Data 624 - HW9 (Fall 2024)

Khyati Naik

```
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.3.3
```

```
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3

## Warning: package 'tidyr' was built under R version 4.3.2

## Warning: package 'readr' was built under R version 4.3.2

## Warning: package 'dplyr' was built under R version 4.3.2

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.3.2
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2

## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
```

```
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(party)
```

```
## Warning: package 'party' was built under R version 4.3.3

## Loading required package: grid
## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 4.3.2

## Loading required package: modeltools
## Loading required package: stats4
## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 4.3.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.3.2

##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 4.3.2

##
## Attaching package: 'strucchange'
##
```

```
## The following object is masked from 'package:stringr':
##
##     boundary
##
##
## Attaching package: 'party'
##
## The following object is masked from 'package:dplyr':
##
##     where
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
##
## The following object is masked from 'package:modeltools':
##
##     prior
##
## The following object is masked from 'package:purrr':
##
##     cross
##
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.3
```

```
## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
library(Cubist)
```

```
## Warning: package 'Cubist' was built under R version 4.3.3
```

```
library(rpart)
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.3.3
```

```
##
## Attaching package: 'ranger'
##
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(partykit)
```

```
## Warning: package 'partykit' was built under R version 4.3.3
```

```
## Loading required package: libcoin
```

```
## Warning: package 'libcoin' was built under R version 4.3.3
```

```
##
## Attaching package: 'partykit'
##
## The following objects are masked from 'package:party':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner, node_surv,
##     node_terminal, varimp
```

8.1 Recreate the simulated data from Exercise 7.2:

(a) Fit a random forest model to all of the predictors, then estimate the variable importance scores: Did the random forest model significantly use the uninformative predictors (V6 – V10)?

```
set.seed(200)
simulated <- mlbench.friedman1(200, sd = 1)
simulated <- cbind(simulated$x, simulated$y)
simulated <- as.data.frame(simulated)
colnames(simulated)[ncol(simulated)] <- "y"
```

```
######## a
model1 <- randomForest(y ~ ., data = simulated,
                       importance = TRUE,
                       ntree =1000)
rfimp1 <- varImp(model1, scale = FALSE)
rfimp1
```

```
##          Overall
## V1   8.732235404
## V2   6.415369387
## V3   0.763591825
## V4   7.615118809
## V5   2.023524577
## V6   0.165111172
## V7  -0.005961659
## V8  -0.166362581
## V9  -0.095292651
## V10 -0.074944788
```

(b) Now add an additional predictor that is highly correlated with one of the informative predictors. For example: Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

```
######## b
simulated$duplicate1 <- simulated$V1 +rnorm(200) * .1
cor(simulated$duplicate1, simulated$V1)
```

```
## [1] 0.9460206
```

```
set.seed(1234)
model2 <- randomForest(y ~ ., data = simulated,
                       importance = TRUE,
                       ntree = 1000)
rfImp2 <- varImp(model2, scale = FALSE)
rfImp2
```

```
##                 Overall
## V1          5.199673214
## V2          6.171549032
## V3          0.610201310
## V4          7.011542316
## V5          1.803730354
## V6          0.206916912
## V7         -0.088020631
## V8         -0.073473693
## V9         -0.008437676
## V10        -0.047077870
## duplicate1  4.425668357
```

```
simulated$duplicate2 <- simulated$V1 + rnorm(200) * .1
cor(simulated$duplicate2, simulated$V1)
```

```
## [1] 0.93356
```

There is a noticeable decline in the importance values of each predictor, indicating a balance among them. However, what's particularly interesting is that the duplicate of V1 ranks as the fourth most important predictor at this stage.

(c) Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

```
############## c
model_cfor <- cforest(y ~ ., data = simulated[, c(1:11)])
rfImp_cfor <- varimp(model_cfor, conditional = TRUE)
rfImp_cfor
```

```
##          V1          V2          V3          V4          V5          V6
##  6.00531194  4.82873682 -0.01681998  5.98257066  1.31215958 -0.15202178
##          V7          V8          V9         V10
## -0.23776018 -0.23554349 -0.34774122 -0.23652919
```

The importance scores reveal a similar pattern to the traditional random forest, with variables V1, V2, and V4 standing out as the strongest predictors. Meanwhile, variables V6 through V10 exhibit relatively low importance, suggesting that they would likely be excluded during parameter tuning.

(d) Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur? Fig. 8.24: A comparison of variable importance magnitudes for differing values of the bagging fraction and shrinkage parameters. Both tuning parameters are set to 0.1 in the left figure. Both are set to 0.9 in the right figure

```
############# d
model_gbmG <- expand.grid(interaction.depth = seq(1, 7, by = 2),
                         n.trees = seq(100, 1000, by = 50),
                         shrinkage = c(0.01, 0.1),
                         n.minobsinnode = 10)
set.seed(1234)

gbmTune <- train(y ~ ., data = simulated[, c(1:11)],
                 method = "gbm",
                 tuneGrid = model_gbmG,
                 verbose = FALSE)

rfImp_gbm <- varImp(gbmTune$finalModel, scale = FALSE)

rfImp_gbm
```

```
##        Overall
## V1  40626.8424
## V2  33748.2371
## V3  11046.8163
## V4  44162.4830
## V5  18959.5588
## V6   1558.3342
## V7   1766.9254
## V8    986.6295
## V9    981.1724
## V10   779.2831
```

```
set.seed(1234)
model_cubistT <- train(y ~ ., data = simulated[, c(1:11)], method = "cubist")
rfImp_cubistT <- varImp(model_cubistT$finalModel, scale = FALSE)

rfImp_cubistT
```

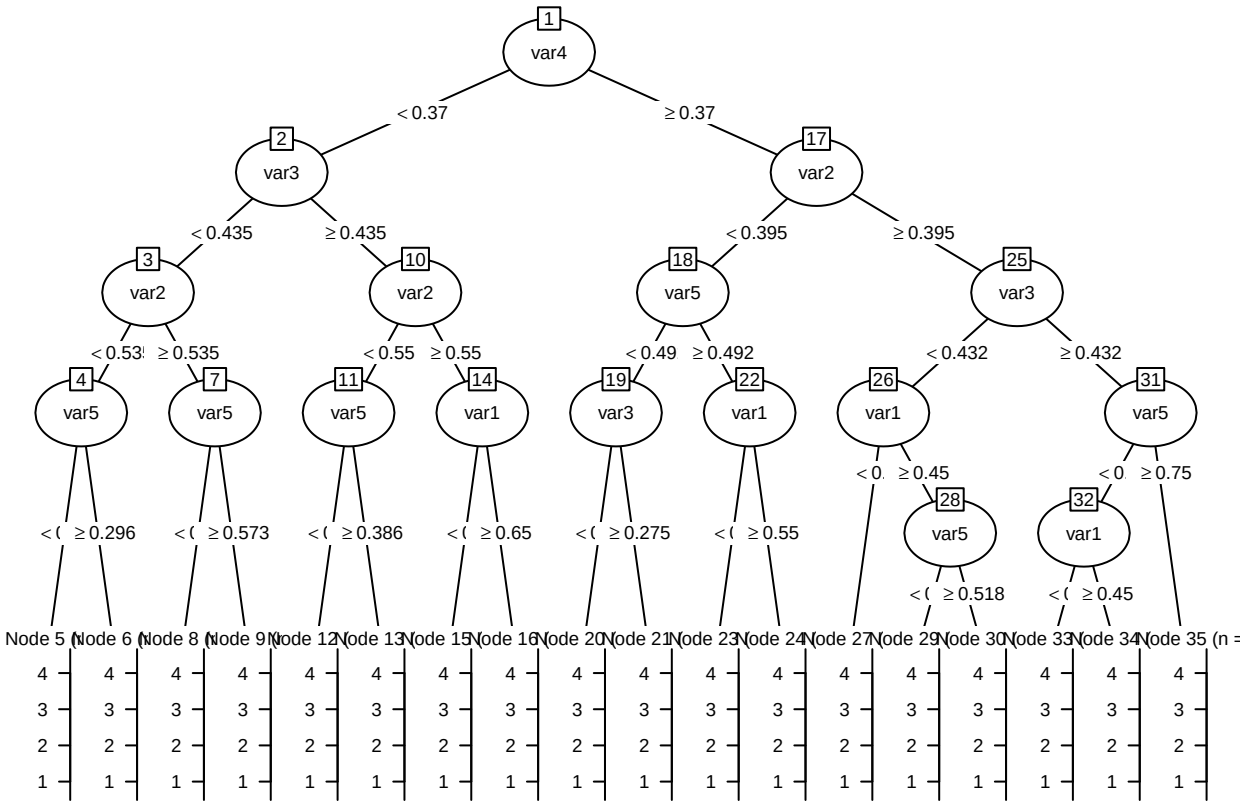
```
##     Overall
## V1     72.0
## V3     42.0
```

```
## V2     54.5
## V4     49.0
## V5     40.0
## V6     11.0
## V7      0.0
## V8      0.0
## V9      0.0
## V10     0.0
```

A similar trend is observed in both the Boosted Trees and Cubist models, where variables V1, V2, and V4 consistently rank as the most influential predictors based on their importance.

8.2. Use a simulation to show tree bias with different granularities.

```r
# Set seed for reproducibility of random data
set.seed(21)

# Generate synthetic data with different levels of granularity
var1 <- sample(seq(0.1, 1, by = 0.1), 500, replace = TRUE)  # Values from 0.1 to 1
var2 <- sample(seq(0.01, 1, by = 0.01), 500, replace = TRUE) # Values from 0.01 to 1
var3 <- sample(seq(0.001, 1, by = 0.001), 500, replace = TRUE) # Values from 0.001 to 1
var4 <- sample(seq(0.0001, 1, by = 0.0001), 500, replace = TRUE) # Values from 0.0001 to 1
var5 <- sample(seq(0.00001, 1, by = 0.00001), 500, replace = TRUE) # Values from 0.00001 to 1

# Create response variable by summing generated variables
response <- var1 + var2 + var3 + var4 + var5

# Assemble data frame
data_frame <- data.frame(var1, var2, var3, var4, var5, response)

# Train a decision tree model using the generated data
model_tree <- rpart(response ~ ., data = data_frame)

# Convert the model for plotting as a party object
plot_tree <- as.party(model_tree)

# Plot the decision tree with customized font size
plot(plot_tree, gp = gpar(fontsize = 7))
```



8.3.

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

(a) Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

The model on the right appears to be overfitting, as it places disproportionate focus on a few predictors. This is reflected in its higher learning rate and bagging fraction. The learning rate determines the proportion of the current prediction added to the previous iteration's prediction, influencing the speed at which the model learns. A higher learning rate increases the weight given to each predictor, leading to quicker adjustments and possibly overfitting. The bagging fraction indicates the proportion of training data used for the model; a higher value suggests more data is used. In contrast, the model on the left has a lower learning rate and bagging fraction, which results in slower learning, better generalization, and a more evenly distributed weight across the predictors. This allows for more balanced model performance, as each predictor is less likely to be overly emphasized.

(b) Which model do you think would be more predictive of other samples?

The model on the left is likely to perform better on unseen data, as it undergoes more iterations, allowing for a more thorough learning process. With each additional iteration, the influence of individual predictors diminishes, leading to a more balanced model. This gradual adjustment helps prevent overfitting, allowing the model to generalize better. As a result, the model on the left is more likely to make accurate predictions on other samples, as it has learned to consider a wider range of factors without overemphasizing specific variables.

(c) How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

Firstly, interaction depth refers to the number of splits made within a tree, or the maximum number of nodes allowed per tree. As the interaction depth increases, the model can capture more complex relationships, which results in greater emphasis on each predictor. This causes the importance of smaller predictors to become more evenly distributed, as the model has more flexibility to account for them. Consequently, the slope of the predictor importance will become steeper for either model, reflecting the higher contribution of each individual predictor to the overall model's predictions.

8.7. Refer to Exercises 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several tree-based models:

```r
# Load dataset and initialize required libraries
data(ChemicalManufacturingProcess)

# Apply KNN imputation to fill missing values
preprocessing_step <- preProcess(ChemicalManufacturingProcess, method = "knnImpute")
processed_data <- predict(preprocessing_step, ChemicalManufacturingProcess)

# Eliminate features with near-zero variance
processed_data <- processed_data %>%
  select_at(vars(-one_of(nearZeroVar(., names = TRUE))))

# Set seed for reproducibility
set.seed(1234)

# Separate features (inputs) and target variable
input_features <- dplyr::select(processed_data, -Yield)
target_variable <- processed_data$Yield

# Split data into training and testing sets
split_indices <- createDataPartition(target_variable, p = 0.8, list = FALSE)
train_inputs <- input_features[split_indices, ] %>% as.matrix()
test_inputs <- input_features[-split_indices, ] %>% as.matrix()
train_target <- target_variable[split_indices]
test_target <- target_variable[-split_indices]
```

(a) Which tree-based regression model gives the optimal resampling and test set performance?

```
# PART A: Training models and evaluating performance

# Build and evaluate a bagging model
set.seed(1234)
bagging_control <- bagControl(fit = ctreeBag$fit, predict = ctreeBag$pred, aggregate = ctreeBag$aggrega
bagging_model <- train(
  x = train_inputs, y = train_target,
  method = "bag",
  bagControl = bagging_control,
  center = TRUE, scale = TRUE,
  trControl = trainControl("cv", number = 10),
  importance = "permutation",
  tuneLength = 25
)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
bagging_predictions <- predict(bagging_model, test_inputs)
postResample(bagging_predictions, test_target)
```

```
##      RMSE  Rsquared       MAE
## 0.5049684 0.6707316 0.4128555
```

```
# Build and evaluate a Random Forest model
set.seed(1234)
random_forest_model <- train(
  x = train_inputs, y = train_target,
  method = "ranger",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10),
  importance = "permutation",
  tuneLength = 25
)
rf_predictions <- predict(random_forest_model, test_inputs)
postResample(rf_predictions, test_target)
```

```
##      RMSE  Rsquared       MAE
## 0.5194250 0.6843342 0.4312098
```

```
# Build and evaluate a Gradient Boosting Machine (GBM) model
gbm_hyperparams <- expand.grid(
  n.trees = c(50, 100),
  interaction.depth = c(1, 5, 10),
  shrinkage = c(0.01, 0.1, 0.2),
  n.minobsinnode = c(5, 10)
)
gbm_model <- train(
  x = train_inputs, y = train_target,
  method = "gbm",
```

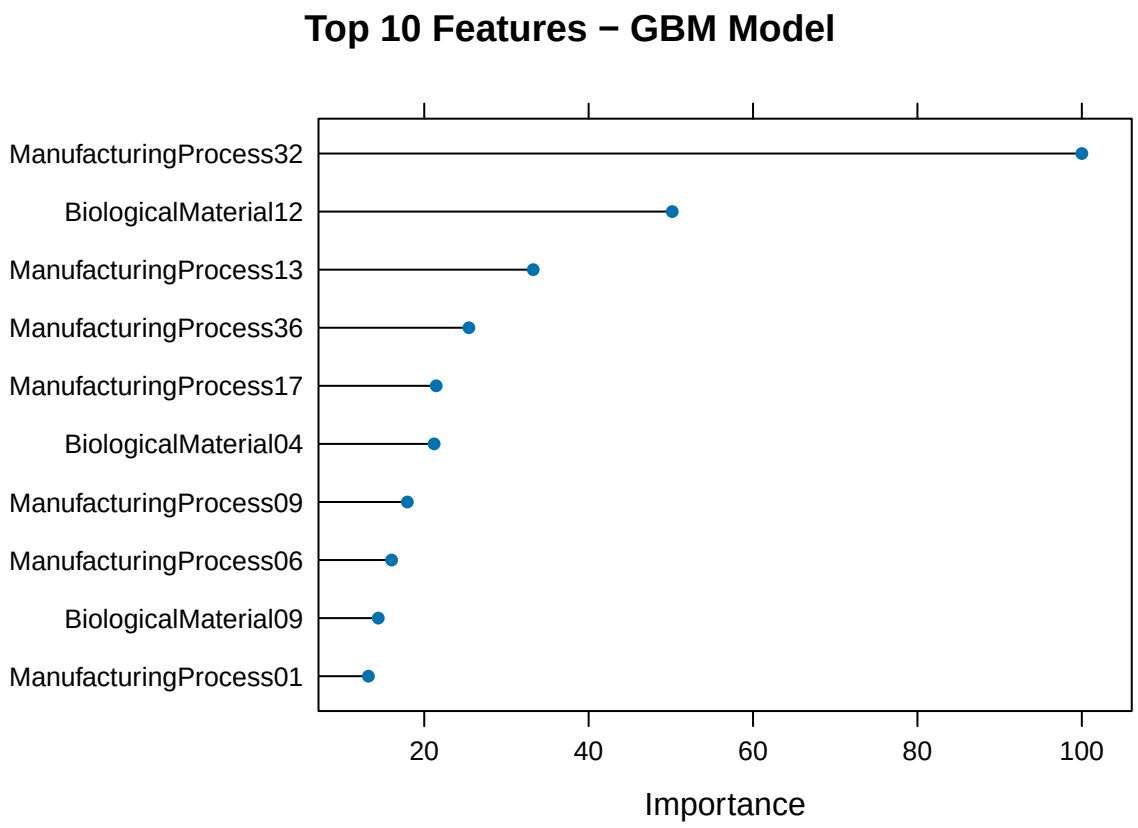
```
  tuneGrid = gbm_hyperparams,
  verbose = FALSE
)
gbm_predictions <- predict(gbm_model, test_inputs)
postResample(gbm_predictions, test_target)
```

```
##      RMSE  Rsquared       MAE
## 0.5430395 0.6109804 0.4144785
```

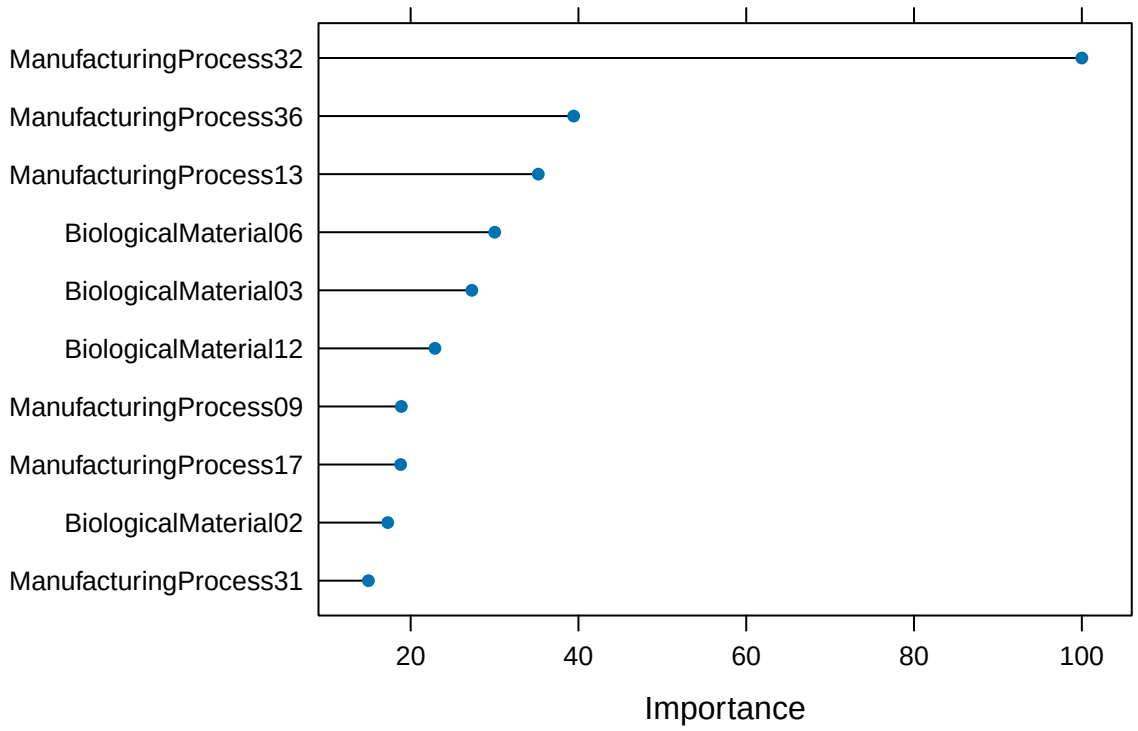
Based on RMSE and R-Squared, GBM outperformed both Bagging and Random Forest models.

(b) Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?

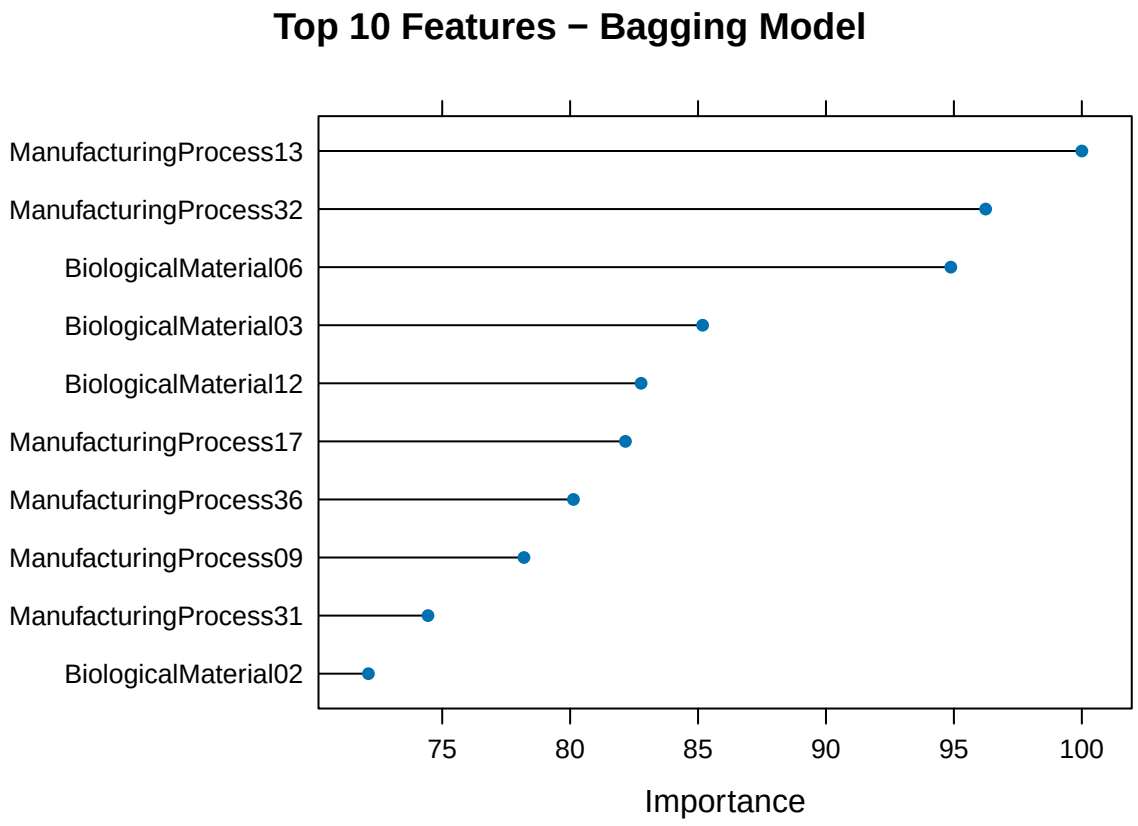
```
# PART B: Feature Importance Analysis

# Visualize top 10 important features for each model
plot(varImp(gbm_model), top = 10, main = "Top 10 Features - GBM Model")
```




```
plot(varImp(random_forest_model), top = 10, main = "Top 10 Features - Random Forest")
```




```
plot(varImp(bagging_model), top = 10, main = "Top 10 Features - Bagging Model")
```



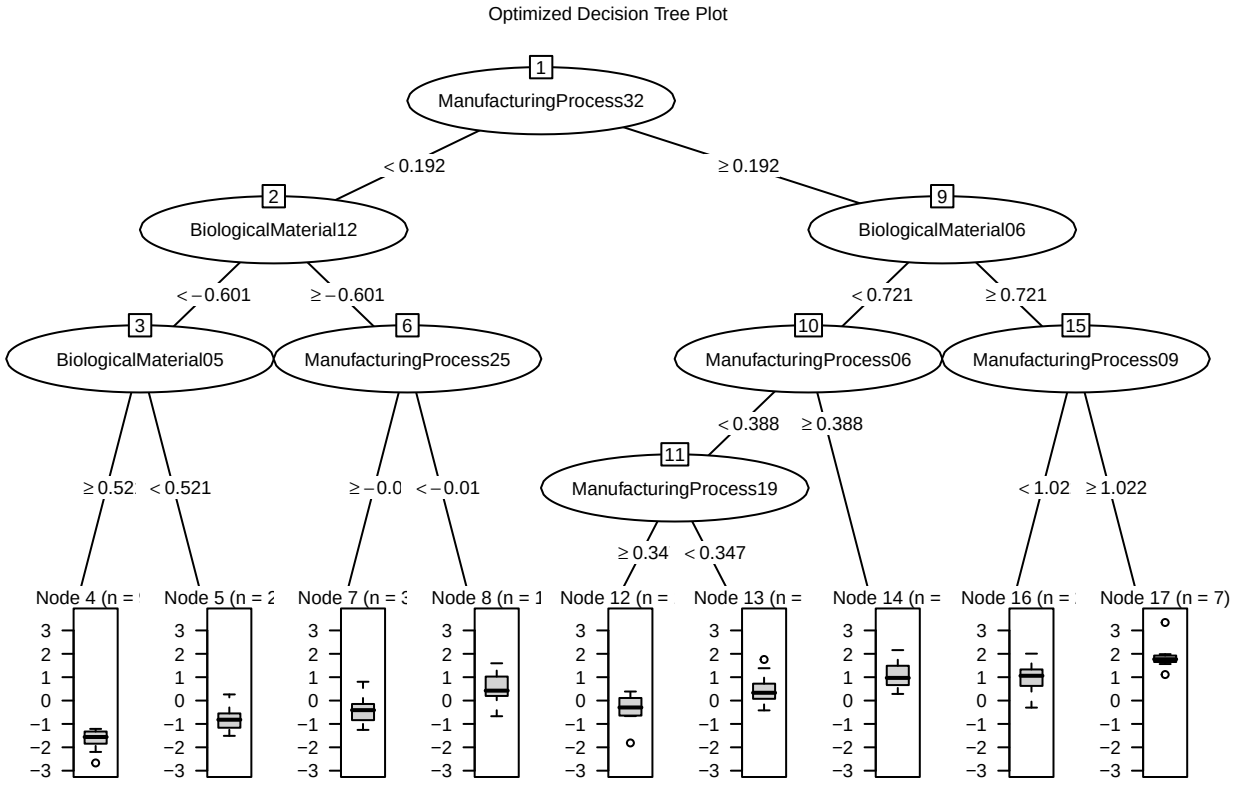
Manufacturing process predictors play a significant role in GBM performance. GBM shows a ratio of 7:3 for Manufacturing Process vs Biological Materials predictors.

(c) Plot the optimal single tree with the distribution of yield in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield?

```
# PART C: Decision Tree Model and Visualization

# Train a single decision tree model
decision_tree_model <- train(
  x = train_inputs, y = train_target,
  method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
# Visualize the final decision tree
optimal_tree <- as.party(decision_tree_model$finalModel)
plot(optimal_tree, main = "Optimized Decision Tree Plot", gp = gpar(fontsize = 7))
```



The decision tree highlights Manufacturing Process 32 as a pivotal factor, with its value influencing the decision path and outcomes.