TP6_Pig

Adrien Gluckman

2024-11-11

```
library(splines2)
library(ggplot2)
```

```
# import data
train <- read.csv("~/Documents/M2QF/MOSA/Projet_Pig_data/pig_data_proj/train1.csv")
test <- read.csv("~/Documents/M2QF/MOSA/Projet_Pig_data/pig_data_proj/test1.csv")
```

Mean function estimation of Pig data

In the first instance, we are interested in identifying the population mean trend in the form of :

$$Weight(t) = r(t) + error(t)$$

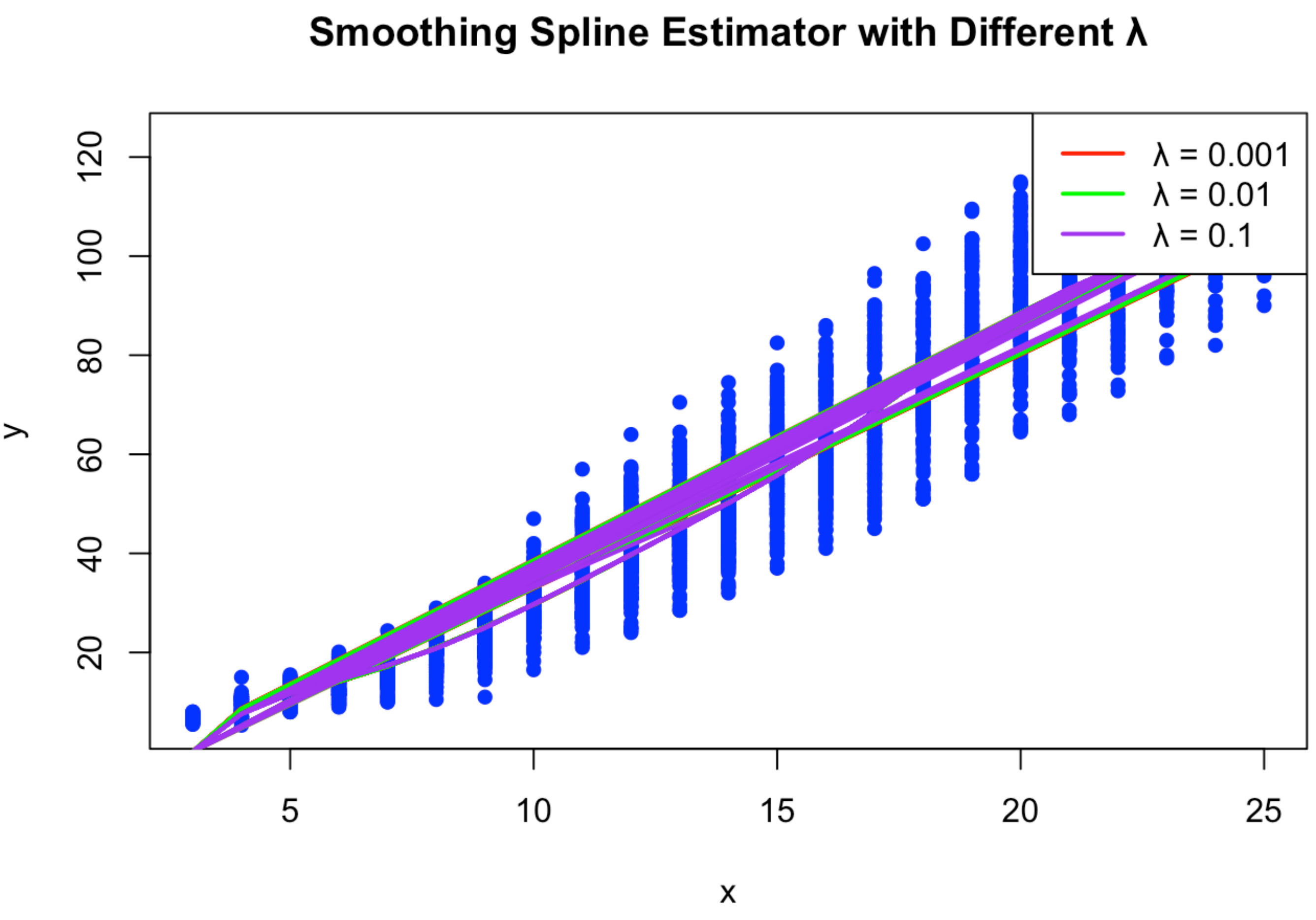
where error has mean zero.

Smoothing splines estimator

```
# Set boundary knots based on the range of x
boundary_knots <- range(train$Age)

# Re-define the penalty matrix function with updated boundary knots
compute_penalty_matrix_dbs_knots <- function(x, num_knots, degree, boundary_knots)
{
  # Generate knots sequence within the boundary range
  knots <- seq(boundary_knots[1], boundary_knots[2], length.out = num_knots + 2)[-
c(1, num_knots + 2)]

  # Bspline matrix using knots
  B <- bSpline(x, knots = knots, degree = degree, Boundary.knots = boundary_knots,
intercept = TRUE)

  # D2 Bspline matrix for the second derivative
  B_second_derivative <- dbs(x, knots = knots, degree = degree, Boundary.knots = bo
undary_knots, derivs = 2)

  # Number of basis functions
  num_basis <- ncol(B_second_derivative)

  # Initialize penalty matrix R
  R <- matrix(0, nrow = num_basis, ncol = num_basis)

  for (i in 1:num_basis) {
    for (j in i:num_basis) {
      # Approximate integral via rectangle method
      R[i, j] <- sum(B_second_derivative[, i] * B_second_derivative[, j]) * (x[2] -
x[1])
      R[j, i] <- R[i, j]  # Symmetric matrix
    }
  }

  return(R)
}

# Example usage
set.seed(123)
x <- train$Age  # Ensure these are your data values
y <- train$Weight

# Update boundary knots based on the actual range of x values
boundary_knots <- range(x)
num_knots <- 8  # Number of internal knots
degree <- 3  # Cubic spline degree

# Compute the penalty matrix
R <- compute_penalty_matrix_dbs_knots(x, num_knots, degree, boundary_knots)
print(R)
```

```
##              [,1]       [,2]        [,3]        [,4]        [,5]        [,6]
##  [1,] 149.294544 -44.803297  -6.1230362   2.4281739   0.0000000   0.0000000
##  [2,] -44.803297  35.810791 -12.7862355  -0.6039620   1.6269998   0.0000000
##  [3,]  -6.123036 -12.786236  25.6845024 -14.5936948  -0.2547323   1.6516513
##  [4,]   2.428174  -0.603962 -14.5936948  25.2504390 -13.9930312  -0.1303174
##  [5,]   0.000000   1.627000  -0.2547323 -13.9930312  25.2631698 -14.0581899
##  [6,]   0.000000   0.000000   1.6516513  -0.1303174 -14.0581899  24.8096630
##  [7,]   0.000000   0.000000   0.0000000   1.6423925  -0.1893421 -13.6087338
##  [8,]   0.000000   0.000000   0.0000000   0.0000000   1.6051259  -0.2370827
##  [9,]   0.000000   0.000000   0.0000000   0.0000000   0.0000000   1.5730095
## [10,]   0.000000   0.000000   0.0000000   0.0000000   0.0000000   0.0000000
## [11,]   0.000000   0.000000   0.0000000   0.0000000   0.0000000   0.0000000
##               [,7]        [,8]        [,9]      [,10]      [,11]
##  [1,]   0.0000000   0.0000000   0.0000000   0.000000   0.000000
##  [2,]   0.0000000   0.0000000   0.0000000   0.000000   0.000000
##  [3,]   0.0000000   0.0000000   0.0000000   0.000000   0.000000
##  [4,]   1.6423925   0.0000000   0.0000000   0.000000   0.000000
##  [5,]  -0.1893421   1.6051259   0.0000000   0.000000   0.000000
##  [6,] -13.6087338  -0.2370827   1.5730095   0.000000   0.000000
##  [7,]  24.2686029 -13.8564063  -0.2203012   1.963788   0.000000
##  [8,] -13.8564063  23.3883240 -10.9178419  -4.100657   4.118538
##  [9,]  -0.2203012 -10.9178419  24.8120067 -25.822371  10.575498
## [10,]   1.9637880  -4.1006574 -25.8223714  84.396967 -56.437726
## [11,]   0.0000000   4.1185384  10.5754984 -56.437726  41.743689
```

```
# Define the smoothing spline estimator function including SE and CI
smoothing_spline_estimator <- function(x, y, lambda, degree = 3, num_knots, boundar
y_knots) {
  # Generate the B-spline basis matrix A using the number of knots
  knots <- seq(boundary_knots[1], boundary_knots[2], length.out = num_knots + 2)[-
c(1, num_knots + 2)]
  A <- bSpline(x, knots = knots, degree = degree, Boundary.knots = boundary_knots,
intercept = FALSE)

  # Calculate the penalty matrix R with the number of knots
  R <- compute_penalty_matrix_dbs_knots(x, num_knots, degree, boundary_knots)

  # Estimate coefficients using the penalized least squares formula
  AtA <- t(A) %*% A
  AtY <- t(A) %*% y

  # Add a small regularization term to the diagonal for numerical stability
  coefficients <- solve(AtA + lambda * R, AtY)

  # Compute the fitted values at observed points
  fitted_values <- A %*% coefficients

  # Calculate residuals and variance (sigma^2)
  residuals <- y - fitted_values
  sigma_squared <- sum(residuals^2) / (length(y) - length(coefficients))

  # Variance of the coefficients
  var_coefficients <- sigma_squared * solve(AtA + lambda * R)
  se_coefficients <- sqrt(diag(var_coefficients))

  # 95% Confidence intervals for coefficients
  ci_lower <- coefficients - 1.96 * se_coefficients
  ci_upper <- coefficients + 1.96 * se_coefficients

  # Return the results
  return(list(
    fitted_values = fitted_values,
    coefficients = coefficients,
    se_coefficients = se_coefficients,
    ci_lower = ci_lower,
    ci_upper = ci_upper,
    spline_function = function(new_x) {
      A_new <- bSpline(new_x, knots = knots, degree = degree, Boundary.knots = boun
dary_knots, intercept = FALSE)
      A_new %*% coefficients
    }
  ))
}


# Parameters for the smoothing spline
lambdas <- c(0.001, 0.01, 0.1)  # Different smoothing parameter values
boundary_knots <- range(train$Age)  # Adjusted boundary knots to the range of x
num_knots <- 8  # Number of internal knots

# Set up plot for comparing different values of lambda
plot(x, y, main = "Smoothing Spline Estimator with Different λ", xlab = "x", ylab =
"y", pch = 16, col = "blue")

# Loop over each lambda, fit the model, and plot the results
colors <- c("red", "green", "purple")
for (i in 1:length(lambdas)) {
  lambda <- lambdas[i]
  result <- smoothing_spline_estimator(x, y, lambda = lambda, degree = 3, num_knots
= num_knots, boundary_knots = boundary_knots)

  # Plot the fitted function for the current lambda
  lines(x, result$fitted_values, col = colors[i], lwd = 2)
}

# Add a legend
legend("topright", legend = c(paste("λ =", lambdas)),
       col = c(colors, "black"), lwd = 2, lty = c(1, 1, 1, 2))
```



Cross-Validation

```r
cross_validation <- function(X, Y, lambdas, num_knots, degree = 3, K = 5, boundary_
knots = NULL) {
  # Determine boundary knots if not provided
  if (is.null(boundary_knots)) {
    boundary_knots <- range(X)
  }

  n <- length(Y)
  fold_indices <- sample(rep(1:K, length.out = n))  # Randomly split data into K fo
lds
  cv_errors <- numeric(length(lambdas))

  for (j in 1:length(lambdas)) {
    lambda <- lambdas[j]
    errors <- numeric(K)

    for (k in 1:K) {
      # Split into train and test sets
      train_idx <- which(fold_indices != k)
      test_idx <- which(fold_indices == k)
      X_train <- X[train_idx]
      Y_train <- Y[train_idx]
      X_test <- X[test_idx]
      Y_test <- Y[test_idx]

      # Fit model on training set
      model <- smoothing_spline_estimator(X_train, Y_train, lambda = lambda, degree
= degree, num_knots = num_knots, boundary_knots = boundary_knots)

      # Predict on the test set
      # Generate the B-spline basis matrix for the test set using the same knots
      knots <- seq(boundary_knots[1], boundary_knots[2], length.out = num_knots +
2)[-c(1, num_knots + 2)]
      B_test <- bSpline(X_test, knots = knots, degree = degree, Boundary.knots = bo
undary_knots, intercept = FALSE)
      Y_pred <- B_test %*% model$coefficients

      # Calculate mean squared error for this fold
      errors[k] <- mean((Y_test - Y_pred)^2)
    }

    # Average CV error for this lambda
    cv_errors[j] <- mean(errors)
  }

  # Return the lambda that gives the minimum CV error
  optimal_lambda <- lambdas[which.min(cv_errors)]
  return(optimal_lambda)
}
```

Generalized cross-validation

```r
generalized_cross_validation <- function(X, Y, lambdas, num_knots, degree = 3, boun
dary_knots = NULL) {
  # Set boundary knots if not provided
  if (is.null(boundary_knots)) {
    boundary_knots <- range(X)
  }

  # Initialize GCV scores
  gcv_scores <- numeric(length(lambdas))

  for (j in 1:length(lambdas)) {
    lambda <- lambdas[j]

    # Fit model with smoothing spline
    model <- smoothing_spline_estimator(X, Y, lambda = lambda, degree = degree, num
_knots = num_knots, boundary_knots = boundary_knots)

    # Calculate residual sum of squares (RSS)
    residual_sum_of_squares <- sum((Y - model$fitted_values)^2)

    # Calculate the smoothing matrix trace (S_lambda)
    knots <- seq(boundary_knots[1], boundary_knots[2], length.out = num_knots + 2)[
-c(1, num_knots + 2)]
    A <- bSpline(X, knots = knots, degree = degree, Boundary.knots = boundary_knot
s, intercept = FALSE)
    AtA <- t(A) %*% A
    R <- compute_penalty_matrix_dbs_knots(X, num_knots, degree, boundary_knots)

    # Calculate smoothing matrix
    S_lambda <- A %*% solve(AtA + lambda * R, t(A))
    trace_S <- sum(diag(S_lambda))

    # Calculate GCV score
    gcv_scores[j] <- residual_sum_of_squares / (length(Y) - trace_S)^2
  }

  # Return the lambda that gives the minimum GCV score
  optimal_lambda <- lambdas[which.min(gcv_scores)]
  return(optimal_lambda)
}
```

(ggplot2)

```
# Define a range of lambda values to test
lambdas <- seq(0.001, 0.1, length.out = 20)

# Set the number of internal knots and degree for the spline
num_knots <- 8
degree <- 3

# Set boundary knots, or let the function automatically set them to the range of X
boundary_knots <- range(train$Age)

# Run cross-validation to find the optimal lambda
cv_optimal_lambda <- cross_validation(train$Age, train$Weight, lambdas = lambdas, n
um_knots = num_knots, degree = degree, boundary_knots = boundary_knots)


# Run generalized cross-validation to find the optimal lambda
gcv_optimal_lambda <- generalized_cross_validation(train$Age, train$Weight, lambdas
= lambdas, num_knots = num_knots, degree = degree, boundary_knots = boundary_knots)

# Run the model with the optimal lambda from cross-validation
cv_model <- smoothing_spline_estimator(train$Age, train$Weight, lambda = cv_optimal
_lambda, degree = 3, num_knots = 8, boundary_knots = boundary_knots)
# Run the model with the optimal lambda from generalized cross-validation (GCV)
gcv_model <- smoothing_spline_estimator(train$Age, train$Weight, lambda = gcv_optim
al_lambda, degree = 3, num_knots = 8, boundary_knots = boundary_knots)

# Generate the B-spline basis matrix for the test data using bSpline
knots <- seq(boundary_knots[1], boundary_knots[2], length.out = num_knots + 2)[-c(
1, num_knots + 2)]
B_test <- bSpline(test$Age, knots = knots, degree = degree, Boundary.knots = bounda
ry_knots, intercept = FALSE)

# Predict using the coefficients
cv_predictions <- B_test %*% cv_model$coefficients
gcv_predictions <- B_test %*% gcv_model$coefficients

# Create a comparison data frame with actual vs predicted values
comparison <- data.frame(
  Actual = test$Weight,
  CV_Predicted = cv_predictions,
  GCV_Predicted = gcv_predictions
)

# Display the first few rows of the comparison data frame
head(comparison)
```

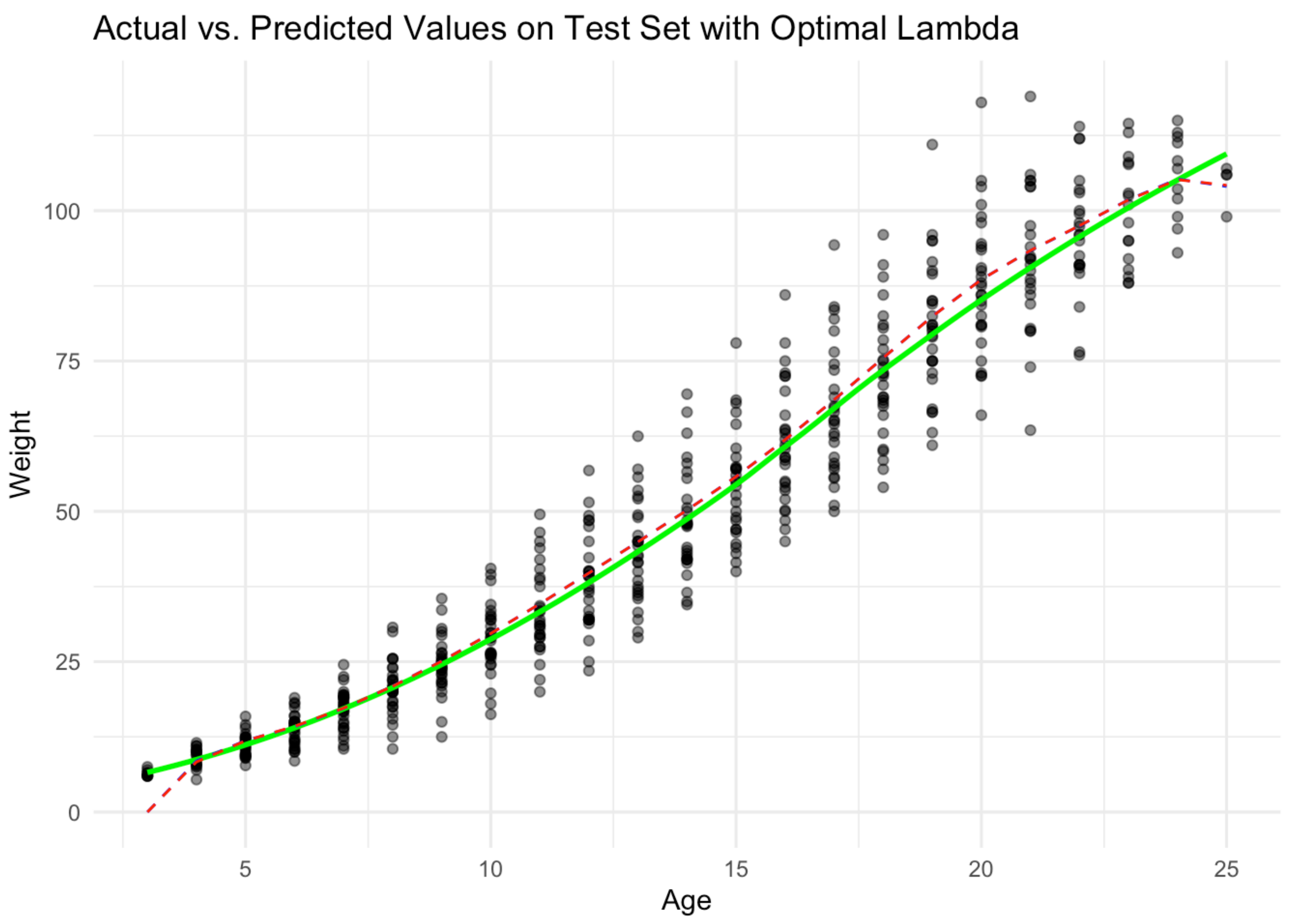
Actual <dbl>	CV_Predicted <dbl>	GCV_Predicted <dbl>

1	9.5	8.497805	8.336532
2	11.5	11.880178	11.848861
3	14.5	14.190589	14.254216
4	15.0	17.245792	17.275701
5	17.5	20.936516	20.911544
6	21.5	25.113504	25.089741

6 rows

```
ggplot() +
  geom_point(data = test, aes(x = Age, y = Weight), alpha = 0.5, color = "black") +
  geom_smooth(data = test, aes(x = Age, y = Weight), method = "loess", color = "gre
en", linetype = "solid", se = FALSE) +
  geom_line(aes(x = test$Age, y = cv_predictions), color = "blue", linetype = "dash
ed") +
  geom_line(aes(x = test$Age, y = gcv_predictions), color = "red", linetype = "dash
ed") +
  labs(title = "Actual vs. Predicted Values on Test Set with Optimal Lambda",
       x = "Age", y = "Weight") +
  theme_minimal()
```

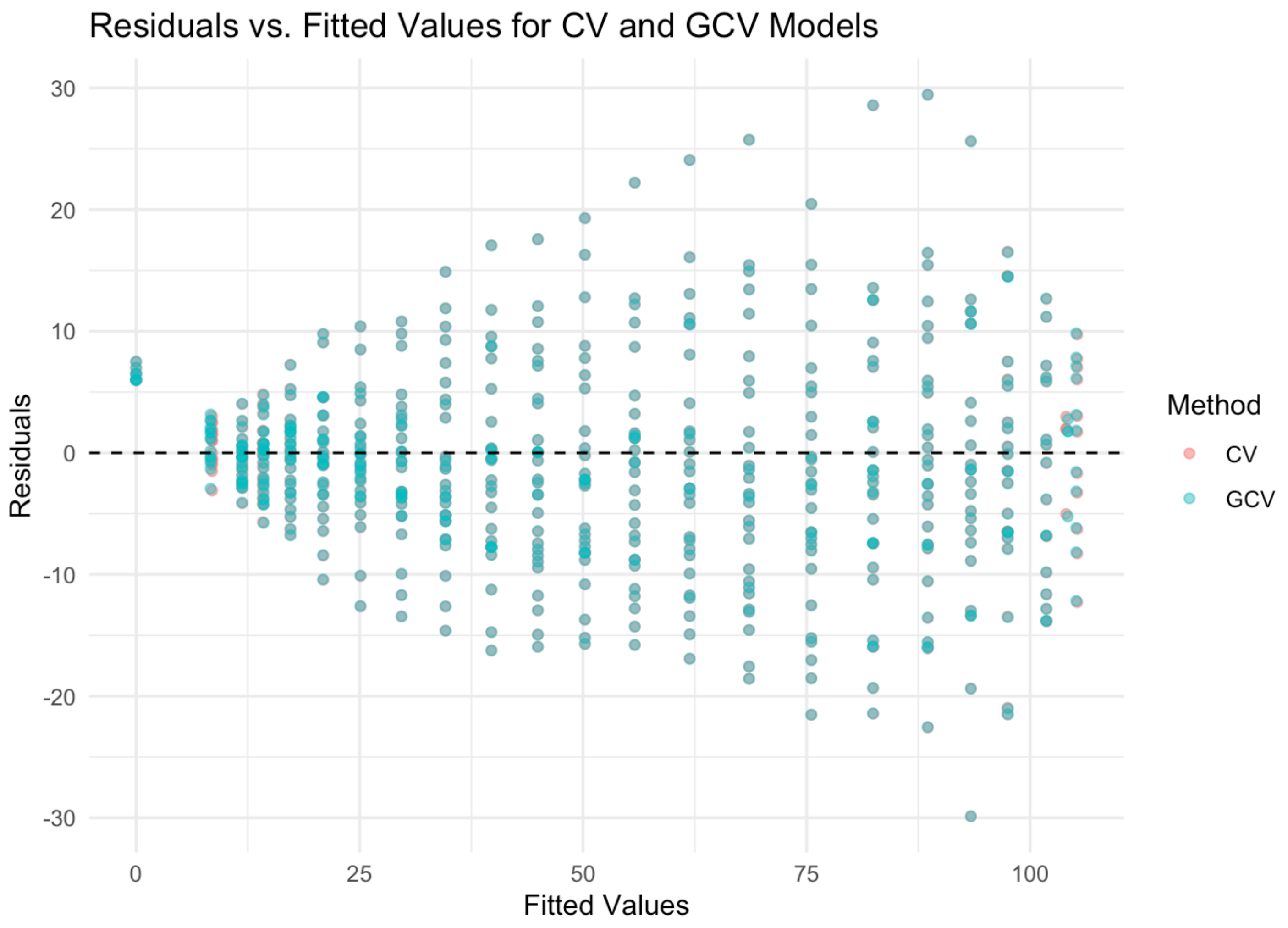
```
## `geom_smooth()` using formula = 'y ~ x'
```



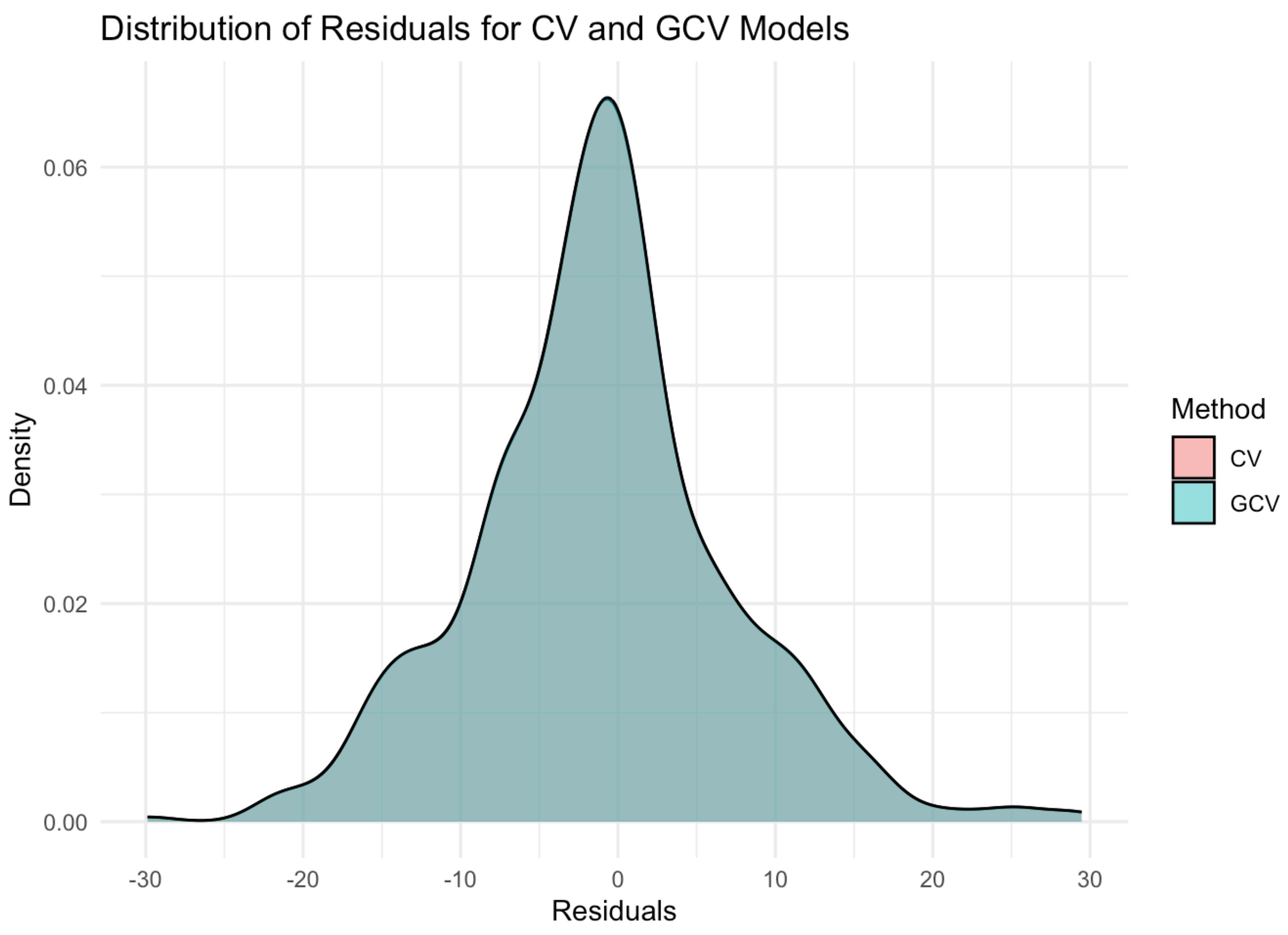
```
# Print summary of coefficients and confidence intervals
summary_cv <- data.frame(Coefficients = cv_model$coefficients,
                         SE = cv_model$se_coefficients,
                         CI_Lower = cv_model$ci_lower,
                         CI_Upper = cv_model$ci_upper)

summary_gcv <- data.frame(Coefficients = gcv_model$coefficients,
                          SE = gcv_model$se_coefficients,
                          CI_Lower = gcv_model$ci_lower,
                          CI_Upper = gcv_model$ci_upper)
```

Residuals vs Fitted values

```
residuals_cv <- test$Weight - cv_predictions
residuals_gcv <- test$Weight - gcv_predictions

residuals_df <- data.frame(
  Fitted = c(cv_predictions, gcv_predictions),
  Residuals = c(residuals_cv, residuals_gcv),
  Method = rep(c("CV", "GCV"), each = length(test$Weight))
)

ggplot(residuals_df, aes(x = Fitted, y = Residuals, color = Method)) +
  geom_point(alpha = 0.5) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  labs(title = "Residuals vs. Fitted Values for CV and GCV Models",
       x = "Fitted Values", y = "Residuals") +
  theme_minimal()
```

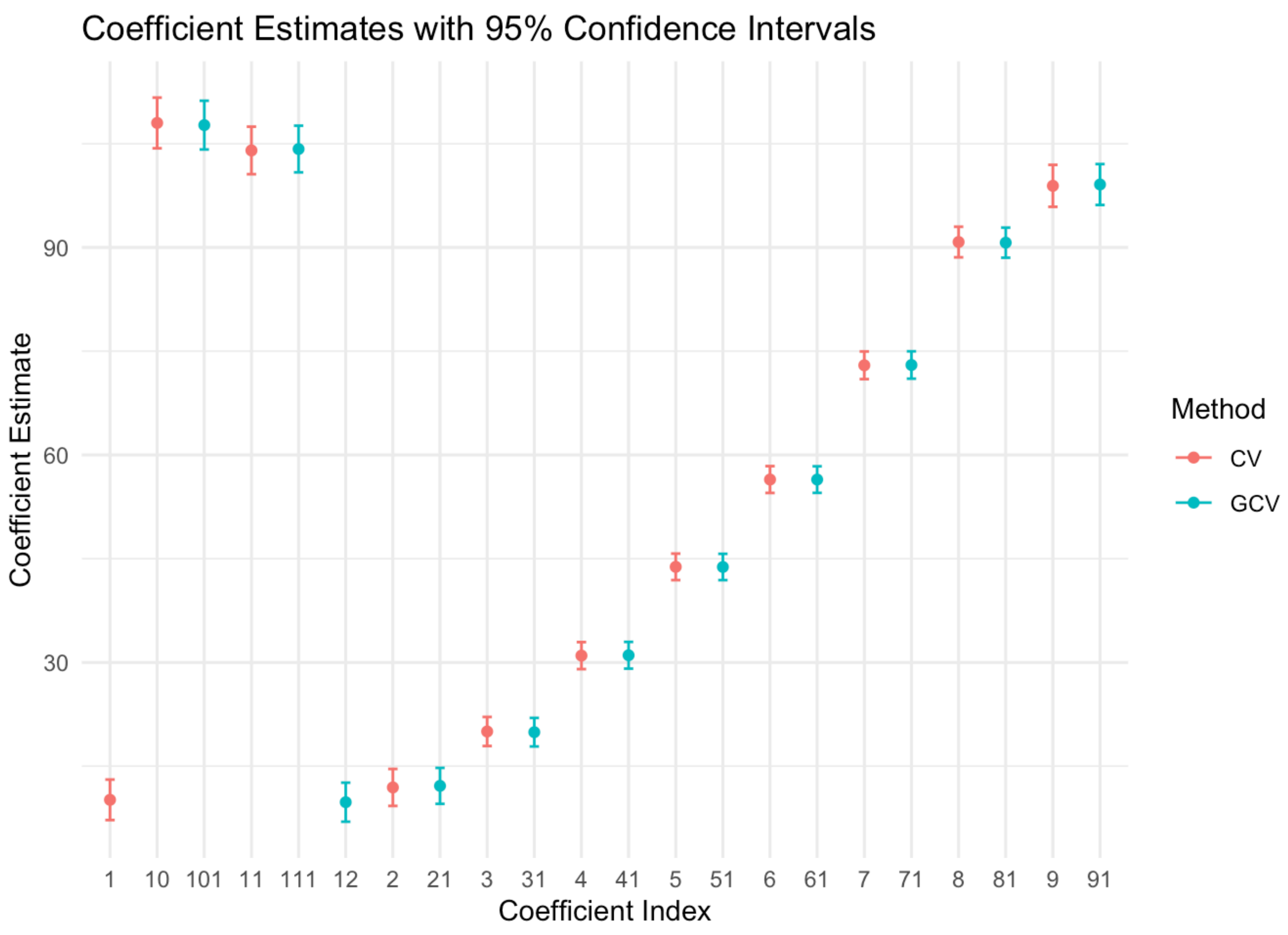


```
ggplot(residuals_df, aes(x = Residuals, fill = Method)) +
  geom_density(alpha = 0.5) +
  labs(title = "Distribution of Residuals for CV and GCV Models",
       x = "Residuals", y = "Density") +
  theme_minimal()
```



Residuals seems to be randomly distibuted around zero and it looks a bit like a normal distribution.

```
summary_cv$Method <- "CV"
summary_gcv$Method <- "GCV"
summary_df <- rbind(summary_cv, summary_gcv)

ggplot(summary_df, aes(x = row.names(summary_df), y = Coefficients, color = Metho
d)) +
  geom_point(position = position_dodge(width = 0.2)) +
  geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, position = posi
tion_dodge(width = 0.2)) +
  labs(title = "Coefficient Estimates with 95% Confidence Intervals",
       x = "Coefficient Index", y = "Coefficient Estimate") +
  theme_minimal()
```



Conclusion

Based on the cross-validation (CV) and generalized cross-validation (GCV) methods, the optimal lambda values were successfully identified to achieve a well-balanced smoothing spline fit. The predicted values align closely with the actual values, as indicated by the **Actual vs. Predicted Values** plot. This shows that the models capture the underlying relationship between Age and Weight reasonably well, with minimal overfitting or underfitting.

Analyzing the residuals was crucial in evaluating model adequacy. The **Residuals vs. Fitted Values** plot indicates that residuals are randomly distributed around zero, suggesting no major patterns or biases in the model. Moreover, the **Residuals Distribution** plot shows a near-normal distribution with minimal skewness or heavy tails. This normal-like distribution of residuals supports the model assumptions, indicating that both the CV and GCV models are reliable for this data without significant unexplained variation.

The **Coefficient Estimates with Confidence Intervals** plot provides insight into the stability of the model's coefficients. Confidence intervals around the coefficients are relatively narrow, suggesting precise and consistent estimates. The similarity in coefficient values between the CV and GCV models further reinforces that both regularization methods yield robust and stable parameter estimates.