Penalized Regression

S20426

2025-04-02

Penalized Regression

Penalized regression methods introduce a penalty term to shrink regression coefficients, preventing overfitting and improving model generalizability. These techniques are particularly useful for datasets with multicollinearity or high dimensionality. Below is an explanation of three key methods with R implementations:

Includes a penalty term to reduce (i.e. shrink) the coefficient values towards zero.

As such, the variables with minor contribution to the outcome have their coefficients close to zero.

Method 1 : Ridge Regression

Ridge regression is a technique used in linear regression to address multicollinearity (high correlation among predictor variables) and prevent overfitting. It achieves this by introducing a penalty term to the cost function, shrinking the regression coefficients towards zero

1 .Multicollinearity Problem:

In ordinary least squares (OLS) regression, multicollinearity can lead to large variances in coefficient estimates, making the model unstable and less interpretable. Ridge regression reduces this instability by penalizing large coefficients.

2. Bias-Variance Tradeoff:

Ridge regression introduces bias into the model but reduces variance, leading to more reliable predictions.

```
x_var <- data.matrix(mtcars[, c("hp", "wt", "drat")])
y_var <- mtcars[, "mpg"]
lambda_seq <- 10^seq(2, -2, by = -.1)
fit <- glmnet(x_var, y_var, alpha = 0, lambda  = lambda_seq)
summary(fit)
```

```
##           Length Class     Mode
## a0         41    -none-    numeric
## beta      123    dgCMatrix S4
## df         41    -none-    numeric
## dim         2    -none-    numeric
## lambda     41    -none-    numeric
## dev.ratio  41    -none-    numeric
## nulldev     1    -none-    numeric
## npasses     1    -none-    numeric
## jerr        1    -none-    numeric
## offset      1    -none-    logical
## call        5    -none-    call
## nobs        1    -none-    numeric
```

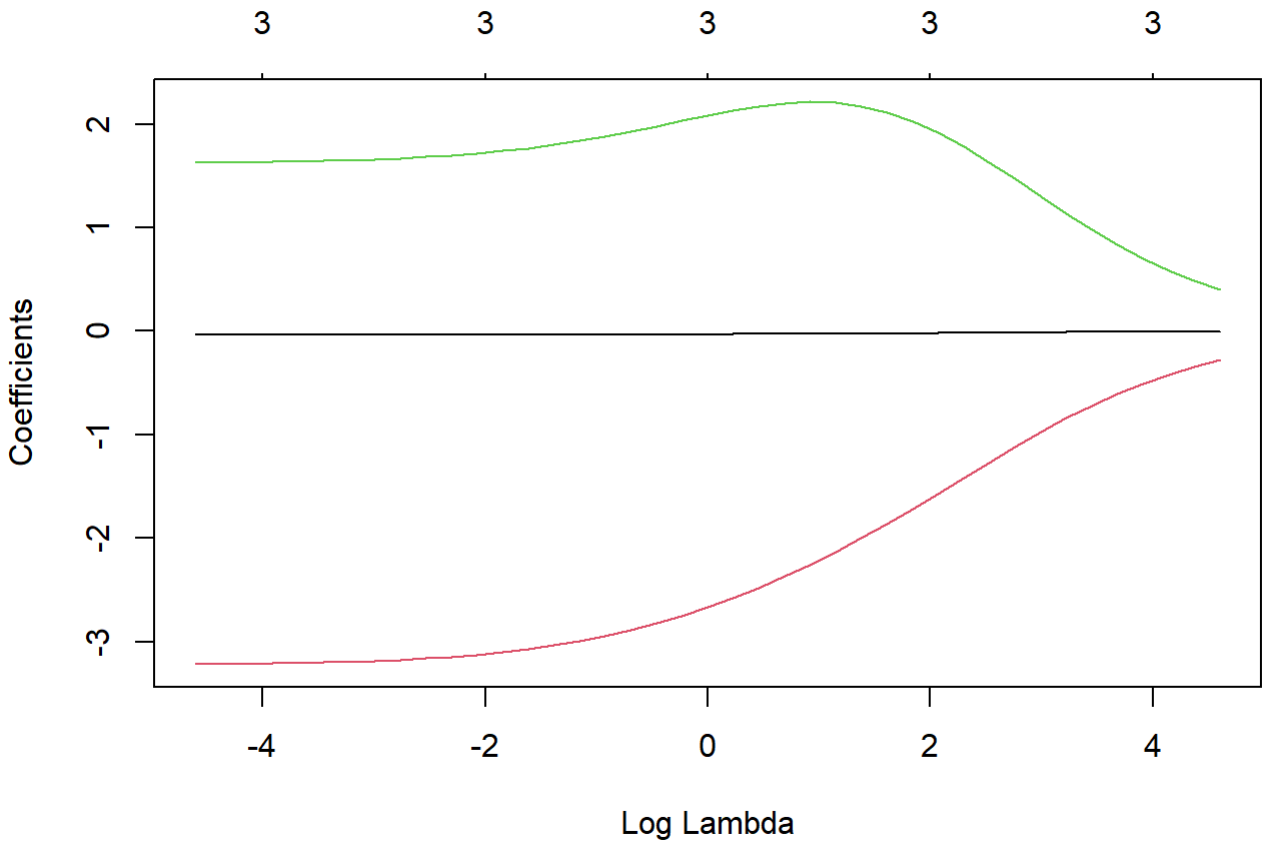
```
# Get coefficients of all 100 models
ridge_coef <- coef(fit)

# Display coefficients for 7 models.
round(ridge_coef[, c(1:3, 38:41)], 3)
```

```
## 4 x 7 sparse Matrix of class "dgCMatrix"
##                  s0     s1     s2    s37    s38    s39    s40
## (Intercept) 20.092 20.100 20.112 29.271 29.293 29.313 29.329
## hp          -0.004 -0.004 -0.005 -0.032 -0.032 -0.032 -0.032
## wt          -0.281 -0.344 -0.418 -3.209 -3.212 -3.215 -3.218
## drat         0.398  0.485  0.587  1.633  1.630  1.627  1.625
```

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

```
plot(fit, xvar ="lambda")
```



Choose an Optimal Value for λ

glmnet has the function cv.glmnet() that performs k-fold cross validation using k = 10 folds.

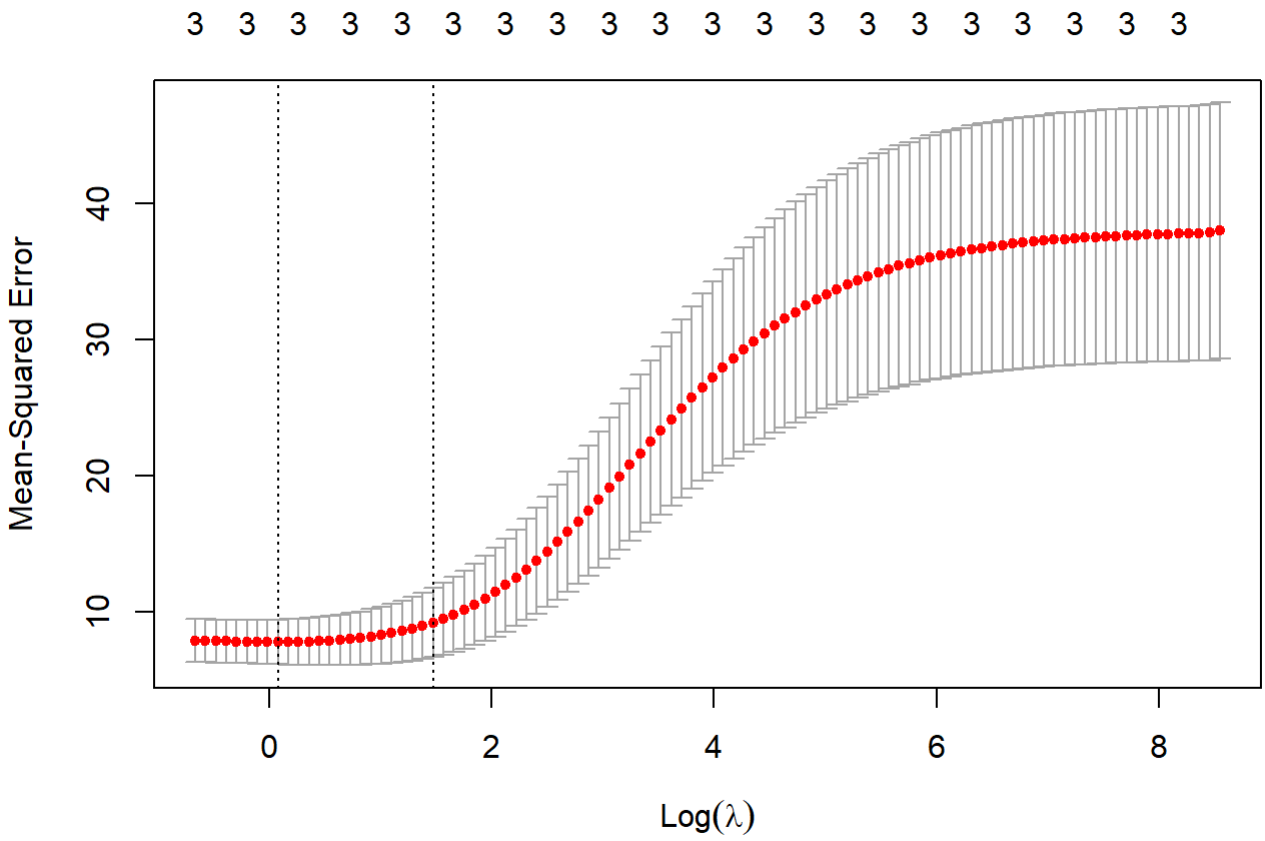
```
set.seed(123)
#perform k-fold cross-validation to find optimal lambda value
cv_model <- cv.glmnet(x_var, y_var, alpha = 0)

#find optimal lambda value that minimizes test MSE
best_lambda <- cv_model$lambda.min
best_lambda
```

```
## [1] 1.08339
```

Plot of MSE by lambda value

```
plot(cv_model)
```



Final Model

```
final_model <- glmnet(x_var, y_var, alpha = 0, lambda = best_lambda)
coef(final_model)
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 25.49107983
## hp          -0.03051333
## wt          -2.63641838
## drat         2.10130824
```

Test the Model using R`2

```
y_predicted <- predict(final_model, s = best_lambda, newx = x_var)

#find SST and SSE
sst <- sum((y_var - mean(y_var))^2)
sse <- sum((y_predicted - y_var)^2)

#find R-Squared
rsq <- 1 - sse/sst
rsq
```

```
## [1] 0.8296018
```

Method 2 : Lasso regression

Lasso Regression (L1 Regularization): Shrinks some coefficients completely to zero, meaning it selects only the most important predictors. This makes the model simpler and easier to interpret.

Ridge Regression (L2 Regularization): Shrinks all coefficients but does not set any to zero. It keeps all predictors in the model but reduces their impact.

When to Use Each:

Lasso is better when only a few predictors have a strong influence, and the rest are almost irrelevant. Since it can eliminate less important variables, it results in a simpler model.

Ridge is better when many predictors contribute equally to the outcome. Instead of removing variables, ridge regression reduces their effect to prevent overfitting.

In short, lasso is good for feature selection, while ridge is better when you expect all predictors to matter to some extent.

Computing Lasso Regression In R

```
# Find the best lambda using cross-validation
set.seed(123)
cvl <- cv.glmnet(x_var, y_var, alpha = 1)
# Display the best lambda value
cvl$lambda.min
```

```
## [1] 0.1034148
```

```
# Fit the final model on the training data
model_lasso <- glmnet(x_var, y_var, alpha = 1, lambda = cvl$lambda.min)
# Dsiplay regression coefficients
coef(model_lasso)
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 29.63708800
## hp          -0.03125498
## wt          -3.21282109
## drat         1.49439939
```

Test the Model using R`2

```
y_predicted <- predict(model_lasso, s = cvl$lambda.min, newx = x_var)

#find SST and SSE
sst <- sum((y_var - mean(y_var))^2)
sse <- sum((y_predicted - y_var)^2)

#find R-Squared
rsq <- 1 - sse/sst
rsq
```

```
## [1] 0.8364553
```

Method 3 : Elastic net regession

Elastic Net Regression is a combination of two popular regression techniques: LASSO (L1 regularization) and Ridge Regression (L2 regularization).

LASSO (L1 penalty) shrinks some coefficients to zero, effectively selecting important features.

Ridge (L2 penalty) shrinks coefficients smoothly, preventing overfitting but keeping all features.

Elastic Net combines both, allowing it to shrink some coefficients while completely eliminating others when needed.

In R programming, the caret package makes it easy to apply Elastic Net regression. It automatically tests different values of alpha (balance between L1 and L2) and lambda (strength of penalty) to find the best combination for the model. Behind the scenes, it uses the glmnet package to perform the actual computation.

Why use Elastic Net?

Handles multicollinearity (highly correlated features).

Can perform feature selection (like LASSO).

Prevents overfitting (like Ridge).

Works well when there are many features.

How to find alpha and lambda?

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
data.new <- data.frame(y_var,x_var)
set.seed(123)
model_ER <- train(
  y_var~.,data=data.new, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# Best tuning parameter
model_ER$bestTune
```

```
##   alpha   lambda
## 6   0.1 0.834975
```

Coefficient of the final model:

```
coef(model_ER$finalModel, model_ER$bestTune$lambda)
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 26.44660083
## hp          -0.03051303
## wt          -2.74813181
## drat         1.93555113
```

R-squared of the elastic net regression model

```
y_predicted <- predict(model_ER, newx = x_var)

#find SST and SSE
sst <- sum((y_var - mean(y_var))^2)
sse <- sum((y_predicted - y_var)^2)

#find R-Squared
rsq <- 1 - sse/sst
rsq
```

```
## [1] 0.8312703
```

Lars Algorithm

Least-angle regression (LARS) is a method used to select important variables in linear regression, especially when there are many variables.

Simple Explanation: 1. Start by normalizing all variables (so they have zero mean and unit variance).

2. Find the most correlated variable with the target (the one that best explains the output).
3. Move in that direction (adjust the regression model towards that variable).
4. If another variable becomes equally correlated, adjust the model in a direction that balances both variables.
5. Repeat until all important variables are included or the model is complex enough.

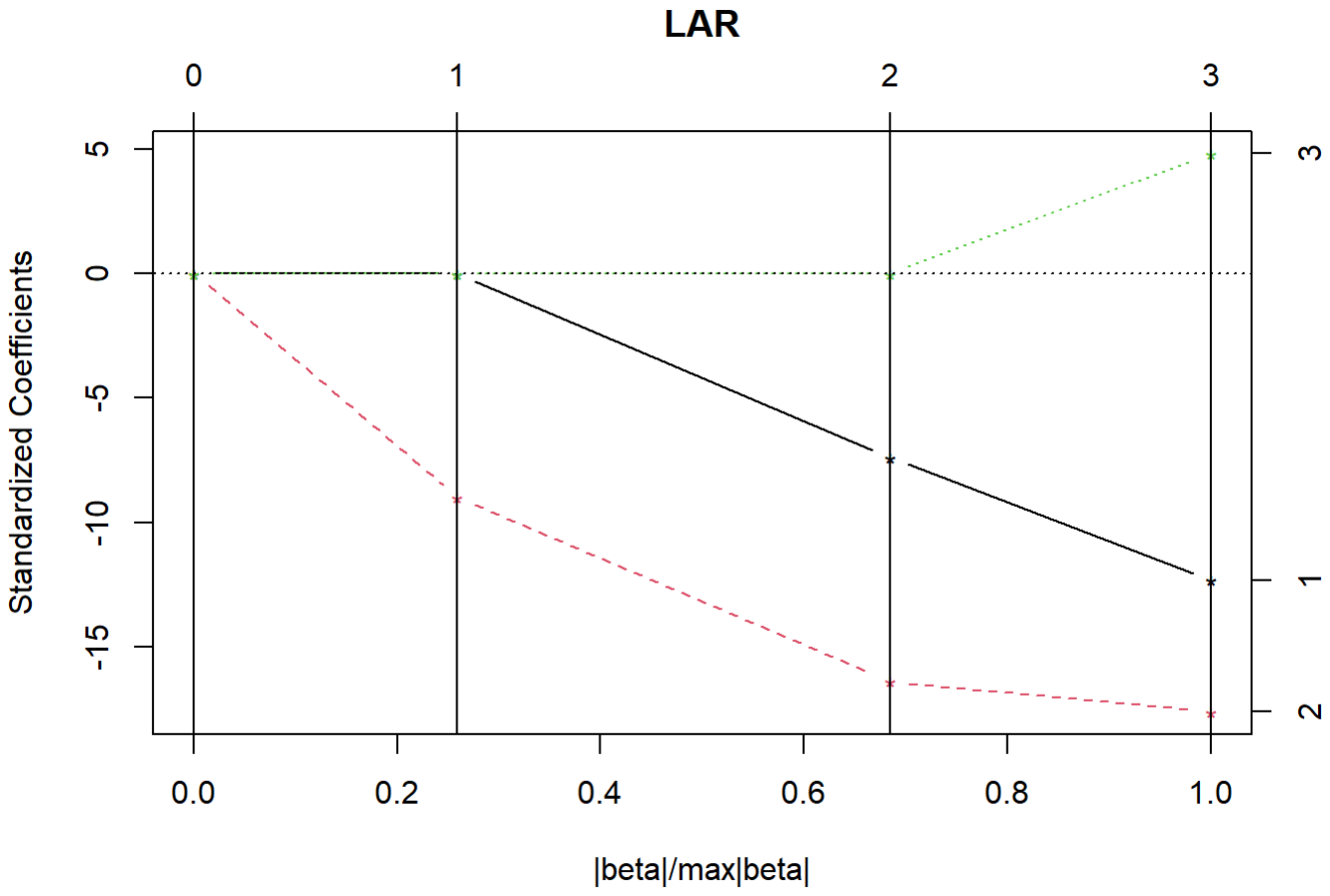
```
library(lars)
```

```
## Loaded lars 1.3
```

```
Lars_obj <- lars(x_var,y_var,type="lar")
Lars_obj
```

```
##
## Call:
## lars(x = x_var, y = y_var, type = "lar")
## R-squared: 0.837
## Sequence of LAR moves:
##      wt hp drat
## Var   2  1    3
## Step  1  2    3
```

```
plot(Lars_obj)
```



Advantages of using LARS:

Computationally as fast as forward selection but may sometimes be more accurate.

Numerically very efficient when the number of features is much larger than the number of data instances.

It can easily be modified to produce solutions for other estimators.

Disadvantages of using LARS:

LARS is highly sensitive to noise and can produce unpredictable results sometimes.

Inclass-Assignment

```
library(MASS)
data("Boston")
#colnames(Boston)
#head(Boston)
```

```
library(glmnet)
x_data <- data.matrix(Boston[,c("crim","zn","indus","chas","nox","rm","age","dis","rad","ta
x","ptratio","black","lstat")])
y_data <- Boston[,"medv"]


set.seed(123)
cv_model <-cv.glmnet(x_data,y_data,alpha=0)
best_lambda <- cv_model$lambda.min

model_Ridge<- glmnet(x_data,y_data,alpha =0 ,lambda=best_lambda)
coef(model_Ridge)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  28.051686825
## crim         -0.087904300
## zn            0.032606201
## indus        -0.038328191
## chas          2.902980774
## nox         -12.005369287
## rm            4.014163735
## age          -0.003862644
## dis          -1.120903830
## rad           0.154161048
## tax          -0.005729860
## ptratio      -0.855862908
## black         0.009068108
## lstat        -0.471596371
```

```
y_predicted <- predict(model_Ridge, s = best_lambda, newx = x_data)

#find SST and SSE
sst <- sum((y_data - mean(y_data))^2)
sse <- sum((y_predicted - y_data)^2)

#find R-Squared
rsq <- 1 - sse/sst
rsq
```

```
## [1] 0.734526
```