

How do I perform Ridge Regression in R using a step-by-step approach?

Authored by
stats writer

April 22, 2024

RECOMMENDED CITATION

stats writer (2024). *How do I perform Ridge Regression in R using a step-by-step approach?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=137952>

Ridge Regression is a statistical technique used to handle the problem of multicollinearity in linear regression models. It works by adding a penalty term to the sum of squared errors, which helps to reduce the impact of highly correlated independent variables on the model. In this formal description, we will explain how to perform Ridge Regression in R using a step-by-step approach.

Step 1: Load the necessary packages

Firstly, you need to load the packages necessary for performing Ridge Regression in R. These include the "glmnet" package, which contains the functions for fitting Ridge Regression models, and the "caret" package, which is used for data preprocessing and model evaluation.

Step 2: Load and prepare the data

Next, you need to load the data that you want to use for your Ridge Regression model. This can be done using the "read.csv" function or by importing data from an external source. Once the data is loaded, it is recommended to check for missing values and outliers, and perform any necessary data cleaning and preprocessing steps.

Step 3: Split the data into training and test sets

To evaluate the performance of our Ridge Regression model, we need to split the data into two sets - a training set and a test set. The training set will be used to build the model, while the test set will be used to evaluate its performance. This can be done using the "createDataPartition" function from the "caret" package.

Step 4: Fit the Ridge Regression model

Now, we can use the "glmnet" function to fit a Ridge Regression model on the training data. This function takes in the independent variables (X) and the dependent variable (Y) as inputs, along with the tuning parameter "alpha" which controls the amount of shrinkage applied to the model. The optimal value of alpha can be determined using cross-validation.

Step 5: Evaluate the model

After fitting the model, we can evaluate its performance on the test set. This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) between the predicted and actual values. A lower value of MSE or RMSE indicates a better performing model.

Step 6: Fine-tune the model

If the model performance is not satisfactory, you can fine-tune the model by changing the value of alpha or by including more variables in the model. This process can be repeated until you are satisfied with the model's performance.

Step 7: Make predictions

Finally, you can make predictions on new data using the final Ridge Regression model. This can be done by using the "predict" function and providing the new data as input.

In conclusion, performing Ridge Regression in R involves loading the necessary packages, preparing the data, fitting the model, evaluating its performance, fine-tuning the model if needed, and making predictions on new data. This step-by-step approach can help you effectively implement Ridge Regression in your data analysis process.

Ridge Regression in R (Step-by-Step)

Ridge regression is a method we can use to fit a regression model when multicollinearity is present in the data.

In a nutshell, least squares regression tries to find coefficient estimates that minimize the sum of squared residuals (RSS):

$$RSS = \sum (y_i - \hat{y}_i)^2$$

where:

**Σ : A greek symbol that means *sum*
 y_i : The actual response value for the i th observation
 \hat{y}_i : The predicted response value based on the multiple linear regression model**

Conversely, ridge regression seeks to minimize the following:

$$RSS + \lambda \sum \beta_j^2$$

where j ranges from 1 to p predictor variables and $\lambda \geq 0$.

This second term in the equation is known as a *shrinkage penalty*. In ridge regression, we select a value for λ that produces the lowest possible test MSE (mean squared error).

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

Step 1: Load the Data

For this example, we'll use the R built-in dataset called `mtcars`. We'll use `hp` as the response variable and the following variables as the predictors:

```
mpgwt drat qsec
```

To perform ridge regression, we'll use functions from the `glmnet` package. This package requires the response variable to be a vector and the set of predictor variables to be of the class `data.matrix`.

The following code shows how to define our data:

```
#define response variable
```

```
y <- mtcars$hp
```

```
#define matrix of predictor variables
```

```
x <- data.matrix(mtcars)
```

Step 2: Fit the Ridge Regression Model

Note that setting alpha equal to 1 is equivalent to using Lasso Regression and setting alpha to some value between 0 and 1 is equivalent to using an elastic net.

Also note that ridge regression requires the data to be standardized such that each predictor variable has a mean of 0 and a standard deviation of 1.

Fortunately glmnet() automatically performs this standardization for you. If you happened to already standardize the variables, you can specify standardize=False.

```
library(glmnet)
```

```
#fit ridge regression model
```

```
model <- glmnet(x, y, alpha = 0)
```

```
#view summary of model
```

summary(model)

Length Class Mode

a0 100 -none- numeric

beta 400 dgCMatrix S4

df 100 -none- numeric

dim 2 -none- numeric

lambda 100 -none- numeric

dev.ratio 100 -none- numeric

nulldev 1 -none- numeric

npasses 1 -none- numeric

jerr 1 -none- numeric

offset 1 -none- logical

call 4 -none- call

nobs 1 -none- numeric

Step 3: Choose an Optimal Value for Lambda

Next, we'll identify the lambda value that produces the lowest test mean squared error (MSE) by using **k-fold cross-validation**.

Fortunately, glmnet has the function `cv.glmnet()` that automatically performs k-fold cross validation using **k = 10 folds**.

#perform k-fold cross-validation to find optimal lambda value

```
cv_model <- cv.glmnet(x, y, alpha = 0)
```

#find optimal lambda value that minimizes test MSE

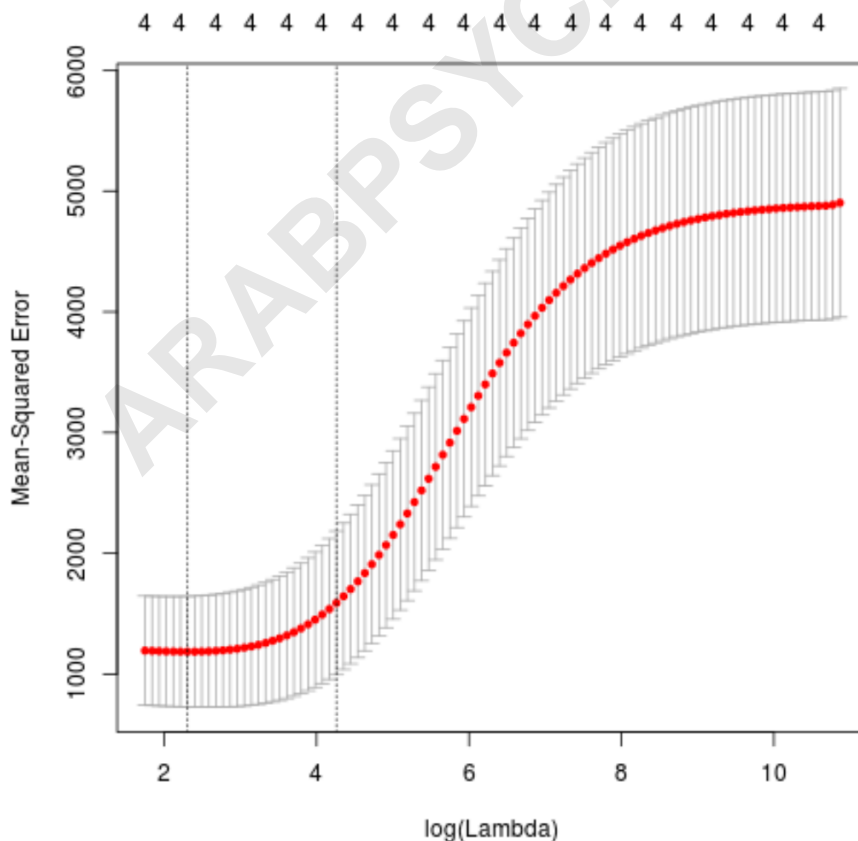
```
best_lambda <- cv_model$lambda.min
```

```
best_lambda
```

10.04567

#produce plot of test MSE by lambda value

```
plot(cv_model)
```



The lambda value that minimizes the test MSE turns out to be 10.04567.

Step 4: Analyze Final Model

Lastly, we can analyze the final model produced by the optimal lambda value.

We can use the following code to obtain the coefficient estimates for this model:

```
#find coefficients of best model  
best_model <- glmnet(x, y, alpha = 0, lambda =  
best_lambda)  
coef(best_model)
```

5 x 1 sparse Matrix of class "dgCMatrix"

s0

(Intercept) 475.242646

mpg -3.299732

wt 19.431238

drat -1.222429

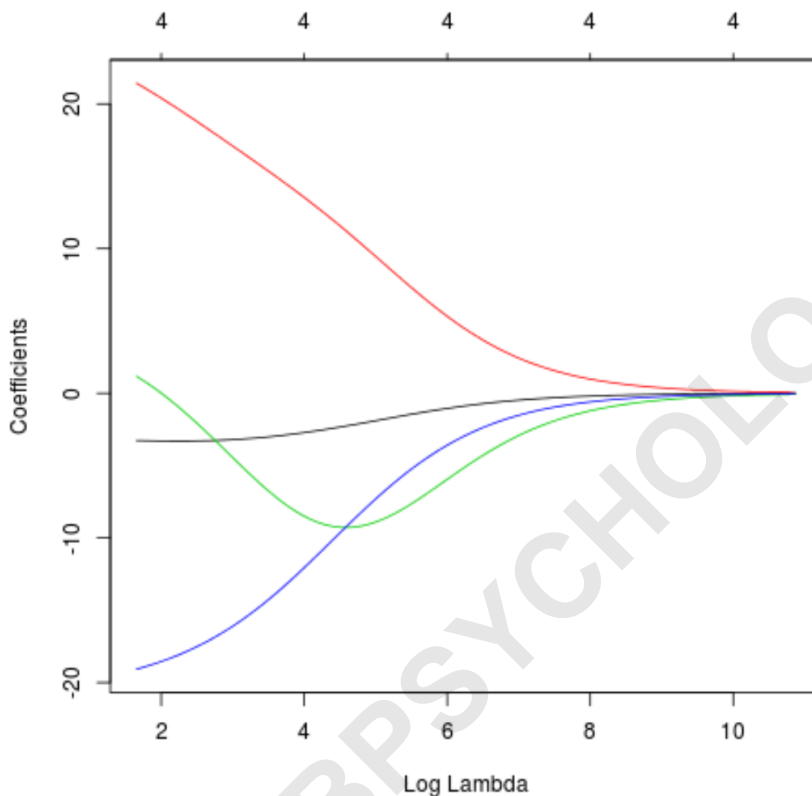
qsec -17.949721

We can also produce a Trace plot to visualize how the coefficient estimates changed as a result of increasing

lambda:

#produce Ridge trace plot

plot(model, xvar = "lambda")



Lastly, we can calculate the R-squared of the model on the training data:

#use fitted best model to make predictions

y_predicted <- predict(model, s = best_lambda, newx = x)

```
#find SST and SSE
```

```
sst <- sum((y - mean(y))^2)
```

```
sse <- sum((y_predicted - y)^2)
```

```
#find R-Squared
```

```
rsq <- 1 - sse/sst
```

```
rsq
```

```
0.7999513
```

The R-squared turns out to be 0.7999513. That is, the best model was able to explain 79.99% of the variation in the response values of the training data.

You can find the complete R code used in this example [here](#).