Regression and Classification: A Comprehensive Analysis

Gabriele Durante

In this report, we aim to explore various statistical learning methodologies applied to regression and classification problems.

In the first part of the report, we examine linear regression models applied to aquatic toxicity data. This involves both traditional linear effects and alternative representations using dummy encoding, providing insight into how these transformations impact model fit and predictive accuracy. We apply techniques such as backward elimination and forward selection to perform variable selection, comparing criteria like AIC and BIC. Moreover, we introduce regularization techniques, specifically ridge regression, to address potential overfitting, followed by a discussion on parameter optimization via cross-validation and bootstrap methods. To extend our analysis to non-linear modeling, we apply generalized additive models (GAMs) with smoothing splines to capture more complex relationships within the data. Regression trees are explored, where we employ cost-complexity pruning to optimize tree size, emphasizing the balance between model complexity and prediction accuracy.

In the second part, we shift to a classification problem using the Pima Indians Diabetes dataset. Here, we experiment with k-nearest neighbors (k-NN), generalized additive models, classification trees, bagged trees, random forests, and neural networks. Each method is rigorously evaluated, using cross-validation techniques to assess generalization error and performance trade-offs. The report concludes with a comparative analysis of these models, offering insights into the advantages and limitations of different statistical learning approaches within the context of high-dimensional data.

Regression Model Comparison for Predicting Aquatic Toxicity

In this study, we investigate the acute aquatic toxicity of various organic molecules, as quantified by the lethal concentration (LC50) that induces mortality in 50% of the planktonic crustacean Daphnia magna over a 48-hour exposure period. The objective is to develop predictive models that leverage molecular descriptors to ascertain the toxicity levels of these compounds. To this end, we utilize a dataset comprising 546 observations, which is sourced from the UCI Machine Learning Repository. The dataset encompasses eight key molecular descriptors identified as significant predictors of LC50:

- TPSA: Topological Polar Surface Area, calculated via a contribution method considering the presence of nitrogen, oxygen, potassium, and sulfur.
- SAacc: Van der Waals Surface Area (VSA) of hydrogen bond acceptor atoms.
- H050: The count of hydrogen atoms bonded to heteroatoms.
- MLOGP: A measure of lipophilicity, serving as a critical determinant of narcosis.
- RDCHI: A topological index encapsulating information related to molecular size and branching.
- GATS1p: An indicator of molecular polarizability.
- nN: The total number of nitrogen atoms within the molecule.
- C040: The count of specific carbon atom types, including those found in esters, carboxylic acids, thioesters, carbamic acids, and nitriles.

```r
summary(data)
```

```
##       TPSA            SAacc             H050            MLOGP
##  Min.   :  0.00   Min.   :  0.00   Min.   : 0.0000   Min.   :-6.446
##  1st Qu.: 15.79   1st Qu.: 11.00   1st Qu.: 0.0000   1st Qu.: 1.232
##  Median : 40.46   Median : 42.68   Median : 0.0000   Median : 2.273
##  Mean   : 48.47   Mean   : 58.87   Mean   : 0.9377   Mean   : 2.313
##  3rd Qu.: 70.02   3rd Qu.: 77.49   3rd Qu.: 1.0000   3rd Qu.: 3.393
##  Max.   :347.32   Max.   :571.95   Max.   :18.0000   Max.   : 9.148
##      RDCHI           GATS1p             nN               C040
##  Min.   :1.000   Min.   :0.281   Min.   : 0.000   Min.   : 0.0000
##  1st Qu.:1.975   1st Qu.:0.737   1st Qu.: 0.000   1st Qu.: 0.0000
##  Median :2.344   Median :1.020   Median : 1.000   Median : 0.0000
##  Mean   :2.492   Mean   :1.046   Mean   : 1.004   Mean   : 0.3535
##  3rd Qu.:2.911   3rd Qu.:1.266   3rd Qu.: 2.000   3rd Qu.: 0.0000
##  Max.   :6.439   Max.   :2.500   Max.   :11.000   Max.   :11.0000
##       LC50
##  Min.   : 0.122
##  1st Qu.: 3.602
##  Median : 4.516
##  Mean   : 4.658
##  3rd Qu.: 5.607
##  Max.   :10.047
```

```r
colSums(is.na(data))
```

```
##   TPSA  SAacc   H050  MLOGP  RDCHI GATS1p     nN   C040   LC50
##      0      0      0      0      0      0      0      0      0
```

With a complete dataset comprising 546 observations, we can ensure that our modeling efforts are not hindered by data gaps. This completeness facilitates a more accurate assessment of the relationships between the molecular descriptors and LC50, allowing for a more confident interpretation of the findings and their implications in predicting aquatic toxicity.

Linear and Dummy Encoding Approach

To evaluate the predictive models effectively, we partition the dataset into training and testing subsets. A random seed is set using `set.seed(2024)` to ensure reproducibility. The training set is constructed by sampling approximately two-thirds of the total observations with the command `index <- sample(1:nrow(data), size = round(2/3 * nrow(data)))`.

- Training Data: `trainData <- data[index, ]` comprises the selected observations for model fitting.
- Testing Data: `testData <- data[-index, ]` contains the remaining observations for model evaluation.

```r
# build the dataset for the training
set.seed(2024)
index <- sample(1:nrow(data), size = round(2/3 * nrow(data)))
trainData <- data[index, ]
testData <- data[-index, ]
```

Linear Model

First we fitted a linear regression model to predict LC50 using molecular descriptors from the training dataset. The model's performance is summarized as follows:

```r
set.seed(2024)
### Modeling Count Variables Directly as Linear Effects
model1 <- lm(LC50 ~., data = trainData)

pred_model1_train <- predict(model1, newdata = trainData)
pred_model1_test <- predict(model1, newdata = testData)
```

```r
error_train <- mean((pred_model1_train - trainData$LC50)^2)
error_test <- mean((pred_model1_test - testData$LC50)^2)
```

```r
cat("Training Error (Linear):", error_train, "\n")
```

```
## Training Error (Linear): 1.377162
```

```r
cat("Test Error (Linear):", error_test, "\n")
```

```
## Test Error (Linear): 1.556978
```

```r
summary(model1)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1890 -0.7482 -0.1153  0.6079  3.7987
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.875587   0.298304   9.640  < 2e-16 ***
## TPSA         0.026589   0.003220   8.258 2.96e-15 ***
## SAacc       -0.012127   0.002522  -4.809 2.25e-06 ***
## H050         0.031343   0.070945   0.442  0.65891
## MLOGP        0.496920   0.077134   6.442 3.83e-10 ***
## RDCHI        0.308016   0.166170   1.854  0.06462 .
## GATS1p      -0.537871   0.187291  -2.872  0.00433 **
## nN          -0.198318   0.057041  -3.477  0.00057 ***
## C040        -0.055902   0.090200  -0.620  0.53582
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.188 on 355 degrees of freedom
## Multiple R-squared:  0.4721, Adjusted R-squared:  0.4602
## F-statistic: 39.68 on 8 and 355 DF,  p-value: < 2.2e-16
```

The R-squared value of 0.472 indicates that while the linear model captures some of the variability in the data, it leaves a considerable amount unexplained. Highly significant predictors, such as TPSA ($p < 2e-16$) and MLOGP ($p < 3.83e-10$), reveal strong relationships with LC50, indicating that increased topological polar surface area and greater lipophilicity correlate with higher toxicity levels.

Dummy Linear Model

After, model dummy encoding was applied to the count variables **nN**, **C040**, and **H050** in both the training and test datasets to transform these continuous variables into binary indicators. Specifically, values greater than zero were recoded to 1, while values of zero were recoded to 0. This approach allows the linear model to assess the presence or absence of these variables as factors influencing acute aquatic toxicity. Predictions were generated for both the training and test datasets, enabling the calculation of the training error and test error.

```r
### Dummy Encoding for Count Variables
train_dummy <- trainData
test_dummy <- testData

# dummy encoding
```

```r
train_dummy$nN <- ifelse(train_dummy$nN > 0, 1, 0)
test_dummy$nN <- ifelse(test_dummy$nN > 0, 1, 0)
test_dummy$C040 <- ifelse(test_dummy$C040 > 0, 1, 0)
train_dummy$C040 <- ifelse(train_dummy$C040 > 0, 1, 0)
test_dummy$H050 <- ifelse(test_dummy$H050 > 0, 1, 0)
train_dummy$H050 <- ifelse(train_dummy$H050 > 0, 1, 0)

model2 <- lm(train_dummy$LC50 ~ ., data = train_dummy)

pred_model2_train <- predict(model2, newdata = train_dummy)
pred_model2_test <- predict(model2, newdata = test_dummy)

error_train_dummy <- mean((pred_model2_train - train_dummy$LC50)^2)
error_test_dummy <- mean((pred_model2_test - test_dummy$LC50)^2)
```

```r
cat("Training Error (Dummy):", error_train_dummy, "\n")
```

```
## Training Error (Dummy): 1.424236
```

```r
cat("Test Error (Dummy):", error_test_dummy, "\n")
```

```
## Test Error (Dummy): 1.616825
```

```r
summary(model2)
```

```
##
## Call:
## lm(formula = train_dummy$LC50 ~ ., data = train_dummy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0143 -0.7807 -0.1313  0.6313  3.7856
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.007963   0.305982   9.831  < 2e-16 ***
## TPSA         0.022380   0.003185   7.026 1.09e-11 ***
## SAacc       -0.010184   0.002193  -4.643 4.83e-06 ***
## H050        -0.097746   0.154467  -0.633  0.52727
## MLOGP        0.502377   0.076886   6.534 2.22e-10 ***
## RDCHI        0.262639   0.167774   1.565  0.11837
## GATS1p      -0.559217   0.185302  -3.018  0.00273 **
## nN          -0.054983   0.149020  -0.369  0.71237
## C040        -0.162055   0.162002  -1.000  0.31783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.208 on 355 degrees of freedom
## Multiple R-squared:  0.454,  Adjusted R-squared:  0.4417
## F-statistic:  36.9 on 8 and 355 DF,  p-value: < 2.2e-16
```

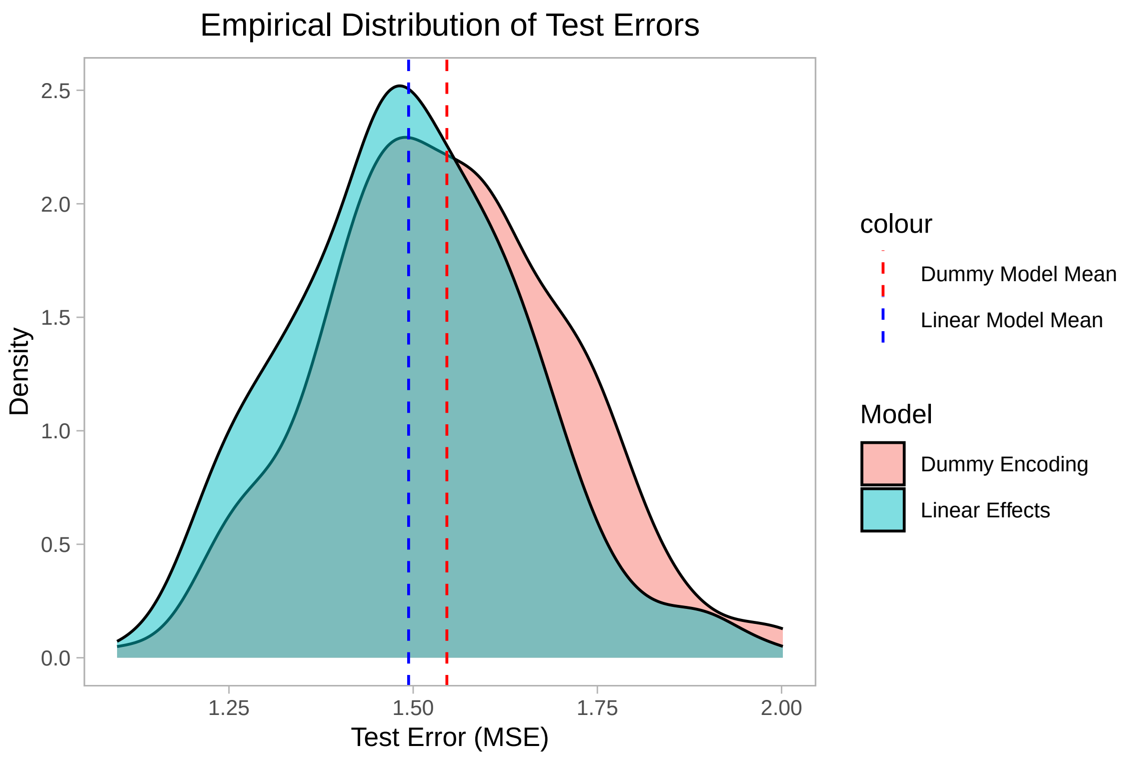
The training error for the dummy-encoded model is 1.424, while the test error is 1.617. Compared to the previous linear model, which yielded a training error of 1.377 and a test error of 1.557, this model exhibits slightly higher error rates on both the training and test datasets. This increase suggests that the dummy encoding may not have improved the model's predictive performance. This could indicate a potential loss of information due to the transformation of continuous variables into binary indicators, which could limit the model's ability to capture the nuances of the underlying relationships between predictors and the response variable.

The overall model performance is reflected in the Multiple R-squared value of 0.454, while the adjusted R-squared of 0.4417 accounts for the number of predictors in the model. The F-statistic (36.9) with a p-value $< 2.2e-16$ indicates that the model is statistically significant.

Empirical Error Distribution Analysis in Regression Models

The function `perform_analysis` is defined to conduct repeated evaluations of the two linear models defined before on the same dataset and returns the values of error test and train. The analysis is repeated 200 times using the `replicate` function, enabling the computation of average test errors for both models. The results are then visualized using a density plot, illustrating the empirical distribution of test errors for each model.

```r
library(ggplot2)

perform_analysis <- function(data) {
  index <- sample(1:nrow(data), size = round(2/3 * nrow(data)))
  trainData <- data[index, ]
  testData <- data[-index, ]

  # model 1
  model1 <- lm(LC50 ~., data = trainData)
  pred_model1_train_200 <- predict(model1, newdata = trainData)
  pred_model1_test_200 <- predict(model1, newdata = testData)
  error_train_200_lm <- mean((pred_model1_train_200 - trainData$LC50)^2)
  error_test_200_lm <- mean((pred_model1_test_200 - testData$LC50)^2)

  # model 2
  train_dummy <- trainData
  test_dummy <- testData

  # dummy encoding
  train_dummy$nN <- ifelse(train_dummy$nN > 0, 1, 0)
  test_dummy$nN <- ifelse(test_dummy$nN > 0, 1, 0)
  test_dummy$C040 <- ifelse(test_dummy$C040 > 0, 1, 0)
  train_dummy$C040 <- ifelse(train_dummy$C040 > 0, 1, 0)
  test_dummy$H050 <- ifelse(test_dummy$H050 > 0, 1, 0)
  train_dummy$H050 <- ifelse(train_dummy$H050 > 0, 1, 0)

  model2 <- lm(train_dummy$LC50 ~ ., data = train_dummy)

  pred_model2_train <- predict(model2, newdata = train_dummy)
  pred_model2_test <- predict(model2, newdata = test_dummy)

  error_train_200_dummy <- mean((pred_model2_train - train_dummy$LC50)^2)
  error_test_200_dummy <- mean((pred_model2_test - test_dummy$LC50)^2)

  return(c(error_test_200_lm, error_test_200_dummy))
}
```



The graph presents a density plot comparing the test error distributions for the two models. The aim should be illustrate the performance of the models on unseen data and their reliability, focusing on the variability and central tendency of their respective errors. The Linear Model exhibits a narrower distribution centered around a lower MSE compared to the Dummy Model, whose distribution appears wider and shifted slightly to the right, indicating higer average test error. Basically, the tigter distribution and lower mean MSE of the Linear Model suggest that it generalize better to the test set compared to the dummy model.

Variable Selection Methods in Linear Regression

```r
library(MASS)
library(leaps)

# Backward Elimination
full_model <- lm(LC50 ~ ., data = trainData)
backward_aic <- stepAIC(full_model, direction = "backward",
                        k = 2, trace = FALSE)
summary(backward_aic)
```

```
##
## Call:
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,
##     data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1632 -0.7485 -0.1143  0.6156  3.8169
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.970390   0.278237  10.676  < 2e-16 ***
## TPSA         0.026358   0.003125   8.435 8.35e-16 ***
## SAacc       -0.011638   0.001946  -5.980 5.40e-09 ***
## MLOGP        0.487552   0.073711   6.614 1.37e-10 ***
## RDCHI        0.290699   0.162996   1.783 0.075358 .
## GATS1p      -0.573581   0.179110  -3.202 0.001485 **
## nN          -0.197504   0.054796  -3.604 0.000357 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.187 on 357 degrees of freedom
## Multiple R-squared:  0.4707, Adjusted R-squared:  0.4618
## F-statistic: 52.91 on 6 and 357 DF,  p-value: < 2.2e-16
```

```
backward_bic <- stepAIC(full_model, direction = "backward",
                        k = log(nrow(trainData)), trace = FALSE)
summary(backward_bic)
```

```
##
## Call:
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + GATS1p + nN, data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0473 -0.7751 -0.0928  0.5879  3.7865
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.123831   0.265406  11.770  < 2e-16 ***
## TPSA         0.028625   0.002863   9.997  < 2e-16 ***
## SAacc       -0.010068   0.001741  -5.783  1.6e-08 ***
## MLOGP        0.594639   0.042886  13.866  < 2e-16 ***
## GATS1p      -0.466421   0.169244  -2.756 0.006152 **
## nN          -0.187790   0.054690  -3.434 0.000665 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.19 on 358 degrees of freedom
## Multiple R-squared:  0.466,  Adjusted R-squared:  0.4585
## F-statistic: 62.47 on 5 and 358 DF,  p-value: < 2.2e-16
```

```
# Forward Selection
null_model <- lm(LC50 ~ 1, data = trainData)
forward_aic <- stepAIC(null_model, direction = "forward",
                       scope = formula(full_model), k = 2, trace = FALSE)
summary(forward_aic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p + RDCHI,
##     data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1632 -0.7485 -0.1143  0.6156  3.8169
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.970390   0.278237  10.676  < 2e-16 ***
## MLOGP        0.487552   0.073711   6.614 1.37e-10 ***
## TPSA         0.026358   0.003125   8.435 8.35e-16 ***
## SAacc       -0.011638   0.001946  -5.980 5.40e-09 ***
## nN          -0.197504   0.054796  -3.604 0.000357 ***
## GATS1p      -0.573581   0.179110  -3.202 0.001485 **
## RDCHI        0.290699   0.162996   1.783 0.075358 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.187 on 357 degrees of freedom
## Multiple R-squared:  0.4707, Adjusted R-squared:  0.4618
## F-statistic: 52.91 on 6 and 357 DF,  p-value: < 2.2e-16
```

```r
forward_bic <- stepAIC(null_model, direction = "forward",
                       scope = formula(full_model), k = log(nrow(trainData)), trace = FALSE)
summary(forward_bic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p, data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0473 -0.7751 -0.0928  0.5879  3.7865
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.123831   0.265406  11.770  < 2e-16 ***
## MLOGP        0.594639   0.042886  13.866  < 2e-16 ***
## TPSA         0.028625   0.002863   9.997  < 2e-16 ***
## SAacc       -0.010068   0.001741  -5.783  1.6e-08 ***
## nN          -0.187790   0.054690  -3.434 0.000665 ***
## GATS1p      -0.466421   0.169244  -2.756 0.006152 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.19 on 358 degrees of freedom
## Multiple R-squared:  0.466,  Adjusted R-squared:  0.4585
## F-statistic: 62.47 on 5 and 358 DF,  p-value: < 2.2e-16
```

```r
# Compare Models
cat("Backward Elimination (AIC) Model:\n")
```

```
## Backward Elimination (AIC) Model:
```

```r
print(backward_aic$call)
```

```
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,
##     data = trainData)
```

```r
cat("\nBackward Elimination (BIC) Model:\n")
```

```
##
```

```
## Backward Elimination (BIC) Model:
```

```r
print(backward_bic$call)
```

```
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + GATS1p + nN, data = trainData)
```

```r
cat("\nForward Selection (AIC) Model:\n")
```

```
##
## Forward Selection (AIC) Model:
```

```r
print(forward_aic$call)
```

```
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p + RDCHI,
##     data = trainData)
```

```r
cat("\nForward Selection (BIC) Model:\n")
```

```
##
## Forward Selection (BIC) Model:
```

```r
print(forward_bic$call)
```

```
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p, data = trainData)
```

```r
# Model Comparisons
cat("\nModels Summary Comparison:\n")
```

```
##
## Models Summary Comparison:
```

```r
cat("AIC Backward:", AIC(backward_aic), "\n")
```

```
## AIC Backward: 1166.421
```

```r
cat("AIC Forward:", AIC(forward_aic), "\n")
```

```
## AIC Forward: 1166.421
```

```r
cat("BIC Backward:", BIC(backward_bic), "\n")
```

```
## BIC Backward: 1194.93
```

```r
cat("BIC Forward:", BIC(forward_bic), "\n")
```

```
## BIC Forward: 1194.93
```

Ridge Regression Model Optimization

```r
library(glmnet)
library(boot)

index <- sample(seq_len(nrow(data)), size = 2/3 * nrow(data))
trainData <- data[index, ]
testData <- data[-index, ]

X_train <- as.matrix(trainData[, -ncol(trainData)])
y_train <- trainData$LC50
X_test <- as.matrix(testData[, -ncol(testData)])
y_test <- testData$LC50
```

```r
lambda_grid <- 10^seq(3, -2, lenght = 100)

# Ridge regression (CV)
cv_ridge <- cv.glmnet(X_train, y_train, alpha = 0,
                      lambda = lambda_grid, nfolds = 10)
optimal_lambda_cv <- cv_ridge$lambda.min

cat("Optimal lambda from Cross-Validation:", optimal_lambda_cv, "\n")
```

```
## Optimal lambda from Cross-Validation: 0.01
```

```r
ridge_model <- glmnet(X_train, y_train, alpha = 0,
                      lambda = optimal_lambda_cv)
train_predictions_ridge <- predict(ridge_model, newx = X_train)
test_predictions_ridge <- predict(ridge_model, newx = X_test)
mse_train_ridge <- mean((y_train - train_predictions_ridge)^2)
mse_test_ridge <- mean((y_test - test_predictions_ridge)^2)
cat("Training Error (Ridge):", mse_train_ridge, "\n")
```

```
## Training Error (Ridge): 1.377282
```

```r
cat("Test Error (Ridge):", mse_test_ridge, "\n")
```

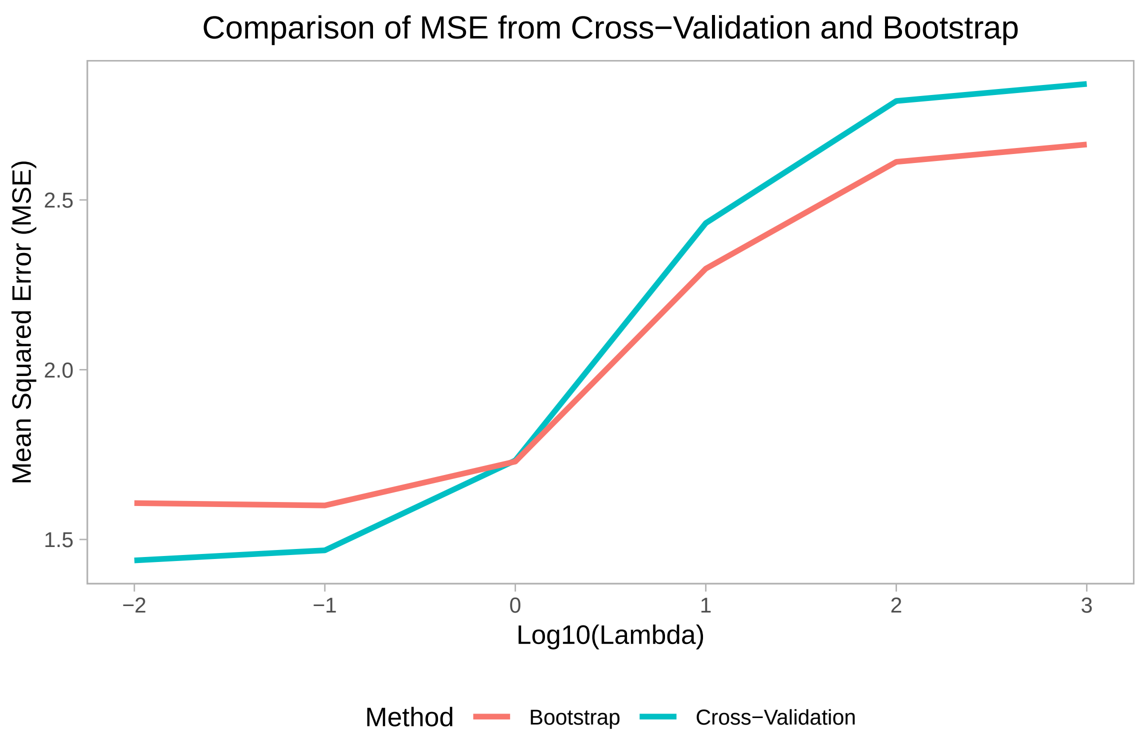
```
## Test Error (Ridge): 1.554029
```

```r
# Ridge regression (bootstrap)
bootstrap_mse <- function(data, indices, lambda) {
  # Create bootstrap sample
  bootstrap_sample <- data[indices, ]
  X_bootstrap <- as.matrix(bootstrap_sample[, -ncol(bootstrap_sample)])
  y_bootstrap <- bootstrap_sample$LC50

  model <- glmnet(X_bootstrap, y_bootstrap, alpha = 0, lambda = lambda)
  y_pred <- predict(model, s = lambda, newx = X_test)
  mse <- mean((y_test - y_pred)^2)
  return(mse)
}
# Bootstrap for multiple lambda values
bootstrap_results <- sapply(lambda_grid, function(lambda) {
  mse_values <- replicate(100, boot(trainData, bootstrap_mse, R = 1,
                                    lambda = lambda)$t)
  return(mean(mse_values))
})
# Optimal Lambda from Bootstrap
optimal_lambda_bootstrap <- lambda_grid[which.min(bootstrap_results)]
cat("Optimal lambda from Bootstrap:", optimal_lambda_bootstrap, "\n")
```

```
## Optimal lambda from Bootstrap: 0.1
```

```r
results_df <- data.frame(
  Lambda = lambda_grid,
  CV_MSE = sapply(lambda_grid,
                  function(l) mean(cv_ridge$cvm[cv_ridge$lambda == l])),
  Bootstrap_MSE = bootstrap_results
)
```

```r
# Plot
ggplot(results_df, aes(x = log10(Lambda)) ) +
  geom_line(aes(y = CV_MSE, color = "Cross-Validation"), size = 1) +
  geom_line(aes(y = Bootstrap_MSE, color = "Bootstrap"), size = 1) +
  labs(title = "Comparison of MSE from Cross-Validation and Bootstrap",
       x = "Log10(Lambda)",
       y = "Mean Squared Error (MSE)",
       color = "Method") +
  theme_light() +
  theme(legend.position = "bottom",
        panel.grid = element_blank(),
        plot.title = element_text(hjust = 0.5),
        text = element_text(size = 10))
```



Non-linear Modeling Using Generalized Additive Models

```r
sapply(trainData, function(x) length(unique(x)))
```

```
##   TPSA  SAacc   H050  MLOGP  RDCHI GATS1p     nN   C040   LC50
##    171    160      9    283    259    289      9      6    347
```

```r
library(mgcv)

set.seed(2024)
index <- sample(seq_len(nrow(data)), size = 2/3 * nrow(data))
trainData <- data[index, ]
testData <- data[-index, ]

X_train <- as.matrix(trainData[, -ncol(trainData)])
```

```r
y_train <- trainData$LC50
X_test <- as.matrix(testData[, -ncol(testData)])
y_test <- testData$LC50

# GAM less complexity (k = -1)
k = 1
gam_model_1 <- gam(LC50 ~ s(TPSA, k=k) + s(SAacc, k=k) + s(H050, k=k) +
                     s(MLOGP, k=k) + s(RDCHI, k=k) + s(GATS1p, k=k) +
                     s(nN, k=k) + s(C040, k=k), data = trainData)

pred_gam_train_1 <- predict(gam_model_1, newdata = trainData)
pred_gam_test_1 <- predict(gam_model_1, newdata = testData)
mse_train_gam_1 <- mean((y_train - pred_gam_train_1)^2)
mse_test_gam_1 <- mean((y_test - pred_gam_test_1)^2)

cat("Training Error (GAM - k=1):","\t",mse_train_gam_1, "\n")
```

```
## Training Error (GAM - k=1):   1.300005
```

```r
cat("Test Error (GAM - k=1):","\t",mse_test_gam_1, "\n")
```

```
## Test Error (GAM - k=1):   1.565394
```

```r
# GAM more complexity (k = 6)
k = 6
gam_model_2 <- gam(LC50 ~ s(TPSA, k=k) + s(SAacc, k=k) + s(H050, k=k) +
                     s(MLOGP, k=k) + s(RDCHI, k=k) + s(GATS1p, k=k) +
                     s(nN, k=k) + s(C040, k=k), data = trainData)

pred_gam_train_2 <- predict(gam_model_2, newdata = trainData)
pred_gam_test_2 <- predict(gam_model_2, newdata = testData)
mse_train_gam_2 <- mean((y_train - pred_gam_train_2)^2)
mse_test_gam_2 <- mean((y_test - pred_gam_test_2)^2)

cat("Training Error (GAM - k=6):","\t",mse_train_gam_2, "\n")
```

```
## Training Error (GAM - k=6):   1.147883
```

```r
cat("Test Error (GAM - k=6):","\t",mse_test_gam_2, "\n")
```

```
## Test Error (GAM - k=6):   1.634775
```

Regression Tree Model with Cost-Complexity Pruning

```r
library(rpart)
library(rpart.plot)

tree_model <- rpart(LC50 ~ ., data = trainData, method = "anova")

train_predictions_tree1 <- predict(tree_model, newdata = trainData)
test_predictions_tree1 <- predict(tree_model, newdata = testData)

mse_train_tree1 <- mean((trainData$LC50 - train_predictions_tree1)^2)
mse_test_tree1 <- mean((testData$LC50 - test_predictions_tree1)^2)
```

```r
printcp(tree_model)
```

```
##
## Regression tree:
## rpart(formula = LC50 ~ ., data = trainData, method = "anova")
##
## Variables actually used in tree construction:
## [1] GATS1p H050   MLOGP  nN     RDCHI  SAacc  TPSA
##
## Root node error: 949.5/364 = 2.6085
##
## n= 364
##
##          CP nsplit rel error  xerror     xstd
## 1  0.202576      0   1.00000 1.00521 0.081444
## 2  0.120003      1   0.79742 0.89217 0.074688
## 3  0.055504      2   0.67742 0.71621 0.060411
## 4  0.029750      4   0.56641 0.68528 0.057679
## 5  0.024807      5   0.53666 0.69253 0.060911
## 6  0.019644      6   0.51186 0.69168 0.061473
## 7  0.018758      7   0.49221 0.68003 0.061525
## 8  0.018127      8   0.47345 0.68003 0.061525
## 9  0.012598      9   0.45533 0.68531 0.063366
## 10 0.011999     10   0.44273 0.70330 0.069257
## 11 0.011431     11   0.43073 0.70041 0.069312
## 12 0.011033     13   0.40787 0.69407 0.069352
## 13 0.010592     14   0.39684 0.69141 0.068352
## 14 0.010000     15   0.38625 0.68556 0.068029
```

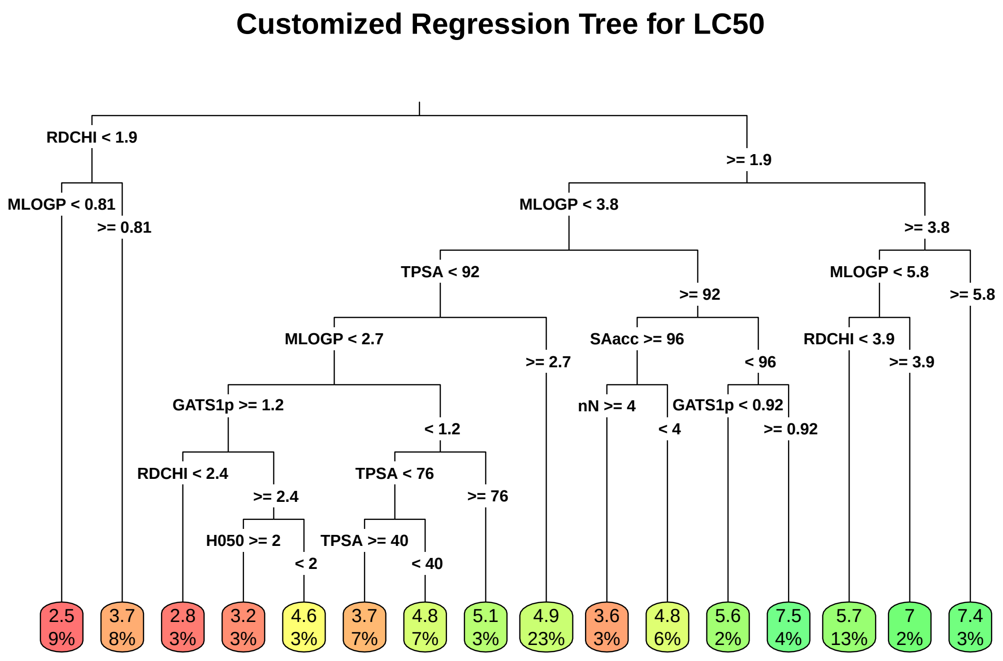
```r
cat("Training Error (Tree):", mse_train_tree1, "\n")
```

```
## Training Error (Tree): 1.007531
```

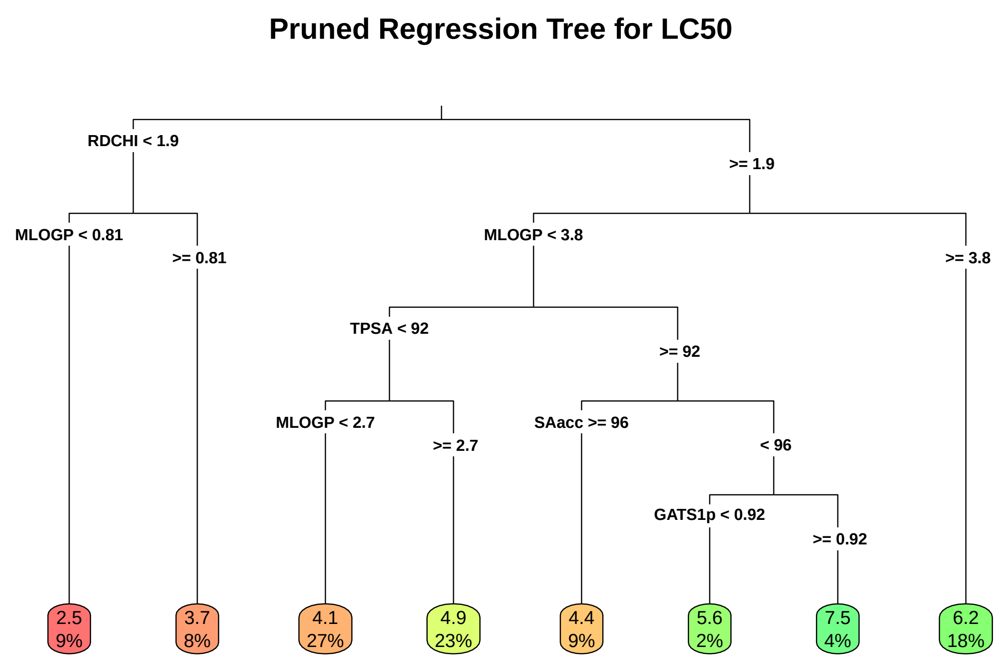
```r
cat("Test Error (Tree):", mse_test_tree1, "\n")
```

```
## Test Error (Tree): 2.053224
```

```r
rpart.plot(tree_model,
           main = "Customized Regression Tree for LC50",
           type = 3,
           fallen.leaves = TRUE,
           box.palette = "RdYlGn",
           cex = 0.6,
           tweak = 1,
           split.cex = 0.9,
           split.lty = 3,
)
```



```
optimal_cp <- tree_model$cptable[which.min(tree_model$cptable[,"xerror"]), "CP"]

pruned_tree <- prune(tree_model, cp = optimal_cp)
train_predictions_tree <- predict(pruned_tree, newdata = trainData)
test_predictions_tree <- predict(pruned_tree, newdata = testData)

mse_train_tree <- mean((trainData$LC50 - train_predictions_tree)^2)
mse_test_tree <- mean((testData$LC50 - test_predictions_tree)^2)

rpart.plot(pruned_tree,
           main = "Pruned Regression Tree for LC50",
           type = 3,
           fallen.leaves = TRUE,
           box.palette = "RdYlGn",
           cex = 0.6,
           tweak = 1,
           split.cex = 0.9,
           split.lty = 3,
)
```



```
printcp(pruned_tree)
```

```
##
## Regression tree:
## rpart(formula = LC50 ~ ., data = trainData, method = "anova")
##
## Variables actually used in tree construction:
## [1] GATS1p MLOGP  RDCHI  SAacc  TPSA
##
## Root node error: 949.5/364 = 2.6085
##
## n= 364
##
##         CP nsplit rel error  xerror     xstd
## 1 0.202576      0   1.00000 1.00521 0.081444
## 2 0.120003      1   0.79742 0.89217 0.074688
## 3 0.055504      2   0.67742 0.71621 0.060411
## 4 0.029750      4   0.56641 0.68528 0.057679
## 5 0.024807      5   0.53666 0.69253 0.060911
## 6 0.019644      6   0.51186 0.69168 0.061473
## 7 0.018758      7   0.49221 0.68003 0.061525
```

```
cat("Training Error (Tree):", mse_train_tree, "\n")
```

```
## Training Error (Tree): 1.283949
```

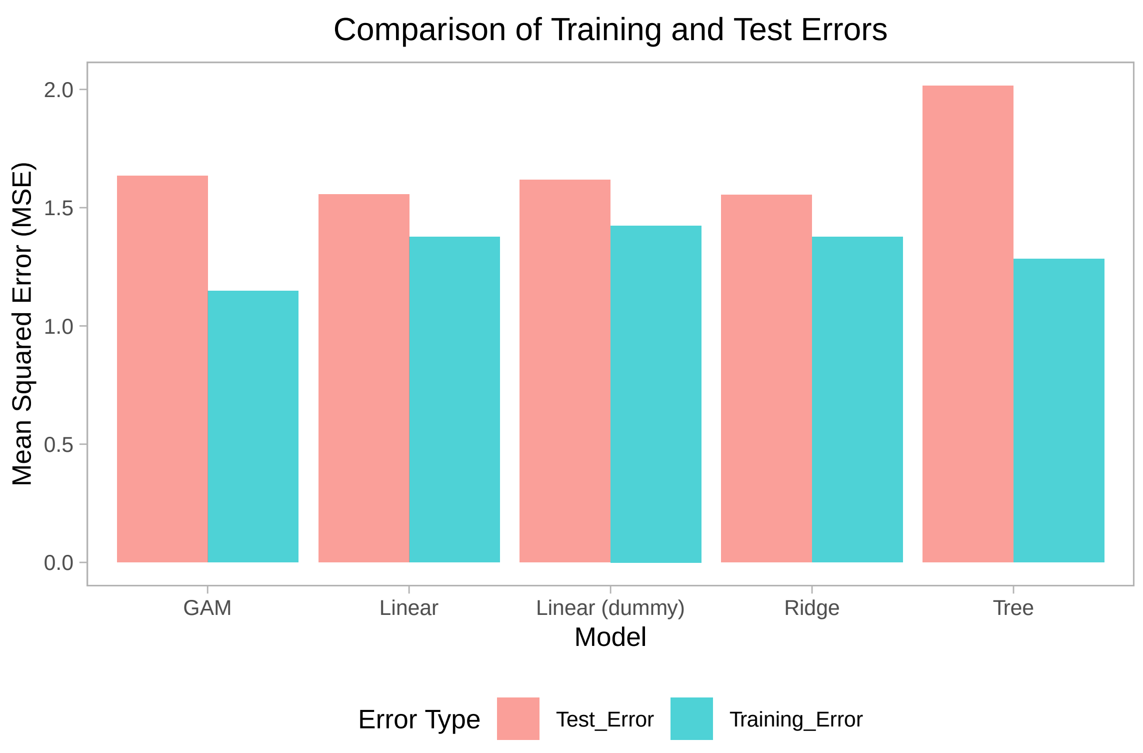
```
cat("Test Error (Tree):", mse_test_tree, "\n")
```

```
## Test Error (Tree): 2.016152
```

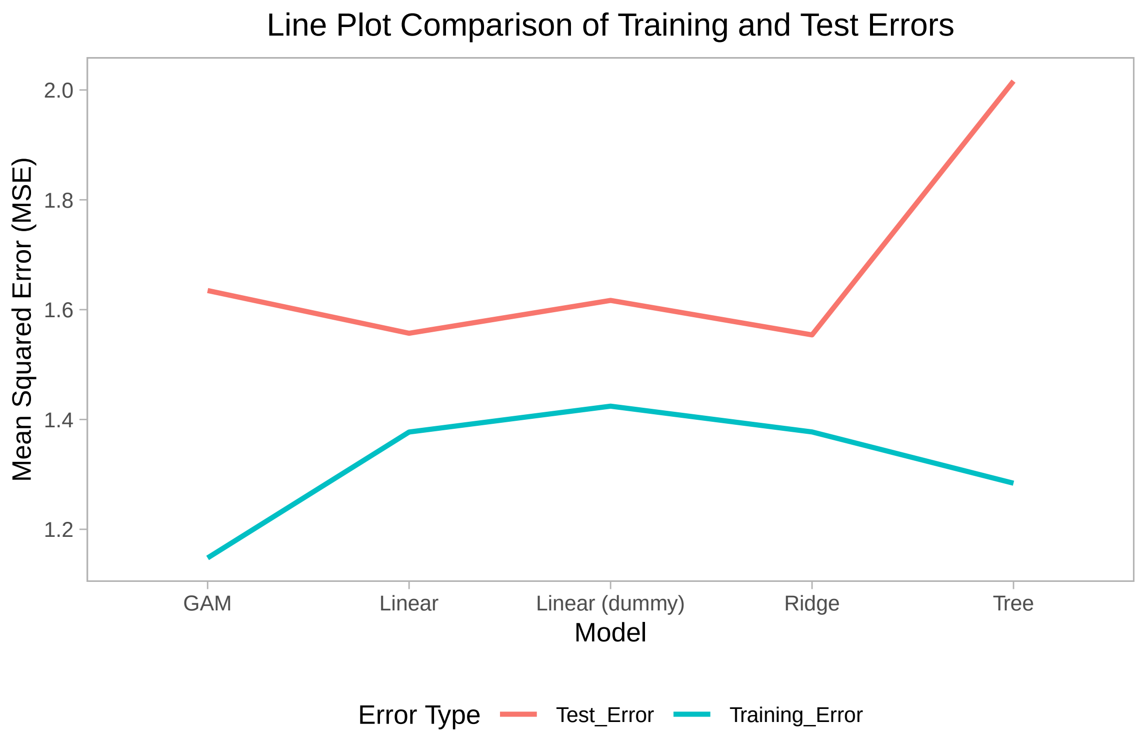
Comparative Analysis of Regression Models

Training and Test Error Assessment for Various Models

```r
library(tidyr)

model_comparison <- data.frame(
  Model = c("Linear", "Linear (dummy)", "Ridge", "GAM", "Tree"),
  Training_Error = c(error_train, error_train_dummy,
                     mse_train_ridge, mse_train_gam_2, mse_train_tree),
  Test_Error = c(error_test, error_test_dummy,
                 mse_test_ridge, mse_test_gam_2, mse_test_tree)
)

model_comparison_long <- pivot_longer(model_comparison,
                                      cols = c("Training_Error", "Test_Error"),
                                      names_to = "Error_Type",
                                      values_to = "MSE")

# Plot
ggplot(model_comparison_long, aes(x = Model, y = MSE, fill = Error_Type)) +
  geom_bar(stat = "identity", position = "dodge", alpha = 0.7) +
  labs(title = "Comparison of Training and Test Errors",
       x = "Model",
       y = "Mean Squared Error (MSE)",
       fill = "Error Type") +
  theme_light() +
  theme(legend.position = "bottom",
        panel.grid = element_blank(),
        plot.title = element_text(hjust = 0.5),
        text = element_text(size = 10))
```



```
ggplot(model_comparison_long, aes(x = Model, y = MSE,
                                   color = Error_Type, group = Error_Type)) +
  geom_line(size = 0.9) +
  labs(title = "Line Plot Comparison of Training and Test Errors",
       x = "Model",
       y = "Mean Squared Error (MSE)",
       color = "Error Type") +
  theme_light() +
  theme(legend.position = "bottom",
        panel.grid = element_blank(),
        plot.title = element_text(hjust = 0.5),
        text = element_text(size = 10))
```



Classification Model Comparison for Diabetes Prediction

k-NN Classification and Cross-Validation Comparison

```
library(mlbench)
data(PimaIndiansDiabetes2)
df <- PimaIndiansDiabetes2
head(df)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35      NA 33.6    0.627  50      pos
## 2        1      85       66      29      NA 26.6    0.351  31      neg
## 3        8     183       64      NA      NA 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74      NA      NA 25.6    0.201  30      neg
```

```
na_count <- sapply(df, function(x) sum(is.na(x)))
print(na_count)
```

```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
##        0        5       35      227      374       11        0        0
## diabetes
##        0
```

```
# KNN imputation

library(mlbench)
library(VIM)
```

```r
df_imp <- kNN(df, k = 5)
df_imp <- df_imp[, 1:9]
df_imp$diabetes <- as.factor(df_imp$diabetes)
head(df_imp)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35     175 33.6    0.627  50      pos
## 2        1      85       66      29      55 26.6    0.351  31      neg
## 3        8     183       64      28     325 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74      27     112 25.6    0.201  30      neg
```

```r
na_count_df_imp <- sapply(df_imp, function(x) sum(is.na(x)))
print(na_count_df_imp)
```

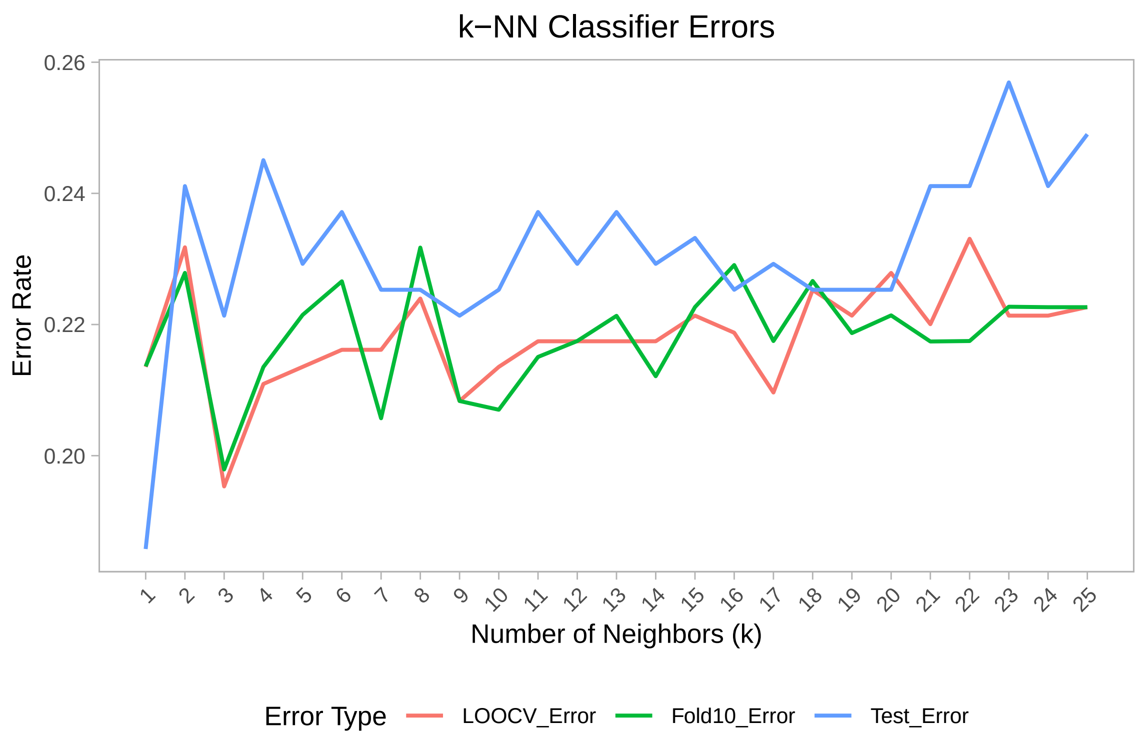
```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
##        0        0        0        0        0        0        0        0
## diabetes
##        0
```

```r
library(caret)
set.seed(2024)
trainIndex <- createDataPartition(df_imp$diabetes,
                                  p = 2/3,
                                  list = FALSE,
                                  times = 1)

trainData <- df_imp[trainIndex, ]
testData <- df_imp[-trainIndex, ]

trainData$diabetes <- ifelse(trainData$diabetes == "pos", 1, 0)
testData$diabetes <- ifelse(testData$diabetes == "pos", 1, 0)
trainData$diabetes <- as.factor(trainData$diabetes)
testData$diabetes <- as.factor(testData$diabetes)

head(trainData)
```

```
##    pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1         6     148       72      35     175 33.6    0.627  50        1
## 2         1      85       66      29      55 26.6    0.351  31        0
## 3         8     183       64      28     325 23.3    0.672  32        1
## 5         0     137       40      35     168 43.1    2.288  33        1
## 9         2     197       70      45     543 30.5    0.158  53        1
## 12       10     168       74      32     171 38.0    0.537  34        1
```

```r
library(class)
library(mlbench)
# Function: cross-validated k-NN errors
cv_knn_errors <- function(data, k_values, cv_type) {
    errors <- numeric(length(k_values))
    for (i in seq_along(k_values)) {
        k <- k_values[i]
        if (cv_type == "LOOCV") {
            control <- trainControl(method = "LOOCV")
        } else if (cv_type == "10-fold") {
            control <- trainControl(method = "cv", number = 10)
        }

        model <- train(diabetes ~ ., data = data, method = "knn",
                       trControl = control, tuneGrid = data.frame(k = k))
        errors[i] <- min(model$results$Accuracy)
    }
    return(1 - errors)
}

# k values (tested that >25 is too much)
k_values <- 1:25

# 5 fold
errors_10fold <- cv_knn_errors(df_imp, k_values, "10-fold")

# LOOCV
errors_loocv <- cv_knn_errors(df_imp, k_values, "LOOCV")

trainIndex <- createDataPartition(df_imp$diabetes, p = .67, list = FALSE)
trainData <- df_imp[trainIndex, ]
testData <- df_imp[-trainIndex, ]

# Loop for each k
test_errors <- numeric(length(k_values))
for (i in seq_along(k_values)) {
    k <- k_values[i]
    predictions <- knn(train = trainData[,-ncol(trainData)],
                       test = testData[,-ncol(testData)],
                       cl = trainData$diabetes, k = k)
    test_errors[i] <- mean(predictions != testData$diabetes)
}
```

```r
# Create a data frame for plotting
error_data <- data.frame(
    k = k_values,
    LOOCV_Error = errors_loocv,
    Fold10_Error = errors_10fold,
    Test_Error = test_errors
)

error_data_long <- reshape2::melt(error_data, id.vars = "k")

# Plotting
ggplot(error_data_long, aes(x = k, y = value, color = variable)) +
    geom_line(size = 0.7) +
    # geom_point(size = 1.3) +
    labs(title = "k-NN Classifier Errors",
         x = "Number of Neighbors (k)",
         y = "Error Rate",
         color = "Error Type") +
    scale_x_continuous(breaks = k_values) +
    theme_light() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          legend.position = "bottom",
          panel.grid = element_blank(),
          plot.title = element_text(hjust = 0.5),
          text = element_text(size = 10))
```



Generalized Additive Model for Diabetes Prediction

```
library(caret)
library(mgcv)
library(dplyr)
library(broom)

# model
gam_model <- gam(diabetes ~ s(pregnant) + s(glucose) + s(pressure) +
                 s(triceps) + s(insulin) + s(mass) +
                 s(pedigree) + s(age), data = trainData, family = binomial)
summary(gam_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## diabetes ~ s(pregnant) + s(glucose) + s(pressure) + s(triceps) +
##     s(insulin) + s(mass) + s(pedigree) + s(age)
##
## Parametric coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.3043     0.1805  -7.225    5e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df Chi.sq  p-value
## s(pregnant) 1.000  1.001  8.278 0.004022 **
## s(glucose)  4.196  5.166 21.735 0.000627 ***
## s(pressure) 1.000  1.000  1.014 0.314007
## s(triceps)  1.107  1.206  2.598 0.112210
## s(insulin)  8.411  8.889 33.158 8.95e-05 ***
## s(mass)     3.916  4.866 16.355 0.005263 **
## s(pedigree) 1.000  1.000  0.001 0.971580
## s(age)      2.222  2.806  5.477 0.131900
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.435   Deviance explained = 41.3%
## UBRE = -0.14736  Scale est. = 1         n = 515
```

```r
# library(MASS)
# # full model
# full_model <- gam(diabetes ~ s(pregnant) + s(glucose) + s(pressure) +
#                    s(triceps) + s(insulin) + s(mass) +
#                    s(pedigree) + s(age),
#                    data = trainData, family = binomial)
# # stepwise AIC
# selected_model <- step(full_model, direction = "backward")
# summary(selected_model)
```

Tree-Based Methods for Diabetes Classification

```r
library(rpart)
library(ipred)
library(randomForest)
library(caret)

classification_tree <- rpart(diabetes ~ ., data = trainData, method = "class")

tree_train_pred <- predict(classification_tree, trainData, type = "class")
tree_test_pred <- predict(classification_tree, testData, type = "class")

# error
tree_train_error <- mean(tree_train_pred != trainData$diabetes)
tree_test_error <- mean(tree_test_pred != testData$diabetes)

cat("Classification Tree Training Error:", tree_train_error, "\n")
```

```
## Classification Tree Training Error: 0.1359223
```

```r
cat("Classification Tree Test Error:", tree_test_error, "\n")
```

```
## Classification Tree Test Error: 0.201581
```

```r
bagged_trees <- bagging(diabetes ~ ., data = trainData)

bagged_train_pred <- predict(bagged_trees, trainData)
bagged_test_pred <- predict(bagged_trees, testData)

# error
bagged_train_error <- mean(bagged_train_pred != trainData$diabetes)
bagged_test_error <- mean(bagged_test_pred != testData$diabetes)

cat("Bagged Trees Training Error:", bagged_train_error, "\n")
```

```
## Bagged Trees Training Error: 0.001941748
```

```r
cat("Bagged Trees Test Error:", bagged_test_error, "\n")
```

```
## Bagged Trees Test Error: 0.1936759
```

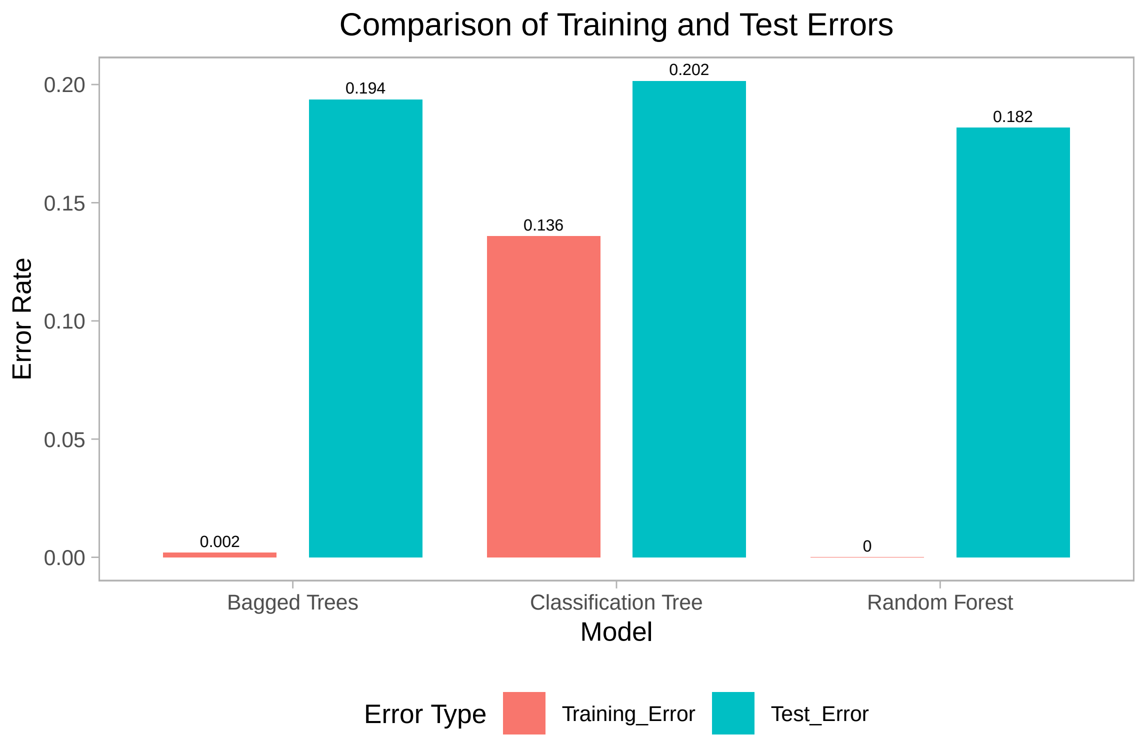
```r
# Fit the random forest
random_forest <- randomForest(diabetes ~ ., data = trainData)

# Predict on training and test data
rf_train_pred <- predict(random_forest, trainData)
rf_test_pred <- predict(random_forest, testData)

# Calculate training and test error
rf_train_error <- mean(rf_train_pred != trainData$diabetes)
rf_test_error <- mean(rf_test_pred != testData$diabetes)

cat("Random Forest Training Error:", rf_train_error, "\n")
```

```
## Random Forest Training Error: 0
```

```r
cat("Random Forest Test Error:", rf_test_error, "\n")
```

```
## Random Forest Test Error: 0.1818182
```

```r
# Store the errors in a data frame
results <- data.frame(
  Model = c("Classification Tree", "Bagged Trees", "Random Forest"),
  Training_Error = c(tree_train_error, bagged_train_error, rf_train_error),
  Test_Error = c(tree_test_error, bagged_test_error, rf_test_error)
)

# Reshape the data for ggplot
results_long <- reshape2::melt(results, id.vars = "Model", variable.name = "Error_Type", value.name = "

ggplot(results_long, aes(x = Model, y = Error, fill = Error_Type)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.9),
           width = 0.7) +
  geom_text(aes(label = round(Error, 3)),
            position = position_dodge(width = 0.9),
            vjust = -0.5, size = 2) +  # Data labels
  theme_light() +
  labs(title = "Comparison of Training and Test Errors",
       x = "Model",
       y = "Error Rate",
       fill = "Error Type") +
  theme(legend.position = "bottom",
        panel.grid = element_blank(),
        plot.title = element_text(hjust = 0.5),
        axis.title = element_text(),
        text = element_text(size = 10))
```



Neural Network Modeling for Diabetes Prediction

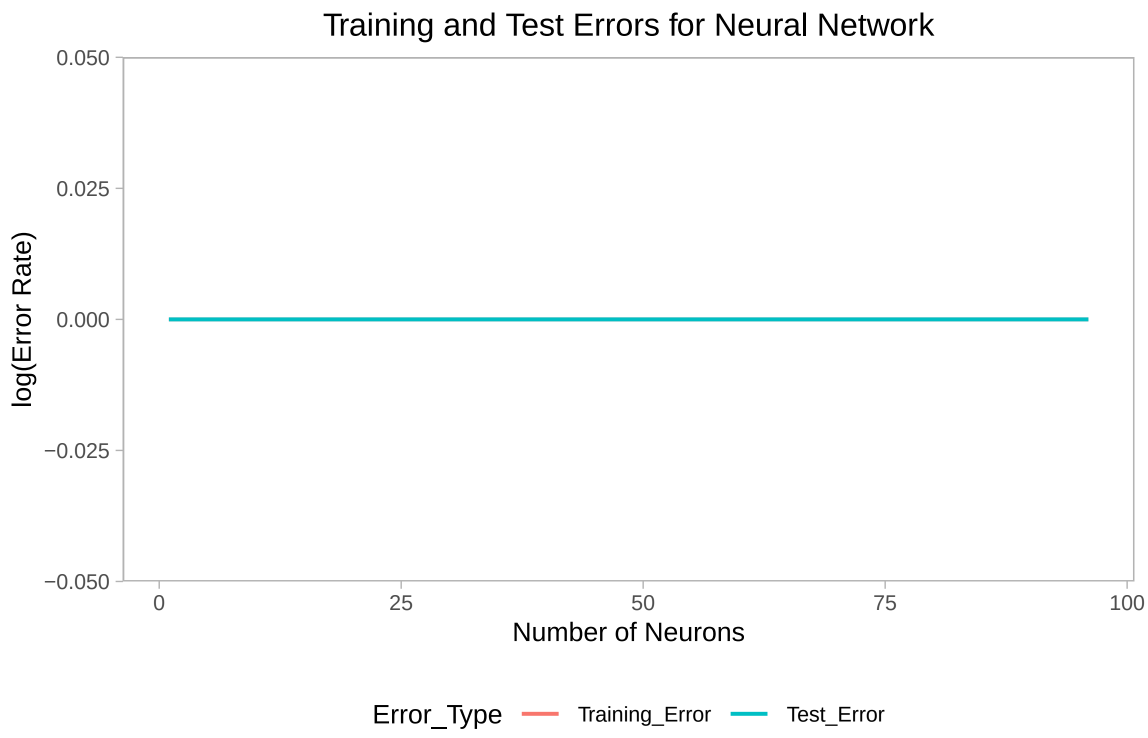
```r
library(nnet)
library(caret)

neurons <- seq(1, 100, by = 5)

# Initializing
train_errors <- c()
test_errors <- c()

# Loop
for (n in neurons) {
  # Fit NN
  nn_model <- nnet(diabetes ~ ., data = trainData,
                   size = n, linout = FALSE, maxit = 500, trace = FALSE)

  # Train
  train_pred <- predict(nn_model, trainData)
  train_pred_class <- ifelse(train_pred > 0.5, 1, 0)  # Convert to binary
  train_error <- mean(train_pred_class != trainData$diabetes)
  train_errors <- c(train_errors, train_error)

  # Test
  test_pred <- predict(nn_model, testData)
  test_pred_class <- ifelse(test_pred > 0.5, 1, 0)  # Convert to binary
  test_error <- mean(test_pred_class != testData$diabetes)
  test_errors <- c(test_errors, test_error)
}

# Results
error_results <- data.frame(
  Neurons = neurons,
  Training_Error = train_errors,
  Test_Error = test_errors
)
print(error_results)
```

```
##    Neurons Training_Error Test_Error
## 1        1              1          1
## 2        6              1          1
## 3       11              1          1
## 4       16              1          1
## 5       21              1          1
## 6       26              1          1
## 7       31              1          1
## 8       36              1          1
## 9       41              1          1
## 10      46              1          1
## 11      51              1          1
## 12      56              1          1
## 13      61              1          1
## 14      66              1          1
## 15      71              1          1
## 16      76              1          1
## 17      81              1          1
## 18      86              1          1
## 19      91              1          1
## 20      96              1          1
```

```r
# Choose the optimal number of neurons based on test error
optimal_neurons <- neurons[which.min(test_errors)]
cat("Optimal number of neurons:", optimal_neurons, "\n")
```

```
## Optimal number of neurons: 1
```

```r
library(ggplot2)
error_results_long <- reshape2::melt(error_results,
                                     id.vars = "Neurons",
                                     variable.name = "Error_Type",
                                     value.name = "Error")

ggplot(error_results_long, aes(x = Neurons, y = log10(Error),
                               color = Error_Type)) +
  geom_line(size = 0.7) +
  labs(title = "Training and Test Errors for Neural Network",
      x = "Number of Neurons",
      y = "log(Error Rate)") +
  theme_light() +
  theme(legend.position = "bottom",
        panel.grid = element_blank(),
        plot.title = element_text(hjust = 0.5),
        text = element_text(size = 10))
```



Comparing Classification Methods for Diabetes Analysis