

Can you explain the step-by-step process for performing Partial Least Squares in R?

Authored by
stats writer

April 22, 2024

RECOMMENDED CITATION

stats writer (2024). *Can you explain the step-by-step process for performing Partial Least Squares in R?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=138033>

Partial Least Squares (PLS) is a statistical method used for modeling relationships between a set of independent variables and a dependent variable. It is commonly used in data analysis and machine learning applications. In order to perform PLS in R, the following steps can be followed:

1. Import the necessary libraries: To begin with, the "pls" library needs to be imported into the R environment. This library contains the functions required for performing PLS.
2. Prepare the data: The data set should be organized into a matrix format, with the independent variables in one matrix and the dependent variable in another.
3. Perform PLS regression: The "pls" function is used to perform PLS regression. It takes the independent and dependent variables as inputs and returns a PLS model object.
4. Assess the model: The "summary" function can be used to obtain the summary of the PLS model. This includes information such as the number of components used, R-squared value, and cross-validation results.
5. Visualize the results: The "plot" function can be used to generate various plots to visualize the results of the PLS model. This includes plots of the predicted values versus the actual values, as well as plots of the regression coefficients.
6. Validate the model: It is important to validate the PLS model to ensure its accuracy and reliability. This can be done using techniques such as cross-validation or by splitting the data into training and testing sets.
7. Make predictions: Once the model has been validated, it can be used to make predictions on new data sets using the "predict" function.

Overall, performing Partial Least Squares in R involves importing the necessary libraries, preparing the data, performing the regression, assessing and visualizing the results, validating the model, and making predictions. With this step-by-step process, one can effectively use PLS for modeling relationships between variables in R.

Partial Least Squares in R (Step-by-Step)

One of the most common problems that you'll encounter in machine learning is multicollinearity. This occurs when two or more predictor variables in a

dataset are highly correlated.

When this occurs, a model may be able to fit a training dataset well but it may perform poorly on a new dataset it has never seen because it overfits the training set.

One way to get around this problem is to use a method known as partial least squares, which works as follows:

Standardize both the predictor and response variables. Calculate M linear combinations (called "PLS components") of the original p predictor variables that explain a significant amount of variation in both the response variable and the predictor variables. Use the method of least squares to fit a linear regression model using the PLS components as predictors. Use k-fold cross-validation to find the optimal number of PLS components to keep in the model.

This tutorial provides a step-by-step example of how to perform partial least squares in R.

Step 1: Load Necessary Packages

The easiest way to perform partial least squares in R is by using functions from the pls package.

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

```
load pls package
library(pls)
```

Step 2: Fit Partial Least Squares Model

For this example, we'll use the built-in R dataset called `mtcars` which contains data about various types of cars:

```
#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
```

For this example we'll fit a partial least squares (PLS) model using `hp` as the response variable and the

following variables as the predictor variables:

`mpgdispdratwtqsec`

The following code shows how to fit the PLS model to this data. Note the following arguments:

`scale=TRUE`: This tells R that each of the variables in the dataset should be scaled to have a mean of 0 and a standard deviation of 1. This ensures that no predictor variable is overly influential in the model if it happens to be measured in different units.
`validation="CV"`: This tells R to use k-fold cross-validation to evaluate the performance of the model. Note that this uses `k=10` folds by default. Also note that you can specify `"LOOCV"` instead to perform leave-one-out cross-validation.

```
#make this example reproducible  
set.seed(1)
```

```
#fit PCR model
```

```
model <- plsrfit(plsr(hp~mpg+disp+drat+wt+qsec,  
data=mtcars, scale=TRUE, validation="CV"))
```

Step 3: Choose the Number of PLS Components

The way to do so is by looking at the test root mean squared error (test RMSE) calculated by the k-fold cross-validation:

```
#view summary of model fitting
```

```
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
```

There are two tables of interest in the output:

1. VALIDATION: RMSEP

This table tells us the test RMSE calculated by the k-fold cross validation. We can see the following:

If we only use the intercept term in the model, the test RMSE is 69.66. If we add in the first PLS component, the test RMSE drops to 40.57. If we add in the second PLS component, the test RMSE drops to 35.48.

We can see that adding additional PLS components actually leads to an increase in test RMSE. Thus, it appears that it would be optimal to only use two PLS components in the final model.

2. TRAINING: % variance explained

This table tells us the percentage of the variance in the response variable explained by the PLS components. We can see the following:

By using just the first PLS component, we can explain 68.66% of the variation in the response variable. By adding in the second PLS component, we can explain

89.27% of the variation in the response variable.

Note that we'll always be able to explain more variance by using more PLS components, but we can see that adding in more than two PLS components doesn't actually increase the percentage of explained variance by much.

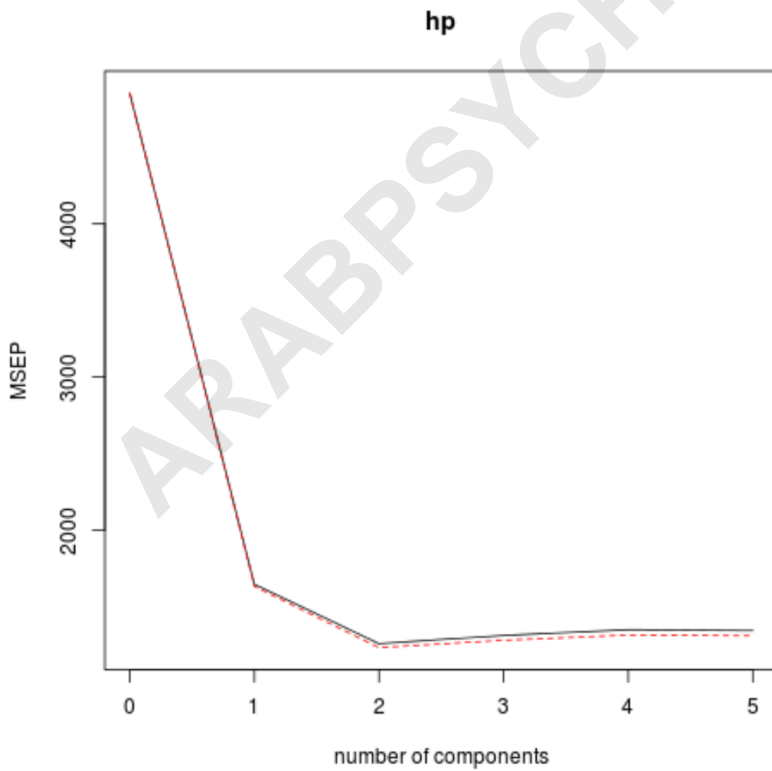
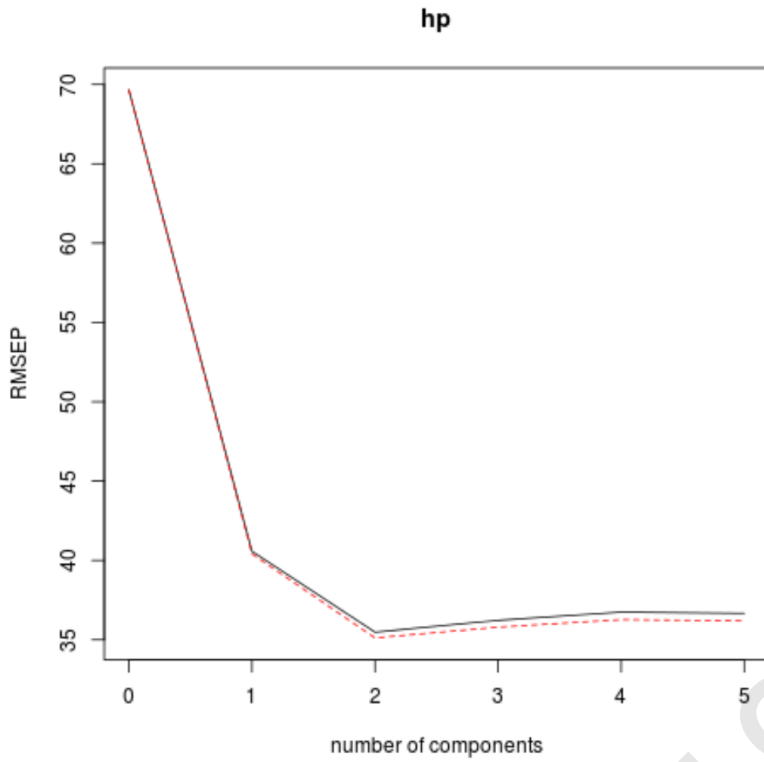
We can also visualize the test RMSE (along with the test MSE and R-squared) based on the number of PLS components by using the validationplot() function.

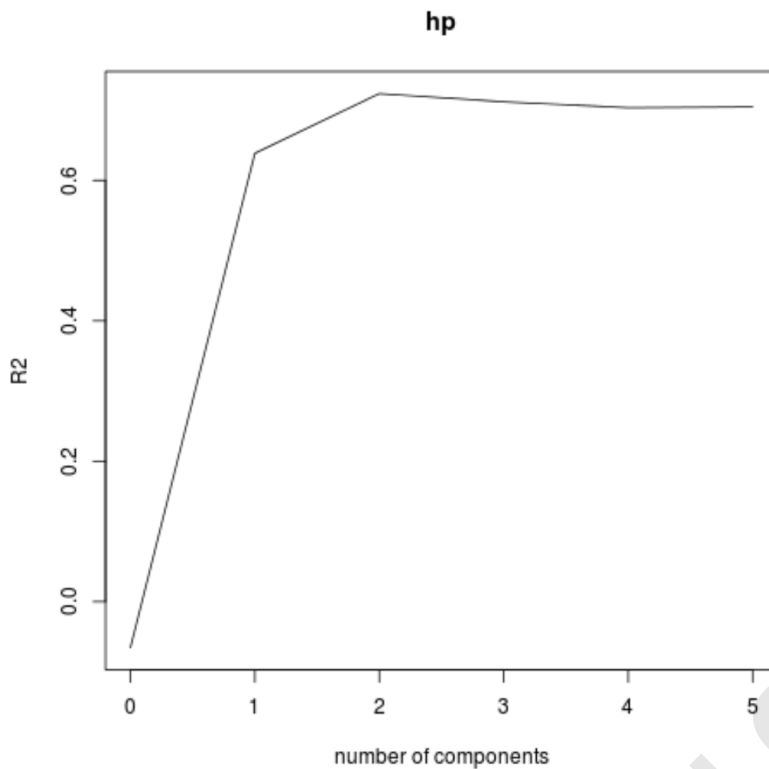
#visualize cross-validation plots

validationplot(model)

validationplot(model, val.type="MSEP")

validationplot(model, val.type="R2")





In each plot we can see that the model fit improves by adding in two PLS components, yet it tends to get worse when we add more PLS components.

Thus, the optimal model includes just the first two PLS components.

Step 4: Use the Final Model to Make Predictions

We can use the final model with two PLS components to make predictions on new observations.

The following code shows how to split the original dataset into a training and testing set and use the final

model with two PLS components to make predictions on the testing set.

```
#define training and testing sets
```

```
train <- mtcars
```

```
y_test <- mtcars
```

```
test <- mtcars
```

```
#use model to make predictions on a test set
```

```
model <- plsr(hp~mpg+disp+drat+wt+qsec, data=train,  
scale=TRUE, validation="CV")
```

```
pcr_pred <- predict(model, test, ncomp=2)
```

```
#calculate RMSE
```

```
sqrt(mean((pcr_pred - y_test)^2))
```

```
54.89609
```

We can see that the test RMSE turns out to be 54.89609. This is the average deviation between the predicted value for *hp* and the observed value for *hp* for the observations in the testing set.

Note that an equivalent principal components regression model with two principal components

produced a test RMSE of 56.86549. Thus, the PLS model slightly outperformed the PCR model for this dataset.

The complete R code use in this example can be found [here](#).

ARABPSYCHOLOGY.COM