**從 pd.get\_dummies 到 sklearn.linear\_model.LinearRegression**

還記得剛接觸Pandas 套件的 get\_dummies時，當下彷彿天降甘霖，解決了在資料清洗當中，如何有效處置類別型變數(Categorical Variable)的困難，想當初最一開始敲代碼時，還一個一個用 if…elif….else 去新增虛擬變數(dummy Variable)，實在是太沒有效率了。

然而現實總是不若想像中美好，將下表(A)使用 pd.get\_dummies 時，會得到表B：

表A

|  |  |  |
| --- | --- | --- |
| weight | age | Gender |
| 40 | 10 | M |
| 72 | 15 | F |
| 51 | 13 | F |
| 80 | 20 | M |

表B

|  |  |  |  |
| --- | --- | --- | --- |
| weight | age | Gender\_F | Gender\_M |
| 40 | 10 | 1 | 0 |
| 72 | 15 | 0 | 1 |
| 51 | 13 | 0 | 1 |
| 80 | 20 | 1 | 0 |

根據經濟學的教法，在建立線性模型OLS時，若對類別型變數取Dummy Variable，且該類別型變數有k個去重覆值，Dummy Variable的個數將會是k-1個。

因此，如果假設體重只受到年齡與性別的影響，模型理論上應該是：

而實際上，若直接使用pd.get\_dummies則會變成one-hot encoding的型態：

所幸，pd.get\_dummies提供一個參數 drop\_first 可以個別將不同類別型變數get\_dummies後的第一個欄位刪除，話雖如此，根據粗略瀏覽網路上各個python機器學習、python資料分析等等的教學或文件，甚少會使用drop\_first，而是直接以one-hot encoding的型態去訓練模型。

**完全共線性**

在說明我的困惑前，先稍微解釋為何經濟學的計量模型會對於Dummy Variable的個數那麼講究。

首先，由於多數線性模型會放入常數項(亦即截距項)，如此一來常數項、Gender\_F和Gender\_M便會產生完全共線性，其中一個變數會被其他變數完全解釋，若以下表表示，可以發現將intercept減去Gender\_F會完全與Gender\_M相等，我們希望每個變數都可以增加模型對y(在此指體重)的解釋力，然而變數Gender\_F提供的訊息早已被intercept與Gender\_M所涵蓋。

|  |  |  |  |
| --- | --- | --- | --- |
| intercept | age | Gender\_F | Gender\_M |
| 1 | 10 | 1 | 0 |
| 1 | 15 | 0 | 1 |
| 1 | 13 | 0 | 1 |
| 1 | 20 | 1 | 0 |

其次，我們希望能夠估計每變數對y(在此指體重)的影響，線性模型中會假設其他條件不變下，平均而言個別變數對y變化的影響，然而從上表可以發現Gender\_F與Gender\_M是完全負相關，我們無法在假設Gender\_F不變下觀察Gender\_M對y的影響。

第三，如果有修過線性代數或經濟所的計量，我們能透過最小平方法去估計，亦即：

使能夠最小化。

上述估計式以矩陣表示則為：

是的不偏估計式，其公式解為，然而由於X有完全共線性，因此不存在，亦代表不存在，因此無法透過公式求得的不偏估計量。

**Python 的sklearn模擬**

然而，實際上在Python的Sklearn中的結果如何呢？以下將以模擬的數字試圖理解pd.get\_dummies 在sklearn.linear\_model.LinearRegression的輸出差異。

首先匯入相關套件：

1. # import package
2. **import** numpy as np
3. **import** pandas as pd
4. **from** sklearn.linear\_model **import** LinearRegression
5. **from** sklearn **import** metrics

接著生成模擬的數據，年齡取5歲至30間隨機樣本100筆，男性、女性亦隨機稱成100筆，實際體重則為年齡與性別的函數再加上隨機項。

1. # define age, gender, weight. create dataset.
2. age = np.random.randint(5,30,100)
3. gender\_set = ['M','F']
4. gender = np.random.choice(gender\_set,100,replace=True)
5. df01 = pd.DataFrame({'age':age,'gender':gender})
7. weight = 30 + age\*1.5 + (gender == 'M')\*15 + np.random.randn(100)

分別取得get\_dummies(drop=False)與get\_dummies(drop=True)的結果。

1. # get dummy variables
2. df01\_dummies = pd.get\_dummies(df01)
3. df01\_dummies\_drop = pd.get\_dummies(df01,drop\_first=True)
4. df01\_dummies.head()
5. df01\_dummies\_drop.head()

|  |  |  |
| --- | --- | --- |
| age | gender\_F | gender\_M |
| 20 | 0 | 1 |
| 15 | 1 | 0 |
| 28 | 1 | 0 |
| 22 | 1 | 0 |
| 28 | 1 | 0 |

|  |  |
| --- | --- |
| age | gender\_M |
| 20 | 1 |
| 15 | 0 |
| 28 | 0 |
| 22 | 0 |
| 28 | 0 |

分別以兩種數據取估計參數

1. lr1 = LinearRegression()
2. lr1.fit(df01\_dummies, weight)
3. **print**('variable: {}'.format(df01\_dummies.columns))
4. **print**('Intercept:{}, params: {}'.format(lr1.intercept\_, lr1.coef\_))
6. lr2 = LinearRegression()
7. lr2.fit(df01\_dummies\_drop, weight)
8. **print**('variable: {}'.format(df01\_dummies\_drop.columns))
9. **print**('Intercept:{}, params: {}'.format(lr2.intercept\_, lr2.coef\_))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 係數 | Intercept | age | gender\_F | gender\_M |
| drop=False | 37.55325505 | 1.4914 | -7.46173457 | 7.46173457 |
| drop=True | 30.09152048 | 1.4914 |  | 14.92346914 |

從上表可以發現，不論是哪種方式，age的係數幾乎是沒有差異，而在具有完全共線性的Intercept、gender\_F、gender\_M中可以發現，由於drop=True的版本缺少gender\_F這個變數，因此其Intercept便是隱含著gender\_F=1的情況，因此drop=True的Intercept係數幾乎相等於drop=False中Intercept與gender\_F的係數相加，同時在drop=True中gender\_M從0變為1亦代表gender\_F從1變為0，因此drop=True的gender\_M的係數幾乎相等於drop=False中gender\_M與gender\_F的相減值。

1. y\_pred\_dropN = lr1.predict(df01\_dummies)
2. y\_pred\_dropY = lr2.predict(df01\_dummies\_drop)
3. abs(y\_pred\_dropN - y\_pred\_dropY).max()
5. mse\_dropN = metrics.mean\_squared\_error(weight, y\_pred\_dropN)
6. mse\_dropY = metrics.mean\_squared\_error(weight, y\_pred\_dropY)
7. **print**('Diff :{}'.format(mse\_dropY - mse\_dropN))

預測結果的部分，可以發現仍然有非常微小的差異，兩者的預測結果最大的差距是1.42e-14，幾乎等於0了，同時兩者的平均誤差平方(MSE)為9.99e-16，也是幾乎為0。

**小結**

是不是要使用drop，是一個好問題！我認為答案仍然要回到當初設定的目標上。使用drop，在影響效果解釋上較為直觀，同時若搭配顯著性，亦可以觀察到個別解釋變數對於被解釋變數是否具有統計上的因果關係。不使用drop，雖然在解釋上較不容易，但整體資料的整理流程會順暢許多，在對test資料作清洗時也較不會發生train資料有此變數但test資料沒有此變數的情況，若我們只關注預測的結果，不使用drop是一個蠻不錯的方案。

同時從sklearn.linear\_model.LinearRegression的測試當中可以發現，它並非使用去估計係數，此部分後續有空會再作細部探討。