Aly6040 Data Mining

Module 2

Technique Practice 2

Professor: - Prof Harpreet Sharma

Submitted by: - Harshil Bhatt

NEUID: - 002780601

Table of Contents

Introduction: -

This dataset contains 8124 entries that describe the edible and physical traits of gilled mushrooms from the families Agaricus and Lepiota. The classification exercise's goal is to decide whether or not the mushrooms can be eaten based on their physical qualities. The test and training sets were made using stratified random sampling, with the test set receiving 20% of the data and the training set receiving 80%.

Using the provided code and data collection, we developed the mining procedure to examine and decipher the findings for this task. suggested the qualities to consider before determining whether or not to eat a mushroom. Created analyses and variables to look at in order to create further classifications, like the utilisation of non-poisonous mushrooms. The error of misclassifying a poisonous mushroom as edible is ten times worse than the error of misclassifying a poisonous mushroom as edible, according to a loss matrix used to penalise misclassifications.

Code Walkthrough

Checking the structure of the dataset

```
```{r}
str(mushrooms)
```
```

```
'data.frame':   8124 obs. of  23 variables:
 $ class                   : chr  "p" "e" "e" "p" ...
 $ cap.shape               : chr  "x" "x" "b" "x" ...
 $ cap.surface             : chr  "s" "s" "s" "y" ...
 $ cap.color               : chr  "n" "y" "w" "w" ...
 $ bruises                 : chr  "t" "t" "t" "t" ...
 $ odor                    : chr  "p" "a" "l" "p" ...
 $ gill.attachment         : chr  "f" "f" "f" "f" ...
 $ gill.spacing            : chr  "c" "c" "c" "c" ...
 $ gill.size               : chr  "n" "b" "b" "n" ...
 $ gill.color              : chr  "k" "k" "n" "n" ...
 $ stalk.shape             : chr  "e" "e" "e" "e" ...
 $ stalk.root              : chr  "e" "c" "c" "e" ...
 $ stalk.surface.above.ring: chr  "s" "s" "s" "s" ...
 $ stalk.surface.below.ring: chr  "s" "s" "s" "s" ...
 $ stalk.color.above.ring  : chr  "w" "w" "w" "w" ...
 $ stalk.color.below.ring  : chr  "w" "w" "w" "w" ...
 $ veil.type               : chr  "p" "p" "p" "p" ...
 $ veil.color              : chr  "w" "w" "w" "w" ...
 $ ring.number             : chr  "o" "o" "o" "o" ...
 $ ring.type               : chr  "p" "p" "p" "p" ...
 $ spore.print.color       : chr  "k" "n" "n" "k" ...
 $ population              : chr  "s" "n" "n" "s" ...
 $ habitat                 : chr  "u" "g" "m" "u" ...
```

Fig 1.1 Structure

The output shows the first few values of each column as well as all the fields in the mushrooms data frame. Additionally, it demonstrates that the dataset has 23 columns and 8124 rows. All of the loaded columns are of the chr datatype. The 'class' attribute of the mushroom specifies whether it is edible (e) or poisonous (p).

Checking for the rows with missing values

```
```{r}
nrow(mushrooms) - sum(complete.cases(mushrooms))
```

[1] 0
```

Fig 1.2 Looking for missing values

The algorithm then checks to see if there are any rows in the dataset with no values before moving on to any further analysis. A boolean result of TRUE or FALSE is returned by the 'complete.cases' function depending on whether every value in a data frame row is missing. The mushrooms data frame's number of rows is provided by the nrow function. The number of rows with missing values is obtained by deducting the total number of rows and the sum of all full rows, or rows with no missing data. This dataset has no rows with any missing values.

Cleaning up data

```
```{r}
mushrooms$veil.type <- NULL
```
```

Fig 1.3 Data clean up

The goal of the command given here is to disable the feature on the toolbar. There is a feature in the mushrooms dataset called veil-type, not veil.type. The script that is provided has code that does nothing.

I wanted to confirm the distinct value of the variable "veil-type" in the dataset because the code comment noted that it is a redundant variable. The screenshot demonstrates that the variable only has a singular value 'p' in each row. The appropriate variable name was utilised to remove the redundant variable from the dataset.

Assess the effect of feature value on the mushroom class.

The code computes the number of different values each feature has and counts the clean splits of each feature value in order to analyse the impact of each feature in determining the mushroom class.

The code first verifies the 'odour' feature's logic before moving on to analyse all of the columns. As you can see from the findings, the class of mushroom is always e (edible) when the odour is 'a' and 'l', and the class is always p (poisonous) when the odour is c, f, m, p, s, or y.

```{r}

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

	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

Fig 1.4 feature value on class

Only mushrooms with odour n are divided into two classes and do not have a clear separation.

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

```
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")
```

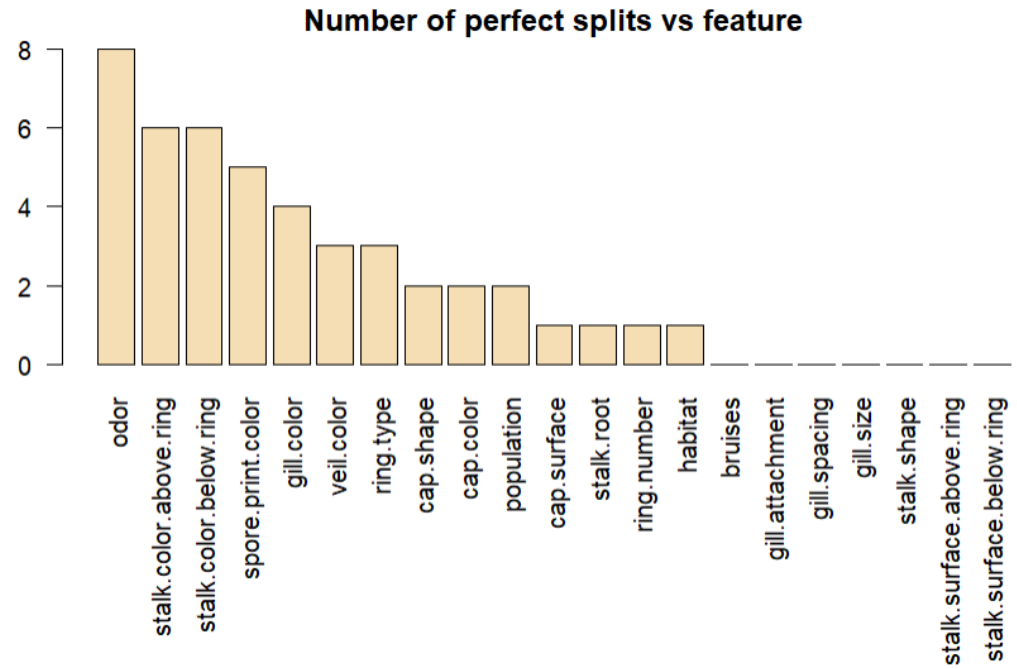


Fig 1.5 Splits vs feature

The code keeps counting the number of values in the features that assist identify the mushroom class as it continues to analyse each column. The features with the highest clean splits in identifying the mushroom class are arranged in descending order after all the features have been analysed and recorded in a table. The table results are displayed as a bar graph, and it can be seen that the top three columns are used to determine whether "Odour," "stalk-color-above-ring," and "stalk-color-below-ring" indicate whether a mushroom is edible. This aids in identifying the characteristics that are most likely to help identify the 'class' of mushrooms.

Split dataset into training and testing sets

```
```{r}
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,]
```
```

Fig 1.6 splitting the data

The dataset must first be divided into training and testing data frames before a model can be built to use it for both training and testing. A list of values can be spliced together using the example command. The values listed above are the dataset's row indexes, as you can see. Two fresh tibbles are made for training and testing once the command successfully identifies 80% of the indexes. The set.seed(12345) function makes sure that randomization produces the same outcomes every time. Making sure that the same value is not chosen from the list of indexes again is ensured by the replace=FALSE option.

Recursive Partitioning Tree is used to create a decision tree.

The algorithm then uses the training dataset to build the recursive partitioning tree model after creating the training and testing datasets. The 'rpart' (recursive partitioning) function is used to construct the classification tree. The function takes four arguments. The data frame, the prediction function, the parameters defining the loss matrix to be used with the decision tree, and the method to be employed are the first four inputs regarding the decision tree. The code uses a categorization approach for the free decision. The decision tree will be plotted and the node numbers will be displayed using the rpart.plot function with nn=TRUE.

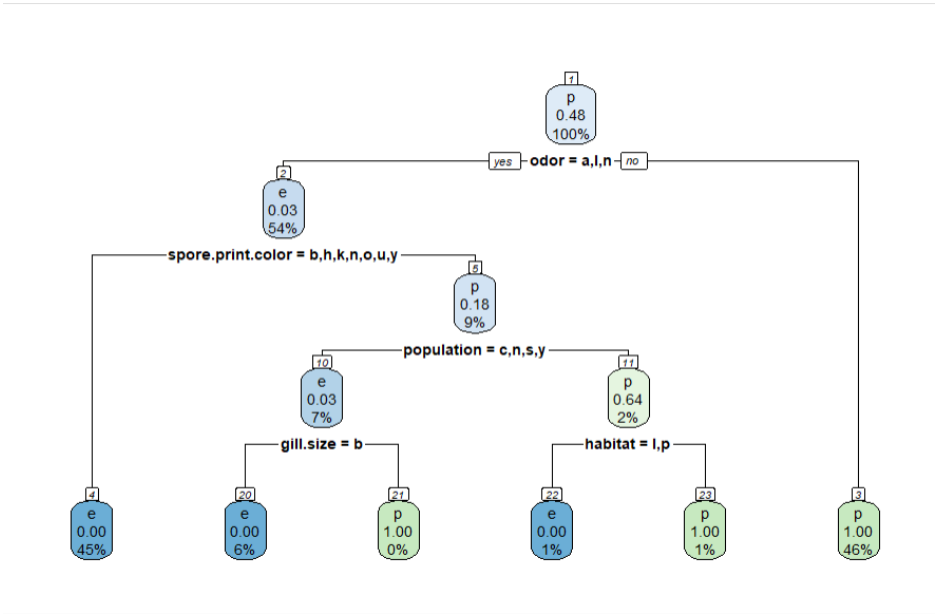


Fig 1.7 Decision tree

Prune tree

```
```{r}
# 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)
```
```

Fig 1.8 Initial steps before prune tree

To prevent data from being overfit, a decision tree must be pruned. The smallest decision trees with the lowest cross-validation error, as indicated by the 'xerror,' are the best. The pruned tree is not plotted once more in the provided piece of code, though. However, the next prediction step uses the pruned tree.

Using a trained decision tree, forecast

The test dataset (mushroom_test), which was spliced, is utilized to obtain the expected class values using the predict function. In the earlier steps. The test dataset, type as 'class' (classification), and recursive partitioning 'tree' model are used by the predict function to make the prediction.

Summary of results

```
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

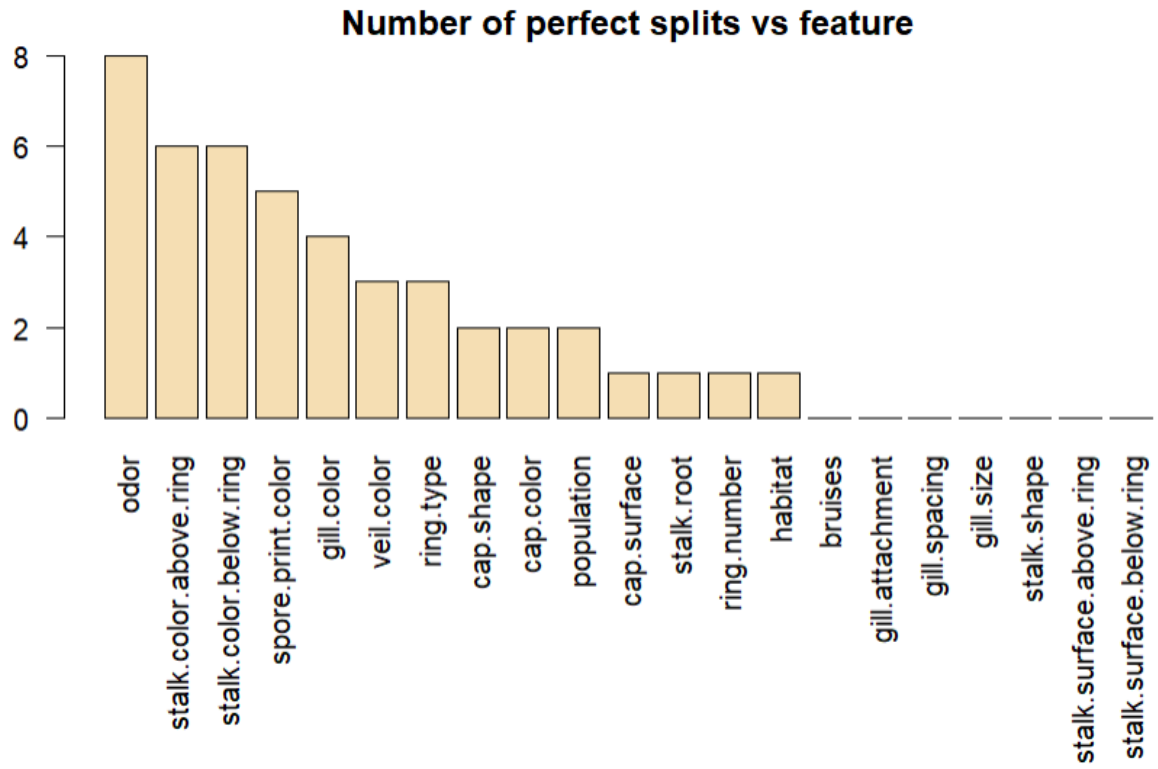
       'Positive' Class : e
```

Fig 1.9 Confusion matrix

The confustionMatrix function is used to calculate accuracy as the last step.

The predicted (pred) values and the values from the mushroom_test$class are used to form a table that is then passed as input to the function. Only True Positives and True Negatives are displayed in the confusion matrix output, indicating that every row in the test dataset was correctly identified. The 'Accuracy' is 100% since there are no False Positives or False Negatives in this analysis. 829 True Negatives and 795 True Positives are present. Due to the decision tree's properties, which aid in determining the model's accuracy

Dataset Analysis and insights



	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

The elements that most likely will help evaluate whether a mushroom is safe to eat are detailed in the bar chart that was created. The most categories under the 'odor' aspect can assist identify the 'class' of the mushroom. The decision tree presented here supports the understanding gained from the clean split analysis. We may learn more about the clean split by taking a closer look at the table that was generated. The table unequivocally demonstrates that the deadly odor values are "c," "f," "m," "p," "s," and "y." The feature 'odor' is the best feature for dividing the data into classification categories and is also the root element of the classification decision tree shown below.

Features to use to find safe mushrooms

The decision tree visualization enables us to pinpoint the key qualities that influence whether a mushroom is suitable for consumption. 'Odor' is this classification tree's primary attribute. Any smell that isn't "a," "l," or "n" is poisonous. 52% of the training dataset is poisonous, whereas 48% of it is edible. "Spore-print-color" is the next attribute of the mushroom used in the decision tree. When the 'spore-print-color' is 'b', 'h', 'k', 'n', 'o', 'u', or

'y', the mushroom is edible. The 'population' feature can be used to assist assess whether the mushroom is safe to eat for the remaining set of spore-print-color.

The following are the primary traits that can be used to distinguish between edible mushrooms:

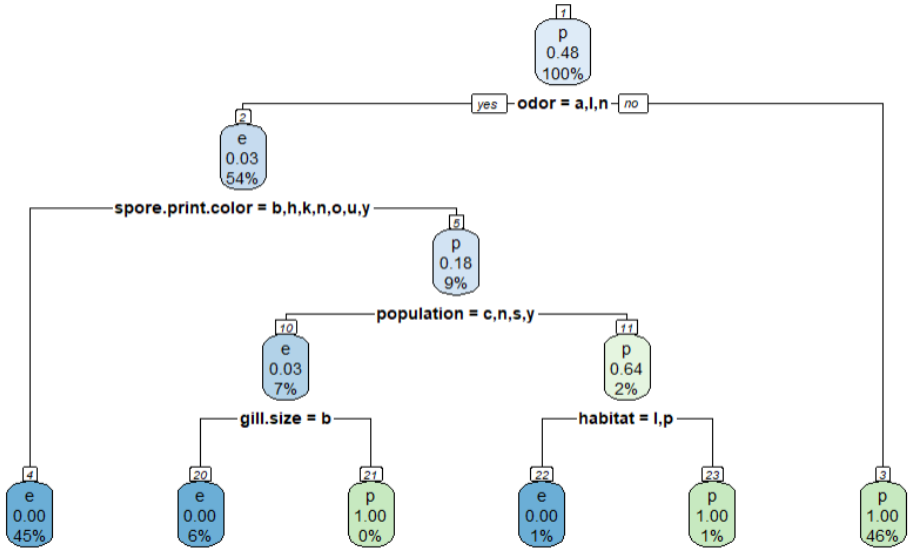
1. odor

2. spore-print-color

3. population

4. gill-size

5. habitat



The 'fancyRpartPlot' programme from the 'RColorBrewer' package was used to create the decision tree shown above.

Only the 'odor - n' did not have a clean split, as can be shown from an earlier investigation of clean splits. 'A' and 'L' had distinct, delicious odors. However, we do not observe the 'odour' attribute being employed for categorization decision-making at the second level. Instead of explicitly employing the odor once more at the second level, the decision tree in this case uses alternative features to get to the final leave node. Although this observation has no bearing on the model's correctness, it would appear to be a simpler decision tree if the 'odor' feature could be employed once more in the intermediate node at the second level. But it also might have led to an additional level, and the mushrooms with "odour - n" would still have to travel along this road.

Interpretations and Recommendations

'Odor' is the most crucial variable in the dataset that determines whether a mushroom is safe to eat, according to the data mining and analysis that was done. Almond and anise-scented mushrooms are edible. The owner can discard such mushrooms because they have no other odors other than pure poison. Spore-print-color can be used to assess whether mushrooms are safe to consume if they don't smell. If the spore-print color is black, brown, buff, chocolate, orange, purple, or yellow, the mushroom is safe to consume. Additional traits must be verified if the owner observes that the spore color is white or yellow.

Conclusion

With the exception of a redundant variable that was eliminated before the model was created, the mushroom dataset is largely clean. The dataset is then split into a training and testing set that is 80/20. The Decision Tree model is utilized after the tree has been felled using the best cp. The model is then tested using the testing set. A confusion matrix is used to gauge performance. The 95% CI for the model's accuracy is [99.77%, 100%]. Furthermore, it is excellent at making precise positive and negative predictions. Since neither a toxic nor an edible mushroom is projected to be edible, there is neither a false positive nor false negative. By looking at the bar plot of the analysis, one may also pick a few traits to swiftly assess a mushroom.

References

Le, J. (2018, June 19). *R decision trees tutorial: Examples & code in R for Regression & Classification*. DataCamp. Retrieved April 25, 2023, from https://www.datacamp.com/tutorial/decision-trees-R

Datawrangler. (2017, February 7). *Cost sensitive decision tree*. Kaggle. Retrieved April 25, 2023, from https://www.kaggle.com/code/datawrangler/cost-sensitive-decision-tree