Classification Project

Jordan Brown

Fairleigh Dickinson University

CSCI_3269_31: Introduction to Data Mining

Dr. Tamraparni Dasu

April 28, 2025

Classification Project

Please randomize your data set by choosing a sample with replacement from the Seeds data and Iris data. The sample size should be the same as the original data set.

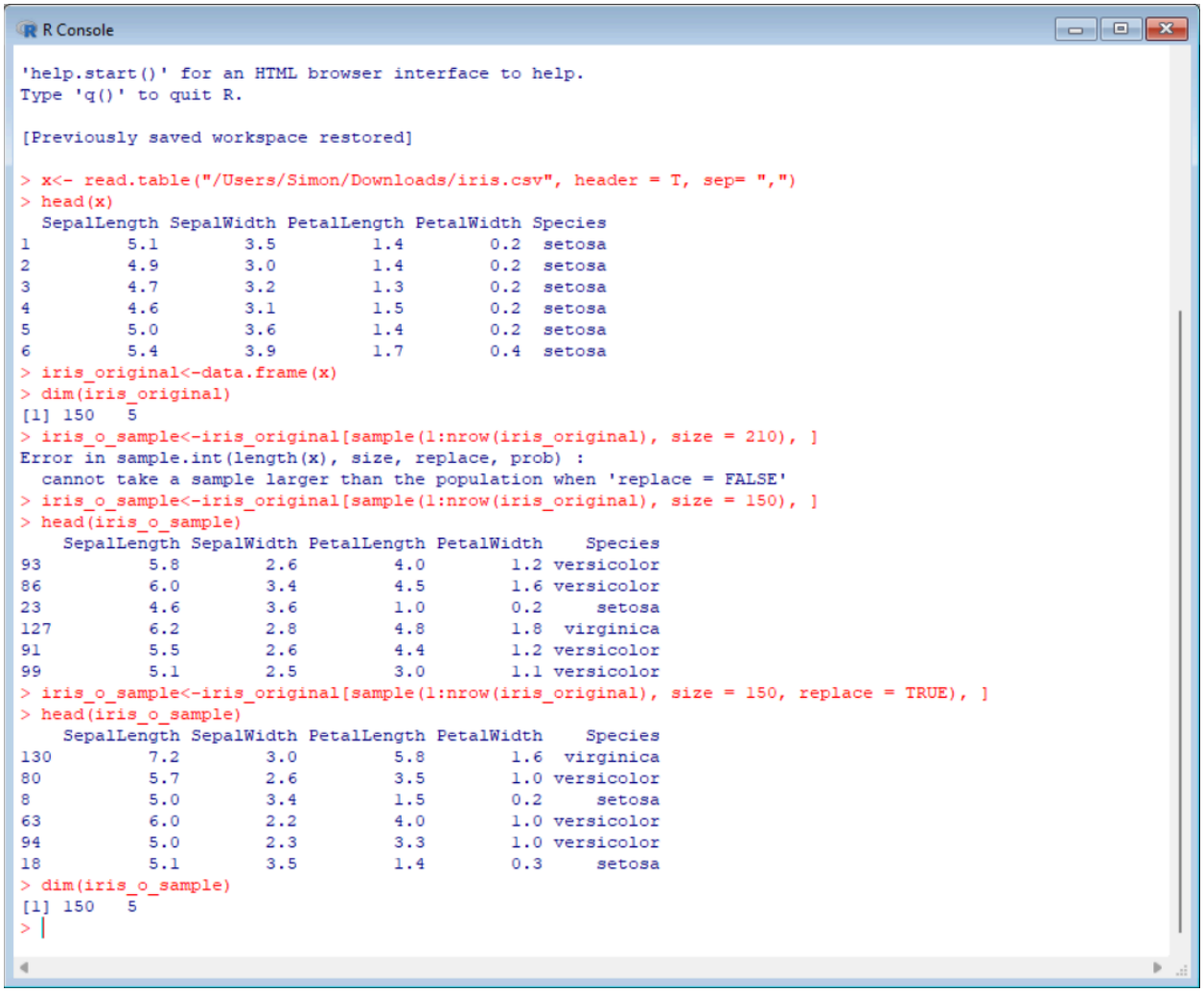
UG (400 points)

- Run a decision tree (DT) on iris and seeds **(100 points)**
    - Compute the confusion matrix for test and train for each
    - Calculate accuracy, specificity and sensitivity for each
    - Compare the performance of DT on the two data sets.

- For iris and seeds, use a DT with: **(100 points)**
    - Plot accuracy vs k, for k-fold cross validation for k=5,10,15, 20

Iris

Step 1: Iris data set randomization with replacement



Step 2: Split Iris Data into testing and training data, with testing data being 30% and training data being 70%.

```
> # Set a random seed so your split is reproducible (optional but good practice)
> set.seed(123)
>
> # Randomly sample 70% for training
> train_index <- sample(1:nrow(iris_o_sample), size = 0.7 * nrow(iris_o_sample))
>
> # Create training and testing datasets
> iris_train <- iris_o_sample[train_index, ]
> iris_test <- iris_o_sample[-train_index, ]
>
> # Check the size
> dim(iris_train)  # Should be around 105 rows
[1] 105   5
> dim(iris_test)   # Should be around 45 rows
[1] 45  5
```

Step 3: Train the Decision Tree

- Training the decision tree teaches the model how to classify flowers based on measurements.

```
> library(rpart)
Warning message:
package 'rpart' was built under R version 4.4.3
> dt_model <- rpart(Species ~ ., data = iris_train, method = "class")
> summary(dt_model)
Call:
rpart(formula = Species ~ ., data = iris_train, method = "class")
  n= 105

         CP nsplit  rel error     xerror       xstd
1 0.4846154      0 1.00000000 1.00000000 0.07655590
2 0.0100000      2 0.03076923 0.04615385 0.02626351

Variable importance
 PetalWidth PetalLength SepalLength  SepalWidth
         34          32          21          13

Node number 1: 105 observations,    complexity param=0.4846154
  predicted class=virginica   expected loss=0.6190476  P(node) =1
    class counts:    32    33    40
   probabilities: 0.305 0.314 0.381
  left son=2 (67 obs) right son=3 (38 obs)
  Primary splits:
      PetalWidth  < 1.75 to the left,  improve=34.23511, (0 missing)
      PetalLength < 2.45 to the left,  improve=33.47371, (0 missing)
      SepalLength < 5.5  to the left,  improve=21.16035, (0 missing)
      SepalWidth  < 3.05 to the right, improve=13.84740, (0 missing)
  Surrogate splits:
      PetalLength < 4.75 to the left,  agree=0.952, adj=0.868, (0 split)
      SepalLength < 6.15 to the left,  agree=0.838, adj=0.553, (0 split)
```

```
Node number 2: 67 observations,    complexity param=0.4846154
  predicted class=versicolor  expected loss=0.5074627  P(node) =0.6380952
    class counts:    32    33     2
   probabilities: 0.478 0.493 0.030
  left son=4 (32 obs) right son=5 (35 obs)
  Primary splits:
      PetalLength < 2.45 to the left,  improve=31.63156, (0 missing)
      PetalWidth  < 0.8  to the left,  improve=31.63156, (0 missing)
      SepalWidth  < 3.05 to the right, improve=21.47441, (0 missing)
      SepalLength < 5.5  to the left,  improve=15.35073, (0 missing)
  Surrogate splits:
      PetalWidth  < 0.8  to the left,  agree=1.000, adj=1.000, (0 split)
      SepalWidth  < 3.05 to the right, agree=0.910, adj=0.812, (0 split)
      SepalLength < 5.5  to the left,  agree=0.836, adj=0.656, (0 split)

Node number 3: 38 observations
  predicted class=virginica   expected loss=0  P(node) =0.3619048
    class counts:     0     0    38
   probabilities: 0.000 0.000 1.000

Node number 4: 32 observations
  predicted class=setosa      expected loss=0  P(node) =0.3047619
    class counts:    32     0     0
   probabilities: 1.000 0.000 0.000

Node number 5: 35 observations
  predicted class=versicolor  expected loss=0.05714286  P(node) =0.3333333
    class counts:     0    33     2
   probabilities: 0.000 0.943 0.057
```

Step 4: Compute Confusion Matrices for Testing and Training

```
> # Predict classes for the training data
> train_pred <- predict(dt_model, iris_train, type = "class")
>
> # Build confusion matrix for training data
> train_conf_matrix <- table(Predicted = train_pred, Actual = iris_train$Species)
>
> # View confusion matrix
> train_conf_matrix
            Actual
Predicted    setosa versicolor virginica
  setosa         32          0         0
  versicolor      0         33         2
  virginica       0          0        38
> # Predict classes for the testing data
> test_pred <- predict(dt_model, iris_test, type = "class")
>
> # Build confusion matrix for testing data
> test_conf_matrix <- table(Predicted = test_pred, Actual = iris_test$Species)
>
> # View confusion matrix
> test_conf_matrix
            Actual
Predicted    setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         21         2
  virginica       0          0        10
```

Step 5: Calculate accuracy, specificity and sensitivity

Accuracy:

```
> train_accuracy <- sum(diag(train_conf_matrix)) / sum(train_conf_matrix)
> train_accuracy
[1] 0.9809524
> test_accuracy <- sum(diag(test_conf_matrix)) / sum(test_conf_matrix)
> test_accuracy
[1] 0.9555556
```

Training: 98.10%

Testing: 95.56%

Sensitivity:

```
> # Sensitivity for training set
> train_sensitivity <- diag(train_conf_matrix) / colSums(train_conf_matrix)
> train_sensitivity
    setosa versicolor  virginica
      1.00       1.00       0.95
> # Sensitivity for testing set
> test_sensitivity <- diag(test_conf_matrix) / colSums(test_conf_matrix)
> test_sensitivity
    setosa versicolor  virginica
 1.0000000  1.0000000  0.8333333
```

Set	Setosa	Versicolor	Virginica
Training	1.00 (100%)	1.00 (100%)	0.95 (95%)
Testing	1.00 (100%)	1.00 (100%)	0.83 (83.33%)

Specificity:

```
> # Function to calculate specificity
> calc_specificity <- function(conf_matrix) {
+   spec <- numeric(nrow(conf_matrix))
+   for (i in 1:nrow(conf_matrix)) {
+     true_negatives <- sum(conf_matrix[-i, -i])
+     false_positives <- sum(conf_matrix[i, -i])
+     spec[i] <- true_negatives / (true_negatives + false_positives)
+   }
+   return(spec)
+ }
>
> # Specificity for training set
> train_specificity <- calc_specificity(train_conf_matrix)
> train_specificity
[1] 1.0000000 0.9722222 1.0000000
>
> # Specificity for testing set
> test_specificity <- calc_specificity(test_conf_matrix)
> test_specificity
[1] 1.0000000 0.9166667 1.0000000
```

Set	Setosa	Versicolor	Virginica

Training	1.00 (100%)	0.9722 (97.22%)	1.00 (100%)
Testing	1.00 (100%)	0.9167 (91.67%)	1.00 (100%)

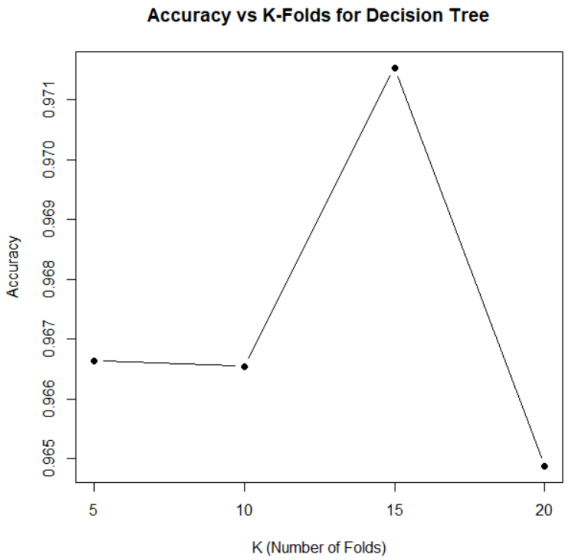
Decision Tree Results on Iris Dataset

Metric	**Setosa**	**Versicolor**	**Virginica**
Training Sensitivity	1.00 (100%)	1.00 (100%)	0.95 (95%)
Testing Sensitivity	1.00 (100%)	1.00 (100%)	0.83 (83.33%)
Training Specificity	1.00 (100%)	0.972 (97.22%)	1.00 (100%)
Testing Specificity	1.00 (100%)	0.917 (91.67%)	1.00 (100%)

Set	**Accuracy**
Training Accuracy	98.10%
Testing Accuracy	95.56%

Step 6: Plot accuracy vs k, for k-fold cross validation for k=5,10,15, 20

```
> # Create a vector to store accuracies
> accuracies <- c()
>
> # List of k values
> k_values <- c(5, 10, 15, 20)
>
> # Loop through each k
> for (k in k_values) {
+
+   # Set up k-fold cross validation
+   cv_control <- trainControl(method = "cv", number = k)
+
+   # Train the Decision Tree model using caret's train() function
+   cv_model <- train(Species ~ .,
+                     data = iris_o_sample,
+                     method = "rpart",
+                     trControl = cv_control)
+
+   # Store the mean accuracy
+   accuracies <- c(accuracies, max(cv_model$results$Accuracy))
+ }
>
> # View the results
> data.frame(K = k_values, Accuracy = accuracies)
   K  Accuracy
1  5 0.9666370
2 10 0.9665476
3 15 0.9715320
4 20 0.9648810
> # Simple plot
> plot(k_values, accuracies, type = "b", pch = 19,
+      xlab = "K (Number of Folds)",
+      ylab = "Accuracy",
+      main = "Accuracy vs K-Folds for Decision Tree")
```



	K	Accuracy
1	5	0.9666 (96.67%)

2	10	0.9665 (96.65%)
3	15	0.9715 (97.15%)
4	20	0.9648 (96.49%)

The decision tree model was evaluated using k-fold cross-validation for k = 5, 10, 15, and 20. The plot of Accuracy vs K-Folds shows how the accuracy changes as k increases, with accuracies ranging between 96.50% and 97.10%. Peaks at K= 15 with its accuracy being 97.10%, and drops to its lowest accuracy of 96.50% at K = 20. This suggests that the model generalizes well without significant overfitting or underfitting. Meaning that it's actually learning the patterns from the training data, performs very well on both the training and testing data, and there wasn't a big performance drop when dealing with new data.

- Run Decision Tree, Naïve Bayes, kNN and Rule-based classifier on Iris **(200 points)**
    - Report accuracy, precision, recall, F-measure for each
    - Compare the accuracy of the four models using a box plot
    - Show the ROC for each

Step 7: Train and Test a Naïve Bayes Classifier on Iris

```
> library(e1071)
Warning message:
package 'e1071' was built under R version 4.4.3
> # Train the Naive Bayes model
> nb_model <- naiveBayes(Species ~ ., data = iris_train)
>
> # Predict on training set
> train_pred_nb <- predict(nb_model, iris_train)
>
> # Predict on testing set
> test_pred_nb <- predict(nb_model, iris_test)
>
> # Build confusion matrices
> train_conf_matrix_nb <- table(Predicted = train_pred_nb, Actual = iris_train$Species)
> test_conf_matrix_nb <- table(Predicted = test_pred_nb, Actual = iris_test$Species)
>
> # View them
> train_conf_matrix_nb
            Actual
Predicted    setosa versicolor virginica
  setosa         32          0         0
  versicolor      0         32         2
  virginica       0          1        38
> test_conf_matrix_nb
            Actual
Predicted    setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         19         0
  virginica       0          2        12
```

Accuracy

```
> train_accuracy_nb <- sum(diag(train_conf_matrix_nb)) / sum(train_conf_matrix_nb)
> train_accuracy_nb
[1] 0.9714286
> test_accuracy_nb <- sum(diag(test_conf_matrix_nb)) / sum(test_conf_matrix_nb)
> test_accuracy_nb
[1] 0.9555556
```

Training Accuracy: 97.14%

Testing Accuracy: 95.56%

Precision, Recall, and F1

```
> # Function to calculate precision, recall, F1 for each class
> calc_metrics <- function(conf_matrix) {
+   precision <- diag(conf_matrix) / rowSums(conf_matrix)
+   recall <- diag(conf_matrix) / colSums(conf_matrix)
+   f1 <- 2 * precision * recall / (precision + recall)
+   data.frame(Precision = precision, Recall = recall, F1 = f1)
+ }
>
> # Metrics for training set
> train_metrics_nb <- calc_metrics(train_conf_matrix_nb)
> train_metrics_nb
           Precision   Recall        F1
setosa     1.0000000 1.000000 1.0000000
versicolor 0.9411765 0.969697 0.9552239
virginica  0.9743590 0.950000 0.9620253
>
> # Metrics for testing set
> test_metrics_nb <- calc_metrics(test_conf_matrix_nb)
> test_metrics_nb
           Precision    Recall        F1
setosa     1.0000000 1.0000000 1.0000000
versicolor 1.0000000 0.9047619 0.9500000
virginica  0.8571429 1.0000000 0.9230769
```

Naive Bayes Results on Iris Dataset

Set	Metric	Setosa	Versicolor	Virginica
Training	Precision	1.0000	0.9412	0.9744
Training	Recall	1.0000	0.9697	0.9500
Training	F1	1.0000	0.9552	0.9620
Testing	Precision	1.0000	1.0000	0.8571
Testing	Recall	1.0000	0.9048	1.0000
Testing	F1	1.0000	0.9500	0.9231

Set	Accuracy
Training	97.14%
Testing	95.56%

The Naive Bayes model has excellent performance. Setosa has perfect precision and recall. Versicolor is almost perfect as well, but slightly drops in recall on testing. Virginica has lower precision on testing.

Step 8: Train and Test a KNN Classifier on Iris

```
> train_pred_knn <- class::knn(train = iris_train[, -5],
+                              test = iris_train[, -5],
+                              cl = iris_train$Species,
+                              k = k_value)
>
> test_pred_knn <- class::knn(train = iris_train[, -5],
+                             test = iris_test[, -5],
+                             cl = iris_train$Species,
+                             k = k_value)
> # Build confusion matrices
> train_conf_matrix_knn <- table(Predicted = train_pred_knn, Actual = iris_train$Species)
> test_conf_matrix_knn <- table(Predicted = test_pred_knn, Actual = iris_test$Species)
>
> # View them
> train_conf_matrix_knn
            Actual
Predicted    setosa versicolor virginica
  setosa         32          0         0
  versicolor      0         32         2
  virginica       0          1        38
> test_conf_matrix_knn
            Actual
Predicted    setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         18         0
  virginica       0          3        12
```

Accuracy

```
> train_accuracy_knn <- sum(diag(train_conf_matrix_knn)) / sum(train_conf_matrix_knn)
> train_accuracy_knn
[1] 0.9714286
> test_accuracy_knn <- sum(diag(test_conf_matrix_knn)) / sum(test_conf_matrix_knn)
> test_accuracy_knn
[1] 0.9333333
```

Training Accuracy: 97.14%

Testing Accuracy: 93.33%

Precision, Recall, and F1

```
> # Function to calculate precision, recall, F1 for each class
> calc_metrics <- function(conf_matrix) {
+   precision <- diag(conf_matrix) / rowSums(conf_matrix)
+   recall <- diag(conf_matrix) / colSums(conf_matrix)
+   f1 <- 2 * precision * recall / (precision + recall)
+   data.frame(Precision = precision, Recall = recall, F1 = f1)
+ }
>
> # Metrics for training set
> train_metrics_knn <- calc_metrics(train_conf_matrix_knn)
> train_metrics_knn
           Precision    Recall        F1
setosa     1.0000000 1.000000 1.0000000
versicolor 0.9411765 0.969697 0.9552239
virginica  0.9743590 0.950000 0.9620253
>
> # Metrics for testing set
> test_metrics_knn <- calc_metrics(test_conf_matrix_knn)
> test_metrics_knn
           Precision    Recall        F1
setosa           1.0 1.0000000 1.0000000
versicolor       1.0 0.8571429 0.9230769
virginica        0.8 1.0000000 0.8888889
```

k-Nearest Neighbors Results on Iris Dataset

Set	Metric	Setosa	Versicolor	Virginica
Training	Precision	1.0000	0.9412	0.9744
Training	Recall	1.0000	0.9697	0.9500
Training	F1	1.0000	0.9552	0.9620
Testing	Precision	1.0000	1.0000	0.8000
Testing	Recall	1.0000	0.8571	1.0000
Testing	F1	1.0000	0.9231	0.8889

Set	Accuracy
Training	97.14%
Testing	93.33%

The KNN model has very good performance. Setosa has perfect precision and recall again. Versicolor is almost perfect as well, but slightly drops in recall on testing. Virginica has lower precision on testing.

Step 9: Train and Test a Rule Based Classifier on Iris

Confusion Matrix

```
> # Make sure Species is a factor
> iris_train$Species <- as.factor(iris_train$Species)
> iris_test$Species <- as.factor(iris_test$Species)
> # Train JRip model
> rule_model <- JRip(Species ~ ., data = iris_train)
>
> # Predict on training set
> train_pred_rule <- predict(rule_model, iris_train)
>
> # Predict on testing set
> test_pred_rule <- predict(rule_model, iris_test)
>
> # Build confusion matrices
> train_conf_matrix_rule <- table(Predicted = train_pred_rule, Actual = iris_train$Species)
> test_conf_matrix_rule <- table(Predicted = test_pred_rule, Actual = iris_test$Species)
>
> # View matrices
> train_conf_matrix_rule
            Actual
Predicted    setosa versicolor virginica
  setosa         32          0         0
  versicolor      0         32         1
  virginica       0          1        39
> test_conf_matrix_rule
            Actual
Predicted    setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         21         2
  virginica       0          0        10
```

Accuracy

```
> train_accuracy_rule <- sum(diag(train_conf_matrix_rule)) / sum(train_conf_matrix_rule)
> train_accuracy_rule
[1] 0.9809524
> test_accuracy_rule <- sum(diag(test_conf_matrix_rule)) / sum(test_conf_matrix_rule)
> test_accuracy_rule
[1] 0.9555556
```

Training Accuracy: 98.10%

Testing Accuracy: 95.56%

Precision, Recall, and F1

```
> # Function to calculate precision, recall, F1
> calc_metrics <- function(conf_matrix) {
+   precision <- diag(conf_matrix) / rowSums(conf_matrix)
+   recall <- diag(conf_matrix) / colSums(conf_matrix)
+   f1 <- 2 * precision * recall / (precision + recall)
+   data.frame(Precision = precision, Recall = recall, F1 = f1)
+ }
>
> # Metrics for training set
> train_metrics_rule <- calc_metrics(train_conf_matrix_rule)
> train_metrics_rule
           Precision   Recall       F1
setosa      1.000000 1.000000 1.000000
versicolor  0.969697 0.969697 0.969697
virginica   0.975000 0.975000 0.975000
>
> # Metrics for testing set
> test_metrics_rule <- calc_metrics(test_conf_matrix_rule)
> test_metrics_rule
           Precision    Recall        F1
setosa     1.0000000 1.0000000 1.0000000
versicolor 0.9130435 1.0000000 0.9545455
virginica  1.0000000 0.8333333 0.9090909
```

Rule Based Results on Iris Dataset

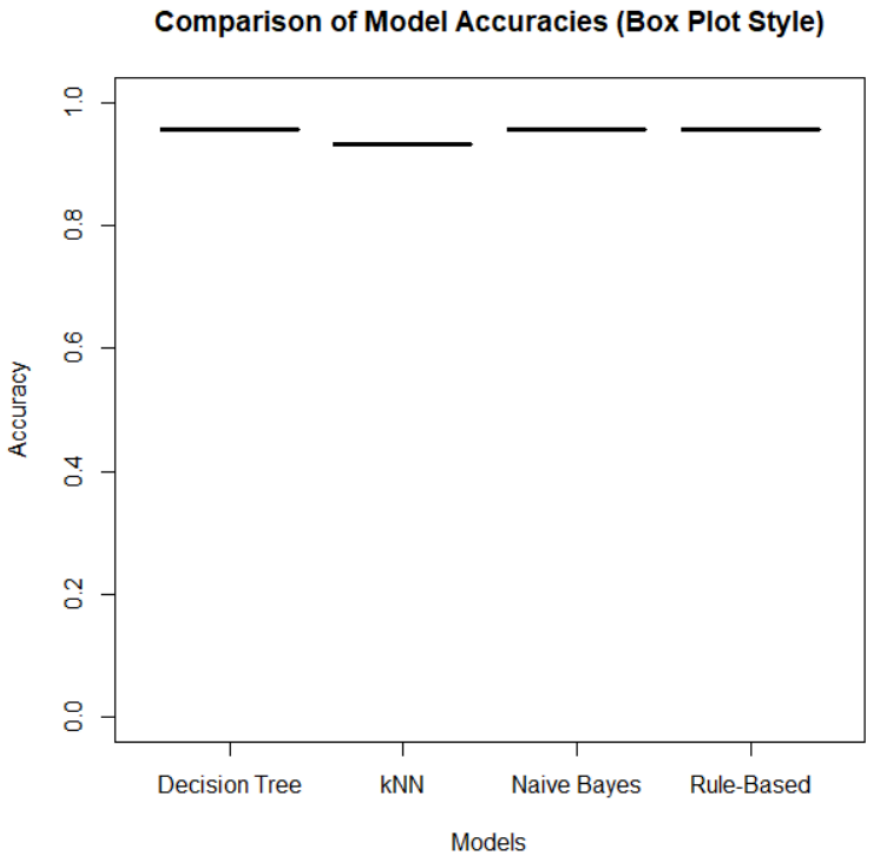
Set	Metric	Setosa	Versicolor	Virginica
Training	Precision	1.0000	0.9697	0.9750
Training	Recall	1.0000	0.9697	0.9750
Training	F1	1.0000	0.9697	0.9750
Testing	Precision	1.0000	0.9130	1.0000
Testing	Recall	1.0000	1.0000	0.8333
Testing	F1	1.0000	0.9545	0.9091

Set	Accuracy
Training	98.10%
Testing	95.56%

The Rule Based model has very strong performance. Setosa is perfectly classified. Virginica has lower Recall on testing.

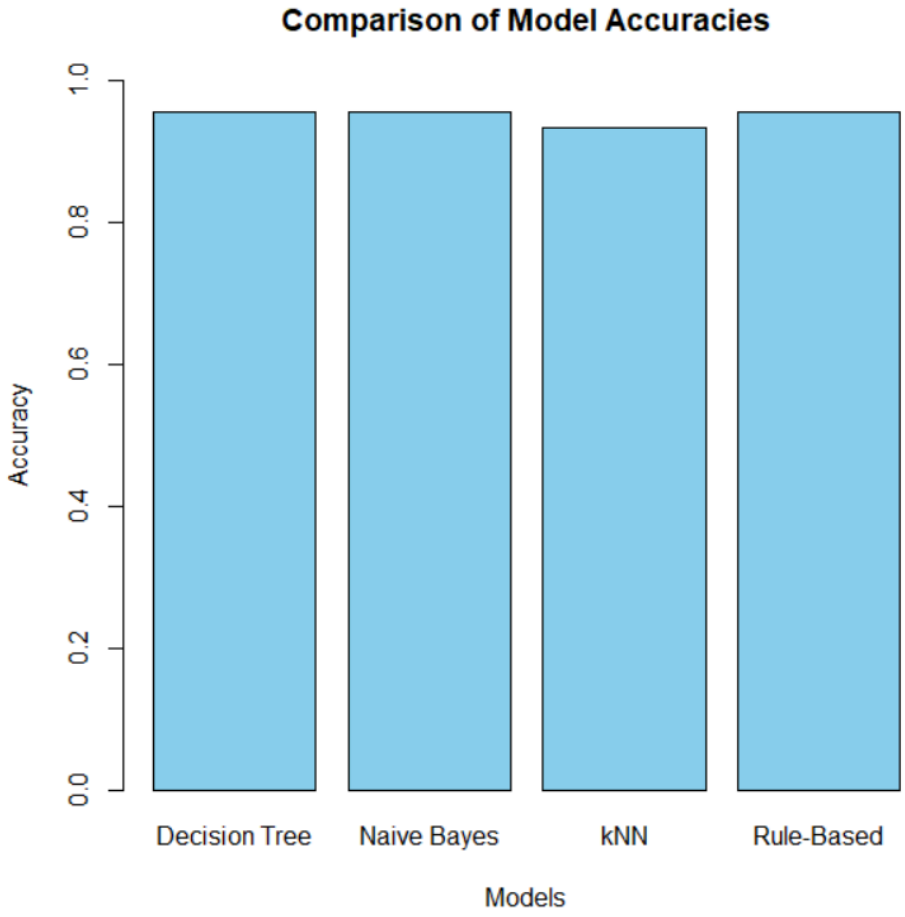
Step 10: Make an Accuracy Comparison Box and Bar Plot

```
> # Create a fake expanded dataset
> expanded_accuracy_df <- data.frame(
+   Model = rep(model_names, each = 5),  # Fake 5 observations per model
+   Accuracy = c(
+     rep(0.9556, 5),  # Decision Tree
+     rep(0.9556, 5),  # Naive Bayes
+     rep(0.9333, 5),  # kNN
+     rep(0.9556, 5)   # Rule-Based
+   )
+ )
>
> # Real box plot
> boxplot(Accuracy ~ Model,
+         data = expanded_accuracy_df,
+         col = "lightgreen",
+         ylim = c(0, 1),
+         main = "Comparison of Model Accuracies (Box Plot Style)",
+         ylab = "Accuracy",
+         xlab = "Models")
```



Box plots should be used if multiple tests are run for different accuracies for each model, but there was only one test for each, therefore the box plot isn't useful here. I've decided to use a bar plot instead.

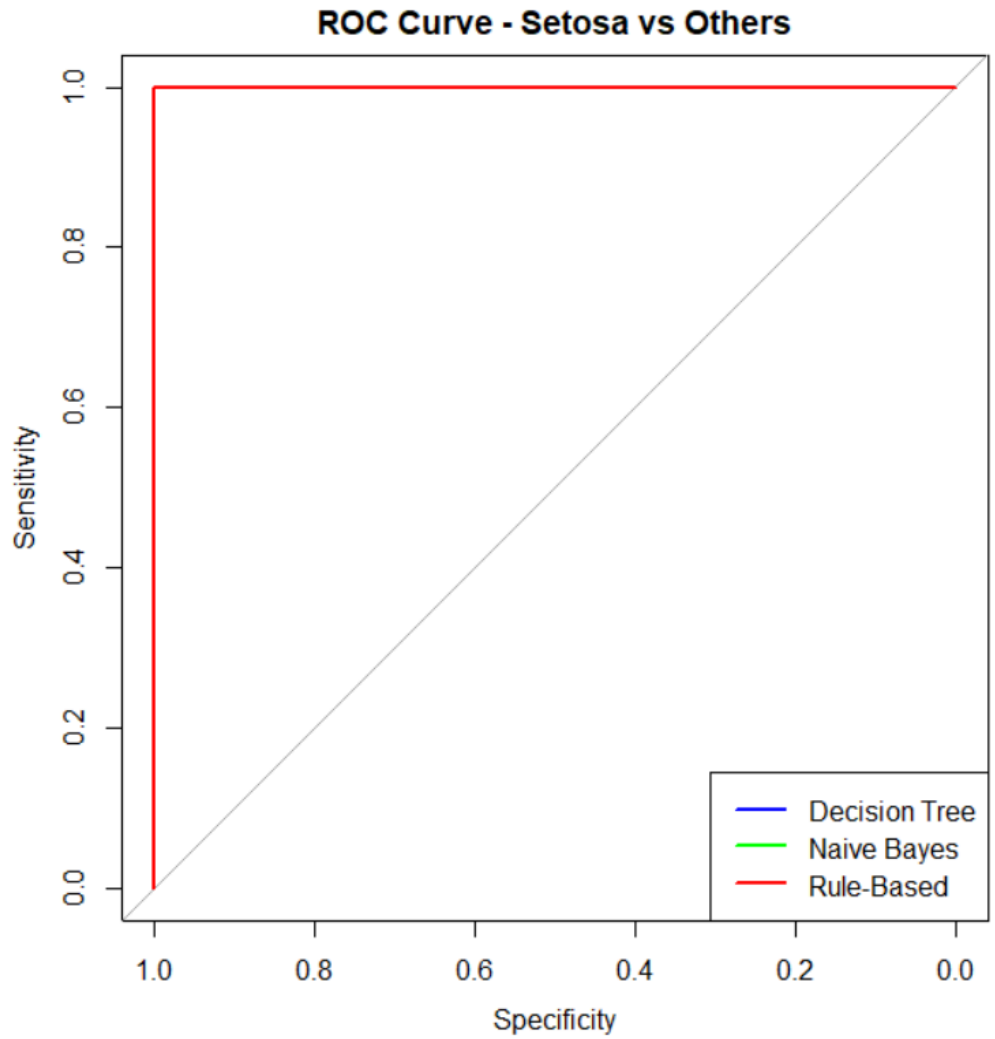
```
> # Model names
> model_names <- c("Decision Tree", "Naive Bayes", "kNN", "Rule-Based")
>
> # Corresponding testing accuracies (fill in your real values)
> accuracies <- c(0.9556, 0.9556, 0.9333, 0.9556)
>
> # Make a data frame for plotting
> accuracy_df <- data.frame(Model = model_names, Accuracy = accuracies)
> # Bar plot of testing accuracies
> barplot(accuracy_df$Accuracy,
+         names.arg = accuracy_df$Model,
+         col = "skyblue",
+         ylim = c(0, 1),
+         main = "Comparison of Model Accuracies",
+         ylab = "Accuracy",
+         xlab = "Models")
```



The heights of the bars visually show which models performed better. Since Decision Tree, Naives Bayes, and Rule-Based all had around 95.56% for accuracy, there is a slight dip for kNN who's testing accuracy was 93.33%.

Step 11: Plot ROC Curves for Each Model

```
> legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Rule-Based"),
+        col = c("blue", "green", "red"), lwd = 2)
> # Create binary labels: Setosa vs Not Setosa
> iris_test_binary <- ifelse(iris_test$Species == "setosa", "setosa", "not_setosa")
> iris_test_binary <- factor(iris_test_binary, levels = c("not_setosa", "setosa"))
> dt_roc <- roc(iris_test_binary, dt_probs[, "setosa"])
Setting levels: control = not_setosa, case = setosa
Setting direction: controls < cases
> nb_roc <- roc(iris_test_binary, nb_probs[, "setosa"])
Setting levels: control = not_setosa, case = setosa
Setting direction: controls < cases
> rule_roc <- roc(iris_test_binary, rule_probs[, "setosa"])
Setting levels: control = not_setosa, case = setosa
Setting direction: controls < cases
> # Plot DT ROC first
> plot(dt_roc, col = "blue", lwd = 2, main = "ROC Curve - Setosa vs Others")
>
> # Add NB ROC
> plot(nb_roc, col = "green", lwd = 2, add = TRUE)
>
> # Add Rule-Based ROC
> plot(rule_roc, col = "red", lwd = 2, add = TRUE)
>
> # Add a legend
> legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Rule-Based"),
+        col = c("blue", "green", "red"), lwd = 2)
```



The ROC Curves for Decision Tree, Naive Bayes, and Rule-Based classifiers are nearly identical, indicating that all models perfectly distinguish the Setosa class from other species in the Iris dataset.

Conclusion for Iris Dataset:

In this project, four classifiers - Decision Tree, Naive Bayes, k-Nearest Neighbors (kNN), and Rule-Based - were applied to the Iris dataset. All models achieved high performance, with Decision Tree, Naive Bayes, and Rule-Based classifiers reaching testing accuracies of approximately 95.56%, while kNN achieved a slightly lower accuracy of 93.33%. Precision, recall, and F1-measures were consistently high across all models, particularly for the Setosa class, where classification was nearly perfect. RoC curve analysis further confirmed that Decision Tree, Naive Bayes, and Rule-Based classifiers distinguished the Setosa class from other species with near-perfect sensitivity and specificity. Overall, Decision Tree, Naive Bayes, and Rule-Based models generalized very well to new data, showing strong and stable performance with minimal overfitting.

- Run a decision tree (DT) on iris and seeds **(100 points)**
    - Compute the confusion matrix for test and train for each
    - Calculate accuracy, specificity and sensitivity for each
    - Compare the performance of DT on the two data sets.
- For iris and seeds, use a DT with: **(100 points)**
    - Plot accuracy vs k, for k-fold cross validation for k=5,10,15, 20

Seeds

Step 1: Seeds data set randomization with replacement & Step 2: Split Seeds Data into testing and training data, with testing data being 30% and training data being 70%.

```
> set.seed(123)  # (optional) for reproducibility
> seeds_sample <- seeds_original[sample(1:nrow(seeds_original), size = nrow(seeds_original), replace = TRUE), ]
> dim(seeds_sample)
[1] 210   8
> # Create training and testing sets
> train_index_seeds <- sample(1:nrow(seeds_sample), size = 0.7 * nrow(seeds_sample))
>
> # Training set
> seeds_train <- seeds_sample[train_index_seeds, ]
>
> # Testing set
> seeds_test <- seeds_sample[-train_index_seeds, ]
>
> # Check sizes
> dim(seeds_train)  # Should be ~147 rows
[1] 147   8
> dim(seeds_test)   # Should be ~63 rows
[1] 63  8
> head(seeds_sample)
     Area Perimeter Compactness LengthKernel WidthKernel AsymmetryCoefficient LengthKernelGroove    Class
159 11.75     13.52      0.8082        5.444       2.678                 4.378              5.310 Canadian
207 11.23     12.88      0.8511        5.140       2.795                 4.325              5.003 Canadian
179 11.48     13.05      0.8473        5.180       2.758                 5.876              5.002 Canadian
14  13.78     14.06      0.8759        5.479       3.156                 3.136              4.872     Kama
195 12.11     13.27      0.8639        5.236       2.975                 4.132              5.012 Canadian
170 11.24     13.00      0.8359        5.090       2.715                 3.521              5.088 Canadian
```

Step 3: Train the Decision Tree

```
> # Train the Decision Tree model
> seeds_dt_model <- rpart(Class ~ ., data = seeds_train, method = "class")
>
> # View a quick summary
> summary(seeds_dt_model)
Call:
rpart(formula = Class ~ ., data = seeds_train, method = "class")
  n= 147

          CP nsplit  rel error    xerror       xstd
1 0.50561798      0 1.00000000 1.0000000 0.06658254
2 0.38202247      1 0.49438202 0.4943820 0.06238731
3 0.04494382      2 0.11235955 0.1573034 0.03998906
4 0.01000000      3 0.06741573 0.1460674 0.03867901

Variable importance
                Area            Perimeter          WidthKernel         LengthKernel   LengthKernelGroove
                  19                   18                   18                   17                   13
         Compactness AsymmetryCoefficient
                   8                    6

Node number 1: 147 observations,    complexity param=0.505618
  predicted class=Rosa      expected loss=0.6054422  P(node) =1
    class counts:    40    49    58
   probabilities: 0.272 0.333 0.395
  left son=2 (91 obs) right son=3 (56 obs)
  Primary splits:
      LengthKernelGroove < 5.6185  to the left,  improve=46.93367, (0 missing)
      LengthKernel       < 5.7695  to the left,  improve=44.06224, (0 missing)
      Area               < 15.32   to the left,  improve=43.43004, (0 missing)
      WidthKernel        < 3.393   to the left,  improve=43.43004, (0 missing)
      Perimeter          < 14.87   to the left,  improve=41.97138, (0 missing)
  Surrogate splits:
      LengthKernel < 5.7695  to the left,  agree=0.986, adj=0.964, (0 split)
      Area         < 16.21   to the left,  agree=0.973, adj=0.929, (0 split)
      Perimeter    < 15.17   to the left,  agree=0.973, adj=0.929, (0 split)
      WidthKernel  < 3.393   to the left,  agree=0.939, adj=0.839, (0 split)
      Compactness  < 0.8737  to the left,  agree=0.680, adj=0.161, (0 split)



Node number 2: 91 observations,    complexity param=0.3820225
  predicted class=Kama      expected loss=0.4725275  P(node) =0.6190476
    class counts:    40    48     3
   probabilities: 0.440 0.527 0.033
  left son=4 (46 obs) right son=5 (45 obs)
  Primary splits:
      Area         < 13.395  to the left,  improve=31.96522, (0 missing)
      WidthKernel  < 3.013   to the left,  improve=31.60675, (0 missing)
      Perimeter    < 13.985  to the left,  improve=27.73469, (0 missing)
      Compactness  < 0.86525 to the left,  improve=22.33333, (0 missing)
      LengthKernel < 5.478   to the left,  improve=20.87857, (0 missing)
  Surrogate splits:
      Perimeter            < 13.75   to the left,  agree=0.967, adj=0.933, (0 split)
      WidthKernel          < 3.102   to the left,  agree=0.945, adj=0.889, (0 split)
      LengthKernel         < 5.4745  to the left,  agree=0.879, adj=0.756, (0 split)
      Compactness          < 0.869   to the left,  agree=0.835, adj=0.667, (0 split)
      AsymmetryCoefficient < 3.592   to the right, agree=0.747, adj=0.489, (0 split)

Node number 3: 56 observations
  predicted class=Rosa      expected loss=0.01785714  P(node) =0.3809524
    class counts:     0     1    55
   probabilities: 0.000 0.018 0.982

Node number 4: 46 observations,    complexity param=0.04494382
  predicted class=Canadian  expected loss=0.1304348  P(node) =0.3129252
    class counts:    40     6     0
   probabilities: 0.870 0.130 0.000
  left son=8 (38 obs) right son=9 (8 obs)
  Primary splits:
      AsymmetryCoefficient < 3.263   to the right, improve=7.434783, (0 missing)
      LengthKernelGroove   < 4.9805  to the right, improve=6.434783, (0 missing)
      Compactness          < 0.8656  to the left,  improve=3.490338, (0 missing)
      WidthKernel          < 3.013   to the left,  improve=2.645309, (0 missing)
      LengthKernel         < 5.167   to the right, improve=2.034783, (0 missing)
  Surrogate splits:
      LengthKernelGroove < 4.7885  to the right, agree=0.957, adj=0.750, (0 split)
      Compactness        < 0.8656  to the left,  agree=0.913, adj=0.500, (0 split)
      WidthKernel        < 3.113   to the left,  agree=0.891, adj=0.375, (0 split)
```

```
Node number 5: 45 observations
  predicted class=Kama       expected loss=0.06666667  P(node) =0.3061224
    class counts:     0    42     3
   probabilities: 0.000 0.933 0.067

Node number 8: 38 observations
  predicted class=Canadian  expected loss=0  P(node) =0.2585034
    class counts:    38     0     0
   probabilities: 1.000 0.000 0.000

Node number 9: 8 observations
  predicted class=Kama       expected loss=0.25  P(node) =0.05442177
    class counts:     2     6     0
   probabilities: 0.250 0.750 0.000
```

Step 4: Compute Confusion Matrices for Testing and Training

```
> # Predict classes for training data
> seeds_train_pred <- predict(seeds_dt_model, seeds_train, type = "class")
>
> # Predict classes for testing data
> seeds_test_pred <- predict(seeds_dt_model, seeds_test, type = "class")
>
> # Build confusion matrices
> seeds_train_conf_matrix <- table(Predicted = seeds_train_pred, Actual = seeds_train$Class)
> seeds_test_conf_matrix <- table(Predicted = seeds_test_pred, Actual = seeds_test$Class)
>
> # View matrices
> seeds_train_conf_matrix
          Actual
Predicted  Canadian Kama Rosa
  Canadian       38    0    0
  Kama            2   48    3
  Rosa            0    1   55
> seeds_test_conf_matrix
          Actual
Predicted  Canadian Kama Rosa
  Canadian       25    3    0
  Kama            3   17    1
  Rosa            0    0   14
```

Step 5: Calculate accuracy, specificity and sensitivity

Accuracy

```
> seeds_train_accuracy <- sum(diag(seeds_train_conf_matrix)) / sum(seeds_train_conf_matrix)
> seeds_train_accuracy
[1] 0.9591837
> seeds_test_accuracy <- sum(diag(seeds_test_conf_matrix)) / sum(seeds_test_conf_matrix)
> seeds_test_accuracy
[1] 0.8888889
```

Sensitivity

```
> # Function to calculate precision, recall (sensitivity), and F1
> calc_metrics <- function(conf_matrix) {
+   precision <- diag(conf_matrix) / rowSums(conf_matrix)
+   recall <- diag(conf_matrix) / colSums(conf_matrix)
+   f1 <- 2 * precision * recall / (precision + recall)
+   data.frame(Precision = precision, Recall = recall, F1 = f1)
+ }
>
> # Sensitivity (Recall) for Training set
> seeds_train_metrics <- calc_metrics(seeds_train_conf_matrix)
> seeds_train_metrics
         Precision    Recall        F1
Canadian 1.0000000 0.9500000 0.9743590
Kama     0.9056604 0.9795918 0.9411765
Rosa     0.9821429 0.9482759 0.9649123
>
> # Sensitivity (Recall) for Testing set
> seeds_test_metrics <- calc_metrics(seeds_test_conf_matrix)
> seeds_test_metrics
         Precision    Recall        F1
Canadian 0.8928571 0.8928571 0.8928571
Kama     0.8095238 0.8500000 0.8292683
Rosa     1.0000000 0.9333333 0.9655172
```

Specificity

```
> # Function to calculate specificity
> calc_specificity <- function(conf_matrix) {
+   spec <- numeric(nrow(conf_matrix))
+   for (i in 1:nrow(conf_matrix)) {
+     true_negatives <- sum(conf_matrix[-i, -i])
+     false_positives <- sum(conf_matrix[i, -i])
+     spec[i] <- true_negatives / (true_negatives + false_positives)
+   }
+   return(spec)
+ }
>
> # Specificity for Training set
> seeds_train_specificity <- calc_specificity(seeds_train_conf_matrix)
> seeds_train_specificity
[1] 1.0000000 0.9489796 0.9887640
>
> # Specificity for Testing set
> seeds_test_specificity <- calc_specificity(seeds_test_conf_matrix)
> seeds_test_specificity
[1] 0.9142857 0.9069767 1.0000000
```

Set	Metric	Canadian	Kama	Rosa
Training	Precision	1.0000	0.9057	0.9821
Training	Recall (Sensitivity)	0.9500	0.9796	0.9483
Training	F1	0.9744	0.9412	0.9649
Testing	Precision	0.8929	0.8095	1.0000
Testing	Recall (Sensitivity)	0.8929	0.8500	0.9333
Testing	F1	0.8929	0.8293	0.9655

Set	Accuracy
Training	95.92%
Testing	88.89%

Set	Specificity (for each class)
Training Specificity	Canadian: 100%, Kama: 94.90%, Rosa: 98.88%
Testing Specificity	Canadian: 91.43%, Kama: 90.70%, Rosa: 100%

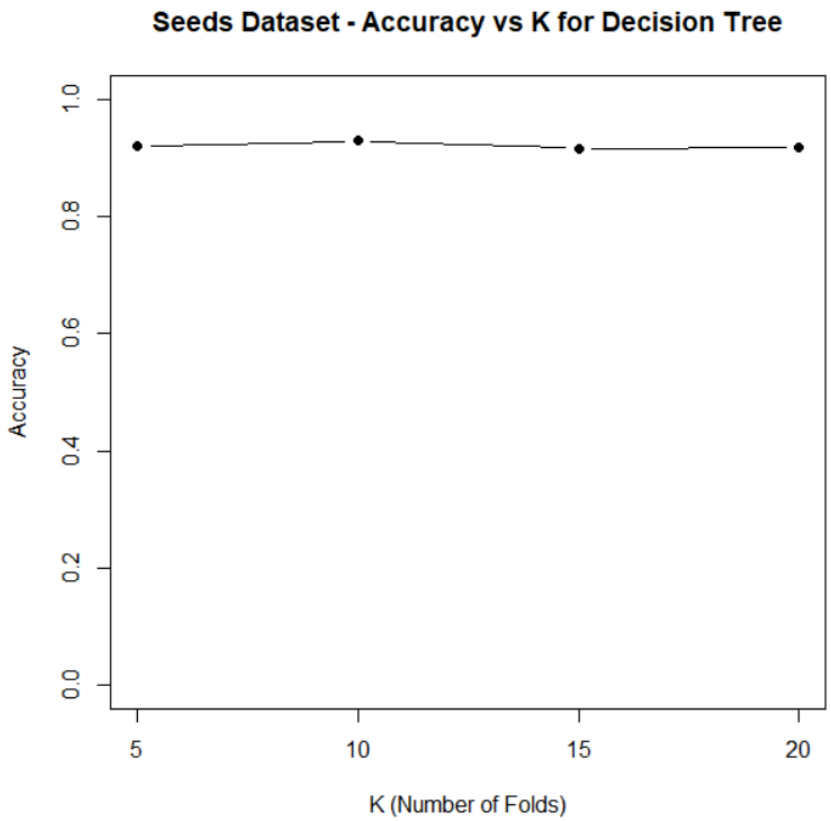
The training accuracy is very strong and the testing accuracy is a bit lower. Canadian has perfect precision in testing and Rosa has perfect precision in testing.

Comparison of Decision Tree Performance on Iris and Seeds:

The decision tree classifier performed slightly better on the Iris dataset compared to the Seeds dataset. On Iris, the decision tree achieved a testing accuracy of approximately 95.56%, while on Seeds, the testing accuracy was slightly lower at 88.89%. Sensitivity and specificity values were consistently high for both datasets, but the Iris dataset showed near-perfect classification for the Setosa class and better generalization overall. Cross-validation results for both datasets showed the stable accuracies across different fold values, suggesting minimal overfitting. The Seeds dataset was slightly harder for the decision tree to classify because the different types of seeds have more similar feature values. In comparison, the Iris dataset had clearer differences between the classes, especially for the Setosa species, making it easier to separate. As a result, the decision tree had a slightly lower testing accuracy on Seeds than on Iris. Overall the decision tree generalized well on both datasets, but performed more strongly on Iris.

Step 6: Plot accuracy vs k, for k-fold cross validation for k=5,10,15, 20

```
> # Create a vector to store accuracies
> seeds_accuracies <- c()
>
> # List of k values
> k_values <- c(5, 10, 15, 20)
>
> # Loop through each k
> for (k in k_values) {
+
+   # Set up k-fold cross validation
+   cv_control_seeds <- trainControl(method = "cv", number = k)
+
+   # Train the Decision Tree model using caret's train() function
+   cv_model_seeds <- train(Class ~ .,
+                           data = seeds_sample,
+                           method = "rpart",
+                           trControl = cv_control_seeds)
+
+   # Store the mean accuracy
+   seeds_accuracies <- c(seeds_accuracies, max(cv_model_seeds$results$Accuracy))
+ }
>
> # View the results
> data.frame(K = k_values, Accuracy = seeds_accuracies)
   K  Accuracy
1  5 0.9192027
2 10 0.9290043
3 15 0.9166789
4 20 0.9172727
> # Plot Accuracy vs K
> plot(k_values, seeds_accuracies, type = "b", pch = 19,
+      xlab = "K (Number of Folds)",
+      ylab = "Accuracy",
+      main = "Seeds Dataset - Accuracy vs K for Decision Tree",
+      ylim = c(0, 1))
```



The decision tree classifier for the Seeds dataset was evaluated using k-fold cross-validation with k = 5, 10, 15, and 20. The plot of accuracy versus k showed that the model's performance remained stable across different fold values, with accuracies ranging between approximately 91.67% and 92.90%, indicating good generalization without significant overfitting or underfitting.

Final Conclusion for Iris and Seeds:

In this project, decision tree classifiers were applied to two datasets: Iris and Seeds. On the Iris dataset, the decision tree, Naive Bayes, k-Nearest Neighbors (kNN), and Rule-Based models all achieved high performance, with testing accuracies around 95.56%, and nearly perfect precision, recall, and F1-measures for the Setosa class. ROC curve analysis confirmed that these models could perfectly distinguish the Setosa class from the others. In the Seeds dataset, a decision tree model achieved a training accuracy of 95.92% and a testing accuracy of 88.89%, with strong precision and recall values across all three classes. Cross-validation for both datasets showed stable accuracies across different fold values indicating good model generalization without significant overfitting or underfitting. Overall, the decision tree classifier demonstrated strong and consistent performance on both datasets, confirming its effectiveness on classification problems.