Regression and Classification

Gabriele Durante

2024-10-21

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Problem 1: regression

(A)

```
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)

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

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

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

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

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

```
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
```

```
### Dummy Encoding for Count Variables
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_dummy <- mean((pred_model2_train - train_dummy$LC50)^2)
error_test_dummy <- mean((pred_model2_test - test_dummy$LC50)^2)

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

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

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

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

```
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
```

(B)

```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))
}
```

```r
num_repeats <- 200
results <- replicate(num_repeats, perform_analysis(data))

avg_errors <- colMeans(results)

results_df <- data.frame(
  Error = c(results[1, ], results[2, ]),
  Model = rep(c("Linear Effects", "Dummy Encoding"), each = num_repeats)
)

# Plot
mean_linear <- mean(results_df$Error[results_df$Model == "Linear Effects"])
mean_dummy <- mean(results_df$Error[results_df$Model == "Dummy Encoding"])

ggplot(results_df, aes(x = Error, fill = Model)) +
  geom_density(alpha = 0.5) +
  geom_vline(aes(xintercept = mean_linear, color = "Linear Model Mean"), linetype = "dashed", size = 0.5
  geom_vline(aes(xintercept = mean_dummy, color = "Dummy Model Mean"), linetype = "dashed", size = 0.5)
  labs(title = "Empirical Distribution of Test Errors",
       x = "Test Error (MSE)",
       y = "Density") +
  scale_color_manual(values = c("Linear Model Mean" = "blue", "Dummy Model Mean" = "red")) +
  theme_minimal()
```

(C)

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

```r
# 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
```

```r
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 = F
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
```

```
forward_bic <- stepAIC(null_model, direction = "forward", scope = formula(full_model), k = log(nrow(tra
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
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(D)

```
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

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
```

```
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
```

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

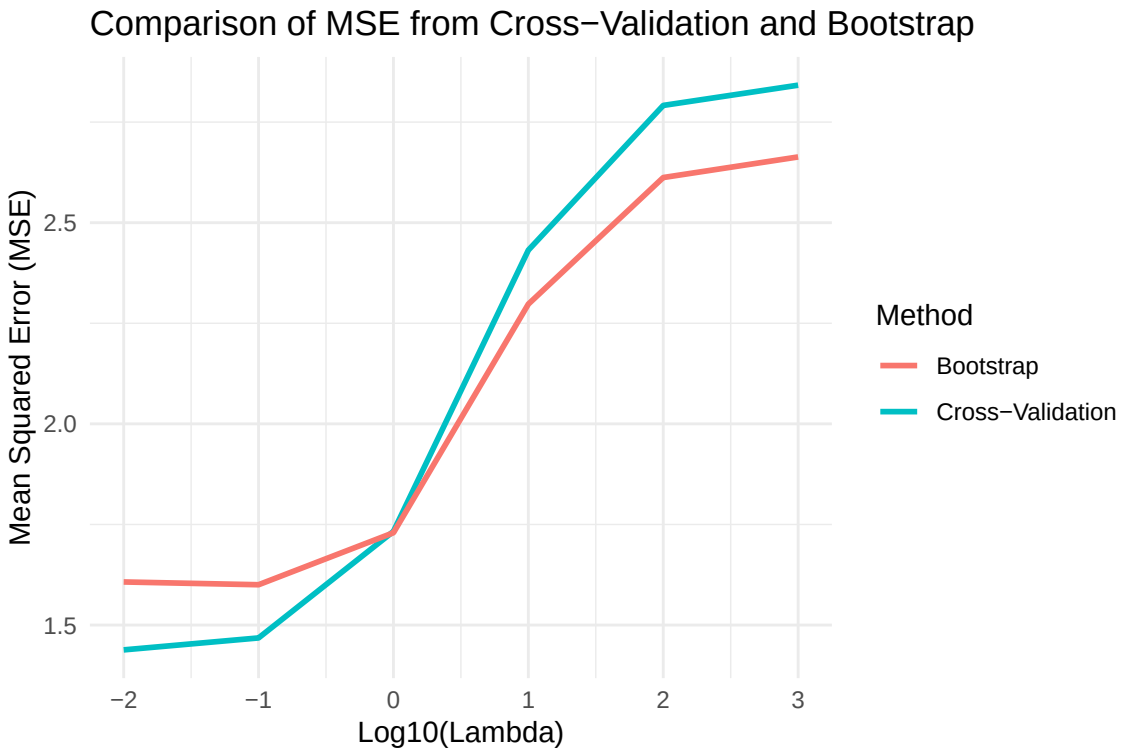
```
## Test Error (Ridge): 1.554029
```

```
# 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)]
```

```r
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
)

# 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_minimal()
```



{E}

```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)
```

```r
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

# 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
```

(F)

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

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

train_predictions_tree1 <- predict(tree_model, newdata = trainData)
```

```r
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)

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
```

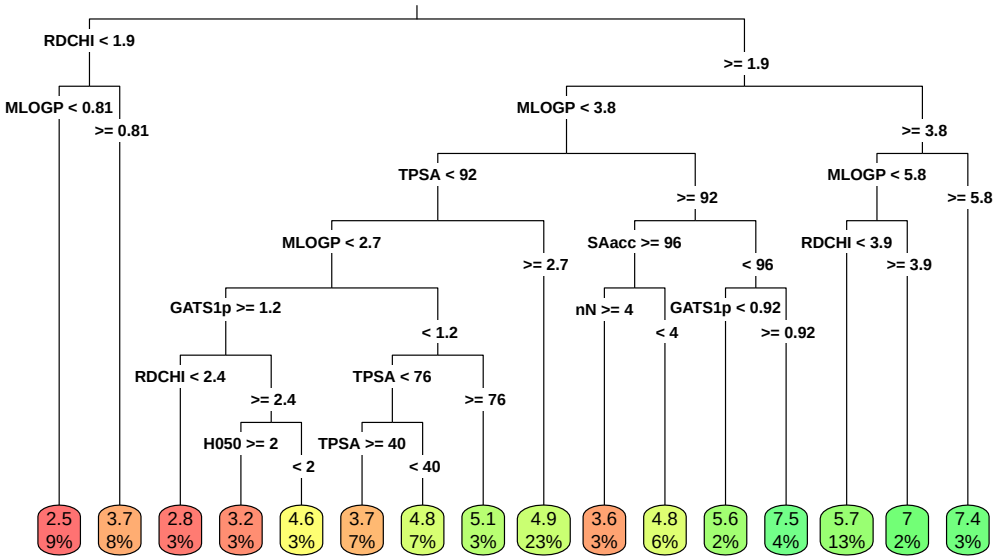
```r
cat("Training Error (Tree):", mse_train_tree1, "\n")
```

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

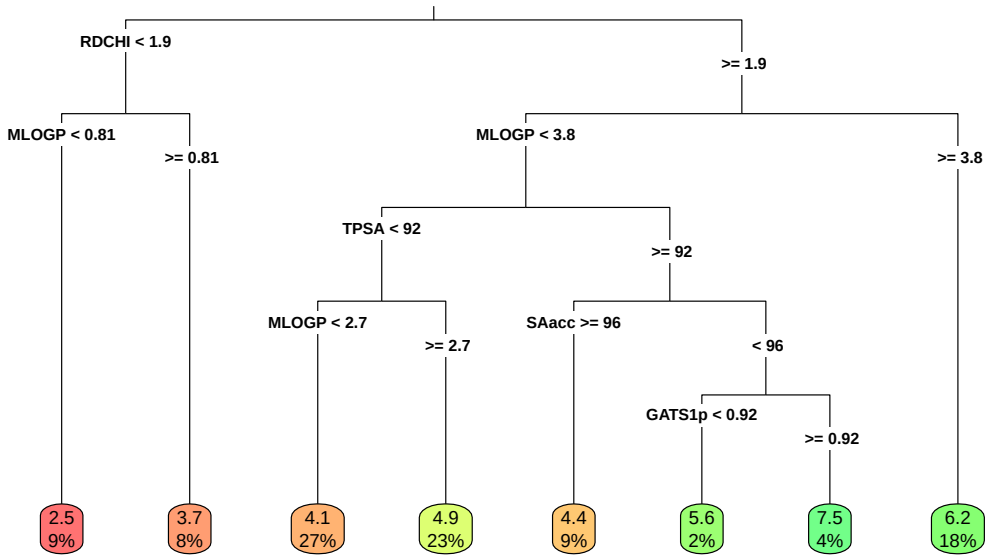
```r
cat("Test Error (Tree):", mse_test_tree1, "\n")
```

```
## 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,
)
```




```r
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")
```

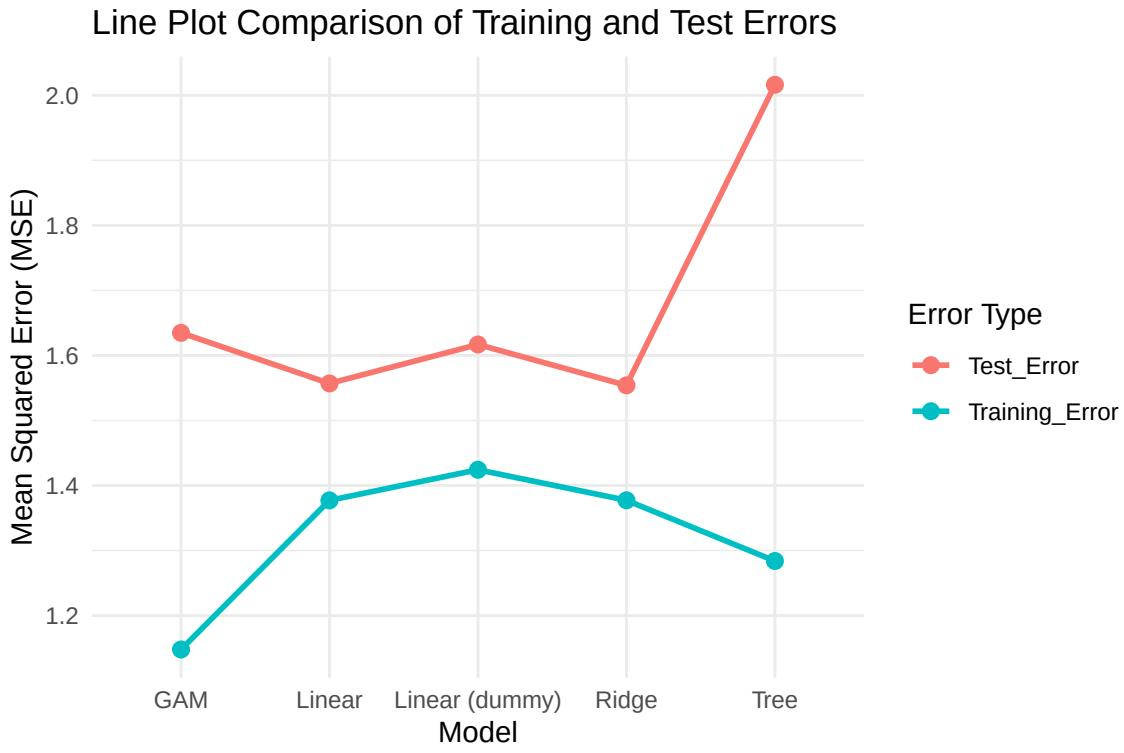
```
## Test Error (Tree): 2.016152
```

```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") +
  # scale_fill_manual(values = c("Training_Error" = "blue", "Test_Error" = "red"), labels = c("Training
  theme_minimal()
```


```r
# print(model_comparison)
```

```r
ggplot(model_comparison_long, aes(x = Model, y = MSE, color = Error_Type, group = Error_Type)) +
  geom_line(size = 1) +
  geom_point(size = 2.5) +
  labs(title = "Line Plot Comparison of Training and Test Errors",
```

```
      x = "Model",
      y = "Mean Squared Error (MSE)",
      color = "Error Type") +
  # scale_color_manual(values = c("blue", "red")) +
  theme_minimal()
```



Problem 2: Classification

(A)

```
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)

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
```

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

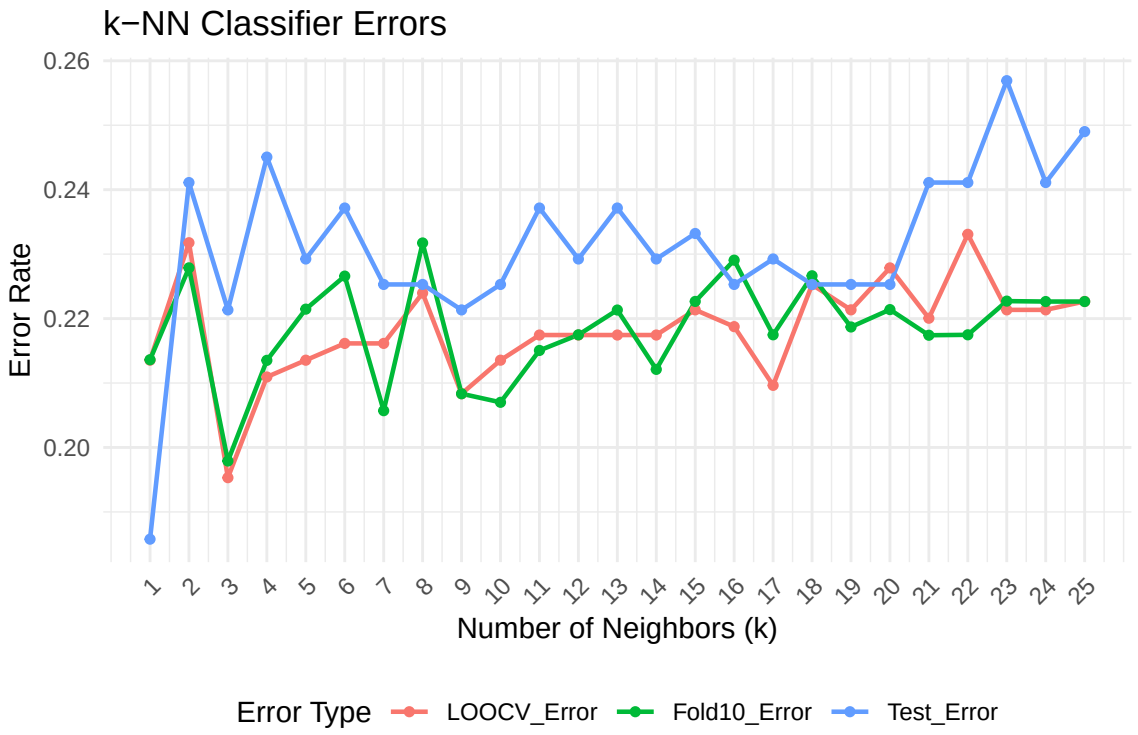
```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
##        0        0        0        0        0        0        0        0
## diabetes
##        0
```

```
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, ]
```

```
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)
}
```

```
# 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 = 0.8) +
    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_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          legend.position = "bottom")
```


k−NN Classifier Errors
Error Rate
0.26
0.24
0.22
0.20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Number of Neighbors (k)
Error Type
LOOCV_Error
Fold10_Error
Test_Error

(B)