Lasso_OLS_comparison

Birnir

Step 1: Simulate the Data

We will generate a simple dataset with two predictor variables (`x1`, `x2`) and a response variable (`y`), which will have a known linear relationship.

```r
# Set the seed for reproducibility
set.seed(123)

# Simulate data
n <- 100  # number of observations
x1 <- rnorm(n)  # predictor 1
x2 <- rnorm(n)  # predictor 2
x3 = rnorm(n)

# Response variable with some noise
y <- 3 + 0.5 * x1 + 2 * x2 + rnorm(n)

# Combine into a data frame
dat <- data.frame(x1 = x1, x2 = x2, x3 = x3, y = y)

# View the first few rows of the data
head(dat)
```

```
           x1          x2         x3         y
1 -0.56047565 -0.71040656  2.1988103 0.5837069
2 -0.23017749  0.25688371  1.3124130 2.6459897
3  1.55870831 -0.24669188 -0.2651451 2.3474317
4  0.07050839 -0.34754260  0.5431941 1.2876557
5  0.12928774 -0.95161857 -0.4143399 0.7242472
6  1.71506499 -0.04502772 -0.4762469 4.0986562
```

Step 2: Split the Data into Training and Test Sets

```r
# Load the required package
library(caret)
```

```
Loading required package: ggplot2

Loading required package: lattice
```

```r
# Split the data: 70% for training, 30% for testing
set.seed(123)
train_index <- createDataPartition(dat$y, p = 0.7, list = FALSE)
train_data <- dat[train_index, ]
test_data <- dat[-train_index, ]
```

Step 3: Fit the OLS (Ordinary Least Squares) Model

```r
# Fit an OLS model
ols_model <- lm(y ~ x1 + x2 + x3, data = train_data)

# View the model summary to see the coefficients
summary(ols_model)
```

```
Call:
lm(formula = y ~ x1 + x2 + x3, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.05946 -0.67524  0.08166  0.63739  1.96975 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.92877    0.11985  24.436  < 2e-16 ***
x1           0.40712    0.13425   3.033  0.00343 ** 
x2           2.14785    0.12826  16.746  < 2e-16 ***
x3           0.01886    0.13549   0.139  0.88973    
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.002 on 68 degrees of freedom
Multiple R-squared:  0.8088,    Adjusted R-squared:  0.8003
F-statistic: 95.86 on 3 and 68 DF,  p-value: < 2.2e-16
```

```r
# Predict on the test data
ols_predictions <- predict(ols_model, newdata = test_data)
```

Step 4: Fit the Lasso Regression Model using `glmnet`.

Use cross-validation to find the optimal value of `lambda`.

```r
# Load the glmnet package
library(glmnet)
```

```
Loading required package: Matrix
```

```
Loaded glmnet 4.1-8
```

```r
# Prepare the predictors and response variable
X_train <- as.matrix(train_data[, c("x1", "x2", "x3")])  # predictors for training
y_train <- train_data$y  # response for training
X_test <- as.matrix(test_data[, c("x1", "x2", "x3")])  # predictors for testing

# Fit a Lasso model with cross-validation to find the best lambda
lasso_cv_model <- cv.glmnet(X_train, y_train, alpha = 1)
#cv.glmnet(): This function fits a generalized linear model using Lasso regularization and

#alpha = 1: This sets the model to use Lasso regression. In the glmnet function, alpha det
#alpha = 1: Lasso (L1 regularization).
#alpha = 0: Ridge regression (L2 regularization).
#alpha between 0 and 1: Elastic Net (a mix of both Lasso and Ridge).


# Find the best lambda
best_lambda <- lasso_cv_model$lambda.min

# Print the best lambda
```

```r
print(paste("Best Lambda (Lasso):", best_lambda))
```

```
[1] "Best Lambda (Lasso): 0.0395780954412222"
```

```r
# Fit the final Lasso model using the best lambda
lasso_model <- glmnet(X_train, y_train, alpha = 1, lambda = .09)
#Tuning of lambda - making the penalty bigger (more bias less variance)
#lasso_model <- glmnet(X_train, y_train, alpha = 1, lambda = .9)


# Predict on the test data using the Lasso model
lasso_predictions <- predict(lasso_model, newx = X_test)
```

Step 5: Calculate Comparison Metrics

There are several metrics to evaluate model performance

1. MSE (**Mean Squared Error)**
2. **RMSE (Root Mean Squared Error)**: The square root of MSE, making it more interpretable in the same units as the dependent variable. Lower RMSE indicates better fit.
3. **MAE (Mean Absolute Error)**: Measures the average absolute difference between predicted and actual values. Unlike MSE, it doesn't square the errors, so it's less sensitive to large (outlier) errors. Lower MSE indicates better fit.
4. **R-squared (R^2)**: Measures how well the model explains the variability of the response variable. Higher values indicate that the model explains more variance in the data.

```r
# Load required libraries
library(glmnet)
library(caret)

# Function to calculate alternative metrics
calculate_metrics <- function(actual, predicted) {
  mse <- mean((actual - predicted)^2)
  rmse <- sqrt(mse)
  mae <- mean(abs(actual - predicted))
  r_squared <- cor(actual, predicted)^2
  return(data.frame(MSE = mse, RMSE = rmse, MAE = mae, R2 = r_squared))
```