Assignment 7 - Tree Based Methods

Ben Abraham Biju

March 18, 2025

1. Introduction

Classification techniques using decision trees and ensemble methods on a drug dataset are explored in this analysis. The following methods have been used:

- Decision tree classification
- Tree pruning
- Bagging
- Random forest
- Parameter tuning

The following dataset was used for this analysis: Drugs dataset (https://www.kaggle.com/datasets/pablomgomez21/drugs-a-b-c-x-y-for-decision-trees (https://www.kaggle.com/datasets/pablomgomez21/drugs-a-b-c-x-y-for-decision-trees))

The workspace has been set to the directory with the dataset.

```
setwd(getwd())
```

2. Import the Dataset

```
df <- read.csv("drug200.csv")

# Display the first few rows
head(df)
```

```
##   Age Sex     BP Cholesterol Na_to_K  Drug
## 1  23   F   HIGH        HIGH  25.355 drugY
## 2  47   M    LOW        HIGH  13.093 drugC
## 3  47   M    LOW        HIGH  10.114 drugC
## 4  28   F NORMAL        HIGH   7.798 drugX
## 5  61   F    LOW        HIGH  18.043 drugY
## 6  22   F NORMAL        HIGH   8.607 drugX
```

The features of this dataset are Age, Sex, Blood Pressure, and the Cholesterol of the patients, and the target is the drug that each patient responded to.

3. Data Cleaning and Pre-processing

```
str(df)
```

```
## 'data.frame':    200 obs. of  6 variables:
##  $ Age        : int  23 47 47 28 61 22 49 41 60 43 ...
##  $ Sex        : chr  "F" "M" "M" "F" ...
##  $ BP         : chr  "HIGH" "LOW" "LOW" "NORMAL" ...
##  $ Cholesterol: chr  "HIGH" "HIGH" "HIGH" "HIGH" ...
##  $ Na_to_K    : num  25.4 13.1 10.1 7.8 18 ...
##  $ Drug       : chr  "drugY" "drugC" "drugC" "drugX" ...
```

```
summary(df)
```

```
##       Age            Sex                BP           Cholesterol
##  Min.   :15.00   Length:200         Length:200         Length:200
##  1st Qu.:31.00   Class :character   Class :character   Class :character
##  Median :45.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :44.31
##  3rd Qu.:58.00
##  Max.   :74.00
##     Na_to_K           Drug
##  Min.   : 6.269   Length:200
##  1st Qu.:10.445   Class :character
##  Median :13.937   Mode  :character
##  Mean   :16.084
##  3rd Qu.:19.380
##  Max.   :38.247
```

```
# Check for missing values
cat("Number of missing values:", sum(is.na(df)), "\n")
```

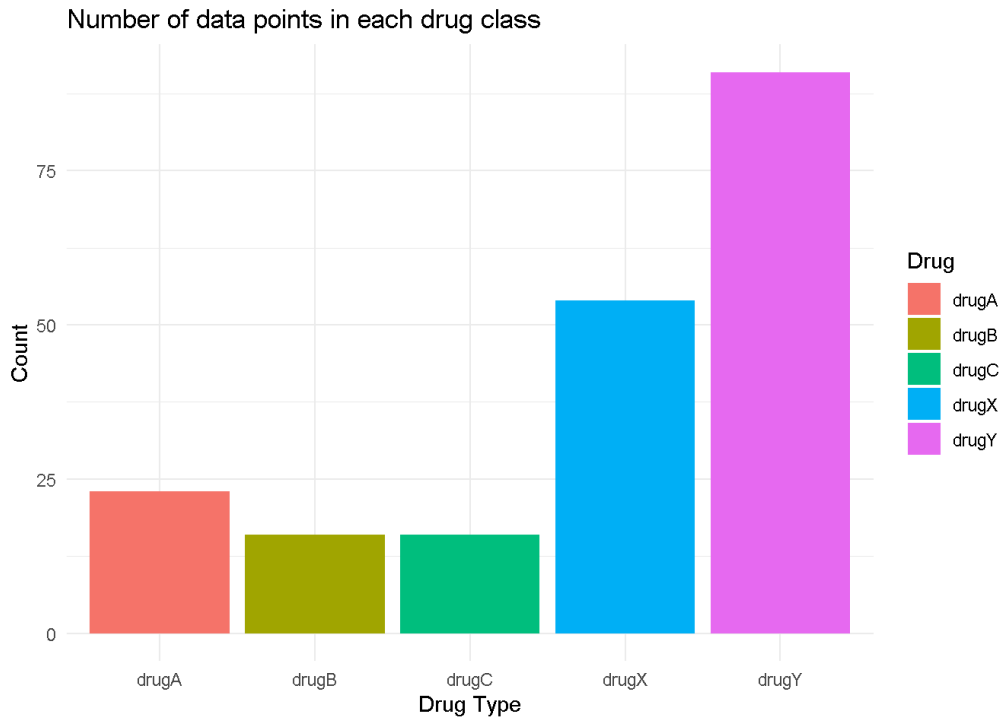
```
## Number of missing values: 0
```

The dataset contains 200 observations with 6 variables:

- Age: Numerical variable ranging from 15 to 74 years
- Sex: Categorical with 2 levels (F, M)
- BP: Categorical with 3 levels (HIGH, LOW, NORMAL)
- Cholesterol: Categorical with 2 levels (HIGH, NORMAL)
- Na_to_K: Numerical variable ranging from 6.269 to 38.247
- Drug: Categorical with 5 levels (drugA, drugB, drugC, drugX, drugY)

The number of data points belonging to each drug class is shown below.

```
library(ggplot2)

ggplot(df, aes(x = Drug, fill = Drug)) +
  geom_bar() +
  labs(title = "Number of data points in each drug class",
       x = "Drug Type",
       y = "Count") +
  theme_minimal()
```



```
df <- na.omit(df)

# Convert categorical inputs to factors
df$Sex <- as.factor(df$Sex)
df$BP <- as.factor(df$BP)
df$Cholesterol <- as.factor(df$Cholesterol)
df$Drug <- as.factor(df$Drug)


str(df)
```

```
## 'data.frame':    200 obs. of  6 variables:
##  $ Age        : int  23 47 47 28 61 22 49 41 60 43 ...
##  $ Sex        : Factor w/ 2 levels "F","M": 1 2 2 1 1 1 1 2 2 2 ...
##  $ BP         : Factor w/ 3 levels "HIGH","LOW","NORMAL": 1 2 2 3 2 3 3 2 3 2 ...
##  $ Cholesterol: Factor w/ 2 levels "HIGH","NORMAL": 1 1 1 1 1 1 1 1 1 2 ...
##  $ Na_to_K    : num  25.4 13.1 10.1 7.8 18 ...
##  $ Drug       : Factor w/ 5 levels "drugA","drugB",..: 5 3 3 4 5 4 5 3 5 5 ...
```

All categorical variables have been successfully converted to factors, which is necessary for proper model fitting.

4. Split Data into Training and Testing Sets

We'll use 70% of the data for training and 30% for testing.

```
set.seed(123)

train_indices <- sample(1:nrow(df), 0.7 * nrow(df))
train_data <- df[train_indices, ]
test_data <- df[-train_indices, ]

cat("Training set size:", nrow(train_data), "rows\n")
```

```
## Training set size: 140 rows
```

```
cat("Testing set size:", nrow(test_data), "rows\n")
```

```
## Testing set size: 60 rows
```

The data is now split into a training set with 140 observations and a test set with 60 observations.

5. Fit a Decision Tree Model

We'll use the `rpart` package to fit a decision tree classification model.

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

# Fit a classification tree
tree_model <- rpart(Drug ~ ., data = train_data, method = "class")

printcp(tree_model)
```

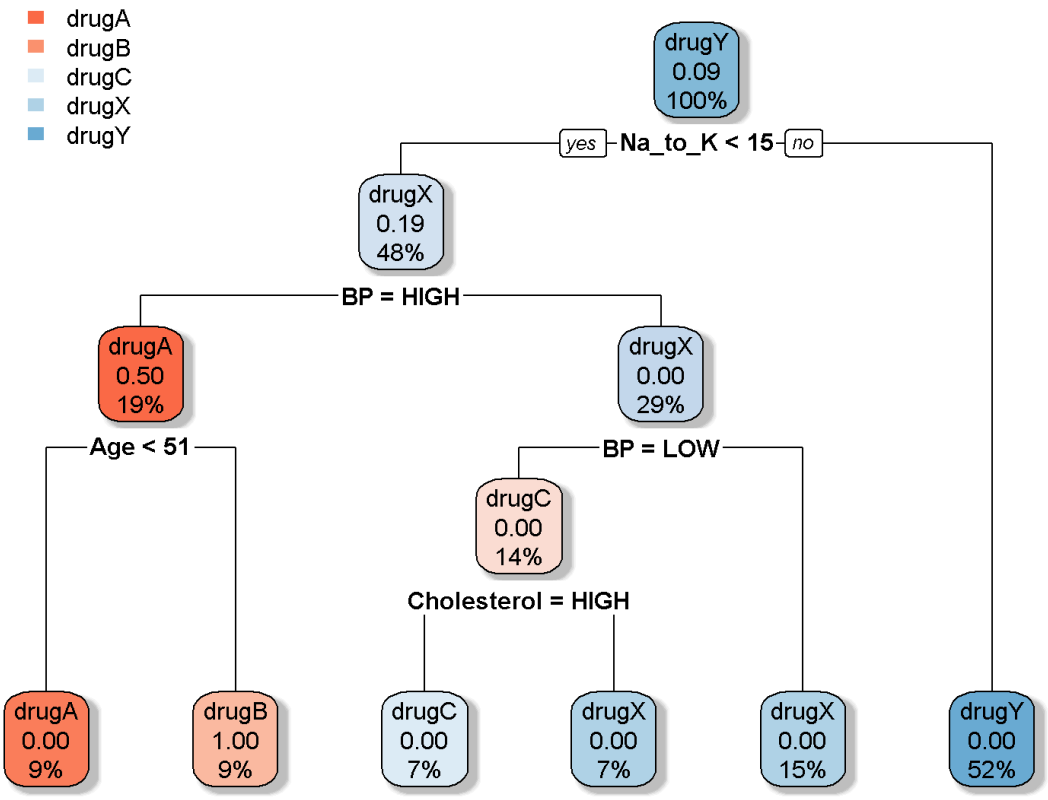
```
##
## Classification tree:
## rpart(formula = Drug ~ ., data = train_data, method = "class")
##
## Variables actually used in tree construction:
## [1] Age         BP          Cholesterol Na_to_K
##
## Root node error: 67/140 = 0.47857
##
## n= 140
##
##         CP nsplit rel error  xerror     xstd
## 1 0.462687      0   1.00000 1.00000 0.088219
## 2 0.194030      1   0.53731 0.55224 0.077872
## 3 0.074627      3   0.14925 0.19403 0.051255
## 4 0.010000      5   0.00000 0.23881 0.056187
```

The classification tree uses Age, BP, Cholesterol, and Na_to_K as predictors. The root node error is 0.47857, indicating that about 48% of observations would be misclassified if we predicted the most frequent class for all observations.

6. Plot the Decision Tree

Visualizing the decision tree helps us understand the classification rules.

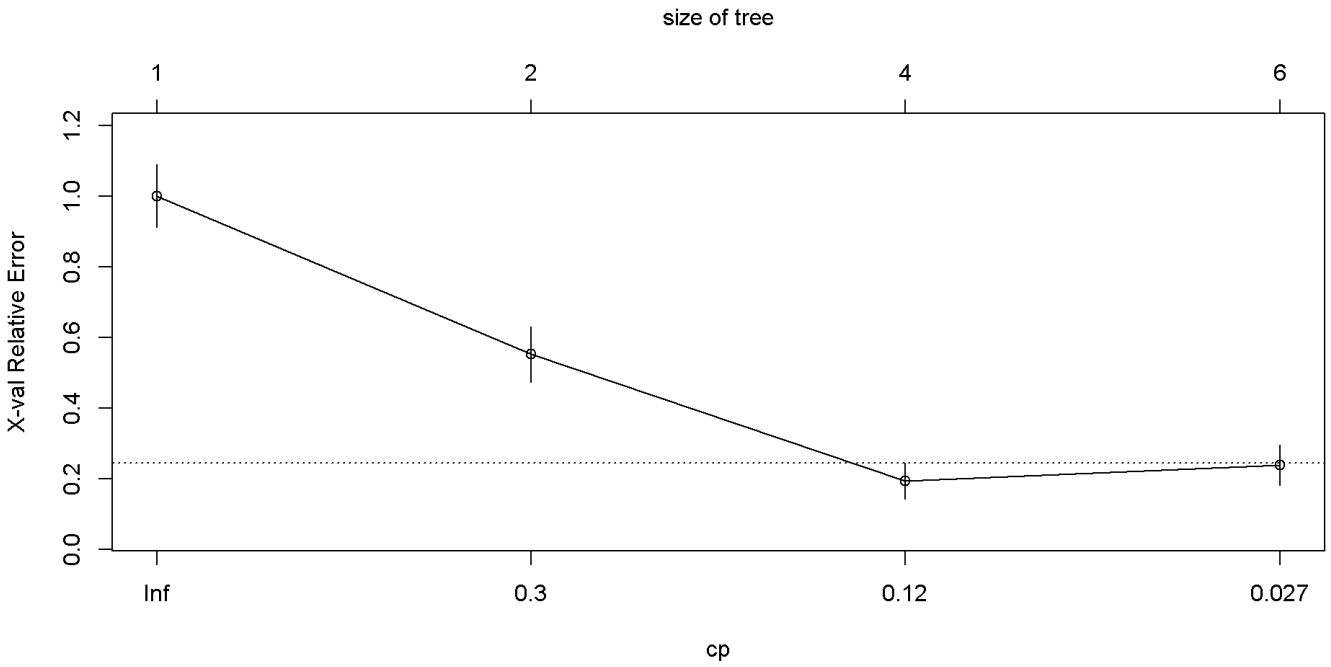
```
rpart.plot(tree_model, extra = 106, box.palette = "RdBu", shadow.col = "gray")
```



The decision tree shows how different features influence drug recommendations. Each node displays the predicted class, probability distribution, and the percentage of observations in that node. The primary split is based on Na_to_K, with subsequent splits using Age, BP, and Cholesterol.

7. Prune the Tree

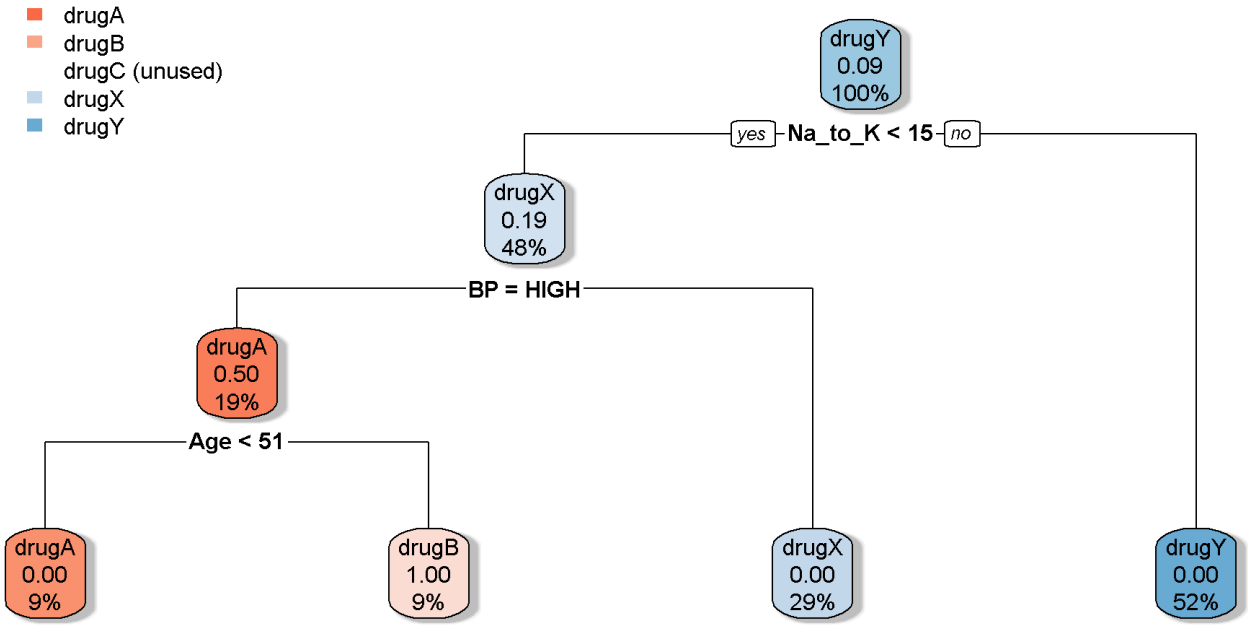
```
# Plot the cross-validation results
plotcp(tree_model)
```




```
# Find the optimal complexity parameter
best_cp <- tree_model$cptable[which.min(tree_model$cptable[,"xerror"]), "CP"]
cat("Best complexity parameter (CP):", best_cp, "\n")
```

```
## Best complexity parameter (CP): 0.07462687
```

```
# Prune the tree using the best CP value
pruned_tree <- prune(tree_model, cp = best_cp)

# Plot the pruned tree
rpart.plot(pruned_tree, extra = 106, box.palette = "RdBu", shadow.col = "gray")
```



The cross-validation plot helps identify the optimal complexity parameter (CP) value of 0.07462687. The pruned tree is simpler with fewer splits, which should help prevent overfitting and improve generalization to new data.

8. Evaluate Model Performance

The accuracy of both the original and pruned trees is calculated.

```
# Predictions for unpruned tree
tree_pred <- predict(tree_model, test_data, type = "class")
tree_accuracy <- mean(tree_pred == test_data$Drug)
cat("Decision Tree Accuracy:", (tree_accuracy * 100), "%\n")
```

```
## Decision Tree Accuracy: 100 %
```

```
# Predictions for pruned tree
pruned_pred <- predict(pruned_tree, test_data, type = "class")
pruned_accuracy <- mean(pruned_pred == test_data$Drug)
cat("Pruned Tree Accuracy:", (pruned_accuracy * 100), "%\n")
```

```
## Pruned Tree Accuracy: 90 %
```

```
# Confusion matrix for pruned tree
conf_matrix_pruned <- table(Predicted = pruned_pred, Actual = test_data$Drug)
print(conf_matrix_pruned)
```

```
##          Actual
## Predicted drugA drugB drugC drugX drugY
##     drugA    10     0     0     0     0
##     drugB     0     3     0     0     0
##     drugC     0     0     0     0     0
##     drugX     0     0     6    23     0
##     drugY     0     0     0     0    18
```

The unpruned decision tree achieves a perfect accuracy of 100% on the test set. The pruned tree has a slightly lower accuracy of 90%, which is still excellent. The confusion matrix for the pruned tree shows that it correctly classified all drugA, drugB, and drugY instances, but had some confusion between drugC and drugX.

9. Fit Bagging and Random Forest Models

```
# Load required library
library(randomForest)

# Bagging
num_predictors <- ncol(df) - 1
bagging_model <- randomForest(Drug ~ ., data = train_data, mtry = num_predictors, ntree = 500)
bagging_pred <- predict(bagging_model, test_data)
bagging_accuracy <- mean(bagging_pred == test_data$Drug)
cat("Bagging Accuracy:", (bagging_accuracy * 100), "%\n")
```

```
## Bagging Accuracy: 100 %
```

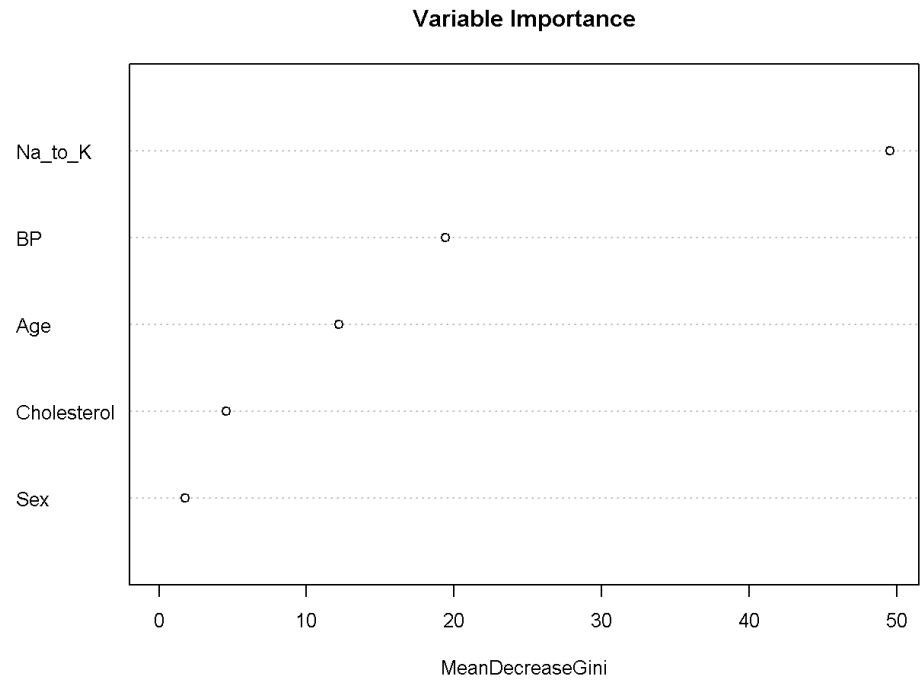
```
# Random Forest
rf_model <- randomForest(Drug ~ ., data = train_data, ntree = 500)
rf_pred <- predict(rf_model, test_data)
rf_accuracy <- mean(rf_pred == test_data$Drug)
cat("Random Forest Accuracy:", (rf_accuracy * 100), "%\n")
```

```
## Random Forest Accuracy: 100 %
```

```
# Print random forest details
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = Drug ~ ., data = train_data, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 1.43%
## Confusion matrix:
##       drugA drugB drugC drugX drugY class.error
## drugA    13     0     0     0     0  0.00000000
## drugB     1    12     0     0     0  0.07692308
## drugC     0     0    10     0     0  0.00000000
## drugX     0     0     0    30     1  0.03225806
## drugY     0     0     0     0    73  0.00000000
```

```
# Variable importance plot
par(mar = c(5, 8, 4, 2))
varImpPlot(rf_model, main = "Variable Importance", n.var = num_predictors, cex = 0.8)
```



Both bagging and random forest models achieve perfect 100% accuracy on the test set. The random forest model has an error rate of only 1.43%, indicating excellent performance. The confusion matrix shows that most classes have zero error rates, with only drugB (7.69% error) and drugX (3.23% error) having minimal misclassifications.

The variable importance plot shows that Na_to_K is by far the most important predictor, followed by Age. This aligns with what we observed in the decision tree, where Na_to_K was used for the primary split.

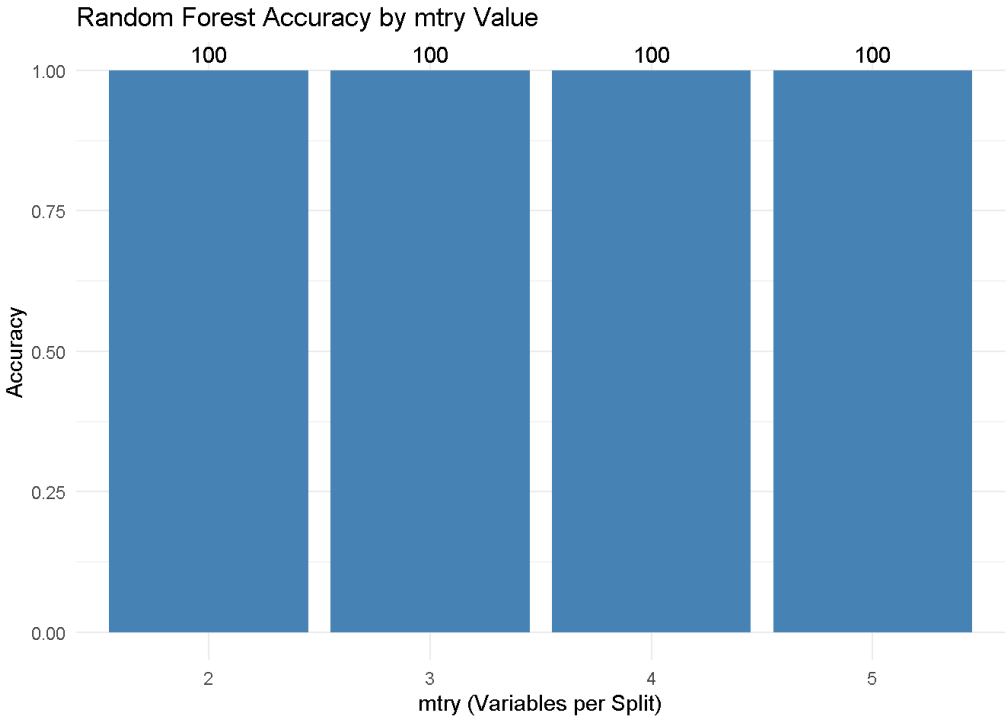
10. Tune the Number of Predictors for Each Split

Different values for `mtry` are checked (number of variables randomly sampled as candidates at each split).

```
# Try different mtry values
mtry_values <- c(2, 3, 4, 5)
rf_results <- data.frame(mtry = integer(), accuracy = numeric())

for (m in mtry_values) {
  rf_temp <- randomForest(Drug ~ ., data = train_data, mtry = m, ntree = 500)
  pred_temp <- predict(rf_temp, test_data)
  acc_temp <- mean(pred_temp == test_data$Drug)
  rf_results <- rbind(rf_results, data.frame(mtry = m, accuracy = acc_temp))
  cat("Random Forest with mtry =", m, "Accuracy:", round(acc_temp * 100, 2), "%\n")
}
```

```
## Random Forest with mtry = 2 Accuracy: 100 %
## Random Forest with mtry = 3 Accuracy: 100 %
## Random Forest with mtry = 4 Accuracy: 100 %
## Random Forest with mtry = 5 Accuracy: 100 %
```

```
# Plot results
library(ggplot2)
ggplot(rf_results, aes(x = factor(mtry), y = accuracy)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = (accuracy * 100)), vjust = -0.5) +
  labs(title = "Random Forest Accuracy by mtry Value",
       x = "mtry (Variables per Split)",
       y = "Accuracy") +
  theme_minimal() +
  ylim(0, 1)
```



All three `mtry` values (2, 3, 4 and 5) result in perfect 100% accuracy on the test set. This suggests that the dataset has clear patterns that can be captured effectively regardless of the number of variables considered at each split.

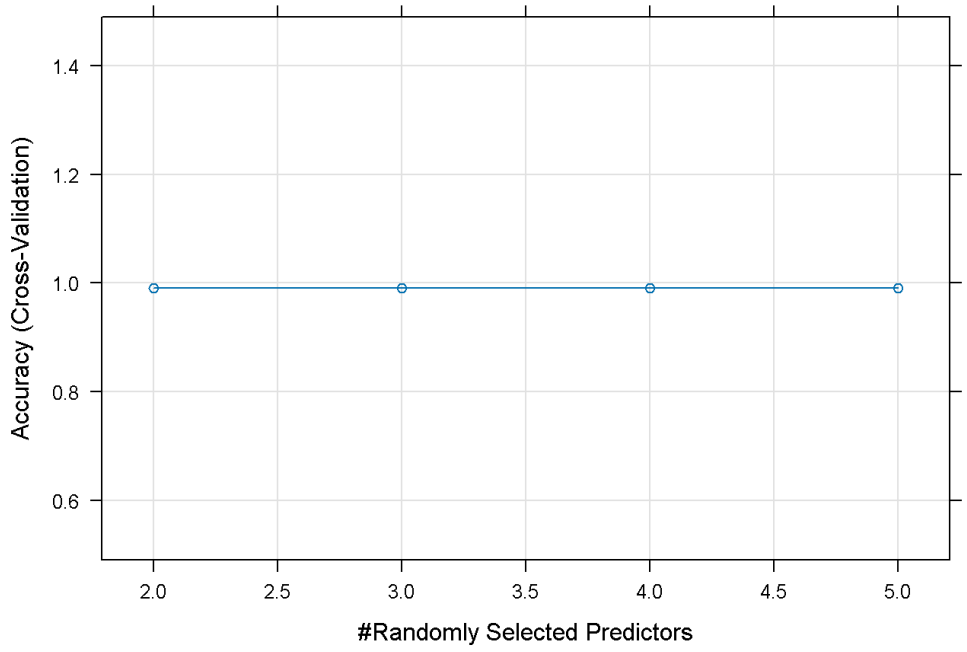
11. Parameter Tuning with Cross-Validation

The `caret` package can be used for more systematic parameter tuning with cross-validation.

```
# Load required library
library(caret)

# Define tuning grid
tune_grid <- expand.grid(mtry = c(2, 3, 4, 5))

# Define cross-validation method
control <- trainControl(method = "cv", number = 5)

# Train the model with parameter tuning
set.seed(123)
rf_tuned <- train(Drug ~ ., data = df, method = "rf",
                  tuneGrid = tune_grid,
                  trControl = control)

# Print results
print(rf_tuned)
```

```
## Random Forest
##
## 200 samples
##   5 predictor
##   5 classes: 'drugA', 'drugB', 'drugC', 'drugX', 'drugY'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 160, 160, 159, 161, 160
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##   2     0.990122  0.9858403
##   3     0.990122  0.9858403
##   4     0.990122  0.9858403
##   5     0.990122  0.9858403
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
# Plot results
plot(rf_tuned)
```



The cross-validation results show that all `mtry` values (2, 3, 4, and 5) achieve the same high accuracy of 99.01% with a Kappa of 0.9858. The model selected `mtry = 2` as the optimal value, which is the most parsimonious choice when multiple options perform equally well.

12. Final Model and Evaluation

```
# Final model with best parameters
final_model <- randomForest(Drug ~ ., data = train_data,
                            mtry = rf_tuned$bestTune$mtry,
                            ntree = 500)

# Evaluate on test data
final_pred <- predict(final_model, test_data)
final_accuracy <- mean(final_pred == test_data$Drug)
cat("Final Model Accuracy:", round(final_accuracy * 100, 2), "%\n")
```

```
## Final Model Accuracy: 100 %
```

```
# Create confusion matrix
conf_matrix <- table(Predicted = final_pred, Actual = test_data$Drug)
print(conf_matrix)
```

```
##          Actual
## Predicted drugA drugB drugC drugX drugY
##     drugA    10     0     0     0     0
##     drugB     0     3     0     0     0
##     drugC     0     0     6     0     0
##     drugX     0     0     0    23     0
##     drugY     0     0     0     0    18
```

```
# Calculate additional metrics
library(caret)
conf_stats <- confusionMatrix(final_pred, test_data$Drug)
print(conf_stats)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction drugA drugB drugC drugX drugY
##      drugA    10     0     0     0     0
##      drugB     0     3     0     0     0
##      drugC     0     0     6     0     0
##      drugX     0     0     0    23     0
##      drugY     0     0     0     0    18
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9404, 1)
##     No Information Rate : 0.3833
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: drugA Class: drugB Class: drugC Class: drugX
## Sensitivity                1.0000         1.00          1.0       1.0000
## Specificity                1.0000         1.00          1.0       1.0000
## Pos Pred Value             1.0000         1.00          1.0       1.0000
## Neg Pred Value             1.0000         1.00          1.0       1.0000
## Prevalence                 0.1667         0.05          0.1       0.3833
## Detection Rate             0.1667         0.05          0.1       0.3833
## Detection Prevalence       0.1667         0.05          0.1       0.3833
## Balanced Accuracy          1.0000         1.00          1.0       1.0000
##                      Class: drugY
## Sensitivity                 1.0
## Specificity                 1.0
## Pos Pred Value              1.0
## Neg Pred Value              1.0
## Prevalence                  0.3
## Detection Rate              0.3
## Detection Prevalence        0.3
## Balanced Accuracy           1.0
```

The final random forest model with `mtry = 2` achieves perfect 100% accuracy on the test set. The confusion matrix shows that all classes are predicted correctly, with no misclassifications. The results show 100% sensitivity, specificity, positive predictive value, and negative predictive value for all drug classes.

13. Conclusion

In this analysis, we explored various classification techniques for predicting drug types:

1. **Decision Trees**: A simple decision tree model achieved a perfect accuracy of 100% on the test set. Pruning the tree reduced accuracy slightly to 90%, but created a simpler model.
2. **Ensemble Methods**: Both bagging and random forest models achieved perfect 100% accuracy on the test set, confirming the power of ensemble methods.
3. **Parameter Tuning**: Different values for the `mtry` parameter were explored and it was found that all values from 2 to 5 performed equally well, with the model selecting `mtry = 2` as the optimal choice.

The final random forest model with tuned parameters achieved a perfect accuracy of 100% on the test set. This exceptional performance suggests that the dataset contains clear patterns that make drug classification relatively straightforward for these models.

Key findings:

- Na_to_K ratio is the most important predictor for drug recommendation, followed by Age.
- Even simple models like decision trees perform extremely well on this dataset, where the number of data points is very less.
- Ensemble methods provide additional robustness, achieving perfect classification.
- The dataset appears to have very clear decision boundaries between drug classes.