# Comparing penalized regression estimators

Mahtab Nahayati

2024-12-18

## Task1

```r
lasso_shooting <- function(X, y, lambda, tol = 1e-8, max_iter = 1000) {
  n <- dim(X)[1]
  p <- dim(X)[2]

  # Initialize coefficients
  beta <- rep(0, p)

  for (iter in 1:max_iter) {
    beta_old <- beta

    for (j in 1:p) {
      # Compute residuals
      r <- y - X %*% beta + X[, j] * beta[j]

      # Update coefficient
      beta[j] <- sign(t(r) %*% X[, j]) * max(0, abs(t(r) %*% X[, j]) - lambda) / sum(X[, j]^2)
    }

    # Check for convergence
    if (sqrt(sum((beta - beta_old)^2)) < tol) {
      break
    }
  }

  return(list(coefficients = beta))
}

lasso_path <- function(X, y, lambda_values) {
  # Initialize empty matrices to store coefficients and lambda values
  coefs <- matrix(0, ncol = length(lambda_values), nrow = ncol(X))
  lambdas <- vector(length = length(lambda_values))

  # Compute lasso for each lambda value
  for (i in 1:length(lambda_values)) {
    lambda <- lambda_values[i]
    lasso_results <- lasso_shooting(X, y, lambda)
    coefs[, i] <- lasso_results$coefficients
    lambdas[i] <- lambda
  }
```

```r
  # Return matrix of coefficients and corresponding lambda values
  list(coefs = coefs, lambdas = lambdas)
}

# Load necessary libraries
library(glmnet)
```

```
## Warning: Paket 'glmnet' wurde unter R Version 4.3.3 erstellt
```

```
## Lade nötiges Paket: Matrix
```

```
## Warning: Paket 'Matrix' wurde unter R Version 4.3.2 erstellt
```

```
## Loaded glmnet 4.1-8
```

```r
library(MASS)
set.seed(123)

# Simulate data
n <- 100
p <- 10
X <- mvrnorm(n, rep(0, p), diag(p))
beta <- runif(p) # true coefficients
y <- X %*% beta + rnorm(n) # response variable

# Apply lasso function
lambda_values <- seq(0.1, 2, by = 0.1)
my_lasso_results <- lapply(lambda_values, function(lambda) lasso_shooting(X, y, lambda))

# Apply glmnet lasso function
glmnet_lasso_results <- glmnet(X, y, alpha = 1, lambda = lambda_values)

# Compare coefficients
my_lasso_coefficients <- sapply(my_lasso_results, function(result) result$coefficients)
glmnet_lasso_coefficients <- as.matrix(coef(glmnet_lasso_results))

# Print coefficients
print("My Lasso Coefficients:")
```

```
## [1] "My Lasso Coefficients:"
```

```r
print(my_lasso_coefficients)
```

```
##               [,1]         [,2]          [,3]        [,4]        [,5]
##  [1,] -0.002997589 -0.001570542 -0.0001434947  0.00000000  0.00000000
##  [2,]  0.121427159  0.120085856  0.1187445532  0.11741906  0.11609534
##  [3,]  0.053941217  0.053389062  0.0528369076  0.05223949  0.05163701
##  [4,]  0.453221950  0.451726433  0.4502309154  0.44874850  0.44726756
##  [5,]  0.523864325  0.522638056  0.5214117877  0.52021657  0.51902483
##  [6,]  0.473747659  0.472669813  0.4715919680  0.47047113  0.46934549
##  [7,]  0.348899313  0.347077896  0.3452564791  0.34376420  0.34230871
##  [8,]  0.152556723  0.151044258  0.1495317935  0.14801424  0.14649612
##  [9,] -0.177945915 -0.176697003 -0.1754480900 -0.17434614 -0.17326063
## [10,]  0.378200108  0.376512528  0.3748249479  0.37316203  0.37150186
##              [,6]        [,7]        [,8]        [,9]       [,10]        [,11]
##  [1,]  0.00000000  0.00000000  0.00000000  0.00000000  0.0000000  0.00000000
```

```
##  [2,]  0.11477162  0.11344789  0.11212417  0.11080045  0.1094767  0.10815300
##  [3,]  0.05103453  0.05043204  0.04982956  0.04922708  0.0486246  0.04802212
##  [4,]  0.44578661  0.44430566  0.44282472  0.44134377  0.4398628  0.43838188
##  [5,]  0.51783308  0.51664134  0.51544959  0.51425784  0.5130661  0.51187435
##  [6,]  0.46821985  0.46709421  0.46596857  0.46484292  0.4637173  0.46259164
##  [7,]  0.34085322  0.33939773  0.33794224  0.33648676  0.3350313  0.33357578
##  [8,]  0.14497800  0.14345988  0.14194176  0.14042364  0.1389055  0.13738740
##  [9,] -0.17217512 -0.17108960 -0.17000409 -0.16891857 -0.1678331 -0.16674754
## [10,]  0.36984170  0.36818154  0.36652137  0.36486121  0.3632010  0.36154088
##             [,12]       [,13]       [,14]      [,15]       [,16]       [,17]
##  [1,]  0.00000000  0.00000000  0.00000000  0.0000000  0.00000000  0.00000000
##  [2,]  0.10682928  0.10550555  0.10418183  0.1028581  0.10153438  0.10021066
##  [3,]  0.04741964  0.04681716  0.04621468  0.0456122  0.04500972  0.04440724
##  [4,]  0.43690093  0.43541998  0.43393904  0.4324581  0.43097714  0.42949619
##  [5,]  0.51068261  0.50949086  0.50829912  0.5071074  0.50591563  0.50472388
##  [6,]  0.46146600  0.46034036  0.45921472  0.4580891  0.45696343  0.45583779
##  [7,]  0.33212029  0.33066480  0.32920931  0.3277538  0.32629834  0.32484285
##  [8,]  0.13586928  0.13435115  0.13283303  0.1313149  0.12979679  0.12827867
##  [9,] -0.16566203 -0.16457651 -0.16349100 -0.1624055 -0.16131997 -0.16023445
## [10,]  0.35988072  0.35822056  0.35656039  0.3549002  0.35324007  0.35157990
##             [,18]       [,19]       [,20]
##  [1,]  0.00000000  0.00000000  0.00000000
##  [2,]  0.09888694  0.09756321  0.09623949
##  [3,]  0.04380476  0.04320228  0.04259980
##  [4,]  0.42801525  0.42653430  0.42505335
##  [5,]  0.50353214  0.50234039  0.50114865
##  [6,]  0.45471215  0.45358651  0.45246087
##  [7,]  0.32338736  0.32193187  0.32047639
##  [8,]  0.12676055  0.12524243  0.12372431
##  [9,] -0.15914894 -0.15806342 -0.15697791
## [10,]  0.34991974  0.34825958  0.34659941
```

```
print("GLMNET Lasso Coefficients:")
```

```
## [1] "GLMNET Lasso Coefficients:"
```

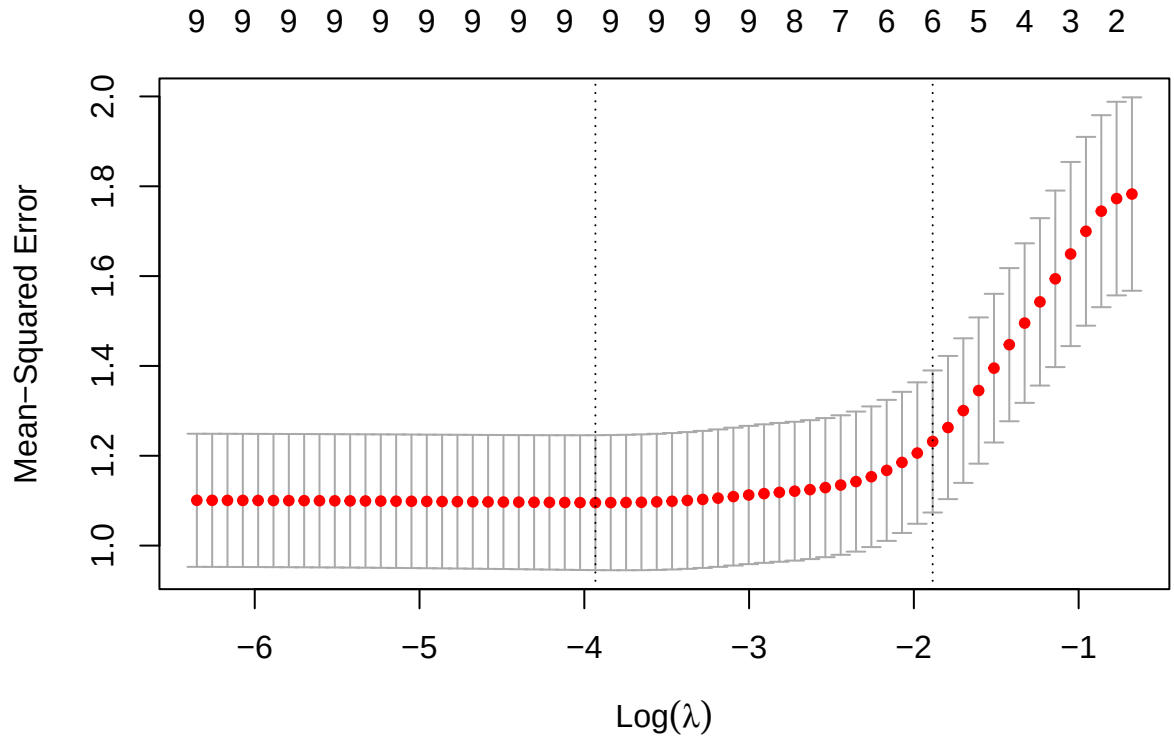
```
print(glmnet_lasso_coefficients)
```

```
##                    s0      s1      s2      s3      s4      s5      s6      s7
## (Intercept) 0.16633 0.16633 0.16633 0.16633 0.16633 0.16633 0.16633 0.16633
## V1          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V2          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V3          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V4          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V5          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V6          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V7          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V8          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V9          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## V10         0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
##                    s8      s9     s10     s11     s12     s13     s14         s15
## (Intercept) 0.16633 0.16633 0.16633 0.16633 0.16633 0.16633 0.16633 0.166712507
## V1          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V2          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V3          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
```

```
## V4          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V5          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.009041818
## V6          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V7          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V8          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V9          0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
## V10         0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000000
##                    s16        s17        s18         s19
## (Intercept) 0.16365863 0.17329917 0.18249801  0.18024634
## V1          0.00000000 0.00000000 0.00000000  0.00000000
## V2          0.00000000 0.00000000 0.00000000  0.00000000
## V3          0.00000000 0.00000000 0.00000000  0.00000000
## V4          0.00000000 0.09641087 0.20825646  0.33074675
## V5          0.10175519 0.19747292 0.30373273  0.42544644
## V6          0.06590066 0.14937318 0.24286139  0.34203244
## V7          0.00000000 0.00000000 0.08500509  0.20933302
## V8          0.00000000 0.00000000 0.00000000  0.00000000
## V9          0.00000000 0.00000000 0.00000000 -0.06106811
## V10         0.00000000 0.00000000 0.05769869  0.20335948
```

```r
cv_lasso <- function(X, y, lambda_seq, nfolds = 10) {
  n <- dim(X)[1]
  p <- dim(X)[2]

  # Initialize matrix to store coefficients
  beta_mat <- matrix(0, nrow = length(lambda_seq), ncol = p)

  # Initialize vector to store mean MSE for each lambda
  mean_mse <- rep(0, length(lambda_seq))

  # Perform k-fold cross-validation
  for (i in 1:length(lambda_seq)) {
    # Define folds
    folds <- sample(1:nfolds, n, replace = TRUE)

    # Initialize vector to store MSE for each fold
    mse <- rep(0, nfolds)

    for (k in 1:nfolds) {
      # Split data into training and test sets
      X_train <- X[folds != k,]
      y_train <- y[folds != k]
      X_test <- X[folds == k,]
      y_test <- y[folds == k]

      # Fit Lasso on training set
      beta <- lasso_shooting(X_train, y_train, lambda_seq[i])

      # Compute MSE on test set
      mse[k] <- mean((y_test - X_test %*% beta$coefficients)^2)
    }

    # Store mean MSE for current lambda
    mean_mse[i] <- mean(mse)
```

```
    # Store coefficients
    beta_mat[i,] <- beta$coefficients
  }

  return(list(lambda = lambda_seq, mse = mean_mse, beta = beta_mat))
}

# Define lambda sequence
lambda_seq <- seq(0.1, 1, length.out = 100)

# Fit Lasso using shooting algorithm
cv_out <- cv_lasso(X, y, lambda_seq)

# Apply glmnet lasso function
cv_model <- cv.glmnet(as.matrix(X), y, alpha = 1, nfolds = 10)
plot(cv_model)
```




```
print(paste("Shooting algorithm MSE:", min(cv_out$mse)))
```

```
## [1] "Shooting algorithm MSE: 1.00843222647659"
```

```
print(paste("glmnet MSE:", min(cv_model$cvm)))
```

```
## [1] "glmnet MSE: 1.09558378673706"
```

## Task 2

```
# Install and load the caret package
install.packages("caret")
```

```
## Installiere Paket nach 'C:/Users/43650/AppData/Local/R/win-library/4.3'
## (da 'lib' nicht spezifiziert)
```

```
## Paket 'caret' erfolgreich ausgepackt und MD5 Summen abgeglichen
```

```
## Warning: kann alte Installation von Paket 'caret' nicht entfernen
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): Problem
## C:\Users\43650\AppData\Local\R\win-library\4.3\00LOCK\caret\libs\x64\caret.dll
## nach C:\Users\43650\AppData\Local\R\win-library\4.3\caret\libs\x64\caret.dll zu
## kopieren: Permission denied
```

```
## Warning: 'caret' wiederhergestellt
```

```
##
## Die heruntergeladenen Binärpakete sind in
##  C:\Users\43650\AppData\Local\Temp\RtmpeYb6BO\downloaded_packages
```

```
library(caret)
```

```
## Warning: Paket 'caret' wurde unter R Version 4.3.3 erstellt
```

```
## Lade nötiges Paket: ggplot2
```

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.3.3 erstellt
```

```
## Lade nötiges Paket: lattice
```

```
## Warning: Paket 'lattice' wurde unter R Version 4.3.2 erstellt
```

```
# Load the Hitters dataset
data(Hitters, package = "ISLR")

# Remove rows with missing salary values
Hitters <- na.omit(Hitters)

# Create model matrix
x <- model.matrix(Salary ~ ., Hitters)[,-1]
y <- Hitters$Salary

# Split data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(y, p = 0.7, list = FALSE)
x_train <- x[trainIndex,]
y_train <- y[trainIndex]
x_test <- x[-trainIndex,]
y_test <- y[-trainIndex]
```

```
library(glmnet)

# Define lambda sequence
lambda_seq <- 10^seq(10, -2, length.out = 50)

# Fit Lasso using shooting algorithm
cv_out <- cv_lasso(x_train, y_train, lambda_seq)
```

```r
# Fit Lasso using glmnet
fit_glmnet <- cv.glmnet(x_train, y_train, alpha = 1, lambda = lambda_seq)

# Compare results
print(paste("Shooting algorithm MSE:", min(cv_out$mse)))
```

```
## [1] "Shooting algorithm MSE: 100394.626253671"
```

```r
print(paste("glmnet MSE:", min(fit_glmnet$cvm)))
```

```
## [1] "glmnet MSE: 104061.293124564"
```

```r
# Fit Ridge regression using glmnet
fit_ridge <- cv.glmnet(x_train, y_train, alpha = 0, lambda = lambda_seq)

# Fit least squares regression using glmnet (lambda = 0)
fit_ls <- glmnet(x_train, y_train, alpha = 0, lambda = 0)
```

```r
# Compute predictions for testing data
pred_lasso <- predict(fit_glmnet, s = fit_glmnet$lambda[which.min(fit_glmnet$cvm)], newx = x_test)

pred_ridge <- predict(fit_ridge, s = fit_ridge$lambda.min, newx = x_test)
pred_ls <- predict(fit_ls, newx = x_test)

# Compute MSE for testing data
mse_lasso <- mean((y_test - pred_lasso)^2)
mse_ridge <- mean((y_test - pred_ridge)^2)
mse_ls <- mean((y_test - pred_ls)^2)

# Print MSE for testing data
print(paste("Lasso MSE:", mse_lasso))
```

```
## [1] "Lasso MSE: 133312.328399838"
```

```r
print(paste("Ridge MSE:", mse_ridge))
```

```
## [1] "Ridge MSE: 137995.27444759"
```

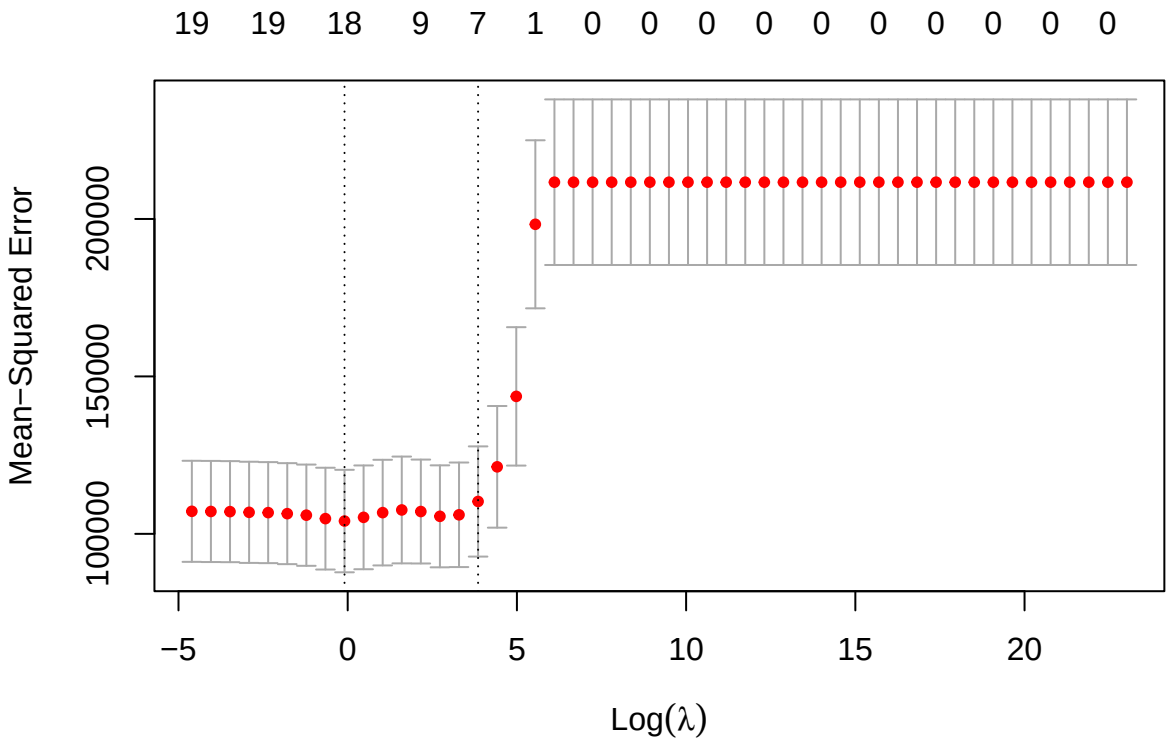
```r
print(paste("LS MSE:", mse_ls))
```

```
## [1] "LS MSE: 140199.789922717"
```

```r
# Plot coefficient paths
plot(fit_glmnet, xvar = "lambda", label = TRUE)
```

```
## Warning in plot.window(...): "xvar" ist kein Grafikparameter

## Warning in plot.window(...): "label" ist kein Grafikparameter

## Warning in plot.xy(xy, type, ...): "xvar" ist kein Grafikparameter

## Warning in plot.xy(xy, type, ...): "label" ist kein Grafikparameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" ist kein
## Grafikparameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" ist kein
## Grafikparameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" ist kein
```

```
## Grafikparameter
## Warning in axis(side = side, at = at, labels = labels, ...): "label" ist kein
## Grafikparameter
## Warning in box(...): "xvar" ist kein Grafikparameter
## Warning in box(...): "label" ist kein Grafikparameter
## Warning in title(...): "xvar" ist kein Grafikparameter
## Warning in title(...): "label" ist kein Grafikparameter
```




The LASSO model has the lowest MSE, indicating it provides the best fit to the testing data. The Ridge model has a slightly higher MSE compared to LASSO but still performs better than the least squares model. The Least squares model has the highest MSE, indicating it provides the worst fit among th e three models. Conclusion: The Lasso regression model is the best choice in this scenario due to its lower MSE and ability to perform variable selection, which can lead to more interpretable models. Ridge regression is a good alternative when we want to retain all predictors in the model. Least squares regression, while simple, is less effective in this context due to its higher MSE and lack of regularization.

## Task 3

The notion of regularised regression, shrinkage and how Ridge regression and LASSO regression differ.

**Regularized regression** is a technique used to prevent overfitting by adding a penalty to the regression model. This penalty discourages the model from fitting the noise in the training data, leading to better generalization on new, unseen data. The two most common forms of regularized regression are Ridge regression and LASSO (Least Absolute Shrinkage and Selection Operator) regression.

**Shrinkage** refers to the process of reducing the magnitude of the regression coefficients. By shrinking the coefficients, the model becomes less sensitive to the variations in the training data, which helps in reducing overfitting. Regularization methods apply shrinkage by adding a penalty term to the loss function that the model tries to minimize.

### Ridge Regression

**Ridge regression** adds a penalty equal to the sum of the squared coefficients (L2 norm) to the loss function. The objective function for Ridge regression is:

$$\text{Minimize} \left( \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

where: - $y_i$ are the observed values, - $\hat{y}_i$ are the predicted values, - $\beta_j$ are the coefficients, - $\lambda$ is the regularization parameter.

The L2 penalty term $\lambda \sum_{j=1}^{p} \beta_j^2$ shrinks the coefficients towards zero but does not set any of them exactly to zero. This means that Ridge regression includes all predictors in the model, but with reduced impact.

### LASSO Regression

**LASSO regression** adds a penalty equal to the sum of the absolute values of the coefficients (L1 norm) to the loss function. The objective function for LASSO regression is:

$$\text{Minimize} \left( \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

The L1 penalty term $\lambda \sum_{j=1}^{p} |\beta_j|$ not only shrinks the coefficients but can also set some of them exactly to zero. This results in a sparse model where only a subset of the predictors are included, making LASSO useful for variable selection.

### Key Differences

1. **Penalty Type**:
   - **Ridge Regression**: Uses L2 norm (sum of squared coefficients).
   - **LASSO Regression**: Uses L1 norm (sum of absolute coefficients).
2. **Effect on Coefficients**:
   - **Ridge Regression**: Shrinks coefficients but does not set any to zero. All predictors remain in the model.
   - **LASSO Regression**: Can shrink some coefficients to zero, effectively performing variable selection.
3. **Use Cases**:
   - **Ridge Regression**: Preferred when we believe all predictors have some effect on the response variable and we want to reduce their impact uniformly.
   - **LASSO Regression**: Preferred when we suspect that only a few predictors are relevant and we want to perform variable selection.