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)

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

pred_model1_train <- predict(model1, new_data = X_train)
pred_model1_test <- predict(model1, new_data = X_test)

error_train <- mean((y_train - pred_model1_train)^2)
error_test <- mean((y_test - pred_model1_test)^2)

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

```
## Training Error (Linear): 1.415068
```

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

```
## Test Error (Linear): 4.208063
```

```
summary(model1)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = trainData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5253 -0.7624 -0.1460  0.5729  4.0203
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.699819   0.288116   9.371  < 2e-16 ***
## TPSA         0.026685   0.003323   8.031 1.45e-14 ***
## SAacc       -0.014975   0.002534  -5.909 8.10e-09 ***
```

```
## H050          0.026646   0.072625   0.367  0.71391
## MLOGP         0.442232   0.079973   5.530 6.24e-08 ***
## RDCHI         0.527074   0.168630   3.126  0.00192 **
## GATS1p       -0.598450   0.189562  -3.157  0.00173 **
## nN           -0.186374   0.060498  -3.081  0.00223 **
## C040         -0.030411   0.121474  -0.250  0.80246
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.205 on 354 degrees of freedom
## Multiple R-squared:  0.4876, Adjusted R-squared:  0.476
## F-statistic: 42.11 on 8 and 354 DF,  p-value: < 2.2e-16
```

```r
### Dummy Encoding for Count Variables
X_train_dummy <- X_train
X_test_dummy <- X_test

# dummy encoding
X_train_dummy$nN <- ifelse(X_train_dummy$nN > 0, 1, 0)
X_test_dummy$nN <- ifelse(X_test_dummy$nN > 0, 1, 0)
# X_test_dummy$C040 <- ifelse(X_test_dummy$C040 > 0, 1, 0)
# X_train_dummy$C040 <- ifelse(X_train_dummy$C040 > 0, 1, 0)

trainData_dummy <- cbind(y_train, X_train_dummy)
testData_dummy <- cbind(y_test, X_test_dummy)

model2 <- lm(y_train ~ ., data = trainData_dummy)

pred_model2_train <- predict(model2, newdata = X_train_dummy)
pred_model2_test <- predict(model2, newdata = X_test_dummy)

error_train_dummy <- mean((y_train - pred_model2_train)^2)
error_test_dummy <- mean((y_test - pred_model2_test)^2)

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

```
## Training Error (Dummy): 1.452133
```

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

```
## Test Error (Dummy): 1.544053
```

```r
summary(model2)
```

```
##
## Call:
## lm(formula = y_train ~ ., data = trainData_dummy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3830 -0.7623 -0.1302  0.5663  4.2226
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.754816   0.294434   9.356  < 2e-16 ***
## TPSA         0.023291   0.003220   7.233 2.94e-12 ***
```

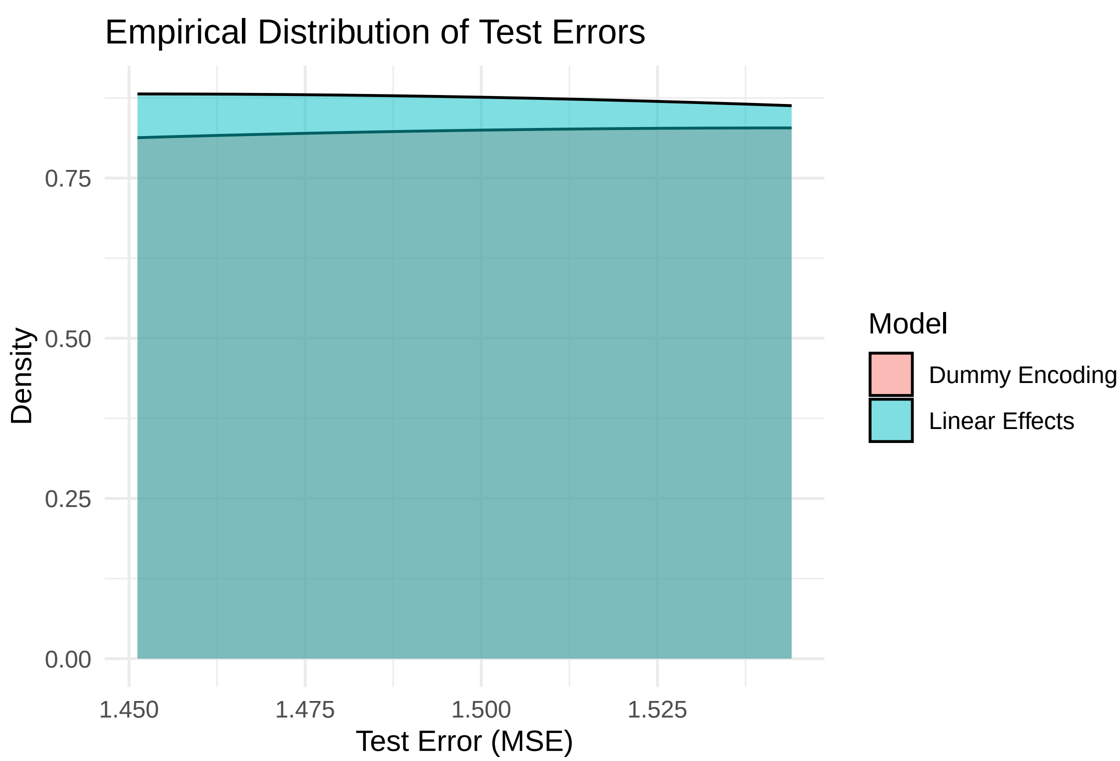
```
## SAacc       -0.013899   0.002567  -5.414 1.14e-07 ***
## H050        -0.010182   0.072587  -0.140  0.88853
## MLOGP        0.439810   0.082697   5.318 1.86e-07 ***
## RDCHI        0.509049   0.173766   2.930  0.00361 **
## GATS1p      -0.626709   0.192887  -3.249  0.00127 **
## nN          -0.069187   0.150148  -0.461  0.64523
## C040        -0.006918   0.122829  -0.056  0.95511
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.22 on 354 degrees of freedom
## Multiple R-squared:  0.4742, Adjusted R-squared:  0.4623
## F-statistic: 39.91 on 8 and 354 DF,  p-value: < 2.2e-16
```

(B)

```r
library(ggplot2)

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

  y_train <- trainData$LC50
  X_train <- trainData[, !names(trainData) %in% c("LC50")]
  y_test <- testData$LC50
  X_test <- testData[, !names(testData) %in% c("LC50")]

  # model 1
  model1 <- lm(LC50 ~ ., data = trainData)
  pred_model1_test <- predict(model1, newdata = testData)
  error_test1 <- mean((y_test - pred_model1_test)^2)

  # model 2
  X_train_dummy <- X_train
  X_test_dummy <- X_test

  X_train_dummy$nN <- ifelse(X_train_dummy$nN > 0, 1, 0)
  X_test_dummy$nN <- ifelse(X_test_dummy$nN > 0, 1, 0)
  #X_train_dummy$C040 <- ifelse(X_train_dummy$C040 > 0, 1, 0)
  #X_test_dummy$C040 <- ifelse(X_test_dummy$C040 > 0, 1, 0)

  trainData_dummy <- cbind(y_train, X_train_dummy)
  model2 <- lm(y_train ~ ., data = trainData_dummy)
  pred_model2_test <- predict(model2, newdata = X_test_dummy)
  error_test2 <- mean((y_test - pred_model2_test)^2)

  return(c(error_test1, error_test2))
}
```

```r
set.seed(2024)
num_repeats <- 200
results <- replicate(num_repeats, perform_analysis(data))
```

```r
# Calculate average test errors
avg_errors <- colMeans(results)

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

# Plotting empirical distributions
ggplot(results_df, aes(x = Error, fill = Model)) +
  geom_density(alpha = 0.5) +
  labs(title = "Empirical Distribution of Test Errors",
       x = "Test Error (MSE)",
       y = "Density") +
  theme_minimal()
```




```r
cat("Average Test Error (Linear):", avg_errors[1], "\n")
```

```
## Average Test Error (Linear): 1.497626
```

```r
cat("Average Test Error (Dummy):", avg_errors[2], "\n")
```

```
## Average Test Error (Dummy): 1.497626
```

(C)

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

```r
# Define full model
full_model <- lm(LC50 ~ ., data = trainData)

# Backward Elimination
backward_aic <- stepAIC(full_model, direction = "backward", k = 2)
backward_bic <- stepAIC(full_model, direction = "backward", k = log(nrow(trainData)))

# Forward Selection
null_model <- lm(LC50 ~ 1, data = trainData)  # Null model with no predictors
forward_aic <- stepAIC(null_model, direction = "forward", scope = formula(full_model), k = 2)
forward_bic <- stepAIC(null_model, direction = "forward", scope = formula(full_model), k = log(nrow(tra
```

```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 + RDCHI + 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 + RDCHI,
##     data = trainData)
```

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

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

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

```
## AIC Backward: 1172.459
```

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

```
## BIC Backward: 1203.614
```

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

```
## AIC Forward: 1172.459
```

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

```
## BIC Forward: 1203.614
```

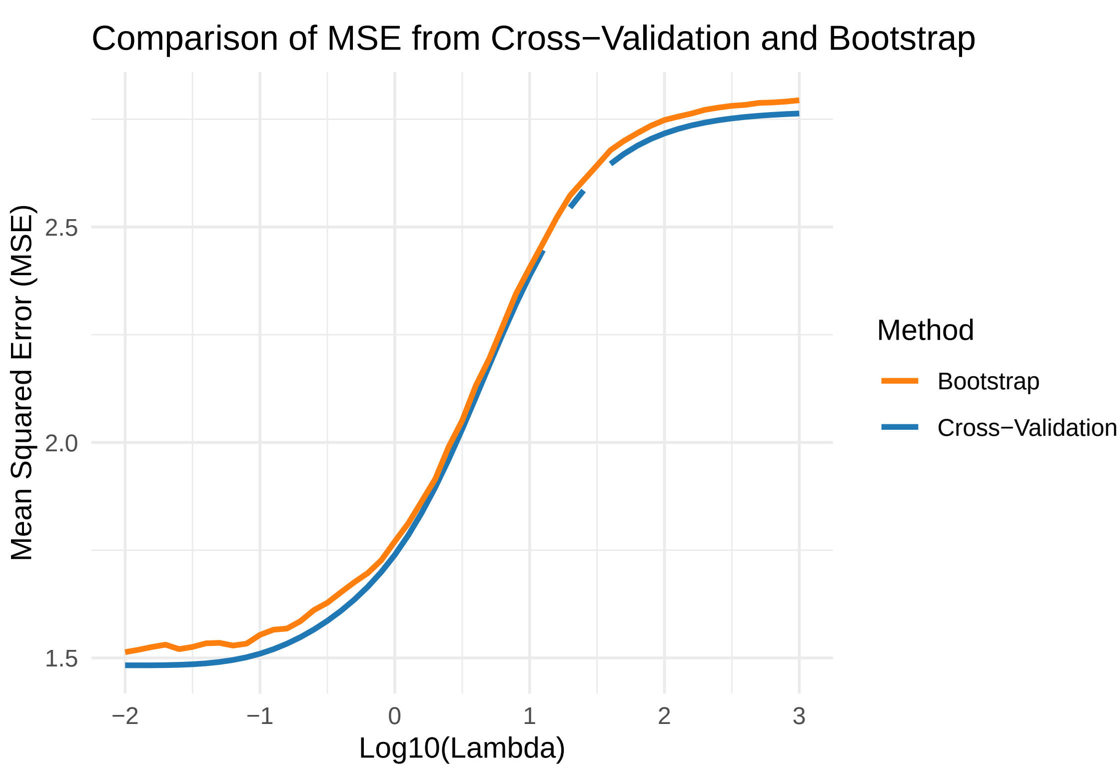
(D)

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

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

lambda_grid <- 10^seq(3, -2, by = -0.1)

# Ridge regression (CV)
set.seed(2024)
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: 0.01258925
```

```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)



# Ridge regression (bootstrap)
set.seed(2024)
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
set.seed(2024)
bootstrap_results <- sapply(lambda_grid, function(lambda) {
  mse_values <- replicate(100, boot(trainData, bootstrap_mse, R = 1, lambda = lambda)$t)
  return(mean(mse_values))
})

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) +
  scale_color_manual(values = c("Cross-Validation" = "#1f77b4", "Bootstrap" = "#ff7f0e")) +
  labs(title = "Comparison of MSE from Cross-Validation and Bootstrap",
       x = "Log10(Lambda)",
       y = "Mean Squared Error (MSE)",
       color = "Method") +
  theme_minimal()
```



```r
# 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: 0.01
```

{E}

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

```
##   TPSA  SAacc   H050  MLOGP  RDCHI GATS1p     nN   C040   LC50
##    165    152     11    277    250    282      8      4    344
```

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

# GAM less complexity (k = -1)
gam_model_1 <- gam(LC50 ~ s(TPSA, k=1) + s(SAacc, k=1) + s(H050, k=1) +
                     s(MLOGP, k=1) + s(RDCHI, k=1) + s(GATS1p, k=1) +
                     s(nN, k=1) + s(C040, k=1), 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=5):","\t",mse_train_gam_1, "\n")
```

```
## Training Error (GAM - k=5):   1.357511
```

```r
cat("Test Error (GAM - k=5):","\t",mse_test_gam_1, "\n")
```

```
## Test Error (GAM - k=5):   1.483875
```

```r
# GAM more complexity (k = 10)
gam_model_2 <- gam(LC50 ~ s(TPSA, k=4) + s(SAacc, k=4) + s(H050, k=4) +
                     s(MLOGP, k=4) + s(RDCHI, k=4) + s(GATS1p, k=4) +
                     s(nN, k=4) + s(C040, k=4), 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=10):","\t",mse_train_gam_2, "\n")
```

```
## Training Error (GAM - k=10):      1.348185
```

```r
cat("Test Error (GAM - k=10):","\t",mse_test_gam_2, "\n")
```

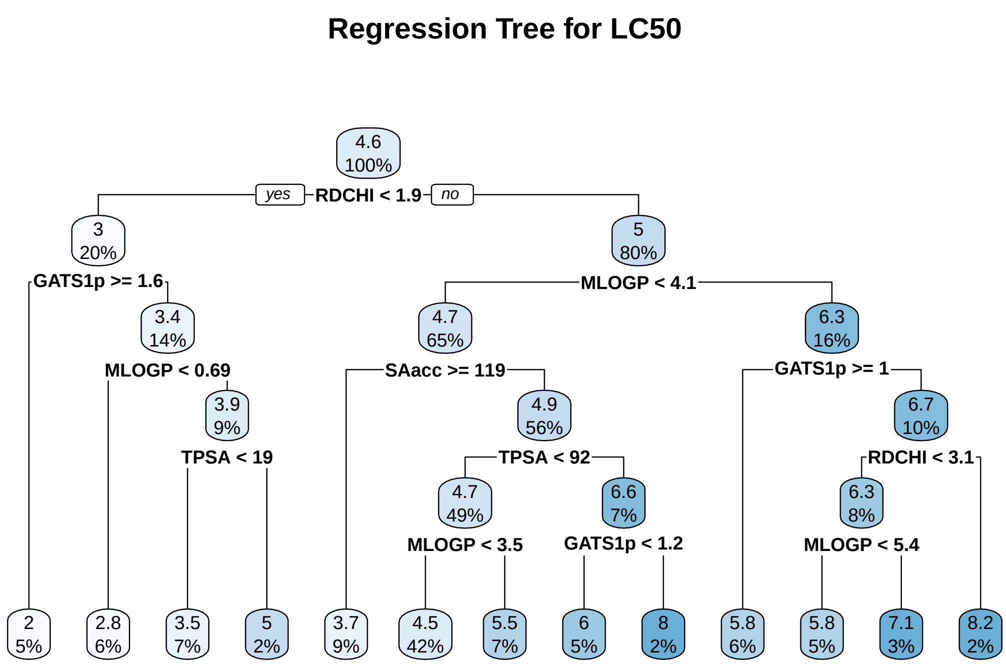
```
## Test Error (GAM - k=10):      1.506197
```

```r
names(X_train)
```

```
## NULL
```

(F)

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

tree_model <- rpart(LC50 ~ ., data = trainData, method = "anova")
rpart.plot(tree_model, main = "Regression Tree for LC50")
```

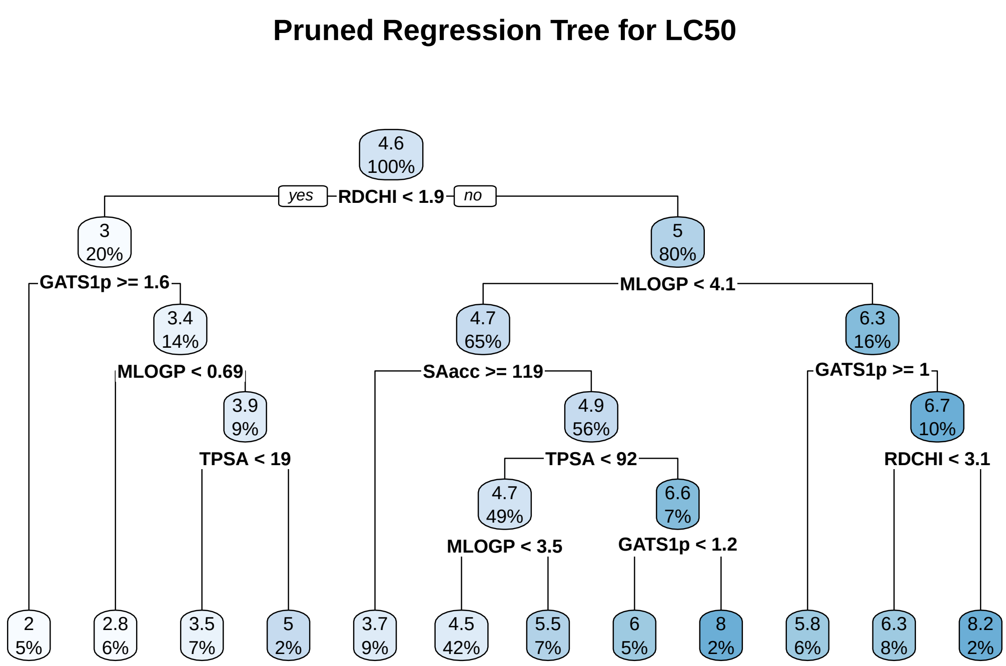



```r
printcp(tree_model)
```

```
##
## Regression tree:
## rpart(formula = LC50 ~ ., data = trainData, method = "anova")
##
## Variables actually used in tree construction:
## [1] GATS1p MLOGP  RDCHI  SAacc  TPSA
##
## Root node error: 1002.5/363 = 2.7618
##
## n= 363
##
##          CP nsplit rel error  xerror     xstd
```

```
## 1  0.225814      0    1.00000 1.00961 0.080992
## 2  0.119423      1    0.77419 0.84506 0.065037
## 3  0.061170      2    0.65476 0.69799 0.058448
## 4  0.031060      4    0.53242 0.59814 0.049249
## 5  0.022793      5    0.50136 0.60025 0.050562
## 6  0.018592      6    0.47857 0.59386 0.050006
## 7  0.014828      7    0.45998 0.59366 0.050797
## 8  0.013643      9    0.43032 0.60437 0.052007
## 9  0.011008     10    0.41668 0.59524 0.050659
## 10 0.010841     11    0.40567 0.59053 0.050782
## 11 0.010000     12    0.39483 0.59225 0.051929
```

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

pruned_tree <- prune(tree_model, cp = optimal_cp)
rpart.plot(pruned_tree, main = "Pruned Regression Tree for LC50")
```




```r
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)

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

```
## Training Error (Tree): 1.120379
```

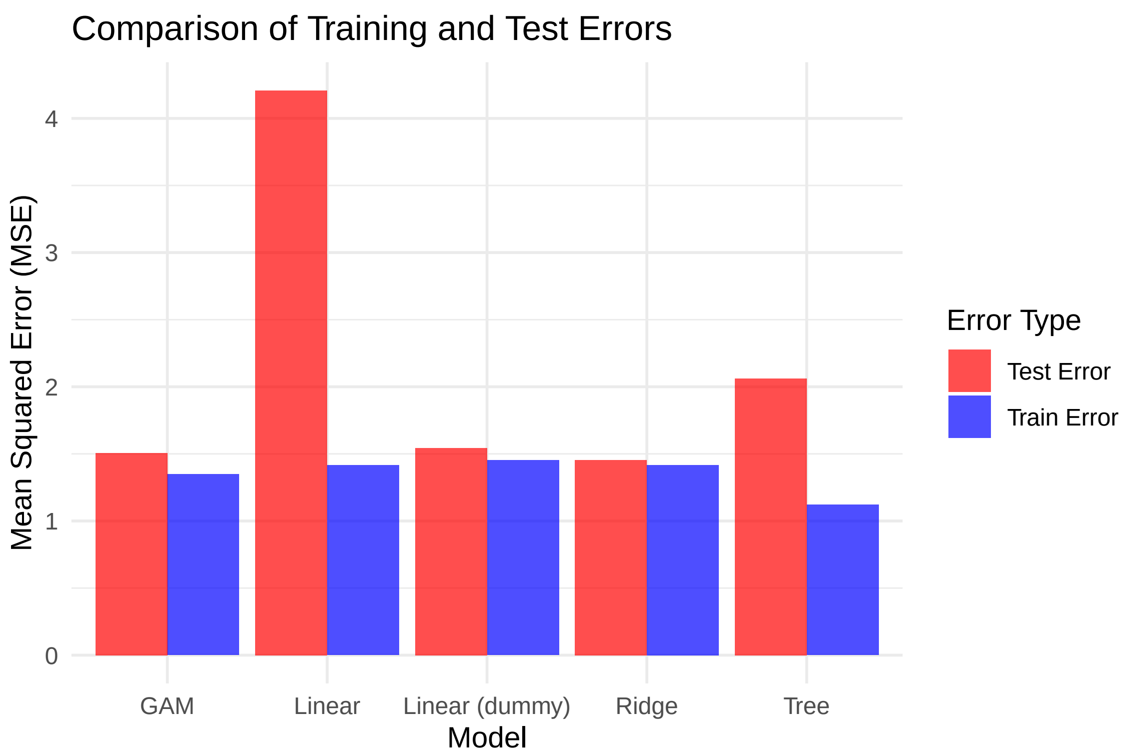
```r
cat("Test Error (Tree):", mse_test_tree, "\n")
```

```
## Test Error (Tree): 2.061929
```

```r
library(tidyr)
library(ggplot2)

# Assuming error_train, error_train_dummy, mse_train_ridge, mse_train_gam_2, mse_train_tree
# are already defined with the respective mean squared error values.

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)
)

# Print the model_comparison data frame
print(model_comparison)
```

```
##            Model Training_Error Test_Error
## 1         Linear       1.415068   4.208063
## 2 Linear (dummy)       1.452133   1.544053
## 3          Ridge       1.416614   1.454867
## 4            GAM       1.348185   1.506197
## 5           Tree       1.120379   2.061929
```

```r
# Convert to long format for ggplot
model_comparison_long <- pivot_longer(model_comparison,
                                      cols = c("Training_Error", "Test_Error"),
                                      names_to = "Error_Type",
                                      values_to = "MSE")

# Create the bar plot with proper legend
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_Error" = "Train Error", "Test_Error" = "Test Error")) +
  theme_minimal()
```



Problem 2: Classification

(A)