Principal Components Regression: A Step-by-Step Guide in R

Authored by

Mohammed looti

November 6, 2025

When researchers and analysts approach the task of building predictive models, they frequently encounter datasets characterized by numerous potential predictor variables (often denoted as *p*) and a single corresponding response variable. The conventional starting point for analyzing such data structures is multiple linear regression. This robust statistical technique seeks to define a linear relationship between the response and the predictors by determining coefficients that minimize the disparity between observed and predicted values.

The core mechanism of fitting this model relies on the principle of least squares estimation. This method systematically calculates the regression line that minimizes the sum of squared residuals (RSS), effectively ensuring the best overall fit to the data points:

RSS = Σ(yi - ?i)2

Understanding the components of this foundational formula is key to grasping model performance evaluation:

Σ: This is the mathematical symbol for summation, signifying the operation of totaling all calculated terms.
yi: Represents the actual, observed value of the response variable for the ith observation within the dataset.
?i: Denotes the predicted value of the response variable, derived directly from the coefficients of the fitted multiple linear regression model.

However, traditional linear regression faces a significant hurdle when the predictor variables themselves are highly correlated--a pervasive statistical phenomenon known as multicollinearity. When multicollinearity is present, the model's coefficient estimates become highly unstable, leading to inflated standard errors and difficulties in interpreting the relative importance of individual predictors. The consequence is often a model that fits the training data well but fails drastically when applied to new, unseen observations due to excessively high variance.

To neutralize the damaging effects of high correlation among predictors, an advanced dimensional reduction technique is often required. Principal components regression (PCR) offers a powerful and elegant solution. PCR operates by first transforming the original *p* predictor variables into a much smaller, orthogonal set of linear combinations, which are termed "principal components." Since these new components are uncorrelated by design, they eliminate the multicollinearity problem entirely. Subsequently, a standard least squares linear regression model is fitted using only a select number of these uncorrelated components as the new input predictors.

This comprehensive, step-by-step guide is designed to walk you through the practical application and interpretation of principal components regression using the powerful statistical programming language, **R**.

Step 1: Preparing the R Environment and Necessary Packages

Executing principal components regression efficiently within the R environment necessitates the use of specialized libraries. The most reliable and straightforward method involves leveraging the **pls** package. This package is specifically engineered to handle regression techniques based on dimension reduction, encompassing both Partial Least Squares (PLS) and Principal Components Regression (PCR) modeling.

Before any modeling can commence, the **pls** package must be installed and loaded into the current **R** session. Installation is a one-time process, but the library must be explicitly loaded every time a new session is started. This ensures that the specialized functions, such as `pcr()`, are available for use.

```
# Install the pls package (if not already installed)
install.packages("pls")

# Load the pls package into the current session
library(pls)
```

Once these commands are executed, the environment is fully prepared to handle the complex computations involved in generating and evaluating principal components.

Step 2: Fitting the Principal Components Regression Model

To illustrate the practical implementation of PCR, we will utilize the popular **mtcars** dataset, which is conveniently built directly into **R**. This dataset comprises detailed specifications for 32 different automobile models, including metrics like mileage, engine displacement, weight, and horsepower, making it an ideal candidate for exploring multivariate relationships.

Familiarizing oneself with the data structure is a crucial first step. We can inspect the initial observations to understand the variables and their scale using the `head()` function:

```
# View first six rows of mtcars dataset
head(mtcars)
```

```
mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
```

Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

In this specific modeling exercise, our goal is to predict the performance metric of horsepower (**hp**), which serves as our response variable. We will select a subset of other relevant attributes as our predictor variables for the PCR model:

mpg (Miles per Gallon)
disp (Displacement)
drat (Rear Axle Ratio)
wt (Weight)
qsec (1/4 Mile Time)

The core of the implementation is the `pcr()` function. It is critical to configure the arguments correctly to ensure both robust data preprocessing and an accurate validation strategy.

scale=TRUE: This command is mandatory for effective PCA. It standardizes the predictor variables, transforming them so that they all possess a mean of 0 and a standard deviation of 1. This standardization prevents predictors measured on vastly different scales (e.g., displacement measured in hundreds, versus weight measured in tens) from artificially dominating the principal components calculation.
validation="CV": This powerful setting invokes k-fold cross-validation (CV). By default, `validation="CV"` performs 10-fold cross-validation, which rigorously estimates the model's predictive performance on unseen data by splitting the training set into ten segments.

 # Ensure the results are reproducible by setting a seed
set.seed(1)

# Fit the PCR model using the selected predictors and cross-validation
model <- pcr(hp~mpg+disp+drat+wt+qsec, data=mtcars, scale=TRUE, validation="CV")

Step 3: Determining the Optimal Number of Components (M)

The primary decision point in principal components regression is selecting the optimal number of principal components (M) to include in the final model. Including too few components may result in high bias (underfitting), while including too many components risks incorporating noise, leading to higher variance and reduced predictive accuracy.

The standard methodology for this crucial selection relies on evaluating the test error, specifically the root mean squared error of prediction (RMSEP), which is calculated across different component counts during the k-fold cross-validation performed in the previous step. We can readily

access these vital metrics by calling the summary function on our fitted model object:

```
# View summary of model fitting
summary(model)
```

Data: X dimension: 32 5
Y dimension: 32 1
Fit method: svdpc
Number of components considered: 5

VALIDATION: RMSEP
Cross-validated using 10 random segments.
(Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps
CV 69.66 44.56 35.64 35.83 36.23 36.67
adjCV 69.66 44.44 35.27 35.43 35.80 36.20

TRAINING: % variance explained
1 comps 2 comps 3 comps 4 comps 5 comps
X 69.83 89.35 95.88 98.96 100.00
hp 62.38 81.31 81.96 81.98 82.03

The output provides two essential tables for analysis. We first focus on the **VALIDATION: RMSEP** table, which reports the cross-validated test RMSE (CV). We are looking for the model complexity (number of components) that achieves the minimum error, indicating the best generalization capacity:

Using only the intercept term (a baseline model), the test RMSE is **69.66**.
The introduction of the first principal component dramatically reduces the test RMSE to **44.56.**
The lowest prediction error, indicating **optimal predictive performance**, is achieved when the second principal component is included, resulting in an RMSE value of **35.64.**

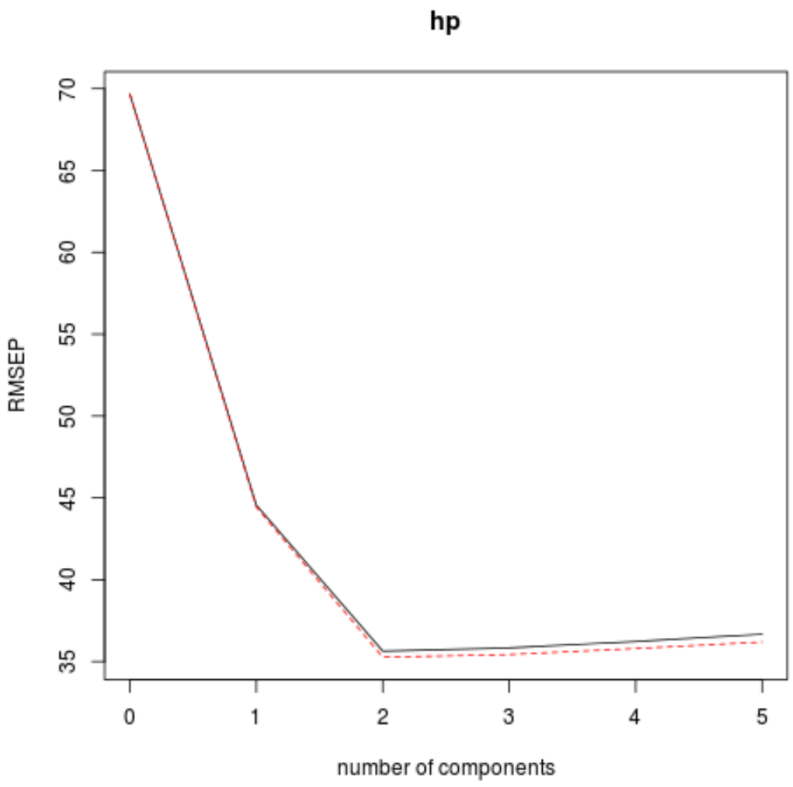
Crucially, we observe a clear pattern of diminishing returns: adding the third, fourth, and fifth principal components results in a marginal increase in test RMSE (35.83, 36.23, and 36.67, respectively). This confirms that retaining only **two principal components** maximizes predictive accuracy while minimizing model complexity.

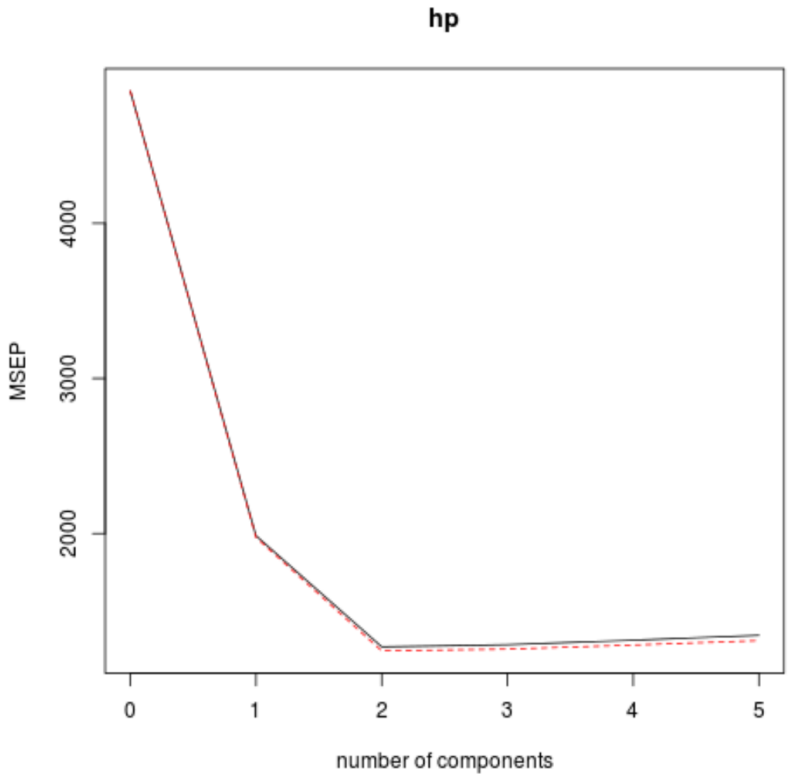
The second table, **TRAINING: % variance explained**, provides insight into how much variance is captured. The row labeled 'X' shows the cumulative variance explained across the predictor space (the components), which quickly approaches 100%. More importantly for regression, the row labeled 'hp' shows the percentage of variance in the response variable explained by the model:

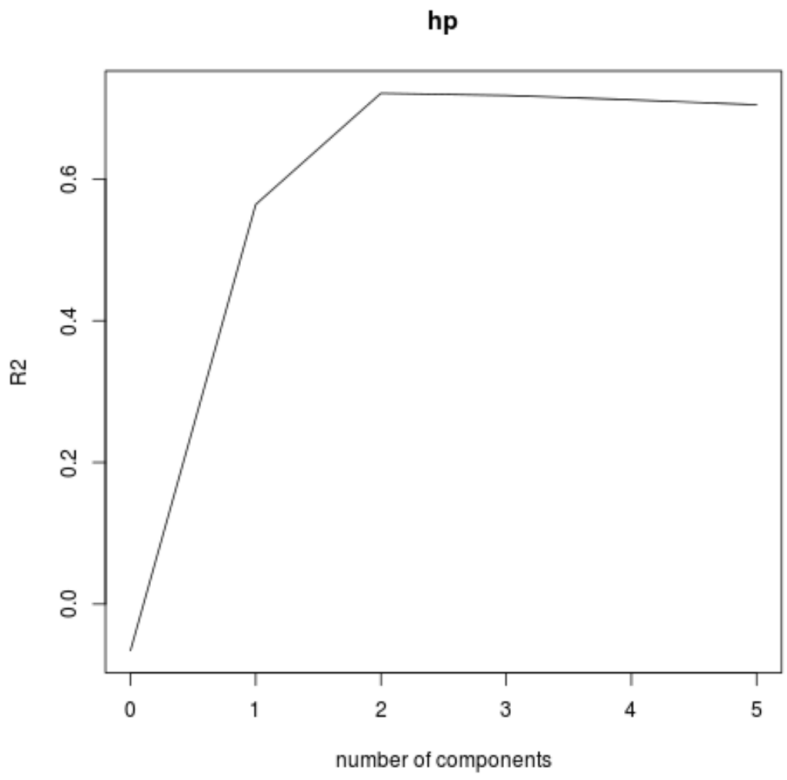
The first principal component alone accounts for a substantial **62.38%** of the variance in horsepower.
When the second principal component is incorporated, the explained variance sharply increases to **81.31%**. This confirms that the first two components capture the vast majority of the signal relevant to predicting horsepower.

To visually reinforce this analytical conclusion, the `validationplot()` function allows us to generate graphical representations of the error metrics (RMSEP, MSEP) and R-squared values across the spectrum of component selection:

 # Visualize cross-validation plots for selection
validationplot(model)
validationplot(model, val.type="MSEP")
validationplot(model, val.type="R2")







These visualizations consistently demonstrate that the model's performance--indicated by the

minimum RMSEP and maximum R-squared--peaks precisely at two principal components. Beyond this point, including additional components only introduces noise, leading to overfitting and confirming our selection of M=2.

Step 4: Evaluating the Final Model and Making Predictions

The final and most crucial validation step involves testing the streamlined PCR model (which uses the identified optimal number of M=2 components) on a holdout testing set. This process accurately quantifies the model's true capacity for generalization--its ability to predict outcomes for data points it has never encountered during the fitting process.

To perform this rigorous assessment, we must first partition the original **mtcars** dataset into two distinct, non-overlapping subsets: a training set (used for fitting the model parameters) and a testing set (reserved solely for final performance evaluation). We then refit the model exclusively on the training data.

The `predict()` function is then employed, ensuring that we explicitly restrict the prediction to utilize only the first two principal components, which were determined as optimal in Step 3. The resulting predictions are compared against the known actual values of horsepower in the test set.

 # Define training and testing sets (using first 25 for training, remainder for testing)
train <- mtcars
y_test <- mtcars
test <- mtcars

# Fit PCR model using only the training data
model <- pcr(hp~mpg+disp+drat+wt+qsec, data=train, scale=TRUE, validation="CV")

# Use the trained model to make predictions on the test set using only 2 components
pcr_pred <- predict(model, test, ncomp=2)

# Calculate the Test Root Mean Squared Error (RMSE)
sqrt(mean((pcr_pred - y_test)^2))

56.86549

The calculation yields a final test RMSE of **56.86549**. This figure represents the average magnitude of the prediction error for the horsepower values on the testing data. In practical terms, this means that the principal components regression model, leveraging the dimensionality reduction benefits of only two components, typically predicts the horsepower of a new car within approximately 56.87 units of its actual value. This demonstrates the model's effectiveness in

providing stable and generalizable predictions even in the presence of highly correlated predictor variables.

For further study and replication, the complete **R** code used throughout this comprehensive example, including all setup and modeling commands, is available for access and download here.