Decision Trees

Exercise 01

Consider the following database composed of four predictor attributes (Outlook, Temperature, Humidity, and Wind) and the Output attribute for classification.

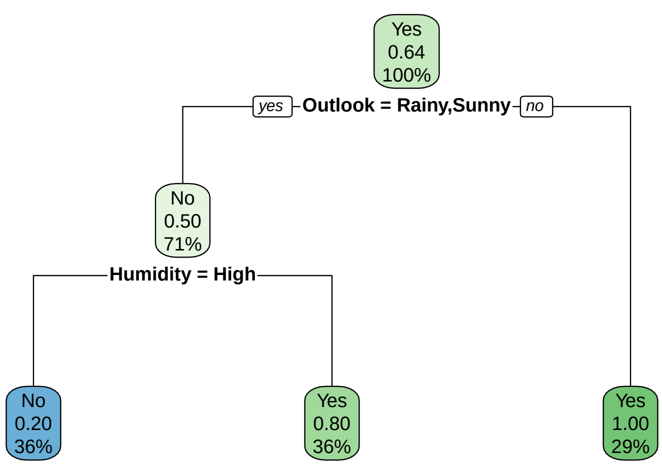
Outlook	Temperature	Humidity	Wind	Output
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Rainy	Cold	Normal	True	No
Sunny	Warm	High	False	No
Rainy	Warm	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Warm	High	False	Yes
Rainy	Cold	Normal	False	Yes
Overcast	Cold	Normal	True	Yes
Sunny	Cold	Normal	False	Yes
Rainy	Warm	Normal	False	Yes
Sunny	Warm	Normal	True	Yes
Overcast	Warm	High	True	Yes
Overcast	Hot	Normal	False	Yes

Build a decision tree using the CART algorithm and selecting features based on the Gini Impurity.

```r
library("rpart.plot")
library("dplyr")
library("tidymodels")

# Create the dataset
df <- tibble( Outlook = c("Sunny", "Sunny", "Rainy",
                          "Sunny", "Rainy", "Overcast", "Rainy",
                          "Rainy", "Overcast", "Sunny", "Rainy",
                          "Sunny", "Overcast", "Overcast"),
              Temperature = c("Hot", "Hot", "Cold", "Warm", "Warm",
                              "Hot", "Warm", "Cold", "Cold", "Cold",
                              "Warm", "Warm", "Warm", "Hot"),
              Humidity = c("High", "High", "Normal", "High", "High", "High",
                           "High", "Normal", "Normal", "Normal", "Normal",
                           "Normal", "High", "Normal"),
              Windy = c("False", "True", "True", "False",
                        "True", "False", "False", "False", "True",
                        "False", "False", "True", "True", "False"),
              Output = c(rep("No", 5), rep("Yes", 9))) %>%
      mutate_if(is.character, as.factor)
df
```

```
## # A tibble: 14 x 5
##    Outlook  Temperature Humidity Windy Output
##    <fct>    <fct>       <fct>    <fct> <fct>
##  1 Sunny    Hot         High     False No
##  2 Sunny    Hot         High     True  No
##  3 Rainy    Cold        Normal   True  No
##  4 Sunny    Warm        High     False No
##  5 Rainy    Warm        High     True  No
##  6 Overcast Hot         High     False Yes
##  7 Rainy    Warm        High     False Yes
##  8 Rainy    Cold        Normal   False Yes
##  9 Overcast Cold        Normal   True  Yes
## 10 Sunny    Cold        Normal   False Yes
## 11 Rainy    Warm        Normal   False Yes
## 12 Sunny    Warm        Normal   True  Yes
## 13 Overcast Warm        High     True  Yes
## 14 Overcast Hot         Normal   False Yes
```

```r
# Create the model for the decision tree
tree <- decision_tree( min_n = 5,
cost_complexity = 0,
tree_depth = 10
) %>%
set_engine("rpart") %>%
set_mode("classification")

# Fit the tree to the data and display the result
t.fit <- tree %>% fit(Output ~.,df)

rpart.plot(t.fit$fit)
```



Exercise 02

Is it possible to improve the performance of a classifier using ensemble methods? What are the advantages of using ensemble methods? What are the main differences between Bagging and Boosting techniques?

Yes, ensemble methods can enhance the performance of a classifier by combining predictions from multiple models. This results in increased accuracy and greater robustness when compared to using a single model.

The Bagging method is particularly valuable when dealing with models that exhibit high variance, such as decision trees. In this scenario, a series of trees are trained using different subsets of the training data, selected randomly with replacement through a process known as Bootstrapping. The final prediction is obtained by averaging the outputs from these individual trees.

On the other hand, Boosting operates in an incremental manner. Similar to Bagging, it initially takes a sample from the original dataset for training. However, instead of using Bootstrap Sampling, Boosting sequentially trains new models, each one refined based on the errors of the preceding model. This iterative process ultimately yields a more robust predictor. In addition to reducing variance like Bagging, Boosting also contributes to reducing bias.

Exercise 03

Using the BostonHousing database from the mlbench library, perform a cross-validation process to choose the optimal regularization parameter α (i.e., the cost_complexity parameter). Let min_n=5 and tree_depth=10. Create a plot of the final tree.

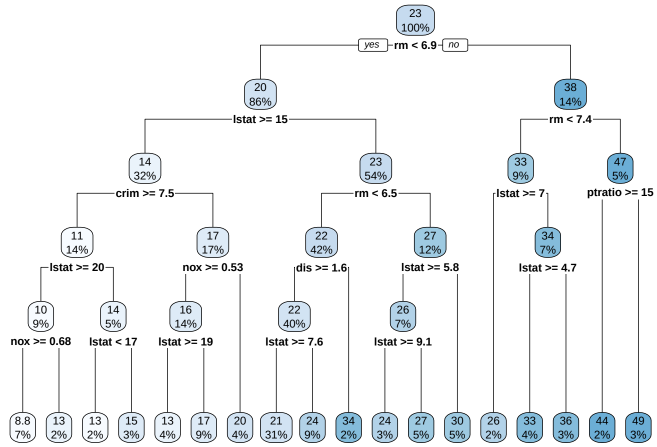
```r
library("rpart.plot")
library("dplyr")
library("tidymodels")
library("mlbench")
library("rsample")
library(tune)

# Load the data
data(BostonHousing)
df <- BostonHousing

# Split the dataset into training and testing sets
set.seed(123)
df_split <- initial_split(df)
df_train <- training(df_split)
df_test  <- testing(df_split)

# Create a model for the decision tree
tune_spec <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = 5
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")
tune_spec
```

```
## Decision Tree Model Specification (regression)
##
## Main Arguments:
##   cost_complexity = tune()
##   tree_depth = 5
##
## Computational engine: rpart
```

```r
# Set up a grid for cross-validation
tree_grid <- grid_regular(cost_complexity(),
                          levels = 5)

set.seed(234)
df_folds <- vfold_cv(df_train)

set.seed(345)
tree_wf <- workflow() %>%
  add_model(tune_spec) %>%
  add_formula(medv ~ .)

tree_res <-
  tree_wf %>%
  tune_grid(
    resamples = df_folds,
    grid = tree_grid
    )
tree_res %>%
  collect_metrics()
```

```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    0.0000000001 rmse    standard   4.47     10  0.478  Preprocesso~
##  2    0.0000000001 rsq     standard   0.743    10  0.0504 Preprocesso~
##  3    0.0000000178 rmse    standard   4.47     10  0.478  Preprocesso~
##  4    0.0000000178 rsq     standard   0.743    10  0.0504 Preprocesso~
##  5    0.00000316   rmse    standard   4.47     10  0.478  Preprocesso~
##  6    0.00000316   rsq     standard   0.743    10  0.0504 Preprocesso~
##  7    0.000562     rmse    standard   4.47     10  0.478  Preprocesso~
##  8    0.000562     rsq     standard   0.743    10  0.0504 Preprocesso~
##  9    0.1          rmse    standard   5.83     10  0.384  Preprocesso~
## 10    0.1          rsq     standard   0.583    10  0.0513 Preprocesso~
```

```r
# Select the optimal parameters based on the RMSE
best_tree <- tree_res %>%
  select_best("rmse")

# Create the final tree with optimal parameters
final_wf <-
  tree_wf %>%
  finalize_workflow(best_tree)

final_fit <-
  final_wf %>%
  last_fit(df_split)

final_tree <- extract_workflow(final_fit)

final_tree %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```



Exercise 04

Using the BostonHousing database once again, fit the RandomForests and Boosting Trees models. Use the rand_forest and boost_tree commands to define the models, using the randomForest and xgboost engines for each of the algorithms, respectively.

For RandomForest, set a high value for the trees parameter (e.g., 500), set min_n = 5, and set the mtry parameter to $\sqrt{}$d, where d represents the number of features.

For Boosting Trees, use the same number of trees as in RandomForest for the trees parameter and a learning rate of learn_rate = 0.01. Leave the other parameters at their default values.

Perform cross-validation to estimate the Root Mean Square Error (RMSE) and discuss the results, taking into account the std_err of your results.

```r
library("rpart.plot")
library("dplyr")
library("tidymodels")
library("mlbench")
library("rsample")
library(tune)

## Random Forest

# Load the data
data(BostonHousing)
df <- BostonHousing

# Split the dataset into training and testing sets
set.seed(123)
df_split <- initial_split(df)
df_train <- training(df_split)
df_test  <- testing(df_split)

# Create a model for the Random Forest
tune_spec <-
  rand_forest(
```

```r
    trees = 500 ,
    min_n = 5,
    mtry = sqrt(14)
  ) %>%
  set_engine("randomForest") %>%
  set_mode("regression")

# Set up the folds for cross-validation
set.seed(234)
df_folds <- vfold_cv(df_train)

set.seed(345)
tree_wf <- workflow() %>%
  add_model(tune_spec) %>%
  add_formula(medv ~ .)

# Apply cross-validation
tree_res <-
  tree_wf %>%
  tune_grid(resamples = df_folds)

# Collect metrics
metrics <- tree_res %>%
  collect_metrics()
print(metrics)
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   3.37     10  0.351  Preprocessor1_Model1
## 2 rsq     standard   0.863    10  0.0336 Preprocessor1_Model1
```

```r
library("rpart.plot")
library("dplyr")
library("tidymodels")
library("mlbench")
library("rsample")
library(tune)

## Boosting Tree

# Load the data
data(BostonHousing)
df <- BostonHousing

# Split the dataset into training and testing sets
set.seed(123)
df_split <- initial_split(df)
df_train <- training(df_split)
df_test  <- testing(df_split)

# Create a model for the boost tree
tune_spec <-
```

```r
  boost_tree(
    trees = 500 ,
    learn_rate = 0.01
  ) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

# Set up the folds for cross-validation
set.seed(234)
df_folds <- vfold_cv(df_train)

set.seed(345)
tree_wf <- workflow() %>%
  add_model(tune_spec) %>%
  add_formula(medv ~ .)

# Apply cross-validation
tree_res <-
  tree_wf %>%
  tune_grid(resamples = df_folds)

# Collect metrics
metrics <- tree_res %>%
  collect_metrics()
print(metrics)
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   3.08     10  0.214  Preprocessor1_Model1
## 2 rsq     standard   0.879    10  0.0240 Preprocessor1_Model1
```

Exercise 05

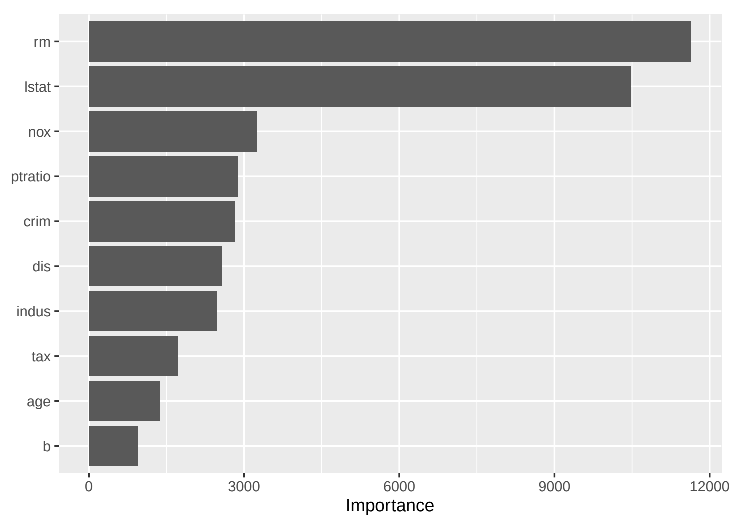
We have seen that Random Forests, like Bagging-based methods, can be used to identify the most important features. Explain how these features are identified and calculate the most important variables within the BostonHousing dataset. Identify the three most important features according to this criterion.

```r
library("rpart.plot")
library("dplyr")
library("tidymodels")
library("mlbench")
library("rsample")
library("vip")

# Load the data
data(BostonHousing)
df <- BostonHousing

# Split the dataset into training and testing sets
set.seed(123)
df_split <- initial_split(df)
```

```r
df_train <- training(df_split)
df_test  <- testing(df_split)

# Create a model for the random forest
tune_spec <-
  rand_forest(
    trees = 500 ,
    min_n = 5,
    mtry = sqrt(14),
  ) %>%
  set_engine("randomForest") %>%
  set_mode("regression")
tune_spec
```

```
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = sqrt(14)
##   trees = 500
##   min_n = 5
##
## Computational engine: randomForest
```

```r
t.fit <- tune_spec %>% fit(medv ~.,df)

# Check the most relevant features
vip(t.fit)
```



The most relevant variables can be obtained by ranking the predictors with the highest average reduction of RSS (Residual Sum of Squares) or Gini index across the built trees. In this case, the top 3 predictor variables are those with the highest values along the X-axis of the above graph, namely rm, lstat, and nox.