HW 7

Jared Brotamonte

11/6/2023

Exercise 1

(a)

```r
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.1.3
```

```r
# Set seed for reproducibility
set.seed(2023)

# Load the Wage dataset
data("Wage")

# Define a function for polynomial regression with cross-validation
custom_poly_cross_val <- function(data, degree, folds) {
  # Create an empty vector to store RMSE values
  rmse_values <- numeric(length = folds)

  # Create indices for cross-validation folds
  fold_indices <- sample(1:folds, nrow(data), replace = TRUE)

  # Perform cross-validation
  for (fold in 1:folds) {
    # Extract the training and test sets
    train_data <- data[fold_indices != fold, ]
    test_data <- data[fold_indices == fold, ]

    # Fit the polynomial regression model
    poly_model <- glm(wage ~ poly(age, degree, raw = TRUE), data = train_data)

    # Predict on the test set
    predictions <- predict(poly_model, newdata = test_data)

    # Calculate RMSE
    rmse_values[fold] <- rmse(predictions, test_data$wage)
  }

  # Return RMSE values
  return(rmse_values)
}

# Create a sequence of polynomial degrees to consider
degrees <- 1:10

# Perform 10-fold cross-validation to find the optimal degree
cv_errors <- sapply(degrees, function(degree) {
  mean(custom_poly_cross_val(Wage, degree, folds = 10))
})

# Find the optimal degree with the lowest cross-validation error
optimal_degree <- which.min(cv_errors)

# Print the optimal degree
cat("Optimal Degree for Polynomial Regression:", optimal_degree, "\n")
```

```
## Optimal Degree for Polynomial Regression: 9
```

(b)

```r
# Define a function for custom cross-validation with linear model
custom_linear_cross_val <- function(data, formula, folds) {
  # Create an empty vector to store RMSE values
  rmse_values <- numeric(length = folds)

  # Create indices for cross-validation folds
  fold_indices <- sample(1:folds, nrow(data), replace = TRUE)

  # Perform cross-validation
  for (fold in 1:folds) {
    # Extract the training and test sets
    train_data <- data[fold_indices != fold, ]
    test_data <- data[fold_indices == fold, ]

    # Fit the linear model
    lm_model <- lm(formula, data = train_data)

    # Predict on the test set
    predictions <- predict(lm_model, newdata = test_data)

    # Calculate RMSE
    rmse_values[fold] <- rmse(predictions, test_data$wage)
  }

  # Return RMSE values
  return(rmse_values)
}

# Perform 10-fold cross-validation to calculate RMSE for linear model
linear_rmse_values <- custom_linear_cross_val(Wage, wage ~ year + age + education, folds = 10)
```

```r
# Calculate average RMSE across folds
linear_average_rmse <- mean(linear_rmse_values)

# Print the average RMSE for the linear model
cat("Average RMSE for Linear Model:", linear_average_rmse, "\n")
```

```
## Average RMSE for Linear Model: 35.82164
```

(c)

```r
k_values <- c(5, 5, 5)

# Define a function for custom cross-validation with GAM
custom_gam_cross_val <- function(data, formula, folds, k_values) {
  # Create an empty vector to store RMSE values
  rmse_values <- numeric(length = folds)

  # Create indices for cross-validation folds
  fold_indices <- sample(1:folds, nrow(data), replace = TRUE)

  # Perform cross-validation
  for (fold in 1:folds) {
    # Extract the training and test sets
    train_data <- data[fold_indices != fold, ]
    test_data <- data[fold_indices == fold, ]

    # Fit the GAM model with specified degrees of freedom
    gam_model <- gam(formula, data = train_data, method = "REML", k = k_values)

    # Predict on the test set
    predictions <- predict(gam_model, newdata = test_data)

    # Calculate RMSE
    rmse_values[fold] <- sqrt(mean((predictions - test_data$wage)^2))
  }

  # Return RMSE values
  return(rmse_values)
}

# Define the formula and degrees of freedom for smooth terms
formula_gam <- wage ~ s(year, k = 5) + s(age, k = 5) + s(education, k = 5)
k_values <- c(5, 5, 5)

# Perform 10-fold cross-validation to calculate RMSE for GAM
gam_rmse_values <- custom_gam_cross_val(Wage, formula_gam, folds = 10, k_values = k_values)
```

```
## Error in data[[txt]]: subscript out of bounds
```

```
# Calculate average RMSE across folds
gam_average_rmse <- mean(gam_rmse_values)
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument 'x' in selecting a method for fu
```

```
# Print the average RMSE for the GAM
cat("Average RMSE for GAM using 10-fold cross-validation:", gam_average_rmse, "\n")
```

```
## Error in cat("Average RMSE for GAM using 10-fold cross-validation:", gam_average_rmse, : object 'gam_
```

(d)

```
# Define a function to perform stepwise selection for GAM
stepwise_gam <- function(data, response, predictors, folds = 10) {
  # Convert predictors to factors
  data[predictors] <- lapply(data[predictors], as.factor)

  # Create an empty vector to store selected predictors
  selected_predictors <- predictors

  # Perform stepwise selection
  while (length(selected_predictors) > 0) {
    # Fit the current model with selected predictors
    formula <- as.formula(paste(response, "~", paste(selected_predictors, collapse = "+")))
    gam_model <- gam(formula, data = data, method = "REML")

    # Perform cross-validation to get p-values for predictors
    cv_results <- cv.gam(gam_model, K = folds)

    # Identify predictors with p-values > 0.05
    non_significant_predictors <- names(coef(gam_model))[cv_results$ps > 0.05]

    # Remove non-significant predictors from the selected set
    selected_predictors <- setdiff(selected_predictors, non_significant_predictors)
  }

  # Return the final set of significant predictors
  return(selected_predictors)
}

# Define the response variable
response_var <- "wage"

# Define the predictors (exclude 'logwage')
predictors <- setdiff(names(Wage), c(response_var, "logwage"))

# Perform stepwise selection for significant predictors
selected_predictors <- stepwise_gam(Wage, response_var, predictors)
```

```
## Error in `contrasts<-`(`*tmp*`, value = contr.funs[1 + isOF[nn]]): contrasts can be applied only to f
```

```r
# Fit the final GAM model with selected predictors
final_formula <- as.formula(paste(response_var, "~", paste(selected_predictors, collapse = "+")))
```

```
## Error in paste(selected_predictors, collapse = "+"): object 'selected_predictors' not found
```

```r
final_gam_model <- gam(final_formula, data = Wage, method = "REML")
```

```
## Error in interpret.gam(formula): object 'final_formula' not found
```

```r
# Perform 10-fold cross-validation to calculate RMSE for the final model
final_gam_rmse_values <- custom_gam_cross_val(Wage, final_formula, folds = 10)
```

```
## Error in interpret.gam(formula): object 'final_formula' not found
```

```r
# Calculate average RMSE across folds
final_gam_average_rmse <- mean(final_gam_rmse_values)
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument 'x' in selecting a method for fu
```

```r
# Print the average RMSE for the final GAM model
cat("Average RMSE for Final GAM Model (10-fold cross-validation):", final_gam_average_rmse, "\n")
```

```
## Error in cat("Average RMSE for Final GAM Model (10-fold cross-validation):", : object 'final_gam_aver
```

```r
# Print the selected significant predictors
cat("Selected Predictors in the Final Model:", selected_predictors, "\n")
```

```
## Error in cat("Selected Predictors in the Final Model:", selected_predictors, : object 'selected_predi
```

Exercise 2

(a)

```r
# Set seed for reproducibility
set.seed(2023)

# Load the College dataset
data("College")

# Split the data into a training set (80%) and a test set (20%)
sample_index <- sample(1:nrow(College), nrow(College) * 0.8)
training_data <- College[sample_index, ]
test_data <- College[-sample_index, ]

# Print the dimensions of the training and test sets
cat("Dimensions of Training Set:", dim(training_data), "\n")
```

```
## Dimensions of Training Set: 621 18
```

```r
cat("Dimensions of Test Set:", dim(test_data), "\n")
```

```
## Dimensions of Test Set: 156 18
```

(b)

```r
# Function to perform forward stepwise selection
forward_stepwise <- function(response_var, predictors, data) {
  selected_predictors <- c()  # Initialize an empty set of selected predictors

  while (length(predictors) > 0) {
    remaining_predictors <- setdiff(predictors, selected_predictors)
    models <- lapply(remaining_predictors, function(predictor) {
      formula <- as.formula(paste(response_var, "~", paste(c(selected_predictors, predictor), collapse =
      lm_model <- lm(formula, data = data)
      return(list(predictor = predictor, model = lm_model))
    })

    # Filter out models that result in NA coefficients (singularities)
    valid_models <- Filter(function(m) all(!is.na(coef(m$model))), models)

    if (length(valid_models) > 0) {
      # Get the model with the lowest AIC from valid models
      best_model <- valid_models[[which.min(sapply(valid_models, function(m) AIC(m$model)))]]

      # Update selected predictors
      selected_predictors <- c(selected_predictors, best_model$predictor)
    } else {
      break  # No valid models left, exit the loop
    }
  }

  return(selected_predictors)
}

# Define response variable and predictors
response_var <- "Outstate"
predictors <- setdiff(names(training_data), response_var)

# Perform forward stepwise selection
selected_predictors <- forward_stepwise(response_var, predictors, training_data)

# Print the selected predictors
cat("Selected Predictors:", selected_predictors, "\n")
```

```
## Selected Predictors: Room.Board perc.alumni Expend Private PhD Grad.Rate S.F.Ratio Personal Terminal
```

(c)

```
# Convert selected predictors to numeric if they are factors
training_data[, predictors] <- lapply(training_data[, predictors], as.numeric)

# Define response variable and predictors based on the selected features
response_var <- "Outstate"

# Create a formula for the GAM model with reduced degrees of freedom (k=5)
formula <- as.formula(paste(response_var, "~ s(", paste(predictors, collapse = ", k = 5) + s("), ", k =

# Fit a GAM on the training data
gam_model <- gam(formula, data = training_data)
```

```
## Error in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots): A term has fewer unique covaria
```

```
# Plot the results
plot(gam_model, se = TRUE, col = "blue")
```

```
## Error in plot(gam_model, se = TRUE, col = "blue"): object 'gam_model' not found
```

```
title("GAM Plot: Out-of-State Tuition vs. Selected Predictors")
```

```
## Error in title("GAM Plot: Out-of-State Tuition vs. Selected Predictors"): plot.new has not been calle
```

(d)

```
# Convert selected predictors in the test set to numeric if they are factors
test_data[, predictors] <- lapply(test_data[, predictors], as.numeric)

# Predict out-of-state tuition using the trained GAM model
predicted_values <- predict(gam_model, newdata = test_data)
```

```
## Error in predict(gam_model, newdata = test_data): object 'gam_model' not found
```

```
# Calculate RMSE
rmse <- sqrt(mean((predicted_values - test_data$Outstate)^2))
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument 'x' in selecting a method for fu
```

```
# Print RMSE
cat("Root Mean Squared Error (RMSE) on Test Set:", rmse, "\n")
```

```
## Root Mean Squared Error (RMSE) on Test Set:

## Error in cat("Root Mean Squared Error (RMSE) on Test Set:", rmse, "\n"): argument 2 (type 'closure')
```

(e)

```r
# Partial dependence plots for selected predictors
for (predictor in predictors) {
  partial_plot <- plot.gam(gam_model, select = predictor, scale = 0, shade = TRUE)
  print(partial_plot)
}
```

```
## Error in plot.gam(gam_model, select = predictor, scale = 0, shade = TRUE): object 'gam_model' not fou
```