### Tóm tắt Linear Regression

#### **Công thức Gradient Descent**

#### **MSE Loss fucntion**

#### **Giải bài toán bằng đại số tuyến tính**

là vector hệ số cần phải tối ưu, w0 thường được gọi là bias.

* *Giải thích : w là ma trận (m+1) hàng và 1 cột để lưu các hệ số cần tìm*

là hàng dữ liệu thứ i trong bộ n số lượng dữ liệu quan sát được, mỗi dữ liệu có m giá trị .

* *Giải thích : X là ma trận đầu vào có n mẫu dữ liệu và m biến đầu vào (feature) thì ta sẽ tạo ra ma trận n hàng (m + 1) cột (do thêm cột 1 ở đầu cho tham số bias w0)*
* *Ta sẽ có ma trận* X =  *(ma trận n hàng và (m+1) cột) (i chạy từ 1 đến n)*

Ta có : Giá trị dự đoán y hat của thàng dữ liệu thứ i là

* *Giải thích : X là ma trận n*

Theo như hàm mất mát MSE Loss function :

**hoặc**

Ta có :

* *Giải thích : MES cho ma trận X là mất mát của n hàng dữ liệu*

Định nghĩa Euclideannorm :

* *Giải thích : Trong công thức Euclideannorm thì*  *, số 2 ở dưới là chuẩn 2 tức là căn bậc 2 và số 2 ở trên là mũ 2 và mũ hai của căn bậc 2 tức là hết căn có nghĩa là đơn giản ta chỉ bình phương của căn thôi .*

Mặc khác ta có :

Ta được

Viết lại

* *Giải thích :*
* *Ta thấy :*
* *Mà ( =*

*= ( + (*

= *Nhân 2 ma trận 1n và n1*

*Từ đó suy ra :*

Bây giờ tìm đi tìm w sao cho L (giá trị mất mát) là nhỏ nhất , như thường lệ ta sẽ khảo sát , nghĩa là ta sẽ đạo hàm L theo w và tìm nghiệm w để cho

Lưu ý :

+ X là một ma trận bất kì thì ta có thì A sẽ luôn là một ma trận đối xứng nghĩa là

+ (công thức liên quan đến ma trận chuyển vị trong đại số tuyến tính)

Giải bài toán

L = . Ta cóthể cho L = vì là hằng

L =

=

Đạo hàm :

=

=

=

=

=

=

* Cần đi tìm

Tìm :

+ \*

+ =

*+*  (Nếu X đối xứng thì )

Quay lại bài toán

*Giải thích : Đặt . Mà H là ma trận đối xứng .*

*Áp dụng kết quả đã được chúng minh ở trên ↔ =*

Ta có :



Kết luận : Nếu khả nghịch (có ma trận nghịch đảo) , thì L có nghiệm duy nhất :

Nếu không khả nghịch, ta có thể sử dụng khái niệm giả nghịch đảo.

Như vậy ta đã có hàm Gradient descent và nghiệm

* Sử dụng công thức w = w – learning\_rate \* gradient sẽ thu được w tối ưu sau **epochs** lần lặp
* Hoặc có thể cho nó dừng sớm nếu và chênh lệch không quá **epsilon** .

#### **Code Linear Regression**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

def matrix\_transformation(x, y) :

    x = np.concatenate((np.ones((x.shape[0], 1)), x), axis=1)

    y = np.array(y).reshape(-1, 1)

    return x, y

def linear\_regression(X, y, lr=0.05, epsilon=1e-10, epochs=100000):

    ep = 0

    N = X.shape[0]

    w = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y)

    while ep < epochs:

        gradient = np.dot(X.T, (np.dot(X, w) - y)) / N

        w = w - lr \* gradient

        if np.linalg.norm(gradient) < epsilon:

            break

        ep += 1

    return w

def predict(X, w):

    predictions = X.dot(w)

    return predictions

data = pd.read\_csv("USA\_Housing.csv")

data.head()

X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population']]

y = data['Price']

np.random.seed(42)

random\_indices = np.random.permutation(len(X))

train\_size = int(0.7 \* len(X))

X\_train = X.iloc[random\_indices[:train\_size]]

y\_train = y[random\_indices[:train\_size]]

X\_test = X.iloc[random\_indices[train\_size:]]

y\_test = y[random\_indices[train\_size:]]

X\_train, y\_train = matrix\_transformation(X\_train, y\_train)

X\_test, y\_test = matrix\_transformation(X\_test, y\_test)

w = linear\_regression(X\_train, y\_train, lr=0.05, epsilon=1e-10, epochs=100000)

predictions = predict(X\_test, w)

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, predictions)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red')

plt.xlabel('Thực tế')

plt.ylabel('Dự đoán')

plt.title('So sánh giữa giá trị thực tế và giá trị dự đoán')

plt.show()

