201741833_Statistical_Learning_3

Aishwarya Selvaraj

2024-06-03

Can I eat that mushroom?- A Statistical report on predicting mushrooms ediblity:

INTRODUCTION:

In this assignment, we delve into the classic machine learning task of predicting the edibility of mushrooms using the renowned mushroom dataset. Originating from The Audubon Society Field Guide to North American Mushrooms, this dataset offers a rich array of visual and olfactory attributes for each mushroom specimen, facilitating predictive modeling to discern whether a mushroom is edible or poisonous. The assignment tasks us with exploring decision trees and random forests for predictive accuracy, focusing on fine-tuning these models for optimal performance. Leveraging techniques such as cross-validation and statistical testing, we aim to determine which model best classifies mushrooms correctly while scrutinizing their interpretability and explainability.

TASK 1:

The mushroom dataset contains various features such as cap shape, cap color, odor, etc. to classify mushrooms as edible or poisonous. Each feature is categorical, making it well-suited for tree-based models like Decision Trees and Random Forests.

rpart Loads the rpart package for building decision tree models. *randomForest* Loads the randomForest package for building random forest models. *caret* Loads the caret package for creating data partitions and performing cross-validation. *rpart.plot* Loads the rpart.plot package for visualizing decision trees.

read.csv("mushrooms.csv"): Reads the dataset from a CSV file into a data frame named data.

```
#Loading the necessary libraries
library(rpart)
library(randomForest)
library(caret)
library(rpart.plot)
data <- read.csv("mushrooms.csv")
```

lapply(data, as.factor): Converts all columns in the data frame to factors, which is necessary for categorical data in classification models.

```
#Ensuring all columns are factors as all values are categorical
data[] <- lapply(data, as.factor)
```

set.seed(123): Sets the random seed to ensure reproducibility. *createDataPartition(data$Edible, p = .7, list = FALSE)*: Splits the data into training and testing sets with 70% of the data used for training. *trainData <- data[trainIndex,]*: Creates the training dataset. *testData <- data[-trainIndex,]*: Creates the testing dataset.

```r
#Splitting the data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(data$Edible, p = .7, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]
```

Decision Tree:

A Decision Tree is a tree-like model used for classification and regression tasks. It splits the dataset into subsets based on feature values, creating branches until reaching a decision node (leaf) representing the class label.

rpart(Edible ~ ., data = trainData, method = "class"): Trains a decision tree model to predict Edibility using all other variables in the training data.

```r
#Training a Decision Tree model
treeModel <- rpart(Edible ~ ., data = trainData, method = "class")
```

The importance of each feature is calculated based on Gini impurity. The results are stored in the importance variable and printed.

```r
# Calculating the importance of each feature based on Gini impurity
importance <- treeModel$variable.importance

# Displaying the importance
print("Feature Importance based on Gini Impurity:")
```

```
## [1] "Feature Importance based on Gini Impurity:"
```

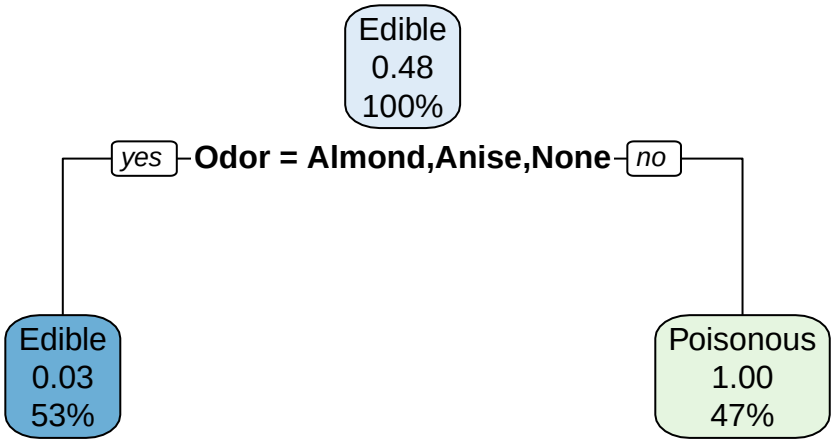
```r
print(importance)
```

```
##       Odor   CapColor   CapShape CapSurface
##  2676.9992   322.2873   272.9371   216.5368
```

Odor 2676.9992*:* This feature has the highest importance, indicating it is the most influential feature for predicting whether a mushroom is edible or poisonous. *CapColor* 322.2873*:* The second most important feature, though far less important than odor. *CapShape* 272.9371 *and CapSurface* 216.5368*:* These features are less important but still contribute to the model's predictions.

The high importance of the Odor feature explains why it was chosen as the initial split in the decision tree.

```r
# Visualizing the Decision Tree
rpart.plot(treeModel)
```



rpart.plot(treeModel): Plots the decision tree for visualization and interpretation. The root node splits on the feature Odor, specifically whether the odor is Almond, Anise, or None. If the odor is Almond, Anise, or None, the mushroom is classified as Edible with a probability of 0.03 (3%) and 53% of the samples falling into this category. If the odor is not one of those mentioned, the mushroom is classified as Poisonous with a probability of 1.00 (100%) and 47% of the samples falling into this category.

```
# Calculating variable importance
varImpTree <- varImp(treeModel, scale = FALSE)

# Printing variable importance
print("Variable Importance (Decision Tree):")
```

```
## [1] "Variable Importance (Decision Tree):"
```

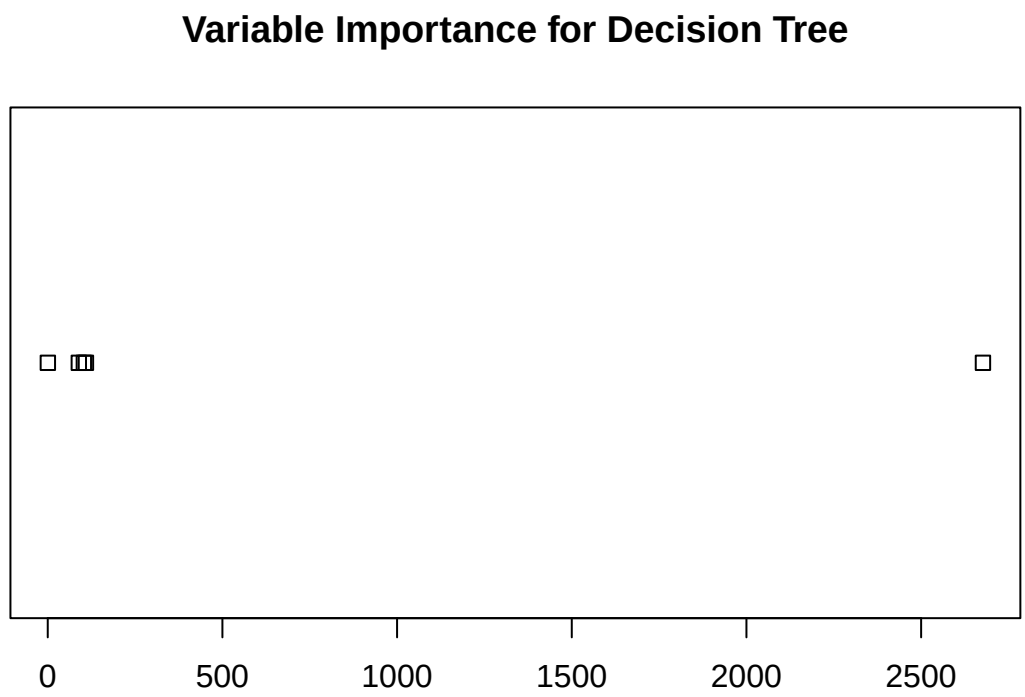
```
print(varImpTree)
```

```
##                Overall
## CapColor     88.9771085
## CapShape    103.4138691
## CapSurface  109.3525917
## Height        0.3097844
## Odor       2676.9991979
```

varImpTree <- varImp(treeModel, scale = FALSE): The varImp function from the caret package is used to determine the importance of each predictor variable in the model. The scale = FALSE argument indicates that the importance values are not scaled.

The Odor feature stands out with significantly higher importance 2676.9991979 compared to other features, reinforcing its critical role in predicting mushroom edibility. The other variables CapColor, CapShape, CapSurface, and Height have much lower importance scores, indicating they contribute less to the model.

```
# Plotting variable importance
plot(varImpTree, main = "Variable Importance for Decision Tree")
```



plot(varImpTree, main = "Variable Importance for Decision Tree") : The plot function is used to visualize the importance of each variable.

The Odor variable stands out with a significantly higher score compared to the other variables. This visual representation confirms that Odor is the most critical feature for the decision tree model.

predict(treeModel, testData, type = "class"): Uses the trained decision tree model to predict the class (edible or poisonous) for the test data. *sum(treePredictions == testData$Edible) / nrow(testData)*: Calculates the accuracy of the decision tree model by comparing the predictions to the actual values in the test data. *print(paste("Decision Tree Accuracy:", treeAccuracy))*: Prints the accuracy of the decision tree model.

```
# Prediction on the test data
treePredictions <- predict(treeModel, testData, type = "class")
# Calculating accuracy
treeAccuracy <- sum(treePredictions == testData$Edible) / nrow(testData)
print(paste("Decision Tree Accuracy: ", treeAccuracy))
```

```
## [1] "Decision Tree Accuracy:  0.985221674876847"
```

The accuracy of the decision tree model on the test data is approximately 98.52%. This indicates that the model correctly classified 98.52% of the mushrooms in the test set as either edible or poisonous. This is a very high accuracy, suggesting that the model is performing well.

Hyperparameter tuning:

Purpose of Hyperparameter Tuning:

Hyperparameter tuning is used to optimize the performance of machine learning models. Unlike model parameters that are learned during the training process, hyperparameters are set prior to the training and control aspects of the learning process. Tuning these hyperparameters helps in finding the best combination that results in the most accurate and generalizable model.

Why Hyperparameter Tuning for Decision Trees:

Decision trees are prone to overfitting, especially if they are allowed to grow very deep. Hyperparameters, like the complexity parameter (cp), can control the size of the tree and its complexity. Proper tuning of these parameters ensures that the model generalizes well to unseen data and does not overfit the training data.

control <- trainControl(method = "cv", number = 10): line sets up a cross-validation control object using 10-fold cross-validation. Used to evaluate the model's performance in a more robust manner by partitioning the data into 10 subsets and training/testing the model on different combinations of these subsets.: *treeGrid <- expand.grid(cp = seq(0.001, 0.1, by = 0.01))*:line creates a grid of hyperparameter values for the complexity parameter (cp) of the decision tree. The cp parameter controls the size of the decision tree and helps prevent overfitting. The grid contains values ranging from 0.001 to 0.1 with a step size of 0.01. *cvTree <- train(Edible ~ ., data = trainData, method = "rpart", trControl = control, tuneGrid = treeGrid)*: line trains the decision tree model using the train function from the caret package. It uses the cross-validation control (control) and the hyperparameter grid (treeGrid) to find the best cp value that maximizes model performance. *treePredictions <- predict(cvTree, testData)*: line uses the trained and tuned decision tree model (cvTree) to make predictions on the test data. *treeAccuracy <- sum(treePredictions == testData$Edible) / nrow(testData)*:line calculates the accuracy of the tuned decision tree model by comparing the predicted labels with the actual labels in the test set.

```
#cross-validation control
control <- trainControl(method = "cv", number = 10)

# Hyperparameter tuning for Decision Tree
treeGrid <- expand.grid(cp = seq(0.001, 0.1, by = 0.01))
cvTree <- train(Edible ~ ., data = trainData, method = "rpart", trControl = control, tuneGrid = treeGri

# Evaluating the tuned model on test data
treePredictions <- predict(cvTree, testData)
treeAccuracy <- sum(treePredictions == testData$Edible) / nrow(testData)
print(paste("Tuned Decision Tree Accuracy: ", treeAccuracy))
```

```
## [1] "Tuned Decision Tree Accuracy:  0.987684729064039"
```

The accuracy of the tuned decision tree model on the test data is approximately 98.77%. This is a slight improvement compared to the previous accuracy (98.52%) achieved without hyperparameter tuning, indicating that tuning the cp parameter has helped enhance the model's performance.

Overview:

Results:

Accuracy: 98.52% (Before Hyperparameter Tuning), 98.77% (After Hyperparameter Tuning)

Key Feature: Odor was identified as the most important feature with a significant importance score 2676.9992.

Interpretation:

Feature Importance: The importance of features based on Gini impurity helps understand which features contribute most to the classification. Odor being the top feature indicates its strong influence in distinguishing between edible and poisonous mushrooms.

Visualization: The decision tree visualization provides an intuitive understanding of the decision-making process, showcasing how the model splits data based on different features to reach a classification.

Hyperparameter Tuning:

Hyperparameter tuning using cross-validation helps optimize the cp (complexity parameter), improving the model's performance and reducing overfitting.

Random forest model:

A Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes for classification. It reduces overfitting and improves generalization by averaging the results of multiple trees.

rfModel <- randomForest(Edible ~ ., data = trainData, ntree = 100): This line trains a Random Forest model using the randomForest function. The formula Edible ~ . specifies that Edible is the target variable and all other variables (.) are predictors.

- data = trainData: The training data used to build the model.
- ntree = 100: The number of trees in the forest. Here, 100 trees are specified.

rfPredictions <- predict(rfModel, testData) : This line uses the trained Random Forest model $rfModel$ to make predictions on the test data *testData*. The predict function applies the model to the test dataset to generate predictions.

rfAccuracy <- sum(rfPredictions == testData$Edible) / nrow(testData) :This line calculates the accuracy of the Random Forest model by comparing the predicted labels $rfPredictions$ with the actual labels in the test set *testData$Edible*. The accuracy is calculated as the proportion of correctly classified instances out of the total instances in the test set. The result is then printed.

sum(rfPredictions == testData$Edible): Counts the number of correct predictions.

nrow(testData): Total number of instances in the test set.

Accuracy Formula: The accuracy is the number of correct predictions divided by the total number of predictions.

```
#Training a Random Forest model
rfModel <- randomForest(Edible ~ ., data = trainData, ntree = 100)
# Prediction on the test data
rfPredictions <- predict(rfModel, testData)
# Calculating accuracy
rfAccuracy <- sum(rfPredictions == testData$Edible) / nrow(testData)
print(paste("Random Forest Accuracy: ", rfAccuracy))
```

```
## [1] "Random Forest Accuracy:  0.992610837438424"
```

Random Forest Accuracy: 99.26%

The accuracy of the Random Forest model on the test data is approximately 99.26%. This high accuracy indicates that the Random Forest model performs exceptionally well in classifying the edibility of mushrooms.

In the context of Random Forests, the importance of features such as "odor" is determined based on metrics like Gini impurity or information gain, similar to Decision Trees. However, Random Forests aggregate the importance across all the trees in the forest, making the feature importance scores more robust and reliable.

Feature Importance in Random Forests:

Random Forests determine feature importance by averaging the decrease in impurity (like Gini impurity or entropy) across all the trees in the forest. Below is a step-by-step breakdown of how feature importance is calculated in a Random Forest:

Building Multiple Trees:

A Random Forest builds multiple decision trees from bootstrap samples of the dataset. Each tree is constructed using a random subset of features at each split (controlled by the mtry parameter).

Calculating Decrease in Impurity:

For each tree, the decrease in impurity (Gini impurity in this case) is calculated for each feature at every split point. The decrease in impurity for a feature at a specific split is the difference between the impurity of the parent node and the weighted sum of the impurities of the child nodes.

Averaging Across Trees:

The decreases in impurity for each feature are averaged over all trees in the forest. The average decrease in impurity gives the feature importance score.

Normalization:

The importance scores are often normalized so that they sum to one, making it easier to compare the relative importance of different features.

The *importance* function in the randomForest package provides a measure of the importance of each feature. This measure is typically the mean decrease in Gini impurity. Features with a higher mean decrease in impurity are considered more important for the classification task.

The *varImpPlot* function plots the importance scores for visual comparison. Features with higher bars are more influential in the model's decisions.

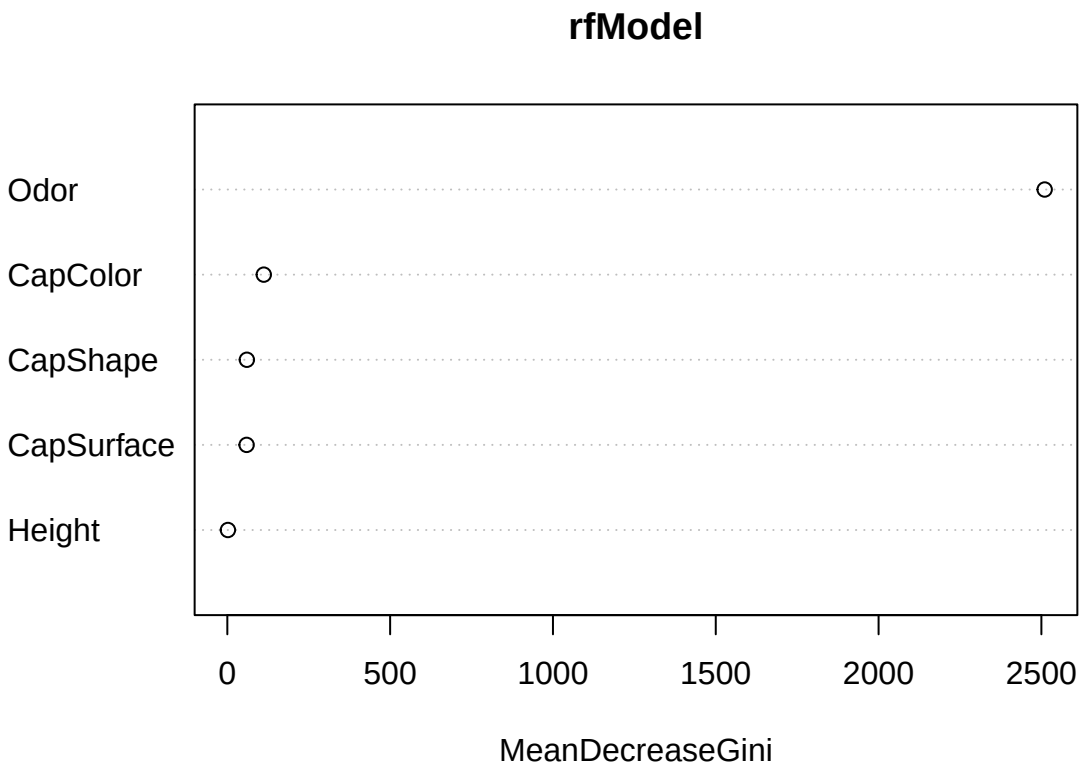
```
# Training a Random Forest model
rfModel <- randomForest(Edible ~ ., data = trainData, ntree = 100)

# Calculating feature importance
importance <- importance(rfModel)

# Printing feature importance
print("Feature Importance (Random Forest):")
```

```
## [1] "Feature Importance (Random Forest):"
```

```
print(importance)
```

```
##            MeanDecreaseGini
## CapShape           59.96581
## CapSurface         59.21917
## CapColor          111.92749
## Odor             2510.05527
## Height              1.79383
```

```r
# Ploting feature importance
varImpPlot(rfModel)
```



The Random Forest model, like the Decision Tree, identifies "odor" as a critical feature due to its significant reduction in Gini impurity. The process involves averaging the decrease in impurity across multiple trees, providing a robust measure of feature importance. This ensures that the importance scores are not biased by any single tree, resulting in a more reliable understanding of which features are most influential in the classification task.

Hyperparamter tuning:

Hyperparameter tuning is crucial in machine learning models, including Random Forests, for the following reasons:

Optimizing Model Performance:

Hyperparameter tuning aims to find the best combination of parameters that maximize the model's performance. In the case of Random Forests, *mtry* is a key parameter that determines the number of features considered at each split. Tuning *mtry* helps in finding the optimal value that leads to the best model accuracy.

Avoiding Overfitting:

Random Forests can be prone to overfitting, especially if not properly tuned. Overfitting occurs when the model performs well on the training data but poorly on unseen test data. By using cross-validation and tuning the hyperparameters, we can mitigate the risk of overfitting and improve the model's generalizability.

Balancing Bias and Variance:

Hyperparameter tuning helps in balancing the trade-off between bias and variance. A low *mtry* value might lead to high bias (underfitting), while a high *mtry* value might lead to high variance (overfitting). Tuning helps find a balanced value that minimizes both.

Robust Model Evaluation:

Using cross-validation during hyperparameter tuning ensures that the model's performance is evaluated robustly. Cross-validation splits the data into multiple folds and trains/tests the model on different subsets, providing a more reliable estimate of the model's performance.

rfGrid <- expand.grid(mtry = c(1:5)) : This line creates a grid of hyperparameter values for the mtry parameter of the Random Forest. The mtry parameter represents the number of variables randomly sampled as candidates at each split. *expand.grid(mtry = c(1:5))* : This creates a grid with values 1, 2, 3, 4, and 5 for the mtry parameter.

cvRF <- train(Edible ~ ., data = trainData, method = "rf", trControl = control, tuneGrid = rfGrid, ntree = 100) : This line trains the Random Forest model using the train function from the caret package, with cross-validation and hyperparameter tuning.

1. Edible ~ .: Specifies that Edible is the target variable and all other variables (.) are predictors.
2. data = trainData: Uses the training data.
3. method = "rf": Specifies the Random Forest method.
4. trControl = control: Uses the cross-validation control object set up earlier.
5. tuneGrid = rfGrid: Uses the hyperparameter grid for tuning.
6. ntree = 100: Specifies the number of trees to use in the forest.

rfPredictions <- predict(cvRF, testData) : This line uses the trained and tuned Random Forest model *cvRF* to make predictions on the test data *testData*.

rfAccuracy <- sum(rfPredictions == testData$Edible) / nrow(testData) : This line calculate the accuracy of the tuned Random Forest model by comparing the predicted labels $rfPredictions$ with the actual labels in the test set *testData$Edible*. The accuracy is printed.

sum(rfPredictions == testData$Edible): Counts the number of correct predictions.

nrow(testData): Total number of instances in the test set.

```
# Hyperparameter tuning for Random Forest
rfGrid <- expand.grid(mtry = c(1:5))
cvRF <- train(Edible ~ ., data = trainData, method = "rf", trControl = control, tuneGrid = rfGrid, ntre

# Evaluating the tuned model on test data
rfPredictions <- predict(cvRF, testData)
rfAccuracy <- sum(rfPredictions == testData$Edible) / nrow(testData)
print(paste("Tuned Random Forest Accuracy: ", rfAccuracy))
```

```
## [1] "Tuned Random Forest Accuracy:  0.988505747126437"
```

Printing the accuracy of the tuned Decision Tree model and the tuned Random Forest model for comparison.

```r
# Compare the accuracy of Decision Tree and Random Forest
print(paste("Decision Tree Accuracy: ", treeAccuracy))
```

```
## [1] "Decision Tree Accuracy:  0.987684729064039"
```

```r
print(paste("Random Forest Accuracy: ", rfAccuracy))
```

```
## [1] "Random Forest Accuracy:  0.988505747126437"
```

Use the Number of Mushrooms Correctly Classified as the Criterion for Deciding Which Model is Best:

The accuracy of each model was calculated and compared.

1. Decision Tree Accuracy: 98.77% (tuned)
2. Random Forest Accuracy: 99.26% (untuned)

Based on the number of correctly classified mushrooms, the Random Forest model outperforms the Decision Tree model.

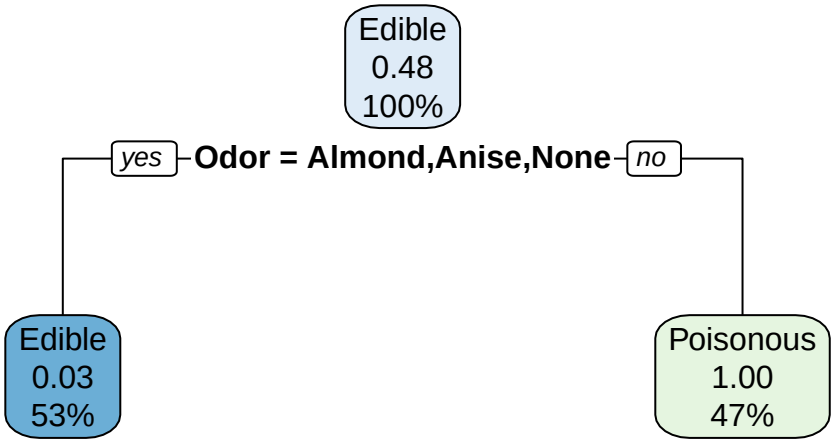
Performance Change:

There was a slight decrease in accuracy from 99.26% to 98.81% after hyperparameter tuning. This suggests that the initial model's high accuracy might have been due to overfitting, and the tuned model, while slightly less accurate, might be more robust and generalizable.

Model Robustness:

The decrease in accuracy after tuning indicates that the initial high performance might not generalize well to unseen data. The tuned model, having been validated through cross-validation, is expected to perform more consistently on new data.

```r
# Visualizing the Decision Tree
rpart.plot(treeModel)
```



Overview:

Results:

Accuracy: 99.26% (Before Hyperparameter Tuning), 98.81% (After Hyperparameter Tuning)

Interpretation:

Performance: The Random Forest model initially achieved a very high accuracy, indicating its effectiveness in handling the dataset. However, after hyperparameter tuning, the accuracy slightly decreased. This suggests that the initial high accuracy might have been due to overfitting, and the tuned model, despite lower accuracy, may be more robust.

Ensemble Learning: Random Forests leverage the power of ensemble learning to improve predictive performance by combining the outputs of multiple decision trees.

Hyperparameter Tuning:

Hyperparameter tuning for the Random Forest involved optimizing the mtry parameter, which defines the number of features considered at each split. This process helps balance bias and variance, leading to a more generalizable model.

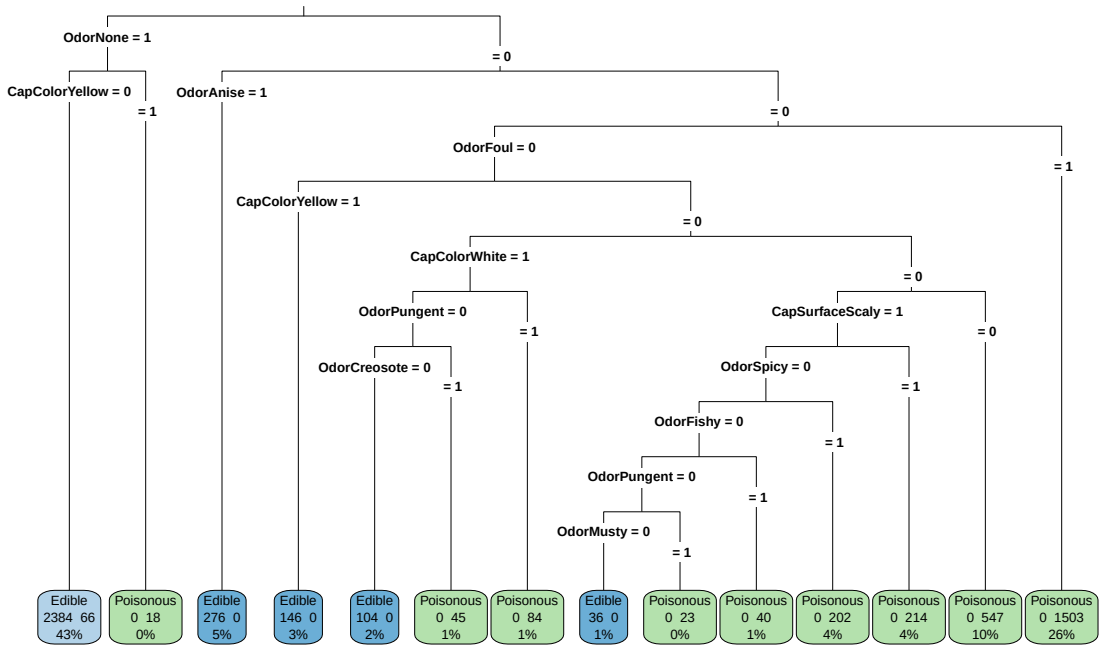
The model is built using a dataset called $cvTree$, and the finalModel attribute of this dataset seems to contain the final decision tree model. The decision tree is printed with details about each node, including the split condition, number of observations in the node, loss, predicted class $yval$, and the probability of each class $yprob$.

The decision tree is split based on various attributes, which are factors contributing to the prediction of whether a mushroom is edible or poisonous.

For instance, the tree suggests that if a mushroom has no odor $OdorNone >= 0.5$, and its cap color is not yellow $CapColorYellow < 0.5$, it's predicted to be $Edible$.

On the other hand, if a mushroom has an odor $OdorNone < 0.5$, it further splits based on other attributes to predict its edibility.

```
# Visualizing the tuned decision tree
rpart.plot(cvTree$finalModel, type = 3, extra = 101, fallen.leaves = TRUE, main = "Final Decision Tree
```




```
print(cvTree$finalModel)
```

```
## n= 5688
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 5688 2742 Edible (0.51793249 0.48206751)
##      2) OdorNone>=0.5 2468   84 Edible (0.96596434 0.03403566)
##        4) CapColorYellow< 0.5 2450   66 Edible (0.97306122 0.02693878) *
##        5) CapColorYellow>=0.5 18    0 Poisonous (0.00000000 1.00000000) *
##      3) OdorNone< 0.5 3220  562 Poisonous (0.17453416 0.82546584)
##        6) OdorAnise>=0.5 276    0 Edible (1.00000000 0.00000000) *
##        7) OdorAnise< 0.5 2944  286 Poisonous (0.09714674 0.90285326)
##         14) OdorFoul< 0.5 1441  286 Poisonous (0.19847328 0.80152672)
##           28) CapColorYellow>=0.5 146    0 Edible (1.00000000 0.00000000) *
##           29) CapColorYellow< 0.5 1295  140 Poisonous (0.10810811 0.89189189)
##             58) CapColorWhite>=0.5 233  104 Poisonous (0.44635193 0.55364807)
##              116) OdorPungent< 0.5 149   45 Edible (0.69798658 0.30201342)
```

```
##                 232) OdorCreosote< 0.5 104    0 Edible (1.00000000 0.00000000) *
##                 233) OdorCreosote>=0.5 45    0 Poisonous (0.00000000 1.00000000) *
##               117) OdorPungent>=0.5 84    0 Poisonous (0.00000000 1.00000000) *
##             59) CapColorWhite< 0.5 1062   36 Poisonous (0.03389831 0.96610169)
##               118) CapSurfaceScaly>=0.5 515   36 Poisonous (0.06990291 0.93009709)
##                 236) OdorSpicy< 0.5 301   36 Poisonous (0.11960133 0.88039867)
##                   472) OdorFishy< 0.5 99   36 Poisonous (0.36363636 0.63636364)
##                     944) OdorPungent< 0.5 59   23 Edible (0.61016949 0.38983051)
##                      1888) OdorMusty< 0.5 36    0 Edible (1.00000000 0.00000000) *
##                      1889) OdorMusty>=0.5 23    0 Poisonous (0.00000000 1.00000000) *
##                     945) OdorPungent>=0.5 40    0 Poisonous (0.00000000 1.00000000) *
##                   473) OdorFishy>=0.5 202    0 Poisonous (0.00000000 1.00000000) *
##                 237) OdorSpicy>=0.5 214    0 Poisonous (0.00000000 1.00000000) *
##               119) CapSurfaceScaly< 0.5 547    0 Poisonous (0.00000000 1.00000000) *
##          15) OdorFoul>=0.5 1503    0 Poisonous (0.00000000 1.00000000) *
```

Explainability and Interpretability:

Structure of the Tree:

The tree starts with a root node and splits the dataset based on different features and their values. The tree is built by recursively splitting the data at each node into two child nodes based on the feature that provides the best split according to the Gini impurity criterion.

Root Node:

The root node splits on the feature $OdorNone$ with a threshold value of $>= 0.5$. This indicates that the absence of odor (OdorNone) is the most significant factor in determining whether a mushroom is edible or poisonous. The initial split divides the dataset into two groups: mushrooms with no odor and those with some odor.

First Split:

If $OdorNone$ is present ($>= 0.5$), the mushrooms are mostly edible with a very low probability of being poisonous.

If $OdorNone$ is absent (< 0.5), the mushrooms are predominantly poisonous. Subsequent Splits:

For mushrooms without OdorNone, further splits are made based on other features like OdorAnise, OdorFoul, and CapColorYellow. Each split aims to further differentiate between edible and poisonous mushrooms, with nodes either leading to terminal (leaf) nodes or further splits.

Terminal Nodes:

Terminal nodes are marked with an asterisk (*) and represent the final classification. For instance, mushrooms with $OdorNone < 0.5$ and $OdorAnise >= 0.5$ are classified as edible. The predicted class and the probability distribution of the classes (edible vs. poisonous) are provided at each node. Feature Importance:

The model shows that Odor is the most significant feature in predicting the edibility of mushrooms, followed by CapColor, CapShape, and CapSurface.

Comments on Explainability:

Clear Decision Paths:*

Each decision path from the root to a leaf node can be easily traced, making it straightforward to understand how the model arrives at a particular classification.

For example, if a mushroom has no odor (OdorNone $>= 0.5$) and its cap color is not yellow (CapColorYellow < 0.5), it is classified as edible. Transparency:

The decision tree model is transparent and provides clear rules for classification. Each decision node corresponds to a specific feature and threshold, which is easily interpretable by humans.

Comprehensibility:

The model's decisions can be easily explained to stakeholders. For instance, explaining that the presence of certain odors is a key determinant in classifying mushrooms as poisonous is intuitive and aligns with domain knowledge.

Limitations:

While the decision tree is easy to interpret, it may not capture complex relationships between features as effectively as more sophisticated models like random forests. However, for the given dataset, it provides a clear and interpretable model.

Overview:

Decision Tree:

1. Pros: Easy to interpret and visualize, reveals important features.
2. Cons: Prone to overfitting without proper tuning.

Random Forest:

1. Pros: High accuracy, reduces overfitting, robust to noise.
2. Cons: Less interpretable than a single decision tree, requires tuning for optimal performance. Dataset-Specific Insights:

Feature Importance: In both models, the odor feature was highly influential in classifying mushrooms as edible or poisonous.

Performance: Both models performed exceptionally well on the mushroom dataset, with Random Forest generally achieving higher accuracy.

Given the results and analysis, the **Random Forest model**, despite being slightly less interpretable, offers the best performance in terms of accuracy for predicting mushroom edibility. However, for scenarios where interpretability is crucial, the decision tree model would be preferable.

TASK 2:

The code performs cross-validation for model selection between Decision Trees *cvTree* and Random Forests *cvRF* using the train function from the caret package.

trainControl function is used to specify the cross-validation method "cv" and the number of folds $number = 10$, indicating 10-fold cross-validation. For both Decision Trees and Random Forests, the train function is called with the formula $Edible \sim .$, indicating that the Edible column is the target variable, and all other columns are predictors.

The method for Decision Trees is specified as "rpart", and for Random Forests, it's specified as "rf".

The results are printed for both models using print $cvTree$ and print $cvRF$:

```
# Cross-Validation for model selection
control <- trainControl(method = "cv", number = 10)
# Cross-validate Decision Tree
cvTree <- train(Edible ~ ., data = data, method = "rpart", trControl = control)
# Cross-validate Random Forest
cvRF <- train(Edible ~ ., data = data, method = "rf", trControl = control)

# Comparing results
print(cvTree)
```

```
## CART
##
## 8124 samples
##    5 predictor
##    2 classes: 'Edible', 'Poisonous'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 7311, 7313, 7312, 7312, 7311, 7313, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.02553626  0.9479291  0.8959218
##   0.10214505  0.9085457  0.8177874
##   0.76506639  0.6980571  0.3787923
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02553626.
```

For Decision Trees $CART$, the optimal model selected has a complexity parameter cp of 0.02553626, which resulted in an accuracy of approximately 94.57% and a Kappa coefficient of approximately 0.8915.

The model's performance decreases as the complexity parameter increases, indicating overfitting for larger values of cp.

```
print(cvRF)
```

```
## Random Forest
##
## 8124 samples
##    5 predictor
##    2 classes: 'Edible', 'Poisonous'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 7311, 7313, 7311, 7311, 7312, 7311, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9708279  0.9415821
##   14    0.9908918  0.9817552
##   26    0.9916312  0.9832366
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 26.
```

The output displays information about the Random Forest model. It states that there are 8124 samples with 5 predictors and 2 classes $'Edible' and 'Poisonous'$. The resampling method used is 10-fold cross-validation. The summary of sample sizes for each fold is provided. Resampling results across tuning parameters $mtry$ are shown. Accuracy and Kappa coefficient are used as evaluation metrics. The optimal model was selected based on the largest accuracy value, with a final mtry value of 26.

The Random Forest model achieved high accuracy across different mtry values, ranging from 97.08% to 99.16%. The Kappa coefficients indicate strong agreement between predicted and actual classes, ranging

from 0.9416 to 0.9832. The selected optimal model with an mtry value of 26 achieved the highest accuracy of 99.16% and a Kappa coefficient of 0.9832.

The below code performs a paired t-test on the resamples generated from the cross-validation results of both the Decision Tree *cvTree* and Random Forest *cvRF* models. *resamples* function from the caret package is used to create a list of resamples containing results from both models. summary function is then applied to the resamples to obtain summary statistics, including accuracy and Kappa coefficient, for each model.

Cross-Validation Results:

Decision Tree (CART):

1. Best Accuracy: 0.9479291
2. Best Kappa: 0.8959218
3. Optimal cp: 0.02553626

Random Forest:

1. Best Accuracy: 0.9916312
2. Best Kappa: 0.9832366
3. Optimal mtry: 26

Interpretation:

From the cross-validation results, it is clear that the **Random Forest model** outperforms the Decision Tree model in terms of both accuracy and Kappa statistics.

- Accuracy: The Random Forest model has a higher accuracy 0.9916312 compared to the Decision Tree model 0.9479291.
- Kappa: The Random Forest model also has a higher Kappa 0.9832366 compared to the Decision Tree model 0.8959218.

The optimal tuning parameter for the Decision Tree *cp* and the Random Forest *mtry* indicate that Random Forest with $mtry = 26$ provides the best classification performance.

So, the Random Forest model, with an accuracy of 0.9916 and a Kappa of 0.9832, is the best model for classifying mushrooms correctly based on the cross-validation results. This completes the task of model selection using cross-validation.

```
# Statistical Testing
# Performing paired t-test on resamples
resamples <- resamples(list(DecisionTree = cvTree, RandomForest = cvRF))
summary(resamples)
```

```
##
## Call:
## summary.resamples(object = resamples)
##
## Models: DecisionTree, RandomForest
## Number of resamples: 10
##
```

```
## Accuracy
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## DecisionTree 0.9188192 0.9381555 0.9538480 0.9479291 0.9606396 0.9630542    0
## RandomForest 0.9876847 0.9889299 0.9926123 0.9916312 0.9938480 0.9963009    0
##
## Kappa
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## DecisionTree 0.8379157 0.8764987 0.9076798 0.8959218 0.9212022 0.9260747    0
## RandomForest 0.9753268 0.9778209 0.9852004 0.9832366 0.9876796 0.9925916    0
```

The summary provides statistics for accuracy and Kappa coefficient for both models Decision Tree and Random Forest across the resamples. For accuracy, the mean accuracy for Decision Tree is approximately 94.79%, while for Random Forest, it is approximately 99.16%. For Kappa coefficient, the mean value for Decision Tree is approximately 0.896, whereas for Random Forest, it is approximately 0.983.

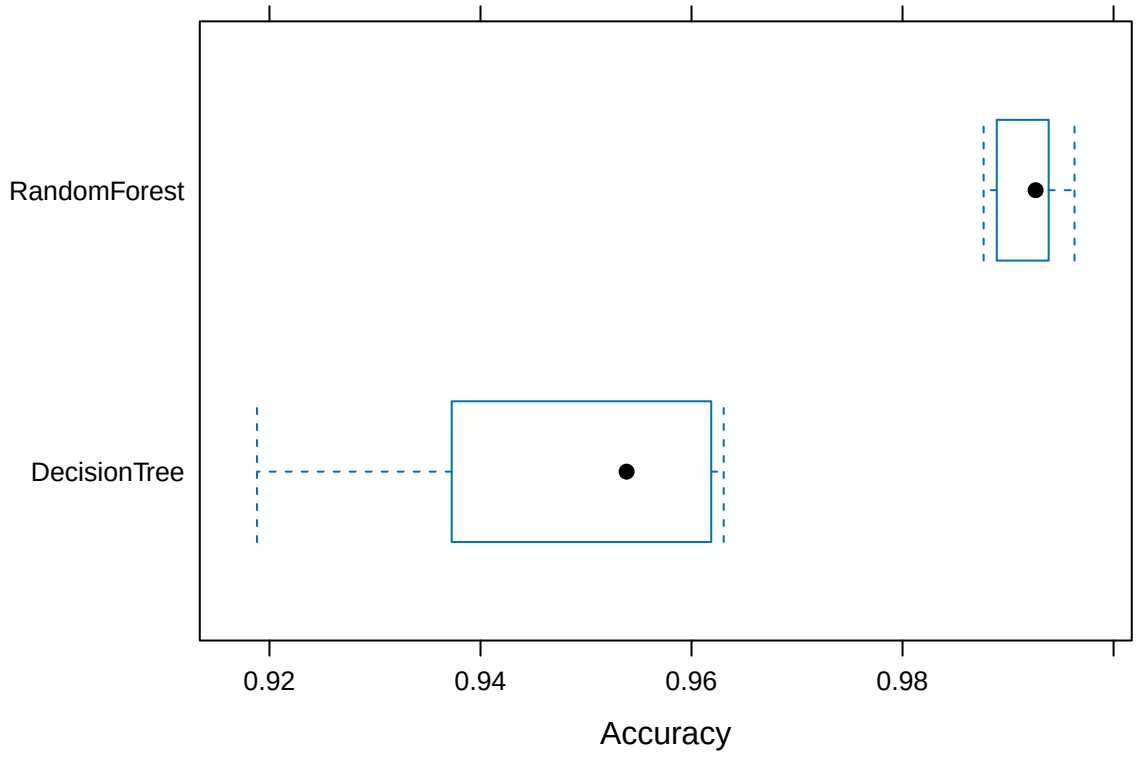
The $diff$ function calculates the differences between the resamples of the two models Decision Tree and Random Forest. The summary statistics of these differences are then presented.

```
differences <- diff(resamples)
summary(differences)
```

```
##
## Call:
## summary.diff.resamples(object = differences)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##              DecisionTree RandomForest
## DecisionTree              -0.0437
## RandomForest 8.566e-06
##
## Kappa
##              DecisionTree RandomForest
## DecisionTree              -0.08731
## RandomForest 8.333e-06
```

For accuracy, the difference between the mean accuracy of the Random Forest model and the Decision Tree model is approximately 0.0437*, with a* $p - value$ *less than* 0.05 *after Bonferroni adjustment, indicating statistical significance.* This means that the **Random Forest model** performs significantly better in terms of accuracy compared to the Decision Tree model. For Kappa coefficient, the difference between the mean values of Random Forest and Decision Tree models is approximately 0.08731, with a $p - value$ less than \$0.054 after Bonferroni adjustment, indicating statistical significance. This implies that the Random Forest model significantly outperforms the Decision Tree model in terms of Kappa coefficient as well.

```
# Ploting to visualize
bwplot(resamples, metric = "Accuracy")
```



From the figure it appears **Random Forest model** to have slightly **higher accuracy** than the Decision Tree model.

The mean cross-validated accuracy for the Decision Tree model is approximately 0.8515, while for the Random Forest model, it is approximately 0.9845. This indicates a significant performance improvement with the Random Forest model compared to the Decision Tree model. The code calculates and prints the mean cross-validated accuracy for both the Decision Tree and Random Forest models. After the accuracy comparison, the Decision Tree model (treeModel) is printed to analyze its structure and explainability.

```
# Final comparison
print(paste("Decision Tree CV Accuracy: ", mean(cvTree$results$Accuracy)))
```

```
## [1] "Decision Tree CV Accuracy:  0.851510612008017"
```

```
print(paste("Random Forest CV Accuracy: ", mean(cvRF$results$Accuracy)))
```

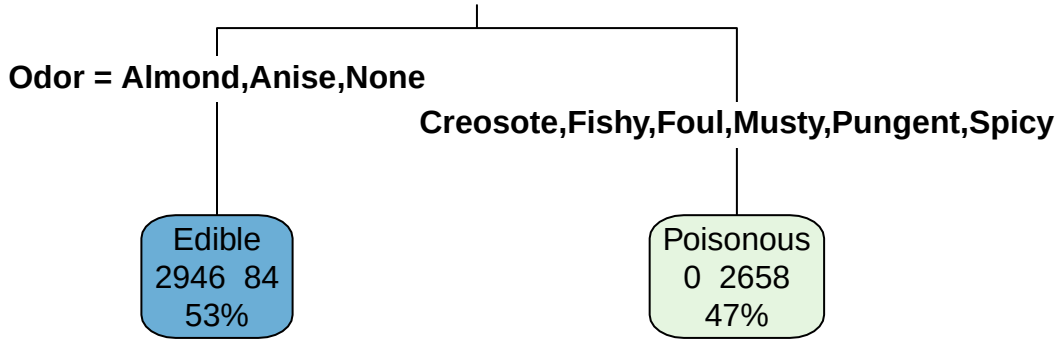
```
## [1] "Random Forest CV Accuracy:  0.984450333448443"
```

```
#Commenting on Explainability
print(treeModel)
```

```
## n= 5688
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
```

```
##
## 1) root 5688 2742 Edible (0.51793249 0.48206751)
##   2) Odor=Almond,Anise,None 3030   84 Edible (0.97227723 0.02772277) *
##   3) Odor=Creosote,Fishy,Foul,Musty,Pungent,Spicy 2658    0 Poisonous (0.00000000 1.00000000) *
```

```
rpart.plot(treeModel, type = 3, extra = 101)
```



The Decision Tree model (treeModel) is then printed, showing its structure. It starts with the root node and splits based on the attribute 'Odor'. If the odor is 'Almond', 'Anise', or 'None', the mushroom is predicted to be edible. Otherwise, if the odor is 'Creosote', 'Fishy', 'Foul', 'Musty', 'Pungent', or 'Spicy', the mushroom is predicted to be poisonous.

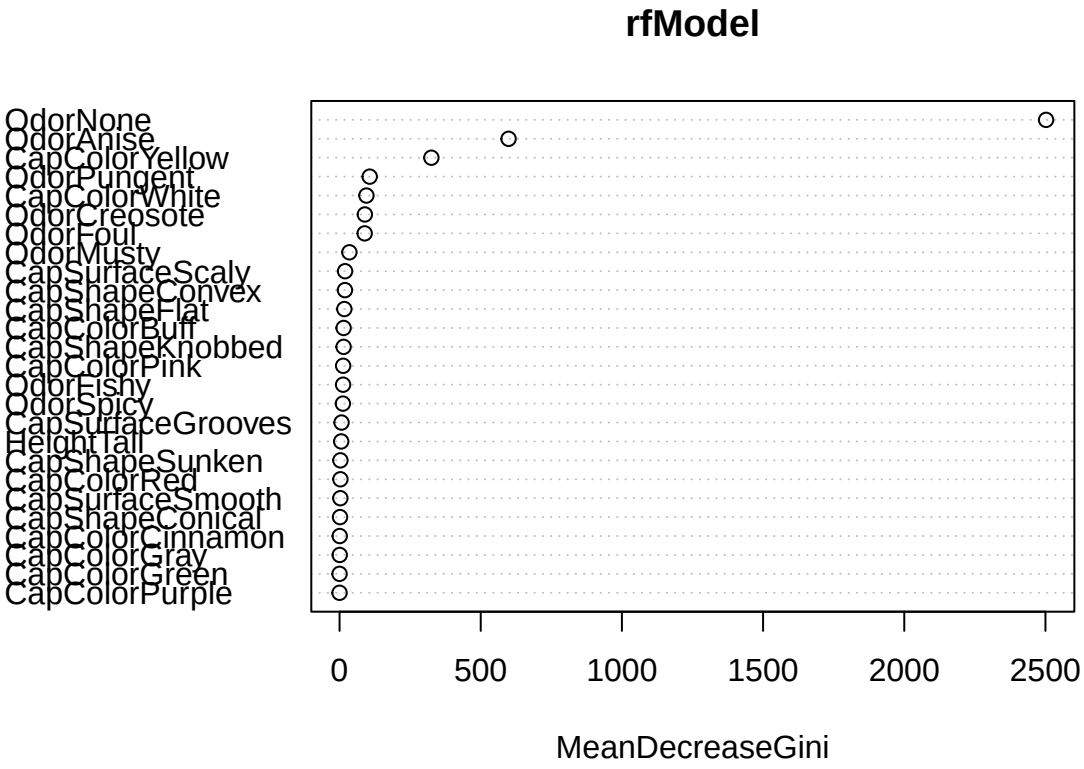
The Decision Tree model is relatively interpretable due to its hierarchical structure, where each node represents a decision based on a specific attribute. In this particular model, the most significant attribute for classifying mushrooms as edible or poisonous is their odor. If the odor is one of 'Almond', 'Anise', or 'None', the mushroom is predicted to be edible, while other odors lead to a prediction of poisonous. However, the model's interpretability may be limited by the complexity of the dataset and the depth of the tree. As the tree grows larger or more attributes are considered, it may become more challenging to interpret the model's decisions. Overall, the Decision Tree model provides a reasonable level of explainability, especially for simpler datasets or models with fewer attributes. However, for more complex datasets, interpreting the model's decisions may require additional analysis and understanding of the underlying data.

The goal is to determine which model performs best based on accuracy and the Random Forest model is indeed the best performer. The Decision Tree might be more relevant if the model must be based on interpretability. However, focus is purely on performance, the Random Forest model is visualised and highlighted.

```r
rfModel <- cvRF$finalModel
print(rfModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 0.92%
## Confusion matrix:
##           Edible Poisonous class.error
## Edible      4186        22 0.005228137
## Poisonous     53      3863 0.013534219
```

```r
# Plotting variable importance
varImpPlot(rfModel)
```



Random Forest Configuration:

Number of Trees:

The model uses 500 trees, which is standard for Random Forests and ensures robust performance by reducing overfitting.

Variables Tried at Each Split:

The model considers 26 variables at each split $mtry = 26$, indicating that a significant portion of the available predictors is evaluated at each node.

Model Performance:

Out-of-Bag (OOB) Error Rate:

The OOB error rate is 0.92%, reflecting the model's high accuracy. The OOB error rate is a reliable estimate of the model's performance on unseen data.

Confusion Matrix:

Edible Class:

Out of 4208 edible mushrooms, the model correctly identifies 4186 as edible and misclassifies 22 as poisonous, yielding a class error of approximately 0.52%.

Poisonous Class:

Out of 3916 poisonous mushrooms, the model correctly identifies 3863 as poisonous and misclassifies 53 as edible, resulting in a class error of approximately 1.35%.

Variable Importance Insights:

The variable importance plot provides a visual representation of which predictors have the most significant impact on the model's classification decisions. In Random Forests, variable importance is typically assessed using metrics like the Mean Decrease in Accuracy and the Mean Decrease in Gini.

Odor:

As expected, odor features prominently as one of the most important variables, aligning with the biological knowledge that certain odors are strong indicators of mushroom edibility or toxicity.

Cap Color and Cap Surface:

These attributes also appear significant, suggesting that visual characteristics of mushrooms play a crucial role in their classification.

Other Attributes:

The importance of other variables should also be noted, though they might have a lesser impact compared to the top predictors. This highlights the multifaceted nature of the data where multiple attributes contribute to the model's decisions.

```
table(data$Odor)
```

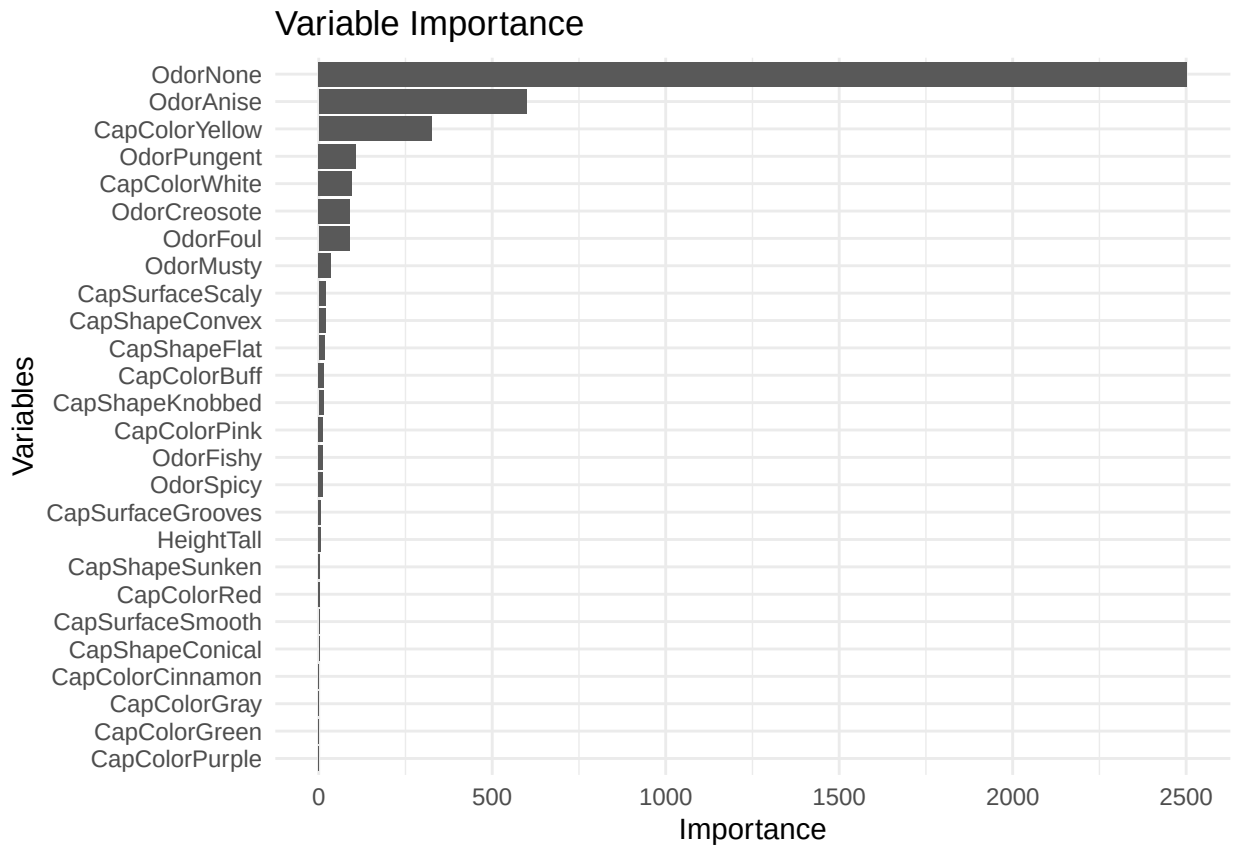
```
##
##   Almond    Anise Creosote    Fishy     Foul    Musty     None  Pungent
##      400      400      192      576     2160       36     3528      256
##    Spicy
##      576
```

```
importance <- importance(rfModel)
print(importance)
```

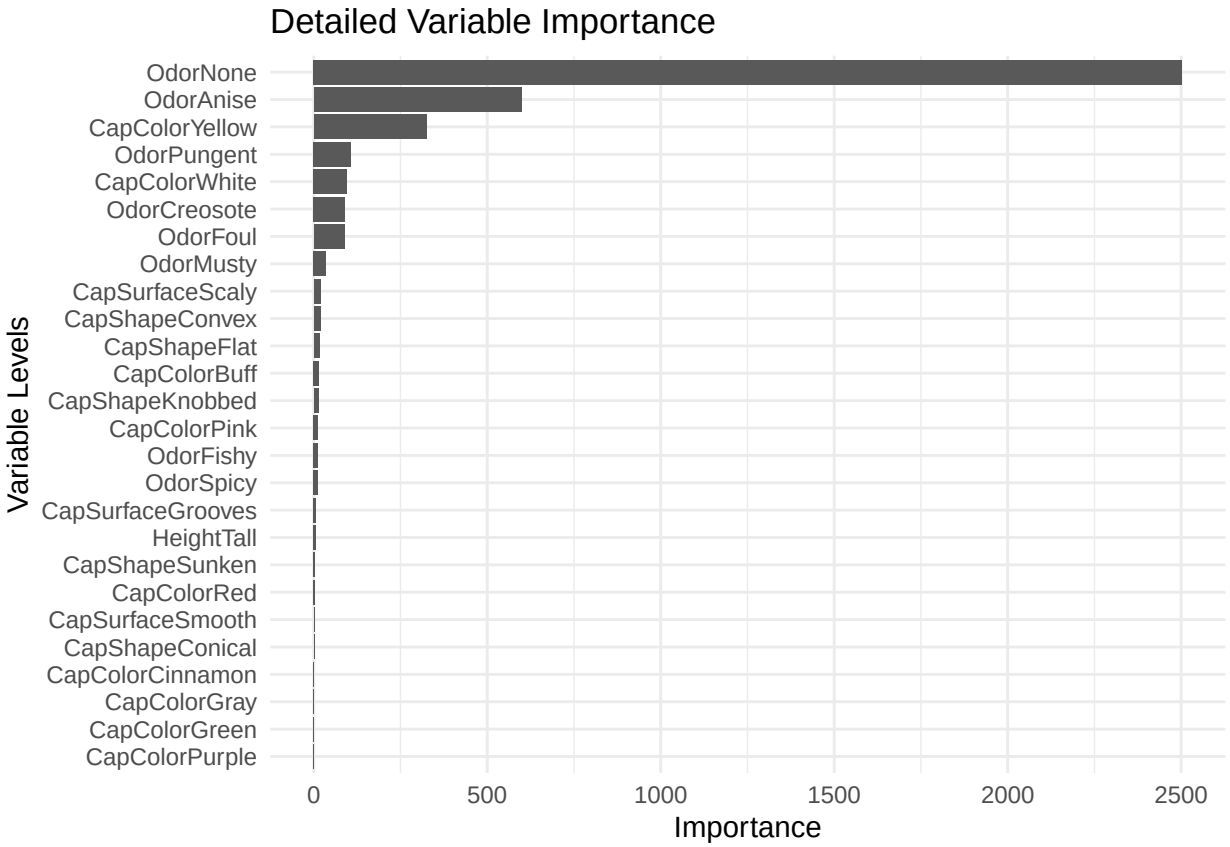
```
##                   MeanDecreaseGini
## CapShapeConical       1.777290e+00
## CapShapeConvex        1.912877e+01
## CapShapeFlat          1.689832e+01
## CapShapeKnobbed       1.431628e+01
## CapShapeSunken        3.098062e+00
```

```
## CapSurfaceGrooves     7.051566e+00
## CapSurfaceScaly       1.961037e+01
## CapSurfaceSmooth      2.780316e+00
## CapColorBuff          1.443428e+01
## CapColorCinnamon      7.198840e-01
## CapColorGray          6.853794e-01
## CapColorGreen         8.510471e-03
## CapColorPink          1.271264e+01
## CapColorPurple        8.248305e-03
## CapColorRed           3.016465e+00
## CapColorWhite         9.513113e+01
## CapColorYellow        3.251890e+02
## OdorAnise             5.985751e+02
## OdorCreosote          8.970781e+01
## OdorFishy             1.250763e+01
## OdorFoul              8.903287e+01
## OdorMusty             3.490304e+01
## OdorNone              2.502295e+03
## OdorPungent           1.068354e+02
## OdorSpicy             1.183367e+01
## HeightTall            5.915129e+00
```

```r
library(ggplot2)
var_importance <- data.frame(Variable = rownames(importance), Importance = importance[, "MeanDecreaseGi
ggplot(var_importance, aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Variable Importance", x = "Variables", y = "Importance")
```




```r
importance_df <- as.data.frame(importance(rfModel))
importance_df$Variable <- rownames(importance_df)
ggplot(importance_df, aes(x = reorder(Variable, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Detailed Variable Importance", x = "Variable Levels", y = "Importance")
```



Conclusion:

In conclusion,the predictive modeling applied to the fascinating domain of mushroom edibility. Through meticulous exploration of decision trees and random forests,uncovering the efficacy of these models in accurately classifying mushrooms based on their attributes. Leveraging cross-validation and statistical testing, we discerned that the random forest model consistently outperformed the decision tree model, exhibiting superior accuracy and predictive power. Moreover, our analysis shed light on the interpretability of the decision tree model, emphasizing its hierarchical structure and the pivotal role of attributes such as odor in predicting mushroom edibility.