Module 2 Practice

ALY 6040

Jeff Hackmeister

2025-04-20

Introduction

For this classification project, we are working with a provided dataset, Mushrooms. To begin, we will load all necessary libraries for the project.

```
library(rpart,quietly = TRUE)
library(caret,quietly = TRUE)
library(rpart.plot,quietly = TRUE)
library(rattle)
library(readxl)
```

To begin, we'll read in the provided Excel spreadsheet as a dataframe and take a look at the structure of the data we'll be working with .

```
mushrooms <- read_excel("mushrooms.xlsx")
str(mushrooms)
```

```
tibble [8,124 x 23] (S3: tbl_df/tbl/data.frame)
 $ class                    : chr [1:8124] "p" "e" "e" "p" ...
 $ cap-shape                : chr [1:8124] "x" "x" "b" "x" ...
 $ cap-surface              : chr [1:8124] "s" "s" "s" "y" ...
 $ cap-color                : chr [1:8124] "n" "y" "w" "w" ...
 $ bruises                  : chr [1:8124] "t" "t" "t" "t" ...
 $ odor                     : chr [1:8124] "p" "a" "l" "p" ...
 $ gill-attachment          : chr [1:8124] "f" "f" "f" "f" ...
 $ gill-spacing             : chr [1:8124] "c" "c" "c" "c" ...
 $ gill-size                : chr [1:8124] "n" "b" "b" "n" ...
 $ gill-color               : chr [1:8124] "k" "k" "n" "n" ...
 $ stalk-shape              : chr [1:8124] "e" "e" "e" "e" ...
 $ stalk-root               : chr [1:8124] "e" "c" "c" "e" ...
 $ stalk-surface-above-ring: chr [1:8124] "s" "s" "s" "s" ...
 $ stalk-surface-below-ring: chr [1:8124] "s" "s" "s" "s" ...
 $ stalk-color-above-ring  : chr [1:8124] "w" "w" "w" "w" ...
 $ stalk-color-below-ring  : chr [1:8124] "w" "w" "w" "w" ...
 $ veil-type                : chr [1:8124] "p" "p" "p" "p" ...
 $ veil-color               : chr [1:8124] "w" "w" "w" "w" ...
 $ ring-number              : chr [1:8124] "o" "o" "o" "o" ...
 $ ring-type                : chr [1:8124] "p" "p" "p" "p" ...
 $ spore-print-color        : chr [1:8124] "k" "n" "n" "k" ...
 $ population               : chr [1:8124] "s" "n" "n" "s" ...
 $ habitat                  : chr [1:8124] "u" "g" "m" "u" ...
```

We can see that the dataset contains 8,124 observations of 23 different character variables. Next, we'll look for missing values. To do this, we will count the number of complete cases in the dataset and subtract that from the total number of rows in the data.

```
# number of rows with missing values
nrow(mushrooms) - sum(complete.cases(mushrooms))
```

```
[1] 0
```

Continuing with data cleanup, it appears the veil-type column only contains one value.

```
unique(mushrooms$`veil-type`)
```

```
[1] "p"
```

Once confirmed, we can remove the column.

```
mushrooms$veil.type <- NULL
```

Analysis

Next, we will examine the odor variable to look at the split between edible and poisonous mushrooms.

```
table(mushrooms$class,mushrooms$odor)
```

```
   a   c   f   l   m   n   p   s   y
e 400   0   0 400   0 3408   0   0   0
p   0 192 2160   0  36  120 256 576 576
```

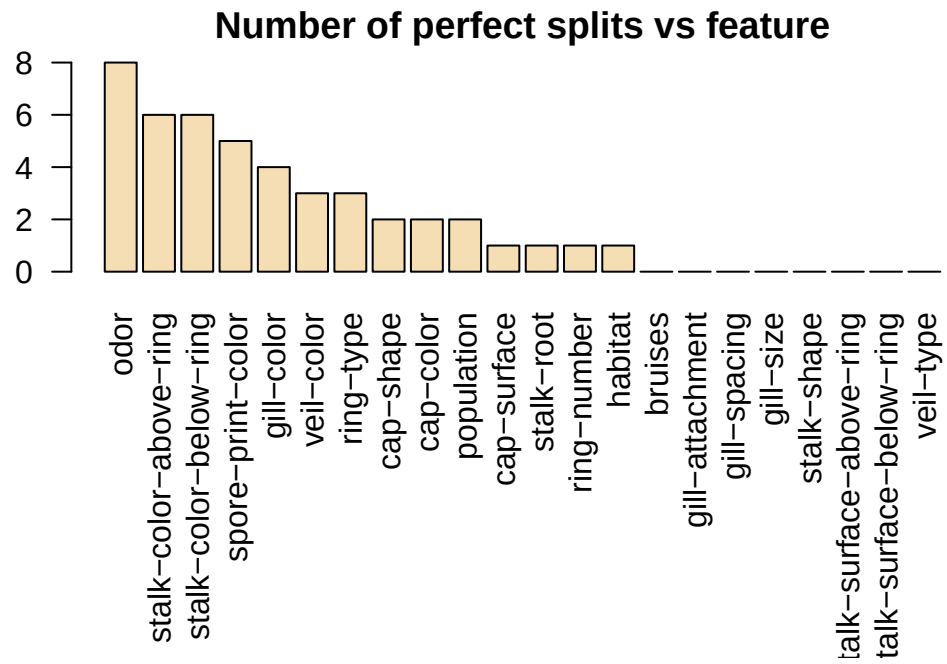
We will use the apply function to review all the remaining variables and their ability to perfectly split the dataset on edible and poisonous mushrooms. The MARGIN = 2 ensures the function is applied to all columns, aside from the first variable (class). This will count the number of 0 values created (indicating a perfect split).

```r
number.perfect.splits <- apply(X=mushrooms[-1], MARGIN = 2, FUN = function(col){
  t <- table(mushrooms$class,col)
  sum(t == 0)
})
```

After the function is run, we will order the variables by the generated value. Using this ordered set, we will create a bar chart to see the relative size of perfect splits among the variables.

```r
# Descending order of perfect splits
order <- order(number.perfect.splits,decreasing = TRUE)
number.perfect.splits <- number.perfect.splits[order]

# Plot graph
par(mar=c(10,2,2,2))
barplot(number.perfect.splits,
        main="Number of perfect splits vs feature",
        xlab="",ylab="Feature",las=2,col="wheat")
```



This shows that odor produced the highest number of perfect splits (8), with the stalk color variables tying for second (each at 6).

Building a Model

We will next split the dataset into test and training sets. We will use set.seed() to be able to reproduce the results. Next, we will randomly place 80% of the data into the training set, leaving the remaining 20% for testing.

```
#data splicing
set.seed(12345)
train <- sample(1:nrow(mushrooms),size = ceiling(0.80*nrow(mushrooms)),replace = FALSE)
# training set
mushrooms_train <- mushrooms[train,]
# test set
mushrooms_test <- mushrooms[-train,]
```

Next, we will create a penalty matrix. Since the potential damage of classifying a poisonous mushroom as edible is potentially much more dangerous that mistakenly calling an edible mushroom poisonous, we set the the penalty value 10 for the former and 1 for the latter miscalculation.

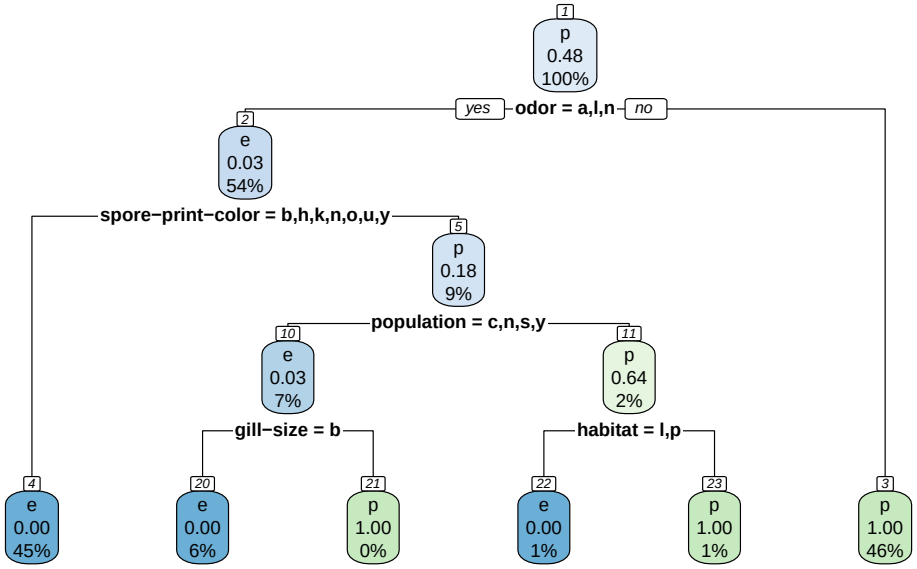
```
penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)
```

And then, using rpart, we'll build the classification tree. We will be using all variables (except class) as predictors in this model, using the training data set. Using loss = penalty.matrix ensures we're employing the values from above. Method = class ensures that we are using a classification tree rather than another model type.

```
tree <- rpart(class~.,
              data=mushrooms_train,
              parms = list(loss = penalty.matrix),
              method = "class")
```

And then visualize the tree using rpart.tree

```
rpart.plot(tree, nn=TRUE)
```



Next, we will use a pruning method to further optimize the model. First we use cp.optimum to find the optimal complexity parameter (cp), and selecting the smallest value using which.min. We then use the prune function t0remove the unnecessary branches (hence, pruning) to help avoid over fitting the model. This is especially important given the large ration in our penalty matrix (1:10).

```
# choosing the best complexity parameter "cp" to prune the tree
cp.optim <- tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
# tree prunning using the best complexity parameter. For more in
tree <- prune(tree, cp=cp.optim)
```

Testing and Evaluation

Finally we will test the accuracy of the model.

```
#Testing the model
pred <- predict(object=tree,mushrooms_test[-1],type="class")


#Calculating accuracy
t <- table(mushrooms_test$class,pred)
confusionMatrix(t)
```

```
Confusion Matrix and Statistics

   pred
      e   p
  e 829   0
  p   0 795

               Accuracy : 1
                 95% CI : (0.9977, 1)
    No Information Rate : 0.5105
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5105
         Detection Rate : 0.5105
   Detection Prevalence : 0.5105
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```

The results of the test are remarkably strong. The model accurately predicted 829 edible and 795 poisonous mushrooms with no false positives. The accuracy number is 100% with a 95% confidence interval of 0.9977, 1 - meaning we're 95% confident that the accuracy of model is between 99.77 and 100%. This is a very strong model that accurately predicted mushrooms.

Recommendations

The accuracy of the model suggests that it would be highly valuable to all participants in the mushroom trade - from growers and distributors to retailers and consumers. Food safety is a very serious concern, and the risks in legal liability and human health are high. Therefore, I would recommend expanding the database to include more observations from more location and including more varieties to ensure the results hold and that mushrooms are being safely evaluated as part of the food chain.

References

[1] Recursive Partitioning and Regression Trees [R package rpart version 4.1-15]. (2019, April 12). Cran.r-Project.org. https://cran.r-project.org/web/packages/rpart/index.html