Assignment3

Surapot Nonpassopon

2024-05-28

Assignment3 - Predict edible of mushrooms

Goal: To determine whether the mushroom is edible or poisonous based on its characteristics.

To progress the task of determining whether the mushroom is edible or poisonous based on its characteristics. It should be realised that this is the task of classification to predict the result (output), which is a binary variable that is edible or poisonous (1 or 0).

In this mushroom data set, the Input variables are: 1. CapShape 2. CapSurface 3. CapColour 4. Odor 5. Height

This mushroom data set output variable which needs to be predicted is edibility.

Thus, this assignment will produce the decision tree and random forest to deal with the classification problem to predict the edible or poisonous mushrooms.

load the nessessry libraries.

```
# Load libraries
library(caret)
library(dplyr)
library(rpart)
library(rpart.plot)
library(combinat)
library(randomForest)
```

First, input the mushroom datasets to R.

```
# input the dataset
mushroom <- read.csv('mushrooms.csv')
```

Second, summarise the data to observe the type, which is the character and number of the row, which is 8124.

```
# data observation
summary(mushroom)
```

```
##     Edible            CapShape          CapSurface          CapColor
##  Length:8124        Length:8124        Length:8124        Length:8124
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      Odor              Height
##  Length:8124        Length:8124
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```
head(mushroom)
```

```
##      Edible CapShape CapSurface CapColor    Odor Height
## 1 Poisonous   Convex     Smooth    Brown Pungent   Tall
```

```
## 2    Edible   Convex     Smooth   Yellow  Almond  Short
## 3    Edible     Bell     Smooth    White   Anise   Tall
## 4 Poisonous   Convex      Scaly    White Pungent  Short
## 5    Edible   Convex     Smooth     Gray    None  Short
## 6    Edible   Convex      Scaly   Yellow  Almond  Short
```

Third, to produce the decision tree and random forest model, it needs to change the mushroom edibility columns type from character to the factor variable, which is represented by the table below.

Table 1: Edibility and Factor Variable

Edibility	Factor variable
Edible	0
Poisonous	1

```
# tranform the data
# Transform the 'Edible' column: 'Edible' = 0, 'Poisonous' = 1
mushroom <- mushroom %>%
  mutate(Edible = ifelse(Edible == "Edible", 0, 1))
# Ensure the 'Edible' column is a factor
mushroom$Edible <- factor(mushroom$Edible, levels = c(0, 1))
```

Task1

1.1 Decision tree for predicting the edibility based on all of the remaining attributes or randomly selected attributes; after that selected the model input variables which gives the best performance to tuning the model later.

Step 1: Define the formula of the decision tree with different input lists, which will have 31 different models.

```
# Define predictor variables
predictors <- c("CapShape", "CapSurface", "CapColor", "Odor", "Height")
# Generate the combinations of predictor variables
all_combinations <- function(x) {
  unlist(lapply(1:length(x), function(i) combn(x, i, simplify = FALSE)), recursive = FALSE)
}

combinations <- all_combinations(predictors)
# Create a list of all formula
formulas <- sapply(combinations, function(combo) {
  as.formula(paste("Edible ~", paste(combo, collapse = " + ")))
})

print(formulas)
```

```
## [[1]]
## Edible ~ CapShape
## <environment: 0x10ffd4cc8>
##
## [[2]]
## Edible ~ CapSurface
## <environment: 0x10ffda2b8>
##
## [[3]]
## Edible ~ CapColor
```

```
## <environment: 0x118144748>
##
## [[4]]
## Edible ~ Odor
## <environment: 0x118146b70>
##
## [[5]]
## Edible ~ Height
## <environment: 0x118148fd0>
##
## [[6]]
## Edible ~ CapShape + CapSurface
## <environment: 0x118152a90>
##
## [[7]]
## Edible ~ CapShape + CapColor
## <environment: 0x118165f90>
##
## [[8]]
## Edible ~ CapShape + Odor
## <environment: 0x11816e438>
##
## [[9]]
## Edible ~ CapShape + Height
## <environment: 0x1181707b8>
##
## [[10]]
## Edible ~ CapSurface + CapColor
## <environment: 0x118174b38>
##
## [[11]]
## Edible ~ CapSurface + Odor
## <environment: 0x118176e48>
##
## [[12]]
## Edible ~ CapSurface + Height
## <environment: 0x118179200>
##
## [[13]]
## Edible ~ CapColor + Odor
## <environment: 0x11817b580>
##
## [[14]]
## Edible ~ CapColor + Height
## <environment: 0x11817d900>
##
## [[15]]
## Edible ~ Odor + Height
## <environment: 0x11817fcb8>
##
## [[16]]
## Edible ~ CapShape + CapSurface + CapColor
## <environment: 0x118182038>
##
```

```
## [[17]]
## Edible ~ CapShape + CapSurface + Odor
## <environment: 0x118184a58>
##
## [[18]]
## Edible ~ CapShape + CapSurface + Height
## <environment: 0x118188d68>
##
## [[19]]
## Edible ~ CapShape + CapColor + Odor
## <environment: 0x11818cfd0>
##
## [[20]]
## Edible ~ CapShape + CapColor + Height
## <environment: 0x11818e4e0>
##
## [[21]]
## Edible ~ CapShape + Odor + Height
## <environment: 0x1181933c0>
##
## [[22]]
## Edible ~ CapSurface + CapColor + Odor
## <environment: 0x118195548>
##
## [[23]]
## Edible ~ CapSurface + CapColor + Height
## <environment: 0x118197858>
##
## [[24]]
## Edible ~ CapSurface + Odor + Height
## <environment: 0x118199b30>
##
## [[25]]
## Edible ~ CapColor + Odor + Height
## <environment: 0x11819be40>
##
## [[26]]
## Edible ~ CapShape + CapSurface + CapColor + Odor
## <environment: 0x1181a00e0>
##
## [[27]]
## Edible ~ CapShape + CapSurface + CapColor + Height
## <environment: 0x11819e400>
##
## [[28]]
## Edible ~ CapShape + CapSurface + Odor + Height
## <environment: 0x1181a0630>
##
## [[29]]
## Edible ~ CapShape + CapColor + Odor + Height
## <environment: 0x1181a2898>
##
## [[30]]
## Edible ~ CapSurface + CapColor + Odor + Height
```

```
## <environment: 0x1181c0b70>
##
## [[31]]
## Edible ~ CapShape + CapSurface + CapColor + Odor + Height
## <environment: 0x1181e0e48>
```

Step 2: Define the different parameters to tune the model. The hyperparameter that chooses to tune the model is given the definition and value in the table below.

```
# Create the table data
hyperparameters <- data.frame(
  Hyperparameters = c("Minsplit", "CP (Complexity Parameter)", "Maxdepth"),
  Definition = c("The minimum number of observations that must exist in the node for the split to be at
                 "This one is used to control the size of the decision tree and to select the optimal t
                 "The maximum depth of any node in the final tree."),
  `Tuning Value` = c("10, 20, 30", "0.0001, 0.001, 0.01", "5, 10, 15")
)

# Print the table using kable
knitr::kable(hyperparameters, caption = "Hyperparameters for Decision Tree Tuning")
```

Table 2: Hyperparameters for Decision Tree Tuning

Hyperparameters	Definition	Tuning.Value
Minsplit	The minimum number of observations that must exist in the node for the split to be attempted.	10, 20, 30
CP (Complexity Parameter)	This one is used to control the size of the decision tree and to select the optimal tree size.	0.0001, 0.001, 0.01
Maxdepth	The maximum depth of any node in the final tree.	5, 10, 15

```
# Define a grid of parameters to tune
tune_grid <- expand.grid(
  minsplit = c(10, 20, 30),
  cp = c(0.0001, 0.001, 0.01),
  maxdepth = c(5, 10, 15))
```

Step 3: Create the data frame to store the accuracy (the percentage of correctly predicting the edible) of the results.

```
# Initialise a data frame to store accuracy results
results_df <- data.frame(
  Formula = character(),
  Minsplit = integer(),
  Cp = numeric(),
  Maxdepth = integer(),
  Accuracy = numeric())
```

Step 4: Sample the data into the training set at 80% of the data and the test set at 20% of the data.

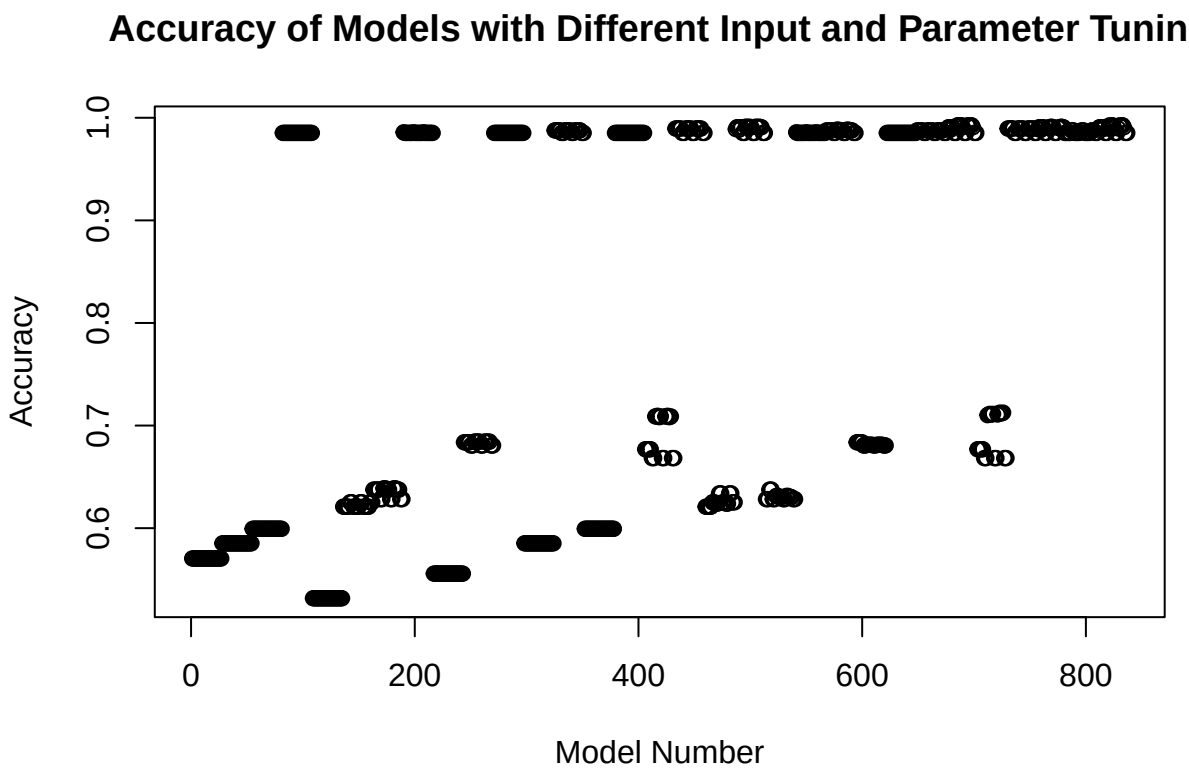
```
# Sample data 80/20 without replacement
set.seed(123)
trainindex <- sample(seq_len(nrow(mushroom)), size = 0.8 * nrow(mushroom))
traindata <- mushroom[trainindex, ]
testdata <- mushroom[-trainindex, ]
```

Step 5: Fitting the model with a different list of inputs and hyperparameters. Store the accuracy evaluation metric, which is the percentage of correctly classified results, to compare the results of different models.

```
# Loop through the formulas
for (f in 1:length(formulas)) {
  formula <- formulas[[f]]

  # Loop through the grid of parameters
  for (i in 1:nrow(tune_grid)) {
    params <- tune_grid[i, ]

    # Fit the decision tree model with adjusted parameters
    mushroomtree <- rpart(formula = formula,
                          data = traindata,
                          method = 'class',
                          control = rpart.control(minsplit = params$minsplit,
                                                  cp = params$cp,
                                                  maxdepth = params$maxdepth))

    # Predict on the test set
    predictionDT <- predict(mushroomtree, newdata = testdata, type = "class")

    # Ensure the predicted values and actual values are factors in same levels
    predictionsDT <- factor(predictionDT, levels = levels(testdata$Edible))

    # Create confusion matrix to evaluate the result
    conf_matrix <- confusionMatrix(predictionsDT, testdata$Edible)

    # Collect the value of accuracy from the confusion matrix
    accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)

    # Store the results
    results_df <- rbind(results_df, data.frame(
      Formula = deparse(formula),
      Minsplit = params$minsplit,
      Cp = params$cp,
      Maxdepth = params$maxdepth,
      Accuracy = accuracy
    ))
  }
}
```

Step 6: Evaluate the result by plotting the graph of accuracy in different models and identify which model has the highest accuracy of prediction. However, this result is non-stable and reliable, which should progress the 100 times cross-validation to confirm and make it stable and reliable result, which will be done in task 2.

```
# Plot the accuracy of models with different parameter combinations
plot(1:nrow(results_df), results_df$Accuracy,
     xlab = "Model Number", ylab = "Accuracy",
     main = "Accuracy of Models with Different Input and Parameter Tuning")
```




```
# Identify the model with the highest accuracy
best_index <- which.max(results_df$Accuracy)
best_model <- results_df[best_index, ]

cat("The model with the highest accuracy:\n")
```

```
## The model with the highest accuracy:
```

```
cat("Formula:", best_model$Formula, "\n")
```

```
## Formula: Edible ~ CapShape + CapSurface + CapColor + Odor
```

```
cat("Parameters: minsplit =", best_model$Minsplit, ", cp =", best_model$Cp, ", maxdepth =", best_model$
```

```
## Parameters: minsplit = 10 , cp = 1e-04 , maxdepth = 10
```

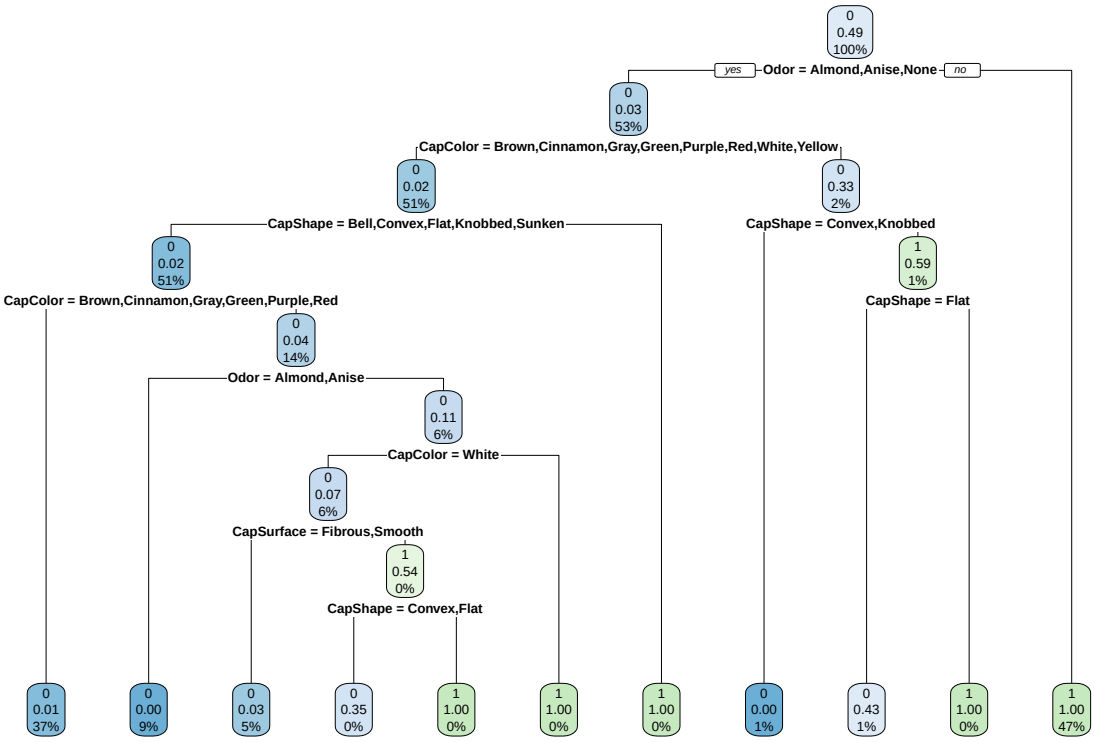
```
cat("Accuracy:", best_model$Accuracy, "\n")
```

```
## Accuracy: 0.9926154
```

Step 7: Visualise the final fitted decision tree model which is the model with 4 input variables (CapShape, CapSurface, Capcolor, Odor) by setting the hyperparameters turning as (minsplit = 10, cp = 0.0001, maxdepth = 10).

```
# Visualise the best model
# Fit and plot the best model based on highest accuracy
best_formula <- as.formula(best_model$Formula)
best_params <- best_model[, c("Minsplit", "Cp", "Maxdepth")]

best_mushroomtree <- rpart(formula = best_formula,
                           data = traindata,
                           method = 'class',
                           control = rpart.control(minsplit = best_params$Minsplit,
```

```
                                                cp = best_params$Cp,
                                                maxdepth = best_params$Maxdepth))
# Plot the decision tree
rpart.plot(best_mushroomtree)
```



comment on the explainability/interpretability of the model.

The decision tree visualisation above provides a clear view of classifying the mushrooms as either edible or poisonous. The tree starts with the root node, which splits based on the first feature, which is Odor. Each subsequent node splits further based on other features like 'Capcolor', 'CapShape' and 'CapSurface'. Each node is represented by 3 values of number; the first one is the predicted class (edible = 0; poisonous = 1). The second one is the probability of the class. The third one is the percentage of the sample at each node. Additionally, the colour of the node represents the edible mushrooms (blue for edible and green for poisonous).

Overall, the decision tree visualisation can be used for some interpretable for the classification of the output; however, it still has some limitations that the decision tree cannot use to interpret the model parameters, for example, logistic regression model or linear regression model that we can understand the magnitude and the sign of the parameter (positive or negative).

The decision tree also has some other limitations, such as the result of the decision tree is not stable; slightly different data can produce very different trees. To tackle this problem, it could be using BAGGING methods to produce many different trees with errors that are largely independent of each other. And then ensemble them together to predict the result as the Condorcet Jury Theorem will give a more accurate result.

1.2 Perform the random forest to predict the edibility based on feature selection and the turning of some model hyperparameters.

Step 1: Define the formula of the random forest with different input lists, which will have 31 different models, same as task 1.1.

```
# Define predictor variables
predictors <- c("CapShape", "CapSurface", "CapColor", "Odor", "Height")
# Generate the combinations of predictor variables
all_combinations <- function(x) {
  unlist(lapply(1:length(x), function(i) combn(x, i, simplify = FALSE)), recursive = FALSE)
}

combinations <- all_combinations(predictors)
# Create a list of all formula
formulas <- sapply(combinations, function(combo) {
  as.formula(paste("Edible ~", paste(combo, collapse = " + ")))
})
```

Step 2: Define the different parameters to tune the model. The parameter that is chosen to tune the model is the number of trees, starting at 50 trees increased by 50 trees until it reaches 500 trees.

Increasing the number of trees by more than 500 might increase the performance of the random forest models. However, it needs longer training times, which might be challenging for a computer with limited CPU power as my computer. Thus, this task will limit the number tree to 500.

```
# Set up the parameter of ntree
ntree_grid <- seq(50, 500, by = 50)
```

Step 3: Create the data frame to store the accuracy (the percentage of correctly predicting the edible) of the results.

```
# Initialise a data frame to store accuracy results
results_df <- data.frame(
  Formula = character(),
  Ntree = integer(),
  Accuracy = numeric()
)
```

Step 4: Sample the data into the training set at 80% of the data and the test set at 20% of the data.

```
# Sample data 80/20 without replacement
set.seed(123)
trainindex <- sample(seq_len(nrow(mushroom)), size = 0.8 * nrow(mushroom))
traindata <- mushroom[trainindex, ]
testdata <- mushroom[-trainindex, ]
```

Step 5: Fitting the random forest models with a different list of inputs and number of trees. Store the accuracy evaluation metric, which is the percentage of correctly classified results, to compare the results of different models.

```
# Make sure that the levels of factor variable in the test set match the training set
for (var in predictors) {
  if (is.factor(traindata[[var]])) {
    levels(testdata[[var]]) <- levels(traindata[[var]])
  }
}

# Loop through the formula
```

```r
for (f in 1:length(formulas)) {
  formula <- formulas[[f]]

  # Loop through the ntree parameters
  for (ntree in ntree_grid) {

    # Fit the random forest model
    mushroom_rf <- randomForest(formula, data = traindata, ntree = ntree)

    # Predict the value based on the test set
    predictionRF <- predict(mushroom_rf, newdata = testdata, type = "prob")[, 2]

    # Rescale the prediction probabilities
    # make sure that not include the way that have only one type result
    prediction_probs <- (ntree * predictionRF + 1) / (ntree + 2)

    # Convert probabilities to class label
    predicted_classes <- ifelse(prediction_probs > 0.5, 1, 0)

    # Make sure the predicted values and actual values are factors with the same levels
    predictionsRF <- factor(predicted_classes, levels = levels(testdata$Edible))

    # Create confusion matrix to evaluate the result
    conf_matrix <- confusionMatrix(predictionsRF, testdata$Edible)

    # Collect the accuracy from confusion metric
    accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)

    # Store the results
    results_df <- rbind(results_df, data.frame(
      Formula = deparse(formula),
      Ntree = ntree,
      Accuracy = accuracy
    ))
  }
}
```

Step 6: Evaluate the result by plotting the graph of accuracy in different models and identify which model has the highest accuracy of prediction.

To confirm a more stable result, the process should be repeated the process by different training and test sets many times and count the model with the most winning (highest accuracy) will be performed in task 2.

```r
# Finding the model with the highest accuracy
best_index <- which.max(results_df$Accuracy)
best_model <- results_df[best_index, ]

cat("The model with the highest accuracy:\n")
```

```
## The model with the highest accuracy:
```

```r
cat("Formula:", best_model$Formula, "\n")
```

```
## Formula: Edible ~ CapShape + CapSurface + CapColor + Odor
```

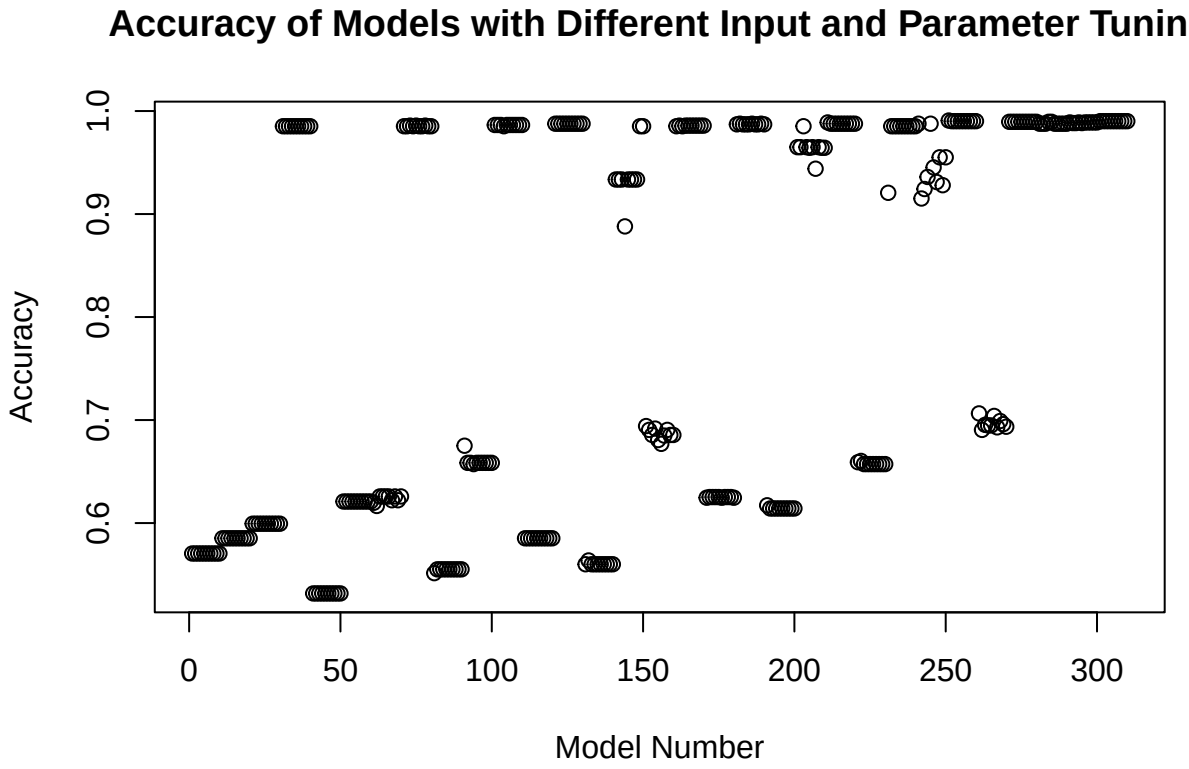
```r
cat("Parameters: ntree =", best_model$Ntree, "\n")
```

```
## Parameters: ntree = 50
```

```r
cat("Accuracy:", best_model$Accuracy, "\n")
```

```
## Accuracy: 0.9907692
```

```r
# Visualise the best model
# Fit and plot the best model based on highest accuracy
best_formula <- as.formula(best_model$Formula)
best_ntree <- best_model$Ntree

best_rf <- randomForest(formula = best_formula,
                        data = traindata,
                        ntree = best_ntree)

# Plot of the accuracy of models with different parameter and tuning
plot(1:nrow(results_df), results_df$Accuracy,
     xlab = "Model Number", ylab = "Accuracy",
     main = "Accuracy of Models with Different Input and Parameter Tuning")
```



comment on the explainability/interpretability of the model.

The figure above presents the accuracy of the model with different inputs and parameters which the insight that we get from the figure is the accuracy of the model ranges between approximately 0.6 to 10, with a very high accuracy of the prediction near the perfect result. It could be seen that clusters of some models perform well, while some models show poorer performance between 0.6 and 0.7.

Moreover, the figure suggests that feature selection and turning of the parameters significantly influence the model performance, with the model having fewer features, and a lower number of trees seeming to have lower accuracy.

This model provides high accuracy of the result; however, In terms of interpreting the model, this model has limitations in that it cannot interpret the same as another model that has regression coefficients and cannot understand the importance of each input. Cannot know the direction of the input or understand the uncertain effect such as confident interval.

2.1 Use cross-validation to determine which model from task 1 performs best (which model classifies most mushrooms correctly) by starting to determine the decision tree model first.

Step 1: Same as task one, define the decision tree with different input lists, which will have 31 different models.

```
# Define predictor variables
predictors <- c("CapShape", "CapSurface", "CapColor", "Odor", "Height")
# Generate the combinations of predictor variables
all_combinations <- function(x) {
  unlist(lapply(1:length(x), function(i) combn(x, i, simplify = FALSE)), recursive = FALSE)
}

combinations <- all_combinations(predictors)
# Create a list of all formula
formulas <- sapply(combinations, function(combo) {
  as.formula(paste("Edible ~", paste(combo, collapse = " + ")))
})
```

Step 2: Define the different parameters for tuning the model, like task 1.1, and the grid of parameters to tune is presented in the code below.

```
# Define a grid of parameters to tune the model
tune_grid <- expand.grid(
  minsplit = c(10, 20, 30),
  cp = c(0.0001, 0.001, 0.01),
  maxdepth = c(5, 10, 15))
```

Step 3: Create the data frame to store the accuracy, which is the percentage of correctly predicting the output. And create the vector of winning to count the number of winnings.

```
# Initialise a data frame to store accuracy results
results_df <- data.frame(
  Formula = character(),
  Minsplit = integer(),
  Cp = numeric(),
  Maxdepth = integer(),
  Accuracy = numeric(),
  Wins = integer())

# Initialise a matrix to store accuracy results
accuracies_all <- matrix(NA, nrow = 20, ncol = length(formulas) * nrow(tune_grid))

# Initialise a vector to count the number of wins
winner <- rep(NA, 20)
```

Step 4: Perform the cross-validation 20 times, with counting the winning model which has the highest accuracy as the winning model.

This step can progress more cross-validation times to obtain more reliable results. However, it still has some limitations on the computer's CPU power and takes time to run, so I decided to run 20 times cross-validation for this task.

```
# Perform cross-validation 100 times
for (iteration in 1:20) {
  # Sample data 80/20 without replacement
  set.seed(iteration)
  trainindex <- sample(seq_len(nrow(mushroom)), size = 0.8 * nrow(mushroom))
```

```r
traindata <- mushroom[trainindex, ]
testdata <- mushroom[-trainindex, ]

# Make sure the levels of factors in the test set match the training set
for (var in predictors) {
  if (is.factor(traindata[[var]])) {
    levels(testdata[[var]]) <- levels(traindata[[var]])
  }
}

# Loop through the formula
count <- 1
for (f in 1:length(formulas)) {
  formula <- formulas[[f]]

  # Loop through the grid of parameters
  for (i in 1:nrow(tune_grid)) {
    params <- tune_grid[i, ]

    # Fit the decision tree model with adjusted parameters
    mushroomtree <- rpart(formula = formula,
                          data = traindata,
                          method = 'class',
                          control = rpart.control(minsplit = params$minsplit,
                                                  cp = params$cp,
                                                  maxdepth = params$maxdepth))

    # Predict the value by using test set
    predictionDT <- predict(mushroomtree, newdata = testdata, type = "class")

    # Make sure the predicted values and actual values are factors with the same levels
    predictionsDT <- factor(predictionDT, levels = levels(testdata$Edible))

    # Create confusion matrix to evaluate the result
    conf_matrix <- confusionMatrix(predictionsDT, testdata$Edible)

    # Collect the accuracy from confusion metric
    accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)

    # Store the accuracy result
    accuracies_all[iteration, count] <- accuracy

    # Add the result to results_df dataframe
    results_df <- rbind(results_df, data.frame(
      Formula = deparse(formula),
      Minsplit = params$minsplit,
      Cp = params$cp,
      Maxdepth = params$maxdepth,
      Accuracy = NA,
      Wins = 0
    ))
    count <- count + 1
  }
```

```
  }

  # Identify the formula with the highest accuracy
  winner[iteration] <- which.max(accuracies_all[iteration, ])
}

# Calculate the average accuracy
average_accuracies <- colMeans(accuracies_all, na.rm = TRUE)

# Calculate the number of wins
wins <- table(factor(winner, levels = 1:(length(formulas) * nrow(tune_grid))))

# Update the accuracy and wins in results_df
results_df$Accuracy <- average_accuracies
results_df$Wins <- as.numeric(wins)
```

Step 5: Identify the model which has the highest average accuracy after running the decision tree tuning model to cross-validate 20 times.

```
# Identify the model with the highest average accuracy
best_index <- which.max(average_accuracies)
best_params <- results_df[best_index, ]
best_average_accuracy <- best_params$Accuracy

cat("The model with the highest average accuracy:\n")
```

```
## The model with the highest average accuracy:
```

```
cat("Formula:", best_params$Formula, "\n")
```

```
## Formula: Edible ~ CapShape + CapSurface + CapColor + Odor
```

```
cat("Parameters: minsplit =", best_params$Minsplit, ", cp =", best_params$Cp, ", maxdepth =", best_para
```

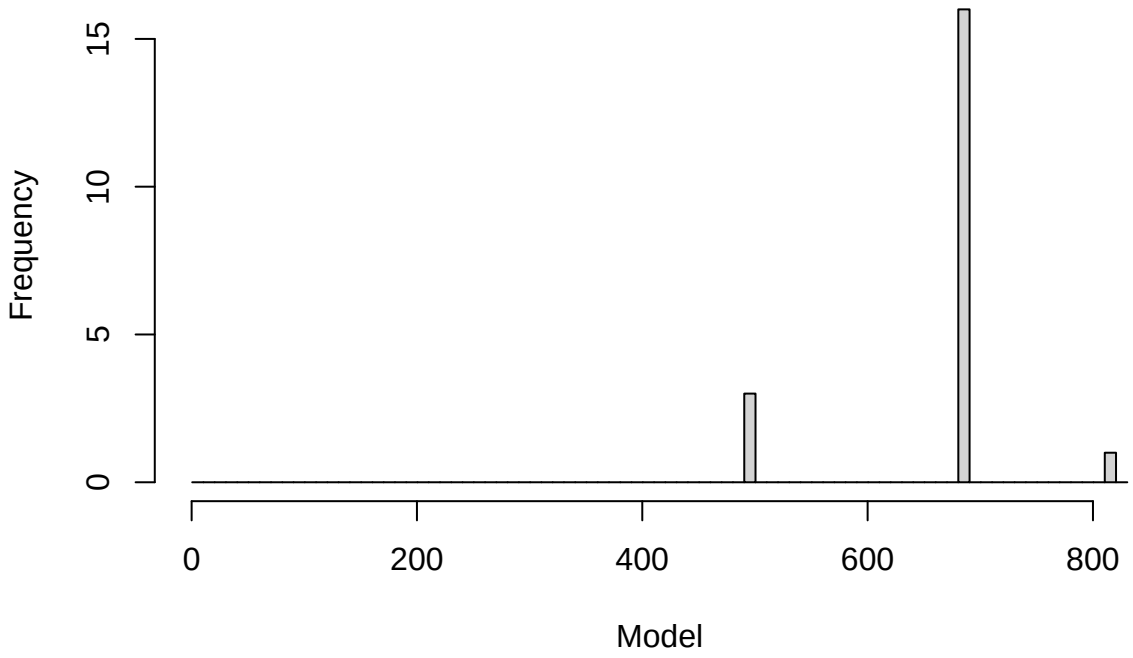
```
## Parameters: minsplit = 10 , cp = 1e-04 , maxdepth = 10
```

```
cat("Average Accuracy:", best_average_accuracy, "\n")
```

```
## Average Accuracy: 0.9910769
```

Step 6: Identify the model which counts as the most winning model, as the histogram below shows the highest winning count.

```
# Plot the histogram of winning models
hist(winner, breaks = seq(0.5, length(formulas) * nrow(tune_grid) + 0.5, 10),
     xlab = 'Model', ylab = 'Frequency', main = 'Frequency of Winning Models')
```



The result suggests in a reliable and stable way that this model should be the model which has the best performance to predict the edible by the criteria of the accuracy of the prediction output.

```
# Identify the most winning model
most_winning_model_index <- which.max(results_df$Wins)
most_winning_model <- results_df[most_winning_model_index, ]

cat("The model with the most wins is:\n")
```

```
## The model with the most wins is:
```

```
cat("Formula:", most_winning_model$Formula, "\n")
```

```
## Formula: Edible ~ CapShape + CapSurface + CapColor + Odor
```

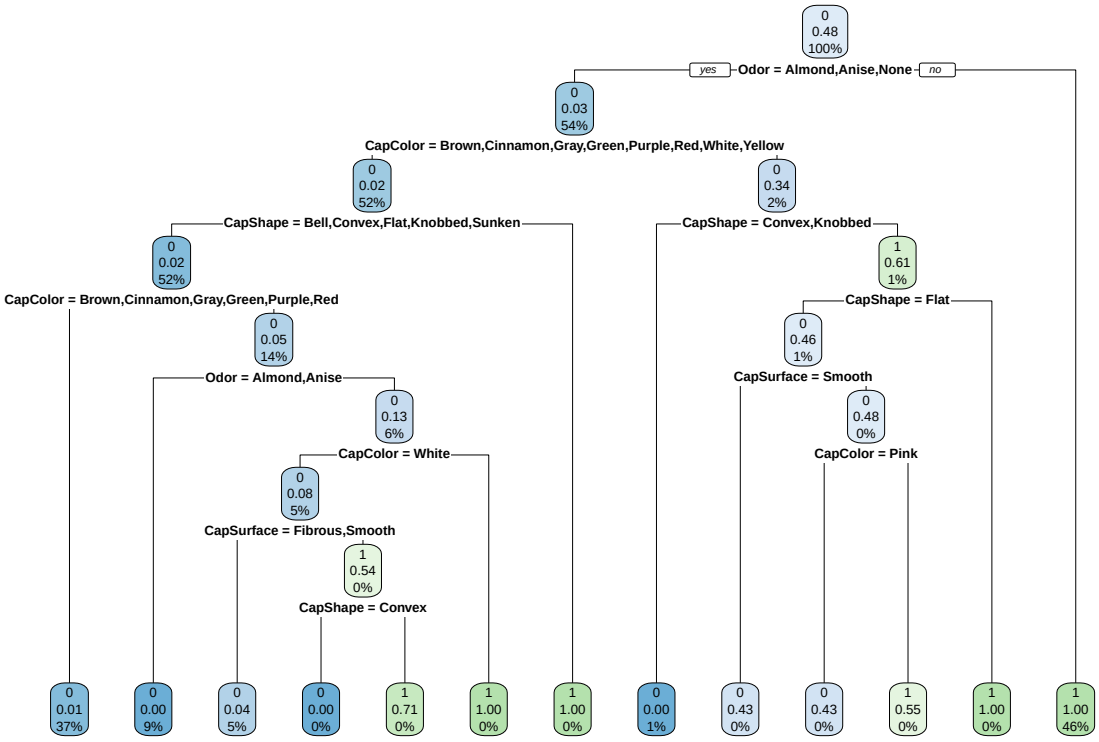
```
cat("Parameters: minsplit =", most_winning_model$Minsplit, ", cp =", most_winning_model$Cp, ", maxdepth
```

```
## Parameters: minsplit = 10 , cp = 1e-04 , maxdepth = 10
```

Step 7: Plot the best performance of the decision tree model.

```
# Visualise the best model
# Fit and plot the best model based on highest accuracy
best_formula <- as.formula(best_params$Formula)
best_minsplit <- best_params$Minsplit
best_cp <- best_params$Cp
best_maxdepth <- best_params$Maxdepth

# Fit the decision tree model with the best parameters
best_mushroomtree <- rpart(formula = best_formula,
                           data = traindata,
                           method = 'class',
                           control = rpart.control(minsplit = best_minsplit,
```

```
                                                cp = best_cp,
                                                maxdepth = best_maxdepth))

# Plot the decision tree
rpart.plot(best_mushroomtree)
```



Overview, the result from doing the cross-validation at task 2.1, we can see the advantage that it could be used to give the best idea by conducting repeated processes of cross-validation and providing the model that seems best make the result more stable and relaible. However, it takes a lot of time and makes the process slower because it needs to fit a lot of models, and sometimes it requires the use of high-power CPU computers.

2.2 Use cross-validation to determine which model from task 1 performs best (which model classifies most mushrooms correctly) by determining the random forest models.

Step 1: Same as task one, define the decision tree with different input lists, which will have 31 different models.

```
# Define predictor variables
predictors <- c("CapShape", "CapSurface", "CapColor", "Odor", "Height")
# Generate the combinations of predictor variables
all_combinations <- function(x) {
  unlist(lapply(1:length(x), function(i) combn(x, i, simplify = FALSE)), recursive = FALSE)
}

combinations <- all_combinations(predictors)
# Create a list of all formula
formulas <- sapply(combinations, function(combo) {
  as.formula(paste("Edible ~", paste(combo, collapse = " + ")))
})
```

Step 2: Define the different parameters for tuning the model in the random forest model. We will use the number of trees to turn the model, as in task 1.2. However, for this task, I decided to reduce the number of maximum trees from 500 to 150 trees to reduce the times of calculation based on the result in task 1.2 that the highest performance model has only 50 trees.

Thus, this task will start with the number of trees at 50 trees, increase by 50 trees, and end with 150 trees as the maximum by the limitation of CPU power.

```
# Set up the parameter of ntree
ntree_grid <- seq(50, 150, by = 50)
```

Step 3: Create the data frame to store the accuracies and create the vector of the winner to store the result of counting the winning models in cross-validation 20 times.

```
# Initialise a data frame to store accuracy results
results_df <- data.frame(
  Formula = character(),
  Ntree = integer(),
  Accuracy = numeric()
)

# Initialize a matrix to store accuracy results
accuracies_all <- matrix(NA, nrow = 20, ncol = length(formulas) * length(ntree_grid))

# Initialise a vector to count the number of wins
winner <- rep(NA, 20)
```

Step 4: Perform the cross-validation 20 times, counting the winning model, which has the highest accuracy as the winning model. This step can progress more cross-validation times to obtain more reliable results. However, it still has some limitations on the computer's CPU power and takes time to run.

```
# Perform cross-validation 20 times
for (iteration in 1:20) {
  # Sample data 80/20 without replacement
  set.seed(iteration)
  trainindex <- sample(seq_len(nrow(mushroom)), size = 0.8 * nrow(mushroom))
  traindata <- mushroom[trainindex, ]
  testdata <- mushroom[-trainindex, ]
```

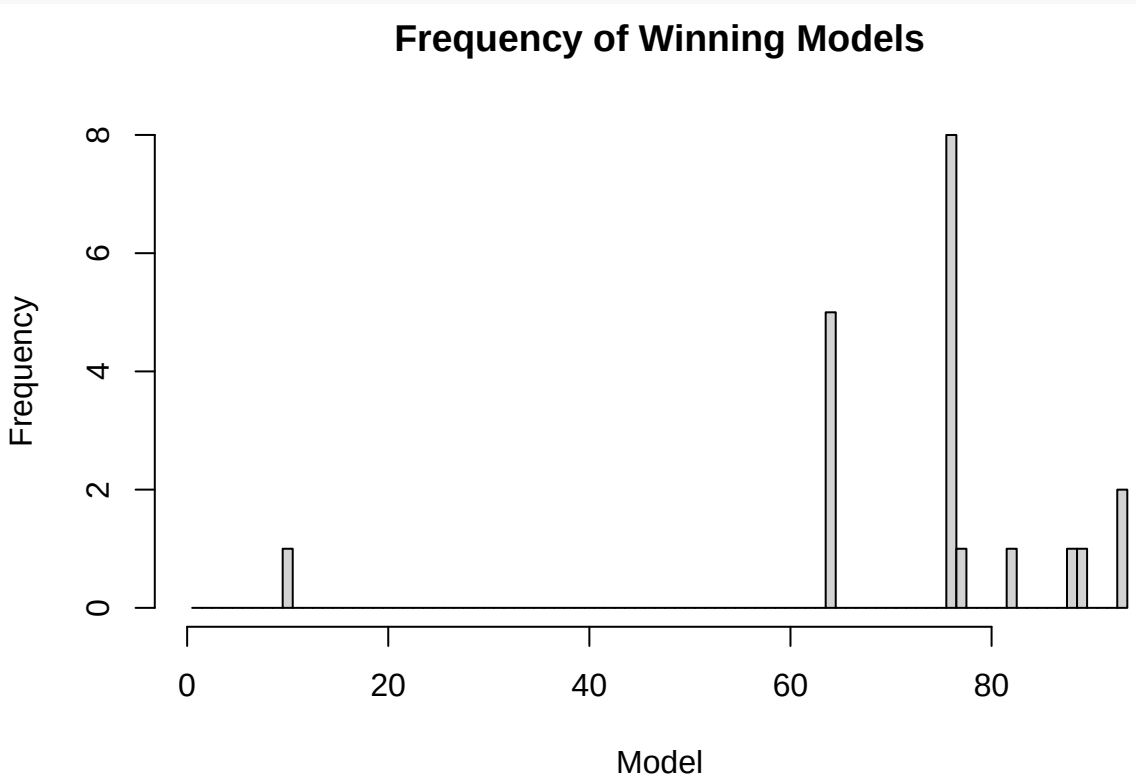
```r
# Make sure the levels of factors in the test set match the training set
for (var in predictors) {
  if (is.factor(traindata[[var]])) {
    levels(testdata[[var]]) <- levels(traindata[[var]])
  }
}

# Loop through the formula
count <- 1
for (f in 1:length(formulas)) {
  formula <- formulas[[f]]

  # Loop through the ntree grid
  for (ntree in ntree_grid) {

    # Fit the random forest model
    mushroom_rf <- randomForest(formula, data = traindata, ntree = ntree)

    # Predict on the test set
    predictionRF <- predict(mushroom_rf, newdata = testdata, type = "prob")[, 2]

    # Rescale the prediction probabilities
    # make sure that not include the way that have only one type result
    prediction_probs <- (ntree * predictionRF + 1) / (ntree + 2)

    # Convert probabilities to class labels
    predicted_classes <- ifelse(prediction_probs > 0.5, 1, 0)

    # Make sure the predicted values and actual values are factors at the same levels
    predictionsRF <- factor(predicted_classes, levels = levels(testdata$Edible))

    # Create confusion matrix to evaluate the result
    conf_matrix <- confusionMatrix(predictionsRF, testdata$Edible)

    # Collect the accuracy from confusion metric
    accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)

    # Store the accuracy result
    accuracies_all[iteration, count] <- accuracy

    # Add the result to results_df dataframe
    if (iteration == 1) {
      results_df <- rbind(results_df, data.frame(
        Formula = deparse(formula),
        Ntree = ntree,
        Accuracy = NA
      ))
    }
    count <- count + 1
  }
}

# Identify the formula which give highest accuracy
```

```
  winner[iteration] <- which.max(accuracies_all[iteration, ])
}

# Calculate the average accuracy
average_accuracies <- colMeans(accuracies_all, na.rm = TRUE)

# Calculate the number of wins
wins <- table(factor(winner, levels = 1:(length(formulas) * length(ntree_grid))))

# Update the accuracy and wins in results_df
results_df$Accuracy <- average_accuracies
results_df$Wins <- as.numeric(wins)
```

Step 5: Identify the random forest model which has the highest winning count after running the 20 times cross-validation, as can seen in the result from the histogram below.

This random forest, which is an ensemble learning of decision trees after the process of cross-validation, will give a stable and reliable model with the best performance to predict the edible.

```
# Plot the histogram of winning models
hist(winner, breaks = seq(0.5, length(formulas) * length(ntree_grid) + 0.5, 1),
     xlab = 'Model', ylab = 'Frequency', main = 'Frequency of Winning Models')
```




```
# Identify the most winning model
most_winning_model_index <- which.max(results_df$Wins)
most_winning_model <- results_df[most_winning_model_index, ]

cat("The model with the most wins is:\n")
```

```
## The model with the most wins is:
```

```r
cat("Formula:", most_winning_model$Formula, "\n")
```

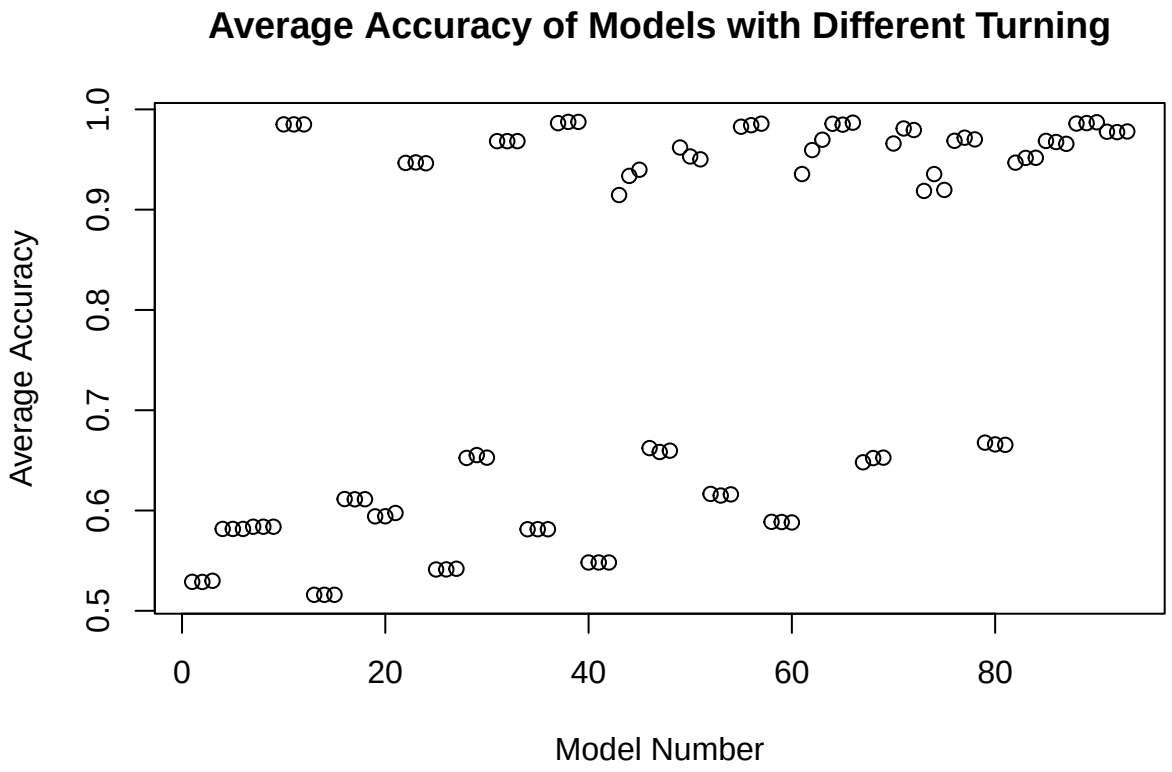
```
## Formula: Edible ~ CapShape + CapSurface + CapColor + Odor
```

```r
cat("Parameters: ntree =", most_winning_model$Ntree, "\n")
```

```
## Parameters: ntree = 50
```

Step 6: Plot the average curve of the accuracy in different models.

```r
# Plot the average accuracies for each parameter combination
plot(1:nrow(results_df), results_df$Accuracy,
     xlab = "Model Number", ylab = "Average Accuracy",
     main = "Average Accuracy of Models with Different Turning")
```



The plot of accuracy above indicates the average model accuracy in different turning models, which seem to have some models which have high accuracy, around 0.9, which suggests the models are well-suited for the classification task for predicting the mushroom edibility. However, it still has models which have lower accuracy, around 0.5, 0.6 and 0.7.

Overall, the process of the cross-validation of random forests in task 2.2 to compare the model with different turning, results will help us to receive the reliable model that has the best prediction performance from the evaluation criteria of the accuracy of predicting the edible. However, the process of cross-validation makes it need to spend more time to fit the model and needs higher performance of CPU.

2.3 Use a suitable statistical test to determine if the performance between the methods is statistically significant.

The method used to do this task is fitting the model 100 times (from the best 2 model tuning that processed in the task 2.1 and 2.2) by cross-validation of different sets of train sets and testing every time to count the accuracy of the two methods in different vectors, counting the most winning model 100 times and finding the statistical significance by using the paired t-test between 2 accuracy sets.

Step 1: Define the model formula and parameter first by selecting the decision tree model and random forest model turning which have highest accuracy count from 100 cross validation from the result of task 2.1 and 2.2 to compared, which

The first model is the best accuracy decision tree model from task 2.1.

Formula: Edible ~ CapShape + CapSurface + CapColor + Odor with turning of parameters minsplit = 10 , cp = 1e-04 , maxdepth = 10

The second model is the best accuracy random forest model from task 2.2.

Formula: Edible ~ CapShape + CapSurface + CapColor + Odor With turning of parameters ntree = 50

```
# Define the formula
formula <- as.formula("Edible ~ CapShape + CapSurface + CapColor + Odor")

# Define decision tree parameters
dt_params <- list(minsplit = 10, cp = 0.0001, maxdepth = 10)

# Define random forest parameters
rf_ntree <- 50

# Initialise vectors to store accuracy results
dt_accuracies <- numeric(100)
rf_accuracies <- numeric(100)
```

Step 2: Perform the cross-validation 100 times. In each iteration, we split the data into training sets(80%) and test sets(20%). Both decision tree and random forest models are trained on the training set, and their accuracy is evaluated on the test set.

```
# Perform cross-validation 100 times
for (i in 1:100) {
  set.seed(i)
  # Split the data into training set (80%) and test set (20%)
  trainindex <- sample(seq_len(nrow(mushroom)), size = 0.8 * nrow(mushroom))
  traindata <- mushroom[trainindex, ]
  testdata <- mushroom[-trainindex, ]

  # Make sure the levels of factors in the test set match the training set
  for (var in c("CapShape", "CapSurface", "CapColor", "Odor")) {
    if (is.factor(traindata[[var]])) {
      levels(testdata[[var]]) <- levels(traindata[[var]])
    }
  }

  # Fit the decision tree model
  dt_model <- rpart(formula = formula,
                    data = traindata,
                    method = 'class',
                    control = rpart.control(minsplit = dt_params$minsplit,
```

```
                                          cp = dt_params$cp,
                                          maxdepth = dt_params$maxdepth))

  # Predict decision tree model
  dt_predictions <- predict(dt_model, newdata = testdata, type = "class")

  # Obtain the accuracy of decision tree
  dt_conf_matrix <- confusionMatrix(dt_predictions, testdata$Edible)
  dt_accuracies[i] <- sum(diag(dt_conf_matrix$table)) / sum(dt_conf_matrix$table)

  # Fit the random forest model
  rf_model <- randomForest(formula = formula, data = traindata, ntree = rf_ntree)

  # Predict random forest model
  rf_predictions <- predict(rf_model, newdata = testdata)

  # Obtain the accuracy for random forest
  rf_conf_matrix <- confusionMatrix(rf_predictions, testdata$Edible)
  rf_accuracies[i] <- sum(diag(rf_conf_matrix$table)) / sum(rf_conf_matrix$table)
}
```

Step 3: Count the number of models winning 100 times cross-validation and suggest a more reliable result of which model has the highest accuracy to predict the edible mushroom.

```
# Count the number of wins for each model
dt_wins <- sum(dt_accuracies > rf_accuracies)
rf_wins <- sum(rf_accuracies > dt_accuracies)

cat("Decision Tree Wins:", dt_wins, "\n")
```

```
## Decision Tree Wins: 84
```

```
cat("Random Forest Wins:", rf_wins, "\n")
```

```
## Random Forest Wins: 9
```

```
cat("Decision Tree Average Accuracy:", mean(dt_accuracies), "\n")
```

```
## Decision Tree Average Accuracy: 0.9911446
```

```
cat("Random Forest Average Accuracy:", mean(rf_accuracies), "\n")
```

```
## Random Forest Average Accuracy: 0.9600369
```

The result suggests that the decision tree was turned with specific parameters of minimum split at 10 to ensure sufficient observation of each decision, the complexity parameter (cp) of 0.0001 to prune some insignificant split and help to prevent overfitting, and the maximum depth of 10 to maintain a manageable of tree size is the model that better to use to predict the edible of the mushroom based on the result of high count of accuracy over 100 times of cross-validation.

Step 4: Find the statistically significant between the decision tree model and the random forest model by performing the paired t-test.

```
# Perform paired t-test
t_test_result <- t.test(dt_accuracies, rf_accuracies, paired = TRUE)

cat("Paired t-test results:\n")
```

```
## Paired t-test results:
```

```r
print(t_test_result)
```

```
##
##  Paired t-test
##
## data:  dt_accuracies and rf_accuracies
## t = 8.5419, df = 99, p-value = 1.634e-13
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  0.02388165 0.03833374
## sample estimates:
## mean difference
##      0.03110769
```

The paired t-test results indicate a significant difference in accuracy between the decision tree and random forest models, which rejects the null hypothesis that the accuracy of the decision tree and random forest models are similar from the confident interval of 95%, not including the 0 and the mean accuracy of decision tree model is higher than the random forest model which suggests the performance of the decision tree models is better. The test statistic (t) is 8.5419 with 99 degrees of freedom (df), and the p-value is a very small value, which might indicate a highly significant difference.

Overall, the cross-validation compared model by counting the model with higher accuracy and the statistical significance found from the paired t-test suggests that the model of decision tree with the formula: Edible ~ CapShape + CapSurface + CapColor + Odor and turned of parameters minsplit = 10 , cp = 1e-04 , maxdepth = 10 could be the best model to predicting the edible mushroom which the high accuracy around 99.11%.