How to perform Partial Least Squares in R (Step-by-Step)

Authored by

stats writer

December 18, 2025

Partial Least Squares (PLS) is a powerful **variable reduction technique** employed extensively in multivariate data analysis. It is specifically designed for modeling relationships between large sets of **predictor variables** (independent variables) and **response variables** (dependent variables), particularly when the predictors are highly correlated or when the number of predictors exceeds the number of observations. Unlike traditional regression methods, PLS handles these complex scenarios by projecting the data onto a new space defined by latent variables, known as PLS components. Successfully implementing a PLS analysis in the R statistical environment requires a structured approach: installing the necessary packages, loading the data, selecting and fitting the appropriate PLS model, optimizing the number of components, and finally, interpreting the detailed results to draw meaningful conclusions about the underlying variable relationships.

This comprehensive tutorial guides you through the practical application of PLS regression using R, providing step-by-step instructions for data preparation, model fitting, validation, and prediction. We will utilize the highly regarded **pls package**, which provides robust tools for tackling data challenges common in fields ranging from chemometrics to econometrics.

Understanding the Challenge: Multicollinearity in Regression

One of the most frequent and challenging issues encountered when building predictive models in machine learning and statistics is **multicollinearity**. This phenomenon occurs when two or more **predictor variables** within a dataset are highly correlated with one another. When predictors move in tandem, it becomes difficult for a standard regression model to uniquely identify the independent effect of each variable on the response. This leads to unstable coefficient estimates, inflated standard errors, and reduced statistical power, making the model unreliable for inference.

The practical consequence of severe multicollinearity is often poor generalization. A model might appear to fit the training dataset exceptionally well, achieving low residuals during the training phase. However, due to the instability caused by highly correlated inputs, the model struggles significantly when presented with new, unseen data, leading to inaccurate predictions. This failure to generalize effectively is a classic symptom of overfitting the noise and specifics of the training set rather than capturing the fundamental relationship structure.

Addressing this statistical obstacle is essential for developing robust predictive models. While techniques like subset selection or ridge regression exist, Partial Least Squares offers an elegant solution by focusing on extracting the crucial information that explains covariance between the predictors and the response.

Introduction to Partial Least Squares (PLS) Regression

PLS regression is an iterative algorithm designed to project both the independent (X) and dependent (Y) variables onto a new, lower-dimensional space. The primary goal of PLS is not

merely to reduce the number of variables, but specifically to find a set of **latent variables** (PLS components) that maximize the covariance between the projected X data and the projected Y data. By constructing these components, PLS effectively summarizes the information in the predictors that is most relevant for predicting the response, naturally mitigating the effects of multicollinearity.

The process of PLS differs significantly from techniques like Principal Component Regression (PCR), which derives components solely based on maximizing the variance in the predictor space (X). In contrast, PLS ensures that the components are not only orthogonal in the X space but are also constructed to be highly predictive of Y, making it a powerful method when the focus is on prediction and handling high-dimensional data where predictors are inter-correlated.

The Mechanics of PLS: Component Creation

The PLS algorithm achieves dimensionality reduction while optimizing predictive power through a series of internal steps. Understanding these steps clarifies why PLS is so effective at solving the multicollinearity challenge:

Standardize both the **predictor** and **response variables**. This initial step ensures that all variables contribute equally to the model fitting process, regardless of their original scale or units.
Calculate *M* linear combinations (termed "PLS components" or latent vectors) of the original *p* predictor variables. These components are strategically selected to maximize the explained variation in both the predictor variables (X) and, crucially, the response variable (Y). This dual optimization is the hallmark of PLS.
Once the optimal components are derived, the method of **least squares** is applied to fit a standard linear regression model. However, instead of using the original, correlated predictors, this regression uses the newly calculated, uncorrelated PLS components as the predictors.
To ensure that the final model is robust and avoids overfitting, a rigorous validation technique, typically k-fold cross-validation, is used to determine the optimal number of PLS components that should be retained in the final predictive model.

This tutorial provides a detailed, practical example demonstrating how to execute and interpret these steps for Partial Least Squares regression in the R environment.

Step 1: Setting Up the R Environment and Loading Packages

The primary tool for performing PLS in R is the dedicated **pls package**, available on CRAN. This package offers efficient functions for fitting PLS models, performing cross-validation, and visualizing the results. Before starting the analysis, we must ensure this package is installed and loaded into the current R session.

If you are running the analysis for the first time, you must execute the installation command.

Otherwise, simply load the library using the standard `library()` function. It is important to remember that all statistical analysis in R relies on having the appropriate packages loaded.

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

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

With the necessary tools initialized, we are ready to proceed with the data preparation and model fitting process using a standard benchmark dataset.

Step 2: Fitting the PLS Model and Defining Validation Parameters

For this illustrative example, we will employ the well-known **mtcars** dataset, which is conveniently built into R and contains observations on 32 different automobiles, detailing specifications like mileage, cylinder count, displacement, and horsepower. Before fitting the model, it is beneficial to examine the structure of the data.

 # View the first six rows of the mtcars dataset to understand its structure
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

Our objective is to predict the car's horsepower (`hp`), which will serve as our **response variable**. We will use a selection of five other variables as our **predictor variables**:

mpg (Miles per gallon)
disp (Displacement)
drat (Rear axle ratio)
wt (Weight)
qsec (1/4 mile time)

The `plsr()` function from the pls package is used to fit the model. We must specify key arguments that control the behavior and validation of the PLS routine.

Two critical arguments in the `plsr()` function govern the robustness and interpretability of the model:

`scale=TRUE`: This instructs R to internally standardize all variables (both predictors and response) so they possess a mean of zero and a standard deviation of one. Scaling is crucial in PLS, as it prevents variables measured on a large scale from unduly dominating the component extraction process, thereby ensuring that all predictors contribute fairly based on their information content, not their magnitude.
`validation="CV"`: This argument dictates the method used to evaluate the model's performance and select the optimal number of components. Specifying `"CV"` triggers **k-fold cross-validation**, which by default uses 10 folds (k=10). This technique partitions the data into ten subsets, training the model on nine subsets and validating it on the remaining one, iterating this process ten times. Alternatively, specifying `"LOOCV"` performs the computationally intensive **leave-one-out cross-validation**. Cross-validation is essential for obtaining unbiased estimates of predictive performance.

 # Ensure reproducibility of component extraction and CV results
set.seed(1)

# Fit the Partial Least Squares model
model <- plsr(hp~mpg+disp+drat+wt+qsec, data=mtcars, scale=TRUE, validation="CV")

Step 3: Determining the Optimal Number of PLS Components

The most critical phase of PLS analysis is selecting the correct number of latent components (M). Using too few components risks underfitting the data, while using too many components can lead to overfitting or incorporating noise, particularly if the response variance is already sufficiently explained. The best way to make this determination is by analyzing the performance metric calculated during the cross-validation process, specifically the **Root Mean Squared Error of Prediction (RMSEP)**.

The `summary()` function provides a concise output detailing the model fitting process and, more importantly, the validation results, allowing us to inspect the RMSEP for models ranging from zero to the maximum possible number of components (which is the minimum of N-1 or P, where N is observations and P is predictors).

 # View summary of model fitting and cross-validation results
summary(model)

Data: X dimension: 32 5
Y dimension: 32 1

Fit method: kernelpls
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 40.57 35.48 36.22 36.74 36.67
adjCV 69.66 40.41 35.12 35.80 36.27 36.20

TRAINING: % variance explained
1 comps 2 comps 3 comps 4 comps 5 comps
X 68.66 89.27 95.82 97.94 100.00
hp 71.84 81.74 82.00 82.02 82.03

Interpreting the Validation Metrics (RMSEP and Variance)

The summary output presents two crucial tables that guide component selection: the VALIDATION: RMSEP table and the TRAINING: % variance explained table.

1. VALIDATION: RMSEP Analysis

This table displays the cross-validated Root Mean Squared Error of Prediction (RMSEP) for models incorporating an increasing number of PLS components. The RMSEP is a measure of the average magnitude of the error (deviation) between the predicted and observed values in the test (cross-validated) folds. Our goal is to select the number of components that minimizes this metric.

The baseline error, achieved using only the intercept term (zero components), is **69.66**.
Adding the first PLS component drastically reduces the error to **40.57**.
Adding the second PLS component further minimizes the error, dropping it to **35.48**.
Crucially, the subsequent addition of the third, fourth, and fifth components results in an increase in the test RMSEP (36.22, 36.74, and 36.67, respectively).

The principle of parsimony suggests selecting the model complexity right before the performance begins to degrade. Since the RMSEP reaches its minimum at two components and then increases, the optimal choice is to use **two PLS components** in the final model.

2. TRAINING: % Variance Explained Analysis

This section details how much variance in both the predictor set (X) and the response variable (`hp`) is explained by the components in the training data. This is useful for understanding the informational coverage of the latent variables.

By using just the first PLS component, we can explain **68.66%** of the total variance within the predictor set (X), and **71.84%** of the variance in the response variable (`hp`).
By adding the second PLS component, the explained variance in X jumps to **89.27%**, and the explained variance in `hp` rises to **81.74%**.

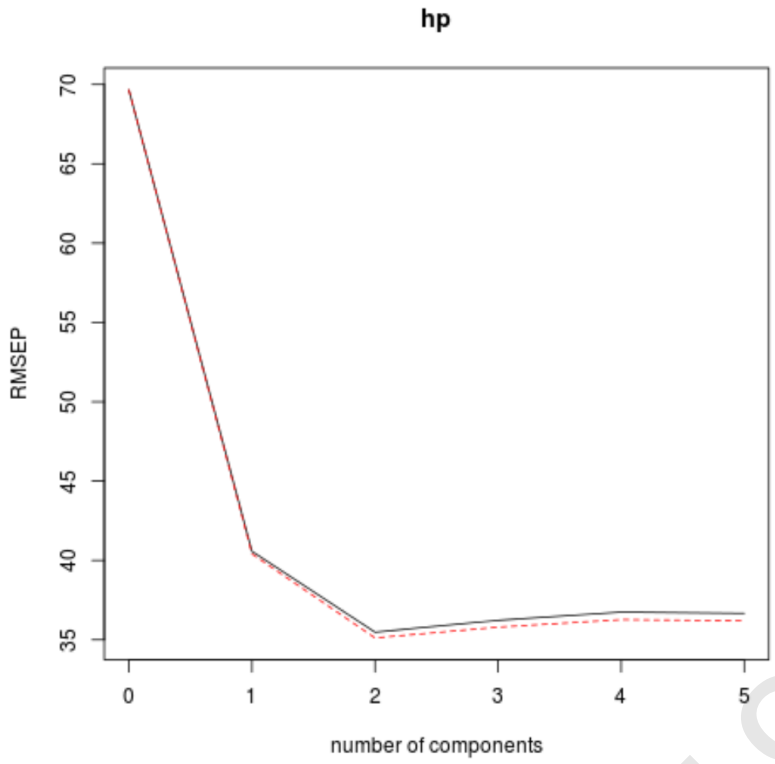
While adding more components will always monotonically increase the explained variance in X (reaching 100% with five components), the gain in explained variance for the response variable (`hp`) plateaus significantly after the second component (82.00% for three components, 82.03% for five components). This reinforces the finding from the RMSEP analysis: two components capture nearly all the predictive structure in the data without incorporating unnecessary noise.

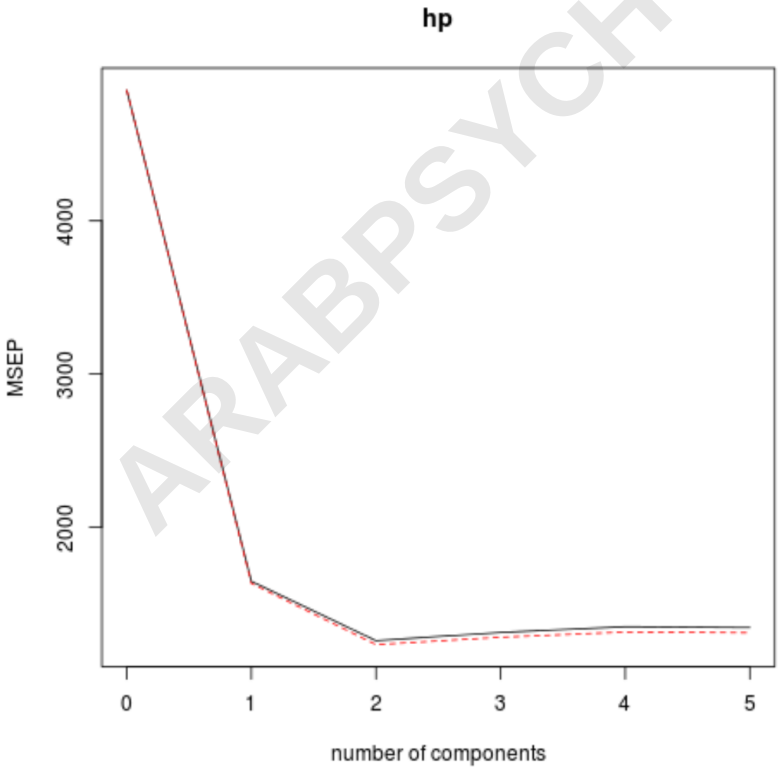
Visualizing Model Optimization

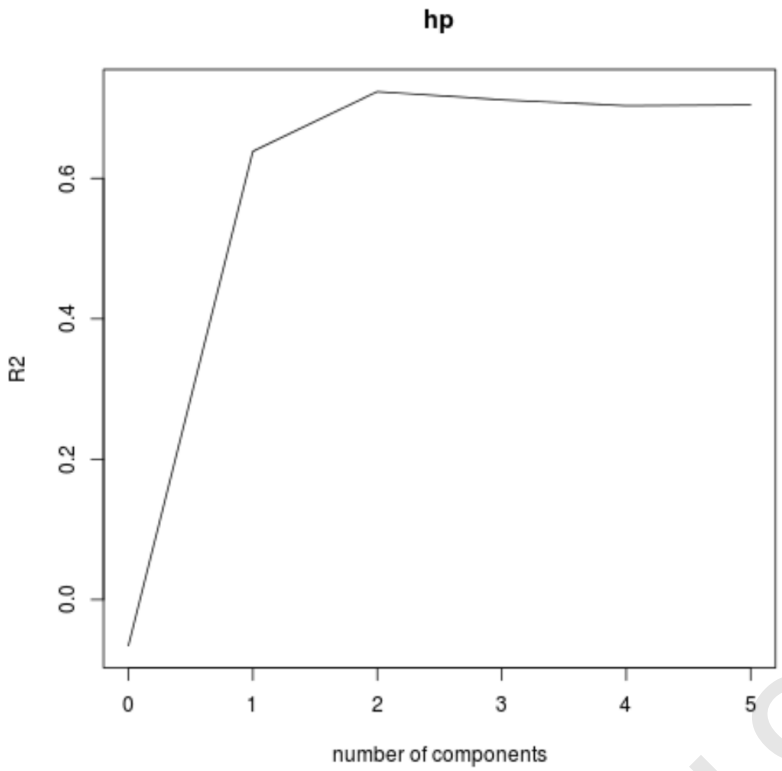
While the summary table is definitive, visualizing the cross-validation metrics is often the clearest way to confirm the optimal component count. The `validationplot()` function allows us to plot the RMSEP, Mean Squared Error of Prediction (MSEP), and R-squared values against the number of components.

```
# Visualize cross-validation plots for RMSEP
validationplot(model)
# Visualize cross-validation plots for MSEP
validationplot(model, val.type="MSEP")
# Visualize cross-validation plots for R-squared
validationplot(model, val.type="R2")
```

The resulting plots clearly illustrate the performance curve:







In all three visualizations (RMSEP, MSEP, and R-squared), the model fit shows a significant improvement when moving from one to two PLS components. After the second component, the error metrics stabilize or begin to slightly increase, indicating diminishing returns and potential degradation due to complexity. Based on this robust evidence, we confidently confirm that the optimal PLS model incorporates only the **first two PLS components**.

Step 4: Using the Optimal Model for Prediction

Once the optimal complexity (M=2) is determined, we can finalize the model and apply it to predict the response variable for new, independent observations. This step validates the generalization capability of the PLS technique. We will demonstrate this by splitting the original `mtcars` data into distinct training and testing sets.

We will train the PLS model on the first 25 observations and then use the resulting model, restricted to the two optimal components, to predict horsepower (`hp`) for the remaining observations in the testing set.

 # Define training set (first 25 rows, including all relevant variables)
train <- mtcars
 # Define the true response values for the test set
y_test <- mtcars

```
# Define the predictor variables for the test set
test <- mtcars

# Fit the PLS model using only the training data
model <- plsr(hp~mpg+disp+drat+wt+qsec, data=train, scale=TRUE, validation="CV")
# Use the trained model to make predictions on the independent test set, specifying 2 components
pcr_pred <- predict(model, test, ncomp=2)

# Calculate the final predictive RMSE (test error)
sqrt(mean((pcr_pred - y_test)^2))

54.89609
```

The calculated test RMSE for our PLS model is **54.89609**. This value represents the average deviation between the predicted horsepower and the actual observed horsepower values for the automobiles in our testing set. This metric provides a tangible measure of the model's prediction accuracy on data it has not previously encountered.

In comparative studies, it is often insightful to compare PLS performance against other dimensionality reduction methods. For instance, an equivalent Principal Components Regression (PCR) model, also using two principal components on this same split dataset, yielded a test RMSE of **56.86549**. In this specific scenario, the PLS approach demonstrated superior predictive accuracy, illustrating its advantage in constructing components that maximize covariance with the response variable, thus often leading to lower generalization error than PCR when dealing with noisy or multicollinear data.

The complete R code used throughout this step-by-step demonstration is publicly available for review and replication here.