A Decision Tree Model for Mushroom Edibility Prediction

1. Introduction:

Data mining is the process of extracting the information and identifying the patterns from large dataset to draw insights by implementing various statistical and machine learning methods. The steps involve data pre-processing which includes exploratory data analysis, data cleaning, data transformation and followed by data modelling and classification and then finally pattern evaluation and knowledge representation. Decision tree is one of the widely used modelling approach for predicting the value of a target variable based on several input variables.

We will analyse the mushroom data and build a decision tree model to predict whether the mushroom is edible or poisonous based on it's specifications like odour, cap shape, cap colour, gill colour, etc,.

The "Mushrooms" dataset contains 8,124 records and uses 23 attributes to describe the mushrooms. Each dataset contains additional attribute to flag if the mushroom is edible or not. I used 80% of the available data for train the model. The remaining 20% will be used as test dataset for evaluating the developed model. The goal is to predict whether a given mushroom is edible or poisonous to humans. Each observation has 23 features including the mushroom class.

2. Methods & analysis

2.1. Libraries used

We use the Rpart library in R for implementing recursive partitioning for classification, regression and survival trees. The caret package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems. The package contains tools for data splitting, pre-processing, feature selection, model tuning using resampling and variable importance estimation.
The rpart.plot package plots rpart trees.

The R Analytic Tool To Learn Easily (Rattle) provides a collection of utilities functions for quickly load data from a CSV file (or via ODBC), transform and explore the data, build and evaluate models, and export models as PMML (predictive modelling markup language) or as scores.

2.2. Loading the dataset

At first, we need to prepare the data set for the analysis. The mushrooms dataset contains 8,124 records about different mushrooms and 23 columns describing its features/characteristics. The dataset "mushrooms. xlsx" file is loaded into the variable *mushrooms* using the function read_excel().

3. Exploratory Data Analysis

3.1. Structure of the dataset

```
> 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" ...
```

Figure 1: Represents the structure of the mushroom dataset

From fig.1, we can understand that all the 23 variables are of character datatype. Since, *class* is our target variable, we can use 22 features as an input or predictor variable for modelling the decision tree.

3.2 Feature analysis

To investigate the distribution of the input features corresponding to the classification of the target variable, I have used the code reference from the kaggle.com(Mushroom classification with decision tree, June 7, 2022) to plot this distribution for each feature.

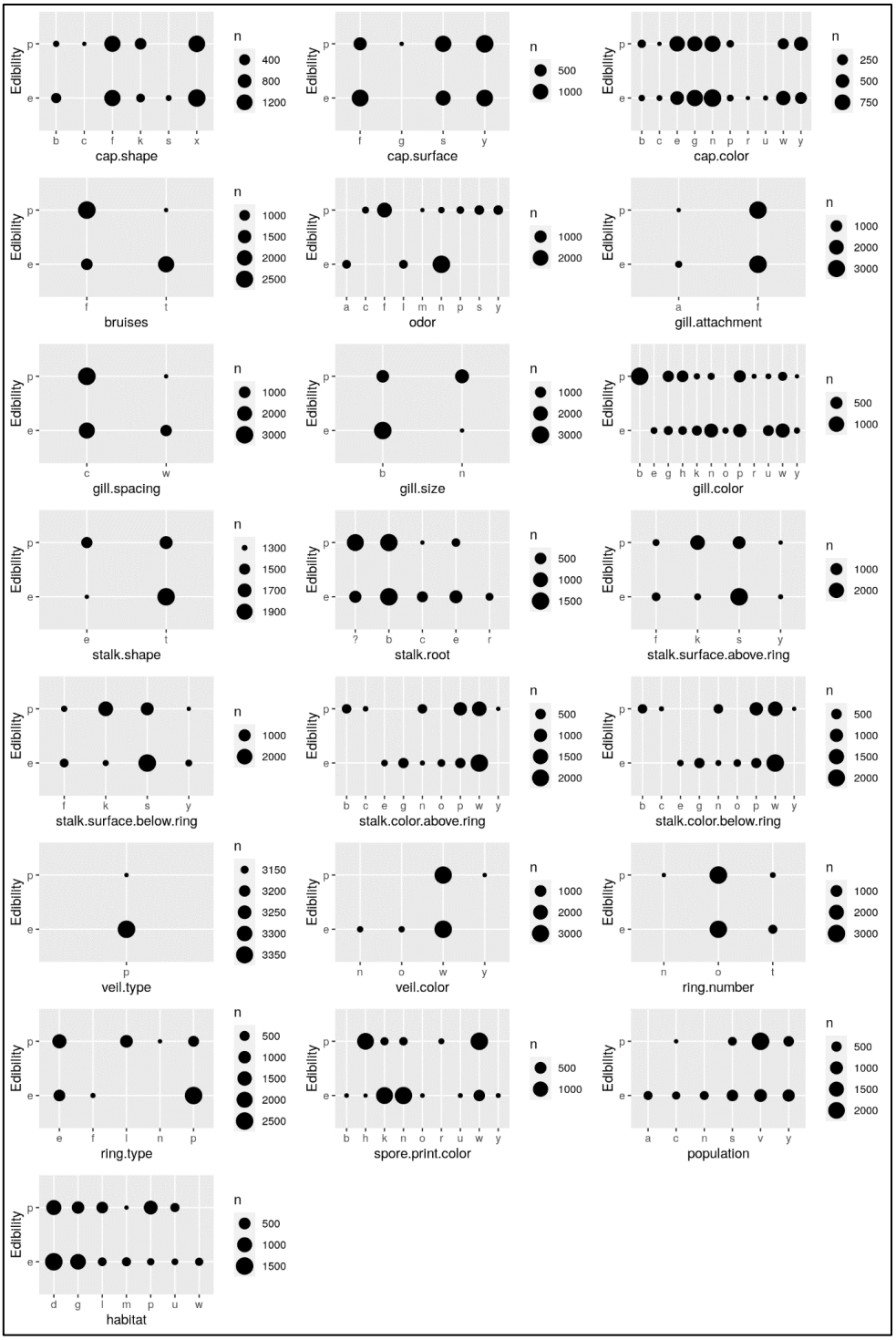

Figure 2: Represents the frequency distribution of each of the 22 predictor features values among the class variable – edible and poisonous.

The variable *veil.type* has only one level and doesn't contribute for classification, so it can be removed from the analysis. We nullify the column *mushrooms$veil.type* using by assigning the NULL values to it.

But, for the variable *odor*, if the factor value is either almond(a) or anise(l), we can classify the mushroom as poisonous. And, if the factor value is any of creosote(c), fishy(y), foul(f), musty(m), pungent(p), spicy(s), we can classify the mushroom as edible. We cannot classify the mushroom, if the odor value is none(n).

3.3 Finding the missing values in the dataset

The data that has been collected may contain null values, missing values, unwanted data that may lead to wrong interpretations. Hence data pre-processing is an essential step where in the irrelevant data can be eliminated so there will be a better accuracy in the final results.

```
> nrow(mushrooms) - sum(complete.cases(mushrooms))
[1] 0
```

Figure 3: The output shows there is no missing values in the mushroom dataset.

We used the function *complete.cases()* which return a logical vector indicating the rows with no missing values.

Since, the sum of the rows with no missing values and the number of rows in the mushrooms dataset are same, we can infer that there are no rows with missing values in the dataset.

3.4. Steps to measure the variable importance

In order to understand the importance of the 22 different variables in identifying the target variable (*edible* or *poisonous*), we follow the following steps:

1. Create a table for each variable versus class type (*edible*, *poisonous*)
2. Report the number of perfect splits (where there is a zero) in each column in the table created in step 1
3. Reorder the list created in step 2 by the number of perfect splits reported
4. Plot the sorted list from step 3

I have explained the above procedure by creating a table using target variable c*lass* against the variable *odor*

```
> 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
```

Figure 4: The output is a contingency table of the counts at each combination of factor levels.

The fig.4 table will help us understand whether that particular predictor feature is significant for classifying the target variable. We can quickly observe that mushrooms with *odor* value "c","f","m","p","s" and "y" are poisonous. While, all mushrooms with odor value "a", "l" are edible in this dataset. We can use this information to develop a classification model for the mushrooms.

we will use the number of perfect splits as a proxy for scoring the input features in splitting the data into class - edible and poisonous

In the R code, we have removed the class column from the mushroom dataset as it's our target variable. We have used the col() which returns column number or column label to create table for each of the predictor variable with the target variable *class* using the apply(). MARGIN = 2 indicates that the function is applied over column. So, it creates a table with one column variable against the class variable and its number of perfect splits is calculated and stored in the variable t. The apply() repeats the same for rest of the column variable and it is stored in number.perfect.splits.

```
> number.perfect.splits
               cap-shape              cap-surface                cap-color
                       2                        1                        2
                 bruises                    odor          gill-attachment
                       0                        8                        0
             gill-spacing               gill-size               gill-color
                       0                        0                        4
              stalk-shape              stalk-root stalk-surface-above-ring
                       0                        1                        0
stalk-surface-below-ring   stalk-color-above-ring   stalk-color-below-ring
                       0                        6                        6
               veil-type              veil-color             ring-number
                       0                        3                        1
               ring-type        spore-print-color               population
                       3                        5                        2
                 habitat
                       1
```

Figure 5: The output shows the number of perfect splits for each input feature

The fig.5 output shows the number of perfect splits for the 22 predictor variable against the target variable *class*. We can observe that variable *odor* has the highest perfect split of 8 followed by stalk ring colour and spore print colour. These variables can be the best predictors for the decision tree. We can also observe that 8 variables don't have perfect splits which are not the right candidate for splitting.

We need to order the feature variable based on the number of splits to be considered for modelling the decision tree. We have used the order() to arrange the feature variable in descending order.

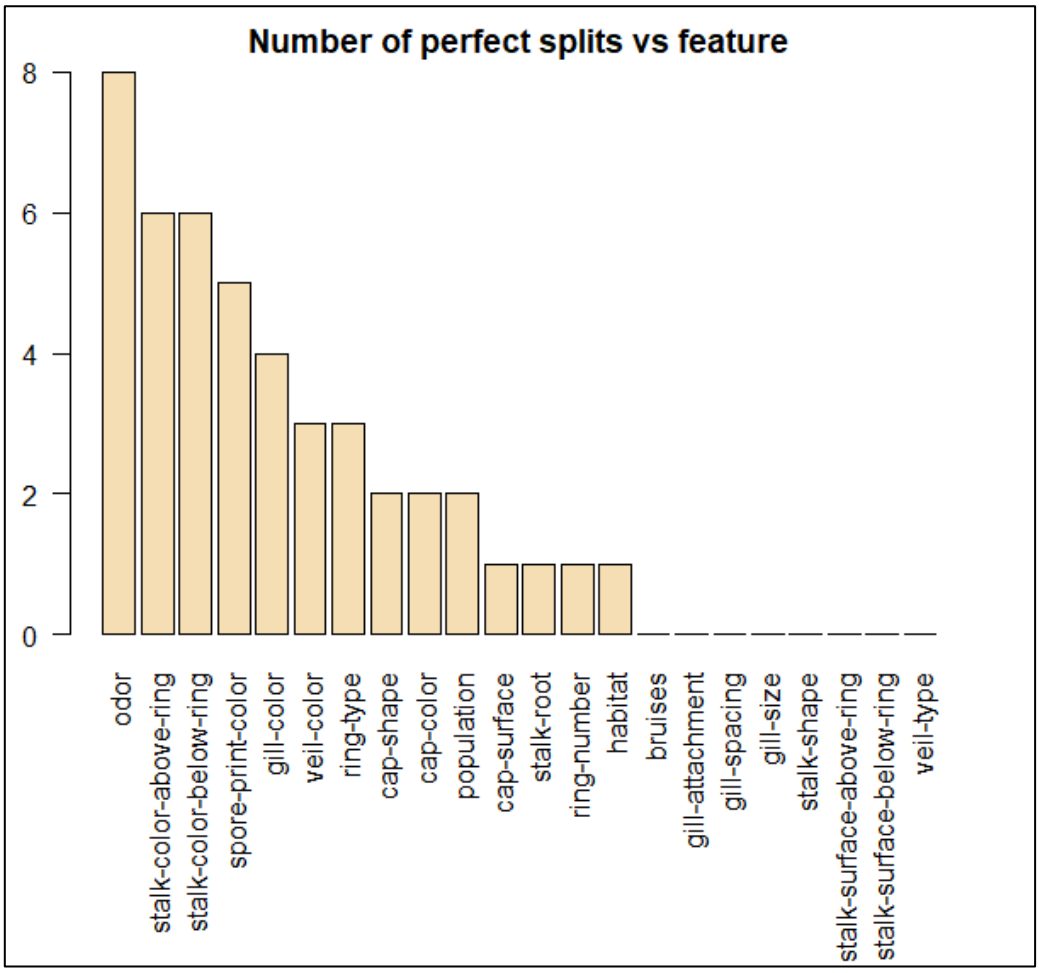

Figure 7: The output shows the feature variable in descending order by number of perfect From fig.7, we see that the feature *odor* can highly classify the target variable mushroom class. It should be noted that the order of the features shown in the fig.7 is not necessarily the order in which the final model will choose features to build up the tree.

4. Data Splicing

Data Splicing is the process of splitting the data into a training set and a testing set. We will split the data using two random samples without replacement:

- training dataset (80%) for model building
- test dataset (20%) to validate the efficiency of the model.

We have used the set.seed() fuction with the argument 12345 to to get the reproducible random numbers. We have used the ceiling() function to return the nearest higher value of the given input (i.e.,. 0.80*nrow(mushrooms) and given the value FALSE for the argument replace for choosing random samples without replacement. The random numbers generated is passed into the mushroom dataset and stored in the variable *mushrooms_train.*

```
> mushrooms_train
# A tibble: 6,500 x 23
   class `cap-shape` `cap-surface` `cap-color` bruises odor  `gill-attachment`
   <chr> <chr>       <chr>         <chr>       <chr>   <chr> <chr>
 1 p     f           y             n           f       s     f
 2 e     x           y             n           t       a     f
 3 e     s           f             n           f       n     f
 4 e     x           y             w           t       l     f
 5 p     f           y             y           f       f     f
 6 e     x           y             e           t       n     f
 7 p     x           s             w           f       c     f
 8 e     b           y             w           t       l     f
 9 p     x           s             e           f       f     f
10 p     k           y             n           f       s     f
# ... with 6,490 more rows, and 16 more variables: gill-spacing <chr>,
#   gill-size <chr>, gill-color <chr>, stalk-shape <chr>, stalk-root <chr>,
#   stalk-surface-above-ring <chr>, stalk-surface-below-ring <chr>,
#   stalk-color-above-ring <chr>, stalk-color-below-ring <chr>, veil-type <chr>,
#   veil-color <chr>, ring-number <chr>, ring-type <chr>, spore-print-color <chr>,
#   population <chr>, habitat <chr>
```

Figure 8: The output shows the dimension and the glimpse of the variable *mushrooms_train*

The training dataset used for model building has 6,500 records(80% of all data) out of the 8,124 records about mushrooms. All the features are categorical with different possible values.

The numbers that are not part of the *train* variable are passed into the mushroom dataset and stored in the variable *mushrooms_test.*.

```
> mushrooms_test
# A tibble: 1,624 x 23
   class `cap-shape` `cap-surface` `cap-color` bruises odor  `gill-attachment`
   <chr> <chr>       <chr>         <chr>       <chr>   <chr> <chr>
 1 e     b           s             w           t       a     f
 2 e     b           y             w           t       l     f
 3 e     x           y             y           t       a     f
 4 e     b           s             y           t       a     f
 5 e     s           f             g           f       n     f
 6 e     f           f             w           f       n     f
 7 p     x           s             n           t       p     f
 8 e     b           s             y           t       l     f
 9 e     s           f             g           f       n     f
10 e     x           f             y           t       a     f
# ... with 1,614 more rows, and 16 more variables: gill-spacing <chr>,
#   gill-size <chr>, gill-color <chr>, stalk-shape <chr>, stalk-root <chr>,
#   stalk-surface-above-ring <chr>, stalk-surface-below-ring <chr>,
#   stalk-color-above-ring <chr>, stalk-color-below-ring <chr>, veil-type <chr>,
#   veil-color <chr>, ring-number <chr>, ring-type <chr>, spore-print-color <chr>,
#   population <chr>, habitat <chr>
```

Figure 9: The output shows the dimension and the glimpse of the variable *mushrooms_test*

The testing dataset used for validating the efficiency of the model has 1,624 mushrooms data (20% of all data).

5. Classification Tree

The classification tree is a structural mapping of binary decisions that lead to a decision about the class of the mushroom. We use the training dataset to build the model for decision tree. In order to ensure that we minimize the number of poisonous mushrooms classified as edible we will give a penalty 10 times bigger than the penalty for classifying an edible mushroom as poisonous.

```
> penalty.matrix
     [,1] [,2]
[1,]    0    1
[2,]   10    0
```

Figure 10: The output shows the values in the penalty matrix

We use rpart (Recursive Partitioning And Regression Trees) algorithm for building the Decision Tree. For the argument *formula,* we have given the input as a dataframe of target variable and the input features. Since it's a classification splitting, we use class for the argument method and penalty matrix as loss matrix for the argument parameter.

```
> tree
n= 6500

node), split, n, loss, yval, (yprob)
      * denotes terminal node
```

Figure 11: The output shows the values in the penalty matrix

The tree shows the split of the training dataset(6500 data points) based on the loss matrix.

The rpart.plot() was used for visualizing the decision tree. The argument nn=TRUE displays the node indices, i.e. the row numbers of the nodes in the object's frame.

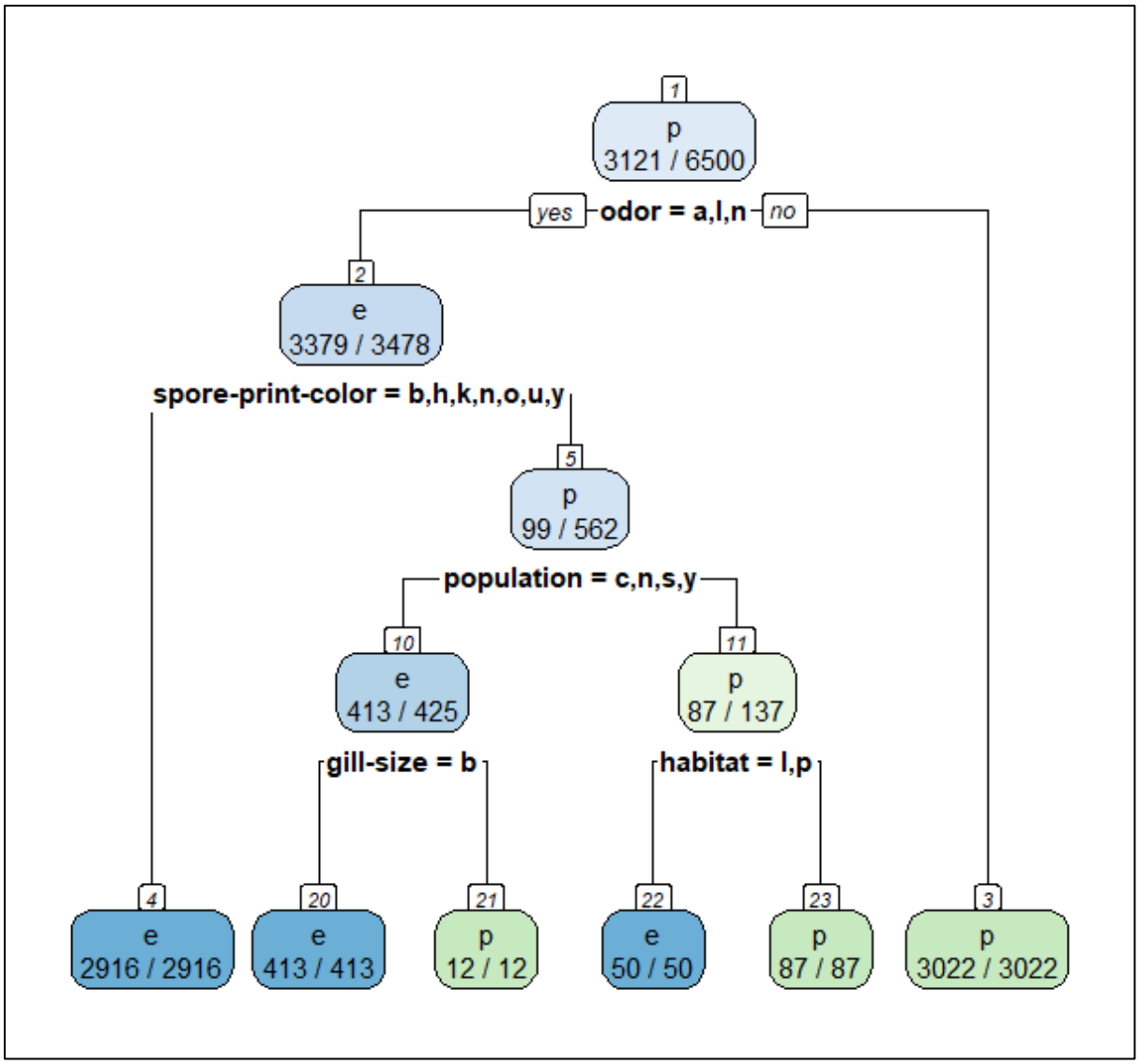

Figure 12: The output shows the values in the penalty matrix

In fig.12, the root node has 6500 training data in which 3121 records are poisonous. It splits 46% of the data as poisonous(3022 records) when the *odor* levels are not "*almond*", "*anise*", "*none*". The remaining 54% of the data contains 3379 records of edible mushroom out of the total mushroom 3478. It contains roughly 3% of the poisonous mushroom records. The second split 45%(2916 records) of the data as edible when the levels of the *spore print colour* is any of the following "*buff*", "*chocolate*", "*black*", "*brown*", "*orange*". "*purple*", "*yellow*". The 9% remaining data contains 99 poisonous mushroom record out of the total 562 record. The third split is when population level is "*clustered*", "*numerous*", "*scattered*", "*solitary*", it splits 7% of the data in which the edible mushroom proportion is 413 out of 425. Since it's not a perfect split, it undergoes further split with the variable *gill size* when its level is equal to "*broad".* The resulting 413 record(6% of the data) is edible mushroom and the remaining 12 records are poisonous. Similarly, when the *population* level is not "*clustered*", "*numerous*", "*scattered*", "*solitary*", it splits 2% of the data in which the poisonous mushroom proportion is 87 out of 137. Since it's not a perfect split, it undergoes further split with the variable *habitat* when its level is either "*leaves" or "paths".* It splits as 50 records of edible mushroom and 87 records of poisonous mushroom.

5.1. Pruning the tree

Pruning is a process to reduce the size of the decison tree by removing the non-critical and redundant nodes. To avoid overfitting, the cptable element of the result generated by rpart can be examined. The cptable provides a brief summary of the overall fit of the model.

```
> tree$cptable
          CP nsplit  rel error       xerror         xstd
1 0.70701391      0 1.00000000 1.000000e+01 0.1192054496
2 0.15596330      1 0.29298609 2.929861e-02 0.0029221105
3 0.08671204      2 0.13702279 1.131400e+00 0.0560459358
4 0.03551347      3 0.05031074 1.535957e-01 0.0210674150
5 0.01479728      4 0.01479728 1.515241e-01 0.0210551689
6 0.01000000      5 0.00000000 5.918911e-04 0.0004184658
```

Figure 13: The output shows the values of the cptable

The "CP" column lists the values of the complexity parameter, the number of splits is listed under"nsplit", and the column "xerror" contains cross-validated classification error rates; the standard deviation of the cross-validation error rates are in the "xstd" column. Normally, we select a tree size that minimizes the cross-validated error. (Online.stat.psu.edu, 2022)

Choosing the best complexity parameter "cp" to prune the tree

```
> cp.optim
[1] 0.01
```

cp.optim stores the optimal complexity parameter. To prune a tree with the best complexity parameter, the function prune() is performed, which takes the full tree as the first argument and the optimised complexity parameter as the second.

Since the cross-validated error is minimum when the nsplit is 5, the decision tree doesn't require any pruning.

5.2. Predictions on test set

Finally, we proceed to test the model on the test dataset.

We use the predict() function to validate the testing dataset with the decision tree model. The tree is given as an input for the argument object. mushrooms_test data with the class column removed is given as a new data. Since it's a classification model, we have given *class* for the argument *type*. The 1,624 testing data was predicted as either edible or poisonous by using the decision tree model we built using the training data.

To check the accuracy of the prediction and to validate the decision tree model, we used table() function that performs categorical tabulation of data with the variable and its frequency. The data provided is actual class of the testing data with the predicted class of the testing data. The table output is given as an input for the function confusionMatrix() that will categorize the predictions against the actual values. It a two dimensions matrix with one indicating the predicted values and another representing the actual values.

```
> 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
```

Figure 14: The output shows the values of the confusion matrix

The fig.14, shows the edible and poisonous mushrooms are predicted with 100% accuracy without any True Negative and False Positive.

Therefore, we obtained:

- 100% accuracy on test set
- 95% confidence interval (0.9977, 1) for the model accuracy
- The sensitivity and specificity of the model is 1.0

RECOMMENDATION OF ACTION:

From the decision tree, we can understand that the following variables have a strong association with the edibility of the mushroom.

Odor, Spore print color, Population, Gill size and Habitat.

For poisonous mushroom:

The general recommendation is to avoid mushroom that has following odor: "*creosote", "foul", "musty", "pungent", "spicy" and "fishy".*

If the mushroom odor is none, then we have to avoid if their spore print colour is any of the following: "*buff*", "*chocolate*", "*black*", "*brown*", "*orange*". "*purple*", "*yellow*".

If the spore print colour is any of the above, we have to look whether its population characteristics is any of the following: "*clustered*", "*numerous*", "*scattered*", "*solitary*". If yes, we should avoid eating mushrooms if its gill size is "*broad".* And if the population characteristics is none of the above, we should avoid mushrooms that has either of the habitat: "*leaves" or "paths".*

For edible mushroom:

The general recommendation is to eat mushroom that has either "*almond*" or "*anise*" odor.

If the mushroom odor is none, we can only eat if their spore print colour is either "*green*" or "*white*".

If the spore print colour of the mushroom is any of the following: "*buff*", "*chocolate*", "*black*", "*brown*", "*orange*". "*purple*", "*yellow*", we have to look whether its population characteristics is any of the following: "*clustered*", "*numerous*", "*scattered*", "*solitary*". If yes, we can eat mushrooms if its gill size is "*broad".* And if the population characteristics is none of the above, we can eat mushrooms with the following habitat: "*grasses", "paths", "meadows", "urban", "waste", "woods".*

CONCLUSION

The mushrooms data set is a special data set, because it contains only categorical features. The decision tree algorithm works perfectly for the mushroom dataset as the model we built is able to classify the mushroom as either poisonous or edible with 100% accuracy. We understood that choosing the right penalty matrix is critical for improving the accuracy pf the decision tree model.

REFERENCES

1. Mushroom classification with decision tree. (n.d.). Kaggle.com. Retrieved June 7, 2022, from https://www.kaggle.com/code/geland/mushroom-classification-with-decision-tree/report
2. 11.9 - R Scripts. (n.d.). Online.stat.psu.edu. Retrieved June 8, 2022, from https://online.stat.psu.edu/stat508/book/export/html/728