

Principal Component Regression (PCR)

Contents

1. Multiple Linear Regression-Quick Recap
2. The Problem of Multicollinearity
3. Principal Component Analysis – General Approach
4. Principal Component Regression (PCR)
 - i. Introduction
 - ii. Statistical Model
5. PCR in Python

Multiple Linear Regression: Statistical Model

$$Y = b_0 + b_1X_1 + b_2X_2 + \dots + b_pX_p + e$$

Where,

Y : Dependent Variable

X_1, X_2, \dots, X_p : Independent Variables

b_0, b_1, \dots, b_p : Parameters of Model

e : Random Error Component

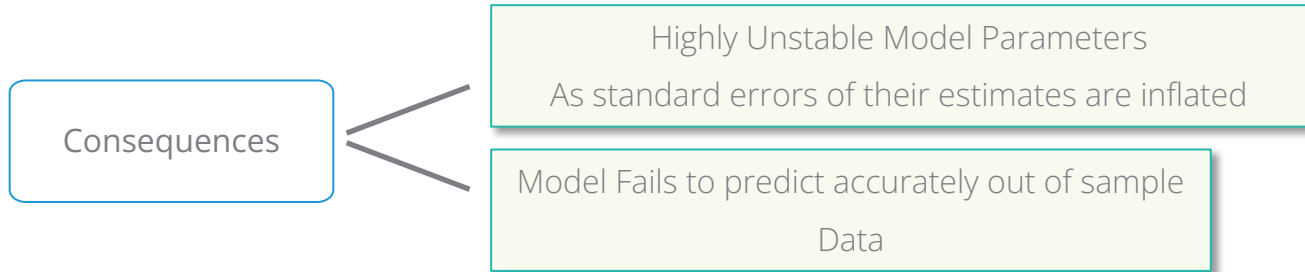
Independent variables can either be **Continuous** or **Categorical**

Problem of Multicollinearity

Multicollinearity exists



if there is a **strong linear relationship** among independent variables.



Multicollinearity is detected using Variance Inflation Factor, VIF

$$\text{Tolerance} = 1 - R_i^2$$

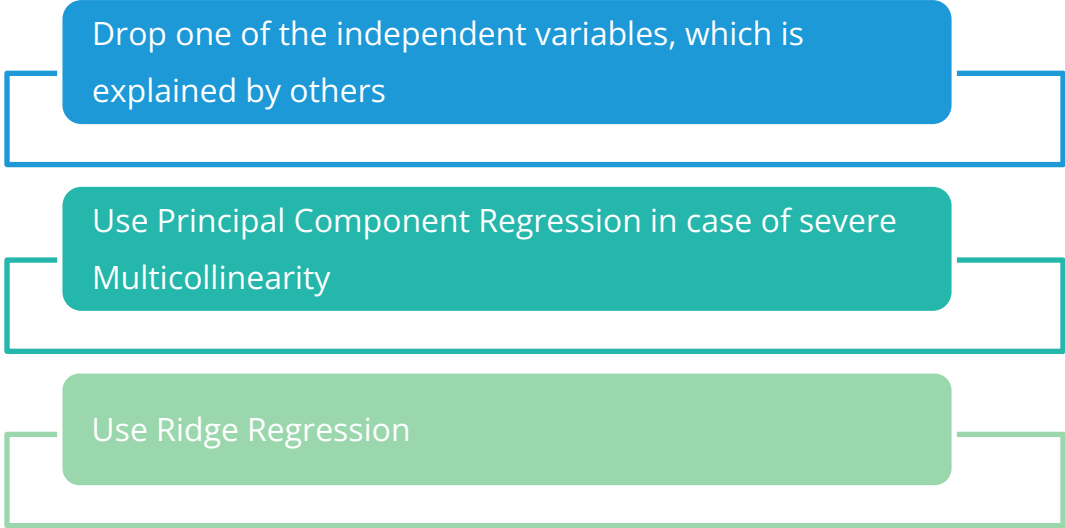
$$\text{VIF} = 1/\text{Tolerance}$$

where R_i^2 (R Squared) is obtained using regression of X_i on other independent variables

Any $\text{VIF} > 5$, indicates presence of multicollinearity

Multicollinearity – Remedial Measures

The problem of Multicollinearity can be solved by different approaches:



Drop one of the independent variables, which is explained by others

Use Principal Component Regression in case of severe Multicollinearity

Use Ridge Regression

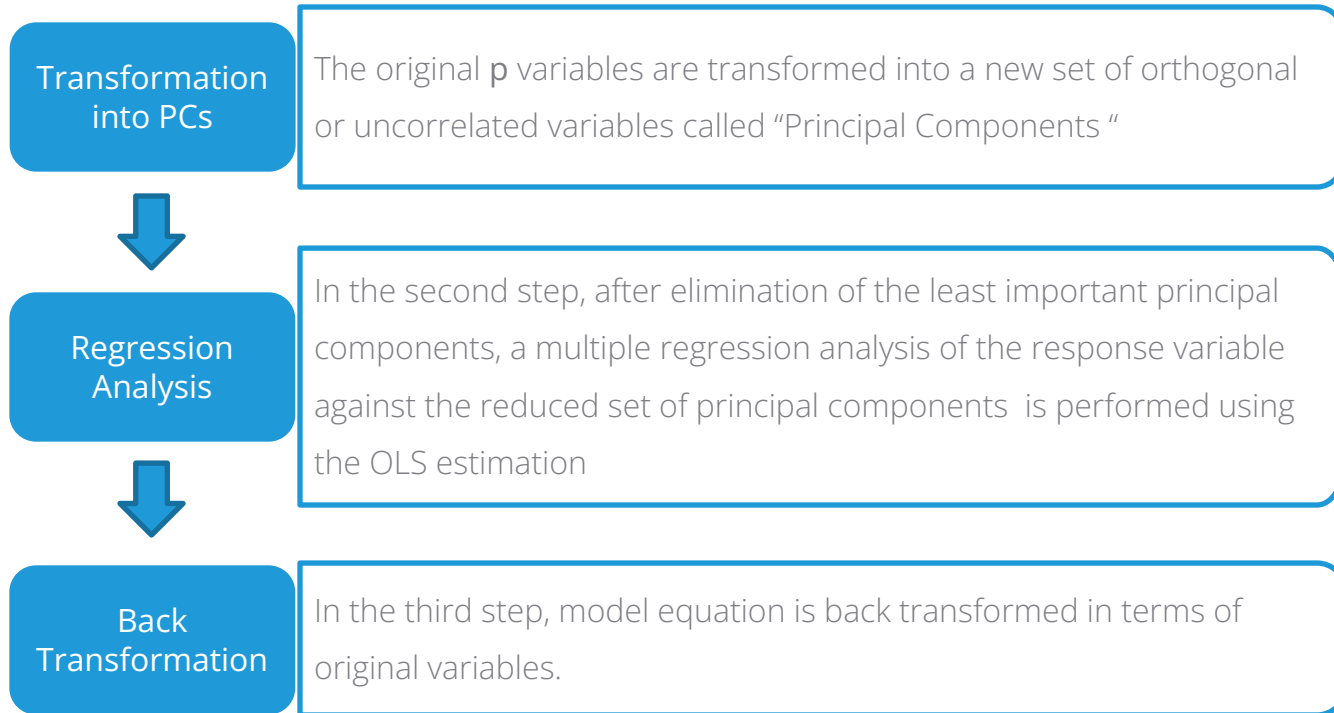
Principal Component Regression

In Principal Component Regression,

First k principal components are used as independent variables instead of original X variables

- Each PC is a linear combination of all X variables
- Final model is expressed in terms of original independent variables for ease of interpretation

Principal Component Regression



PCR-Statistical Model

Model in terms of original X variables:

$$Y = b_0 + b_1x_1 + b_2x_2 + \dots + b_px_p + e$$

Model in terms of Principal Components:

$$Y = a_0 + a_1PC_1 + a_2PC_2 + \dots + a_kPC_k + e'$$

Case Study

Background

- A company periodically records data for sales and expenses. The company wishes to model the relationship between its sales and sales related expenses and obtain predictions

Objective

- To predict incremental sales based on planned sales related expenses

Available Information

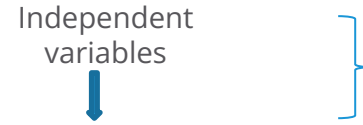
- Data available for 143 micro business zones
- Sales is the Dependent Variable
- Expenditure towards advertisements and promotions in the current and previous months are Predictors

Data Snapshot

Dependent
variable



Independent
variables



SRNO	SALES	AD	PRO	SALEXP	ADPRE	PROPRES
1	20.11	1.98	0.9	0.31	2.02	0

Columns	Description	Type	Measurement	Possible values
SRNO	Serial Number	-	-	Integers
SALES	Incremental Sales	Numerical	INR Million	positive value
AD	Current Advertising Expenses	Numerical	INR Million	positive value
PRO	Current Promotional Expenses	Numerical	INR Million	positive value
SALEXP	Misc. Sales Expenses	Numerical	INR Million	positive value
ADPRE	Previous Period's Advertising Expenses	Numerical	INR Million	positive values
PROPRES	Previous Period's Promotional Expenses	Numerical	INR Million	Positive value

PCR in Python

```
# Importing csv file "pcrdata"
```

```
import pandas as pd
salesdata = pd.read_csv('pcrdata.csv')
```

```
# Fit a Linear Model :
```

```
import statsmodels.formula.api as smf
predsales=smf.ols('SALES~AD+PRO+SALEXP+ADPRE+PROPRE',
data=salesdata).fit()
predsales.summary()
```

- ❑ **smf.ols()** fits a linear regression model.
- ❑ **summary()** generates model summary.

```
# Output:
```

OLS Regression Results						
Dep. Variable:	SALES		R-squared:	0.909		
Model:	OLS		Adj. R-squared:	0.906		
Method:	Least Squares		F-statistic:	273.2		
Date:	Fri, 10 Jan 2020		Prob (F-statistic):	2.09e-69		
Time:	11:13:37		Log-Likelihood:	-226.08		
No. Observations:	143		AIC:	464.2		
Df Residuals:	137		BIC:	481.9		
Df Model:	5					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	-10.8147	6.531	-1.656	0.100	-23.730	2.101
AD	4.6762	1.410	3.316	0.001	1.888	7.464
PRO	7.7886	1.263	6.168	0.000	5.292	10.286
SALEXP	22.4089	0.770	29.089	0.000	20.886	23.932
ADPRE	3.1856	1.244	2.560	0.012	0.725	5.646
PROPRE	3.4970	1.370	2.553	0.012	0.789	6.205
Omnibus:	8.788		Durbin-Watson:	2.153		
Prob(Omnibus):	0.012		Jarque-Bera (JB):	4.669		
Skew:	0.233		Prob(JB):	0.0969		
Kurtosis:	2.247		Cond. No.	206.		

Interpretation

- :
 - ❑ Multiple R-Squared is 0.909, showing model to be a good fit.

PCR in Python

Checking for Multicollinearity

```
from patsy import dmatrices
from statsmodels.stats.outliers_influence import
variance_inflation_factor
y, X = dmatrices('SALES~AD+PRO+SALEXP+ADPRE+PROPRE', data=salesdata,
return_type="dataframe")
vif = pd.Series([variance_inflation_factor(X.values, i)for i in
range(X.shape[1])],index=X.columns)
vif
```

Output of VIF

Intercept	4226.760949
AD	36.159771
PRO	31.846727
SALEXP	1.076284
ADPRE	24.781948
PROPRE	42.346468
dtype: float64	

- ❑ **patsy** is a library that helps convert data frames into design matrices.
- ❑ **dmatrices** Construct two design matrices given a formula_like and data. By convention, the first matrix is the "y" data, and the second is the "x" data.
- ❑ **variance_inflation_factor** requires a design matrix as input to calculate vif.
- ❑ **variance_inflation_factor()** calculates VIFs.

Interpretation:

VIF values are very high (>5, except for SALEXP) indicating severe multicollinearity problem.

PCR in Python

```
# PCA in Python  
# Subsetting data for PCA and using PCA function
```

```
from sklearn.decomposition import PCA  
from sklearn.preprocessing import scale  
  
salesdata2=salesdata.drop(['SRNO','SALES'], axis=1)  
standardisedX = scale(salesdata2)
```

- ❑ **drop()** is used to remove the columns from the data.
- ❑ The data should be standardised using the function **scale()**

```
X = pd.DataFrame(standardisedX, index=salesdata2.index,  
columns=salesdata2.columns)  
  
pca = PCA().fit(X)
```

PCR in Python

Summary of PCA

```
import numpy as np
names = ["PC"+str(i) for i in range(1,6)]

SD = list(np.std(pca.transform(X), axis=0))
VarProp = list(pca.explained_variance_ratio_)
CumProp = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1,6)]
```

□ Define names of columns as PC1,PC2 and so on.

- Extract Standard Deviation and Proportion of variance explained. Define Cumulative proportion for the summary table
- `pca.transform(X)` computes scores

```
summary = pd.DataFrame(list(zip(SD, VarProp, CumProp)), index=names,
columns=['Standard Deviation', 'Proportion of Variance', 'Cumulative Proportion'])
summary
```

□ Creates a dataframe of summary output

PCR in Python

Output

	Standard Deviation	Proportion of Variance	Cumulative Proportion
PC1	1.301556	0.338810	0.338810
PC2	1.131848	0.256216	0.595026
PC3	1.070535	0.229209	0.824235
PC4	0.933433	0.174259	0.998494
PC5	0.086770	0.001506	1.000000

Interpretation:

The first three principal components explain 82% of the variation in the data. Therefore, we will use 3 components in PCR

PCR in Python

Import library hoggorm for PCR in Python

```
import hoggorm
```

```
pcmodel = hoggorm.pcr.nipalsPCR(standardisedX, y)
```

- ❑ **pcr.nipalsPCR()** in library **hoggorm** performs Principal Component
- ❑ **standardisedX** is the X in the PCR model and **y** is object of dependent variable
- ❑ **ncomp=3** is the number of components to be included in the model,

```
salesdata['pcrpred'] = pcmodel.Y_predCal()[3]
```

```
salesdata.head()
```

.Y_predCal() is used to get predictions using PCR.

PCR in Python

Output

	SRNO	SALES	AD	PRO	SALEXP	ADPRE	PROPRE	pcrpred
0	1	20.11	1.98	0.9	0.31	2.02	0.0	21.290490
1	2	15.10	1.94	0.0	0.30	1.99	1.0	18.169736
2	3	18.68	2.20	0.8	0.35	1.93	0.0	21.271483
3	4	16.05	2.00	0.0	0.35	2.20	0.8	17.621114
4	5	21.30	1.69	1.3	0.30	2.00	0.0	22.979224

pcrpred column gives the predicted values of SALES using PCR.

Comparing Linear Regression Model and PCR model on Test data

Importing Test Data

```
salesdata_test = pd.read_csv('pcrdata_test.csv')
```

Getting RMSE of PCR model

```
salesdata_test2=salesdata_test.drop(['SRNO','SALES'], axis=1)
standardisedX2 = scale(salesdata_test2)
salesdata_test['pcrpredict'] = pcmodel.Y_predict(standardisedX2,
numComp=3)
salesdata_test['pcrres'] = salesdata_test['SALES'] -
salesdata_test['pcrpredict']
```

```
import math
```

```
import statistics
```

```
RMSE_pcr = math.sqrt(statistics.mean(salesdata_test['pcrres'])**2)
```

□ RMSE_pcr stores RMSE value using PCR Model.

Getting RMSE of linear regression model

```
salesdata_test['lmpred'] = predsales.predict(salesdata_test)
salesdata_test['lmres'] = salesdata_test['SALES'] -
salesdata_test['lmpred']
```

□ RMSE_lm stores RMSE using normal regression

```
RMSE_lm= math.sqrt(statistics.mean(salesdata_test['lmres'])**2)
```

Comparing Linear Regression Model and PCR model on Test data

Viewing data after adding predicted & residual variables

```
salesdata_test.head()
```

Output

	SRNO	SALES	AD	PRO	...	pcrpredict	pcrres	lmpred	lmres
0	1	28.93	2.75	1.00	...	22.564787	6.365213	32.313678	-3.383678
1	2	25.96	1.73	1.06	...	21.752245	4.207755	34.369246	-8.409246
2	3	31.25	2.19	1.26	...	26.662864	4.587136	32.298212	-1.048212
3	4	25.05	1.82	1.45	...	24.720436	0.329564	35.217508	-10.167508
4	5	27.32	2.38	1.01	...	25.930237	1.389763	28.226159	-0.906159

```
RMSE_lm
```

```
[1] 7.949631207886278
```

```
RMSE_pcr
```

```
[1] 0.773461538461537
```

Interpretation:

- **RMSE using PCR is less than RMSE using linear regression**, we may conclude that PCR model predicts SALES better than linear regression model when multicollinearity exists.

Quick Recap

Multiple Linear Regression and Multicollinearity

- Fundamental assumption for building a good multiple linear regression model is to have non-correlated predictor variables.
- However, highly correlated predictor variables is a very frequent phenomenon in real world analytics.

Principal Component Regression

- Such severe multicollinearity can be effectively handled by combining the regression with Principal Component Analysis.
- PCR is a three way process where the variables are first transformed to principal components, regression is run by considering these components as regressors and finally, they are transformed back to their original forms.

PCR in Python

- **`pcr.nipalsPCR()`** function in library **hoggorm** performs PCR.