MATH5743M: Statistical Learning: Assessed Practical 3 - Mushroom

Anupam Bose, 201570198, School of Mathematics

Semester 2 2022

```
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.1.2
```

```
## Linking to GEOS 3.9.1, GDAL 3.2.1, PROJ 7.2.1; sf_use_s2() is TRUE
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.2
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
## v purrr   0.3.4
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'tidyr' was built under R version 4.1.2
```

```
## Warning: package 'readr' was built under R version 4.1.2
```

```
## Warning: package 'purrr' was built under R version 4.1.2
```

```
## Warning: package 'dplyr' was built under R version 4.1.2

## Warning: package 'stringr' was built under R version 4.1.2

## Warning: package 'forcats' was built under R version 4.1.2

## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.1.3
```

The data has been imported into a dataframe (df) by using the **read.csv** function.

```
df <- read_csv("mushrooms.csv", col_names = TRUE)
```

```
## Rows: 8124 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (6): Edible, CapShape, CapSurface, CapColor, Odor, Height

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can get some idea about the features of our dataset by glimpse() function

```
glimpse(df)
```

```
## Rows: 8,124
## Columns: 6
## $ Edible     <chr> "Poisonous", "Edible", "Edible", "Poisonous", "Edible", "Ed~
## $ CapShape   <chr> "Convex", "Convex", "Bell", "Convex", "Convex", "Convex", "~
## $ CapSurface <chr> "Smooth", "Smooth", "Smooth", "Scaly", "Smooth", "Scaly", "~
## $ CapColor   <chr> "Brown", "Yellow", "White", "White", "Gray", "Yellow", "Whi~
## $ Odor       <chr> "Pungent", "Almond", "Anise", "Pungent", "None", "Almond", ~
## $ Height     <chr> "Tall", "Short", "Tall", "Short", "Short", "Short", "Short"~
```

We have total 8,124 values in our dataset with 6 features. Edible column is our target variable which tells us if the mushroom is edible or poisonous.

As all the variables are categorical we must convert them to factors for the analysis.

```
##Making each variable as a factor
df <-df %>%
  map_df(function(.x) as.factor(.x))
```

We can now see that the variables have changed to factors:

```
glimpse(df)
```

```
## Rows: 8,124
## Columns: 6
## $ Edible     <fct> Poisonous, Edible, Edible, Poisonous, Edible, Edible, Edibl~
## $ CapShape   <fct> Convex, Convex, Bell, Convex, Convex, Convex, Bell, Bell, C~
## $ CapSurface <fct> Smooth, Smooth, Smooth, Scaly, Smooth, Scaly, Smooth, Scaly~
## $ CapColor   <fct> Brown, Yellow, White, White, Gray, Yellow, White, White, Wh~
## $ Odor       <fct> Pungent, Almond, Anise, Pungent, None, Almond, Almond, Anis~
## $ Height     <fct> Tall, Short, Tall, Short, Short, Short, Short, Tall, Tall, ~
```

We can now see the number of levels in our each variable present in our dataset.

```
number_class <- function(x){
  x <- length(levels(x))
}

#levels of each column
x <- df %>% map_dbl(function(.x) number_class(.x)) %>% as_tibble() %>%
  rownames_to_column() %>% arrange(desc(value))
colnames(x) <- c("Variable name", "Number of levels")

x <-x %>%
  mutate(Variable_Name=colnames(df[as.numeric(x$`Variable name`)])) %>%
  select(Variable_Name,`Number of levels`)
x
```

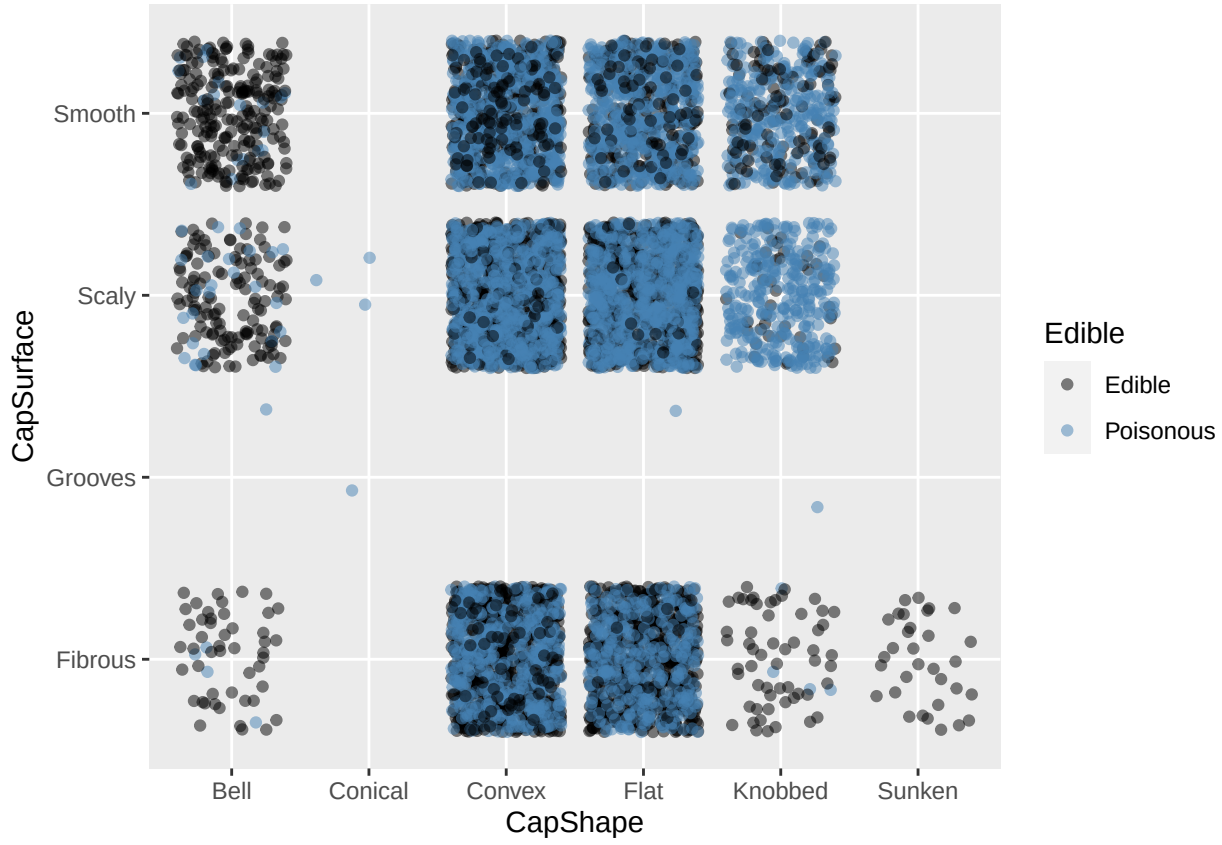
```
## # A tibble: 6 x 2
##   Variable_Name `Number of levels`
##   <chr>                      <dbl>
## 1 CapColor                      10
## 2 Odor                           9
## 3 CapShape                       6
## 4 CapSurface                     4
## 5 Edible                         2
## 6 Height                         2
```

```
#checking missing data
map_dbl(df, function(.x) {sum(is.na(.x))})
```

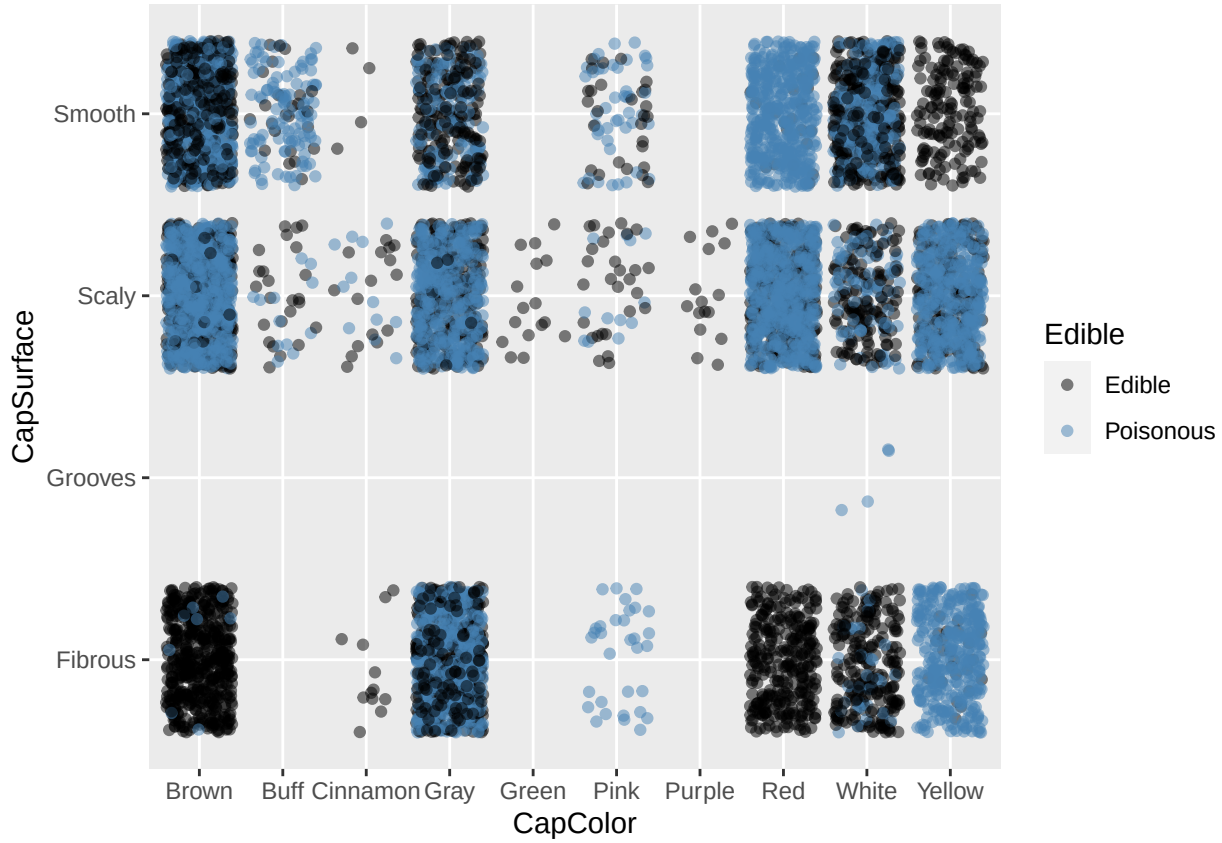
```
##     Edible   CapShape CapSurface   CapColor       Odor     Height
##          0          0          0          0          0          0
```

Here we can see that we don't have any missing values in our dataset.

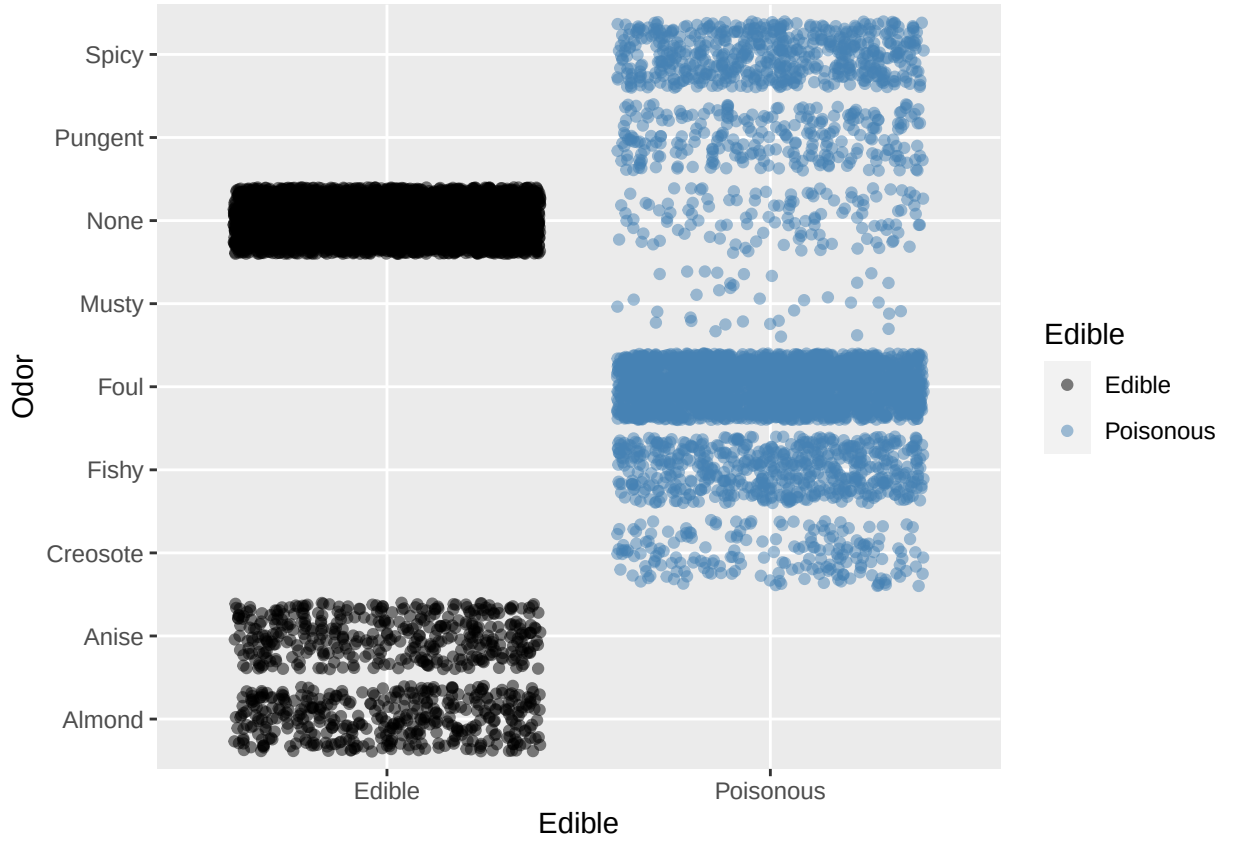
```
#visualisations
ggplot(df, aes(x = CapShape, y = CapSurface, col = Edible)) +
  geom_jitter(alpha = 0.5) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("black", "steelblue"))
```




```
ggplot(df, aes(x = CapColor, y = CapSurface, col = Edible)) +
  geom_jitter(alpha = 0.5) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("black", "steelblue"))
```




```
ggplot(df, aes(x = Edible, y = Odor, col = Edible)) +
  geom_jitter(alpha = 0.5) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("black", "steelblue"))
```




```
set.seed(1800) #For reproducibility
#creating partition for training data
df_sample = createDataPartition(df$Edible, p=0.7, list= F)
df_train <- df[df_sample, ]
df_test <- df[-df_sample, ]
```

The complexity parameter (cp) is used to determine the optimal tree size and control the size of the decision tree. Tree building terminates if the cost of adding another variable to the decision tree from the current node exceeds the value of cp. We could alternatively argue that tree construction will stop unless the overall lack of fit is reduced by a factor of cp.

Putting CP value as zero will build a tree with its maximum depth. Hence, it will build a very large tree.

```
TreeModel1 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.002)
TreeModel2 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.003)
TreeModel3 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.004)
TreeModel4 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.00001)

cfmat1=confusionMatrix(data=predict(TreeModel1, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
cfmat2=confusionMatrix(data=predict(TreeModel2, type = "class"),
                        reference = df_train$Edible,
                        positive="Edible")
cfmat3=confusionMatrix(data=predict(TreeModel3, type = "class"),
                        reference = df_train$Edible,
```

```
                        positive="Edible")
cfmat4=confusionMatrix(data=predict(TreeModel4, type = "class"),
                        reference = df_train$Edible,
                        positive="Edible")

cfmat1$overall[1]
```

```
##  Accuracy
## 0.9868143
```

```
cfmat2$overall[1]
```

```
##  Accuracy
## 0.9868143
```

```
cfmat3$overall[1]
```

```
##  Accuracy
## 0.9832982
```

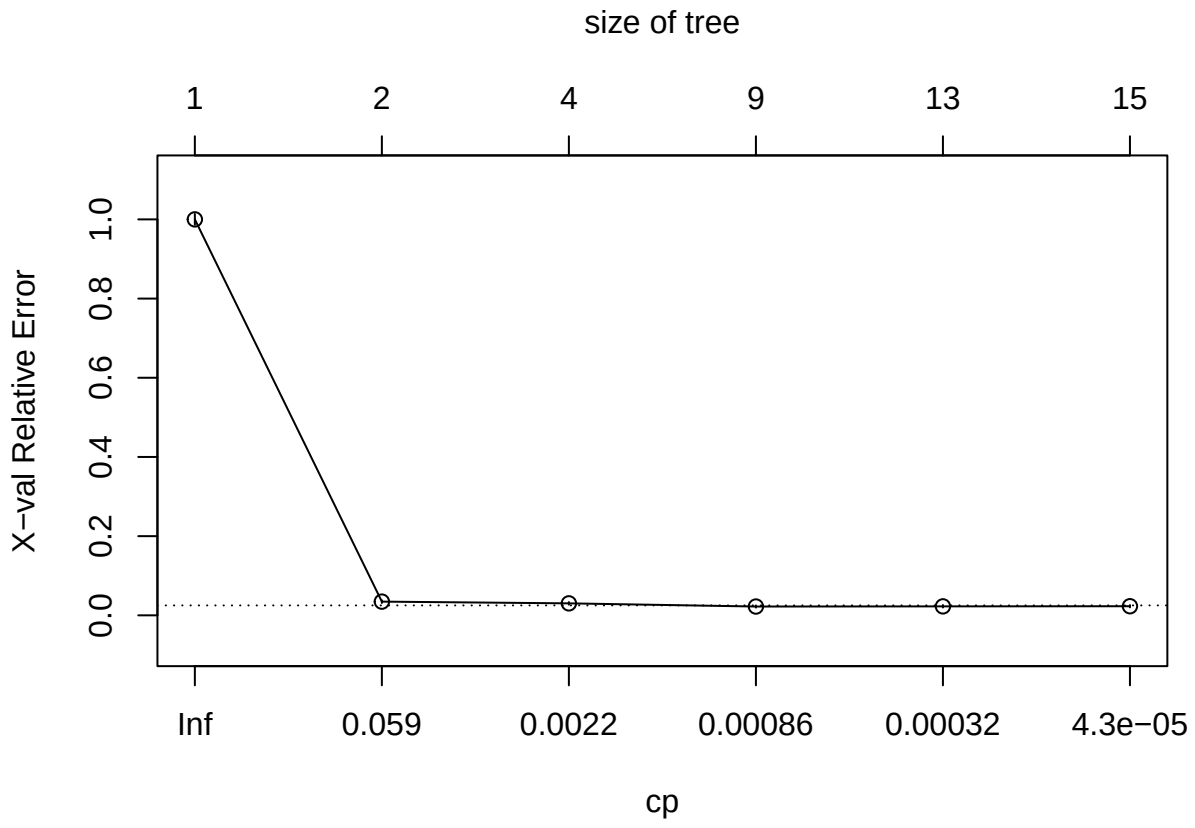
```
cfmat4$overall[1]
```

```
## Accuracy
## 0.991737
```

We used four randomly chosen cp values to train the decision tree model. Then we examine how well each model predicts.

- Model 1 and model 2 with cp value of 0.002 and 0.003 respectively have same accuracy.
- Model 4 predicts most accurately with the accuracy of 99.17%
- It's also worth noting that there's not much of a difference in accuracy between all of the models when comparing tree size, as larger cp values indicate a less complex tree that's easier to grasp and more computationally efficient.
- But for now we will choose cp value of 0.00001 for further analysis.

```
plotcp(TreeModel4)
```



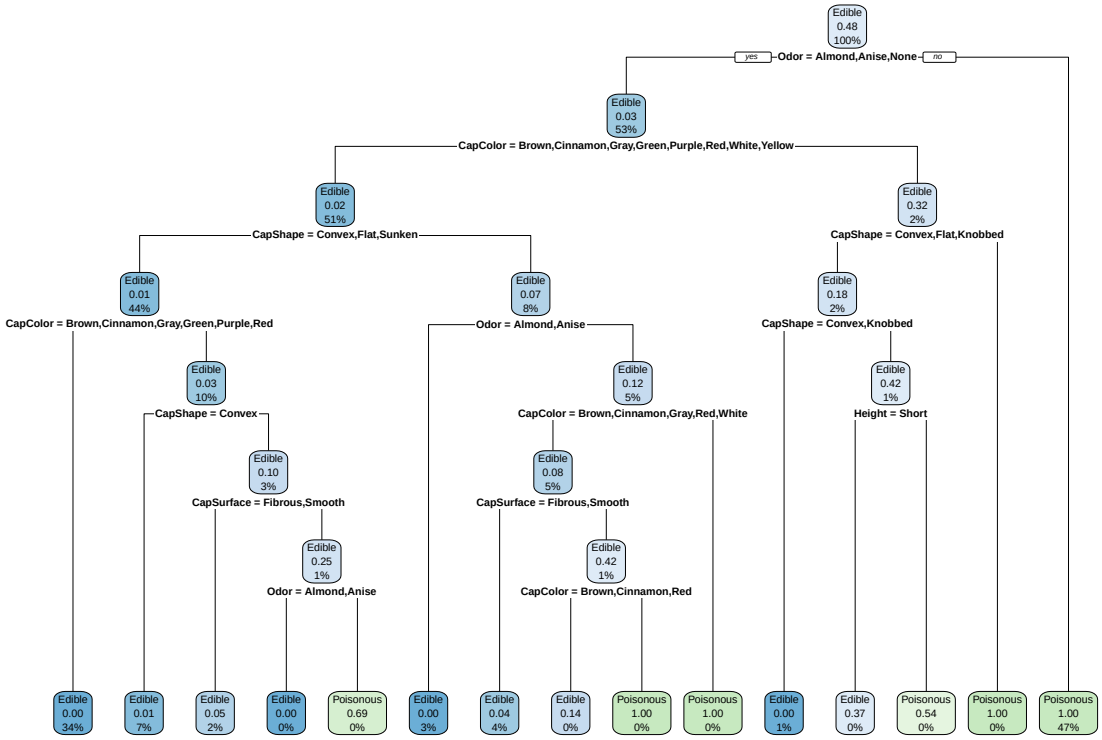

```
printcp(TreeModel4)
```

```
##
## Classification tree:
## rpart(formula = Edible ~ ., data = df_train, method = "class",
##     cp = 1e-05)
##
## Variables actually used in tree construction:
## [1] CapColor   CapShape   CapSurface Height     Odor
##
## Root node error: 2742/5688 = 0.48207
##
## n= 5688
##
##           CP nsplit rel error   xerror      xstd
## 1 0.96535376      0  1.000000 1.000000 0.0137437
## 2 0.00364697      1  0.034646 0.034646 0.0035248
## 3 0.00133722      3  0.027352 0.030270 0.0032982
## 4 0.00054705      8  0.019694 0.022247 0.0028331
## 5 0.00018235     12  0.017505 0.022611 0.0028559
## 6 0.00001000     14  0.017141 0.022976 0.0028786
```

- We can see from the plot and table that the cross validation error decreases as the cp value drops until the number of splits reaches 8, at which point it begins to increase. As a result, the cp value for it is 0.00054705, and the tree can be pruned with a cross validation error of 0.022247.

- Choosing a very small value of cp is very helpful to fit the model as we can see various values of cp for various tree sizes. The optimal value of cp can be chosen with lowest cross validation error. In our case the optimal number of splits is 8.
- We can now check the accuracy of the model by using optimum value of the cp.

```
rpart.plot(TreeModel4)
```



The plot here shows the the decision tree with cp value of 0.00001.

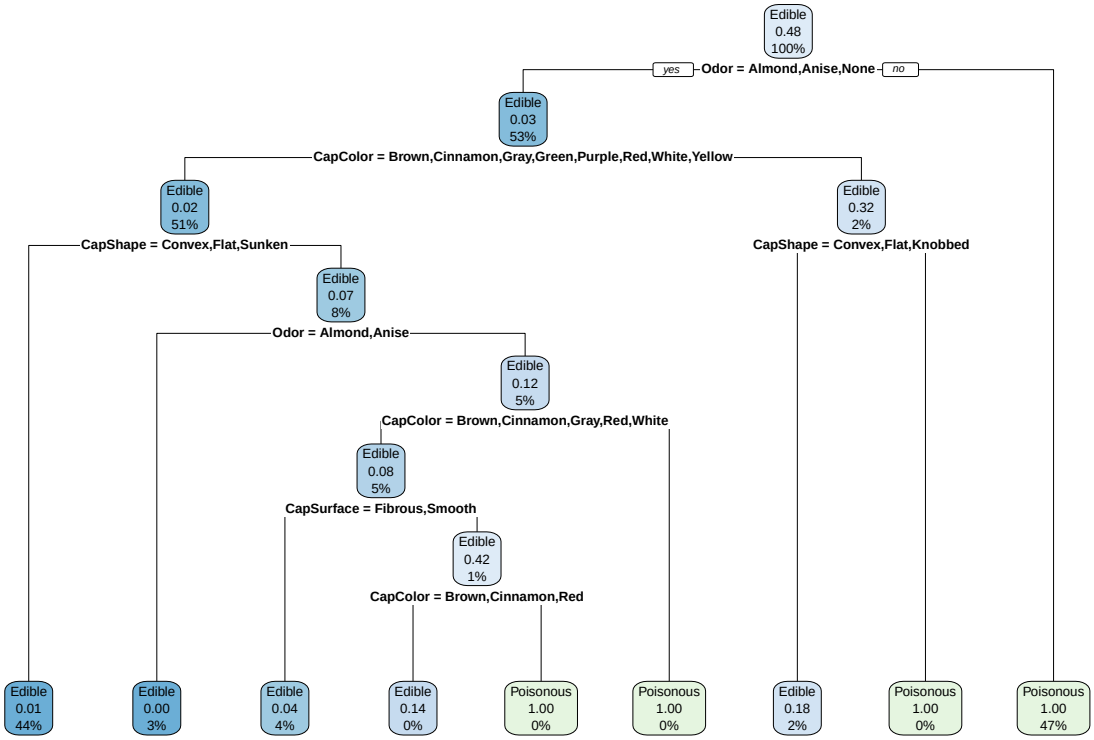
Now we will train the model with optimal value of cp with 8 splits and check the accuracy.

```
TreeModel5 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.00054705)
cfmat5=confusionMatrix(data=predict(TreeModel5, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
cfmat5$overall[1]
```

```
##  Accuracy
## 0.9905063
```

Here we see that the accuracy is 99.05% which is very close to 99.17%. Let's also look at the size of the decision tree.

```
rpart.plot(TreeModel5)
```



It's interesting to see from the plot that the accuracy doesn't seem to alter much as the tree gets smaller. As a result, it makes sense to use this model, which are both computationally efficient and easy to understand while maintaining the same level of accuracy.

Then, when the training data changes, we'll see how much the accuracy changes. As a result, we'll randomly split the data into testing and training sets fifty times with a cp value of 0.00054705.

```
#change in accuracy with variation in dataset
#Optimal cp value is chosen in every loop with lowest cross validation error and highest accuracy.
winner = rep(NA, 50)
for (iteration in 1:50){
  #Make a new random training data - test data split
  idx=createDataPartition(df$Edible, p=0.7, list= F)
  #idx = sample(1:8124, 5687)
  train_data = df[idx, ]
  test_data = df[-idx, ]
  cpval = rep(NA, 50)
  BestAcc = rep(NA, 50)

  for (i in 1:50){
    #Fit model with the training data
    TreeModelCV <- rpart(Edible ~ ., data = train_data,
                         method = "class", cp = 0.00054705)

    cpval[i]=TreeModelCV$cptable[which.min(TreeModelCV$cptable[, "xerror"]), "CP"]
    TreeModelCVPruned=prune(TreeModelCV, cp = cpval[i])
    cfmat=confusionMatrix(data=predict(TreeModelCVPruned, type = "class"),
```

```
                          reference = train_data$Edible,
                          positive="Edible")

    BestAcc[i]=cfmat$overall[1]
  }
  #Winning accuracy for this iteration is stored
  winner[iteration] = max(BestAcc)
}
```

```
#accuracy of 50 randomly chosen training dataset with
#cp value = 0.00054705
winner
```

```
##  [1] 0.9924402 0.9927918 0.9913854 0.9919128 0.9917370 0.9919128 0.9938467
##  [8] 0.9920886 0.9910338 0.9926160 0.9934951 0.9906821 0.9922644 0.9924402
## [15] 0.9905063 0.9908579 0.9912096 0.9910338 0.9933193 0.9915612 0.9912096
## [22] 0.9913854 0.9910338 0.9912096 0.9922644 0.9912096 0.9905063 0.9906821
## [29] 0.9912096 0.9931435 0.9906821 0.9910338 0.9920886 0.9931435 0.9931435
## [36] 0.9929677 0.9901547 0.9920886 0.9912096 0.9917370 0.9905063 0.9913854
## [43] 0.9922644 0.9913854 0.9906821 0.9912096 0.9926160 0.9908579 0.9912096
## [50] 0.9926160
```

The results above show that even when the training dataset varies, the accuracy does not change significantly. We acquire a maximum accuracy of 99.38%.

```
#cp value corresponding to max accuracy
MostAccModel=which.max(winner)
bestcpval=cpval[MostAccModel]
bestcpval
```

```
## [1] 0.00054705
```

The best cp value with lowest cross validation error and highest accuracy(99.38%) is still the same. Hence, we choose our final CP value as 0.00054705.

Let's check the performance of the final model on the testing data set.

```
#Accuracy in testing data with optimal cp val
#C.P = 0.00054705
confusionMatrix(data = predict(TreeModel5, newdata = df_test, type = "class"),
                reference = df_test$Edible,
                positive = "Edible")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Edible Poisonous
##   Edible      1262        16
##   Poisonous      0      1158
##
##                Accuracy : 0.9934
##                  95% CI : (0.9894, 0.9962)
```

```
##     No Information Rate : 0.5181
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9868
##
##  Mcnemar's Test P-Value : 0.0001768
##
##             Sensitivity : 1.0000
##             Specificity : 0.9864
##          Pos Pred Value : 0.9875
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5181
##          Detection Rate : 0.5181
##    Detection Prevalence : 0.5246
##       Balanced Accuracy : 0.9932
##
##        'Positive' Class : Edible
##
```

- The accuracy of prediction using the final model on the testing dataset is 99.34%.
- It is fine to eat mushroom which has been predicted as poisonous but when it is edible in reality. On the other hand, if the mushroom is poisonous in reality but the model predicted it as edible is very dangerous.
- From the confusion matrix, we can see that the probability of eating a poisonous mushroom predicted as edible is 0.65%. Hence, the chances of eating a poisonous mushroom is extremely low.
- Odor, CapColor, CapShape, and CapSurface are the four most important characteristics in predicting the target variables.

```
#Accuracy of model with cp = 0.00001 on testing data
confusionMatrix(data = predict(TreeModel4, newdata = df_test, type = "class"),
                reference = df_test$Edible,
                positive = "Edible")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Edible Poisonous
##   Edible      1252        13
##   Poisonous     10      1161
##
##                Accuracy : 0.9906
##                  95% CI : (0.9859, 0.994)
##     No Information Rate : 0.5181
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9811
##
##  Mcnemar's Test P-Value : 0.6767
##
##             Sensitivity : 0.9921
##             Specificity : 0.9889
```

```
##          Pos Pred Value : 0.9897
##          Neg Pred Value : 0.9915
##              Prevalence : 0.5181
##          Detection Rate : 0.5140
##    Detection Prevalence : 0.5193
##       Balanced Accuracy : 0.9905
##
##        'Positive' Class : Edible
##
```