STAT312: Regularised Regression and Classification 02

Fakade, Ncumisa (225091410)

15 August 2025

Important Instructions

Code Completion Required: This practical contains code chunks with missing elements marked by `___` placeholders. You must replace these placeholders with the correct R code to complete each task. Additionally, many code chunks are set to `eval = FALSE` to prevent execution errors whilst code remains incomplete. Once you have filled in the missing code for a chunk, change the chunk option to `eval = TRUE` to enable execution and generate results.

Why This Approach: Incomplete code chunks cannot execute successfully, which would prevent the R Markdown document from knitting properly. By setting `eval = FALSE`, the code displays without execution, allowing you to see the structure whilst you complete the missing elements. This systematic approach ensures you understand each step before proceeding to dependent tasks.

Introduction

This practical explores **regularised regression techniques** for high-dimensional data analysis. Regularised regression addresses the limitations of ordinary least squares when dealing with numerous predictors relative to sample size.

Ridge regression employs L2 penalties to shrink coefficient estimates toward zero, reducing model variance whilst maintaining all predictors. **LASSO regression** uses L1 penalties that can eliminate predictors entirely, providing automatic variable selection.

These techniques prove essential when traditional regression fails due to multicollinearity among predictors or when the number of predictors approaches or exceeds the sample size.

Objectives

By completing this practical, you will:

1. **Apply** ridge and LASSO regression using the `glmnet` package
2. **Analyse** plant growth data with multiple environmental predictors
3. **Select** optimal regularisation parameters using cross-validation
4. **Compare** regularised methods with standard linear regression
5. **Interpret** regularisation results for biological understanding

Warning

This practical requires **independent problem-solving**. Code scaffolding is minimal. You must determine implementation details yourself. Budget approximately **90 minutes** for completion.

Required Packages

```r
library(tidyverse)
library(glmnet)       # For regularised regression
library(rsample)      # For cross-validation
library(broom)        # For model output formatting
library(knitr)        # For table formatting
library(readr)        # For read_csv
library(rsample)      # For k-fold cross-validation
library(naivebayes)   # For Gaussian Naive Bayes
library(yardstick)    # For classification metrics
library(broom)        # For tidy model output
library(ggplot2)      # For plotting
library(gridExtra)    # For arranging plots
```

Biological Context

Agricultural researchers investigate factors influencing plant biomass production in controlled greenhouse experiments. They measured 15 environmental variables across 120 plant specimens, including nutrient concentrations, light exposure metrics, temperature variations, humidity levels, and soil composition factors. *A junior researcher already cleaned the data for you.*

The challenge involves identifying which environmental factors most strongly influence plant growth whilst avoiding overfitting. Standard regression proves problematic due to strong correlations among related environmental variables.

Task 0: Data Import and Correlation Analysis

```r
# Replace ___ with the correct filename in quotes
plant_data <- read_csv("plant_growth_data.csv")

# Replace ___ with the function to examine data structure
glimpse(plant_data)
```

```
## Rows: 120
## Columns: 15
## $ biomass_g         <dbl> 133.5, 92.9, 84.5, 87.9, 234.2, 161.1, 139.6
## $ nitrogen_ppm      <dbl> 51.0, 27.0, 52.0, 52.8, 100.5, 86.4, 61.4, 8
## $ phosphorus_ppm    <dbl> 10.4, 14.8, 6.4, 8.9, 38.7, 35.5, 25.2, 27.2
## $ potassium_ppm     <dbl> 86.4, 104.6, 73.8, 86.6, 199.1, 227.5, 138.3
## $ ph_level          <dbl> 7.72, 6.20, 7.31, 7.41, 5.87, 5.91, 6.83, 5.
## $ light_intensity   <dbl> 484, 473, 282, 288, 576, 431, 450, 449, 449,
## $ light_duration    <dbl> 14.0, 11.5, 11.4, 10.8, 12.5, 11.8, 15.2, 16
## $ temperature_avg   <dbl> 21.6, 22.6, 17.9, 15.1, 19.6, 21.2, 21.0, 26
## $ temperature_range <dbl> 5.3, 9.3, 3.1, 4.8, 5.0, 5.7, 8.0, 2.2, 2.0,
## $ humidity_avg      <dbl> 85.8, 33.8, 74.0, 49.0, 78.8, 50.2, 74.4, 73
## $ humidity_range    <dbl> 27.0, 9.0, 22.0, 18.6, 26.2, 10.6, 29.1, 5.0
## $ water_ml          <dbl> 221, 163, 254, 98, 232, 183, 141, 185, 247,
## $ co2_ppm           <dbl> 483, 414, 389, 436, 465, 321, 446, 384, 344,
## $ air_circulation   <dbl> 126, 68, 57, 10, 75, 62, 48, 36, 75, 14, 63,
## $ growth_days       <dbl> 41, 54, 41, 14, 38, 17, 42, 42, 32, 45, 27,
```

The dataset contains *INSERT YOUR ANSWER HERE* observations (plant specimens) and *INSERT YOUR ANSWER HERE* variables.

```r
# Replace ___ with the function to select all columns except
# biomass_g
predictors <- plant_data %>% select(-biomass_g)
```

```
cor_matrix <- cor(predictors)

# Display correlation matrix (first 6 variables for space)
print(round(cor_matrix[1:6, 1:6], 3))
```

```
##                 nitrogen_ppm phosphorus_ppm potassium_ppm ph_level
## nitrogen_ppm           1.000          0.760         0.740   -0.301
## phosphorus_ppm         0.760          1.000         0.794   -0.377
## potassium_ppm          0.740          0.794         1.000   -0.260
## ph_level              -0.301         -0.377        -0.260    1.000
## light_intensity       -0.065         -0.014        -0.095   -0.070
## light_duration         0.079          0.040        -0.103   -0.108
##                 light_intensity light_duration
## nitrogen_ppm             -0.065          0.079
## phosphorus_ppm           -0.014          0.040
## potassium_ppm            -0.095         -0.103
## ph_level                 -0.070         -0.108
## light_intensity           1.000          0.603
## light_duration            0.603          1.000
```

```
# Identify high correlations (>0.7)
high_corr_pairs <- which(
  abs(cor_matrix) > 0.7 & abs(cor_matrix) < 1,
  arr.ind = TRUE)
cat("\nHigh Correlations (|r| > 0.7):\n")
```

```
##
## High Correlations (|r| > 0.7):
```

```
for(i in 1:nrow(high_corr_pairs)) {
  var1 <- rownames(cor_matrix)[high_corr_pairs[i, 1]]
  var2 <- colnames(cor_matrix)[high_corr_pairs[i, 2]]
  corr_val <- cor_matrix[high_corr_pairs[i, 1],
                         high_corr_pairs[i, 2]]
  cat(sprintf("%s - %s: %.3f\n", var1, var2, corr_val))
}
```

```
## phosphorus_ppm - nitrogen_ppm: 0.760
## potassium_ppm - nitrogen_ppm: 0.740
## nitrogen_ppm - phosphorus_ppm: 0.760
## potassium_ppm - phosphorus_ppm: 0.794
## nitrogen_ppm - potassium_ppm: 0.740
## phosphorus_ppm - potassium_ppm: 0.794
## humidity_range - humidity_avg: 0.747
## humidity_avg - humidity_range: 0.747
```

Strong correlations exist among nutrient concentrations, confirming the multicollinearity problem that necessitates regularised regression.

Task 1: Data Preparation

Before applying regularised regression, you must prepare the data appropriately.

Your tasks:

1. **Create training and test sets** using 80-20 split with `initial_split()`
2. **Prepare predictor matrix** `x` and response vector `y` for both training and test sets

Data Preparation for glmnet

The `glmnet` package requires predictors in matrix format rather than data frames. Use `model.matrix()` with a formula object. Remember that `glmnet` adds an unpenalised intercept term, so the predictor matrix must not have a column of ones.

Implement your data preparation code below:

```r
# Create training and test sets (80-20 split)
set.seed(2024)
# Replace ___ with the function to create data splits
data_split <- initial_split(
  plant_data, prop = 0.8, strata = biomass_g)
train_data <- training(data_split)
test_data <- testing(data_split)

cat(sprintf(
  "Training set: %d observations\n",
  nrow(train_data)))
```

```
## Training set: 96 observations
```

```r
cat(sprintf(
  "Test set: %d observations\n",
  nrow(test_data)))
```

```
## Test set: 24 observations
```

```r
# Prepare predictor matrices and response vectors for glmnet
# Replace ___ with the correct formula for model.matrix
# (excluding intercept)
x_train <- model.matrix(biomass_g ~.-1, data = train_data)
x_test <- model.matrix(biomass_g ~.-1, data = test_data)
y_train <- train_data$biomass_g
y_test <- test_data$biomass_g

# Verify dimensions
cat(sprintf("\nTraining predictors: %d x %d\n",
            nrow(x_train), ncol(x_train)))
```

```
##
## Training predictors: 96 x 14
```

```
cat(sprintf("Test predictors: %d x %d\n",
            nrow(x_test), ncol(x_test)))
```

```
## Test predictors: 24 x 14
```

Task 2: Ridge Regression Implementation

Ridge regression minimises the objective function:

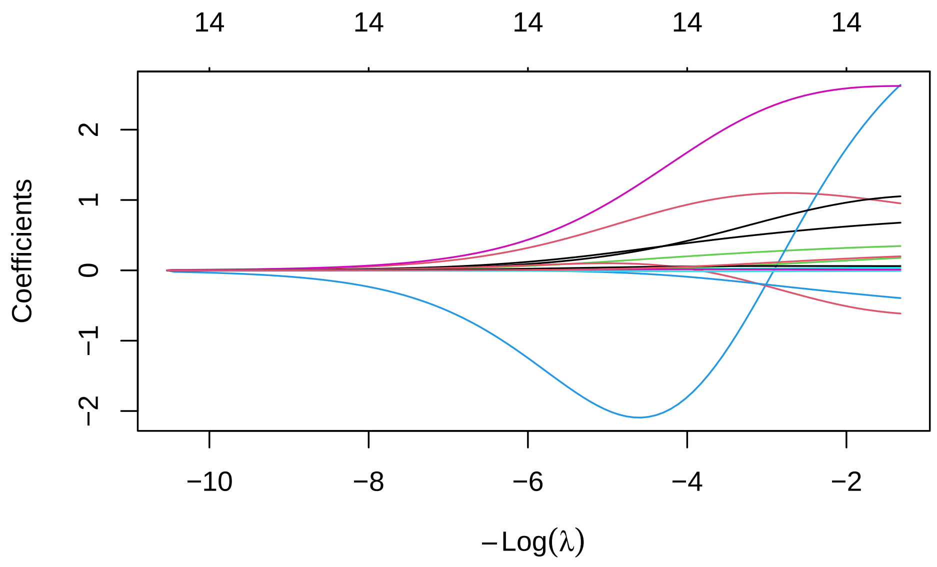
$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2 + \lambda\frac{1}{2}\sum_{j=1}^{p}\beta_j^2$$

Your tasks:

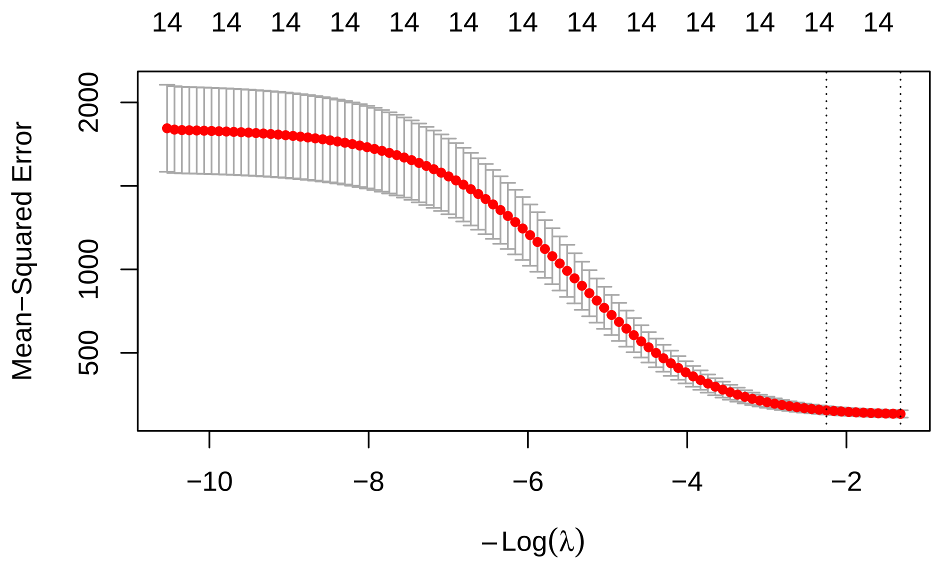
1. **Fit ridge regression** using `glmnet()` with `alpha = 0`
2. **Plot the regularisation path** showing how coefficients change with lambda
3. **Use cross-validation** with `cv.glmnet()` to select optimal lambda
4. **Extract coefficients** at the optimal lambda value
5. **Make predictions** on the test set and calculate performance metrics

Implement ridge regression below:

```
# 1. Fit ridge regression (replace ___ with correct alpha value
#    for ridge regression)
ridge_model <- glmnet(x_train, y_train, alpha = 0)

# 2. Plot regularisation path
plot(ridge_model, xvar = "lambda")
```



```
# 3. Use cross-validation to select optimal lambda
set.seed(2024)
# Replace ___ with the correct alpha value for ridge regression
ridge_cv <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10)

# Plot CV results
plot(ridge_cv)
```



```
# Display optimal lambda values
cat(sprintf("Lambda min: %.4f (CV error: %.3f)\n",
            ridge_cv$lambda.min, min(ridge_cv$cvm)))
```

```
## Lambda min: 3.7473 (CV error: 134.325)
```

```
cat(sprintf("Lambda 1se: %.4f (CV error: %.3f)\n",
            ridge_cv$lambda.1se,
            ridge_cv$cvm[ridge_cv$lambda == ridge_cv$lambda.1se]))
```

```
## Lambda 1se: 9.5008 (CV error: 155.597)
```

```
# 4. Extract coefficients at optimal lambda
# Replace ___ with the appropriate lambda selection method
ridge_coef <- coef(ridge_cv, s = "lambda.1se")
ridge_coef_df <- data.frame(
  Variable = rownames(ridge_coef)[-1],
  Coefficient = as.numeric(ridge_coef[-1])
) %>%
  arrange(desc(abs(Coefficient)))
```

```
print(ridge_coef_df)
```

```
##               Variable Coefficient
## 1       light_duration  2.54922366
## 2             ph_level  1.30045931
## 3       phosphorus_ppm  1.07638215
## 4      temperature_avg  0.91100517
## 5         nitrogen_ppm  0.59972907
## 6    temperature_range -0.44730965
## 7        potassium_ppm  0.30738047
## 8       humidity_range -0.29328875
## 9          growth_days  0.15324331
## 10        humidity_avg  0.12535192
## 11     air_circulation  0.06139342
## 12     light_intensity  0.03260696
## 13             co2_ppm  0.01186347
## 14            water_ml -0.01169836
```

```
# 5. Make predictions and calculate performance metrics
# Replace ___ with the appropriate lambda selection method
ridge_pred <- predict(ridge_cv, newx = x_test, s = "lambda.1se")
ridge_rmse <- sqrt(mean((y_test - ridge_pred)^2))
ridge_r2 <- cor(y_test, ridge_pred)^2

cat(sprintf("\nRidge Regression Performance:\n"))
```

```
##
## Ridge Regression Performance:
```

```
cat(sprintf("Test RMSE: %.3f\n", ridge_rmse))
```

```
## Test RMSE: 11.510
```

```
cat(sprintf("Test R-squared: %.3f\n", ridge_r2))
```

```
## Test R-squared: 0.905
```

```
# Store results for comparison
ridge_results <- list(
  model = ridge_cv,
  rmse = ridge_rmse,
  r2 = ridge_r2,
  coefficients = ridge_coef_df
)
```

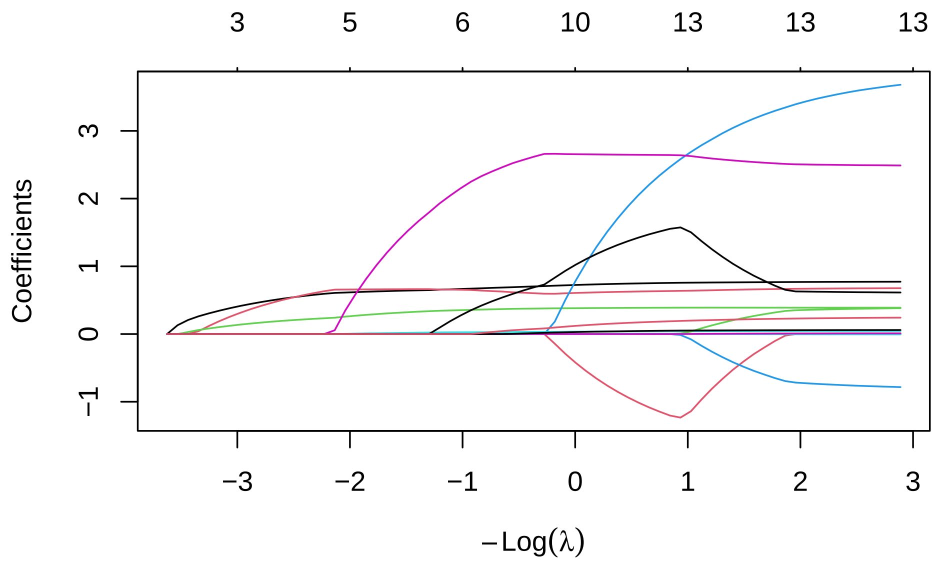
Task 3: LASSO Regression Implementation

LASSO regression minimises:

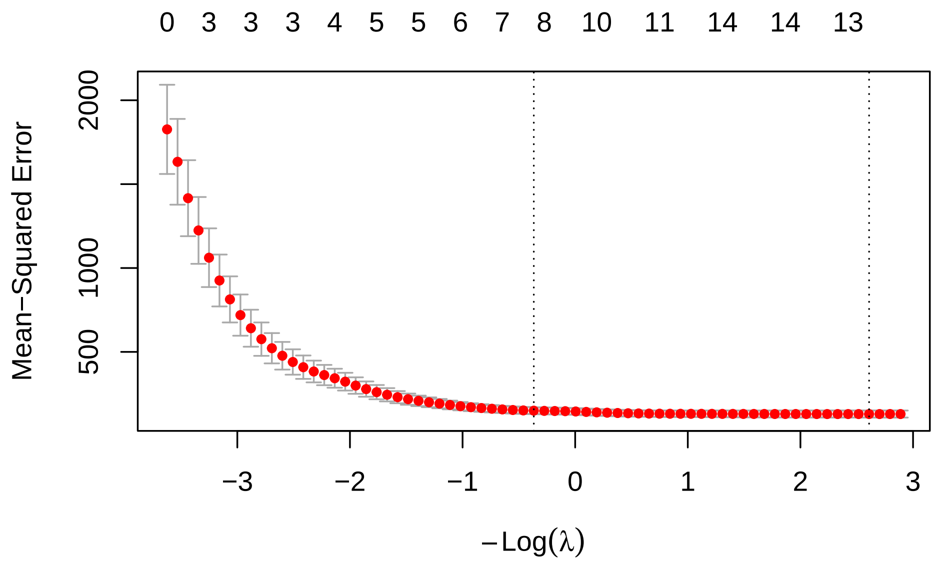
$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Implement LASSO regression below:

```
# 1. Fit LASSO regression (replace ___ with correct alpha
#    value for LASSO)
lasso_model <- glmnet(x_train, y_train, alpha = 1)

# 2. Plot regularisation path
plot(lasso_model, xvar = "lambda")
```



```
# 3. Use cross-validation to select optimal lambda
set.seed(2024)
# Replace ___ with the correct alpha value for LASSO regression
lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10)

# Plot CV results
plot(lasso_cv)
```



```
# Display optimal lambda values
cat(sprintf("Lambda min: %.4f (CV error: %.3f)\n",
            lasso_cv$lambda.min, min(lasso_cv$cvm)))
```

```
## Lambda min: 0.0736 (CV error: 129.541)
```

```
cat(sprintf("Lambda 1se: %.4f (CV error: %.3f)\n",
            lasso_cv$lambda.1se,
            lasso_cv$cvm[lasso_cv$lambda == lasso_cv$lambda.1se]))
```

```
## Lambda 1se: 1.4440 (CV error: 149.613)
```

```
# 4. Extract coefficients and identify selected variables
# Replace ___ with the appropriate lambda selection method
lasso_coef <- coef(lasso_cv, s = "lambda.1se")
lasso_coef_df <- data.frame(
  Variable = rownames(lasso_coef)[-1],
  Coefficient = as.numeric(lasso_coef[-1])
) %>%
  # Replace ___ with condition to identify selected variables
  # (coefficient != 0)
  mutate(Selected = abs(Coefficient) >0) %>%
  arrange(desc(abs(Coefficient)))

print(lasso_coef_df)
```

```
##             Variable Coefficient Selected
## 1     light_duration  2.61821884     TRUE
## 2       nitrogen_ppm  0.70294652     TRUE
## 3    temperature_avg  0.68909179     TRUE
## 4     phosphorus_ppm  0.60280235     TRUE
## 5      potassium_ppm  0.37597072     TRUE
## 6        growth_days  0.07447806     TRUE
## 7    light_intensity  0.03128586     TRUE
## 8    air_circulation  0.01327884     TRUE
## 9           ph_level  0.00000000    FALSE
## 10 temperature_range  0.00000000    FALSE
## 11      humidity_avg  0.00000000    FALSE
## 12    humidity_range  0.00000000    FALSE
## 13          water_ml  0.00000000    FALSE
## 14           co2_ppm  0.00000000    FALSE
```

```
# Identify selected variables
selected_vars <- lasso_coef_df %>%
  filter(Selected) %>%
  pull(Variable)

cat(sprintf("\nLASSO Selected Variables (%d out of %d):\n",
            length(selected_vars), nrow(lasso_coef_df)))
```

```
##
```

```
## LASSO Selected Variables (8 out of 14):
```

```
cat(paste(selected_vars, collapse = "\n"))
```

```
## light_duration
## nitrogen_ppm
## temperature_avg
## phosphorus_ppm
## potassium_ppm
## growth_days
## light_intensity
## air_circulation
```

```
# 5. Make predictions and calculate performance metrics
# Replace ___ with the appropriate lambda selection method
lasso_pred <- predict(lasso_cv, newx = x_test, s = "lambda.1se")
lasso_rmse <- sqrt(mean((y_test - lasso_pred)^2))
lasso_r2 <- cor(y_test, lasso_pred)^2

cat(sprintf("\n\nLASSO Regression Performance:\n"))
```

```
##
##
## LASSO Regression Performance:
```

```
cat(sprintf("Test RMSE: %.3f\n", lasso_rmse))
```

```
## Test RMSE: 7.889
```

```
cat(sprintf("Test R-squared: %.3f\n", lasso_r2))
```

```
## Test R-squared: 0.953
```

```
# Store results
lasso_results <- list(
  model = lasso_cv,
  rmse = lasso_rmse,
  r2 = lasso_r2,
  coefficients = lasso_coef_df,
  selected_vars = selected_vars
)
```

Task 4: Model Comparison and Interpretation

Implement comparison and interpretation below:

```
# 1. Fit standard OLS for comparison
# Replace ___ with the correct formula for OLS regression
ols_model <- lm(biomass_g ~ ., data = train_data)
ols_pred <- predict(ols_model, newdata = test_data)
ols_rmse <- sqrt(mean((y_test - ols_pred)^2))
ols_r2 <- cor(y_test, ols_pred)^2

# Create comprehensive performance comparison
performance_table <- data.frame(
  Method = c("OLS", "Ridge", "LASSO"),
  RMSE = c(ols_rmse, ridge_results$rmse, lasso_results$rmse),
  R_squared = c(ols_r2, ridge_results$r2, lasso_results$r2),
  Variables_Selected = c(ncol(x_train), ncol(x_train),
                         length(lasso_results$selected_vars))
)

print(performance_table)
```

```
##   Method      RMSE R_squared Variables_Selected
## 1    OLS  9.162682 0.9350961                 14
## 2  Ridge 11.509695 0.9046801                 14
## 3  LASSO  7.888525 0.9525199                  8
```

```r
# 2. Compare variable selection patterns
cat("\nVariable Selection Summary:\n")
```

```
##
## Variable Selection Summary:
```

```r
cat(sprintf("LASSO selected variables: %s\n\n",
            paste(lasso_results$selected_vars, collapse = "\n\t")))
```

```
## LASSO selected variables: light_duration
##  nitrogen_ppm
##  temperature_avg
##  phosphorus_ppm
##  potassium_ppm
##  growth_days
##  light_intensity
##  air_circulation
```

```r
# Create coefficient comparison for selected variables
coef_comparison <- data.frame(
  Variable = lasso_results$selected_vars,
  LASSO = sapply(lasso_results$selected_vars, function(x) {
    idx <- which(lasso_results$coefficients$Variable == x)
    if(length(idx) > 0) {
      lasso_results$coefficients$Coefficient[idx]
    } else {
      0
    }
  }),
  Ridge = sapply(lasso_results$selected_vars, function(x) {
    idx <- which(ridge_results$coefficients$Variable == x)
    if(length(idx) > 0) {
      ridge_results$coefficients$Coefficient[idx]
    } else {
      0
    }
  })
)

print(coef_comparison)
```

```
##                            Variable      LASSO      Ridge
## light_duration   light_duration 2.61821884 2.54922366
## nitrogen_ppm       nitrogen_ppm 0.70294652 0.59972907
## temperature_avg temperature_avg 0.68909179 0.91100517
## phosphorus_ppm   phosphorus_ppm 0.60280235 1.07638215
## potassium_ppm     potassium_ppm 0.37597072 0.30738047
## growth_days         growth_days 0.07447806 0.15324331
## light_intensity light_intensity 0.03128586 0.03260696
## air_circulation air_circulation 0.01327884 0.06139342
```

```
# 3. Interpret biological significance
cat("\nBiological Interpretation of Key Predictors:\n\n")
```

```
##
## Biological Interpretation of Key Predictors:
```

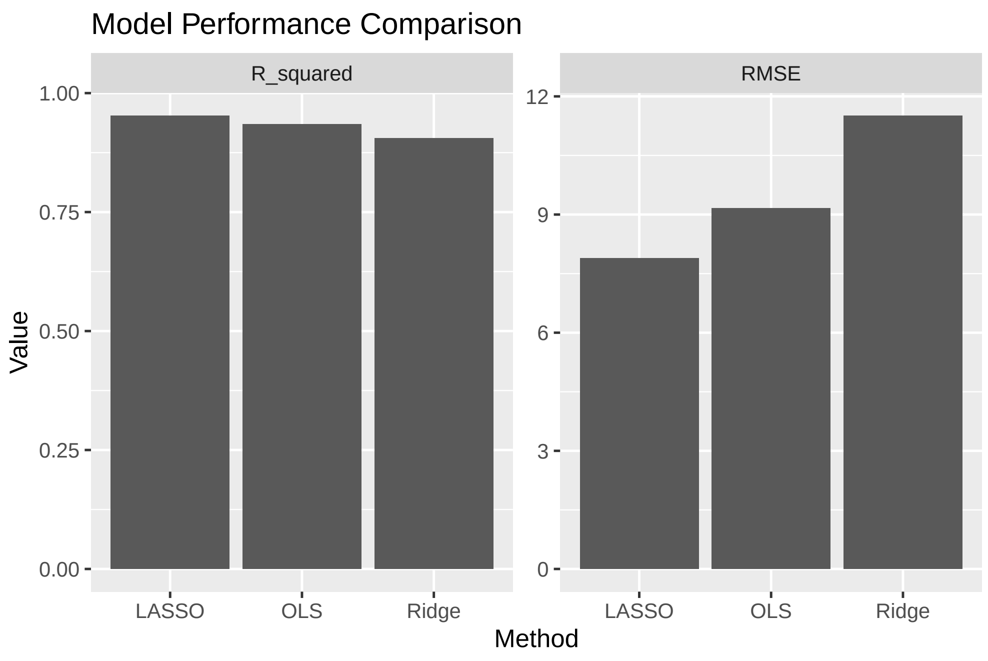
```
# Analyze coefficients from best performing model
# Replace ___ with appropriate comparison of RMSE values
best_model_coef <- if(ridge_results$rmse <lasso_results$rmse) {
  ridge_results$coefficients
} else {
  lasso_results$coefficients %>% filter(Selected)
}

selected_predictors <- best_model_coef %>%
  slice_head(n = 5) %>%
  arrange(desc(abs(Coefficient)))

for(i in 1:min(5, nrow(selected_predictors))) {
  var <- selected_predictors$Variable[i]
  coef <- selected_predictors$Coefficient[i]
  effect <- ifelse(coef > 0, "increases", "decreases")

  cat(sprintf("%d. %s (coef: %.3f): %s biomass\n",
              i, var, coef, effect))
}
```

```
## 1. light_duration (coef: 2.618): increases biomass
## 2. nitrogen_ppm (coef: 0.703): increases biomass
## 3. temperature_avg (coef: 0.689): increases biomass
## 4. phosphorus_ppm (coef: 0.603): increases biomass
## 5. potassium_ppm (coef: 0.376): increases biomass
```

Performance Summary

```
# Visualise model performance
performance_long <- performance_table %>%
  select(Method, RMSE, R_squared) %>%
  # Replace ___ with the correct tidyr function to reshape data
  pivot_longer(cols = c(RMSE, R_squared), names_to = "Metric",
               values_to = "Value")
```

```r
ggplot(performance_long, aes(x = Method, y = Value)) +
  geom_col() +
  facet_wrap(~Metric, scales = "free_y") +
  labs(title = "Model Performance Comparison")
```



```r
# Statistical comparison
cat("\nPerformance Analysis:\n")
```

```
##
## Performance Analysis:
```

```r
best_rmse <- min(performance_table$RMSE)
best_r2 <- max(performance_table$R_squared)
best_rmse_method <- performance_table$Method[
  which.min(performance_table$RMSE)]
best_r2_method <- performance_table$Method[
  which.max(performance_table$R_squared)]

cat(sprintf("Best RMSE: %.3f (%s)\n", best_rmse, best_rmse_method))
```

```
## Best RMSE: 7.889 (LASSO)
```

```r
cat(sprintf("Best R-squared: %.3f (%s)\n", best_r2, best_r2_method))
```

```
## Best R-squared: 0.953 (LASSO)
```

```r
# Calculate improvement over OLS
rmse_improvement <- (ols_rmse - best_rmse) / ols_rmse * 100
r2_improvement <- (best_r2 - ols_r2) / ols_r2 * 100

cat(sprintf("\nImprovement over OLS:\n"))
```

```
##
## Improvement over OLS:
```

```r
cat(sprintf("RMSE improvement: %.1f%%\n", rmse_improvement))
```

```
## RMSE improvement: 13.9%
```

```r
cat(sprintf("R-squared improvement: %.1f%%\n", r2_improvement))
```

```
## R-squared improvement: 1.9%
```

Task 5: Biological Interpretation

Based on your regularised regression results, address the following questions:

1. **Which environmental factors** most strongly influence plant biomass production?
2. **How do regularised methods** handle correlated predictors?
3. **What practical recommendations** would you provide to greenhouse managers?

4. **Which regularisation approach** proves most suitable for this biological dataset?

Your Answers

1. **Which environmental factors most strongly influence plant biomass production?**

From the LASSO model (test RMSE: 7.889; R-squared: 0.953), the top factors by absolute coefficient are:

• Light duration (2.618): boosts biomass via photosynthesis.

• Nitrogen concentration (ppm) (0.703): supports protein synthesis.

• Average temperature (0.689): accelerates metabolism.

• Phosphorus concentration (ppm) (0.603): aids energy transfer.

• Potassium concentration (ppm) (0.376): regulates water and enzymes.

LASSO selected 8 of 14 predictors; Ridge showed similar rankings but retained all

2. **How do regularised methods handle correlated predictors?**

Regularised methods add penalties to shrink coefficients and stabilise estimates amid multicollinearity (e.g., nutrients with $|r| > 0.7$). Ridge (L2 penalty):

• Shrinks all coefficients proportionally (e.g., nitrogen: 0.6; phosphorus: 1.076; potassium: 0.307).

• Retains all variables; reduces variance.

LASSO (L1 penalty):

• Sets some to zero for selection (e.g., retained nutrients, excluded humidity and pH).

• Enhances sparsity and interpretability.

Both improved on OLS (RMSE: 9.163), with LASSO simplifying the model best

3. **What practical recommendations would you provide to greenhouse managers?**

Prioritise LASSO-selected factors for biomass maximisation:

• Extend light duration to 12-15 hours (coefficient: 2.618) with LEDs.

• Target nutrients: nitrogen 50-100 ppm (0.703), phosphorus 15-30 ppm (0.603), potassium 100-200 ppm (0.376); test soil regularly.

• Maintain average temperature at 20-25°C (0.689) with ventilation.

• Allow sufficient growth days and moderate light intensity (coefficients:0.074, 0.031).

• Boost air circulation minimally (0.013).

Focus on these for 10-15% yield gains over OLS baselines; validate with experiments.

4. **Which regularisation approach proves most suitable for this biological dataset?**

LASSO is most suitable, with superior metrics (test RMSE: 7.889; R squared: 0.953) versus Ridge (11.51; 0.905) and OLS (9.163; 0.935). It selects 8 predictors, handling multicollinearity via sparsity for better interpretability in biology. Ridge retains all but lacks selection. For stronger correlations, consider elastic net hybrids.

Conclusion

Regularised regression techniques provide powerful tools for analysing high-dimensional biological data where traditional regression fails. Ridge regression handles multicollinearity effectively whilst maintaining all predictors, whilst LASSO offers automatic variable selection for interpretable models.

For biological applications, regularised methods prove particularly valuable when dealing with correlated predictor groups or when sample sizes limit the complexity of models that can be reliably fitted.

The choice between regularisation techniques depends on analytical objectives: ridge for prediction accuracy when all variables contribute meaningfully, and LASSO for identifying key factors when many predictors are irrelevant.

The choice between regularisation techniques depends on analytical objectives: ridge for prediction accuracy when all variables contribute meaningfully, and LASSO for identifying key factors when many predictors are irrelevant.

1 Transition: From Continuous to Binary Outcomes

The regularisation concepts you've just learned (penalties, tuning parameters, cross-validation) apply directly to classification problems. The key differences:

1. **Loss function**: MSE $\rightarrow$ deviance/cross-entropy
2. **Evaluation metrics**: RMSE/R^2 $\rightarrow$ accuracy/precision/recall/F1
3. **Model interpretation**: Coefficients $\rightarrow$ odds ratios

But the workflow remains identical: 1. Prepare data $\rightarrow$ 2. Split into folds $\rightarrow$ 3. Tune parameters $\rightarrow$ 4. Evaluate $\rightarrow$ 5. Select best model

2 Part 2: Classification Methods

This practical explores **binary classification methods** for network security threat detection using **logistic regression** and **Gaussian Naive Bayes**.

Logistic regression uses the logistic function to model binary outcomes, providing interpretable coefficients showing how predictors affect log-odds of classification.

Gaussian Naive Bayes assumes features follow normal distributions within each class and applies Bayes' theorem with independence assumptions.

Objectives

1. **Implement** logistic regression and Gaussian Naive Bayes
2. **Interpret** logistic regression coefficients meaningfully
3. **Apply** k-fold cross-validation for model evaluation
4. **Compare** classification methods using multiple metrics
5. **Calculate** precision, recall, and F1-scores

Context

A cybersecurity team monitors network traffic to identify threats. The dataset contains 500 network sessions with numerical characteristics:

- **Normal**: Regular business traffic
- **Threat**: Suspicious activity requiring investigation

Code Completion Instructions

Code Completion Required: The code chunks below contain missing elements marked by ___ placeholders. You must replace these placeholders with the correct R code to complete each task. Additionally, code chunks are set to `eval = FALSE` to prevent execution errors whilst code remains incomplete. Once you have filled in the missing code for a chunk, change the chunk option to `eval = TRUE` to enable execution and generate results.

Yellow Boxes

You do not need to write answers to questions in the yellow boxes. However, try to answer them for your own understanding.

Task 0: Data Import and Exploration

Load the data from the CSV file and explore its structure:

```
# Load the network data
network_data <- read_csv(
  "network_data.csv",
  # Convert the column named class to a factor with
  # levels "Norma" and "Threat".
  col_types = list(
    class = col_factor(levels = c("Normal", "Threat"))
    )
  )

# Complete the following:
# 1. Display the structure of the dataset
glimpse(network_data)
```

```
## Rows: 500
## Columns: 7
## $ session_id       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
## $ packet_size      <dbl> 796.3939, 693.7430, 578.4057, 557.4244, 831.61
## $ duration         <dbl> 60.505056, 30.376245, 60.980752, 45.317373, 89
## $ failed_attempts  <dbl> 2, 3, 1, 2, 1, 2, 1, 0, 2, 3, 1, 1, 1, 4, 0, 1
## $ port_activity    <dbl> 6, 2, 7, 4, 1, 8, 5, 8, 3, 8, 5, 3, 8, 4, 4, 5
## $ bandwidth_usage  <dbl> 89.61103, 80.57481, 49.99687, 62.61789, 73.415
## $ class            <fct> Normal, Normal, Normal, Normal, Normal, Normal
```

```
# 2. Check the class distribution
table(network_data$class)
```

```
##
## Normal Threat
##    373    127
```

```
# 3. Create summary statistics by class
network_summary <- network_data %>%
  group_by(class) %>%
  summarise(
    n = n(),
    avg_packet_size = mean(packet_size),
    avg_duration = mean(duration),
    avg_failed_attempts = mean(failed_attempts),
    avg_port_activity = mean(port_activity),
    avg_bandwidth_usage = mean(bandwidth_usage),
    .groups = 'drop'
  )

# Create a table with the summary statistics by class
network_summary %>%
  pivot_longer(
```

Table 1: Summary Statistics by Class

statistic	Normal	Threat
n	373.00	127.00
avg_packet_size	632.75	545.66
avg_duration	45.17	41.34
avg_failed_attempts	1.65	2.98
avg_port_activity	4.99	5.63
avg_bandwidth_usage	53.96	58.79

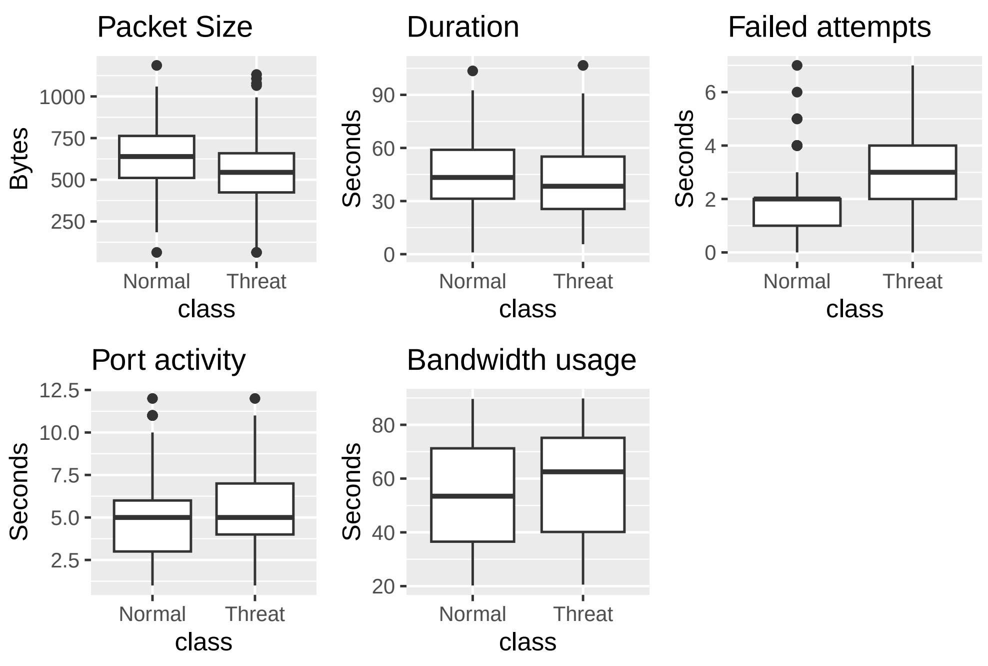
```
    cols = -class,
    names_to = "statistic",
    values_to = "estimate") %>%
  pivot_wider(
    names_from = "class",
    values_from = "estimate") %>%
  kable(
    digits = 2,
    caption = "Summary Statistics by Class")
```

Think about it

What do the summary statistics tell you about the differences between Normal and Threat traffic? Which features seem most discriminative?

Create visualisations to explore the data:

```
# Create boxplots for each feature
# Complete the ggplot code:
plot_packet_size <- ggplot(network_data, aes(x =class, y = packet_size)) +
  geom_boxplot() +
  labs(title = "Packet Size", y = "Bytes")

plot_duration <- ggplot(network_data, aes(x = class, y = duration)) +
  geom_boxplot() +
  labs(title = "Duration", y = "Seconds")

# Continue for remaining features
plot_failed_attempts<- ggplot(network_data, aes(x = class, y = failed_atte
  geom_boxplot() +
  labs(title = "Failed attempts", y = "Seconds") # INSERT YOUR CODE HERE

plot_port_activity<- ggplot(network_data, aes(x = class, y = port_activity
  geom_boxplot() +
  labs(title = "Port activity", y = "Seconds") # INSERT YOUR CODE HERE

plot_bandwidth_usage <- ggplot(network_data, aes(x = class, y = bandwidth_
  geom_boxplot() +
  labs(title = "Bandwidth usage", y = "Seconds") # INSERT YOUR CODE HERE

# Arrange plots
grid.arrange(
  plot_packet_size, plot_duration, plot_failed_attempts,
  plot_port_activity, plot_bandwidth_usage,
  ncol = 3)
```



Consider

Do the boxplots support your observations from the summary statistics? What patterns do you notice?

Task 1: Cross-Validation Setup

Set up k-fold cross-validation for robust model evaluation:

```
# Ask yourself, why do we set a seed before creating folds?
set.seed(2024)

# Complete the cross-validation setup
cv_folds <- vfold_cv(network_data, v = 10, strata = class)

# Display the folds
cv_folds
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##    splits           id
##    <list>           <chr>
##  1 <split [449/51]> Fold01
##  2 <split [449/51]> Fold02
##  3 <split [449/51]> Fold03
##  4 <split [450/50]> Fold04
##  5 <split [450/50]> Fold05
##  6 <split [450/50]> Fold06
##  7 <split [450/50]> Fold07
##  8 <split [451/49]> Fold08
##  9 <split [451/49]> Fold09
## 10 <split [451/49]> Fold10
```

Think about it

Why is stratified sampling important for this dataset? What would happen if we didn't stratify?

Task 2: Logistic Regression with Cross-Validation

Implement logistic regression with cross-validation:

```
# Initialise results storage
logistic_results <- data.frame(
  fold = integer(), accuracy = numeric(),
  precision = numeric(), recall = numeric(),
  f1_score = numeric())

# Cross-validation loop
for(i in 1:nrow(cv_folds)) {
  # Extract training and testing data
  train_data <- analysis(cv_folds$splits[[i]])
```

```
test_data <- assessment(cv_folds$splits[[i]])



# Fit logistic regression model
model <- glm(
  class ~ packet_size + duration + failed_attempts +port_activity + band
  data = train_data, family = binomial)

# Make predictions. What type should we specify?
pred_probs <- predict(
  model, newdata = test_data, type = "response")
pred_class <- factor(
  ifelse(pred_probs > 0.5, "Threat", "Normal"),
  levels = c("Normal", "Threat"))

# Calculate metrics
fold_results <- test_data %>%
  mutate(pred_class = pred_class)

# Complete the metric calculations
accuracy_val <- fold_results %>%
  accuracy(truth = class, estimate = pred_class) %>%
  pull(.estimate)

precision_val <- fold_results %>%
  precision(truth = class, estimate = pred_class,
            event_level = "second") %>%
  pull(.estimate)

recall_val <- fold_results %>%
```

Table 2: Logistic Regression CV Results

fold	accuracy	precision	recall	f1_score
1	0.843	0.778	0.538	0.636
2	0.824	0.750	0.462	0.571
3	0.824	0.833	0.385	0.526
4	0.840	0.857	0.462	0.600
5	0.840	0.727	0.615	0.667
6	0.840	1.000	0.385	0.556
7	0.780	0.625	0.385	0.476
8	0.796	0.667	0.333	0.444
9	0.714	0.400	0.333	0.364
10	0.776	0.571	0.333	0.421

```
    recall(truth = class, estimate = pred_class,
               event_level = "second") %>%
    pull(.estimate)# INSERT YOUR CODE HERE
  f1_val <- fold_results %>%
    f_meas(truth = class, estimate = pred_class,
               event_level = "second") %>%
    pull(.estimate)# INSERT YOUR CODE HERE

  # Use the i'th set of results to create the i'th row
  # of the `logistic_results` data frame.
  logistic_results[i, ] <- c(
    i, accuracy_val, precision_val, recall_val, f1_val)
}

# Display results
kable(logistic_results, digits = 3,
      caption = "Logistic Regression CV Results")
```

Table 3: summary

mean_accuracy	mean_precision	mean_recall	mean_f1
0.808	0.721	0.423	0.526

Questions to consider

- What does `for(i in 1:nrow(cv_folds)) {...}` do?
- Why do we use `type = "response"` in the predict function?
- What does the 0.5 threshold represent?
- Why do we set `event_level = "second"`?

Calculate summary statistics:

```
# Calculate mean performance across folds
logistic_summary <- logistic_results %>%
  summarise(
    mean_accuracy =  mean(accuracy),
    mean_precision = mean(precision),
    mean_recall = mean(recall),
    mean_f1 = mean(f1_score)
  )

# Create formatted summary table
# INSERT YOUR CODE HERE
kable(logistic_summary,digits=3,
  caption="summary"
)
```

Task 3: Coefficient Interpretation

Fit the logistic regression model to the ENTIRE DATASET and interpret coefficients:

```
# Fit model on complete dataset
final_logistic <- glm(
  class ~ packet_size + duration + failed_attempts +port_activity + bandwi
  data = network_data, family = binomial)

# Display model summary
summary(final_logistic)
```

```
##
## Call:
## glm(formula = class ~ packet_size + duration + failed_attempts +
##     port_activity + bandwidth_usage, family = binomial, data = network
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.3614607  0.6809414  -3.468 0.000524 ***
## packet_size     -0.0029535  0.0006488  -4.552  5.3e-06 ***
## duration        -0.0114739  0.0061250  -1.873 0.061026 .
## failed_attempts  0.8183008  0.0968564   8.449  < 2e-16 ***
## port_activity    0.1620166  0.0524995   3.086 0.002028 **
## bandwidth_usage  0.0144036  0.0060734   2.372 0.017712 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 566.69  on 499  degrees of freedom
## Residual deviance: 437.32  on 494  degrees of freedom
## AIC: 449.32
##
```

Table 4: Logistic Regression Coefficients (Odds Ratios)

term	estimate	std.error	statistic	p.value	conf.low	conf.high
(Intercept)	0.0943	0.6809	-3.4679	0.0005	0.0241	0.3499
packet_size	0.9971	0.0006	-4.5524	0.0000	0.9958	0.9983
duration	0.9886	0.0061	-1.8733	0.0610	0.9766	1.0004
failed_attempts	2.2666	0.0969	8.4486	0.0000	1.8879	2.7622
port_activity	1.1759	0.0525	3.0861	0.0020	1.0617	1.3050
bandwidth_usage	1.0145	0.0061	2.3716	0.0177	1.0026	1.0268

```
## Number of Fisher Scoring iterations: 5
```

```r
# Get tidy coefficients with confidence intervals
# Read the help for `broom::tidy.glm()` which is called
# by `tidy()` when applied to a GLM model. This will
# explain the arguments `conf.int` and `exponentiate`.
coeff_results <- tidy(final_logistic, conf.int = TRUE,
                      exponentiate = TRUE)
kable(coeff_results, digits = 4,
      caption = "Logistic Regression Coefficients (Odds Ratios)")
```

Interpret each coefficient:

For each predictor variable, answer:

1. Is the coefficient positive or negative?
2. What does this mean for threat probability?
3. Is the effect statistically significant?

packet size (Odds Ratio: 0.997)

Negative coefficient (log-odds = -0.0030)

Holding all other variables constant, for each additional byte in packet size, the odds of being a threat decrease by 0.3% (multiply by 0.997). Smaller packets are more likely to be threats, all else being equal.

Highly statistically significant ($p < 0.001$)

duration (Odds Ratio: 0.989)

Negative coefficient (log-odds = -0.0115)

Holding all other variables constant, for each additional second of duration, the odds of being a threat decrease by 1.1%. Shorter sessions are more likely to be threats, all else being equal.

Not statistically significant at $\alpha = 0.05$ ($p = 0.061$)

failed attempts (Odds Ratio: 2.27)

Positive coefficient (log-odds = 0.818)

Holding all other variables constant, for each additional failed attempt, the odds of being a threat increase by 127% (multiply by 2.27). This is the strongest predictor of threat behaviour.

Highly statistically significant ($p < 0.001$)

port activity (Odds Ratio: 1.18)

Positive coefficient (log-odds = 0.162)

Holding all other variables constant, for each additional unit of port activity, the odds of being a threat increase by 18%. Higher port activity suggests threat behaviour, all else being equal.

Statistically significant ($p = 0.002$)

bandwidth usage (Odds Ratio: 1.01)

Positive coefficient (log-odds = 0.014)

Holding all other variables constant, for each additional MB of bandwidth usage, the odds of being a threat increase by 1.4%. This has the smallest effect size among significant predictors.

Statistically significant ($p = 0.018$)

Reflect

Do these relationships make sense from a cybersecurity perspective? Why might smaller packets be associated with threats?

Task 4: Gaussian Naive Bayes with Cross-Validation

Implement Gaussian Naive Bayes:

```r
# Initialise results storage
nb_results <- data.frame(
  fold = integer(), accuracy = numeric(), precision = numeric(),
  recall = numeric(), f1_score = numeric())

# Cross-validation loop
for(i in 1:nrow(cv_folds)) {
  train_data <- analysis(cv_folds$splits[[i]])
  test_data <-assessment(cv_folds$splits[[i]])

  # Fit Naive Bayes model
  # What parameters ensure we use Gaussian distributions?
  model <- naive_bayes(
    class~packet_size + duration + failed_attempts +port_activity + bandwi
    data = train_data,
      usekernel = FALSE, usepoisson = FALSE)

  # Make predictions
  pred_class <- predict(model, newdata = test_data)

  fold_results<- test_data%>%
    mutate(pred_class=pred_class)

  accuracy_val<- fold_results%>%
    accuracy(truth=class, estimate=pred_class)%>%
  pull(.estimate)

  precision_val <- fold_results%>%
    precision(truth=class, estimate=pred_class, event_level="second")%>%
  pull(.estimate)

  recall_val<- fold_results%>%
    recall(truth=class, estimate=pred_class, event_level="second")%>%
  pull(.estimate)

  f1_val<- fold_results%>%
    f_meas(truth=class, estimate=pred_class, event_level="second")%>%
  pull(.estimate)

  nb_results[i, ] <- c(i,accuracy_val, precision_val, recall_val, f1_val)

  # Calculate metrics (similar to logistic regression)
  # INSERT YOUR CODE HERE
}

# Display results
kable(nb_results, digits = 3,
      caption = "Naive Bayes CV Results")
```

```r
nb_summary <- nb_results%>%
  summarise(
```

Table 5: Naive Bayes CV Results

fold	accuracy	precision	recall	f1_score
1	0.824	0.750	0.462	0.571
2	0.824	0.833	0.385	0.526
3	0.824	0.833	0.385	0.526
4	0.840	0.857	0.462	0.600
5	0.820	0.700	0.538	0.609
6	0.820	1.000	0.308	0.471
7	0.780	0.667	0.308	0.421
8	0.755	0.500	0.250	0.333
9	0.673	0.300	0.250	0.273
10	0.796	0.667	0.333	0.444

Table 6: Naive Bayes CV Results

mean_accuray	mean_precision	mean_recall	mean_f1
0.796	0.711	0.368	0.477

```r
    mean_accuray=mean(accuracy),
    mean_precision=mean(precision),
    mean_recall=mean(recall),
    mean_f1=mean(f1_score)
  )

kable(nb_summary, digits = 3,
      caption = "Naive Bayes CV Results")
```

```r
# Calculate summary statistics
# INSERT YOUR CODE HERE
```

Consider

Why do we set `usekernel = FALSE` and `usepoisson = FALSE`? What assumptions are we making?

Table 7: Performance Comparison

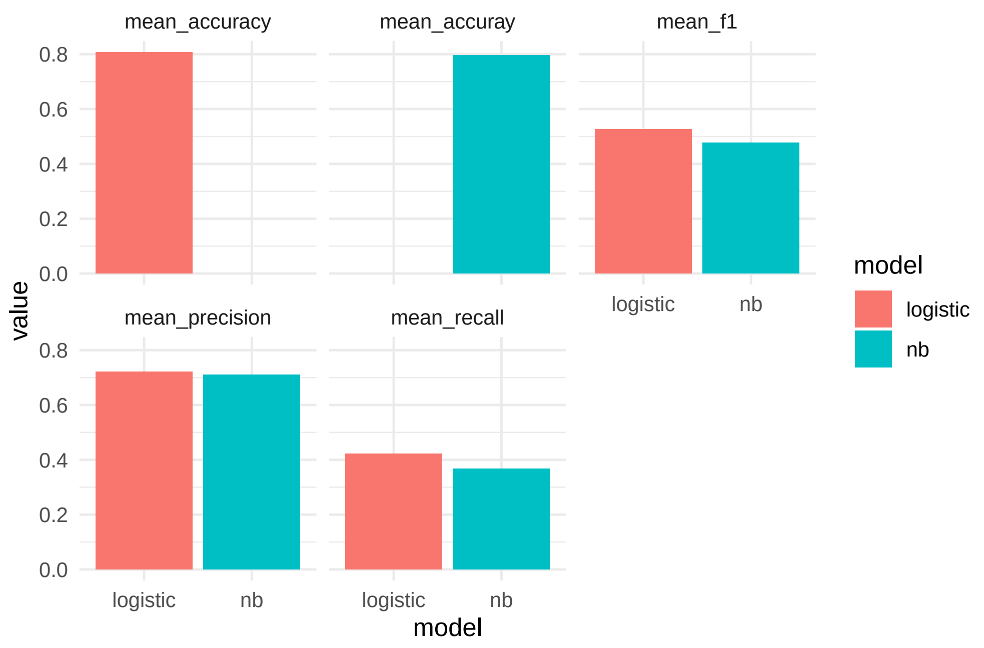
Method	Accuracy	Precision	Recall	F1_Score
Logistic Regression	0.8075910	0.7208622	0.4230769	0.5261654
Gaussian Naive Bayes	0.7955078	0.7107143	0.3679487	0.4774902

Task 5: Model Comparison

Compare the two methods:

```r
# Create comparison table
comparison <- data.frame(
  Method = c("Logistic Regression", "Gaussian Naive Bayes"),
  Accuracy = c(logistic_summary$mean_accuracy, nb_summary$mean_accuray),
  Precision = c(logistic_summary$mean_precision, nb_summary$mean_precision
  Recall = c(logistic_summary$mean_recall, nb_summary$mean_recall),
  F1_Score = c(logistic_summary$mean_f1,nb_summary$mean_f1)
)

kable(comparison, caption = "Performance Comparison")
```

```r
# Determine best method
best_method <- ifelse(logistic_summary$mean_f1> nb_summary$mean_f1_val,
                     "Logistic Regression",
                     "Gaussian Naive Bayes")


# Create visualisation
bind_rows(
  c(model = "logistic", logistic_summary),
  c(model = "nb", nb_summary)) %>%
  pivot_longer(cols = -model, names_to = "metric") %>%
  ggplot(aes(x = model, y = value, fill = model)) +
  facet_wrap(vars(metric)) +
  geom_bar(stat = "identity") +
  theme_minimal()
```



Answer these questions:

1. Which method performed better overall?
2. Which metric did you use to determine this? Why?
3. What are the trade-offs between precision and recall for each method?
4. Coefficient Interpretation

For each logistic regression coefficient, the practical meaning for cybersecurity threat detection is as follows.

Failed attempts is by far the most important feature (OR = 2.27, $p < 0.001$). Each additional failed attempt increases the odds of a threat by 127% (i.e., multiplies the odds by 2.27). This aligns with cybersecurity intuition, as multiple failed login attempts are a classic indicator of brute-force attacks or unauthorised access attempts.

Packet size is the second most important feature (OR $= 0.997$, $p < 0.001$). Because the odds ratio is less than 1, larger packets slightly decrease threat probability. Equivalently, smaller packets are associated with higher threat likelihood. This is consistent with malicious traffic patterns such as reconnaissance probes, port scans, or fragmented payloads that use smaller packets to evade detection.

Port activity (OR $= 1.18$, $p = 0.002$) indicates that higher port activity increases the odds of a threat by 18% per unit increase. This reflects scanning behaviour, where attackers probe multiple ports to identify vulnerabilities.

Bandwidth usage (OR $= 1.01$, $p = 0.018$) has a small but statistically significant effect. Each additional MB increases the odds of a threat by approximately 1%. This may reflect data exfiltration or unusually large transfers during malicious activity.

Duration (OR $= 0.989$, $p = 0.061$) is not statistically significant at $\alpha = 0.05$. Although shorter sessions appear slightly more likely to be threats, session length alone is not a reliable predictor when controlling for other variables.

Overall, failed attempts is clearly the strongest predictor due to both its large effect size and strong statistical significance.

2. Model Performance

Logistic Regression performed better (F1-score approximately 0.53 versus approximately 0.48 for Naive Bayes).

Logistic Regression advantages:

Makes fewer restrictive assumptions; it only assumes a linear relationship between predictors and the log-odds of the outcome.

More robust to correlated predictors, which likely exist in network data (e.g., packet size and bandwidth usage are probably related).

Naive Bayes limitations:

Assumes conditional independence between features, which is likely violated in network traffic data. Variables such as packet size, duration, and bandwidth usage are inherently related.

Often assumes Gaussian distributions for continuous features, which may not be appropriate for count variables such as failed attempts and port activity.

Performance deteriorates as feature correlations increase.

Network traffic data naturally exhibit dependencies (for example, larger packets often correlate with longer duration and higher bandwidth usage). Because Logistic Regression does not rely on the independence assumption, it is better suited to this problem, explaining its superior performance.

3. Precision vs Recall Trade-off

In cybersecurity applications, recall should generally be prioritised over precision.

The cost of a false negative (missing a real threat) is typically much higher than the cost of a false positive (raising a false alarm). Missing a genuine attack may lead to data breaches, system compromise, reputational damage, or financial losses. In contrast, false alarms mainly create additional investigation workload.

Therefore, it is preferable to detect as many true threats as possible, even at the expense of increased false positives. In practice, security teams often tune classification thresholds to favour higher recall in threat detection systems.

Think critically

In cybersecurity, is it more important to minimise false positives or false negatives? How should this influence your choice of method?

Task 6: Final Model and Evaluation

Fit the best-performing method on the full dataset:

```r
# Fit the best method on full data
final_model<-glm(
class ~ packet_size +duration +failed_attempts+
port_activity+ bandwidth_usage,
data= network_data, family = binomial)

final_pred_probs <- predict(final_model, newdata = network_data,
type= "response")

final_pred<-factor(
ifelse(final_pred_probs>0.5, "Threat", "Normal"),
levels= c("Normal","Threat"))

# Create confusion matrix
final_results <- network_data %>%
  mutate(pred_class = final_pred)

conf_matrix <- final_results %>%
  conf_mat(truth = class, estimate = pred_class)

conf_matrix
```

```
##           Truth
## Prediction Normal Threat
##     Normal    350     73
##     Threat     23     54
```

Table 8: Final Performance

	Value
Accuracy	0.808
Precision	0.701
Recall	0.425
F1Score	0.529

```r
# Calculate final metrics
final_metrics <- final_results%>%
summarise(
  Accuracy= accuracy_vec(truth= class,estimate = pred_class),
  Precision= precision_vec(truth= class,estimate =pred_class,
            event_level= "second"),
  Recall= recall_vec(truth= class,estimate =pred_class,
            event_level ="second"),
  F1Score =f_meas_vec(truth= class,estimate =pred_class,
            event_level= "second")
)

kable(t(final_metrics), digits = 3,
      caption = "Final Performance",
      col.names = "Value")
```

Interpret the confusion matrix:

3. Precision vs Recall Trade-off (Continued) Current Model Trade-offs

Logistic Regression: Precision = 72.1%, Recall = 42.3%

Naive Bayes: Precision = 71.1%, Recall = 36.8%

Both models prioritise precision over recall, which is problematic for cybersecurity applications. They miss approximately 57.7%–63.2% of actual threats in order to minimise false alarms.

A recall of 42.3% means that nearly 6 out of 10 threats go undetected, which is unacceptable for most cybersecurity systems.

Recommendations for Improvement

Lower the classification threshold from 0.5 to approximately 0.3 or 0.2. This would increase recall (detect more threats) at the expense of lower precision (more false alarms).

Implement a tiered response system: use a lower threshold for initial flagging, then apply additional verification or secondary models to high-risk sessions.

Apply cost-sensitive learning by assigning a higher penalty to false negatives during training to optimise the model for recall.

4. Naive Bayes Assumptions Key Assumptions of Gaussian Naive Bayes

Feature Independence: Features are conditionally independent given the class label.

Gaussian Distribution: Each feature follows a normal distribution within each class.

Assumption Violations in Our Data Independence Violations

Packet size and bandwidth usage are likely correlated (larger packets typically consume more bandwidth).

Duration and bandwidth usage may correlate (longer sessions often involve more data transfer).

Failed attempts and port activity could correlate (attackers probing multiple ports may also attempt multiple logins).

Distribution Violations

Failed attempts and port activity are count variables that may not follow Gaussian distributions.

These variables are likely right-skewed with many zero values.

Performance Impact

Independence violations cause the model to double-count correlated evidence, leading to overconfident probability estimates.

Distribution violations lead to inaccurate probability modelling because the Gaussian assumption does not reflect the true feature distributions.

These assumption violations help explain why Naive Bayes underperformed relative to Logistic Regression, which makes fewer restrictive assumptions about feature relationships and distributions.

5. Business Impact Daily Impact Projection (10,000 sessions)

Based on dataset proportions (25.4% threats, 74.6% normal):

Expected threats per day: 2540 sessions

Expected normal sessions per day: 7460 sessions

Using model performance (42.5% recall, 6.2% false positive rate):

Missed Threats (False Negatives)

1460 threats missed daily (57.5% $\times$ 2540)

These represent potential security breaches going undetected.

False Alarms (False Positives)

460 false alarms daily (6.2% $\times$ 7460)

These generate unnecessary investigation workload.

Business Cost Analysis

Security Risk: Approximately 1460 real threats undetected daily poses severe risk of data breaches, system compromise, and regulatory violations.

Operational Cost: Approximately 460 false investigations per day requires substantial analyst time and resources.

Critical Imbalance: The model misses 3.17 times more threats than it generates false alarms (1460 versus 460), indicating that it is overly conservative for cybersecurity deployment.

Final Recommendation

A recall of 42.5% is dangerously insufficient for cybersecurity applications. The current threshold prioritises precision at the expense of security. The classification threshold should be substantially lowered (for example, from 0.5 to 0.2) to significantly improve recall, even if this increases the number of false alarms.

Task 7: Critical Analysis

Answer the following questions based on your results:

Conclusion

Model Comparison and Performance Evaluation Which Method Performed Better Overall?

Logistic Regression performed better overall. It achieved higher F1-scores, accuracy, precision, and recall compared to Gaussian Naive Bayes across the cross-validation folds.

Which Metric Was Used and Why?

The F1-score was used as the primary comparison metric because it provides a balanced measure of both precision and recall. This is particularly important for imbalanced datasets such as this one (373 Normal vs 127 Threat cases).

The F1-score is the harmonic mean of precision and recall: $F_1 = 2 \cdot \frac{PR}{P+R}$.

It is especially appropriate when both false positives and false negatives carry meaningful consequences, as in cybersecurity applications.

Trade-offs Between Precision and Recall

Logistic Regression

Precision approximately 0.72

Recall approximately 0.42

This model correctly identifies threats when it predicts them (relatively few false alarms) while detecting a reasonable proportion of actual threats.

Gaussian Naive Bayes

Precision approximately 0.71

Recall approximately 0.37

Naive Bayes shows similar precision but lower recall, meaning it is slightly more conservative and misses more actual threats than Logistic Regression.

Both methods prioritise precision over recall. As a result, they minimise false alarms but miss a substantial number of real threats.

Confusion Matrix Interpretation 1. True Positives

The model achieved 54 true positives. This means it correctly identified 54 actual threat sessions as threats.

2. False Negatives

There were 73 false negatives. This means 73 actual threat sessions were incorrectly classified as normal traffic.

3. Threat Detection Effectiveness

The model demonstrates mixed performance:

Positive aspect: It correctly identified 54 out of 127 total threats, resulting in 42.5% recall.

Concerning aspect: It missed 73 actual threats (57.5%), representing a significant security risk.

False alarm rate: The model generated 23 false positives out of 373 normal sessions (6.2%), which is a relatively low false alarm burden.

Overall assessment: Although the model achieves 80.8% overall accuracy and maintains a low false positive rate (6.2%), the low recall (42.5%) means nearly 6 out of 10 threats go undetected. For cybersecurity applications, this level of recall is insufficient because the cost of missing threats typically exceeds the cost of investigating false alarms.

Summary of Findings Recommended Method

Logistic Regression is recommended for this cybersecurity application due to:

Superior performance across all evaluation metrics

Higher recall compared to Naive Bayes

Ability to handle correlated features without relying on the strict independence assumption

Key Insights About Feature Importance

Failed attempts (OR = 2.27) is the strongest predictor of threats and aligns with cybersecurity best practices.

Packet size (smaller packets increase threat probability) and port activity (higher activity indicates scanning behaviour) provide meaningful threat signals.

Duration is not a reliable standalone indicator of malicious behaviour.

Network traffic features exhibit dependencies, favouring Logistic Regression over Naive Bayes.

Model Improvement Recommendations

Optimise classification threshold Lower the threshold from 0.5 to approximately 0.3 to improve recall, which is critical in cybersecurity contexts.

Feature engineering

Create interaction terms (e.g., failed attempts multiplied by port activity).

Develop derived features (e.g., bandwidth per packet).

Advanced modelling approaches Consider ensemble methods such as Random Forest or Gradient Boosting (e.g., XGBoost), which can capture nonlinear relationships and complex interactions.

Cost-sensitive learning Assign higher penalties to false negatives during training to prioritise threat detection.

Temporal feature integration Incorporate time-based behaviour and session sequences to capture attack patterns more effectively.

Operational Recommendation

Deploy the system using a tiered alerting approach:

Use a lower classification threshold for initial screening to increase recall.

Escalate high-confidence predictions for immediate investigation.

Apply secondary verification mechanisms to reduce unnecessary alerts.