Learning Principal Components Regression with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

When constructing statistical models to define the complex relationship between a collection of predictor variables and a specific response variable, the traditional approach often defaults to multiple linear regression (MLR). This foundational technique, central to quantitative analysis, relies fundamentally on the method of least squares. The core objective of this process is to meticulously determine the set of optimal regression coefficients that minimizes the overall discrepancy between the values observed in the dataset and the values predicted by the model. This discrepancy is rigorously quantified by the metric known as the Residual Sum of Squares (RSS).

The calculation of RSS provides a clear summary of the magnitude of prediction error, representing the sum of the squared differences between every actual response and its corresponding prediction. Minimizing this value ensures the model line is the best possible fit for the data points:

RSS = Σ(yi – ŷi)2

Understanding the components of this formula is essential for grasping the mechanics of coefficient estimation in linear modeling:

Σ: This is the standard mathematical symbol denoting the operation of summation across all data points in the sample.
yi: Represents the actual, observed response value recorded for the ith data point within the dataset.
ŷi: Denotes the predicted response value generated specifically by the fitted multiple linear regression model for the ith data point.

Despite the widespread utility of standard MLR, a significant and frequently encountered statistical hurdle arises when the predictor variables exhibit high correlation among themselves. This structural issue, termed multicollinearity, poses a severe threat to the stability and reliability of the resulting model. When multicollinearity is present, the estimation process for the regression coefficients becomes highly volatile, leading to inflated variance in these estimates. Consequently, interpreting the contribution of individual predictors and making reliable inferences about their true effect on the response variable becomes exceptionally difficult, necessitating robust alternative modeling strategies.

Introduction to Principal Components Regression

To systematically address and successfully overcome the detrimental effects of high multicollinearity, data scientists frequently leverage sophisticated dimensionality reduction techniques, most notably Principal Components Regression (PCR). PCR stands out as an elegant hybrid modeling approach, seamlessly integrating the powerful dimension reduction capabilities

inherent in Principal Component Analysis (PCA) with the established framework of traditional linear modeling. This combination transforms the input space into a more stable and interpretable domain.

The methodological foundation of PCR is meticulous. It begins by performing PCA on the set of original **p** predictor variables. This analysis identifies a series of **M** linear combinations, referred to as "principal components." These newly constructed components possess the critical property of being mathematically orthogonal--meaning they are completely uncorrelated with one another. Crucially, they are generated and ordered sequentially: the first component captures the maximum possible variance from the original data, the second captures the maximum remaining variance, and so on. This hierarchical structure ensures that the most informative dimensions are prioritized.

After the PCA stage, the data scientist selects an optimal subset of these uncorrelated principal components ($M < p$). Once this optimal dimensionality is determined, a standard least squares regression model is then fitted. However, instead of using the original, correlated predictors, the model utilizes these selected, uncorrelated principal components as the new input features. This strategic substitution effectively sidesteps the multicollinearity problem entirely, resulting in coefficient estimates that are far more stable and reliable for prediction tasks. This comprehensive tutorial will now provide a rigorous, step-by-step walkthrough detailing how to implement and evaluate a high-performing PCR model using Python's extensive data science ecosystem.

Step 1: Setting Up the Python Environment and Importing Packages

The successful execution of Principal Components Regression in a Python environment mandates the initial prerequisite step of properly configuring the workspace and importing the necessary specialized statistical and machine learning packages. For any serious data science endeavor involving numerical operations, complex data structures, advanced model decomposition, and robust performance evaluation, we rely heavily on foundational libraries. Specifically, the implementation of PCR requires core modules drawn from **NumPy** for high-performance array operations, **Pandas** for efficient data manipulation and structuring, and the versatile **Scikit-learn** library, which provides the majority of the machine learning algorithms.

The following Python code block encapsulates the essential setup, importing all required tools necessary for the subsequent steps of the analysis. This includes specific functions critical for data preprocessing, such as scaling the features to a common range, defining repeatable cross-validation methodologies, executing the core Principal Component Analysis via decomposition modules, and ultimately performing the final linear regression tasks. Careful organization of these imports ensures the environment is ready for robust statistical modeling.

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Step 2: Loading and Inspecting the Data

To provide a clear and practical demonstration of the PCR methodology, we will utilize the widely recognized **mtcars** dataset. This dataset is a classic benchmark in statistical analysis, compiling detailed specifications for 33 distinct automobile models. Our specific analytical goal is to construct a model capable of accurately predicting a vehicle's **horsepower (hp)**, which will serve as our designated response variable (Y). We will base this prediction on a carefully chosen subset of five intrinsic vehicle performance and structural metrics, which constitute our predictor variables (X).

The selected predictor variables are known to often exhibit intercorrelations, making them ideal candidates for a PCR approach to mitigate potential multicollinearity. The input features include:

mpg (Miles per Gallon): A measure of fuel efficiency.
disp (Displacement): The total volume swept by the pistons of the engine.
drat (Rear Axle Ratio): A factor influencing vehicle acceleration and speed.
wt (Weight): The vehicle's curb weight, often measured in thousands of pounds.
qsec (1/4 Mile Time): A performance metric indicating acceleration capability.

The Python commands presented below handle the crucial initial steps of data acquisition. They retrieve the structured dataset from a remote online repository, load the information into a robust **Pandas DataFrame**, and then meticulously isolate the six variables required for our analysis (five predictors plus the response). Finally, the head of the DataFrame is printed, allowing for immediate visual confirmation of the successful data load and verification of the expected data structure and variable types.

```
#define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

#read in data
data_full = pd.read_csv(url)

#select subset of data
```

data = data_full]

#view first six rows of data
data

mpg disp drat wt qsec hp
0 21.0 160.0 3.90 2.620 16.46 110
1 21.0 160.0 3.90 2.875 17.02 110
2 22.8 108.0 3.85 2.320 18.61 93
3 21.4 258.0 3.08 3.215 19.44 110
4 18.7 360.0 3.15 3.440 17.02 175
5 18.1 225.0 2.76 3.460 20.22 105

Step 3: Fitting the PCR Model and Determining Optimal Components

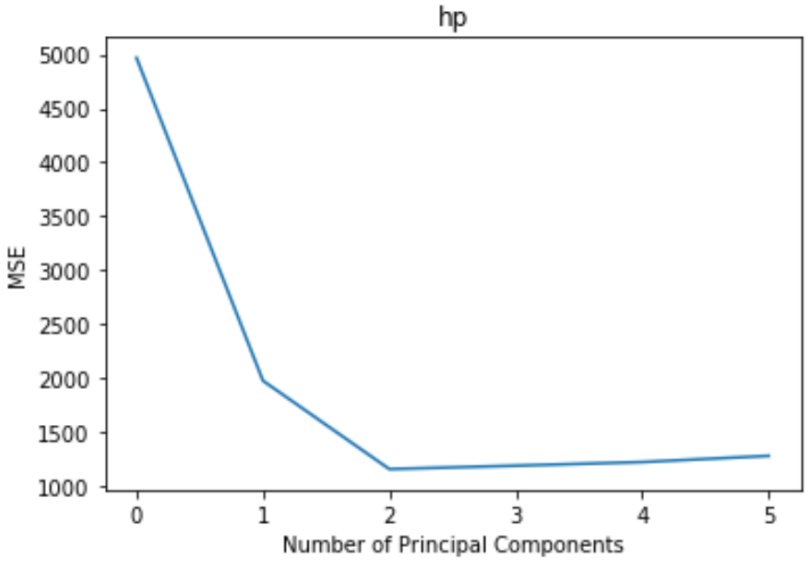
The core of the Principal Components Regression methodology resides in the fitting process, which seamlessly integrates dimension reduction with predictive modeling. This stage is critically divided into two sequential tasks: executing PCA on the predictor matrix, and then employing rigorous cross-validation to precisely identify the optimal number of components (M) necessary to achieve the best predictive accuracy while minimizing complexity.

Prior to applying the PCA decomposition, it is absolutely essential to standardize the data. The command **pca.fit_transform(scale(X))** ensures that all predictor variables are scaled such that they possess a mean of zero and a standard deviation of one. This normalization step is vital because PCA is sensitive to the scale of the features; without it, variables measured in larger units would disproportionately influence the direction and magnitude of the resulting principal components, potentially leading to misleading results.

To reliably estimate the model's performance on unseen data, we implement the robust technique of k-fold cross-validation, specifically defining it using **cv = RepeatedKFold(n_splits=10, n_repeats=3)**. This involves partitioning the data into 10 distinct folds and repeating the entire validation process 3 times, yielding a highly stable estimate of the generalization error. Our objective is to calculate the prediction error, quantified by the Mean Squared Error (MSE), for models built iteratively--starting from a simple intercept-only model (zero components) up to the maximum possible model incorporating all five principal components. The model that minimizes the MSE provides the ideal balance between variance reduction and predictive power.

#define predictor and response variables
X = data]
y = data]

```
#scale predictor variables and perform PCA
pca = PCA()
X_reduced = pca.fit_transform(scale(X))

#define cross validation method (10 folds, repeated 3 times)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

regr = LinearRegression()
mse =

# Calculate MSE with only the intercept (baseline model)
score = -1*model_selection.cross_val_score(regr,
np.ones((len(X_reduced),1)), y, cv=cv,
scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using cross-validation, adding one component at a time
for i in np.arange(1, 6):
score = -1*model_selection.cross_val_score(regr,
X_reduced, y, cv=cv, scoring='neg_mean_squared_error').mean()
mse.append(score)

# Plot cross-validation results
plt.plot(mse)
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.title('hp')
```



The resulting scree plot, a critical visualization tool, clearly displays the relationship between model complexity (number of principal components on the x-axis) and the estimated test **MSE** (on the y-axis). A careful examination of this plot demonstrates that the test MSE experiences a sharp and significant reduction as we transition from zero components to the first two principal components. This initial drop confirms that these components capture the vast majority of the predictive signal within the dataset.

However, a crucial observation is made after the second component is included. The MSE curve begins to flatten, and marginally increases when the third, fourth, and fifth components are added. This behavior is symptomatic of overfitting: the later components, while mathematically necessary to explain the remaining variance, primarily capture noise rather than meaningful predictive structure, consequently degrading the model's ability to generalize to new data. Based on the principle of parsimony and the minimization of generalization error, the optimal structure for our Principal Components Regression model decisively utilizes only the first two principal components.

Analyzing Explained Variance

To provide a quantitative justification for selecting two components, beyond the visual evidence provided by the MSE plot, it is highly informative to analyze the cumulative percentage of variance explained by the principal components derived from the PCA step. This metric reveals how efficiently the dimensions are capturing the information contained within the original predictor set.

The following code calculates the cumulative explained variance ratio, scaled to a percentage, providing an immediate understanding of the information retention:

np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)

array()

The resulting array clearly illustrates the dramatic gains achieved by the initial components:

The first principal component alone accounts for approximately **69.83%** of the total variation present in the five original predictor variables. This confirms it is the single most important dimension.
By incorporating the second principal component, the cumulative explained variance surges substantially, reaching an impressive total of **89.35%**. This indicates that nearly 90% of the relevant information is preserved using only two features instead of five.

While mathematically, including all five components would capture virtually 100% of the variance, the marginal explanatory gain achieved after the second component is minimal (e.g., component three only adds 6.53%). This quantitative evidence strongly reinforces our initial model selection decision, derived from the cross-validation MSE results: utilizing just two principal components offers an exceptionally effective and efficient dimension reduction, thereby maximizing predictive utility while maintaining model simplicity and robustness against noise.

Step 4: Utilizing the Optimal Model for Prediction

With the optimal model complexity established--a model relying on precisely two principal components--the final phase involves building the definitive predictive model and rigorously assessing its performance on completely unseen data. This step is critical for simulating real-world application accuracy and confirming the model's generalization capabilities. The methodology requires partitioning the entire dataset into distinct training and testing subsets.

We implement a standard partition, allocating 70% of the data for training the model (X_train, y_train) and reserving the remaining 30% for evaluating its performance (X_test, y_test). It is absolutely paramount that the PCA transformation--meaning the calculation of the principal components' loadings--is fitted exclusively using the training data. The calculated loadings are then applied consistently to both the training data (for fitting the regression) and the testing data (for generating predictions). Crucially, we enforce the selection of only the first two principal components in both steps, aligning with our findings from the cross-validation analysis.

The final linear regression model is trained on the reduced feature space (the first two principal components of the training data). The prediction accuracy is then quantified using the Root Mean Squared Error (RMSE), calculated against the true response values (y_test) for the observations in the testing subset. RMSE provides an easily interpretable measure of the average magnitude of the prediction errors, expressed directly in the units of the response variable (horsepower).

#split the dataset into training (70%) and testing (30%) sets

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

#scale the training data and apply transformation to test data
X_reduced_train = pca.fit_transform(scale(X_train))
X_reduced_test = pca.transform(scale(X_test)) # Selects the first 2 components

#train PCR model on training data using two components
regr = LinearRegression()
regr.fit(X_reduced_train, y_train)

#calculate RMSE
pred = regr.predict(X_reduced_test)
np.sqrt(mean_squared_error(y_test, pred))

40.2096
```

The final calculated test RMSE is observed to be **40.2096**. For a response variable like horsepower, this metric confirms the average deviation between the model's predictions and the actual horsepower values in the held-out testing set. This relatively precise result demonstrates that the two-component PCR model successfully captures the underlying structure of the data while effectively managing potential issues arising from inter-predictor correlation. Principal Components Regression thus proves to be a powerful and pragmatic technique for generating stable and accurate predictive models in the presence of challenging data structures.

The complete, executable Python code utilized for this step-by-step Principal Components Regression analysis, including all necessary imports and modeling procedures, is available for download and further study here.