# Regression Trees and Rule-Based Models

Frederick Jones

2024-04-14

## 8.1. Recreate the sim_dta data from Exercise 7.2:

```
packages <- c("mlbench", "caret", "gbm", "randomForest","pracma","ipred", "AppliedPredictiveModeling","
```

```
## Loading required package: mlbench
```

```
## Loading required package: caret
```

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

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

```
## Loading required package: gbm
```

```
## Loaded gbm 2.1.9
```

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

```
## Loading required package: randomForest
```

```
## randomForest 4.7-1.1
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
## Loading required package: pracma
```

```
## Loading required package: ipred
```

```
## Loading required package: AppliedPredictiveModeling
```

```
## Loading required package: party

## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Loading required package: rpart

## Loading required package: randomForestExplainer

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

## Loading required package: partykit

## Loading required package: libcoin

##
## 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

##                   mlbench                      caret                        gbm
##                      TRUE                       TRUE                       TRUE
##              randomForest                     pracma                      ipred
##                      TRUE                       TRUE                       TRUE
## AppliedPredictiveModeling                      party                    ggplot2
##                      TRUE                       TRUE                       TRUE
##                     rpart               randomForest     randomForestExplainer
##                      TRUE                       TRUE                       TRUE
##                  partykit
##                      TRUE
```

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

### (a) Fit a random forest model to all of the predictors, then estimate the variable importance scores:

```r
use_conditional_true = T # whether to use the conditional argument in the cforest function call


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


model_one = randomForest( y ~ ., data=sim_dta, importance=TRUE, ntree=1000 )
rf_importtance1 = varImp(model_one, scale=FALSE)
rf_importtance1 <- rf_importtance1[order(-rf_importtance1$Overall), , drop = FALSE]
print("randomForest (no correlated predictor)")
```

```
## [1] "randomForest (no correlated predictor)"
```

```r
print(rf_importtance1)
```

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

```r
rf_importtance1
```

```
##          Overall
## V1   8.732235404
## V4   7.615118809
## V2   6.415369387
## V5   2.023524577
## V3   0.763591825
```

```
## V6   0.165111172
## V7  -0.005961659
## V10 -0.074944788
## V9  -0.095292651
## V8  -0.166362581
```

**(b) Now add an additional predictor that is highly correlated with one of the informative predictors. For example:**

```
sim_dta$duplicate1 = sim_dta$V1 + rnorm(200) * 0.1
cor(sim_dta$duplicate1,sim_dta$V1)
```

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

```
model_two = randomForest( y ~ ., data=sim_dta, importance=TRUE, ntree=1000 )
rf_importtance2 = varImp(model_two, scale=FALSE)
rf_importtance2 <- rf_importtance2[order(-rf_importtance2$Overall), , drop = FALSE]
print("randomForest (one correlated predictor)")
```

```
## [1] "randomForest (one correlated predictor)"
```

```
print(rf_importtance2)
```

```
##                Overall
## V4          7.04752238
## V2          6.06896061
## V1          5.69119973
## duplicate1  4.28331581
## V5          1.87238438
## V3          0.62970218
## V6          0.13569065
## V10         0.02894814
## V9          0.00840438
## V7         -0.01345645
## V8         -0.04370565
```

```
sim_dta$duplicate2 = sim_dta$V1 + rnorm(200) * 0.1
cor(sim_dta$duplicate2,sim_dta$V1)
```

```
## [1] 0.9408631
```

```
model_three = randomForest( y ~ ., data=sim_dta, importance=TRUE, ntree=1000 )
rf_importtance3 = varImp(model_three, scale=FALSE)
rf_importtance3 <- rf_importtance3[order(-rf_importtance3$Overall), , drop = FALSE]
print("randomForest (two correlated predictors)")
```

```
## [1] "randomForest (two correlated predictors)"
```

```
print(rf_importtance3)
```

```
##                Overall
## V4          7.04870917
## V2          6.52816504
## V1          4.91687329
## duplicate1  3.80068234
## V5          2.03115561
## duplicate2  1.87721959
## V3          0.58711552
## V6          0.14213148
## V7          0.10991985
## V10         0.09230576
## V9         -0.01075028
## V8         -0.08405687
```

**(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?**

```
sim_dta$duplicate1 = NULL
sim_dta$duplicate2 = NULL

model_one = cforest( y ~ ., data=sim_dta )
cfImp1 = as.data.frame(varimp(model_one),conditional=use_conditional_true)
cfImp1 <- cfImp1[order(-cfImp1$`varimp(model_one)`), , drop = FALSE]
print(sprintf("cforest (no correlated predictor); varimp(*,conditional=%s)",use_conditional_true))
```

```
## [1] "cforest (no correlated predictor); varimp(*,conditional=TRUE)"
```

```
print(cfImp1)
```

```
##     varimp(model_one)
## V1          8.3013758
## V4          7.3074416
## V2          6.3849498
## V5          2.1097038
## V3          0.2021611
## V7          0.1021704
## V9         -0.1127212
## V10        -0.1345254
## V8         -0.1382315
## V6         -0.1612203
```

```r
# Now we add correlated predictors one at a time
sim_dta$duplicate1 = sim_dta$V1 + rnorm(200) * 0.1

model_two = cforest( y ~ ., data=sim_dta )
cfImp2 = as.data.frame(varimp(model_two),conditional=use_conditional_true)
cfImp2 <- cfImp2[order(-cfImp2$`varimp(model_two)`), , drop = FALSE]

print(sprintf("cforest (one correlated predictor); varimp(*,conditional=%s)",use_conditional_true))
```

```
## [1] "cforest (one correlated predictor); varimp(*,conditional=TRUE)"
```

```r
print(cfImp2)
```

```
##            varimp(model_two)
## V1                6.97066651
## V4                6.74683660
## V2                5.76558317
## duplicate1        4.79651139
## V5                2.01720755
## V7                0.25474477
## V3                0.11164501
## V8                0.08391749
## V6               -0.01329420
## V9               -0.01771803
## V10              -0.14461065
```

```r
sim_dta$duplicate2 = sim_dta$V1 + rnorm(200) * 0.1

model_three = cforest( y ~ ., data=sim_dta )
cfImp3 = as.data.frame(varimp(model_three),conditional=use_conditional_true)
cfImp3 <- cfImp3[order(-cfImp3$`varimp(model_three)`), , drop = FALSE]
print(sprintf("cforest (two correlated predictor); varimp(*,conditional=%s)",use_conditional_true))
```

```
## [1] "cforest (two correlated predictor); varimp(*,conditional=TRUE)"
```

```r
print(cfImp3)
```

```
##            varimp(model_three)
## V4                 6.532270768
## V1                 6.473812958
## V2                 5.819611691
## duplicate1         4.183957119
## duplicate2         2.754094974
## V5                 1.963606817
## V3                 0.135392125
## V7                 0.125061550
## V6                 0.065960522
## V8                 0.005934407
## V10               -0.009292788
## V9                -0.161707667
```

**(d) Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?**

```
sim_dta$duplicate1 = NULL
sim_dta$duplicate2 = NULL

model_one = gbm( y ~ ., data=sim_dta, distribution="gaussian", n.trees=1000 )
print(sprintf("gbm (no correlated predictor)"))
```

```
## [1] "gbm (no correlated predictor)"
```

```
print(summary(model_one,plotit=F)) # the summary method gives variable importance ...
```

```
##     var   rel.inf
## V4   V4 24.939515
## V1   V1 24.157793
## V2   V2 19.606749
## V5   V5 11.376158
## V3   V3  9.012202
## V6   V6  3.127628
## V7   V7  2.384312
## V8   V8  1.937256
## V10 V10  1.906294
## V9   V9  1.552092
```

```
# Now we add correlated predictors one at a time
sim_dta$duplicate1 = sim_dta$V1 + rnorm(200) * 0.1

model_two = gbm( y ~ ., data=sim_dta, distribution="gaussian", n.trees=1000 )
print(sprintf("gbm (one correlated predictor)"))
```

```
## [1] "gbm (one correlated predictor)"
```

```
print(summary(model_two,plotit=F))
```

```
##                   var   rel.inf
## V4                 V4 24.876312
## V2                 V2 18.948598
## duplicate1 duplicate1 16.229257
## V5                 V5 11.033707
## V1                 V1  9.396453
## V3                 V3  9.299401
## V7                 V7  2.680668
## V6                 V6  2.446402
## V8                 V8  1.768820
## V9                 V9  1.729158
## V10               V10  1.591224
```

```r
sim_dta$duplicate2 = sim_dta$V1 + rnorm(200) * 0.1

model_three = gbm( y ~ ., data=sim_dta, distribution="gaussian", n.trees=1000 )
print(sprintf("gbm (two correlated predictor)"))
```

```
## [1] "gbm (two correlated predictor)"
```

```r
print(summary(model_three,plotit=F))
```

```
##                    var   rel.inf
## V4                  V4 24.022427
## V2                  V2 20.689180
## duplicate1 duplicate1 12.604438
## V1                  V1 12.318191
## V5                  V5 10.622554
## V3                  V3  8.078223
## V6                  V6  2.450551
## duplicate2 duplicate2  2.440458
## V7                  V7  2.006252
## V8                  V8  1.721103
## V9                  V9  1.696819
## V10                V10  1.349803
```

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

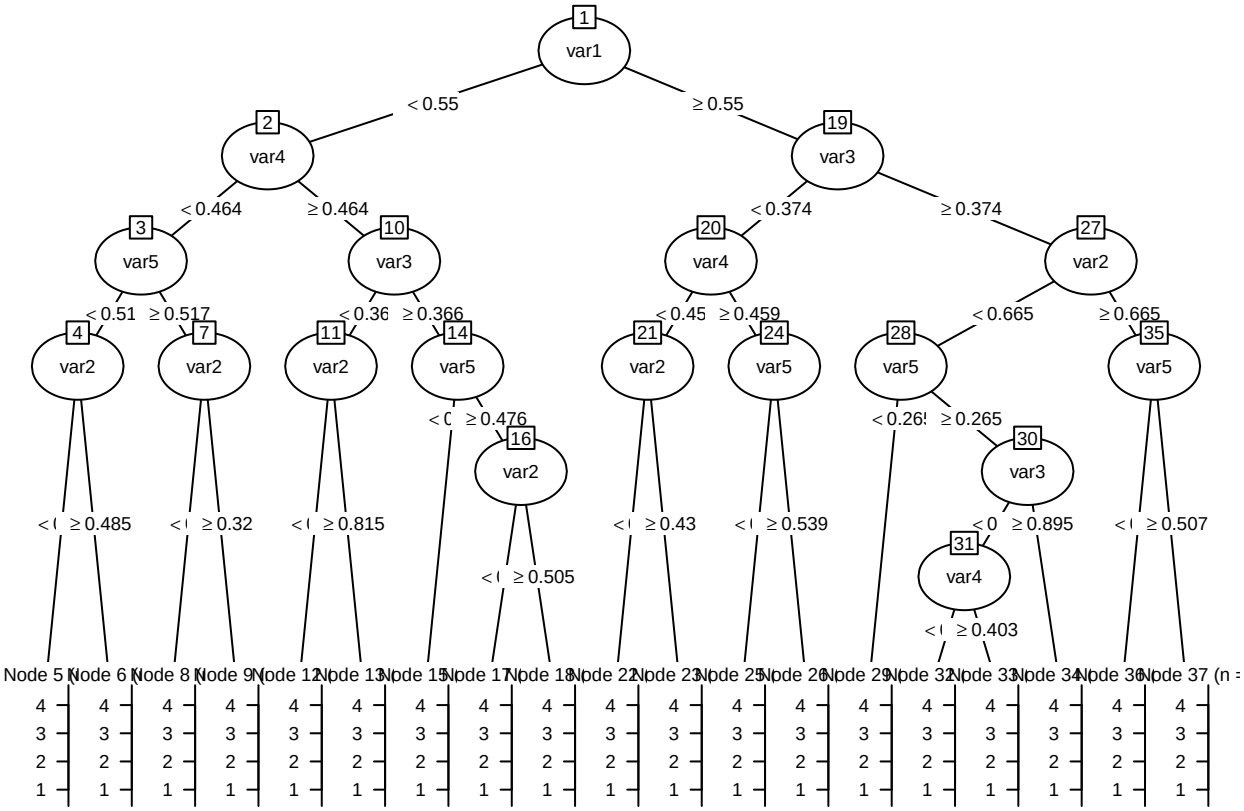
```r
# Set seed for reproducibility
set.seed(624)

# Generate simulated data with different variable names
var1 <- sample(1:10 / 10, 500, replace = TRUE)
var2 <- sample(1:100 / 100, 500, replace = TRUE)
var3 <- sample(1:1000 / 1000, 500, replace = TRUE)
var4 <- sample(1:10000 / 10000, 500, replace = TRUE)
var5 <- sample(1:100000 / 100000, 500, replace = TRUE)
response <- var1 + var2 + var3 + var4 + var5
simData <- data.frame(var1, var2, var3, var4, var5, response)

# Fit a decision tree model
rpartTree <- rpart(response ~ ., data = simData)

# Convert the rpart model to a party object
rpart_party <- as.party(rpartTree)

# Plot the decision tree
plot(rpart_party, 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 bagging fraction represents the proportion of training data used, while the learning rate determines the impact of current predictions on subsequent iterations. Optimal performance often involves a lower learning rate, allowing for more iterations. The left model, with reduced bagging fraction and learning rate, learns slowly and requires more computation but tends**

to perform better. It utilizes a smaller subset of data and promotes better generalization. Conversely, the right model, with higher values for both parameters, likely overfits the data. It learns faster, using more training data per iteration, which may lead to disproportionate emphasis on select predictors. This rapid learning process can result in a narrowed focus on a subset of predictors, potentially hindering predictive performance on unseen data.

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

The left model's predictive capability surpasses that of the right model due to its extensive iterations, leading to reduced weight on individual predictors. This characteristic fosters enhanced generalization, resulting in superior accuracy when applied to unseen samples.

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

Increasing the interaction depth, which denotes the number of splits or maximum nodes per tree, amplifies the importance of predictors. This heightened significance enables smaller yet influential predictors to contribute more substantially to the model's decision-making process. Consequently, the slope of predictor importance escalates, becoming steeper with each increment in interaction depth.

## 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:

```
set.seed(100)

data(ChemicalManufacturingProcess)

# imputation
miss <- preProcess(ChemicalManufacturingProcess, method = "bagImpute")
Chemical <- predict(miss, ChemicalManufacturingProcess)

# filtering low frequencies
Chemical <- Chemical[, -nearZeroVar(Chemical)]

set.seed(624)

# index for training
index <- createDataPartition(Chemical$Yield, p = .8, list = FALSE)

# train
train_x <- Chemical[index, -1]
train_y <- Chemical[index, 1]

# test
```

```
test_x <- Chemical[-index, -1]
test_y <- Chemical[-index, 1]
```

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

**The lowest RMSE is found in the cubist model, giving the best optimal resampling and test set performance.**

```
# single tree cart
set.seed(100)

cartTune <- train(train_x, train_y,
                  method = "rpart",
                  tuneLength = 10,
                  trControl = trainControl(method = "cv"))
```

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

```
cartPred <- predict(cartTune, test_x)

postResample(cartPred, test_y)
```

```
##      RMSE  Rsquared       MAE
## 1.5632389 0.5035625 1.1913334
```

```
# bagged tree

set.seed(100)

baggedTree <- ipredbagg(train_y, train_x)

baggedPred <- predict(baggedTree, test_x)

postResample(baggedPred, test_y)
```

```
##      RMSE  Rsquared       MAE
## 1.2311358 0.7141633 0.8911824
```

```
# random forest
set.seed(100)

rfModel <- randomForest(train_x, train_y,
                        importance = TRUE,
                        ntree = 1000)


rfPred <- predict(rfModel, test_x)

postResample(rfPred, test_y)
```

```
##      RMSE  Rsquared       MAE
## 1.2598015 0.7216121 0.8998200
```

```
# boosted tree
gbmGrid <- 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(100)

gbmTune <- train(train_x, train_y,
                 method = "gbm",
                 tuneGrid = gbmGrid,
                 verbose = FALSE)

gbmPred <- predict(gbmTune, test_x)

postResample(gbmPred, test_y)
```

```
##      RMSE  Rsquared       MAE
## 1.1824945 0.7195344 0.8678498
```

```
# cubist
set.seed(100)
cubistTuned <- train(train_x, train_y,
                     method = "cubist")

cubistPred <- predict(cubistTuned, test_x)

postResample(cubistPred, test_y)
```

```
##      RMSE  Rsquared       MAE
## 0.9804618 0.7964499 0.7363520
```

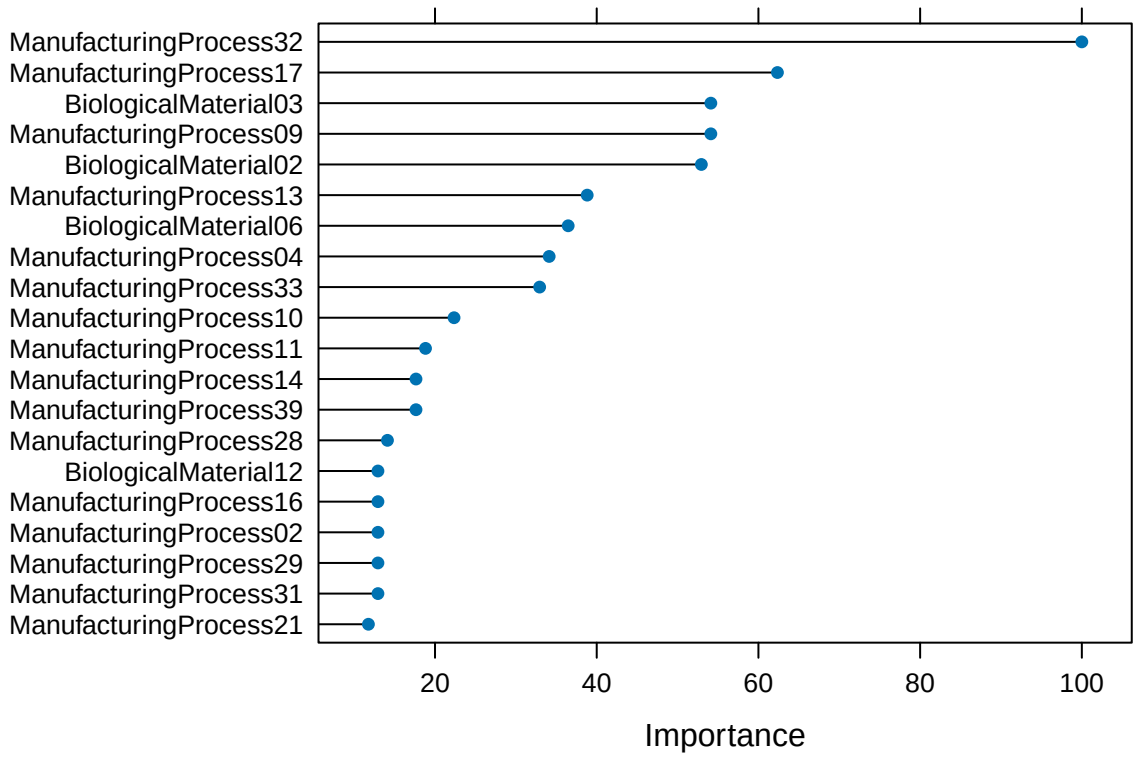
**(b) Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?**

```
varImp(cubistTuned, scale = TRUE)
```

```
## cubist variable importance
##
##   only 20 most important variables shown (out of 56)
##
##                         Overall
## ManufacturingProcess32  100.00
## ManufacturingProcess17   62.35
## ManufacturingProcess09   54.12
## BiologicalMaterial03     54.12
```

```
## BiologicalMaterial02     52.94
## ManufacturingProcess13   38.82
## BiologicalMaterial06     36.47
## ManufacturingProcess04   34.12
## ManufacturingProcess33   32.94
## ManufacturingProcess10   22.35
## ManufacturingProcess11   18.82
## ManufacturingProcess14   17.65
## ManufacturingProcess39   17.65
## ManufacturingProcess28   14.12
## ManufacturingProcess16   12.94
## BiologicalMaterial12     12.94
## ManufacturingProcess31   12.94
## ManufacturingProcess29   12.94
## ManufacturingProcess02   12.94
## ManufacturingProcess21   11.76
```

```
plot(varImp(cubistTuned), top = 20)
```




**The manufacturing process variables dominate the list at a ratio of 16:4, whereas the optimal linear and nonlinear models had ratios of 11:9**

For the tree-based model, only 3 are biological variables out of the top 10, compared to 4 in the linear and nonlinear models. `ManufactingProcess32` still is deemed the most important. The other predictors have less variable importance. `BiologicalMaterial06` was deemed only the seventh most important, where it was the second most important in other variables. There are some predictors that were not in the top 10 previously, that are in the top 10 now, such as Manufacting Processes number 17, 4, 33, and 10.

## (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?

Yes, this model does provide detailed insights into the biological processes, offering comprehensive information to enhance understanding.

```
rpartTree <- rpart(Yield ~ ., data = Chemical[index, ])

plot(as.party(rpartTree), ip_args = list(abbreviate = 4), gp = gpar(fontsize = 7))
```