) như sau:

```
k = 8

def distance(array, value):
    array = np.array(array)
    return abs(array - value)

def find_nearest_index(array, value, k):
    array_D = distance(array, value)

    return np.argsort(array_D)[:k]


data_len = len(x)

X_train = np.array(x[:64])
Y_train = np.array(y[:64])

X_test = np.array(x[64:data_len])
Y_test = np.array(y[64:data_len])

k = 8
```

```
Y_pred = np.zeros(len(X_test))

for i in range(len(X_test)):
    indexis = find_nearest_index(X_train, X_test[i], k)
    for id in indexis:
        Y_pred[i] = Y_pred[i] + Y_train[id]
    Y_pred[i] = Y_pred[i]/len(indexis)
    print(Y_pred[i], ' | ', Y_test[i])
```

- Hãy sử dụng các độ đo: MSE; MAE; R-square để đánh giá kết quả.

Ví dụ A.2. (Xem bài giảng lý thuyết phần Hồi quy tuyến tính): Ước lượng áp suất khí quyển (tính theo atm) tại một địa điểm dựa trên nhiệt độ sôi của nước (theo độ F) tại địa điểm đó. Mẫu dữ liệu quan sát từ các thí nghiệm được cho như trong bảng sau

STT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Nhiệt độ	194.5	194.3	197.9	198.4	199.4	199.9	200.9	201.1	201.4	201.3	203.6	204.6	209.5	208.6	210.7	211.9	212.2
Áp suất	20.79	20.79	22.4	22.67	23.15	23.35	23.89	23.99	24.02	24.01	25.14	26.57	28.49	27.76	29.04	29.88	30.06

Khởi tạo dữ liệu (sau đó vẽ ra để hình dung dữ liệu) trong python như sau:

```
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt

# Temp (F degree)

X = np.array([[194.5, 194.3, 197.9, 198.4, 199.4, 199.9, 200.9, 201.1, 201.4, 201.3, 20
3.6, 204.6, 209.5, 208.6, 210.7,   211.9, 212.2]]).T

# Press (Atm)

y = np.array([[20.79,20.79,22.4,22.67,23.15,23.35,23.89,23.99,24.02,24.01,25.14,26.57,2
8.49,27.76,29.04,29.88,30.06]]).T

# Visualize data

plt.plot(X, y, 'ro')

plt.axis([193, 213, 19, 31])

plt.xlabel('Temperature (F)')

plt.ylabel('Pressure (Atm)')

plt.show()
```

- Hãy chia dữ liệu thành phần Training gồm 16 mẫu đầu tiên; phần Validation là các mẫu còn lại.
- Chọn k = 4
- Thực hiện thuật toán K-NN dự đoán các giá trị đầu ra của tập Validation.
- Đánh giá kết quả dựa trên các độ đo MSE, MAE và R-square.

Ví dụ A.3. (Xem bài giảng lý thuyết phần Hồi quy tuyến tính): Trong ví dụ này, ta sử dụng phương pháp **K-NN** để dự báo mức độ tiêu thụ nhiên liệu trong 50 bang của Hoa Kỳ và quận Columbia và tìm hiểu hiệu ứng của tiêu thụ nhiên liệu đối với thuế xăng của các bang.

Các biến dự báo được sử dụng trong *Applied Linear Regression*, 3rd edition, Sanford Weisberg, Wiley-Interscience, 2005. Dữ liệu được thu thập bởi Cục Đường bộ Hoa Kỳ vào năm 2001 và được cho trong tệp fuel.txt, gồm các trường như sau:

Các biến dự báo được sử dụng trong *Applied Linear Regression*, 3rd edition, Sanford Weisberg, Wiley-Interscience, 2005. Dữ liệu được thu thập bởi Cục Đường bộ Hoa Kỳ vào năm 2001 và được cho trong tệp fuel.txt, gồm các trường như sau:

State	Tên bang
Drivers	Số bằng lái được cấp phép trong bang
FuelC	Lượng xăng sử dụng cho giao thông đường bộ, theo ngàn gallons
Income	Thu nhập bình quân đầu người năm 2000, theo ngàn đôla
Miles	Số dặm đường cao tốc của bang được hỗ trợ từ liên bang
MPC	(bỏ qua trường này)
Pop	Dân số lớn hơn hoặc bằng 16 tuổi
Tax	Thuế xăng của bang, theo cents trên một gallon

Phần đọc dữ liệu chúng ta sẽ bỏ qua trường State và MPC. Tuy nhiên ở đây chúng ta sẽ sử dụng **một số thuộc tính quy đổi**:

Fuel	1000 × FuelC/Pop
Dlic	1000 × Drivers/Pop
log(Miles)	Loga cơ số 2 của Miles

Đoạn code đọc dữ liệu như sau:

```
import math
import numpy as np
with open('fuel.txt') as f:
   lines = f.readlines()

x_data = []
y_data = []
lines.pop(0)

for line in lines:
    splitted = line.replace('\n', '').split(',')
    splitted.pop(0)
    splitted = list(map(float, splitted))
    fuel = 1000 * splitted[1] / splitted[5]
    dlic = 1000 * splitted[0] / splitted[5]
    logMiles = math.log2(splitted[3])
    y_data.append([fuel])
    x_data.append([splitted[-1], dlic, splitted[2], logMiles])

x_data = np.asarray(x_data)
```

```
y_data = np.asarray(y_data)
```

- Hãy chia dữ liệu thành phần Training gồm 40 mẫu đầu tiên; phần Validation là các mẫu còn lại.
- Chọn k = 6
- Thực hiện thuật toán K-NN dự đoán các giá trị đầu ra của tập Validation.
- Đánh giá kết quả dựa trên các độ đo MSE, MAE và R-square.

Ở đây do dữ liệu là nhiều chiều (d = 4), nên hàm tính khoảng cách cần được xây dựng lại. Do dữ liệu dạng số nên chúng ta có thể sử dụng khoảng cách Euclide (chuẩn 2 của vector hiệu):

```
def distance(array, value):
    array = np.array(array)
    return np.linalg.norm(array - value, ord = 2, axis=0)
```

(Cần tự xem xét axis = 0 hay bằng 1!)

B. PHƯƠNG PHÁP HỒI QUY TUYẾN TÍNH

Ví dụ B.1. Lấy lại dữ liệu như trong Ví dụ A.2 - Ước lượng áp suất khí quyển (tính theo atm) tại một địa điểm dựa trên nhiệt độ sôi của nước (theo độ F) tại địa điểm đó (Xem bài giảng lý thuyết):

- Viết chương trình để xây dựng công thức hồi quy tuyến tính cho mối liên hệ giữa nhiệt độ và áp suất.
- Ta có:
    - Đây là mô hình hồi quy đơn, một biến, vậy cần tìm 2 hệ số θ_0, θ_1 cho phương trình đường hồi quy $y = f(x) = \theta_0 + \theta_1 x$.
    - Ta có thể tính các hệ số mà không cần dùng khai triển QR.

a. **Tính toán trực tiếp theo công thức cho trường hợp riêng hồi quy đơn, một biến**.

- Trung bình cộng của X: $\bar{x} = 202.9529412$; Trung bình cộng của Y: $\bar{y} = 25.05882353$;
- Phương sai SXX = 530.7823529; Hiệp phương sai SXY = 277.5420588.
- Theo công thức:
    - θ_1 = SXY/SXX = 0.522892401; $\theta_0 = : \bar{y} - \theta_1\bar{x}$= -81.06372713.

b. Chương trình Python

- Khởi tạo dữ liệu (sau đó vẽ ra để hình dung dữ liệu) – Xem trong Ví dụ A.2.
- Đoạn chương trình tính các hệ số w0 = $\theta_0 + \theta_1$ và w1=θ_1 (cho phương trình y = $\theta_0 + \theta_1$x = w0 + w1x). Ở đây ta dùng thư viện numpy.linalg để thực hiện tính nghịch đảo (dùng phương thức pinv để tính tựa nghịch đảo (pseudo inverse) của ma trận

```
# Building Xbar
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1)


# Calculating weights of the fitting line
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
```

```
w = np.dot(np.linalg.pinv(A), b)
print('w = ', w)
# Preparing the fitting line
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(193, 213, 2)
y0 = w_0 + w_1*x0

# Drawing the fitting line
plt.plot(X.T, y.T, 'ro')     # data
plt.plot(x0, y0)               # the fitting line
plt.axis([193, 213, 19, 31])
plt.xlabel('Temperature (F)')
plt.ylabel('Pressure (Atm)')
plt.show()
```

- Trong đoạn chương trình trên ta dùng thư viện matplotlib.pyplot để vẽ đường xấp xỉ (siêu phẳng xấp xỉ) của mô hình hồi quy. Các bạn tự tìm hiểu các phương thức plot, axis, xlabel, ylabel, show đã được sử dụng để có thể dùng lại sau này.
- Kết quả được in ra sẽ tương đối khớp với kết quả trong bài giảng hoặc tính theo công thức.
- Một số phương thức chúng ta cần tự tìm hiểu (trong ví dụ này thì chủ yếu trong thư viện Numpy): numpy.dot (nhân hai ma trận); numpy.linalg.pinv (tính nghịch đảo suy rộng của một ma trận)

Ví dụ B.2. (Xem bài giảng lý thuyết): Trong ví dụ này, ta sử dụng mô hình hồi quy tuyến tính để dự báo mức độ tiêu thụ nhiên liệu trong 50 bang của Hoa Kỳ và quận Columbia và tìm hiểu hiệu ứng của tiêu thụ nhiên liệu đối với thuế xăng của các bang.

Các biến dự báo được có trong tệp như được mô tả trong **Ví dụ A.3.** Lưu ý trong mô hình chúng ta vẫn sử dụng **một số thuộc tính quy đổi**:

Fuel	1000 × FuelC/Pop
Dlic	1000 × Drivers/Pop
log(Miles)	Loga cơ số 2 của Miles

Chúng ta cần xác định các tham số cho mô hình

$$\text{Fuel} = \text{E(X)} = \theta_0 + \theta_1\text{Tax} + \theta_2\text{Dlic} + \theta_3\text{Income} + \theta_4\text{logMiles}$$

Đây là trường hợp hồi quy đơn, nhiều biến (4 biến) và số dữ liệu là 51. Vậy chúng ta cần đến khai triển QR để giải hệ nếu chỉ sử dụng thư viện numpy. Nếu ta sử dụng thư viện scikit-learn, chúng ta có thể gọi luôn mô hình hồi quy tuyến tính. Chúng ta sẽ xem xét cả hai phương pháp.

Đọc dữ liệu từ tệp

```
import math
import numpy as np
with open('fuel.txt') as f:
```

```
    lines = f.readlines()

x_data = []
y_data = []
lines.pop(0)

for line in lines:
    splitted = line.replace('\n', '').split(',')
    splitted.pop(0)
    splitted = list(map(float, splitted))
    fuel = 1000 * splitted[1] / splitted[5]
    dlic = 1000 * splitted[0] / splitted[5]
    logMiles = math.log2(splitted[3])
    y_data.append([fuel])
    x_data.append([splitted[-1], dlic, splitted[2], logMiles])

x_data = np.asarray(x_data)
y_data = np.asarray(y_data)
```

a) **Trường hợp sử dụng Numpy** – Dùng thuật toán HoldHouse để khai triển QR

Trường hợp này chúng ta sẽ tự viết hàm hồi quy và hàm khai triển QR, chỉ dùng thư viện Numpy:

```
def qr_householder(A):
    """ Compute QR decomposition of A using Householder reflection"""
    M = A.shape[0]
    N = A.shape[1]

    # set Q to the identity matrix
    Q = np.identity(M)

    # set R to zero matrix
    R = np.copy(A)

    for n in range(N):
        # vector to transform
        x = A[n:, n]
        k = x.shape[0]
```

```
        # compute ro=-sign(x0)||x||
        ro = -np.sign(x[0]) * np.linalg.norm(x)

        # compute the householder vector v
        e = np.zeros(k)
        e[0] = 1
        v = (1 / (x[0] - ro)) * (x - (ro * e))

        # apply v to each column of A to find R
        for i in range(N):
            R[n:, i] = R[n:, i] - (2 / (v@v)) * ((np.outer(v, v)) @ R[n:, i])

        # apply v to each column of Q
        for i in range(M):
            Q[n:, i] = Q[n:, i] - (2 / (v@v)) * ((np.outer(v, v)) @ Q[n:, i])

    return Q.transpose(), R

def linear_regression(x_data, y_data):
#     """
#     This function calculate linear regression base on x_data and y_data
#     :param x_data: vector
#     :param y_data: vector
#     :return: w (regression estimate)
#     """

    # add column 1
    x_bars = np.concatenate((np.ones((x_data.shape[0], 1)), x_data), axis=1)

    Q, R = qr_householder(x_bars)  # QR decomposition
    R_pinv = np.linalg.pinv(R)  # calculate inverse matrix of R
    A = np.dot(R_pinv, Q.T)  # apply formula

    return np.dot(A, y_data)
```

Đoạn lệnh gọi và chạy chương trình

```
w = linear_regression(x_data, y_data)  # get result
w = w.T.tolist()
```

```
line = ['Intercept', 'Tax', "Dlic", "Income", 'LogMiles']
res = list(zip(line, w[0]))
for o in res:
    print("{: >20}: {: >10}".format(*o))
```

b) Trường hợp sử dụng thư viện Scikit-Learn

Trước hết chúng ta vẫn cần đoạn code để lấy dữ liệu như ở trên. Sau đó, đoạn code để gọi thư viện hồi quy tuyến tính trong gói scikit learn chỉ đơn giản như sau:

```
from sklearn import datasets, linear_model
# Load training data here and assign to Xbar (obs. Data) and y (label)
# fit the model by Linear Regression
regr = linear_model.LinearRegression(fit_intercept=False)
# fit_intercept = False for calculating the bias
regr.fit(Xbar, y)
```

Bài tập B.1. Sử dụng 40 mẫu trong tập dữ liệu của ví dụ 2 (coi là tập training) để huấn luyện các đoạn chương trình đã có, sau đó dự đoán đầu ra với các mẫu dữ liệu còn lại (coi là tập validation). Hãy dùng các độ đo sau đây để đánh giá độ chính xác của mô hình trên các tập training và validation: MSE; MAE; R-square.

Ví dụ B.3 (Bài tập tự thực hành 1). Sử dụng lại tệp dữ liệu cũng như đoạn lệnh đọc dữ liệu từ **Ví dụ A.1**. Tuy nhiên trong bài tập này các bạn cần xây dựng mô hình **Hồi quy tuyến tính** để dự đoán. Thực hiện các yêu cầu sau đây:

(i) Sử dụng hệ số tương quan Pearson correlative để đánh giá mức độ phụ thuộc của đầu ra (GPA) với đầu vào (SAT).

(ii) Hãy chia dữ liệu thành phần training với 64 mẫu đầu và validation với các mẫu còn lại.

(iii) Tham khảo các ví dụ trước, sau đó lập công thức hồi quy tuyến tính ứng với dữ liệu training nói trên. Hiển thị đồ thị của đường hồi quy với các điểm dữ liệu đã vẽ ở phần code đã cho.

(iv) Chạy thử mô hình với dữ liệu validation và tính các độ đo đánh giá sau: MSE; MAE; R-square.

(v) So sánh kết quả này với kết quả sử dụng phương pháp K-NN (K-Nearest Neighbors) trong Ví dụ A.1.

Nếu đã có các hệ số t_0, t_1 (θ_0, θ_1), đoạn code sau sẽ vẽ ra đường hồi quy:

```
plt.scatter(x1,y)
yhat = t_1*x1 + t_0
fig = plt.plot(x1,yhat, lw=4, c='orange', label = 'regression line')
plt.xlabel('SAT', fontsize = 20)
```

```
plt.ylabel('GPA', fontsize = 20)

plt.show()
```

Ví dụ B.4 (Bài tập 2). Trong y sinh học, bề dày lớp nội trung mạc (NTM) phản ánh một số bệnh lý của cơ thể. Thực tế hiện tượng dày lớp NTM động mạch cảnh do nhiều yếu tố như di truyền, chủng tộc, mắc bệnh tim mạch, tuổi, giới, BMI, tăng huyết áp, đái tháo đường.... cùng tác động. Trong ví dụ này ta không đề cập các yếu tố di truyền, chủng tộc, giới, mắc bệnh tim mạch... mà chỉ lưu ý đến các biến số như: tuổi, cholesterol, glucose, huyết áp tâm thu và BMI tác động lên độ dày NTM.

Hãy dùng dữ liệu cho trong tệp vidu4_lin_reg.txt (tệp văn bản) để xây dựng mô hình hồi quy tuyến tính cho thấy sự phụ thuộc của bề dày lớp NTM theo các biến số khác. Tham khảo phần đọc dữ liệu từ tệp văn bản đã có trong ví dụ trước. Các trường dữ liệu gồm:

ID	Mã bệnh nhân
TUOI	Tuổi
BIM	chỉ số khối lượng cơ thể (Body Mass Index)
HA	huyến áp tâm thu
GLUCOSE	đường huyết
CHOLESTEROL	độ Cholesterol trong máu
BEDAYNTM	độ dày NTM

Mô hình cần xây dựng có dạng:

$$\textbf{Bề dày NTM} = \beta_0 + \beta_1(\textbf{tuổi}) + \beta_2(\textbf{cholesterol}) + \beta_3(\textbf{glucose}) + \beta_4(\textbf{huyết áp TT}) + \beta_5(\textbf{BMI})$$

a) Xác định các hệ số với 100 dữ liệu trên

b) Chia dữ liệu thành: 80 dòng đầu dùng cho training; 20 dòng sau dùng cho testing. Tính lại các hệ số với bộ dữ liệu này, sau đó chạy thử trên bộ dữ liệu test và tính các đại lượng kỳ vọng, phương sai của sai số.

Ví dụ B.5 (Bài tập 3). Trong tệp dữ liệu Real_estate.csv đính kèm chứa thông tin các giao dịch mua bán bất động sản. Chúng ta có 414 mẫu dữ liệu, mỗi bản ghi có 8 cột theo thứ tự là

- Cột x1: Số thứ tự (chúng ta sẽ bỏ qua trường này)
- Cột x2: Ngày giao dịch mua bán (ta chỉ lấy phần nguyên là năm)
- Cột x3: Tuổi của căn nhà (theo năm)
- Cột x4: Khoảng cách tới ga MRT (phương tiện công cộng nội đô) gần nhất
- Cột x5: Số cửa hàng tiện ích gần đó
- Cột x6: Kinh độ căn nhà; Cột X7: Vĩ độ căn nhà;
- Cột Y (đầu ra dự báo): Giá của căn nhà

Hãy chia dữ liệu thành phần training với 350 mẫu đầu tiên, phần validation với số mẫu còn lại. Hãy tham khảo các bài trên và xây dựng mô hình hồi quy tuyến tính mô tả sự phụ thuộc của Y vào các cột từ X2 đến X6. Sau đó hãy chạy dự đoán cho phần dữ liệu validation và đưa ra tổng bình phương sai số của dự đoán.

So sánh với phương pháp K-NN, lấy K = 18.