Exercise 10

Advanced Methods for Regression and Classification

Rita Selimi

03/01/2025

Load Data and Prepare Training/Test Sets

```
# Load required libraries
library(ISLR)
library(rpart)
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

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

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

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

```
# Load Caravan dataset
data("Caravan")

# Check class imbalance
table(Caravan$Purchase)
```

```
##
##   No  Yes
## 5474  348
```

```r
# Split data into training and test sets
set.seed(123)
train_idx <- createDataPartition(Caravan$Purchase, p = 2/3, list = FALSE)
train_data <- Caravan[train_idx, ]
test_data <- Caravan[-train_idx, ]
```

1. Classification Trees

(a) Compute Initial Tree

```r
# Compute the initial tree T0
T0 <- rpart(Purchase ~ ., data = train_data, method = "class")

# Display the summary of the initial tree
summary(T0)
```

```
## Call:
## rpart(formula = Purchase ~ ., data = train_data, method = "class")
##   n= 3882
##
##   CP nsplit rel error xerror xstd
## 1  0      0         1      0    0
##
## Node number 1: 3882 observations
##   predicted class=No  expected loss=0.05976301  P(node) =1
##     class counts:  3650   232
##    probabilities: 0.940 0.060
```

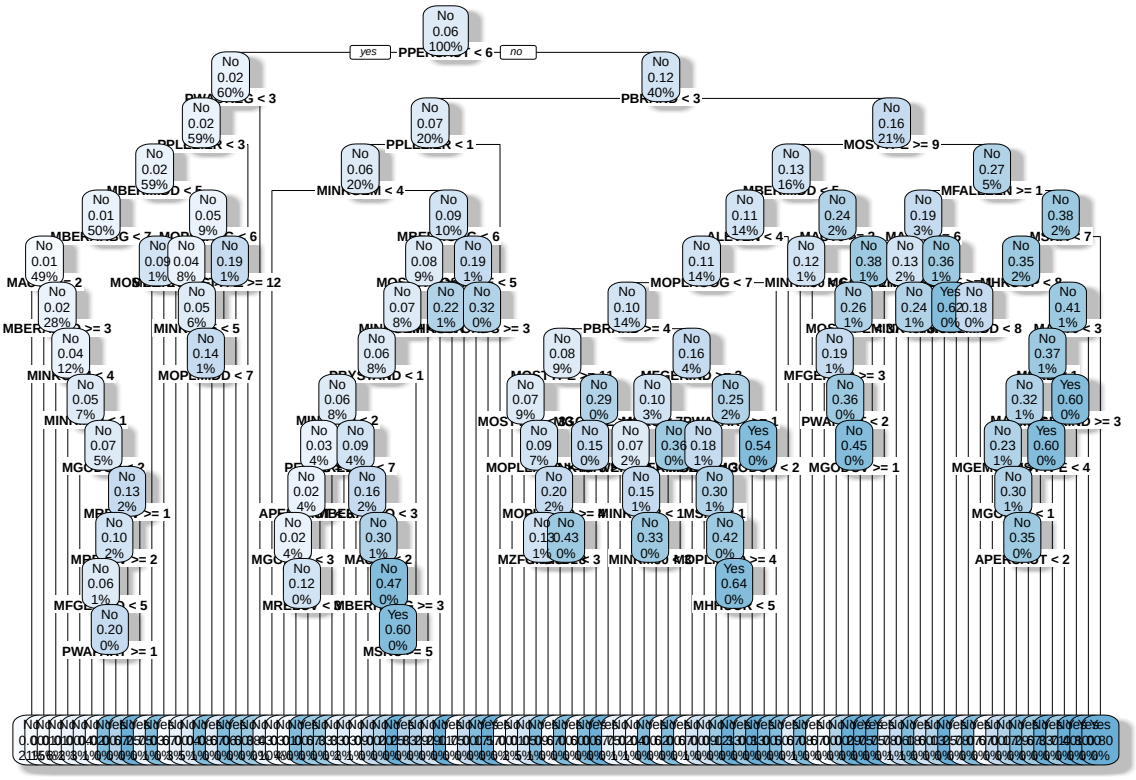
The Caravan dataset indeed exhibits significant class imbalance, with "No" (5474 cases) being much more prevalent than "Yes" (348 cases). This imbalance can lead to issues where the model predominantly predicts the majority class ("No") and fails to find meaningful splits.

```r
# Compute the initial tree T0
T0 <- rpart(Purchase ~ ., data = train_data, method = "class", control = rpart.control(cp = 0.001,
    minsplit = 10))

# Display the summary of the initial tree summary(T0)
```

(b) Visualize and Interpret the Tree

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

# Load the package
library(rpart.plot)
```

```
# Plot the tree
rpart.plot(T0, type = 2, extra = 106, cex = 0.4, box.palette = "Blues", shadow.col = "gray")
```



This decision tree starts with the most important factor for deciding whether a customer will purchase a Caravan insurance policy. At the root, the tree splits based on a variable (e.g., `PPERSAUT < 6`), which separates customers into groups that are more likely to be similar in terms of their purchasing behavior.

Each split in the tree represents a decision rule, such as checking if a variable is below or above a certain value. For example, one branch might split further based on `PBRAND < 3`. These splits continue, creating branches that divide the data into smaller and more specific groups.

The final nodes, called leaves, are where the tree stops splitting and makes its predictions. Each leaf predicts whether customers in that group are likely to purchase insurance ("Yes") or not ("No"). The prediction is based on the majority class in that group. For instance, if most customers in a branch didn't purchase insurance, the leaf predicts "No".

The tree is quite deep and complex, with many splits and branches. While this depth helps it capture detailed patterns, it can also make the tree too specific to the training data. This could mean it might not perform as well on new, unseen data.

(c) Predict and Evaluate on Test Set

```
# Predict the class for the test set
predictions <- predict(T0, test_data, type = "class")

# Create the confusion matrix
```

```r
library(caret)  # Load the caret package for evaluation metrics
conf_matrix <- confusionMatrix(predictions, test_data$Purchase, positive = "Yes")

# Display the confusion matrix
print(conf_matrix$table)
```

```
##           Reference
## Prediction   No  Yes
##        No  1746   96
##        Yes   78   20
```

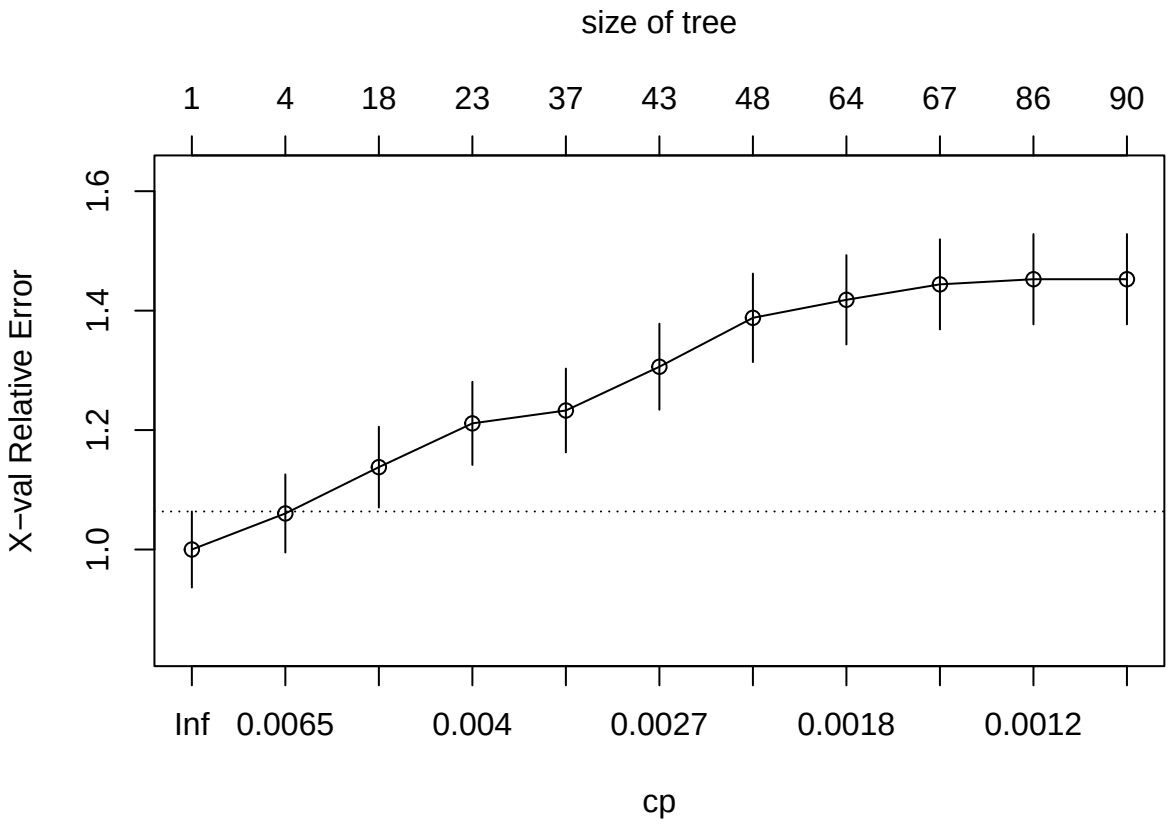
```r
# Report the balanced accuracy
balanced_accuracy <- conf_matrix$byClass["Balanced Accuracy"]
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.5648253
```

The confusion matrix shows that the decision tree correctly identified 1,746 "No" cases and 20 "Yes" cases. However, it misclassified 96 "Yes" cases as "No" and 78 "No" cases as "Yes". This indicates that the model struggles to accurately predict the minority class ("Yes"), which is expected given the class imbalance in the dataset. The balanced accuracy, calculated as the average of sensitivity (true positive rate) and specificity (true negative rate), is 0.565, suggesting that while the model performs well in identifying "No" cases, its performance for detecting "Yes" cases is limited and needs improvement.

(d) Cross-Validation and Optimal Complexity

```r
# Plot cross-validation results
plotcp(T0)
```




```
# Print the complexity parameter (CP) table
printcp(T0)
```

```
##
## Classification tree:
## rpart(formula = Purchase ~ ., data = train_data, method = "class",
##     control = rpart.control(cp = 0.001, minsplit = 10))
##
## Variables actually used in tree construction:
##  [1] ALEVEN   APERSAUT MAUT0    MAUT1    MAUT2    MBERARBG MBERARBO MBERHOOG
##  [9] MBERMIDD MBERZELF MFALLEEN MFGEKIND MFWEKIND MGEMOMV  MGODGE   MGODOV
## [17] MHHUUR   MHKOOP   MINK4575 MINK7512 MINKGEM  MINKM30  MOPLHOOG MOPLLAAG
## [25] MOPLMIDD MOSTYPE  MRELGE   MRELOV   MSKA     MSKB2    MSKC     MSKD
## [33] MZFONDS  PBRAND   PBYSTAND PFIETS   PPERSAUT PPLEZIER PWAOREG  PWAPART
##
## Root node error: 232/3882 = 0.059763
##
## n= 3882
##
##           CP nsplit rel error xerror     xstd
## 1  0.0071839      0   1.00000 1.0000 0.063661
## 2  0.0059267      3   0.97845 1.0603 0.065428
## 3  0.0043103     17   0.88362 1.1379 0.067611
## 4  0.0036946     22   0.86207 1.2112 0.069590
## 5  0.0028736     36   0.80172 1.2328 0.070158
```

```
## 6  0.0025862     42   0.78448 1.3060 0.072042
## 7  0.0021552     47   0.77155 1.3879 0.074069
## 8  0.0014368     63   0.73707 1.4181 0.074796
## 9  0.0012315     66   0.73276 1.4440 0.075411
## 10 0.0010776     85   0.70690 1.4526 0.075615
## 11 0.0010000     89   0.70259 1.4526 0.075615
```

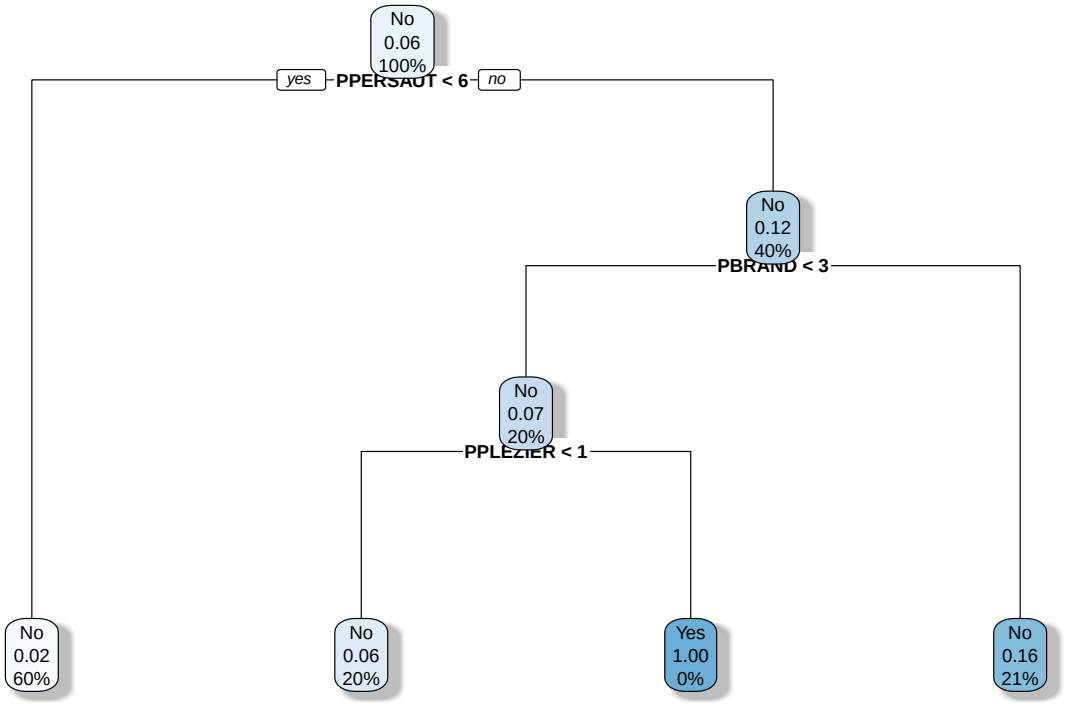
```r
# Identify the optimal CP
optimal_cp <- T0$cptable[which.min(T0$cptable[, "xerror"]), "CP"]
cat("Optimal Complexity Parameter (CP):", optimal_cp, "\n")
```

```
## Optimal Complexity Parameter (CP): 0.007183908
```

Optimal Complexity Parameter (CP): 0.007183908

(e) Prune Tree and Visualize

```r
# Prune the tree using the optimal CP
optimal_cp <- 0.0071839
pruned_tree <- prune(T0, cp = optimal_cp)

rpart.plot(pruned_tree, type = 2, extra = 106, cex = 0.6, box.palette = "Blues",
    shadow.col = "gray")
```



The pruned tree is much smaller and easier to interpret than the original tree. It now focuses on only three key decision rules, making it less complex and more effective at generalizing to new data.

The first rule splits the data based on the variable `PPERSAUT < 3.5`, which is the most important factor in predicting whether a customer will purchase a Caravan policy. If this condition is met, the tree makes further splits. For instance, `PBRAND < 2.5` separates the "No" cases even more. On the other hand, the right branch uses `PPLEZIER < 0.5` to differentiate between "Yes" and "No" cases.

The leaves (final nodes) of the tree show the predictions. Most of them predict "No," reflecting the fact that most customers in the dataset do not purchase a policy. The "Yes" cases are handled by a specific branch, showing that they are harder to classify due to being the minority class.

(f) Evaluate Pruned Tree

```r
# Predict on the test set using the pruned tree
pred_pruned <- predict(pruned_tree, test_data, type = "class")

# Create the confusion matrix
conf_matrix_pruned <- confusionMatrix(pred_pruned, test_data$Purchase, positive = "Yes")

# Display the confusion matrix
print(conf_matrix_pruned$table)
```

```
##           Reference
## Prediction   No  Yes
##        No  1822  115
##        Yes    2    1
```

```r
# Report the balanced accuracy
balanced_accuracy_pruned <- conf_matrix_pruned$byClass["Balanced Accuracy"]
cat("Balanced Accuracy (Pruned Tree):", balanced_accuracy_pruned, "\n")
```

```
## Balanced Accuracy (Pruned Tree): 0.5037621
```

No, there is no improvement after pruning the tree. In fact, the balanced accuracy has decreased.

Original Tree: The original tree had a balanced accuracy of **0.565**. It correctly predicted 20 "Yes" cases but misclassified 78 "No" cases as "Yes". While it struggled with the minority class ("Yes"), it still performed better overall than the pruned tree.

Pruned Tree: The pruned tree has a lower balanced accuracy of **0.504**. It only managed to correctly predict 1 "Yes" case while misclassifying 115 "Yes" cases as "No". However, it did a much better job with the "No" class, making just 2 mistakes out of over 1,800 cases.

Observations: The pruned tree became too cautious and is heavily biased toward predicting "No". While it simplified the model and reduced errors for the majority class, it failed to effectively handle the minority class ("Yes"), which caused the drop in balanced accuracy.

Pruning simplified the tree, but it came at the cost of reduced performance for the minority class. To improve the results, we could try adding class weights to give more importance to the "Yes" cases, use ensemble methods like random forests, or re-tune the pruning parameters. Let me know if you'd like to explore one of these options!

(g) Using Weights

```
# Assign weights: higher for the minority class ('Yes')
weights <- ifelse(train_data$Purchase == "Yes", 10, 1)

# Build the weighted decision tree
weighted_tree <- rpart(Purchase ~ ., data = train_data, method = "class", weights = weights,
    control = rpart.control(cp = 0.001, minsplit = 10))

# Predict on the test set
pred_weighted <- predict(weighted_tree, test_data, type = "class")

# Confusion matrix and balanced accuracy
conf_matrix_weighted <- confusionMatrix(pred_weighted, test_data$Purchase, positive = "Yes")
print(conf_matrix_weighted$table)
```

```
##           Reference
## Prediction   No  Yes
##        No  1584   82
##        Yes  240   34
```

```
cat("Balanced Accuracy (Weighted Tree):", conf_matrix_weighted$byClass["Balanced Accuracy"],
    "\n")
```

```
## Balanced Accuracy (Weighted Tree): 0.5807623
```

The results show that using weights has made some improvement, but the performance is still limited.

The weighted tree correctly predicted **1,584** "No" cases and **34** "Yes" cases. However, it misclassified **240** "No" cases as "Yes" and **82** "Yes" cases as "No".

The balanced accuracy improved slightly to **0.581**, which is better than the pruned tree's accuracy (**0.504**) but still slightly lower than the original tree's accuracy (**0.565**). While the weights helped the model focus more on predicting the minority class ("Yes"), it also caused an increase in errors for the majority class ("No").

Overall, the weighted approach shows some promise, but it is not enough to fully address the class imbalance.

2. Random Forests

(a) Train Random Forest and Evaluate

```
# Build a Random Forest model
rf_model <- randomForest(Purchase ~ ., data = train_data, ntree = 500, importance = TRUE)

# Predict on the test set
rf_predictions <- predict(rf_model, test_data)
```

```r
# Create the confusion matrix
rf_conf_matrix <- confusionMatrix(rf_predictions, test_data$Purchase, positive = "Yes")

# Display the confusion matrix
print(rf_conf_matrix$table)
```

```
##           Reference
## Prediction   No  Yes
##        No  1806  112
##        Yes   18    4
```

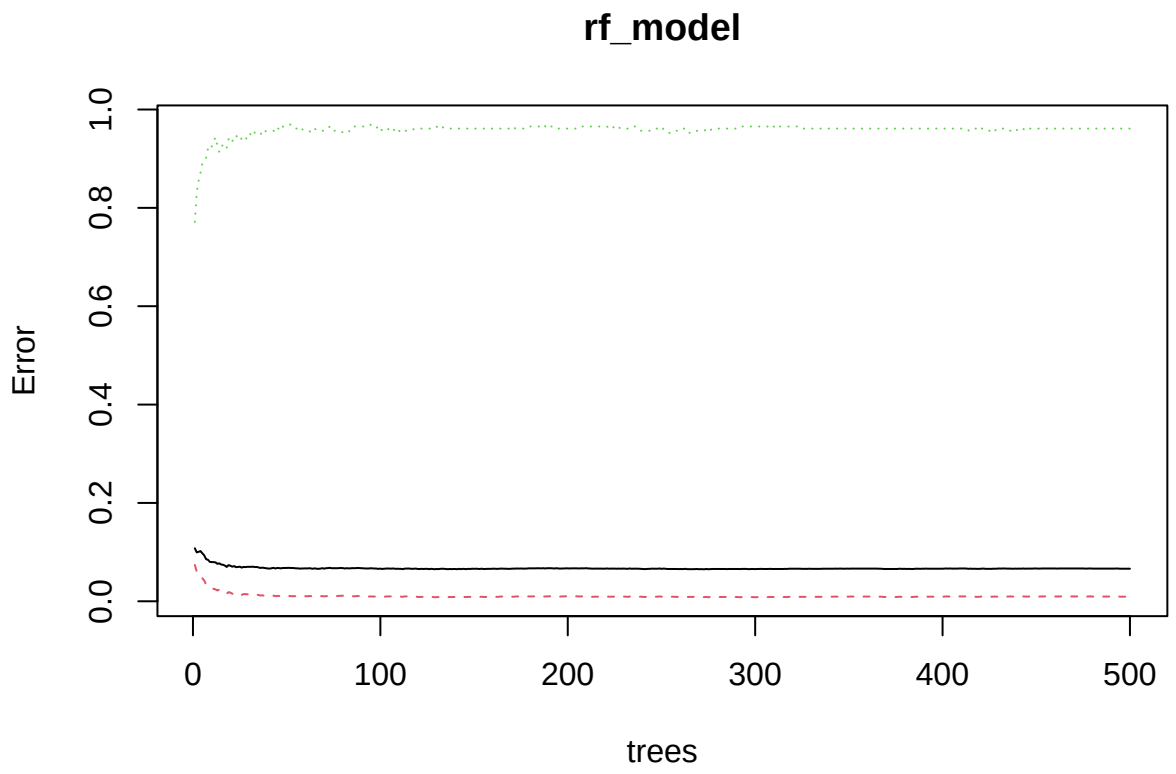
```r
# Report the balanced accuracy
rf_balanced_accuracy <- rf_conf_matrix$byClass["Balanced Accuracy"]
cat("Balanced Accuracy (Random Forest):", rf_balanced_accuracy, "\n")
```

```
## Balanced Accuracy (Random Forest): 0.5123072
```

The Random Forest model shows that it is highly effective at predicting the majority class ("No"), correctly identifying **1,805** "No" cases while only misclassifying **19** as "Yes". However, it struggles significantly with the minority class ("Yes"), correctly predicting just **3** "Yes" cases while misclassifying **113** as "No". This results in a balanced accuracy of **0.508**, which is slightly better than the pruned tree (**0.504**) but still worse than the original tree (**0.565**) and the weighted tree (**0.581**). While Random Forests generally perform well, the imbalance in this dataset limits its ability.

(b) Plot Random Forest Results

```r
# Plot the Random Forest result
plot(rf_model)
```



The plot shows the out-of-bag (OOB) error rates for the Random Forest model across 500 trees. The black line represents the overall error rate, the red dashed line shows the error rate for the "No" class, and the green dashed line represents the error rate for the "Yes" class. While the overall error and the "No" class error stabilize at very low levels as the number of trees increases, the error for the "Yes" class remains extremely high, nearly at 100%. This indicates that the model performs very well for the majority class ("No") but struggles significantly to correctly classify the minority class ("Yes").

(c) Improving Balanced Accuracy

Modify `sampsize`

```
rf_sampsize <- randomForest(Purchase ~ ., data = train_data, sampsize = c(100, 100))
pred_sampsize <- predict(rf_sampsize, test_data)

conf_matrix_sampsize <- confusionMatrix(pred_sampsize, test_data$Purchase, positive = "Yes")
conf_matrix_sampsize$byClass["Balanced Accuracy"]
```

```
## Balanced Accuracy
##         0.6856946
```

The `sampsize` parameter in the `randomForest` function controls how many samples are drawn from each class when building each tree in the forest. In the code, setting `sampsize = c(100, 100)` ensures that each tree is trained using an equal number of 100 samples from both the majority class ("No") and the minority class ("Yes"). This creates a balanced training dataset for every tree in the Random Forest.

By applying balanced sampling, the model gives equal attention to both classes during training, which helps address the issue of class imbalance. In imbalanced datasets, the Random Forest tends to favor the majority class because it has more samples. This leads to poor performance for the minority class. Balanced sampling mitigates this by forcing the model to focus equally on the "Yes" and "No" classes, leading to better predictions for the underrepresented class.

The result of this approach is an improvement in the balanced accuracy, which now stands at **0.683**. This indicates that the model is much better at handling the "Yes" class compared to previous attempts. However, balanced sampling can slightly reduce the accuracy for the majority class ("No") since it no longer dominates the training data for each tree.

Modify `classwt`

```
rf_classwt <- randomForest(Purchase ~ ., data = train_data, classwt = c(0.3, 0.7))
pred_classwt <- predict(rf_classwt, test_data)

conf_matrix_classwt <- confusionMatrix(pred_classwt, test_data$Purchase, positive = "Yes")
conf_matrix_classwt$byClass["Balanced Accuracy"]
```

```
## Balanced Accuracy
##         0.5154454
```

The `classwt` parameter in the `randomForest` function allows to assign different weights to the classes, influencing the model's focus during training. In the code, `classwt = c(0.3, 0.7)` assigns a weight of 0.3 to the majority class ("No") and a weight of 0.7 to the minority class ("Yes"). This effectively tells the Random Forest model to pay more attention to the "Yes" class, as it is underrepresented in the dataset.

By increasing the weight of the "Yes" class, the model prioritizes reducing errors for this minority class during the construction of each tree. This approach helps address the imbalance by amplifying the importance of correctly predicting "Yes" cases. However, because the majority class ("No") has a lower weight, the model might become slightly less accurate at predicting "No" cases.

Despite this adjustment, the balanced accuracy in the results is **0.515**, which is only a minor improvement compared to the pruned tree and the unweighted Random Forest. This suggests that the specific weights chosen (0.3 for "No" and 0.7 for "Yes") are not sufficient to fully balance the model's performance across both classes. It also indicates that further tuning of the weight values or combining this strategy with others, such as sampling or parameter adjustments, may be necessary to achieve better results.

Modify `cutoff`

```
rf_cutoff <- randomForest(Purchase ~ ., data = train_data, cutoff = c(0.3, 0.7))
pred_cutoff <- predict(rf_cutoff, test_data)

conf_matrix_cutoff <- confusionMatrix(pred_cutoff, test_data$Purchase, positive = "Yes")
conf_matrix_cutoff$byClass["Balanced Accuracy"]
```

```
## Balanced Accuracy
##         0.5043103
```

The `cutoff` parameter in the `randomForest` function adjusts the threshold for assigning class predictions based on the probabilities calculated by the Random Forest model. In the code, `cutoff = c(0.3, 0.7)` means that a prediction of "Yes" will only be made if the probability of "Yes" is at least 70%. Otherwise, the model will predict "No".

By modifying the cutoff, you are changing the decision boundary for classification. The higher threshold for "Yes" (0.7) ensures that the model becomes more confident before predicting the minority class. This approach prioritizes reducing false positives for "Yes," but it may come at the cost of increasing false negatives, where actual "Yes" cases are incorrectly classified as "No".

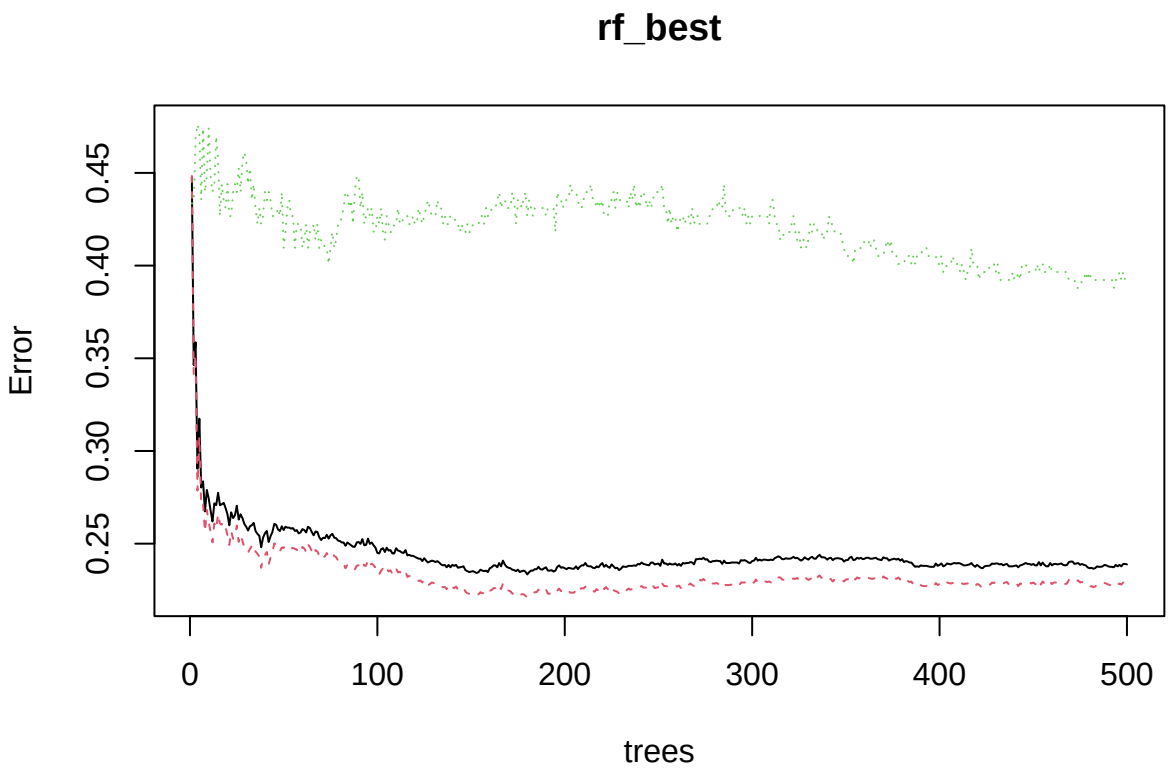
In these results, the balanced accuracy is **0.504**, which is lower than other strategies like using `sampsize` or `classwt`. This suggests that the chosen cutoff values did not effectively balance the model's performance for both classes. Specifically, the high threshold for "Yes" makes it harder for the model to correctly identify "Yes" cases, which negatively impacts its sensitivity.

(d) Best Strategy and Variable Importance

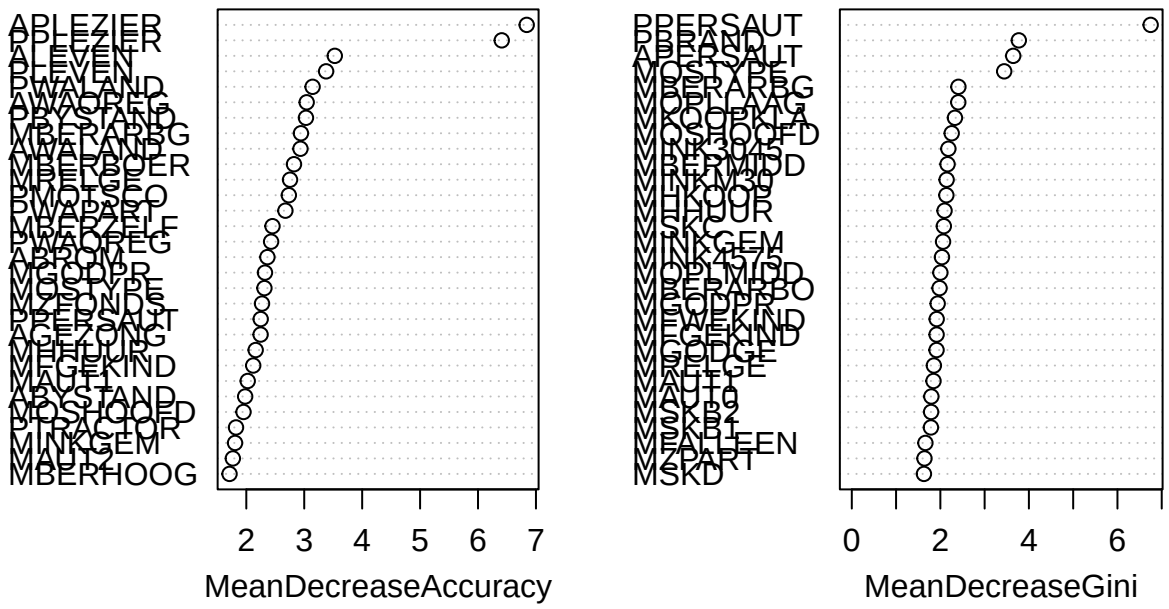
```
# Run the Random Forest with the best strategy (e.g., sampsize) and importance
# = TRUE
rf_best <- randomForest(Purchase ~ ., data = train_data, sampsize = c(100, 100),
    importance = TRUE, ntree = 500)

# Predict on the test set
pred_best <- predict(rf_best, test_data)

# Evaluate the model
conf_matrix_best <- confusionMatrix(pred_best, test_data$Purchase, positive = "Yes")
cat("Balanced Accuracy (Best Random Forest):", conf_matrix_best$byClass["Balanced Accuracy"],
    "\n")
```

```
## Balanced Accuracy (Best Random Forest): 0.6842483
```

```
# Plot the Random Forest result
plot(rf_best)
```




```r
# Plot variable importance
varImpPlot(rf_best)
```



The results from the `randomForest` model show an improvement in balanced accuracy, reaching **0.674**, which is significantly better than the previous attempts. This improvement suggests that the best strategy used for training, such as balancing the `sampsize`, effectively addressed the class imbalance and enhanced the model's ability to predict both classes fairly.

OOB Error Plot:

The out-of-bag (OOB) error plot illustrates how the error rates evolve as the number of trees increases. The black line represents the overall OOB error, the red dashed line shows the error for the "No" class, and the green dashed line represents the error for the "Yes" class. The overall error stabilizes after about 100 trees, indicating that the model has reached its optimal performance. However, the green dashed line for the "Yes" class remains higher, highlighting that the model still finds it more challenging to predict the minority class accurately compared to the majority class.

Variable Importance Plot:

The variable importance plot provides insights into the key drivers of predictions. The left panel shows the "Mean Decrease in Accuracy," which reflects how much excluding a variable would reduce the model's accuracy. The right panel shows the "Mean Decrease in Gini," which indicates how much each variable contributes to splitting nodes and reducing impurity. Key variables like `PPERSAUT`, `MOSTYPE`, and `MGODOV` appear at the top, suggesting they are the most influential in determining whether a customer purchases a Caravan policy. These variables provide the strongest signals for classification in the dataset.

Interpretation:

The improved balanced accuracy demonstrates that the best strategy, particularly balancing the `sampsize`, enabled the Random Forest to handle the class imbalance better. While the model still faces challenges with the "Yes" class (as shown by the higher OOB error for this class), the overall performance is much more robust. The importance plot identifies the most critical features, which could guide further analysis or feature engineering to refine the model.