Multicollinearity Problems in Linear Regression. Clearly Explained!

A behind-the-scenes look at the infamous multicollinearity

Manoj Mangam · Follow

11 min read · Mar 22, 2023

We often hear that multicollinearity is one of the major problems in linear regression, but what exactly is devastating about it ?!

Multicollinearity_Photo by 浮萍 闪电 on Unsplash

Let's try deciphering it to its bits & pieces and do a simulation experiment in python to understand its problems better.

First things first! What exactly is multicollinearity?

Well, the textbook definition goes like this,

Multicollinearity refers to a situation in which two or more predictor variables in a multiple regression model are highly correlated with each other.

I know it is not of much help. Let's break it down.

> *Multicollinearity essentially is a situation in which a feature/predictor variable can be derived using one or more other features.*

Mathematically, this means that features Xp, Xp-1,...are linearly dependent.

$$X_p = \alpha_1 X_{p-1} + \alpha_2 X_{p-m} + \alpha_3 X_{p-k} + \ldots..$$

What is a linear dependency?

Formally, if we have a set of p vectors { V1, V2, ...,Vp } in a vector space, and there exist scalars c1, c2,...cp, such that:

$$c_1 V_1 + c_2 V_2 + c_3 V_3 + \cdots c_p V_p = 0$$

where at least one of the scalars is not zero, then the vectors are said to be linearly dependent. Geometrically, linearly dependent features lie in the same plane, and they do not provide any additional information beyond what is already contained in the other features.

> So, why in the world do we want redundant features when there is no value addition rather there is value degradation, which we will see shortly.

So, we ideally look for features that are ***linearly independent*** *i.e., the features cannot be expressed as linear combinations of the other features i.e.,*

$$c_1 V_1 + c_2 V_2 + c_3 V_3 + \cdots c_p V_p = 0$$

$$\text{iff } c_1 = c_2 = \cdots = c_p = 0$$

which means that all the features provide unique information.

Since the definition of multicollinearity is out of our way, let's look at the different types of multicollinearity.

Types of Multicollinearity

Broadly, there are **2 types** of multicollinearity **exact and near multicollinearity.**

1. **Exact multicollinearity:** This occurs when two or more features in the model are perfectly correlated, meaning that one feature can exactly be expressed as a linear combination of the other features as below,

$$X_p = \alpha_1 X_{p-1} + \alpha_2 X_{p-m} + \alpha_3 X_{p-k} + \ldots..$$

2. Near multicollinearity: This occurs when two or more features are highly correlated, but not perfectly correlated like above i.e., we can't exactly express a feature as a linear combination like above but with additional terms such as a constant, etc.

3. Structural multicollinearity: This is similar to near multicollinearity. It occurs when there is a theoretical relationship among the features i.e. when two features are measuring the same underlying concept. For example, consider the case of predicting income level based on age & work experience. Here age & work-exp. are structurally collinear in the sense that, in general, the higher the age the higher the work-ex.

4. Data/Incidental multicollinearity: This can either be exact or near multicollinearity. It occurs when the correlation among the features is due to chance or bias as in biased sampling, rather than a true underlying relationship.

We'll make sense of these types in the python experiments below.

Now, let's address the elephant in the room- the problems of multicollinearity in linear regression.

Problems of Multicollinearity in Linear Regression

There are many problems with multicollinearity as listed below,

1. **High Standard Errors** — Multicollinearity leads to high standard errors for the coefficient estimates, which can make it difficult to determine the significance of individual features in the model. ***It means that the estimates of β coefficients in the equation below will have high variance resulting in wider confidence intervals or imprecise estimates.*** Simply speaking it results in a situation where the actual value of *βs* will fall in a broader range of values. High-standard errors can also lead to incorrect inferences and predictions.

$$\hat{y} = E(y|X) = \beta_o + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots\ldots + \beta_p X_p$$

2. **Unstable estimates of *βs***- Multicollinearity can lead to unstable estimates of the coefficients, as small changes in the data or model can result in large changes in the coefficients. This instability can make it difficult to interpret the results of the regression analysis and to make accurate predictions.

3. **Incorrect Inferencing** — Multicollinearity can make it difficult to determine the unique contribution of each feature (feature importance) to the model, as the effects of the collinear variables are confounded. This can make it challenging to identify which features are important predictors of the outcome variable.

4. Incorrect sign and magnitude of regression coefficients- Multicollinearity can lead to incorrect sign and magnitude of the regression coefficients, which can make it difficult to interpret the results of the analysis and can lead to incorrect conclusions about the relationship between the features and the target variable.

5. Overfitting the model: Multicollinearity can lead to overfitting of the regression model, which occurs when the model is too complex and fits the ***noise or redundant info.*** in the data rather than the underlying pattern. This can result in a model that is

not generalizable to new data and performs poorly in predicting the outcome variable.

> Given these problems, the model becomes unreliable! These problems are experienced to varying degrees based on the type of multicollinearity., the worst case being exact multicollinearity.

Now let's mathematically look at why multicollinearity leads to the above problems.

Why does multicollinearity result in the above problems?

Why multicollinearity results in high standard errors for β coefficient estimates?

High standard errors of *β* coefficients essentially translate to high variance of these random variables, but why? Let's dive in.

Let X be an n x p matrix of features, where n is the number of observations and p is the number of features. The feature matrix for the regression model is given by:

$$X = [X_1 X_2 \dots X_p]_{nxp}$$

where Xi is an n x 1 vector of observations for the ith feature.

> *The OLS estimate of beta coefficients is given by,*

$$\hat{\beta} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Normal Equation Proof: https://www.geeksforgeeks.org/ml-normal-equation-in-linear-regression/

Where y is a vector containing the actual values of the target/independent variable. The variance-covariance matrix i.e., Var(*β_hat*) of the regression coefficients is given below, where σ^2 is the variance of the error term in the model. The diagonal

terms of the variance-covariance terms are nothing but the variances of the β coefficients.

$$Var(\hat{\beta}) = \sigma^2(X'X)^{-1}$$

Proof: https://math.stackexchange.com/questions/687310/variance-of-coefficients-in-a-simple-linear-regression

Where,

$$(X'X)^{-1} = \frac{1}{|X'X|} Adj(X'X)$$

When multicollinearity is present in the data, the matrix X'X becomes ill-conditioned, meaning that the determinant of the matrix approaches zero. This is because the columns of this matrix are linearly dependent. (proof: https://math.stackexchange.com/questions/2111833/prove-that-if-the-columns-are-linearly-dependent-then-deta-0). In other words, the matrix [X'X] becomes singular.

$$\Longrightarrow |X'X| \to 0$$
$$\Longrightarrow Var(\hat{\beta}) \to \infty$$

Python Simulation

Now, let's grasp the effects of multicollinearity better with 3 simulation experiments.

First, let's consider the clean case i.e., without multicollinearity. Let's randomly generate the values of a feature x1 and define the true population line y as below,

```
# create a simulated dataset without multicollinearity
np.random.seed(123)
n = 100
#Features
x1 = np.random.normal(size=n)
#True Population Line
```

```
y = 1 + 2*x1 + np.random.normal(size=n, scale=0.5)

data = pd.DataFrame({'x1': x1,'y': y})
```

	x1	y
0	-1.085631	-0.850234
1	0.997345	2.005747
2	0.282978	1.922089
3	-1.506295	-0.713437
4	-0.578600	-0.169513
...	...	...
95	1.031114	1.446701
96	-1.084568	-1.303783
97	-1.363472	-1.782368
98	0.379401	1.588170
99	-0.379176	0.132674

100 rows × 2 columns

Now, let's find the ordinary least squares (OLS) estimate of the population line. The summary of OLS regression is shown below,

```
# fit a linear regression model using statsmodels
X = sm.add_constant(data['x1'])
model = sm.OLS(data['y'], X).fit()

# print the summary statistics of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.956
Model:                            OLS   Adj. R-squared:                  0.955
Method:                 Least Squares   F-statistic:                     2104.
Date:                Tue, 21 Mar 2023   Prob (F-statistic):           4.83e-68
Time:                        15:30:24   Log-Likelihood:                -69.520
No. Observations:                 100   AIC:                             143.0
Df Residuals:                      98   BIC:                             148.3
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9905      0.049     20.213      0.000       0.893       1.088
x1             1.9917      0.043     45.873      0.000       1.906       2.078
==============================================================================
Omnibus:                        5.027   Durbin-Watson:                   1.860
Prob(Omnibus):                  0.081   Jarque-Bera (JB):                5.131
Skew:                          -0.308   Prob(JB):                       0.0769
Kurtosis:                       3.924   Cond. No.                         1.13
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Observe that the intercept (const.) β0 & x1 coeff. β1 are close to the actual values i.e.,1 & 2. Also, the 95% confidence intervals (CIs) are precise or tighter. We'll observe that β coeff. values, standard errors, and CIs go astray when we start introducing multicollinearity in different amounts.

We can get a sense of the magnitude of multicollinearity (exact or near multicollinearity) by looking at the values of *(X'X)^-1* matrix in the following equation.

$$Var(\hat{\beta}) = \sigma^2(X'X)^{-1}$$

The matrix for experiment 1 looks as below,

```
#check the (X'X)^-1 matrix, an indication of coeff. variances
X = np.vstack(x1)
X_t= np.transpose(X)
XtX = np.dot(X_t, X)
np.linalg.inv(XtX)
```

```
array([[0.00785138]])
```

The value did not grow out of bounds indicating that there's no multicollinearity. Also, we can observe that the condition number in the model summary is a diagnostic metric for multicollinearity. Higher the cond. No. higher is the multicollinearity. The Cond. No. for this case is 1.13.

Now, let's introduce milder multicollinearity in terms of a feature x2 which is defined as a linear function of x1, with the same population line y as below,

```
# create a simulated dataset with milder multicollinearity
np.random.seed(123)
n = 100
#Features
x1 = np.random.normal(size=n)
x2 = 0.5*x1 + np.random.normal(size=n, scale=0.1)
#True Population Line
y = 1 + 2*x1 + np.random.normal(size=n, scale=0.5)

data = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})
```

	x1	x2	y
0	-1.085631	-0.478610	-0.819606
1	0.997345	0.300884	2.695638
2	0.282978	0.212716	2.666308
3	-1.506295	-0.493317	-1.668441
4	-0.578600	-0.291763	-0.160354
...	...	...	...
95	1.031114	0.192452	3.410123
96	-1.084568	-0.569213	-0.231236
97	-1.363472	-0.692821	-1.519096
98	0.379401	0.155574	1.839073
99	-0.379176	-0.211383	0.651527

100 rows × 3 columns

The summary of OLS regression for this case is shown below,

```
# fit a linear regression model using statsmodels
X = sm.add_constant(data[['x1', 'x2']])
model = sm.OLS(data['y'], X).fit()

# print the summary statistics of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.955
Model:                            OLS   Adj. R-squared:                  0.954
Method:                 Least Squares   F-statistic:                     1028.
Date:                Tue, 21 Mar 2023   Prob (F-statistic):           5.02e-66
Time:                        15:46:48   Log-Likelihood:                -69.269
No. Observations:                 100   AIC:                             144.5
Df Residuals:                      97   BIC:                             152.4
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9534      0.049     19.402      0.000       0.856       1.051
x1             1.9175      0.256      7.488      0.000       1.409       2.426
x2             0.1136      0.506      0.224      0.823      -0.892       1.119
==============================================================================
Omnibus:                        1.671   Durbin-Watson:                   1.841
Prob(Omnibus):                  0.434   Jarque-Bera (JB):                1.328
Skew:                           0.280   Prob(JB):                        0.515
Kurtosis:                       3.079   Cond. No.                         14.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Now observe that β0 & β1 values are less closer to the actual values than that of the previous case coefficients. Also, the 95% confidence intervals (CIs) are wider or less precise. Also, the standard error of β1 went high. One good thing here is that the p-value i.e., p > |t| is very high for β1, implying that the values of the feature x2 are statistically not significant in predicting the target variable. We'll see in the next case (exact or stronger multicollinearity) that the feature x2 is statistically significant in predicting y which is not true (see the population line).

Now let's look at the values of (X'X)^-1 matrix,

```
#check the (X'X)^-1 matrix, an indication of coeff. variances

X = np.concatenate((np.vstack(x1),np.vstack(x2)),axis=1)
```

```
X_t= np.transpose(X)
XtX = np.dot(X_t, X)
np.linalg.inv(XtX)
```

```
array([[ 0.27171788, -0.52953328],
       [-0.52953328,  1.06267942]])
```

Though the values are higher in magnitude than in the previous case, they still look pretty much under control. Btw, after multiplying the above matrix with σ^2 i.e., the error variance, the diagonal values represent the variance of the coefficients and the rest are coeff. covariances w.r.t each other. Observe that the cond. No. for this case is 14.5 which is higher than the previous case as expected.

Multicollinearity can also be assessed at a feature level based on **VIF (Variance Inflation Factor)**. The VIF for each variable can be computed using the formula,

$$\mathrm{VIF}(\hat{\beta}_j) = \frac{1}{1 - R^2_{X_j|X_{-j}}}$$

where the R^2_Xj |X-j in the denominator is the R^2 from a regression of Xj onto all of the other predictors except Xj. If R^2_Xj |X−j is close to one, then multicollinearity is present, and so the VIF will be large. *VIF is preferred to a simple correlation coefficient matrix of the features because not all collinearity problems can be detected by inspection of the correlation matrix as it is quite possible for* ***collinearity to exist between three or more variables*** *even if no pair of variables has a high correlation coefficient.*

As a rule of thumb, VIF > 5 indicates a problematic amount of multicollinearity.

The VIF in this case is as below,

```
#find the VIF of each variable to get a sense of amount of multicollenarity
#Rule of thumb VIF > 5 indicates a problematic amount of multicollinearity

vif_data = pd.DataFrame()
data.drop(columns='y', inplace=True)
vif_data["feature"] = data.columns
```

```
# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(data.values, i)
                          for i in range(len(data.columns))]

vif_data
```

	feature	VIF
0	x1	34.607659
1	x2	34.607659

VIF values clearly indicate that there's a good amount of multicollinearity.

Now, let's introduce exact multicollinearity in terms of a feature x2 which is defined as a linear function of x1 but without an intercept and with the same population line y as below,

```
# create a simulated dataset with exact multicollinearity
np.random.seed(123)
n = 100
#Features
x1 = np.random.normal(size=n)
x2 = 0.5*x1
#True Population Line
y = 1 + 2*x1 + np.random.normal(size=n, scale=0.5)

data = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})
```

	x1	x2	y
0	-1.085631	-0.542815	-0.850234
1	0.997345	0.498673	2.005747
2	0.282978	0.141489	1.922089
3	-1.506295	-0.753147	-0.713437
4	-0.578600	-0.289300	-0.169513
...	...	...	...
95	1.031114	0.515557	1.446701
96	-1.084568	-0.542284	-1.303783
97	-1.363472	-0.681736	-1.782368
98	0.379401	0.189700	1.588170
99	-0.379176	-0.189588	0.132674

100 rows × 3 columns

The summary of OLS regression for this case is shown below,

```
# fit a linear regression model using statsmodels
X = sm.add_constant(data[['x1', 'x2']])
model = sm.OLS(data['y'], X).fit()

# print the summary statistics of the model
print(model.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.956
Model:	OLS	Adj. R-squared:	0.955
Method:	Least Squares	F-statistic:	2104.
Date:	Tue, 21 Mar 2023	Prob (F-statistic):	4.83e-68
Time:	15:54:56	Log-Likelihood:	-69.520
No. Observations:	100	AIC:	143.0
Df Residuals:	98	BIC:	148.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.9905	0.049	20.213	0.000	0.893	1.088
x1	1.5934	0.035	45.873	0.000	1.524	1.662
x2	0.7967	0.017	45.873	0.000	0.762	0.831

Omnibus:	5.027	Durbin-Watson:	1.860
Prob(Omnibus):	0.081	Jarque-Bera (JB):	5.131
Skew:	-0.308	Prob(JB):	0.0769
Kurtosis:	3.924	Cond. No.	1.35e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 8.79e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Observe that the β1 value is now way less-closer to the actual value. Also, the 95% confidence intervals (CIs) are wider, ***infact the CI for β1 doesn't even include the actual value in the interval.*** Also, see the magnitude of feature importance got affected as the coefficients' magnitudes got affected.

Also, the p-value for β2 says that it is statistically significant in predicting y, which is incorrect and leads to wrong inferencing. **This is exactly why the model gets unreliable in the presence of multicollinearity.**

Now let's look at the values of (X'X)^-1 matrix,

```
array([[ 7.03687442e+13, -1.40737488e+14],
       [-1.40737488e+14,  2.81474977e+14]])
```

Observe the values of the matrix and the cond. No. for this case i.e., 1.35e+16 have blown out of proportion as expected.

The VIF in this case is as below,

	feature	VIF
0	x1	inf
1	x2	inf

Oops! The VIF values here are ∞ which indicate that R2 =1 or there is a perfect linear correlation between the features, implying exact multicollinearity.

> ***Note:*** *In practice however, the βs are not solved for using the normal equation, but using gradient descent algorithm. Multicollinearity is a problem even in this case as it leads to solution convergence issues because the cost function tends to have more than one local minima or is flatter. So, gradient descent may get stuck in one of these local minima or oscillate back and forth between different solutions, making it difficult to find the global minimum.*

Conclusion:

Multicollinearity can present in varying degrees in the data, resulting in exact to milder multicollinearity. It can be diagnosed using metrics such as VIF (Variance Inflation Factor) & condition number. The problems of multicollinearity such as high standard errors, model unreliability, imprecise confidence intervals, incorrect inferencing, etc., can be forgivable to devastating based on the extent of multicollinearity. It can be appropriately treated using various techniques such as the removal of features with high VIFs, feature selection through PCA, regularization, etc.

I hope you've enjoyed reading the article. Please leave your feedback.

> *Feel free to play with the full code- Multicollinearity Problems in Linear Regression.ipynb*

Until next time!

Photo by Max van den Oetelaar on Unsplash

References:

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: with Applications in R. New York: Springer.

2. Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). Introduction to Linear Regression Analysis. Hoboken, NJ: John Wiley & Sons.

3. Strang, G. (2009). Introduction to Linear Algebra (4th ed.). Wellesley, MA: Wellesley-Cambridge Press.

4. https://www.geeksforgeeks.org/ml-normal-equation-in-linear-regression/

5. https://stats.stackexchange.com/questions/68151/how-to-derive-variance-covariance-matrix-of-coefficients-in-linear-regression

Data Science | Linear Regression | Machine Learning | Feature Engineering