Exercise 7

107.330 - Statistical Simulation and Computerintensive Methods, WS24

11912007 - Yahya Jabary

09.10.2024

```
set.seed(11912007)
```

Task 1

1.1. Write your own function for the lasso using the shooting algorithm. Give default values for the tolerance limit and the maximum number of iterations. Do not forget to compute the coefficient for the intercept.

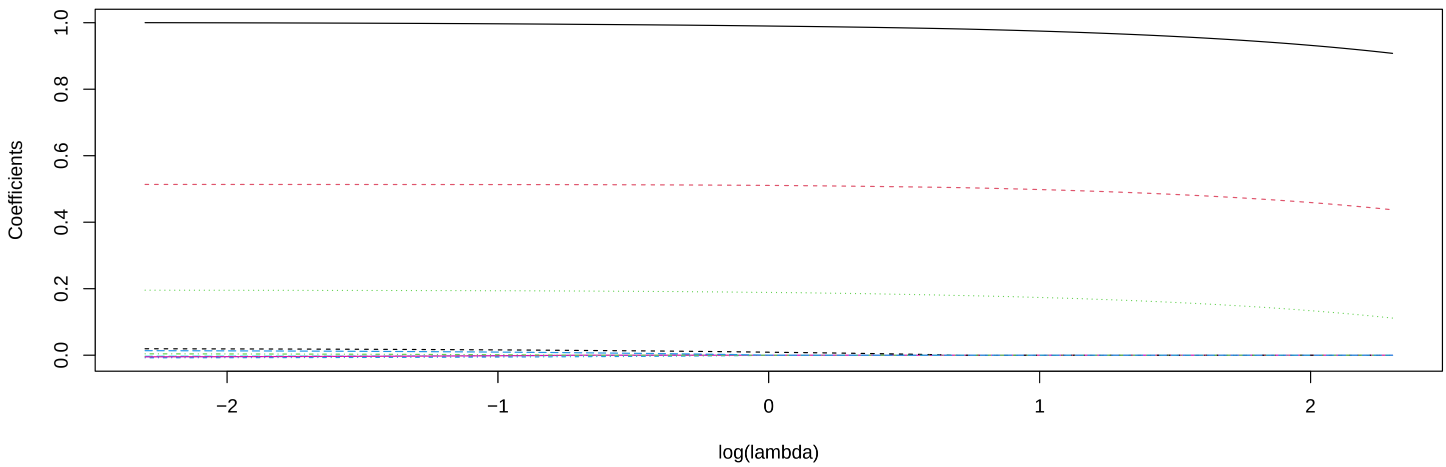
```
lasso_shooting <- function(X, y, lambda, tol = 1e-6, max_iter = 1000) {
  X_centered <- scale(X, center = TRUE, scale = FALSE)
  y_centered <- y - mean(y)
  p <- ncol(X_centered)
  beta <- rep(0, p)

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

    for (j in 1:p) {
      r_j <- y_centered - X_centered[, -j, drop = FALSE] %*% beta[-j] # partial residual
      z_j <- sum(X_centered[, j] * r_j)
      if (z_j > lambda) {
        beta[j] <- (z_j - lambda) / sum(X_centered[, j]^2)
      } else if (z_j < -lambda) {
        beta[j] <- (z_j + lambda) / sum(X_centered[, j]^2)
      } else {
        beta[j] <- 0
      }
    }

    has_converged <- max(abs(beta - beta_old)) < tol
    if (has_converged) {
      break
    }
  }

  intercept <- mean(y) - mean(X %*% beta)
  return(list(beta = beta, intercept = intercept, iterations = iter))
}

# sanity check
X <- matrix(rnorm(100 * 20), 100, 20)
y <- rnorm(100)
lambda <- 0.1

result <- lasso_shooting(X, y, lambda)
beta_coefficients <- result$beta
intercept <- result$intercept
iterations <- result$iterations

print(beta_coefficients)
```

```
##  [1]  0.16265480  0.02823328  0.04291118  0.06198040 -0.05805325  0.13529904
```

```
##  [7] -0.07983239 -0.03530292 -0.01850763 -0.04508443 -0.09551044 -0.24322328
## [13]  0.08264428  0.07490187  0.01890597 -0.09258179 -0.09990968  0.01367262
## [19] -0.04060789 -0.12753408
```

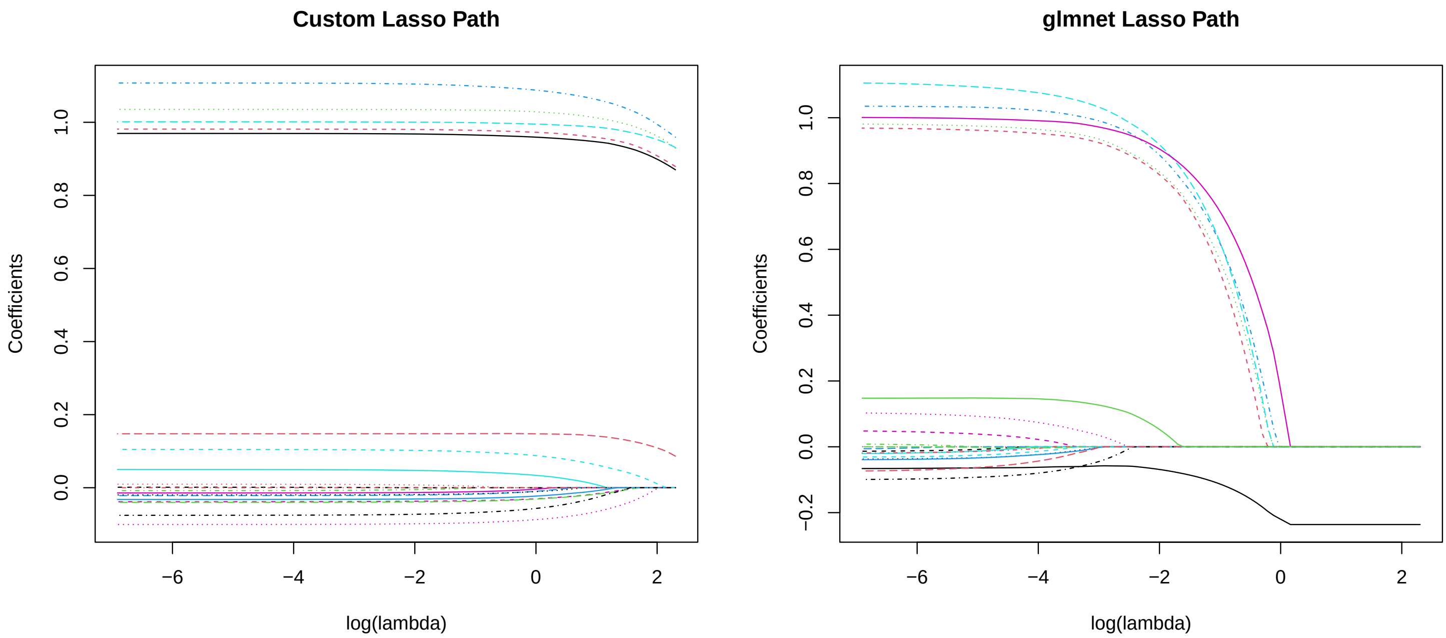
1.2. Write a function which computes the lasso using your algorithm for a vector of λs and which returns the matrix of coefficients and the corresponding λ values.

```
lasso_path <- function(X, y, lambda_vec, tol = 1e-6, max_iter = 1000) {
  n <- nrow(X)
  p <- ncol(X)

  X_centered <- scale(X, center = TRUE, scale = FALSE)
  y_centered <- y - mean(y)

  beta_matrix <- matrix(0, nrow = p, ncol = length(lambda_vec))
  intercept_vec <- numeric(length(lambda_vec))

  for (i in seq_along(lambda_vec)) {
    lambda <- lambda_vec[i]
    beta <- rep(0, p)

    for (iter in 1:max_iter) {
      beta_old <- beta
      for (j in 1:p) {
        r_j <- y_centered - X_centered[, -j, drop = FALSE] %*% beta[-j]
        z_j <- sum(X_centered[, j] * r_j)
        if (z_j > lambda) {
          beta[j] <- (z_j - lambda) / sum(X_centered[, j]^2)
        } else if (z_j < -lambda) {
          beta[j] <- (z_j + lambda) / sum(X_centered[, j]^2)
        } else {
          beta[j] <- 0
        }
      }

      has_converged <- max(abs(beta - beta_old)) < tol
      if (has_converged) {
        break
      }
    }

    beta_matrix[, i] <- beta
    intercept_vec[i] <- mean(y) - mean(X %*% beta)
  }

  return(list(beta = beta_matrix, intercept = intercept_vec, lambda = lambda_vec))
}

# sanity check
n <- 100
p <- 10
X <- matrix(rnorm(n * p), n, p)
y <- X %*% c(1, 0.5, 0.2, rep(0, p-3)) + rnorm(n, 0, 0.1)

lambda_seq <- exp(seq(log(0.1), log(10), length.out = 100))
result <- lasso_path(X, y, lambda_seq)
matplot(log(result$lambda), t(result$beta), type = "l", xlab = "log(lambda)", ylab = "Coefficients")
```



1.3. Compare the performance and output of your functions against the lasso implementation from `glmnet`. Simulate data which helps you to check these properties.

```
n <- 100   # num observations
p <- 20    # num predictors
X <- matrix(rnorm(n*p), nrow=n)
true_beta <- c(rep(1, 5), rep(0, p-5))  # first 5 coefficients are non-zero
y <- X %*% true_beta + rnorm(n, sd=0.5)

lambda_seq <- 10^seq(1, -3, length.out=100)
custom_path <- lasso_path(X, y, lambda_seq)
glmnet_path <- glmnet(X, y, alpha=1, lambda=lambda_seq) # ground truth

# plot
par(mfrow=c(1, 2))
matplot(log(custom_path$lambda), t(custom_path$beta), type="l", xlab="log(lambda)", ylab="Coefficients", mai
matplot(log(glmnet_path$lambda), t(coef(glmnet_path)), type="l", xlab="log(lambda)", ylab="Coefficients", ma
```




```
# perdictive performance
lambda_index <- 50  # arbitrary lambda index
cat("Custom Lasso coefficients:\n")
```

```
## Custom Lasso coefficients:
```

```
print(custom_path$beta[, lambda_index])
```

```
##  [1]  0.9683752923  0.9805452582  1.0348358006  1.1053323653  1.0005863779
##  [6] -0.0133370561  0.0002000387  0.0081002317 -0.0057642964 -0.0203141036
## [11]  0.0480263352 -0.0372408775 -0.0186239027  0.0000000000 -0.0390479892
## [16] -0.0311276918  0.1027915397 -0.0990378537 -0.0732677500  0.1477710542
```

```r
cat("\nglmnet Lasso coefficients:\n")
```

```
##
## glmnet Lasso coefficients:
```

```r
print(coef(glmnet_path)[, lambda_index])
```

```
## (Intercept)          V1          V2          V3          V4          V5
## -0.06276479  0.86133298  0.86934497  0.92605346  0.95649169  0.92972892
##          V6          V7          V8          V9         V10         V11
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         V12         V13         V14         V15         V16         V17
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         V18         V19         V20
##  0.00000000  0.00000000  0.08160352
```

```r
# computational performance
custom_time <- system.time(lasso_path(X, y, lambda_seq))
glmnet_time <- system.time(glmnet(X, y, alpha=1, lambda=lambda_seq))
cat("custom time:", custom_time[3], "s\n")
```

```
## custom time: 0.17 s
```

```r
cat("glmnet time:", glmnet_time[3], "s\n")
```

```
## glmnet time: 0.002 s
```

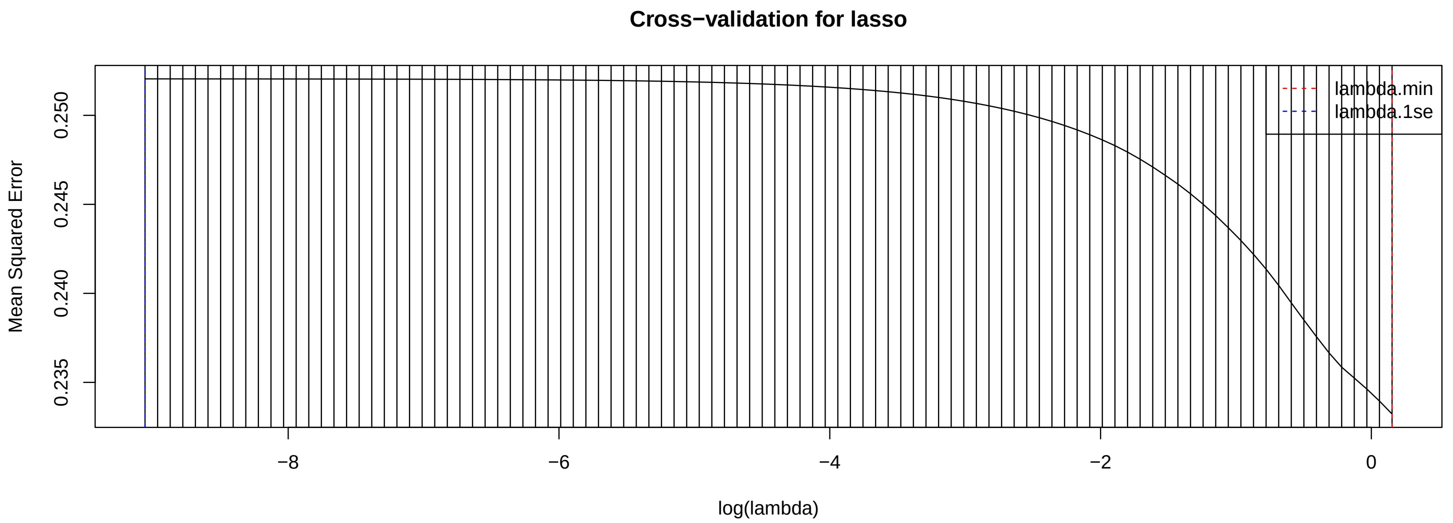
The reason our custom implementation and that of the `glmnet` library don't fully align is due to several subtle implementation differences. The optimization algorithms differ, with `glmnet` using a highly optimized coordinate descent method. Convergence criteria, numerical precision and the use of warm starts in glmnet can lead to slight variations. The handling of the intercept term and the exact sequence of lambda values may also differ between implementations. Additionally, `glmnet`'s approach to cyclical versus random coordinate updates can result in minor path differences.

Despite these nuances the overall trends and major features of the path still align between the two implementations.

1.4. Write a function to perform 10-fold cross-validation for the lasso using MSE as the performance measure. The object should be similarly to the `cv.glmnet` give the same plot and return the λ which minimizes the Root Mean Squared Error and Mean Squared Error, respectively.

```r
cv_lasso <- function(X, y, lambda_vec, nfolds = 10) {
  n <- nrow(X)
  folds <- sample(rep(1:nfolds, length.out = n))
  mse_matrix <- matrix(NA, nrow = nfolds, ncol = length(lambda_vec))
  for (k in 1:nfolds) {
    train_idx <- which(folds != k)
    test_idx <- which(folds == k)

    X_train <- X[train_idx, ]
    y_train <- y[train_idx]
    X_test <- X[test_idx, ]
    y_test <- y[test_idx]

    lasso_fit <- lasso_path(X_train, y_train, lambda_vec)

    for (i in seq_along(lambda_vec)) {
      y_pred <- X_test %*% lasso_fit$beta[, i] + lasso_fit$intercept[i]
      mse_matrix[k, i] <- mean((y_test - y_pred)^2)
    }
  }

  mean_mse <- colMeans(mse_matrix)
  se_mse <- apply(mse_matrix, 2, sd) / sqrt(nfolds)

  lambda_min_idx <- which.min(mean_mse)
  lambda_min <- lambda_vec[lambda_min_idx]
  lambda_1se_idx <- max(which(mean_mse <= mean_mse[lambda_min_idx] + se_mse[lambda_min_idx]))
  lambda_1se <- lambda_vec[lambda_1se_idx]
```

```r
  plot(log(lambda_vec), mean_mse, type = "l", xlab = "log(lambda)", ylab = "Mean Squared Error", main = "Cros
  arrows(log(lambda_vec), mean_mse - se_mse, log(lambda_vec), mean_mse + se_mse, length = 0.05, angle = 90, c
  abline(v = log(lambda_min), lty = 2, col = "red")
  abline(v = log(lambda_1se), lty = 2, col = "blue")
  legend("topright", legend = c("lambda.min", "lambda.1se"), lty = 2, col = c("red", "blue"))

  result <- list(
    lambda = lambda_vec,
    mse = mean_mse,
    se = se_mse,
    lambda.min = lambda_min,
    lambda.1se = lambda_1se,
    mse.min = mean_mse[lambda_min_idx],
    mse.1se = mean_mse[lambda_1se_idx],
    rmse.min = sqrt(mean_mse[lambda_min_idx]),
    rmse.1se = sqrt(mean_mse[lambda_1se_idx])
  )

  class(result) <- "cv_lasso"
  return(result)
}

print.cv_lasso <- function(x, ...) {
  cat("Cross-validated Lasso\n")
  cat("\tlambda.min:", x$lambda.min, "\n")
  cat("\tlambda.1se:", x$lambda.1se, "\n")
  cat("\tMinimum MSE:", x$mse.min, "\n")
  cat("\t1se MSE:", x$mse.1se, "\n")
  cat("\tMinimum RMSE:", x$rmse.min, "\n")
  cat("\t1se RMSE:", x$rmse.1se, "\n")
}

lambda_max <- max(abs(t(X) %*% y)) / nrow(X)
lambda_vec <- exp(seq(log(lambda_max), log(lambda_max * 1e-4), length.out = 100))
cv_result <- cv_lasso(X, y, lambda_vec)
```




```r
print(cv_result)
```

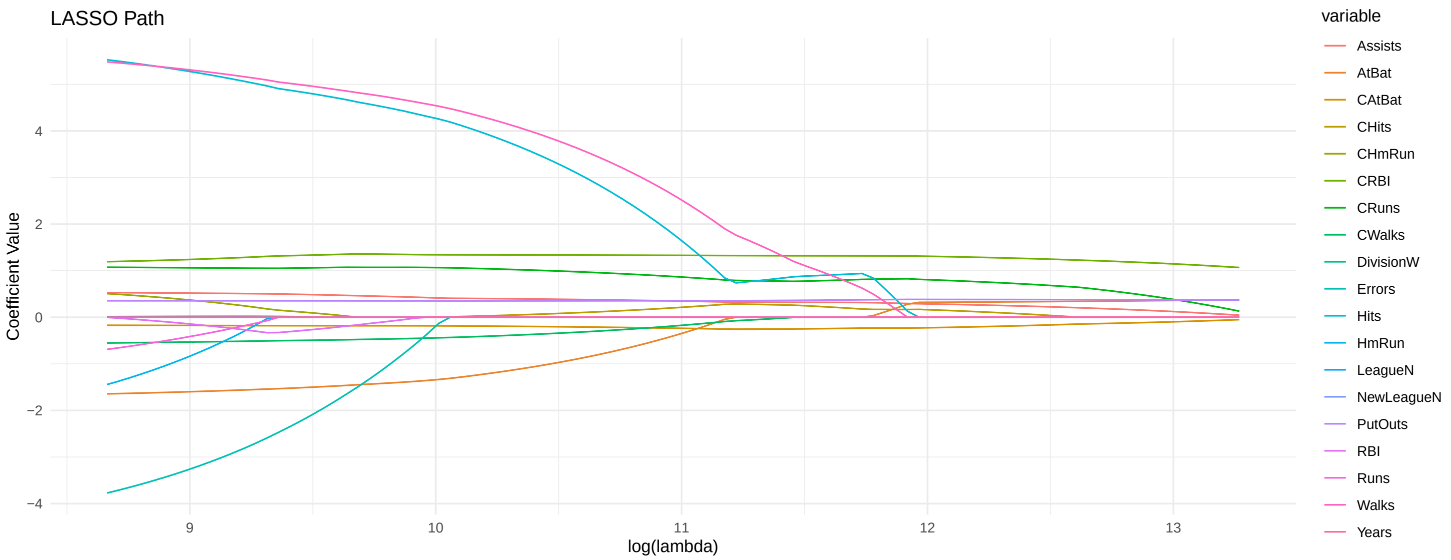
```
## Cross-validated Lasso
##  lambda.min: 1.165089
##  lambda.1se: 0.0001165089
##  Minimum MSE: 0.2332238
##  1se MSE: 0.2520493
##  Minimum RMSE: 0.4829325
##  1se RMSE: 0.5020451
```

Task 2

We will work with the Hitters data in the ISLR package. Take the salary variable as the response variable and create the model matrix x based on all other variables in the data set. Then divide the data into training and testing data with a ratio of 70:30.

2.1. Use your lasso function to decide which λ is best here. Plot also the whole path for the coefficients.

```r
data("Hitters", package = "ISLR")

Hitters <- na.omit(Hitters) # drop rows with missing values
y <- Hitters$Salary
X <- model.matrix(Salary ~ ., data = Hitters)[, -1]  # drop intercept column

# 70/30 holdout
train_indices <- sample(1:nrow(X), 0.7 * nrow(X))
X_train <- X[train_indices, ]
y_train <- y[train_indices]
X_test <- X[-train_indices, ]
y_test <- y[-train_indices]

# lasso path
lambda_max <- max(abs(t(X_train) %*% (y_train - mean(y_train)))) / nrow(X_train)
lambda_min <- 0.01 * lambda_max
lambda_vec <- exp(seq(log(lambda_max), log(lambda_min), length.out = 100))

lasso_result <- lasso_path(X_train, y_train, lambda_vec)

# plot
coef_df <- data.frame(lambda = rep(lasso_result$lambda, each = nrow(lasso_result$beta)),
                      coefficient = as.vector(lasso_result$beta),
                      variable = rep(colnames(X), times = ncol(lasso_result$beta)))
ggplot(coef_df, aes(x = log(lambda), y = coefficient, color = variable)) + geom_line() + theme_minimal() + la
```




```r
# choosing best lambda
k_fold_cv <- function(X, y, lambda_vec, k = 5) {
  n <- nrow(X)
  folds <- sample(rep(1:k, length.out = n))
  mse <- matrix(0, nrow = k, ncol = length(lambda_vec))

  for (i in 1:k) {
    X_train <- X[folds != i, ]
    y_train <- y[folds != i]
    X_val <- X[folds == i, ]
    y_val <- y[folds == i]

    lasso_result <- lasso_path(X_train, y_train, lambda_vec)
```

```r
    for (j in 1:length(lambda_vec)) {
      y_pred <- X_val %*% lasso_result$beta[, j] + lasso_result$intercept[j]
      mse[i, j] <- mean((y_val - y_pred)^2)
    }
  }

  mean_mse <- colMeans(mse)
  best_lambda_index <- which.min(mean_mse)
  best_lambda <- lambda_vec[best_lambda_index]

  return(list(best_lambda = best_lambda, mse = mean_mse))
}

cv_result <- k_fold_cv(X_train, y_train, lambda_vec)
best_lambda <- cv_result$best_lambda
cat("best lambda:", best_lambda, "\n")
```

```
## best lambda: 22295.09
```

```r
# fit final model with best lambda
final_model <- lasso_shooting(X_train, y_train, best_lambda)

non_zero_coefs <- final_model$beta[abs(final_model$beta) > 1e-6]
cat("non-zero coefficients:\n")
```

```
## non-zero coefficients:
```

```r
print(non_zero_coefs)
```

```
##  [1] -1.335367723  4.256662467  4.531359187 -0.183482053  0.007036278
##  [6]  1.066694629  1.342427841 -0.440232595  0.351730058  0.413981125
## [11] -0.130575865
```

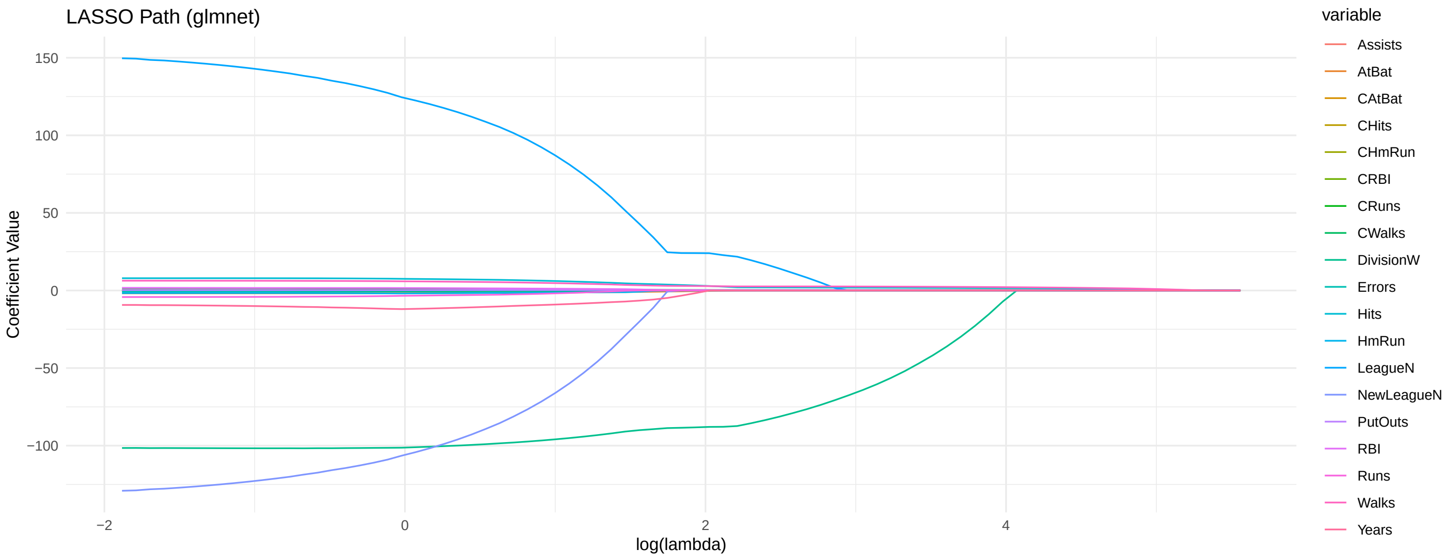
```r
y_pred_test <- X_test %*% final_model$beta + final_model$intercept
ss_total <- sum((y_test - mean(y_test))^2)
ss_residual <- sum((y_test - y_pred_test)^2)
r_squared <- 1 - ss_residual / ss_total
cat("\nr-squared on test set:", r_squared, "\n")
```

```
##
## r-squared on test set: 0.2290198
```

2.2. Compare your fit against the lasso implementation from glmnet.

```r
data("Hitters", package = "ISLR")

Hitters <- na.omit(Hitters)
y <- Hitters$Salary
X <- model.matrix(Salary ~ ., data = Hitters)[, -1]

# 70/30 holdout
train_indices <- sample(1:nrow(X), 0.7 * nrow(X))
X_train <- X[train_indices, ]
y_train <- y[train_indices]
X_test <- X[-train_indices, ]
y_test <- y[-train_indices]

lasso_fit <- glmnet(X_train, y_train, alpha = 1)
coef_matrix <- as.matrix(coef(lasso_fit)[-1, ])  # convert to dense matrix and exclude intercept
lambda_seq <- lasso_fit$lambda
coef_df_glmnet <- data.frame(
  lambda = rep(lambda_seq, each = nrow(coef_matrix)),
  coefficient = as.vector(coef_matrix),
  variable = rep(rownames(coef_matrix), times = ncol(coef_matrix))
)
ggplot(coef_df_glmnet, aes(x = log(lambda), y = coefficient, color = variable)) +
```

```r
  geom_line() +
  theme_minimal() +
  labs(title = "LASSO Path (glmnet)", x = "log(lambda)", y = "Coefficient Value") +
  theme(legend.position = "right")
```




```r
# choosing best lambda
cv_fit <- cv.glmnet(X_train, y_train, alpha = 1)
best_lambda <- cv_fit$lambda.min
cat("best lambda (glmnet):", best_lambda, "\n")
```

```
## best lambda (glmnet): 13.23483
```

```r
# fit final model with best lambda
final_model_glmnet <- glmnet(X_train, y_train, alpha = 1, lambda = best_lambda)

coef_glmnet <- coef(final_model_glmnet)
non_zero_coefs_glmnet <- coef_glmnet[abs(coef_glmnet) > 1e-6]
cat("non-zero coefficients (glmnet):\n")
```

```
## non-zero coefficients (glmnet):
```

```r
print(non_zero_coefs_glmnet)
```

```
## [1] -68.87691640   1.93793631   2.63504948   0.14192952   0.31891699
## [6]   0.04467227  11.03836253 -79.06867777   0.26994393
```

```r
y_pred_test_glmnet <- predict(final_model_glmnet, newx = X_test)
ss_total <- sum((y_test - mean(y_test))^2)
ss_residual <- sum((y_test - y_pred_test_glmnet)^2)
r_squared_glmnet <- 1 - ss_residual / ss_total
cat("\nR-squared on test set (glmnet):", r_squared_glmnet, "\n")
```

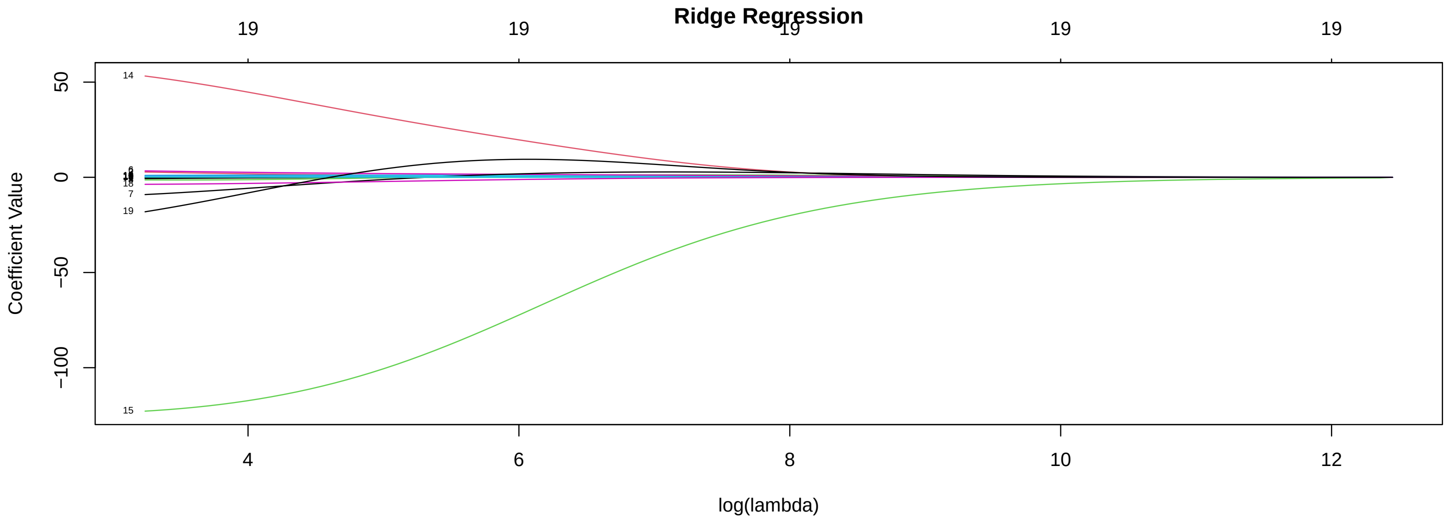
```
##
## R-squared on test set (glmnet): 0.3474943
```

2.3. Fit a ridge regression and a least squares regression for the data (you can use here glmnet).

```r
data("Hitters", package = "ISLR")

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

# fit least squares regression
lsr <- glmnet(x, y, alpha = 0, lambda = 0)
cat("number of non-zero coefficients in least squares regression:", sum(coef(lsr) != 0), "\n")
```

```
## number of non-zero coefficients in least squares regression: 20
```

```r
# fit ridge regression
ridge <- glmnet(x, y, alpha = 0)
plot(ridge, xvar = "lambda", label = TRUE, main = "Ridge Regression", xlab = "log(lambda)", ylab = "Coeffici
```




```r
# find optimal lambda
cv_ridge <- cv.glmnet(x, y, alpha = 0)
best_lambda <- cv_ridge$lambda.min
cat("optimal lambda:", best_lambda, "\n")
```

```
## optimal lambda: 25.52821
```

```r
# fit the final ridge model with optimal lambda
final_ridge <- glmnet(x, y, alpha = 0, lambda = best_lambda) # see: coef(final_ridge)

# make predictions
lsr_pred <- predict(lsr, newx = x)
ridge_pred <- predict(final_ridge, newx = x)
lsr_mse <- mean((y - lsr_pred)^2)
ridge_mse <- mean((y - ridge_pred)^2)
cat("least Squares MSE:", lsr_mse, "\n")
```

```
## least Squares MSE: 92024.46
```

```r
cat("ridge Regression MSE:", ridge_mse, "\n")
```

```
## ridge Regression MSE: 98078.46
```

2.4. Compute the lasso, ridge regression and ls regression predictions for the testing data. Which method gives the better predictions? Interpret all three models and argue about their performances.

```r
data("Hitters", package = "ISLR")

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

# 70/30 holdout
train_index <- createDataPartition(y, p = 0.7, list = FALSE)
x_train <- x[train_index,]
y_train <- y[train_index]
x_test <- x[-train_index,]
y_test <- y[-train_index]

# fit Least Squares Regression
lsr <- glmnet(x_train, y_train, alpha = 0, lambda = 0)

# fit Ridge Regression
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0)
best_lambda_ridge <- cv_ridge$lambda.min
```

```r
ridge <- glmnet(x_train, y_train, alpha = 0, lambda = best_lambda_ridge)

# fit Lasso Regression
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1)
best_lambda_lasso <- cv_lasso$lambda.min
lasso <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda_lasso)

# make predictions on the test set
lsr_pred <- predict(lsr, newx = x_test)
ridge_pred <- predict(ridge, newx = x_test)
lasso_pred <- predict(lasso, newx = x_test)

# get MSE loss
lsr_mse <- mean((y_test - lsr_pred)^2)
ridge_mse <- mean((y_test - ridge_pred)^2)
lasso_mse <- mean((y_test - lasso_pred)^2)
cat("Least Squares MSE:", lsr_mse, "\n")
```

```
## Least Squares MSE: 114992.8
```

```r
cat("Ridge Regression MSE:", ridge_mse, "\n")
```

```
## Ridge Regression MSE: 100813.8
```

```r
cat("Lasso Regression MSE:", lasso_mse, "\n")
```

```
## Lasso Regression MSE: 110085.1
```

```r
best_method <- which.min(c(lsr_mse, ridge_mse, lasso_mse))
methods <- c("Least Squares", "Ridge", "Lasso")
cat("The best method is:", methods[best_method], "\n")
```

```
## The best method is: Ridge
```

Task 3

3.1. Explain the notion of regularised regression, shrinkage and how Ridge regression and LASSO regression differ.

Regularized regression is a technique used to prevent overfitting in linear regression models by adding a penalty term to the cost function. This approach, known as shrinkage, reduces the magnitude of regression coefficients, effectively shrinking them towards zero. The two main types of regularized regression are Ridge regression and LASSO (Least Absolute Shrinkage and Selection Operator) regression.

Ridge regression, also called L2 regularization, adds a penalty term proportional to the sum of squared coefficients. This method shrinks all coefficients towards zero but does not eliminate any features entirely. It is particularly useful when dealing with multicollinearity or when there are many predictors with similar importance.

LASSO regression, or L1 regularization, differs from Ridge regression in that it adds a penalty term proportional to the sum of the absolute values of the coefficients. This approach can shrink some coefficients exactly to zero, effectively performing feature selection. LASSO is beneficial when dealing with high-dimensional data or when you want to identify the most important predictors.

The key difference between Ridge and LASSO lies in their effect on the model coefficients. While Ridge regression reduces all coefficients but keeps them non-zero, LASSO can completely eliminate some features by setting their coefficients to zero. This makes LASSO particularly useful for feature selection and creating simpler, more interpretable models.

Both methods require tuning a hyperparameter (λ) that controls the strength of regularization. The choice between Ridge and LASSO often depends on the specific characteristics of the dataset and the goals of the analysis. In practice, a combination of both methods, known as Elastic Net regression, is sometimes used to leverage the strengths of both approaches.