# Classification

Blake Pappas

2023-12-17

## Classification in R

## Load the following packages:

```
library(caret)
library(rpart)
library(rpart.plot)
library(class)
library(dplyr)
```

## In this exercise, we use the "wine.csv" file.

## The goal is to predict the wine "Type" based on the other attributes.

## P1: Import the dataset:

```
Wine = read.csv("wine.csv")
```

## P2: Split the dataset into 75% training and 25% testing.

## Use createDataPartition() function to do it:

```
library(caret)

train_rows = createDataPartition(y = Wine$Type,
                                 p = 0.75, list = FALSE)
WineData_train = Wine[train_rows, ]
WineData_test = Wine[-train_rows, ]
```

## P3: Build a decision tree model:

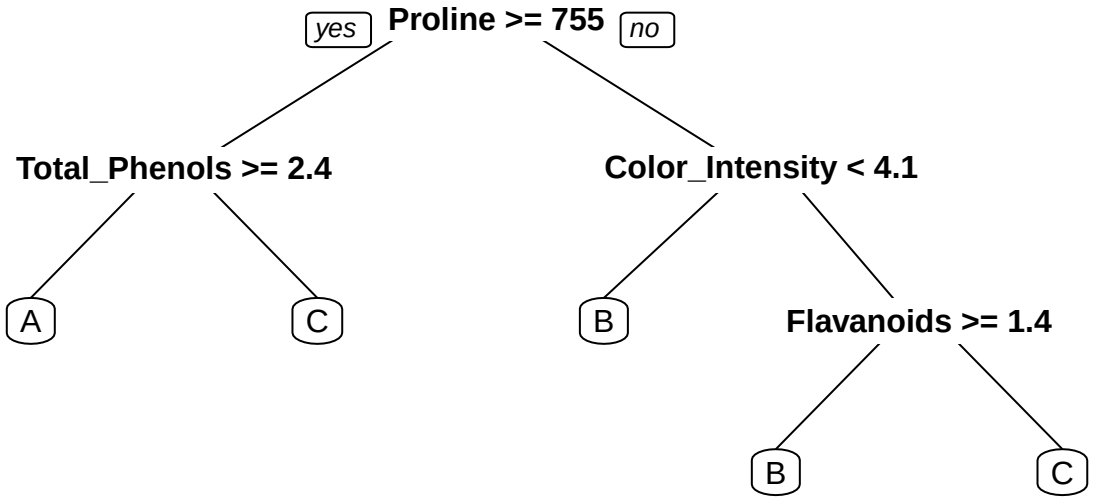
```
library(rpart)

tree = rpart(Type ~ ., data = WineData_train,
             method = "class",
             parms = list(split = "information"))
```

## P4: Plot the tree, and interpret a decision rule of your choice:

```
library(rpart.plot)

prp(tree, varlen = 0)
```



**Answer: IF (Flavanoids $\geq$ 1.6) AND (Proline $\geq$ 725) AND Alcohol $\geq$ 13 THEN Type A.**

**P5: Evaluate the performance of your tree.**

**Specifically, get the confusion matrix, and report the accuracy, precision, and recall.**

**Answer: The accuracy is 0.8837. The precision is 1.0000, 0.8000, and 0.9231 for Types A, B, and C, respectively. The recall is 0.7143, 0.9412, and 1.0000 for Types A, B, and C, respectively.**

```
pred_tree = predict(tree, WineData_test, type = "class")

confusionMatrix(pred_tree, as.factor(WineData_test$Type), mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C
##          A 13  2  0
##          B  1 15  2
##          C  0  0 10
##
## Overall Statistics
##
##                Accuracy : 0.8837
##                  95% CI : (0.7492, 0.9611)
##     No Information Rate : 0.3953
##     P-Value [Acc > NIR] : 4.107e-11
##
##                   Kappa : 0.8228
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C
## Precision              0.8667   0.8333   1.0000
## Recall                 0.9286   0.8824   0.8333
## F1                     0.8966   0.8571   0.9091
## Prevalence             0.3256   0.3953   0.2791
## Detection Rate         0.3023   0.3488   0.2326
## Detection Prevalence   0.3488   0.4186   0.2326
## Balanced Accuracy      0.9298   0.8835   0.9167
```

**P6: Now, try K-NN.**

**The first thing is to consider whether or not to normalize your data.**

**Answer: Yes, the data needs to be normalized.**

```
library(dplyr)

normalize = function(x) {
  return ((x - min(x))/(max(x) - min(x)))
  }

Wine_normalized = Wine %>% mutate_at(2:14, normalize)

train_rows = createDataPartition(y = Wine_normalized$Type,
                                 p = 0.75, list = FALSE)

Wine_normalized_train = Wine_normalized[train_rows, ]

Wine_normalized_test = Wine_normalized[-train_rows, ]
```

**P7: Build a K-NN classifier. Use 5-fold cross-validation to evaluate its performance based on average accuracy.**

**Report accuracy measure for k = 2, . . . , 10.**

```
# Cross-Validation
cv = createFolds(y = Wine_normalized$Type, k = 5)

# Make a Vector to Store Accuracy from Each Fold
accuracy = c()

# Loop Through Each Fold
for (test_rows in cv) {
  Wine_normalized_train = Wine_normalized[-test_rows, ]
  Wine_normalized_test = Wine_normalized[test_rows, ]

  # Then, Train Your Model and Evaluate Its Performance
  tree = rpart(Type ~ ., data = Wine_normalized_train,
               method = "class", parms = list(split = "information"))

  pred_tree = predict(tree, Wine_normalized_test, type = "class")

  cm = confusionMatrix(pred_tree, as.factor(Wine_normalized_test$Type))
```

```
  # Add the Accuracy of Current Fold
  accuracy = c(accuracy, cm$overall[1])
  print(accuracy)
}
```

```
##  Accuracy
## 0.8333333
##  Accuracy  Accuracy
## 0.8333333 0.8888889
##  Accuracy  Accuracy  Accuracy
## 0.8333333 0.8888889 0.9428571
##  Accuracy  Accuracy  Accuracy  Accuracy
## 0.8333333 0.8888889 0.9428571 0.9714286
##  Accuracy  Accuracy  Accuracy  Accuracy  Accuracy
## 0.8333333 0.8888889 0.9428571 0.9714286 0.8888889
```

```
# Average Accuracy Across Folds
print(mean(accuracy))
```

```
## [1] 0.9050794
```

**P8: Imagine that you forgot to normalize the data.**

**Build a K-NN model without normalization.**

**Pick any k value.**

**Answer: I picked a k value of three, for the three nearest neighbors.**

**What happens to performance?**

**Answer: By building a K-NN model without normalization, performance degrades, as is evidenced by a lower accuracy number in relation to the accuracy number for the model that uses normalization.**

```
library(caret)

train_rows = createDataPartition(y = Wine$Type,
                                 p = 0.75, list = FALSE)
WineData_train = Wine[train_rows, ]
WineData_test = Wine[-train_rows, ]

pred_knn = knn(train = WineData_train[, 2:14],
```

```
              test = WineData_test[, 2:14],
              cl = WineData_train$Type, k = 3)

confusionMatrix(pred_knn, as.factor(WineData_test$Type))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C
##          A 14  2  2
##          B  0 12  5
##          C  0  3  5
##
## Overall Statistics
##
##                Accuracy : 0.7209
##                  95% CI : (0.5633, 0.8467)
##     No Information Rate : 0.3953
##     P-Value [Acc > NIR] : 1.541e-05
##
##                   Kappa : 0.5743
##
##  Mcnemar's Test P-Value : 0.2123
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C
## Sensitivity            1.0000   0.7059   0.4167
## Specificity            0.8621   0.8077   0.9032
## Pos Pred Value         0.7778   0.7059   0.6250
## Neg Pred Value         1.0000   0.8077   0.8000
## Prevalence             0.3256   0.3953   0.2791
## Detection Rate         0.3256   0.2791   0.1163
## Detection Prevalence   0.4186   0.3953   0.1860
## Balanced Accuracy      0.9310   0.7568   0.6599
```