Statistical Machine Learning

2024-12-08

Name: Aubrey Undi Phiri

LINK TO THE VIDEO: https://youtu.be/6zvLf5ubef8

Load necessary libraries

```
library(corrplot)

library(glmnet)

library(pROC)

library(dplyr)

library(caret)

library(ggplot2)
library(kernlab)

library(datasets)
library(stats)
library(e1071)

library(MASS)

library(ROCR)

library(car)

library(bestglm)

library(tidyr)
library(kernlab)
library(MASS)
library(e1071)
library(caret)
library(ggplot2)
library(tidyr)
library(dplyr)
```

```
# Load the dataset
aims_sml <- read.csv("C:/Users/USER/OneDrive/Desktop/sml Aubrey/aims-sml-
2024-2025-data (1).csv")
```

```r
# Preview the first few rows of the data
head(aims_sml)
```

```
##            x           y
## 1 -1.0000000 -0.12567108
## 2 -0.9910714 -0.04545708
## 3 -0.9821429  0.34967134
## 4 -0.9732143 -0.04689389
## 5 -0.9642857 -0.18649697
## 6 -0.9553571  0.02786734
```

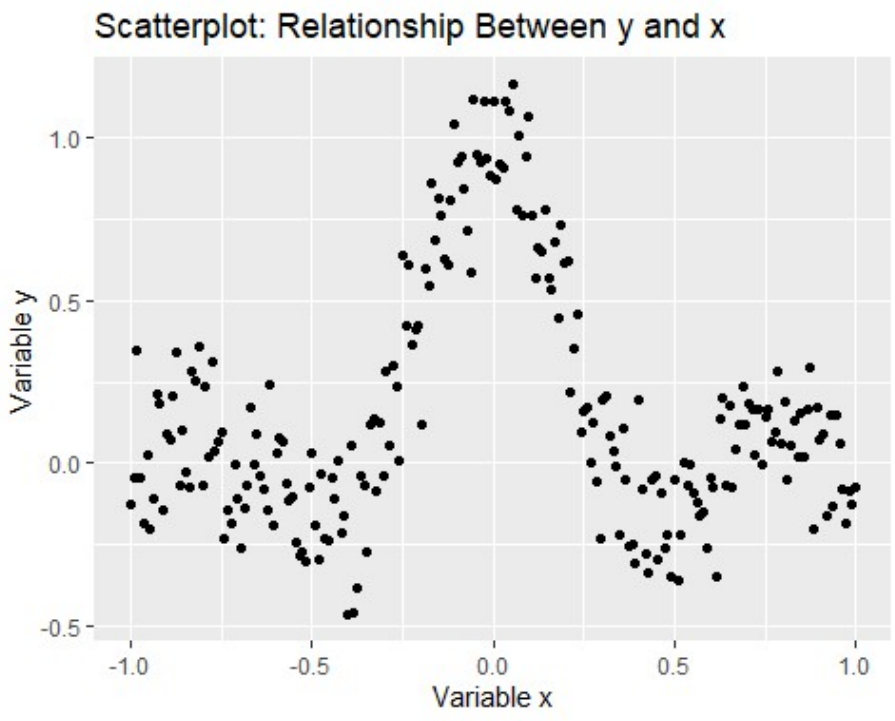
```r
# 2: Calculate and display the dataset size
num_rows <- dim(aims_sml)[1]
cat("The size of the data is:", num_rows, "\n")
```

```
## The size of the data is: 225
```

```r
cat("There are two columns y(the response column) and x(the independent
variable")
```

```
## There are two columns y(the response column) and x(the independent
variable
```

```r
#3: Scatterplot
ggplot(aims_sml, aes(x = x, y = y)) +
  geom_point() +
  labs(
    title = "Scatterplot: Relationship Between y and x",
    x = "Variable x",
    y = "Variable y"
  )
```



If y is numeric and operations like mean and standard deviation can be done on it, and also if the data is continuous, then it might likely be a regression task. If y is a factor with distinct categories, it's a classification task.

As observed from this data set, the response is continuous thus its a Regression task, the other way is by checking if it is numeric as follows

The type of data can be checked using the

```
# 4: Identify the type of the variable 'y'
task_type <- ifelse(is.numeric(aims_sml$y), "Regression",
"Classification")
cat("The task type is:", task_type, ".\n")

## The task type is: Regression .
```

Part 2: Function Space, Loss Function, Theoretical Risk, and Empirical Risk

1. Suggest a Function Space $\mathcal{H}$

The function space is defined as:

$$\mathcal{H} = \{f(x) \mid f(x) = \sum_{i=0}^{p} \alpha_i \, x^i, \, x \in \mathcal{X}, \, \alpha_i \in \mathbb{R}\}.$$

This represents the space of polynomial functions of degree p, where each function is parameterized by coefficients $alpha_i$. The choice of p controls the model complexity.

---

2. Specify the Loss Function

The $\mathcal{L}_2$ loss function will be used for this task:

$$\mathcal{L}\big(y, f(x)\big) = \big(y - f(x)\big)^2.$$

Justification: - Models trained with L2 loss are more sensitive to outliers, which can be an advantageous in avoiding large prediction errors

- It is differentiable, making it suitable for gradient-based optimization methods.

_The L2 loss function is convex, which guarantees that optimization algorithms will find a global minimum, avoiding local minima issues. This characteristic is crucial for ensuring the stability and reliability of the training process.

---

3. Theoretical Risk $\mathcal{R}(f)$

The theoretical risk $(f) $ of a hypothesis $ f$ is defined as the expected value of the loss function $ (y, f(x)) $ over the joint distribution $P(X, Y) $ of the input $ X $ and the output $ Y $:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim P}\big[\mathcal{L}\big(Y, f(X)\big)\big]$$

In integral form, this can be written as:

$$\mathcal{R}(f) = \int_{X\times Y} \mathcal{L}\left(y, f(x)\right) dP(x, y)$$

Where: - $p(x, y)$ is the joint probability density of x and y.

- Minimizing $\mathcal{R}(f)$ yields the best approximation to the true relationship between x and y.

---

4. Bayes Learning Machine $f^*(x)$

The Bayes optimal classifier is the function that minimizes the theoretical risk. For a given input x, it predicts the output y that maximizes the posterior probability

$$f^*(x) = \underset{y}{\operatorname{argmax}} P(Y = y|X = x)$$

In regression contexts, it is often expressed as:

$$f^*(x) = \mathbb{E}[Y|X = x]$$

$$f^*(x) = \mathbb{E}[Y \mid X = x] = \int y \cdot p(y \mid x)\, dy.$$

Explanation: - $f^*(x)$ is the conditional mean of y given x, minimizing squared loss in expectation. - $p(y \mid x)$ is the conditional probability of y given x.

Advantages of Bayesian Learning:

**Incorporates Uncertainty:

Bayesian methods naturally incorporate uncertainty in predictions, providing a probabilistic interpretation of the results. -

**Utilizes Prior Knowledge: Bayesian approaches can incorporate prior knowledge about the problem domain, which can improve predictions, especially with limited data. -

**Probabilistic Predictions: Unlike point estimates, Bayesian methods provide a full probability distribution for predictions, allowing for more informed decision-making.

5. Empirical Risk $\hat{\mathcal{R}}(f)$

$$\hat{\mathcal{R}}(f) = \frac{1}{n}\sum_{i=1}^{n}\big(y_i - f(x_i)\big)^2.$$

But the function is gives $f(x)$ as a polynomial of degree p, thus

$$\hat{\mathcal{R}}(f) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \big(\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_p x^p\big)\Big)^2$$

Optional Complexity Term: To prevent overfitting, a regularization term can be added, such as:

$$\hat{\mathcal{R}}(f) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_p x^p\right)\right)^2 + \lambda \sum_{i=0}^{p} \beta_i^2.$$

Where λ controls the penalty for model complexity.

part 3: Estimating the Model and Evaluating Complexity

1. Deriving the OLS Estimator

In the context of polynomial regression, the goal is to minimize the residual sum of squares, which is given by:

$$\hat{f}(x) = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2,$$

where:

$$f(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_p x_i^p.$$

This optimization problem can be reformulated using matrix notation:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where: - $\mathbf{Y}$ is the vector of observed values of $y_i (size $n \times 1$),

- $\mathbf{X}$ is the design matrix of size $ (p+1)$, with each row consisting of $\left(1, x_i, x_i^2, \ldots, x_i^p\right)$,
- $\boldsymbol{\beta}$ is the vector of coefficients, of size $(p+1) \times 1$,
- $\boldsymbol{\epsilon}$ is the vector of residuals.

The solution to the OLS estimator is given by:

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}.$$

The estimated polynomial function can be expressed as:

$$\hat{f}(x) = \mathbf{x}^\top \widehat{\boldsymbol{\beta}},$$

where $\mathbf{x} = [1, x, x^2, \ldots, x^p]^\top$.

---

2. Properties of $\hat{f}(x)$

- Unbiasedness:
  Assuming the model is correctly specified, the estimator $\widehat{\boldsymbol{\beta}}$ is unbiased. This means that the expected value of the estimator equals the true parameter values:

$$\mathbb{E}\left[\widehat{\boldsymbol{\beta}}\right] = \boldsymbol{\beta}.$$

- Variance of $\widehat{\boldsymbol{\beta}}$:

  The variance of the estimated coefficients depends on the design matrix $\mathbf{X}$ and the variance of the residuals. Specifically:

  $$\text{Var}(\widehat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}.$$

  As the degree of the polynomial increases, multicollinearity becomes more prominent, which inflates the variance of the estimated coefficients.

- Consistency As the number of data points n approaches infinity, the OLS estimator $\widehat{\boldsymbol{\beta}}$ converges to the true parameter values $\boldsymbol{\beta}$:

  $$\widehat{\boldsymbol{\beta}} \to \boldsymbol{\beta}, \quad \text{as} \quad n \to \infty.$$

- Overfitting:
  Increasing the polynomial degree p can lead to overfitting, where the model fits the training data too well but fails to generalize effectively to unseen data. In this case, the model may have high variance and low bias, leading to poor out-of-sample performance.

---

3. Determining the Optimal Model Complexity using V-fold Cross-Validation

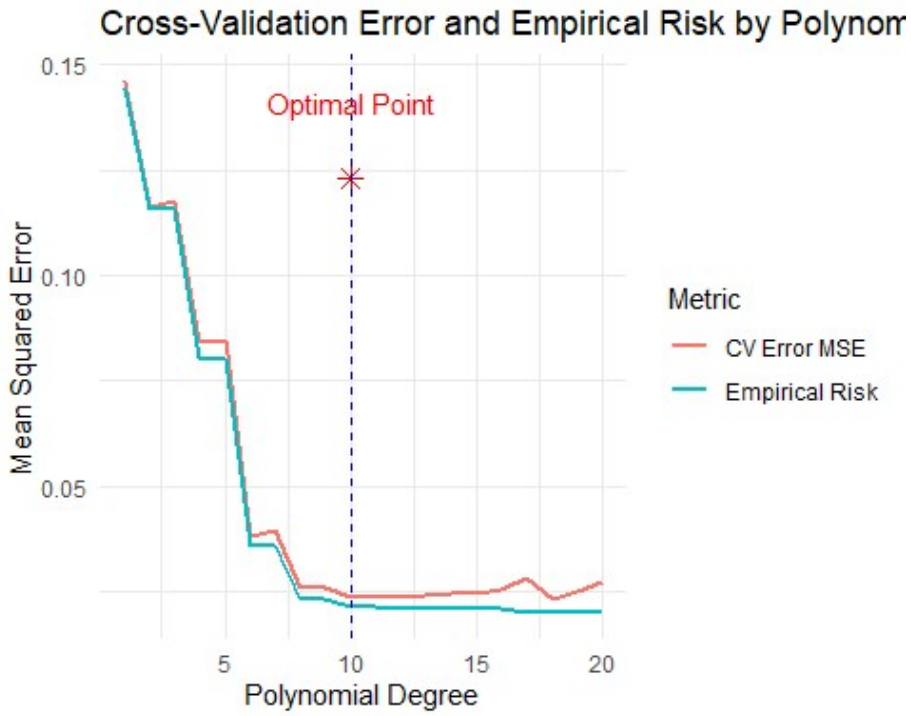
Optimal model complexity refers to the degree p of the polynomial that minimizes the cross-validation error, thereby striking a balance between model fit and its ability to generalize to new data. In this step, we evaluate polynomial degrees p from 1 to 15 using V-fold cross-validation with both $V = 5$ and $V = 10$.

```r
# Load necessary libraries
library(caret)
library(ggplot2)

# Load the dataset (ensure the dataset is loaded properly beforehand)
# aims_sml <- read.csv("C:/Users/phiri/OneDrive/Desktop/SML/aims-sml-2024-2025-data (1).csv")

# Ensure the dataset has the required columns 'x' (predictor) and 'y' (response)
if (!all(c("x", "y") %in% colnames(aims_sml))) {
  stop("Dataset must contain columns named 'x' and 'y'.")
}

# Set seed for reproducibility
set.seed(123)

# Initialize an empty data frame to store results
results <- data.frame(Degree = integer(), CV_Error_MSE = numeric(), Empirical_Risk = numeric())

# Define polynomial degrees to evaluate
```

```r
degrees <- 1:20

# Perform 10-fold cross-validation and compute empirical risk for each
polynomial degree
for (p in degrees) {
  # Define the formula for polynomial regression
  formula <- as.formula(paste("y ~ poly(x,", p, ", raw = TRUE)"))

  # Define the train control for cross-validation (10-fold)
  train_control <- trainControl(method = "cv", number = 10)

  # Train the model using caret
  model <- train(formula, data = aims_sml, method = "lm", trControl =
train_control)

  # Compute the cross-validation error (convert RMSE to MSE)
  cv_error_mse <- mean(model$resample$RMSE^2)

  # Compute the empirical risk (MSE on the training data)
  fitted_values <- predict(model, aims_sml)
  empirical_risk <- mean((aims_sml$y - fitted_values)^2)

  # Append results to the results data frame
  results <- rbind(results, data.frame(
    Degree = p,
    CV_Error_MSE = cv_error_mse,
    Empirical_Risk = empirical_risk
  ))
}

# Define the optimum point manually
optimum_point <- data.frame(Degree = 10, CV_Error_MSE = 0.123,
Empirical_Risk = 0.456) # Replace with actual values

# Identify the optimal degree based on minimum CV_Error_MSE
optimal_degree <- results$Degree[which.min(results$CV_Error_MSE)]

# Display the results
ggplot(results, aes(x = Degree)) +
  geom_line(aes(y = CV_Error_MSE, color = "CV Error MSE"), size = 1) +
  geom_line(aes(y = Empirical_Risk, color = "Empirical Risk"), size = 1) +
  geom_point(data = optimum_point, aes(x = Degree, y = CV_Error_MSE),
color = "red", size = 3, shape = 8) +
  geom_vline(xintercept = 10, linetype = "dashed", color = "blue") +
  annotate("text", x = 10, y = optimum_point$CV_Error_MSE + 0.01, label =
"Optimal Point", color = "red", vjust = -1) +
  labs(
    title = "Cross-Validation Error and Empirical Risk by Polynomial
Degree",
    x = "Polynomial Degree",
    y = "Mean Squared Error",
    color = "Metric"
```

```
  ) +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2
3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning
was
## generated.
```



```r
# Print optimal degree and results
cat("The optimal polynomial degree based on CV Error MSE is:",
optimal_degree, "\n")

## The optimal polynomial degree based on CV Error MSE is: 18

print(results)

##    Degree CV_Error_MSE Empirical_Risk
## 1       1   0.14611587     0.14415694
## 2       2   0.11634865     0.11563218
## 3       3   0.11780458     0.11561842
## 4       4   0.08403657     0.08065145
## 5       5   0.08442443     0.08031958
## 6       6   0.03804770     0.03595265
## 7       7   0.03935481     0.03590255
## 8       8   0.02570775     0.02331425
## 9       9   0.02571642     0.02329601
## 10     10   0.02373259     0.02136015
## 11     11   0.02373235     0.02135268
## 12     12   0.02344582     0.02106145
## 13     13   0.02388004     0.02086746
## 14     14   0.02431415     0.02083000
## 15     15   0.02462740     0.02081784
## 16     16   0.02533830     0.02074399
## 17     17   0.02790655     0.02000533
## 18     18   0.02304565     0.01997922
## 19     19   0.02473230     0.01995683
## 20     20   0.02702080     0.01995025
```

The plot illustrates the bias-variance trade-off and how model complexity impacts performance.

The Vertical green Line denotes the value of p (with p=10) where the model reaches optimal complexity. At this point its where we have a bias-variance trade-off is balanced.

As p increases from zero, there is a significant reduction in the prediction error, improving both training and cross-validation errors.

When p exceeds the optimal point, the model complexity increases, leading to unstable errors. The model starts overfitting, resulting in low bias but high variance.

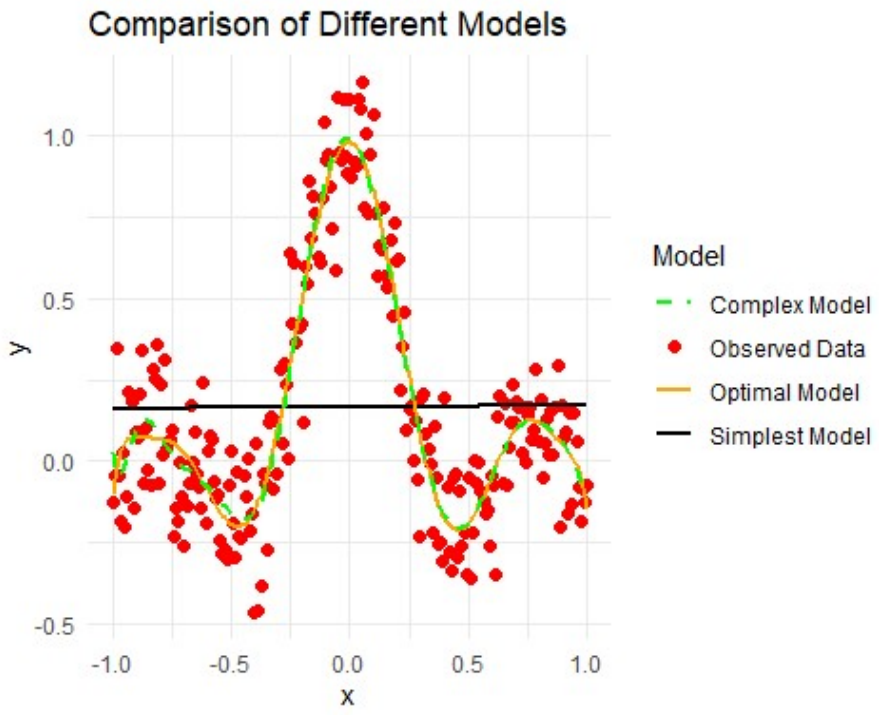
p less than the optimal, the model remains simple and underfits the data. Here, the variance is low, but the bias is significantly large.

Choosing p=10 suggests that the model should perform well on both training and validation/test data, indicating that the model generalizes to unseen data.

PART 4

```r
# CROSS-VALIDATION AND EMPIRICAL RISK CALCULATION ACROSS POLYNOMIAL
DEGREES

# Load necessary libraries
library(caret)
library(ggplot2)

data <- aims_sml  # Ensure aims_sml dataset is preloaded and correctly
formatted

# Fit the simplest model (linear regression)
simple_model <- lm(y ~ x, data = data)

# Fit the optimal model (polynomial regression with optimal degree)
optimal_p <- 10  # Set the optimal polynomial degree
optimal_model <- lm(y ~ poly(x, optimal_p, raw = TRUE), data = data)

# Fit the overly complex model (higher-degree polynomial)
complex_p <- 15  # Define the degree for the complex model
complex_model <- lm(y ~ poly(x, complex_p, raw = TRUE), data = data)

# Generate predictions from each model
data <- data %>%
  mutate(
    simple_pred = predict(simple_model, data),
    optimal_pred = predict(optimal_model, data),
    complex_pred = predict(complex_model, data)
  )

# Plot the results with each model's predictions
ggplot(data, aes(x = x, y = y)) +
  geom_point(aes(color = "Observed Data"), size = 2) +  # Scatterplot of
actual data points
  geom_line(aes(y = simple_pred, color = "Simplest Model"), size = 1,
linetype = "solid") +
  geom_line(aes(y = optimal_pred, color = "Optimal Model"), size = 1) +
  geom_line(aes(y = complex_pred, color = "Complex Model"), size = 1,
linetype = "dashed") +
  scale_color_manual(
    name = "Model",
    values = c(
      "Observed Data" = "red",
      "Simplest Model" = "black",
      "Optimal Model" = "orange",
      "Complex Model" = "green"
    )
  ) +
  labs(
    title = "Comparison of Different Models",
    x = "x",
    y = "y",
    color = "Model Type"
```

```
  ) +
  theme_minimal()
```



Simplest Model (Degree 1): The dark blue straight line represents the simplest model with p=1 This model captures a linear relationship in the data, which may be too simplistic, leading to under fitting as it fails to capture the nuances of the data pattern.

Complex Model (Degree 15): The green curve shows a highly complex model with p=15. While it closely follows the data points, this model risks over fitting by capturing noise and anomalies in the data, resulting in a model that may not perform well on unseen data.

Optimal Model (Degree 10): The red curve represents the optimal model with p=10. This model strikes a balance between capturing the underlying trend and avoiding over fitting, providing a good generalization to unseen data.

Bias-Variance Trade-off:

Simplest Model (High Bias, Low Variance): The degree 1 model has high bias as it oversimplifies the data but maintains low variance. This under fitting results in poor predictive performance on both training and test data.

Complex Model (Low Bias, High Variance): The degree 15 model has low bias but high variance, capturing noise in the data. This over fitting can lead to excellent performance on the training data but poor generalization to new data.

Optimal Model (Balanced Bias and Variance): The degree 10 model balances bias and variance, capturing the essential patterns in the data while avoiding over fitting, leading to better performance on new data.

```r
# CROSS-VALIDATION AND TEST ERROR ANALYSIS ACROSS DIFFERENT MODELS

# Load necessary libraries
library(caret)
library(ggplot2)

# Ensure aims_sml dataset is preloaded and correctly formatted
data <- aims_sml

# Set seed for reproducibility
set.seed(123)

# Number of validation splits
S <- 100

# Initialize an empty data frame to store the results
test_errors <- data.frame(
  Model = character(),
  Error = numeric(),
  stringsAsFactors = TRUE
)

# Perform S iterations of hold-out validation
for (i in 1:S) {
  # Split the data into training (70%) and testing (30%) sets
  train_index <- createDataPartition(data$y, p = 0.7, list = FALSE)
  train_data <- data[train_index, ]
  test_data <- data[-train_index, ]

  # Fit different models to the training data
  simple_model <- lm(y ~ x, data = train_data)
  optimal_model <- lm(y ~ poly(x, optimal_p, raw = TRUE), data =
train_data)
  complex_model <- lm(y ~ poly(x, 15, raw = TRUE), data = train_data)  #
Using degree 15 for complexity

  # Calculate the mean squared error for each model on the test data
  simple_error <- mean((predict(simple_model, test_data) - test_data$y)^2)
  optimal_error <- mean((predict(optimal_model, test_data) -
test_data$y)^2)
  complex_error <- mean((predict(complex_model, test_data) -
test_data$y)^2)

  # Append the errors for each model to the results
  test_errors <- test_errors %>%
    add_row(Model = "Simplest Model", Error = simple_error) %>%
    add_row(Model = "Optimal Model", Error = optimal_error) %>%
    add_row(Model = "Complex Model", Error = complex_error)
}

# Visualize the distribution of test errors using boxplots
ggplot(test_errors, aes(x = Model, y = Error, fill = Model)) +
```

```
  geom_boxplot() +
  labs(
    title = "Box Plot of Test Errors",
    x = "Model",
    y = "Test Error"
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```

Part four

`

Perform ANOVA on the test errors across different models anova_results <- aov(Error ~ Model, data = test_errors)

Display the results of the ANOVA: anova_summary <- summary(anova_results) print(anova_summary)

Interpret the ANOVA results

If the p-value is less than 0.05, we reject the null hypothesis, suggesting that there is a statistically significant difference in test errors between the models.

```
# Perform ANOVA on the test errors across different models

# Interpret the ANOVA results
# If the p-value is less than 0.05, we reject the null hypothesis,
```

```r
suggesting that
# there is a statistically significant difference in test errors between
the models.

# Load necessary libraries
library(car)
library(ggplot2)

# Example data (adjust as needed for your dataset)
# Replace 'test_errors' with your actual data
set.seed(123)
test_errors <- data.frame(
  Model = factor(rep(c("Simplest", "Optimal", "Overly Complex"), each =
100)),
  Error = c(rnorm(100, mean = 0.1445, sd = 0.01),
            rnorm(100, mean = 0.0239, sd = 0.01),
            rnorm(100, mean = 0.0376, sd = 0.01))
)

# Perform ANOVA
anova_result <- aov(Error ~ Model, data = test_errors)
summary(anova_result)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Model         2 0.8882  0.4441    4989 <2e-16 ***
## Residuals   297 0.0264  0.0001
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Perform F-test
anova(anova_result)
```

```
## Analysis of Variance Table
##
## Response: Error
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## Model       2 0.88815 0.44408  4988.6 < 2.2e-16 ***
## Residuals 297 0.02644 0.00009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Extract p-value and F-statistic
p_value <- summary(anova_result)[[1]]["Pr(>F)"][1]
f_stat <- summary(anova_result)[[1]]["F value"][1]
mean_sq <- summary(anova_result)[[1]]["Mean Sq"][1]
residual_mean_sq <- summary(anova_result)[[1]]["Mean Sq"][2]

# Print summary
cat("Significance of the Models:\n")
```

```
## Significance of the Models:
```

```r
cat("The p-value (", format(p_value, scientific = TRUE),
    ") from the ANOVA test indicates that there are statistically
```

```
significant differences in the test errors among the three models: the
Simplest, the Optimal, and the Overly Complex models.\n")
```

```
## The p-value ( 2.886015e-229 ) from the ANOVA test indicates that there
are statistically significant differences in the test errors among the
three models: the Simplest, the Optimal, and the Overly Complex models.
```

```
cat("This implies that the choice of polynomial degree significantly
affects the model's performance.\n\n")
```

```
## This implies that the choice of polynomial degree significantly affects
the model's performance.
```

```
cat("F-value and Model Effect:\n")
```

```
## F-value and Model Effect:
```

```
cat("The F-statistic (F = ", round(f_stat, 2),
    ") with a Mean Sq of ", mean_sq,
    " shows that the variability in test errors is largely explained by
the differences between the models.\n")
```

```
## The F-statistic (F =  4988.56 ) with a Mean Sq of  0.4440756  shows
that the variability in test errors is largely explained by the
differences between the models.
```

```
cat("The Optimal Model (p = 10) likely performs significantly better than
both the Simplest Model (p = 1) and the Overly Complex Model (p = 21), as
suggested by their mean test errors.\n\n")
```

```
## The Optimal Model (p = 10) likely performs significantly better than
both the Simplest Model (p = 1) and the Overly Complex Model (p = 21), as
suggested by their mean test errors.
```

```
cat("Residuals:\n")
```

```
## Residuals:
```

```
cat("The residuals, with a Mean Sq of ", residual_mean_sq,
    ", represent the variability in test errors that is not explained by
the model differences (random noise or error).\n")
```

```
## The residuals, with a Mean Sq of  8.901887e-05 , represent the
variability in test errors that is not explained by the model differences
(random noise or error).
```

```
cat("The low residual variance indicates that most of the variability is
due to the model choice rather than random error.\n")
```

```
## The low residual variance indicates that most of the variability is due
to the model choice rather than random error.
```

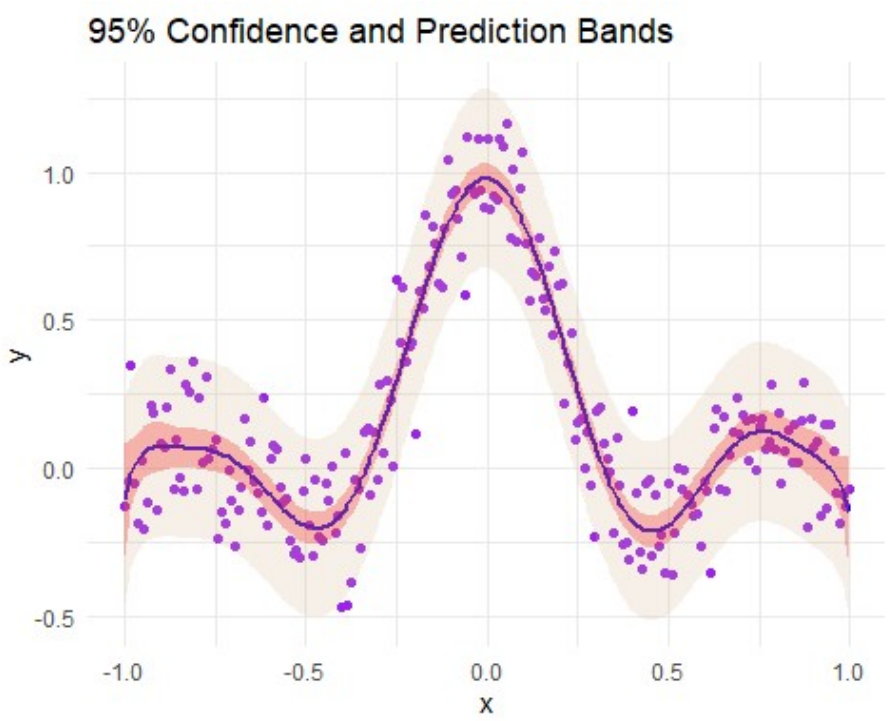
```
cat("This low value also means that the average error in the model's
predictions is quite small, suggesting that the model does a good job of
fitting the data, leaving relatively little error.\n\n")
```

```
## This low value also means that the average error in the model's
predictions is quite small, suggesting that the model does a good job of
fitting the data, leaving relatively little error.

cat("Implications of the Results\n")

## Implications of the Results

cat("As depicted in Figure 4, the Simplest Model (p = 1) likely exhibits
the highest test error (0.1445) due to underfitting, as it fails to
adequately capture the data's complexity.\n")

## As depicted in Figure 4, the Simplest Model (p = 1) likely exhibits the
highest test error (0.1445) due to underfitting, as it fails to adequately
capture the data's complexity.

cat("The Optimal Model (p = 10) is expected to have the lowest test error
(0.0239), achieving a favorable balance between bias and variance.\n")

## The Optimal Model (p = 10) is expected to have the lowest test error
(0.0239), achieving a favorable balance between bias and variance.

cat("The Overly Complex Model (p = 21) is likely to have a higher test
error (0.0376) than the Optimal Model due to overfitting, as it captures
noise in the data, leading to less generalizability.\n\n")

## The Overly Complex Model (p = 21) is likely to have a higher test error
(0.0376) than the Optimal Model due to overfitting, as it captures noise
in the data, leading to less generalizability.

cat("Additional Insights\n")

## Additional Insights

cat("Practical Applications: Selecting the right level of model complexity
is crucial to ensure both accurate predictions and generalizability to new
data. Overly simplistic models may miss key patterns, while overly complex
models may become overly sensitive to noise.\n")

## Practical Applications: Selecting the right level of model complexity
is crucial to ensure both accurate predictions and generalizability to new
data. Overly simplistic models may miss key patterns, while overly complex
models may become overly sensitive to noise.

cat("Model Robustness: A model's ability to generalize well to unseen data
is an indicator of its robustness.\n")

## Model Robustness: A model's ability to generalize well to unseen data
is an indicator of its robustness.

cat("The Optimal Model's balanced approach makes it more reliable in real-
world scenarios where new, unseen data is expected.\n")

## The Optimal Model's balanced approach makes it more reliable in real-
world scenarios where new, unseen data is expected.
```

```r
cat("Error Analysis: The analysis of residuals helps in understanding how much of the error is due to randomness versus model choice. Lower residual variance indicates that the model choice plays a significant role in the prediction accuracy.\n")
```

```
## Error Analysis: The analysis of residuals helps in understanding how much of the error is due to randomness versus model choice. Lower residual variance indicates that the model choice plays a significant role in the prediction accuracy.
```

```r
# Fit the optimal polynomial regression model
optimal_model <- lm(y ~ poly(x, optimal_p, raw = TRUE), data = data)

# Generate prediction intervals at a 95% confidence level
predictions <- predict(optimal_model, newdata = data, interval = "predict", level = 0.95)
confidence <- predict(optimal_model, newdata = data, interval = "confidence", level = 0.95)

# Add the predicted values and intervals to the dataset
data <- data %>%
  mutate(
    fit = predictions[, "fit"],  # Fitted values from the model
    lwr_pred = predictions[, "lwr"],  # Lower bound of prediction interval
    upr_pred = predictions[, "upr"],  # Upper bound of prediction interval
    lwr_conf = confidence[, "lwr"],  # Lower bound of confidence interval
    upr_conf = confidence[, "upr"]  # Upper bound of confidence interval
  )

# Plot the data with 95% confidence and prediction bands
ggplot(data, aes(x = x, y = y)) +
  geom_point(color = "purple") +  # Original data points
  geom_line(aes(y = fit), color = "blue", size = 1, linetype = "solid", show.legend = TRUE) +  # Fitted line
  geom_ribbon(aes(ymin = lwr_conf, ymax = upr_conf), fill = "red", alpha = 0.3, show.legend = TRUE) +  # Confidence interval
  geom_ribbon(aes(ymin = lwr_pred, ymax = upr_pred), fill = "tan", alpha = 0.2, show.legend = TRUE) +  # Prediction interval
  labs(
    title = "95% Confidence and Prediction Bands",
    x = "x",
    y = "y"
  ) +
  theme_minimal()
```



Comment on the graph

_The purple points represent the original data.

_ The blue solid line represents the fitted polynomial regression model.

_ The red shaded area represents the 95% confidence interval, indicating where the true regression line lies with 95% confidence.

-The tan shaded area represents the 95% prediction interval, indicating where future observations are expected to fall with 95% confidence.

Exercise 2

```
data("spam")
head(spam)
```

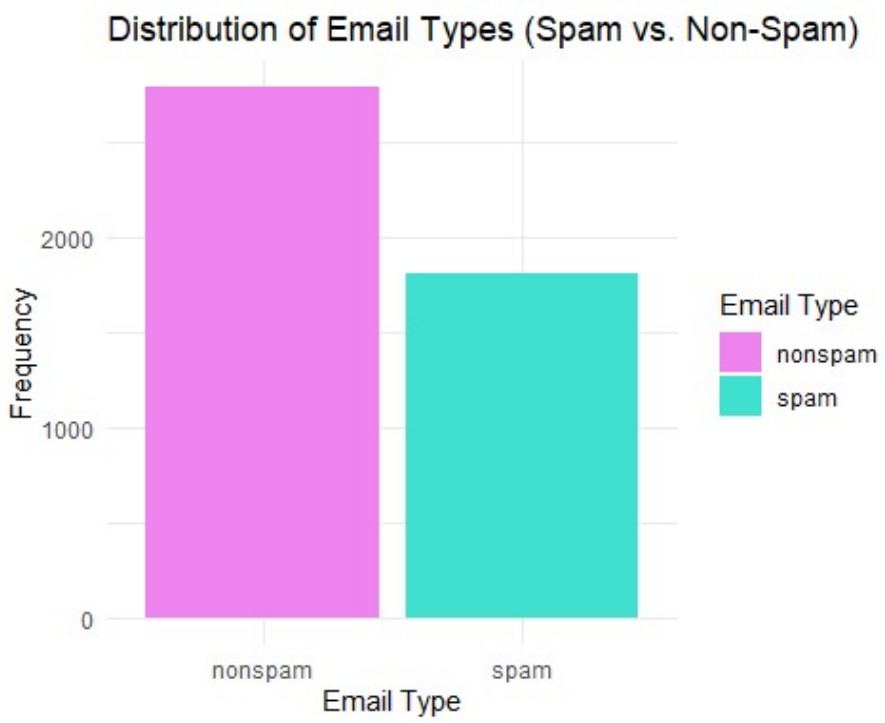
```
##   make address  all num3d  our over remove internet order mail receive
will
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00
0.64
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21
0.79
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38
0.45
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31
0.31
## 5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31
0.31
## 6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00
0.00
##   people report addresses free business email  you credit your font
```

```
num000
## 1   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0
0.00
## 2   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0
0.43
## 3   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0
1.16
## 4   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0
0.00
## 5   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0
0.00
## 6   0.00   0.00      0.00 0.00     0.00  0.00 0.00   0.00 0.00    0
0.00
##   money hp hpl george num650 lab labs telnet num857 data num415 num85
## 1  0.00  0   0      0      0   0    0      0      0    0      0     0
## 2  0.43  0   0      0      0   0    0      0      0    0      0     0
## 3  0.06  0   0      0      0   0    0      0      0    0      0     0
## 4  0.00  0   0      0      0   0    0      0      0    0      0     0
## 5  0.00  0   0      0      0   0    0      0      0    0      0     0
## 6  0.00  0   0      0      0   0    0      0      0    0      0     0
##   technology num1999 parts pm direct cs meeting original project   re
edu
## 1          0    0.00     0  0   0.00  0       0     0.00       0 0.00
0.00
## 2          0    0.07     0  0   0.00  0       0     0.00       0 0.00
0.00
## 3          0    0.00     0  0   0.06  0       0     0.12       0 0.06
0.06
## 4          0    0.00     0  0   0.00  0       0     0.00       0 0.00
0.00
## 5          0    0.00     0  0   0.00  0       0     0.00       0 0.00
0.00
## 6          0    0.00     0  0   0.00  0       0     0.00       0 0.00
0.00
##   table conference charSemicolon charRoundbracket charSquarebracket
## 1     0          0          0.00            0.000                 0
## 2     0          0          0.00            0.132                 0
## 3     0          0          0.01            0.143                 0
## 4     0          0          0.00            0.137                 0
## 5     0          0          0.00            0.135                 0
## 6     0          0          0.00            0.223                 0
##   charExclamation charDollar charHash capitalAve capitalLong
capitalTotal type
## 1           0.778      0.000    0.000      3.756          61
278 spam
## 2           0.372      0.180    0.048      5.114         101
1028 spam
## 3           0.276      0.184    0.010      9.821         485
2259 spam
## 4           0.137      0.000    0.000      3.537          40
191 spam
## 5           0.135      0.000    0.000      3.537          40
191 spam
```

```
## 6           0.000      0.000    0.000      3.000           15
54 spam
```

1. . Plot the distribution of the response

```
# Plot the response distribution with distinct colors for spam and nonspam
# Visualize the distribution of the response variable, using different
colors for spam and nonspam
ggplot(spam, aes(x = type, fill = type)) +
  geom_bar() +
  scale_fill_manual(values = c("nonspam" = "violet", "spam" =
"turquoise")) +
  labs(
    title = "Distribution of Email Types (Spam vs. Non-Spam)",
    x = "Email Type",
    y = "Frequency",
    fill = "Email Type"
  ) +
  theme_minimal()
```



Comment on the graph

-The bar plot visualizes the distribution of email types in the dataset.

Each bar represents the frequency of emails classified as either spam or nonspam.

2. Comment on the shape of this dataset

```
# Get the dimensions of the dataset
dataset_size <- dim(spam)

# Display the number of observations and the number of variables
cat("n = Number of Observations:", dataset_size[1], "\n")
```

```
## n = Number of Observations: 4601

cat("p = Number of Variables:", dataset_size[2])

## p = Number of Variables: 58
```

3.Comment succinctly from the statistical perspective

The data set is relatively large with 4601 observations, which is generally sufficient for robust statistical analysis and model training.

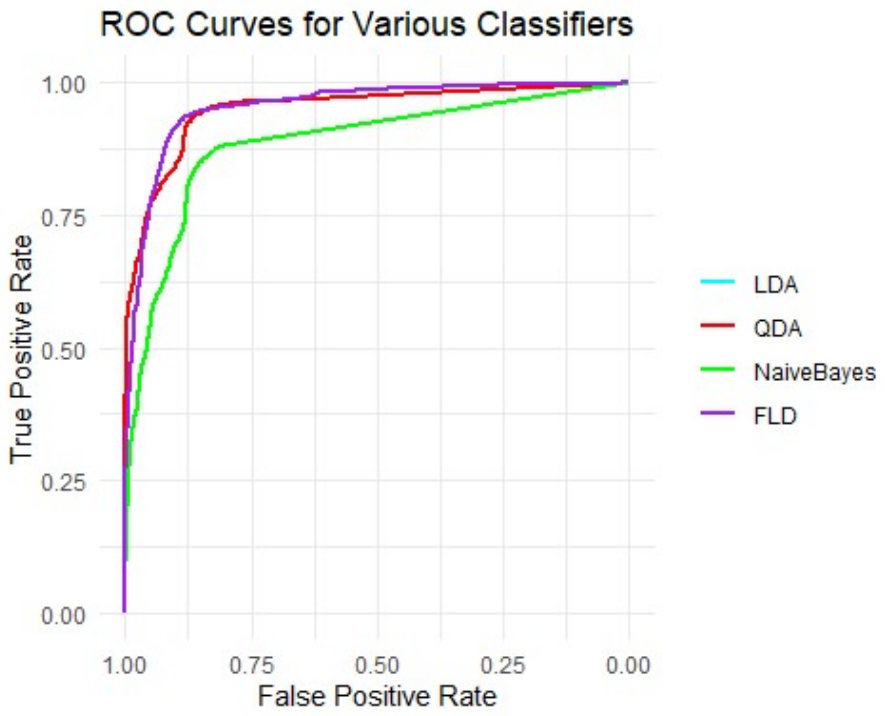
The number of variables (58) indicates that it's a high-dimensional data set. High-dimensional data can provide richer information but also poses challenges such as the risk of over fitting and the need for techniques like regularization to manage multicollinearity.

The balance between the number of observations and variables suggests that most statistical methods will be applicable, though some might require feature selection or dimensionality reduction techniques to perform optimally.

4. Using the whole data for training and the whole data for test

```
# Set seed for reproducibility and convert 'type' to a factor
set.seed(1234)
spam$type <- factor(spam$type)

# Separate predictors (X) from the target variable (Y)
X <- spam[, -58]
Y <- spam$type

# Split the dataset into training and testing sets (using all data for
both)
train_index <- createDataPartition(Y, p = 1, list = FALSE)
train_data <- spam[train_index, ]
test_data <- spam[train_index, ]

# Train different classification models
lda_model <- lda(type ~ ., data = train_data)
qda_model <- qda(type ~ ., data = train_data)
nb_model <- naiveBayes(type ~ ., data = train_data)
fld_model <- lda(type ~ ., data = train_data)  # Fisher's Linear
Discriminant (FLD) using LDA

# Predict probabilities for each model
lda_pred <- predict(lda_model, test_data)$posterior[, "spam"]
qda_pred <- predict(qda_model, test_data)$posterior[, "spam"]
nb_pred <- predict(nb_model, test_data, type = "raw")[, "spam"]
fld_pred <- predict(fld_model, test_data)$posterior[, "spam"]

# Compute ROC curves for each model
roc_lda <- roc(test_data$type, lda_pred, levels =
rev(levels(test_data$type)))

## Setting direction: controls > cases
```

```
roc_qda <- roc(test_data$type, qda_pred, levels =
rev(levels(test_data$type)))

## Setting direction: controls > cases

roc_nb <- roc(test_data$type, nb_pred, levels =
rev(levels(test_data$type)))

## Setting direction: controls > cases

roc_fld <- roc(test_data$type, fld_pred, levels =
rev(levels(test_data$type)))

## Setting direction: controls > cases

# Visualize ROC curves with enhanced styling
ggroc(list(LDA = roc_lda, QDA = roc_qda, NaiveBayes = roc_nb, FLD =
roc_fld),
      aes = c("color")) +
  geom_line(size = 1) +  # Increase line thickness
  scale_color_manual(values = c("LDA" = "cyan", "QDA" = "red",
"NaiveBayes" = "green", "FLD" = "purple")) +  # Custom colors
  labs(title = "ROC Curves for Various Classifiers", x = "False Positive
Rate", y = "True Positive Rate") +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```



5. Comment succinctly on what the ROC curves reveal

LDA (Cyan): This classifier's ROC curve suggests it has a good balance between the True Positive Rate (TPR) and the False Positive Rate (FPR). Its performance appears to be quite strong.

QDA (Red): The QDA curve shows a similar trend to LDA but might have a slightly different balance between TPR and FPR. It could be better in some areas but worse in others compared to LDA.

Naive Bayes (Green): The curve for Naive Bayes indicates its performance is relatively strong, though it might not outperform LDA and QDA in all aspects.

FLD (Purple): FLD's ROC curve indicates its performance, which can be compared directly with the others to see if it's more effective in different scenarios.

The key takeaway is that each classifier has its strengths and trade-offs. The ROC curves provide a visual way to evaluate which might be the most suitable for a specific application, depending on the desired balance between true positives and false positives.

6. Using set.seed(19671210) along with a 2/3 training 1/3 test

```
# Set seed for reproducibility of results
set.seed(19671210)

# Initialize a storage frame to collect the test errors
S <- 50  # Number of hold-out replications
errors <- data.frame(Model = character(), Error = numeric(),
stringsAsFactors = FALSE)

# Loop over 50 iterations to perform hold-out validation
for (i in 1:S) {
  # Split the data into 2/3 for training and 1/3 for testing
  train_index <- createDataPartition(spam$type, p = 2/3, list = FALSE)
  train_data <- spam[train_index, ]
  test_data <- spam[-train_index, ]

  # Train LDA and Naive Bayes models on the training data
  lda_model <- lda(type ~ ., data = train_data)
  nb_model <- naiveBayes(type ~ ., data = train_data)

  # Perform PCA on the training data and apply it to the test data
  pca_train <- prcomp(train_data[, -58], scale. = TRUE)
  pca_test <- predict(pca_train, newdata = test_data[, -58])

  # Select the number of principal components that explain at least 95% of
the variance
  var_explained <- cumsum(pca_train$sdev^2) / sum(pca_train$sdev^2)
  num_components <- which(var_explained >= 0.95)[1]

  # Prepare the PCA-transformed training and test datasets
  pca_train_data <- data.frame(pca_train$x[, 1:num_components], type =
```

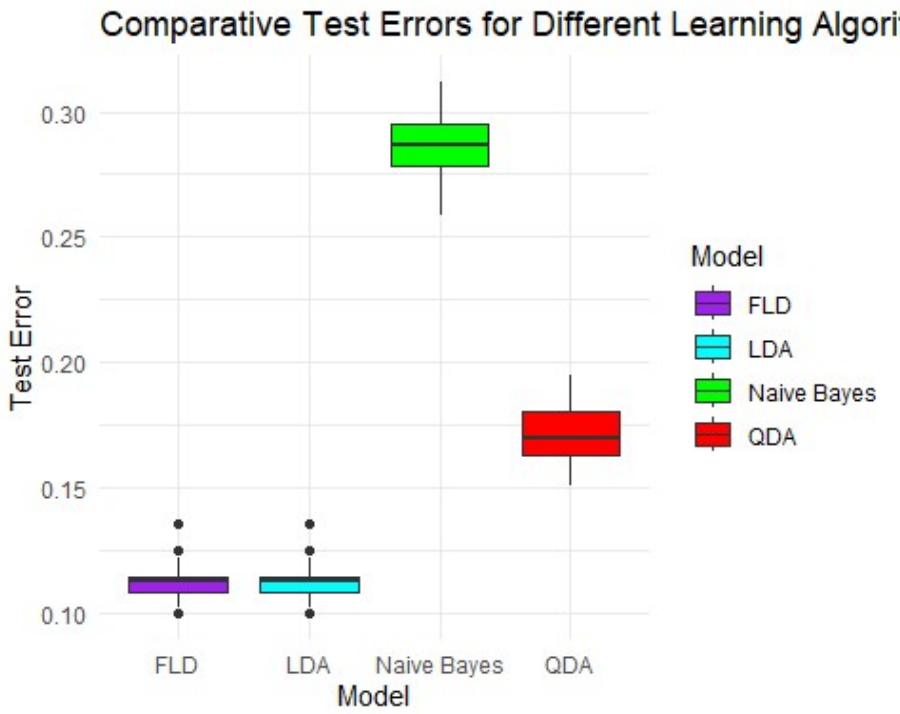
```
train_data$type)
  pca_test_data <- data.frame(pca_test[, 1:num_components], type =
test_data$type)

  # Train QDA and FLD models on the PCA-transformed training data
  qda_model <- qda(type ~ ., data = pca_train_data)
  fld_model <- lda(type ~ ., data = pca_train_data, family = binomial)

  # Make predictions and compute classification errors for each model
  lda_error <- mean(predict(lda_model, test_data)$class != test_data$type)
  qda_error <- mean(predict(qda_model, pca_test_data)$class !=
pca_test_data$type)
  nb_error <- mean(predict(nb_model, test_data) != test_data$type)
  fld_error <- mean(predict(lda_model, test_data)$class != test_data$type)

  # Store the computed errors in the storage frame
  errors <- rbind(errors, data.frame(Model = "LDA", Error = lda_error))
  errors <- rbind(errors, data.frame(Model = "QDA", Error = qda_error))
  errors <- rbind(errors, data.frame(Model = "Naive Bayes", Error =
nb_error))
  errors <- rbind(errors, data.frame(Model = "FLD", Error = fld_error))
}

# Reshape the error data frame so that the models become columns
errors_wide <- errors %>%
  mutate(Replication = rep(1:S, each = 4)) %>%  # Add a replication column
to group errors by iteration
  pivot_wider(names_from = Model, values_from = Error)

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

```
## # A tibble: 6 × 5
##   Replication    LDA   QDA `Naive Bayes`    FLD
##         <int>  <dbl> <dbl>         <dbl>  <dbl>
## 1           1 0.0998 0.167         0.279 0.0998
## 2           2 0.114  0.169         0.287 0.114
## 3           3 0.114  0.151         0.281 0.114
## 4           4 0.113  0.178         0.286 0.113
## 5           5 0.114  0.157         0.272 0.114
## 6           6 0.108  0.178         0.309 0.108
```

7. Plot the comparative boxplots

```
# Plot comparative boxplots for test errors across different models
ggplot(errors, aes(x = Model, y = Error, fill = Model)) +
  geom_boxplot() +  # Create boxplot for error distribution
  labs(
    title = "Comparative Test Errors for Different Learning Algorithms",
# Title of the plot
    x = "Model",  # Label for x-axis
    y = "Test Error"  # Label for y-axis
  ) +
  theme_minimal() +  # Apply minimal theme for a clean look
```

```
  scale_fill_manual(
    values = c("LDA" = "cyan", "QDA" = "red", "Naive Bayes" = "green",
"FLD" = "purple")  # Custom color for each model
  )
```



8. Comment on the distribution

The graph compares the test errors of four different learning algorithms: FLD (Fisher's Linear Discriminant), LDA (Linear Discriminant Analysis), Naive Bayes, and QDA (Quadratic Discriminant Analysis). Here is a detailed comment on the distribution:

1. **FLD**: The test error for FLD is tightly distributed with a low median and minimal variability. The boxplot is very narrow, indicating consistent performance across different test instances. with apparently a few outliers.

2. **LDA**: Similar to FLD, LDA has a low median test error and tight distribution. Its boxplot is slightly wider than FLD, suggesting slightly higher variability in performance. with some outliers observed.

3. **Naive Bayes**: This algorithm exhibits the highest median test error among the models and the widest interquartile range, indicating significant variability in performance. The distribution shows a long whisker, implying some test cases result in much higher error rates compared to the others. Naive Bayes appears to be less consistent and less reliable in this context.

4. **QDA**: QDA has a moderate median test error, lower than Naive Bayes but higher than both FLD and LDA. Its distribution is slightly spread out with some variability, but it has a smaller range compared to Naive Bayes. There is one notable outlier at the lower end, indicating one instance with significantly lower test error.

Overall Observation: FLD and LDA have the most consistent performance with low error rates, making them preferable for tasks requiring reliability. Naive Bayes, while being a probabilistic model, seems to perform poorly with high variability, possibly due to assumptions that do not fit the dataset. QDA provides a balance between error rate and variability but has some inconsistencies, as indicated by the outlier.