**Multicollinearity (Đa cộng tuyến) => Double Errors**

Trong mô hình hồi quy, nếu các biến độc lập có quan hệ chặt với nhau, các biến độc lập có mối quan hệ tuyến tính, nghĩa là các biến độc lập có tương quan chặt, mạnh với nhau thì sẽ có hiện tượng đa cộng tuyến, đó là hiện tượng các biến độc lập trong mô hình phụ thuộc lẫn nhau và thể hiện được dưới dạng hàm số. Ví dụ có hai biến độc lập A và B, khi A tăng thì B tăng, A giảm thì B giảm…. thì đó là một dấu hiệu của đa cộng tuyến. Nói một cách khác là hai biến độc lập có quan hệ rất mạnh với nhau, đúng ra hai biến này nó phải là 1 biến nhưng thực tế trong mô hình nhà nghiên cứu lại tách làm 2 biến. Hiện tượng đa cộng tuyến vi phạm giả định của mô hình hồi qui tuyến tính cổ điển là các biến độc lập không có mối quan hệ tuyến tính với nhau.

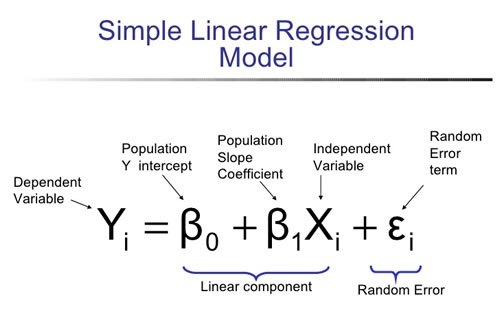
Đa cộng tuyến hoàn hảo:

|  |  |  |
| --- | --- | --- |
| **X2** | **X3** | **X4** |
| 10 | 50 | 52 |
| 15 | 75 | 75 |
| 18 | 90 | 97 |
| 24 | 120 | 129 |

**Nhận Xét:** **X2** và **X3** có mối quan hệ tuyến tính chính xác: **X3** = 5**X2**

Giả sử chúng ta ước lượng hàm tiêu dùng. **Y** = tiêu dùng, **X2** = thu nhập và **X3** = của cải

* **Y** = β1 + β2**X2** + β3**X3 +**
* **X3** = 5**X2**
* **Y** = β1 + β2**X2** + β35**X2 +**
* **Y** = β1 + (β2 + 5β3)**X3 +**



**Nguồn Gốc Của Multicollinearity**

Do phương pháp thu thập dữ liệu, các giá trị của các biến độc lập phụ thuộc lẫn nhau trong mẫu, nhưng không phụ thuộc lẫn nhau trong tổng thể.

Ví dụ: người có thu nhập cao hơn khuynh hướng sẽ có nhiều của cải hơn. Điều này có thể đúng với mẫu mà không đúng với tổng thể

Trong tổng thể sẽ có các quan sát về các cá nhân có thu nhập cao nhưng không có nhiều của cải và ngược lại.

**Cách phát hiện trường hợp đa cộng tuyến**

Có 2 cách: dựa vào hệ số phóng đại phương sai VIF, hoặc dựa vào ma trận hệ số tương quan. Tuy nhiên cách dùng ma trận hệ số tương quan ít được sử dụng, chủ yếu sửa dụng cách nhận xét chỉ số VIF.

**Cách 2:** Nhận dạng **Multicollinearity** dựa vào hệ số tương quan,có hay không tương quan tuyến tính mạnh giữa các biến độc lập. Cách làm: xây dựng ma trận hệ số tương quan cặp giữa các biến độc lập và quan sát để nhận diện độ mạnh của các tương quan giữa từng cặp biến số độc lập. Cũng có thể nhìn vào kết quả hồi quy, ta thấy R2 cao( tầm trên 0.8) và thống kê t thấp. Tuy nhiên như đã nói thì ít khi sử dụng cách hai này. Vì nó dựa vào phán đoán chủ quan hơn là công thức như cách 1.

**Hệ quả của đa cộng tuyến:**

* Đa cộng tuyến hoàn hảo:
* Không ước lượng được mô hình
* Gặp các cảnh báo trong khi xây dựng model:
  + “Matrix singular”: ma trận khác thường mà máy tính không thể thực hiện được khi ước lượng các hệ số hồi qui
  + “Exact collinearity encounted”: trường hợp đa cộng tuyến hoàn hảo (chính xác)
* Đa cộng tuyến không hoàn hảo:

**Cách giải quyết đa cộng tuyến:**

Cách 1: Principal Component Analysis (PCA) => Demention Reduction (Decomposition)

Dùng kiến thức Eigenvector và Eigenvalue



Cách 2: Partial Least Square Regression (PLSQ) =>

**Feature Scaling and Normalization:**

Các điểm dữ liệu đôi khi được đo đạc với những đơn vị khác nhau, m và feet chẳng hạn. Hoặc có hai thành phần (của vector dữ liệu) chênh lệch nhau quá lớn, một thành phần có khoảng giá trị từ 0 đến 1000, thành phần kia chỉ có khoảng giá trị từ 0 đến 1 chẳng hạn. Lúc này, chúng ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

**Rescaling**

Phương pháp đơn giản nhất là đưa tất cả các thành phần về cùng một khoảng, [0,1] hoặc [-1,1]. Nếu muốn đưa một thành phần (feature) về khoảng [0,1] công thức sẽ là:

Trong đó, **x** là giá trị ban đầu, **x’** là giá trị sau khi chuẩn hóa. **min(x), max(x)** được tính trên toàn bộ dữ liệu training data ở cùng một thành phần. Việc này được thực hiện trên từng thành phần của vector dữ liệu **x**.

Ở Python, thư viện sklearn cho phép thực hiện scale khoảng [0,1] đơn giản với hàm: min\_max\_scaler

*#thêm thư viện xử lý ma trận***import** numpy **as** np  
*#thư viện sklearn để lấy hàm min\_max\_scaler***from** sklearn **import** preprocessing  
*#Tạo một ma trận tên X\_Train*X\_train = np.array([[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]])  
*#gán biến min\_max\_scaler từ hàm MinMaxScaler() cho khoảng [0,1]  
#gán biến max\_abs\_scaler từ hàm MaxAbsScaler() cho khoảng [-1,1]*min\_max\_scaler = preprocessing.MinMaxScaler()  
max\_abs\_scaler = preprocessing.MaxAbsScaler()  
*#Thực hiện chuyển đổi về khoảng [0,1] và [-1,1] bằng hàm fit\_transform*X\_train\_minmax = min\_max\_scaler.fit\_transform(X\_train)  
X\_train\_maxabs = max\_abs\_scaler.fit\_transform(X\_train)  
*#in ra màn hình ma trận sau khi scaled!*print(**"Ma trận ban đầu:\n{}\n\nMa trận sau khi scale:\nĐối với khoảng [0,1]:\n{}\n\n""Đối với khoảng [-1,1]:\n{}"** .format(X\_train,X\_train\_minmax,X\_train\_maxabs))

Kết Quả:

Ma trận ban đầu:

[[ 1. -1. 2.]

[ 2. 0. 0.]

[ 0. 1. -1.]]

Ma trận sau khi scale:

Đối với khoảng [0,1]:

[[ 0.5 0. 1. ]

[ 1. 0.5 0.33333333]

[ 0. 1. 0. ]]

Đối với khoảng [-1,1]:

[[ 0.5 -1. 1. ]

[ 1. 0. 0. ]

[ 0. 1. -0.5]]

**Standardization**