Classification of Poisonous Mushrooms

Zeynep Afra Sezer - Beste Ünal

This dataset is a cleaned up version of the original Mushroom Dataset to identify which features are most indicative of a poisonous mushroom. The Mushroom Dataset includes 8124 mushrooms of various species, both edible and poisonous. The data consists of attributes such as class, hat shape, hat surface, hat color, bruises, odor, gill attachment, gill spacing, gill size, gill color, stem shape, stem root, above-ring stem surface, below-ring stem surface, above-ring stem color, below-ring stem color, cover type, cover color, number of rings, ring type, spore print color, population, habitat.

In the dataset we are interested in, the Mushroom Dataset, we used Modal imputation to fill in missing data and one-shot coding to represent categorical data in binary form. Thus, each categorical value is transformed into a set of binary variables where the corresponding category is 1 and all others are 0. The data were then cleaned using various techniques such as z-score normalization to scale the mean to 0 and standard deviation to 1, and feature selection to improve the performance of the model and make it run faster.

Packages

```r
# install.packages("DALEX")
# install.packages("caret")
# install.packages("ROCR")
# install.packages("readr")
# install.packages("yardstick")
# install.packages("rsample") #helps us splitting the dataset.
# install.packages("tidymodels") # it helps us training models.
# install.packages("parsnip") # for model fitting
# install.packages("tune") # for model tuning

library(DALEX)
library(caret)
library(ROCR)
library(readr)
library(yardstick)
library(rsample)
library(tidymodels)
library(parsnip)
library(tune)
```

The data set is a csv file, so we can use readr package to import the data set. After that, we can assign the data set as "mushroom".

```r
mushroom <- read_csv("mushroom_cleaned.csv")
str(mushroom)
```

```
## spc_tbl_ [54,035 x 9] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ cap-diameter   : num [1:54035] 1372 1461 1371 1261 1305 ...
##  $ cap-shape      : num [1:54035] 2 2 2 6 6 6 2 6 6 6 ...
##  $ gill-attachment: num [1:54035] 2 2 2 2 2 2 2 2 2 2 ...
##  $ gill-color     : num [1:54035] 10 10 10 10 10 10 10 10 10 10 ...
##  $ stem-height    : num [1:54035] 3.81 3.81 3.61 3.79 3.71 ...
##  $ stem-width     : num [1:54035] 1545 1557 1566 1566 1464 ...
##  $ stem-color     : num [1:54035] 11 11 11 11 11 11 11 11 11 11 ...
##  $ season         : num [1:54035] 1.804 1.804 1.804 1.804 0.943 ...
##  $ class          : num [1:54035] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `cap-diameter` = col_double(),
##   ..   `cap-shape` = col_double(),
##   ..   `gill-attachment` = col_double(),
##   ..   `gill-color` = col_double(),
##   ..   `stem-height` = col_double(),
##   ..   `stem-width` = col_double(),
##   ..   `stem-color` = col_double(),
##   ..   season = col_double(),
##   ..   class = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

The Mushroom data set consists of 54,035 observations and 9 variables. The variables and their contents are as follows:

1. cap-diameter: Diameter of the cork cap
2. cap-shape: Mushroom cap shape
3. gill-attachment: Attachment of the lamellae (thin leaves of the cork) to the cork
4. gill-color: Color of the lamellae
5. stem-height: Height of the mushroom stem
6. stem-width: Width of the mushroom stem
7. stem-color: Color of the mushroom stem
8. season: Season.
9. class: Mushroom class The Target Class contains two values - 0 or 1 - where 0 refers to edible and 1 refers to poisonous.

The dataset contains various physical properties of mushrooms and the enumerated values of these properties. These numbers represent specific properties for each mushroom sample in the dataset. In this dataset, our target variable is class, because we want to predict whether the mushrooms are poisonous or not.

```r
mushroom <- na.exclude(mushroom) #First, we have to exclude all the NA's on the data set.
```

Splitting the Data set

We have to split the data set to compute target variable. We have to split the data set as a two subset by using "sample()" function. Let's allocate 80% of the dataset as the training set and the rest will be the test set.

```r
set.seed(123)
mushroom_split <- initial_split(data = mushroom, # dataset to split
                                 prop = 0.80)     # proportion of train set
```

```
mushroom_train <- mushroom_split |> training()
mushroom_test  <- mushroom_split |> testing()
```

1.Logistic Regression

In this section we build our model using the "glm()" function. First, we need to write the model formula, which is the target variable (class), and the properties as a '.'. Second, we use the training model we specified in the previous section. Finally, we add the family distribution of the target variable. We are interested in binary logistic regression with only two outcomes, so we should set it to 'binomial'.

```
lr_model <- glm(class ~., data = mushroom_train, family = "binomial")
lr_model
```

```
##
## Call:  glm(formula = class ~ ., family = "binomial", data = mushroom_train)
##
## Coefficients:
##       (Intercept)     `cap-diameter`         `cap-shape`  `gill-attachment`
##         1.6524319         -0.0003208          -0.0905950          0.0083937
##      `gill-color`      `stem-height`        `stem-width`       `stem-color`
##        -0.0086438          0.7377154          -0.0003459         -0.0677267
##            season
##        -0.4783652
##
## Degrees of Freedom: 43227 Total (i.e. Null);  43219 Residual
## Null Deviance:       59480
## Residual Deviance: 54970     AIC: 54990
```

According to the model, attributes such as stem height, cap shape and stem color appear to be important in the logistic regression. However, the positive intercept value suggests that it may not be an important attribute to consider in classification.

In addition, we can say that our model runs on a total of 43227 data points and estimates 8 parameters. The null bias indicates that the model tries to fit the data using only the intercept term, while the residual bias shows how well the model actually fits the data. The AIC value balances the fit and complexity of the model, in this case 54990 indicates that the model fits well but has a certain level of complexity.

```
summary(lr_model) #to see the output in detail
```

```
##
## Call:
## glm(formula = class ~ ., family = "binomial", data = mushroom_train)
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)        1.652e+00  5.218e-02  31.668  < 2e-16 ***
## `cap-diameter`    -3.208e-04  5.205e-05  -6.164 7.09e-10 ***
## `cap-shape`       -9.059e-02  4.917e-03 -18.424  < 2e-16 ***
## `gill-attachment`  8.394e-03  4.771e-03   1.759  0.07849 .
## `gill-color`      -8.644e-03  3.332e-03  -2.594  0.00948 **
## `stem-height`      7.377e-01  1.761e-02  41.883  < 2e-16 ***
## `stem-width`      -3.459e-04  2.412e-05 -14.341  < 2e-16 ***
## `stem-color`      -6.773e-02  3.249e-03 -20.844  < 2e-16 ***
## season            -4.784e-01  3.476e-02 -13.764  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 59479  on 43227  degrees of freedom
## Residual deviance: 54971  on 43219  degrees of freedom
## AIC: 54989
##
## Number of Fisher Scoring iterations: 4
```

When we look at the results:

Null Deviance: Indicates how well the model explains the data when only the intercept is included. A value of 59479 represents the amount of variance explained by the intercept alone. Lower null deviance values generally indicate a better fit of the model to the data.

Residual Deviance: Indicates how well the model fits the actual data after considering the independent variables. A value of 54971 indicates that the model fits the data well.

AIC (Akaike Information Criterion): Balances the model's goodness of fit with its complexity. Lower AIC values indicate a better fit with less complexity. A value of 54989 suggests that the model is fitting well but has a certain level of complexity.

In summary, these values help evaluate how well your model fits the data and its complexity. Lower deviance values and AIC indicate better model performance.

Model Performance:

We can check the model performance of the model on test set. For model performance, we have to compute the predicted value of target variable on test set. We should not for get to exclude the target variable on test set. Predicting the values, we can check the first six values using "head()" function.

```
predicted_probs <- predict(lr_model,mushroom_test[,-9], type= "response")
head(predicted_probs)
```

```
##         1         2         3         4         5         6
## 0.8358573 0.7822819 0.5147352 0.4642393 0.3560796 0.4294214
```

We should transform these probabilities to classes by using "ifelse()" function. We set the condition like that: If there is greater than 0.5, it means "1". If it is smaller than 0.05, it assigned "0".

```
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)
head(predicted_classes)
```

```
## 1 2 3 4 5 6
## 1 1 1 0 0 0
```

We can create confusion matrix using the metrics. We assign the positive and negative classes as a 1 and 0.

Confusion Matrix:

```r
confusionMatrix(table(ifelse(mushroom_test$class == "1", "1", "0"),
                      predicted_classes),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##    predicted_classes
##        0    1
##   0 2722 2221
##   1 1699 4165
##
##                Accuracy : 0.6373
##                  95% CI : (0.6281, 0.6463)
##     No Information Rate : 0.5909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2631
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6522
##             Specificity : 0.6157
##          Pos Pred Value : 0.7103
##          Neg Pred Value : 0.5507
##              Prevalence : 0.5909
##          Detection Rate : 0.3854
##    Detection Prevalence : 0.5426
##       Balanced Accuracy : 0.6340
##
##        'Positive' Class : 1
##
```

The overall accuracy of the model is 63.73%, meaning that it correctly classified 63.73% of the instances. The model correctly identifies 65.22% of the actual poisonous mushrooms. The model correctly identifies 61.57% of the actual non-poisonous mushrooms. When the model predicts a mushroom as poisonous, it is correct 71.03% of the time. When the model predicts a mushroom as non-poisonous, it is correct 55.07% of the time.

Balanced Accuracy: 0.6340. The average of sensitivity and specificity, providing a balanced measure of model performance.

2.Decision Tree

We need some packages for the decision tree.

```r
# install.packages("rpart.plot") # it helps us visualizing the decision tree.
# install.packages("recipes")

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

The purpose of decision trees is to partition the trained data into homogeneous subgroups. Furthermore, decision trees are often used to model non-linear relationships. They break down features in detail by dividing them into small parts.

Step 1- Defining model specification:

```
dt_model <- decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification")
```

Step 2- Model training:

```
mushroom_train$class <- as.factor(mushroom_train$class)

dt_mushroom <- dt_model |>
 fit(class ~., data = mushroom_train)
dt_mushroom
```

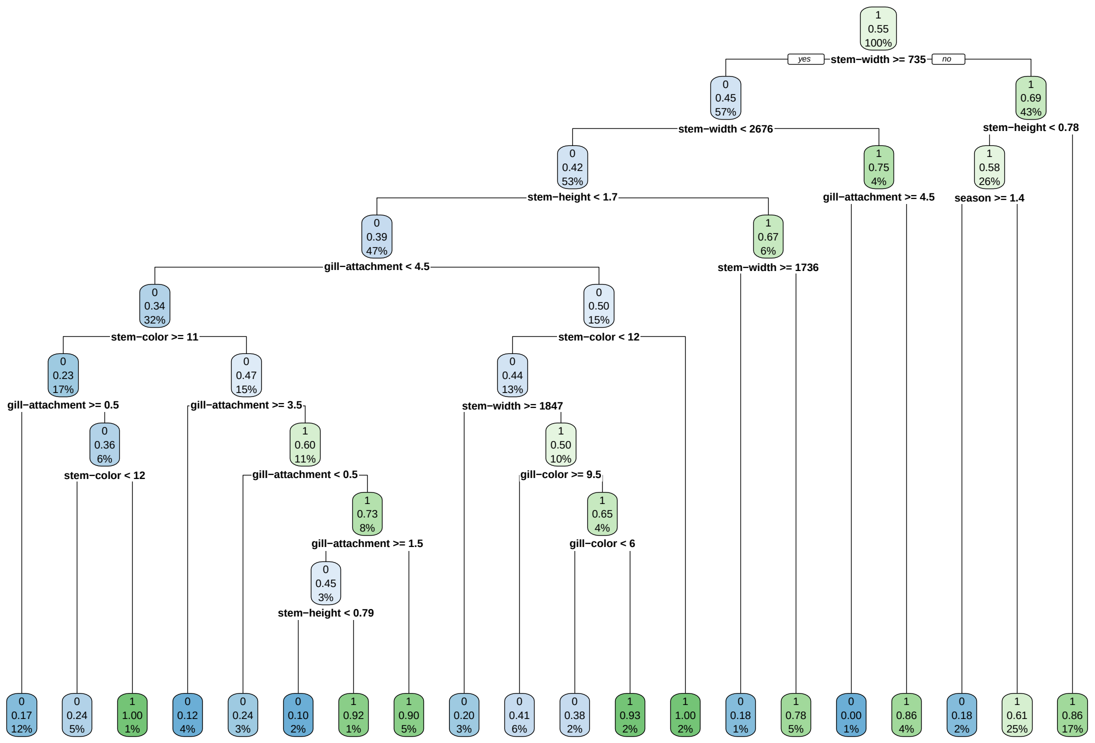
```
## parsnip model object
##
## n= 43228
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 43228 19417 1 (0.44917646 0.55082354)
##     2) stem-width>=734.5 24582 10989 0 (0.55296558 0.44703442)
##       4) stem-width< 2675.5 22793  9650 0 (0.57662440 0.42337560)
##         8) stem-height< 1.716508 20171  7894 0 (0.60864608 0.39135392)
##          16) gill-attachment< 4.5 13822  4735 0 (0.65743018 0.34256982)
##            32) stem-color>=10.5 7522  1746 0 (0.76788088 0.23211912)
##              64) gill-attachment>=0.5 5067   852 0 (0.83185317 0.16814683) *
##              65) gill-attachment< 0.5 2455   894 0 (0.63584521 0.36415479)
##               130) stem-color< 11.5 2053   492 0 (0.76035071 0.23964929) *
##               131) stem-color>=11.5 402     0 1 (0.00000000 1.00000000) *
##            33) stem-color< 10.5 6300  2989 0 (0.52555556 0.47444444)
##              66) gill-attachment>=3.5 1596   190 0 (0.88095238 0.11904762) *
##              67) gill-attachment< 3.5 4704  1905 1 (0.40497449 0.59502551)
##               134) gill-attachment< 0.5 1258   297 0 (0.76391097 0.23608903) *
##               135) gill-attachment>=0.5 3446   944 1 (0.27394080 0.72605920)
##                 270) gill-attachment>=1.5 1358   617 0 (0.54565538 0.45434462)
##                   540) stem-height< 0.794234 776    80 0 (0.89690722 0.10309278) *
##                   541) stem-height>=0.794234 582    45 1 (0.07731959 0.92268041) *
##                 271) gill-attachment< 1.5 2088   203 1 (0.09722222 0.90277778) *
##          17) gill-attachment>=4.5 6349  3159 0 (0.50244133 0.49755867)
##            34) stem-color< 11.5 5693  2503 0 (0.56033726 0.43966274)
##              68) stem-width>=1846.5 1229   250 0 (0.79658259 0.20341741) *
##              69) stem-width< 1846.5 4464  2211 1 (0.49529570 0.50470430)
##               138) gill-color>=9.5 2709  1107 0 (0.59136213 0.40863787) *
##               139) gill-color< 9.5 1755   609 1 (0.34700855 0.65299145)
##                 278) gill-color< 6 870   327 0 (0.62413793 0.37586207) *
##                 279) gill-color>=6 885    66 1 (0.07457627 0.92542373) *
##            35) stem-color>=11.5 656     0 1 (0.00000000 1.00000000) *
##         9) stem-height>=1.716508 2622   866 1 (0.33028223 0.66971777)
##          18) stem-width>=1735.5 501    92 0 (0.81636727 0.18363273) *
##          19) stem-width< 1735.5 2121   457 1 (0.21546440 0.78453560) *
##       5) stem-width>=2675.5 1789   450 1 (0.25153717 0.74846283)
##        10) gill-attachment>=4.5 230     0 0 (1.00000000 0.00000000) *
##        11) gill-attachment< 4.5 1559   220 1 (0.14111610 0.85888390) *
##     3) stem-width< 734.5 18646  5824 1 (0.31234581 0.68765419)
##       6) stem-height< 0.7759029 11442  4785 1 (0.41819612 0.58180388)
##        12) season>=1.373734 766   141 0 (0.81592689 0.18407311) *
##        13) season< 1.373734 10676  4160 1 (0.38965905 0.61034095) *
##       7) stem-height>=0.7759029 7204  1039 1 (0.14422543 0.85577457) *
```

We created and trained a decision tree classification model called dt_mushroom, which will be used to classify mushroom species based on data from the mushroom_train dataset.

Step 3- Visualizing the decision tree:

Now let's visualize our decision tree using the "rpart" function.

```r
rpart.plot(dt_mushroom$fit)
```



In the decision tree structure, it starts with the root at the beginning and branches out the following parts in turn. Here we can also see the features in the dataset and how which feature was obtained. We can say that the root node is called cp at the top and is the starting point of the decision tree. The next node is called an internal/sub-node. Branches help to connect these nodes together. Finally, the last nodes are called leaf/terminal nodes and they are the endpoints of the decision tree.

```r
mushrom_predictions <- dt_mushroom|>
 predict(new_data = mushroom_test)
 mushrom_predictions
```

```
## # A tibble: 10,807 x 1
##    .pred_class
##    <fct>
##  1 1
##  2 1
##  3 0
##  4 0
##  5 0
##  6 0
##  7 0
##  8 0
##  9 0
## 10 0
## # i 10,797 more rows
```

A vector called mushrom_predictions was created, which contains the classification predictions made for each sample in the mushroom_test dataset. These predictions will be used to evaluate the performance of the model on the test dataset.

```r
dt_mushroom |>
 predict(new_data = mushroom_test,
 type = "prob")
```

```
## # A tibble: 10,807 x 2
##    .pred_0 .pred_1
##      <dbl>   <dbl>
##  1   0.215   0.785
##  2   0.215   0.785
##  3   0.832   0.168
##  4   0.832   0.168
##  5   0.832   0.168
##  6   0.832   0.168
##  7   0.832   0.168
##  8   0.832   0.168
##  9   0.832   0.168
## 10   0.832   0.168
## # i 10,797 more rows
```

This code will return a probability value for each class for each test instance. For example, if there are two classes and the probabilities for a sample are 0.3 and 0.7, the probability that this sample belongs to the first class is 0.3 and the probability that it belongs to the second class is 0.7. Such outputs are important for assessing how reliable the predictions of the classification model are.

Step 4- Evaluating model performance:

```r
mushroom_results <- tibble(predicted = mushrom_predictions$.pred_class,
                              actual = mushroom_test$class)
```

Both columns must be of factor type.

```r
mushroom_results$actual <- as.factor(mushroom_results$actual)
mushroom_results$predicted <- as.factor(mushroom_results$predicted)

mushroom_results|> conf_mat(truth = actual, estimate = predicted)
```

```
##            Truth
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
```

The True Negatives (TN) value indicates that there were 3423 samples that the model correctly predicted as 0 (edible). False Positives (FP) value indicates that there were 1520 samples that the model predicted as 1 (poisonous) but were actually 0 (edible). False Negatives (FN) value indicates that there were 971 samples that the model predicted 0 (edible) but were actually 1 (poisonous). The True Positives (TP) value indicates that there were 4893 instances where the model correctly predicted 1 (poisonous).

```r
 mushroom_results |> accuracy(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.770
```

```r
mushroom_results |> sens(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.692
```

```r
mushroom_results |> spec(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.834
```

The accuracy of this model is about 77%. For this value we can say that the model predicts mostly correctly. Sensitivity; the model correctly predicts the positive class (1: poisonous) about 83% of the time, indicating that the model is quite good at detecting the positive class. Specificity; the model correctly predicts the negative class (0: edible) at 69%, indicating that the model is slightly less successful in detecting the negative class. The positive predictive value is about 76% of the model's positive predictions are correct, indicating that the model's positive predictions are mostly correct.

We can say that the model has the ability to predict the positive class well and its overall accuracy is quite high. However, the rate of detecting the negative class is slightly lower. This means that the number of false positives is high.

Imbalance Control Interpretation

Here, we use the “bal_accuracy” function in the “yardstick” package to calculate the balanced accuracy metric. This metric is used to evaluate the performance of classification models, while providing more reliable results when there is class imbalance. Balance-corrected accuracy is calculated. This metric evaluates the accuracy of the model considering the imbalance between classes. It can be used for more balanced classification results.

```
mushroom_results |> bal_accuracy(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric      .estimator .estimate
##   <chr>        <chr>          <dbl>
## 1 bal_accuracy binary         0.763
```

Balanced accuracy takes a value between 0 and 1, the closer to 1 the better the model performs. In this case, a value of 0.76 indicates that the model performs quite well when class imbalance is taken into account.

HYPERPARAMETER ADJUSTMENT IN DECISION TREE MODELS

A hyperparameter helps us to find the most appropriate model. For a decision tree model, hyperparameters can include values such as maximum depth, minimum sampling fraction, maximum number of features and Decoupling criterion. These hyperparameters affect the complexity and generalization ability of the model. Well-tuned hyperparameters can help the model perform better and generalize better.

-minsplit: This hyperparameter determines the minimum number of observations required to split a node. If the number of observations in a node is below this value, the node is not split. This can help reduce the risk of overfitting.

-minbucket: Sets the minimum number of observations required for a leaf node to be created. If a node has a number of observations below this value, it becomes a leaf and the split stops.

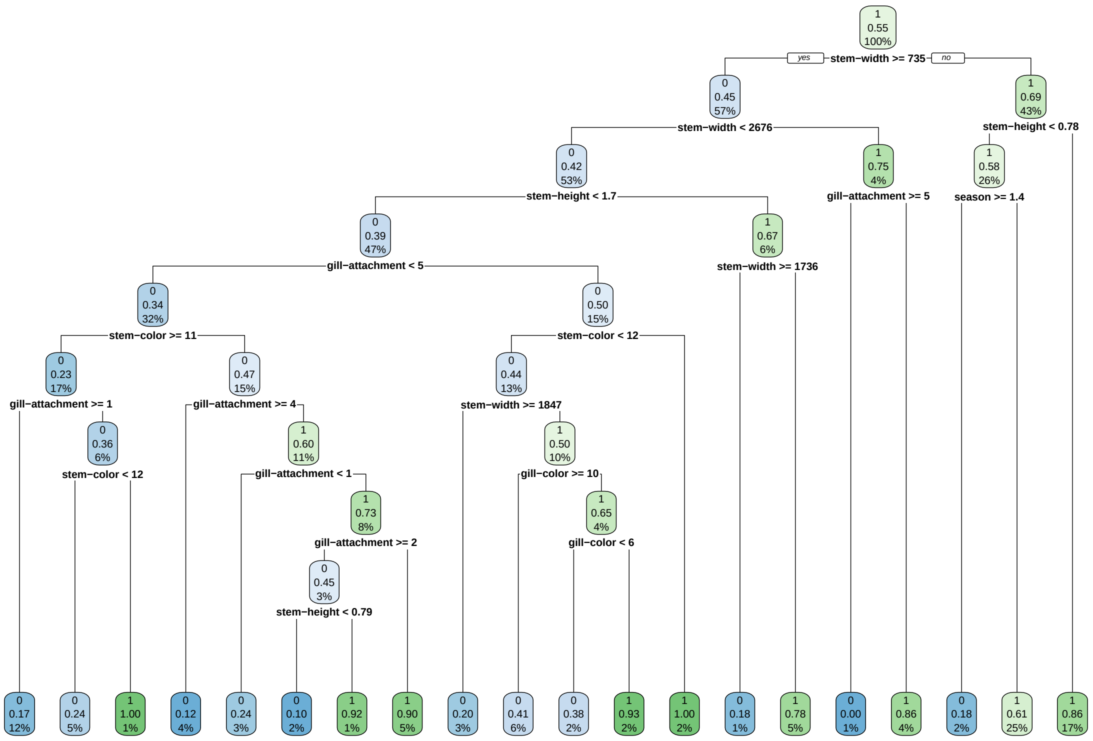
-max depth: This hyperparameter determines the maximum depth of the decision tree. The tree stops growing when it reaches the specified maximum depth or when an unseen node remains. This can help prevent overfitting of the model.

- cp: This hyperparameter is an important parameter that controls the complexity of the decision tree. High cp values lead to simpler tree structures and fewer branching nodes. This can help prevent overfitting and make generalization better.

Training a vanilla decision tree

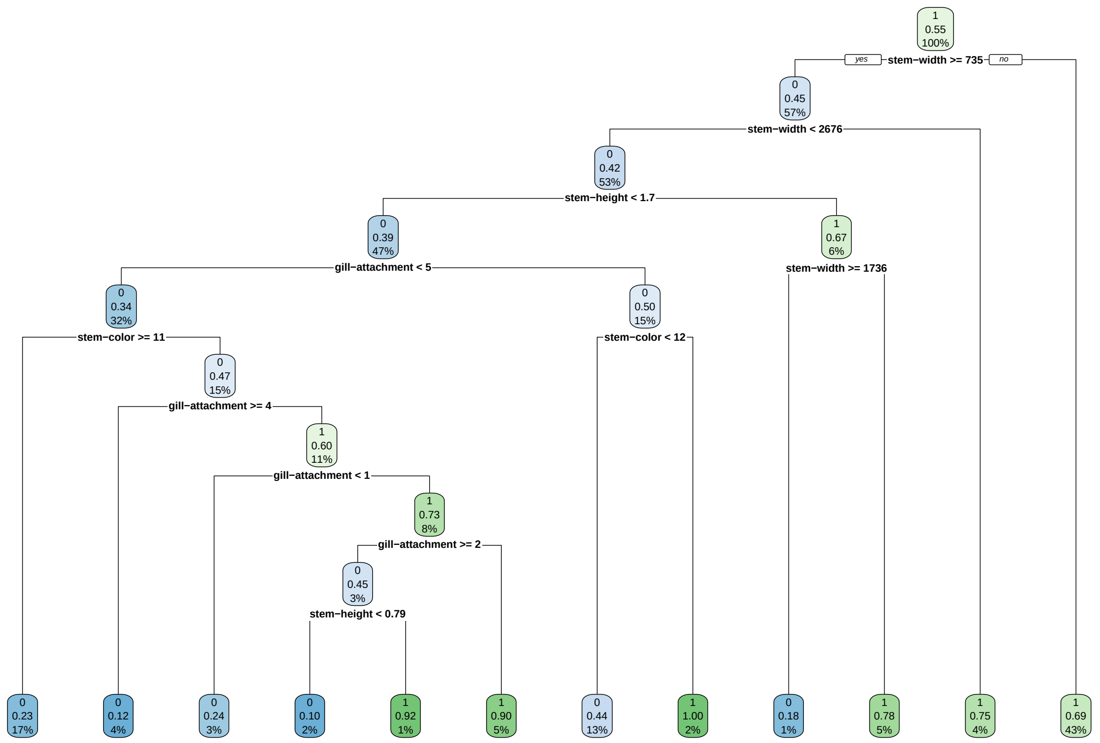
The vanilla decision tree is a decision tree in the simplest form, usually created without the use of any hyperparameter settings or complex optimization techniques. This is based on the direct adaptation of the model to the data set and making decisions based on the values of the characteristics.

```
vanilla_dt <- rpart( class ~ .,
 data = mushroom_train,
 method = "class")
 rpart.plot(vanilla_dt)
```



Training a less deeper decision tree by tuning cp

```
less_dt1 <- rpart(class ~ .,
 data = mushroom_train,
 method = "class",
 cp = 0.015)
 rpart.plot(less_dt1)
```



Compare the performance of the vanilla dt and less deeper dt1

```r
mushroom_test$class <- as.factor(mushroom_test$class)

# performance metrics of the vanilla dt
 vanilla_preds <- predict(vanilla_dt, mushroom_test, type = "class")
 confusionMatrix(vanilla_preds,
 mushroom_test$class,
 positive = "1")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0    1
##           0 3423  971
##           1 1520 4893
##
##                Accuracy : 0.7695
##                  95% CI : (0.7614, 0.7774)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

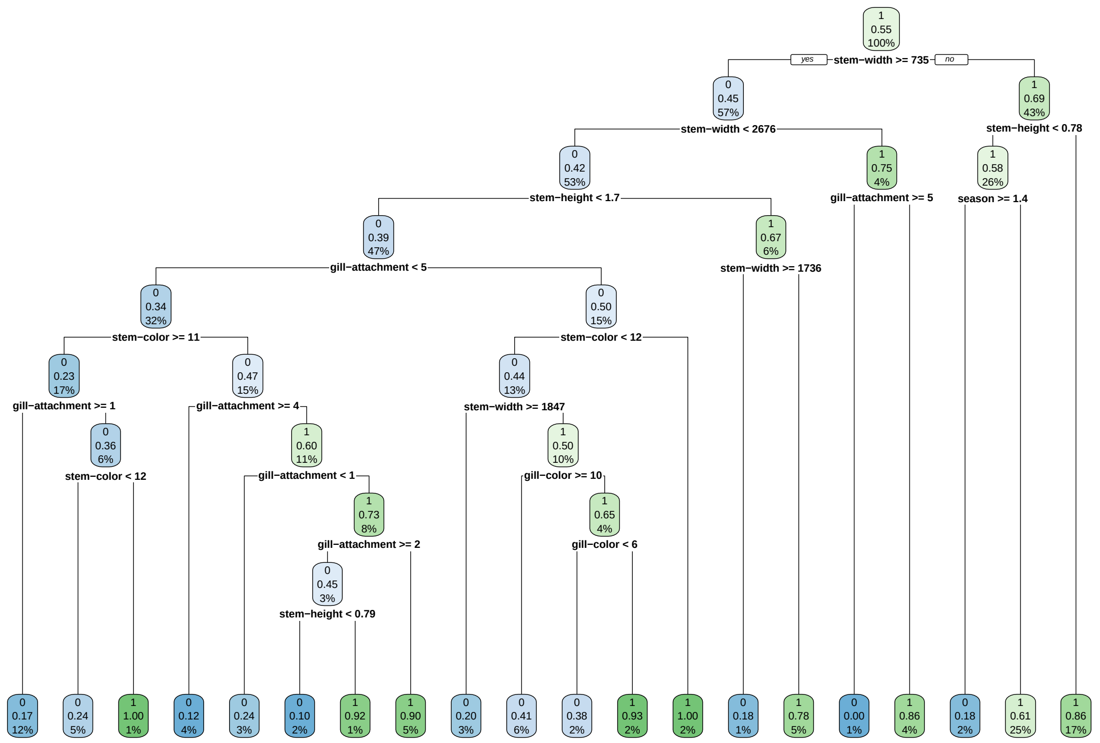
```
##             Sensitivity : 0.8344
##             Specificity : 0.6925
##          Pos Pred Value : 0.7630
##          Neg Pred Value : 0.7790
##              Prevalence : 0.5426
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5934
##       Balanced Accuracy : 0.7635
##
##        'Positive' Class : 1
##
```

```
# performance metrics of the less deeper dt
less_preds1 <- predict(less_dt1, mushroom_test, type = "class")
confusionMatrix(less_preds1,
mushroom_test$class,
positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3231 1248
##          1 1712 4616
##
##                Accuracy : 0.7261
##                  95% CI : (0.7176, 0.7345)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4441
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7872
##             Specificity : 0.6537
##          Pos Pred Value : 0.7295
##          Neg Pred Value : 0.7214
##              Prevalence : 0.5426
##          Detection Rate : 0.4271
##    Detection Prevalence : 0.5855
##       Balanced Accuracy : 0.7204
##
##        'Positive' Class : 1
##
```

Training a less deeper decision tree by tuning minsplit

```
less_dt2 <- rpart(class ~ .,
data = mushroom_train,
method = "class",
minsplit = 30)
rpart.plot(less_dt2)
```



Compare the performance of the vanilla dt and less deeper dt2

```
# performance metrics of the vanilla dt
confusionMatrix(vanilla_preds,
mushroom_test$class,
positive = "1")
```

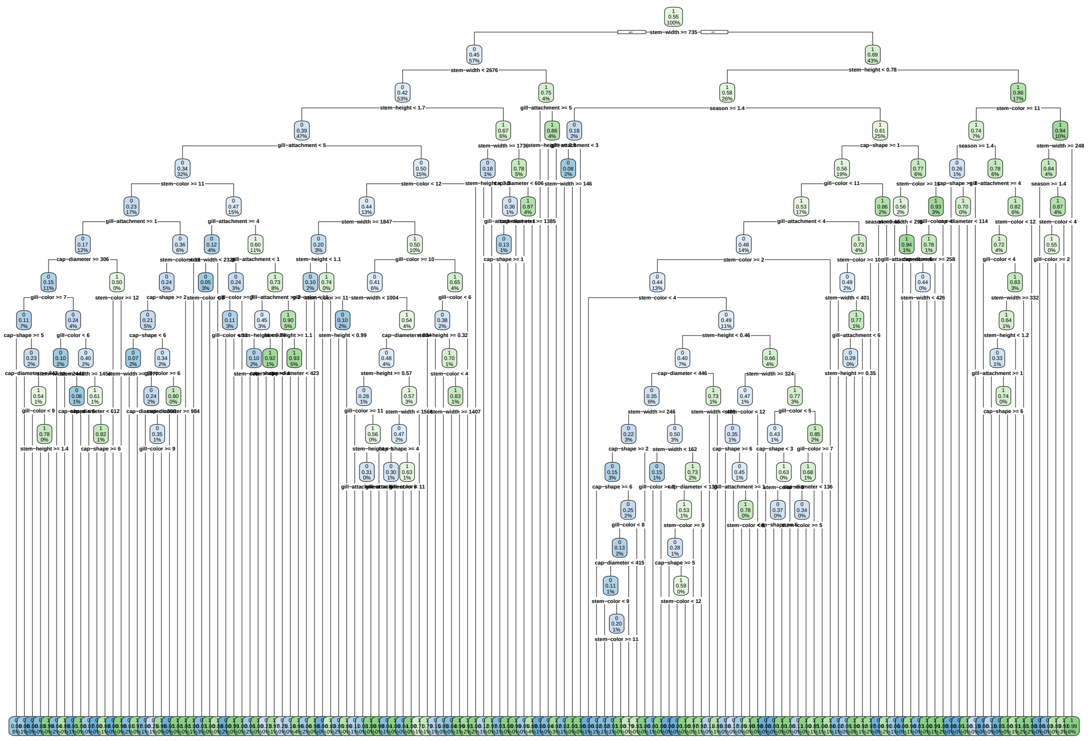
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
##
##                Accuracy : 0.7695
##                  95% CI : (0.7614, 0.7774)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8344
##             Specificity : 0.6925
##          Pos Pred Value : 0.7630
##          Neg Pred Value : 0.7790
##              Prevalence : 0.5426
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5934
##       Balanced Accuracy : 0.7635
##
##        'Positive' Class : 1
##
```

```
# performance metrics of the less deeper dt2
 less_preds2 <- predict(less_dt2, mushroom_test, type = "class")
 confusionMatrix(less_preds2,
 mushroom_test$class,
 positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
##
##                Accuracy : 0.7695
##                  95% CI : (0.7614, 0.7774)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8344
##             Specificity : 0.6925
##          Pos Pred Value : 0.7630
##          Neg Pred Value : 0.7790
##              Prevalence : 0.5426
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5934
##       Balanced Accuracy : 0.7635
##
##        'Positive' Class : 1
##
```

Training a deeper decision tree by tuning cp

```
deeper_dt <- rpart(class ~ .,
 data = mushroom_train,
 method = "class",
 cp = 0.001)
 rpart.plot(deeper_dt)
```



Compare the performance of the vanilla dt, less deeper dt2, deeper tree

```
# performance metrics of the vanilla dt
 confusionMatrix(vanilla_preds,
 mushroom_test$class,
 positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
##
##                Accuracy : 0.7695
##                  95% CI : (0.7614, 0.7774)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8344
##             Specificity : 0.6925
##          Pos Pred Value : 0.7630
##          Neg Pred Value : 0.7790
##              Prevalence : 0.5426
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5934
##       Balanced Accuracy : 0.7635
##
##        'Positive' Class : 1
##
```

```
# performance metrics of the less deeper dt2
 confusionMatrix(less_preds2,
 mushroom_test$class,
 positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
##
##                Accuracy : 0.7695
##                  95% CI : (0.7614, 0.7774)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8344
##             Specificity : 0.6925
##          Pos Pred Value : 0.7630
##          Neg Pred Value : 0.7790
##              Prevalence : 0.5426
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5934
##       Balanced Accuracy : 0.7635
##
##        'Positive' Class : 1
##
```

```
# performance metrics of the deeper tree
 deeper_preds <- predict(deeper_dt, mushroom_test, type = "class")
 confusionMatrix(deeper_preds,
 mushroom_test$class,
 positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4583  301
##          1  360 5563
##
##                Accuracy : 0.9388
##                  95% CI : (0.9342, 0.9433)
##     No Information Rate : 0.5426
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8767
##
##  Mcnemar's Test P-Value : 0.02407
##
##             Sensitivity : 0.9487
##             Specificity : 0.9272
##          Pos Pred Value : 0.9392
##          Neg Pred Value : 0.9384
##              Prevalence : 0.5426
##          Detection Rate : 0.5148
##    Detection Prevalence : 0.5481
##       Balanced Accuracy : 0.9379
##
##        'Positive' Class : 1
##
```

3.Bagging Trees

Bagging is an ensemble method that combines many decision trees together to create a more powerful model. First, let's install the randomForest package, which is a suitable package for building a bagging model.

```r
# install.packages("randomForest")
# install.packages("ranger")

library(randomForest)
library(ranger)
```

```r
set.seed(123)

# To manually correct column names
colnames(mushroom_train) <- make.names(colnames(mushroom_train), unique = TRUE)

# Let's check the column names again
colnames(mushroom_train)
```

```
## [1] "cap.diameter"    "cap.shape"       "gill.attachment" "gill.color"
## [5] "stem.height"     "stem.width"      "stem.color"      "season"
## [9] "class"
```

```r
# Let's try to train the model again
trained_bt <- ranger(class ~ ., data = mushroom_train, mtry = 3)
trained_bt
```

```
## Ranger result
##
```

```
## Call:
##  ranger(class ~ ., data = mushroom_train, mtry = 3)
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      43228
## Number of independent variables:  8
## Mtry:                             3
## Target node size:                 1
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error:             1.00 %
```

In this model, 500 trees and 5000 observations of the model were used. There are 8 independent variables in the dataset. The number of random variables evaluated to find the best split at each node is mtry:3. The minimum target (leaf) size at the leaf nodes of each tree is 5. Smaller leaf sizes allow the model to learn more, but can also lead to overlearning. Splitrule: variance indicates that splits are based on variance, i.e. variance reduction is maximized for each split.

These results indicate that the model performs quite well. Particularly, the OOB prediction error of 1.02% is a very low value, suggesting that the model generally makes accurate predictions. Additionally, it can be seen that 8 independent variables are used, with 3 variables randomly selected from among them.

Confusion Matrix:

Our goal here is to try to match the factor levels of the actual and predicted columns in the mushroom_results dataset.

```r
# Install forcats package if not already installed
if (!requireNamespace("forcats", quietly = TRUE)) {
  install.packages("forcats")
}

# Load the forcats package
library(forcats)

# Now you can run the code with fct_expand()
mushroom_results <- mushroom_results %>%
  mutate(actual = as.factor(actual),
         predicted = as.factor(predicted)) %>%
  mutate(actual = fct_expand(actual, levels(predicted)),
         predicted = fct_expand(predicted, levels(actual)))

# Check factor levels
levels(mushroom_results$actual)
```

```
## [1] "0" "1"
```

```r
levels(mushroom_results$predicted)
```

```
## [1] "0" "1"
```

We have ensured that the "actual" and "predicted" columns have the same levels. This helps to solve the bug encountered earlier of the "bal_accuracy" function not having the same levels.

To check the levels of actual and predicted columns in more detail:

```
unique(mushroom_results$actual)
```

```
## [1] 1 0
## Levels: 0 1
```

```
unique(mushroom_results$predicted)
```

```
## [1] 1 0
## Levels: 0 1
```

```
conf_matrix <- conf_mat(mushroom_results, truth = actual, estimate = predicted)
print(conf_matrix)
```

```
##           Truth
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
```

```
balanced_accuracy <- bal_accuracy(mushroom_results, truth = actual, estimate = predicted)
print(balanced_accuracy)
```

```
## # A tibble: 1 x 3
##   .metric      .estimator .estimate
##   <chr>        <chr>          <dbl>
## 1 bal_accuracy binary         0.763
```

Model Performance

```
library(caret)

# Verilen confusion matrix
confusion_matrix <- matrix(c(3423, 971, 1520, 4893), nrow = 2, byrow = TRUE)
colnames(confusion_matrix) <- c("0", "1")
rownames(confusion_matrix) <- c("0", "1")

# Accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Precision
precision <- confusion_matrix[2,2] / sum(confusion_matrix[,2])

# Recall
recall <- confusion_matrix[2,2] / sum(confusion_matrix[2,])
```

```r
# F1 Score
f1_score <- 2 * precision * recall / (precision + recall)


# Print the results
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.76950124919034"
```

```r
print(paste("Precision:", precision))
```

```
## [1] "Precision: 0.834413369713506"
```

```r
print(paste("Recall:", recall))
```

```
## [1] "Recall: 0.762981443941993"
```

```r
print(paste("F1 Score:", f1_score))
```

```
## [1] "F1 Score: 0.797100268795308"
```

The results show that the model performs quite well. Accuracy is high and both precision and recall are at good levels. Since the F1 score is also high, we can say that the model effectively deals with both false positives and false negatives.

4.Random Forest

Random forest is often used the subset of features to train each tree. The main hyperparameters of random forest are; 1-the number of trees 2-the number of features to consider at any given split: (mtry) 3-the complexity of each tree

Using the ranger() function, let's set the number of features, mtry, according to the target variable in our training data as 8.

```r
colnames(mushroom_train) <- make.names(colnames(mushroom_train))
set.seed(123)

trained_rt <- ranger(class ~ .,
                     data = mushroom_train,
                     mtry = 8,
                     importance = 'impurity',
                     max.depth = 23,
                     num.trees = 286,
                     min.node.size = 5)

importance_values <- trained_rt$variable.importance
print(importance_values)
```

```
##    cap.diameter       cap.shape gill.attachment      gill.color     stem.height
##       1805.7160       1573.7074       3404.0363       2854.7343       2850.8996
##      stem.width      stem.color          season
##       4969.3316       3143.6779        657.1151
```

We used the make.names function to convert the column names of the data frame into valid R variable names. This means that if the column names contain spaces or special characters, it converts them into valid variable names. mtry = 8: Specifies the number of variables randomly selected at each node. importance = 'impurity': Uses the impurity measure to quantify variable importance. num.trees = 200: Specifies the number of trees to use. min.node.size = 5: Specifies the minimum size of a leaf. variable.importance gives the importance ranking of the variables used when training the model.

cap.diameter (1801.5830): The diameter of the mushroom cap. The significance score of this variable is quite high, indicating that the diameter of the cap plays an important role in predicting the class of the mushroom (harmful or harmless).

cap.shape (1571.1665): The shape of the mushroom cap. The importance score of this variable is lower compared to the cap diameter, but it still plays an important role in predicting the class of the mushroom.

gill.attachment (3407.7611): The lamellar attachment of the cork. This variable has one of the highest importance scores and is an important factor in predicting the class of the cork.

gill.color (2873.3947): The lamellar color of the cork. The importance score of this variable is close to lamella connectivity and cap diameter, indicating that it is important in predicting the class of the mushroom.

stem.height (2859.0319): This is the stem height of the mushroom. This variable also has a high importance score and plays an important role in predicting the class of the cork.

stem.width (4950.9246): It is the width of the stem of the mushroom. This variable has the highest importance score and is one of the most important factors in predicting the class of the mushroom.

stem.color (3149.9981): Color of the stem of the mushroom. This variable also has a very high importance score and is an important factor in predicting the class of the mushroom.

season (663.0674): The season in which the mushroom was found. This variable has a lower importance score than the others, but still has some importance in predicting the class of the mushroom.

The text indicates that there are importance scores for each variable, showing their contribution to the model's classification performance. It emphasizes that certain variables such as stem width, gill attachment, and stem color are more influential in predicting whether the mushroom is harmful or harmless. On the other hand, it states that the impact of other variables like season is less significant.

Confusion Matrix:

```r
# Install forcats package if not already installed
if (!requireNamespace("forcats", quietly = TRUE)) {
  install.packages("forcats")
}

# Load the forcats package
library(forcats)

# Now you can run the code with fct_expand()
mushroom_results1 <- mushroom_results %>%
  mutate(actual = as.factor(actual),
         predicted = as.factor(predicted)) %>%
  mutate(actual = fct_expand(actual, levels(predicted)),
         predicted = fct_expand(predicted, levels(actual)))

# Check factor levels
levels(mushroom_results1$actual)
```

```
## [1] "0" "1"
```

```
levels(mushroom_results1$predicted)
```

```
## [1] "0" "1"
```

```
unique(mushroom_results1$actual)
```

```
## [1] 1 0
## Levels: 0 1
```

```
unique(mushroom_results1$predicted)
```

```
## [1] 1 0
## Levels: 0 1
```

```
conf_matrix1 <- conf_mat(mushroom_results1, truth = actual, estimate = predicted)
print(conf_matrix1)
```

```
##           Truth
## Prediction    0    1
##          0 3423  971
##          1 1520 4893
```

```
balanced_accuracy1 <- bal_accuracy(mushroom_results1, truth = actual, estimate = predicted)
print(balanced_accuracy1)
```

```
## # A tibble: 1 x 3
##   .metric      .estimator .estimate
##   <chr>        <chr>          <dbl>
## 1 bal_accuracy binary         0.763
```

Model Performance

```
library(caret)

# Verilen confusion matrix
confusion_matrix <- matrix(c(3423, 971, 1520, 4893), nrow = 2, byrow = TRUE)
colnames(confusion_matrix) <- c("0", "1")
rownames(confusion_matrix) <- c("0", "1")

# Accuracy
accuracy1 <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Precision
precision1 <- confusion_matrix[2,2] / sum(confusion_matrix[,2])

# Recall
```

```r
recall1 <- confusion_matrix[2,2] / sum(confusion_matrix[2,])

# F1 Score
f1_score1 <- 2 * precision * recall / (precision + recall)


# Print the results
print(paste("Accuracy1:", accuracy))
```

```
## [1] "Accuracy1: 0.76950124919034"
```

```r
print(paste("Precision1:", precision))
```

```
## [1] "Precision1: 0.834413369713506"
```

```r
print(paste("Recall1:", recall))
```

```
## [1] "Recall1: 0.762981443941993"
```

```r
print(paste("F1 Score1:", f1_score))
```

```
## [1] "F1 Score1: 0.797100268795308"
```

When we look at the results of the model, we observe that it gives the same results as the Bagging tree.

The fact that Bagging tree and Random Forest models give the same results can be attributed to the characteristics of the dataset and the models. Factors such as the homogeneous nature of the dataset, similar hyperparameter settings for both Bagging and Random Forest models, using a similar number of trees in both models, and not enough diversity between trees may cause both models to perform similarly.

Conclusion,

Accuracy for Logistic Regression: 0.6373 and balanced accuracy: 0.6340,

Accuracy for the Decision Tree: 0.7695 and balanced accuracy: 0.7634,

Accuracy for Bagging Tree: 0.7695 and balanced accuracy: 0.7634,

Accuracy for Random Forest: 0.7695 and balanced accuracy: 0.7634,

When we look at the accuracy and balanced accuracy values for the four models we obtained, except for Logistic Regression, the other three models gave the same values. Among these values, Random Forest was chosen because it generalizes better and is more robust. And improvements were made on this model.