1 General information

MSSV	Họ và Tên	Môn học	Bài tập
21120566	Nguyễn Hữu Thuận	Học Thống Kê	Lab 1 - Multivariate Linear Regression

2 Explanation for variables

```
{
    "alpha": 0.01,
    "iters": 1000
}
```

2.1 alpha

Đây được gọi là *learning rate* (tốc độ học), được sử dụng để điều chỉnh tốc độ học của hầu hết các thuật toán *optimization* (tối ưu hoá), bao gồm *gradient descent* và nhiều thuật toán cải thiến khác.

Hệ số này thể hiện mức độ ảnh hưởng của từng *parameter* (tham số) đến giá trị *loss function* (hàm mất mát). *Learning rate* càng lớn thì mức độ ảnh hưởng càng lớn, và ngược lại.

Tùy theo mục đích của model mà ta có thể tăng/giảm *learning rate*. *Learning rate* lớn hơn sẽ giúp model học nhanh hơn và tiết kiệm thời gian, nhưng cũng sẽ gây ra sự thay đổi giữa các parameter càng lớn, dẫn đến việc training không ổn định, dao động mạnh và có thể không hội tụ được. Ngược lại, *learning rate* nhỏ hơn sẽ giúp model có thể hội tụ, nhưng cũng sẽ tốn nhiều thời gian hơn và dễ bị mắc kẹt ở *local minima* (cực tiểu địa phương) hoặc *saddle points* (điểm yên ngựa).

```python
def calculate_absolute_error(self, X, y, theta):
    error = np.abs(np.dot(X, theta) - y)
    return np.mean(error)
```

Ở đây, *loss function* là một *mean absolute error* (sai số trung bình tuyệt đối), như công thức sau:

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}|h_\theta(x^{(i)}) - y^{(i)}|$$

2.2 iters

Đây là số lần lặp của thuật toán *gradient descent*.

Thuật toán *gradient descent* sẽ cố gắng tìm ra giá trị của *parameter* sao cho *loss function* đạt giá trị nhỏ nhất. Để làm được điều này, thuật toán sẽ cố gắng tìm ra đạo hàm của *loss function* theo từng *parameter*, sau đó cập nhật giá trị của *parameter* theo hướng ngược với *gradient vector*.

Thường thuật toán toán sẽ lặp lại quá trình này cho đến khi đạt được số lần lặp đã định trước, hoặc khi đạo hàm của *loss function* theo từng *parameter* đạt giá trị nhỏ hơn một ngưỡng ϵ nhất định. Tuy nhiên, trong source code này, ta không xét đến điều kiện thứ hai.

```python
def gradient_descent(self, X, y):
    X = np.c_[np.ones(len(X),dtype='int64'), X]
    theta = np.zeros(X.shape[1])
    for i in range(self.iters):
        J, dJ = self.gradient(X, y, theta)
        theta -= self.alpha * dJ
    return theta
```