How do you perform Principal Components Regression in R, step-by-step?

Authored by

stats writer

April 22, 2024

Principal Components Regression (PCR) is a statistical technique used for dimensionality reduction and predictive modeling, especially when dealing with high-dimensional data. In this method, the original variables are transformed into a smaller set of uncorrelated variables, known as principal components, which capture most of the variation in the data. This allows for a more efficient and accurate regression analysis.

To perform Principal Components Regression in R, follow these steps:

1. Load the necessary packages: Start by loading the "caret" and "pls" packages in R, as they contain the necessary functions for performing PCR.

2. Pre-process the data: Before applying PCR, it is important to pre-process the data by scaling and centering all the variables. This ensures that each variable has a similar scale and prevents any bias towards variables with larger values.

3. Split the data into training and test sets: Next, split the data into two sets, one for training the model and the other for testing its performance. The training set typically contains 70-80% of the data, while the remaining 20-30% is used for testing.

4. Perform Principal Components Analysis (PCA): Use the "prcomp" function to perform PCA on the training data. This function calculates the principal components and their corresponding loadings.

5. Determine the number of components: Plot a scree plot to visualize the proportion of variation explained by each principal component. Based on the plot, select the number of components that explain the majority of the variation in the data.

6. Build the PCR model: Use the "pcr" function from the "pls" package to build the PCR model. Specify the number of components chosen in the previous step and also set the method for cross-validation.

7. Tune the model: Use the "train" function from the "caret" package to tune the PCR model by selecting the optimal number of components through cross-validation.

8. Test the model: Once the model is trained and tuned, use it to predict the values of the test data. Measure the model's performance using metrics such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE).

9. Interpret the results: Finally, interpret the results by examining the importance of each principal component and its corresponding loading values. This can help identify which variables have the most influence on the outcome.

Overall, Principal Components Regression in R is a useful and powerful tool for reducing the

dimensionality of data and building predictive models. Following these steps will help you effectively implement this technique in your data analysis.

Principal Components Regression in R (Step-by-Step)

Given a set of *p* predictor variables and a response variable, multiple linear regression uses a method known as least squares to minimize the sum of squared residuals (RSS):

RSS = Σ(yi - ?i)2

where:

Σ: A greek symbol that means *sum*yi: The actual response value for the ith observation?i: The predicted response value based on the multiple linear regression model

However, when the predictor variables are highly correlated then multicollinearitycan become a problem. This can cause the coefficient estimates of the model to be unreliable and have high variance.

One way to avoid this problem is to instead use principal components regression, which finds *M* linear

combinations (known as "principal components") of the original *p* predictors and then uses least squares to fit a linear regression model using the principal components as predictors.

This tutorial provides a step-by-step example of how to perform principal components regression in R.

Step 1: Load Necessary Packages

The easiest way to perform principal components regression 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 PCR 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 principal components regression (PCR) model using *hp* as the response variable and the following variables as the predictor variables:

```
mpgdispdratwtqsec
```

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

scale=TRUE: This tells R that each of the predictor variables 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 <- pcr(hp~mpg+disp+drat+wt+qsec, data=mtcars,
scale=TRUE, validation="CV")
```

Step 3: Choose the Number of Principal Components

Once we've fit the model, we need to determine the number of principal components worth keeping.

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: 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

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 principal

component, the test RMSE drops to 44.56.If we add in the second principal component, the test RMSE drops to 35.64.

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

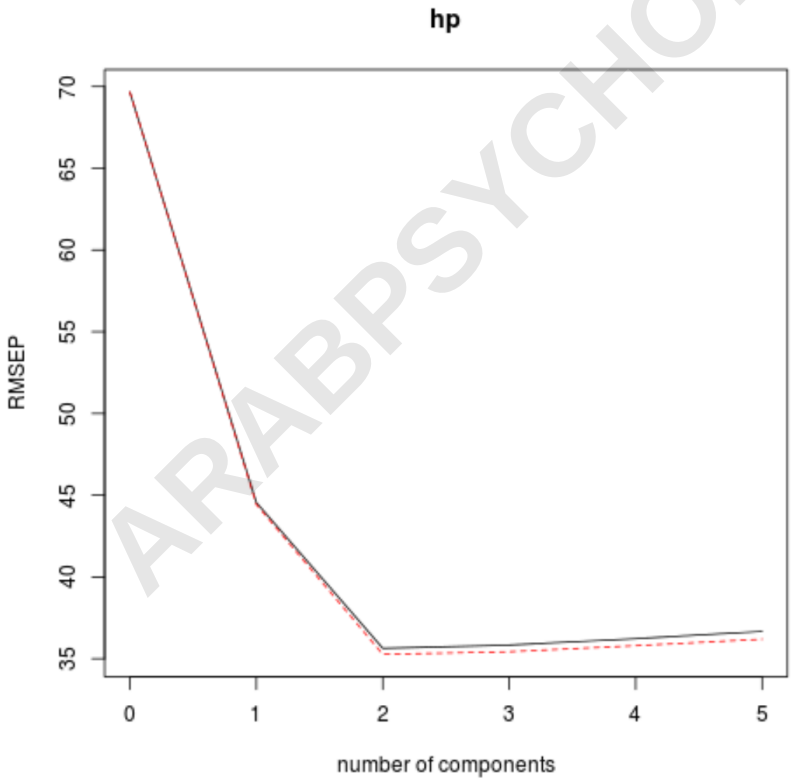
By using just the first principal component, we can explain 69.83% of the variation in the response variable.By adding in the second principal component, we can explain 89.35% of the variation in the response variable.

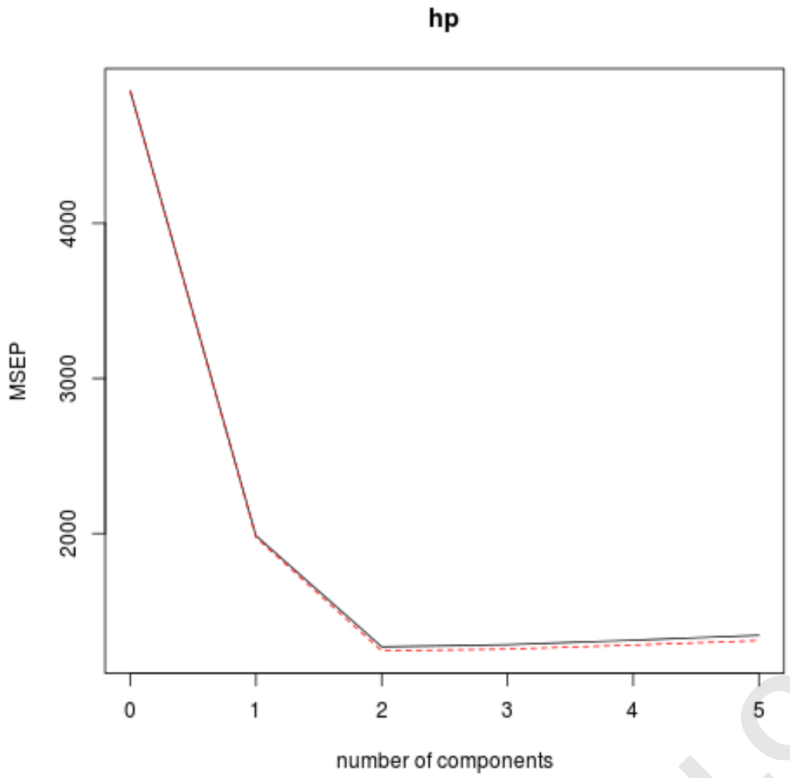
Note that we'll always be able to explain more variance by using more principal components, but we can see that adding in more than two principal components doesn't actually increase the percentage of explained

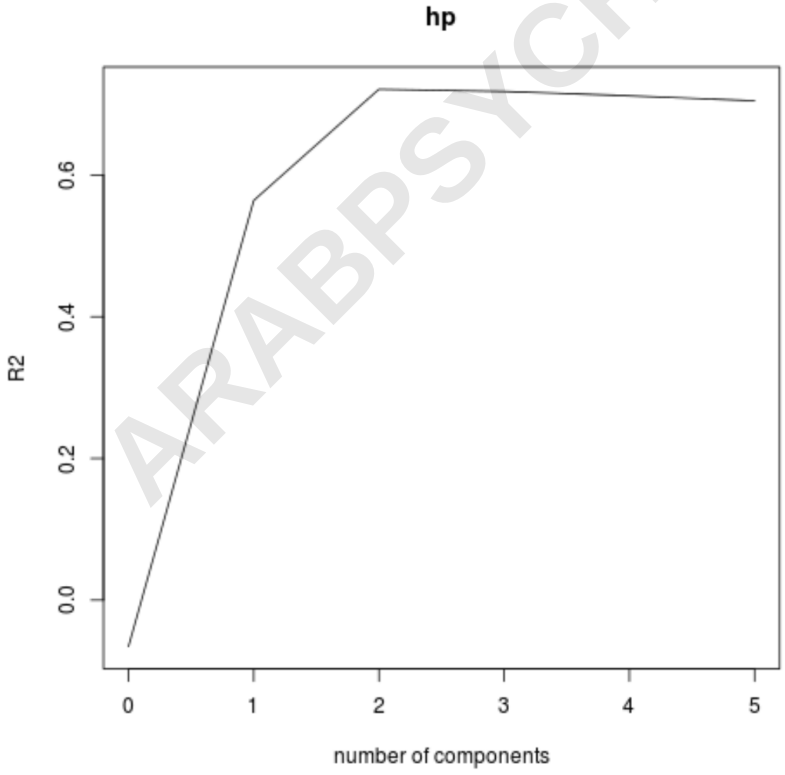
variance by much.

We can also visualize the test RMSE (along with the test MSE and R-squared) based on the number of principal 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 principal components, yet it tends to get worse when we add more principal components.

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

Step 4: Use the Final Model to Make Predictions

We can use the final PCR model with two principal 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 PCR model with two principal 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 <- pcr(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))

56.86549

We can see that the test RMSE turns out to be 56.86549. This is the average deviation between the predicted value for *hp* and the observed value for *hp* for the observations in the testing set.

The complete R code use in this example can be found here.