Regression Tree and Rule Based Models

Shovan Biswas

2020/11/21

Libraries

```
library(tidyverse)
library(kableExtra)
library(corrplot)
library(reshape2)
library(Amelia)
library(dlookr)
library(fpp2)
library(plotly)
library(gridExtra)
library(readxl)
library(ggplot2)
library(urca)
library(tseries)
library(AppliedPredictiveModeling)
library(RANN)
library(psych)
library(e1071)
library(corrplot)
library(glmnet)
library(mlbench)
library(caret)
library(earth)
library(randomForest)
library(party)
library(Cubist)
library(gbm)
library(rpart)
```

Exercise 8.1

Recreate the simulated data from Exercise 7.2:

(This exercise is based on library(mlbench), which I included in libraries at the top.)

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

(This exercise is based on library(randomForest) and library(caret), which I included in libraries at the top.)

```
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)
rfImp1 <- varImp(model1, scale = FALSE)
```

Did the random forest model significantly use the uninformative predictors (V6 – V10)?

```
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
```

Almost 0 to negative values evince that Random Forest didn't use the predictors (V6 thru V10).

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

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

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

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?

```
model_new <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)
rfImp_new <- varImp(model_new, scale = FALSE)

rfImp_new
```

```
##               Overall
## V1         5.69119973
## V2         6.06896061
```

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

V1's score indeed changed. While the score of V1 reduced from 8.732235404 to 5.69119973. importance of highly correlated duplicate1 is the least.

Here's the answer to the last question. quoted from text book:

"An advantage of tree-based models is that, when the tree is not large, the model is simple and interpretable. Also, this type of tree can be computed quickly (despite using multiple exhaustive searches). Tree models intrinsically conduct feature selection; if a predictor is never used in a split, the prediction equation is independent of these data. **This advantage is weakened when there are highly correlated predictors. If two predictors are extremely correlated, the choice of which to use in a split is somewhat random.**"

(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?

(Requires library(party), which I included in libraries at the top.)

```
cforestModel <- cforest(y ~ ., data = simulated)
sort(varimp(cforestModel, conditional = FALSE), decreasing = TRUE)
```

```
##            V4            V2    duplicate1            V1            V5
##  7.6223892727  6.0579730772  5.0941897280  4.6171158805  1.7161194047
##            V7            V9            V3            V6           V10
##  0.0465374951  0.0046062409  0.0003116115 -0.0289427183 -0.0310326410
##            V8
## -0.0380965511
```

```
sort(varimp(cforestModel, conditional = TRUE), decreasing = TRUE)
```

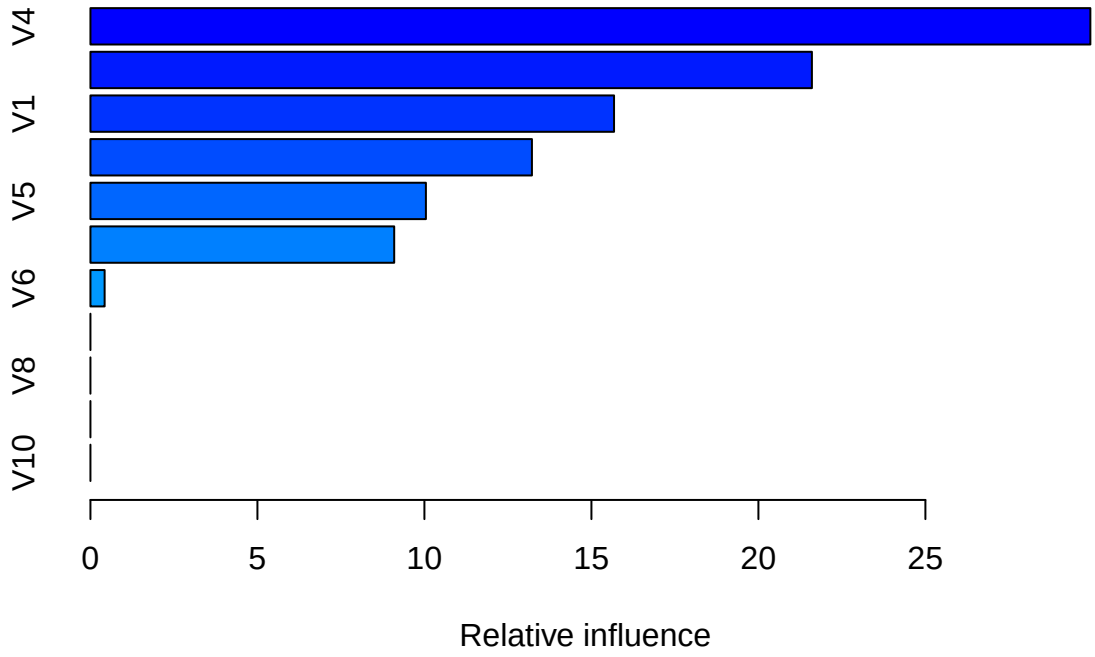
```
##           V4           V2   duplicate1           V1           V5           V6
##  6.190578351  4.688980360  1.926751729  1.807953145  1.051666850  0.028174759
##           V9           V3           V8           V7          V10
##  0.015118505  0.012878752 -0.004356587 -0.011709437 -0.022190587
```

By setting conditional = TRUE, it took longer to process and the results were different. The uninformative predictors (V6 thru V10) remain the same. By setting conditional = TRUE, V3 has become become uninformative.

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

(Requires library(gbm), which I included in libraries at the top.)

```
gbmModel <- gbm(y ~ ., data = simulated, distribution = 'gaussian')    # refer page 216, under "Boosted
summary(gbmModel)
```




```
##                    var    rel.inf
## V4                  V4 29.9389549
## V2                  V2 21.6012918
## V1                  V1 15.6771729
## duplicate1 duplicate1 13.2189845
## V5                  V5 10.0440211
## V3                  V3  9.0948053
## V6                  V6  0.4247697
## V7                  V7  0.0000000
## V8                  V8  0.0000000
## V9                  V9  0.0000000
## V10                V10  0.0000000
```

For Boosted Trees, I used gbm(). Same patten occurs. V4 is still the highest and V6 thru V10 are still low.

(Requires library(Cubist), which I included in libraries at the top.)

```
cubistModel <- cubist(x = simulated[, -(ncol(simulated) - 1)], y = simulated$y)    # refer page 217, und
varImp(cubistModel)
```

```
##             Overall
## V1               50
## V2               50
## V4               50
## V5               50
## duplicate1       50
## V3                0
## V6                0
## V7                0
## V8                0
## V9                0
## V10               0
```

For Cubist, I used cubist(). Same patten occurs. Here also V6 thru V10 remains the lowest in importance. Only duplicate1 has moved up by one notche.

Exercise 8.2

Use a simulation to show tree bias with different granularities.

We'll use rpart package, which is One of the widely used implementations for single regression trees in R. The function rpart splits based on CART methodology.

First let's create some samples and collect them into variables x1, x2, x3, x4. Then we'll create a dataframe of 150 observations of x1 thru x4 and the response variable y.

(Requires library(rpart), which I included in libraries at the top.)

```
set.seed(123)
x1 <- sample(0:10000 / 10000, 150)                    # 150 possible values in x1.
x2 <- sample(0:1000 / 1000, 150)                      # 150 possible values in x2.
x3 <- sample(0:100 / 100, 150, replace = TRUE)        # 100 possible values in x3, but with replacement
x4 <- sample(0:10 / 10, 150, replace = TRUE)          # 10  possible values in x4, but with replacement

y <- x1 + x2

tree_bias <- data.frame(x1, x2, x3, x4, y)

head(tree_bias)
```

```
##       x1    x2   x3  x4      y
## 1 0.2462 0.184 0.29 1.0 0.4302
## 2 0.2510 0.764 0.88 0.4 1.0150
## 3 0.8717 0.412 0.49 0.8 1.2837
## 4 0.2985 0.626 0.46 0.6 0.9245
## 5 0.1841 0.521 0.66 0.2 0.7051
## 6 0.9333 0.308 0.81 0.6 1.2413
```

Now, we'll run the run rpart and find the variables of importance.

Note: I tried function **ctree()** of the same rpart package rpart. It worked, but the function varImp() failed on its output.

```
rpartTree <- rpart(y ~ ., data = tree_bias)
varImp(rpartTree)
```

```
##      Overall
## x1 2.7986562
## x2 3.6019240
## x3 0.5142012
## x4 0.3309620
```

The order x1, x2, x3, x4 expresses the degree of granularity. Variable x1 is most split and x4 is least split.

Exercise 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?

Bagging fraction of 0.1 means only 10% of the full data is randomly sampled. So, each tree may be built with different datasets altogether. Implies more variance. Therefore, the trees split very differently from each other. If the bagging is 0.9, in each run same dataset or almost same dataset is exposed to the trees. So, less variance. So, lesser splits.

(Please refer page 207, 3rd para of textbook.)

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

Based on the variance argument in (a) above, I would think that a model with a learning rate of 0.1 is more predictive.

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

As interaction depth is increased, the trees grow more deep. This causes more predictors to split. Therefore, vairable importance gets distributed over more variables, rather than less.

Exercise 8.7

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:

Below portion was brought forward from 7.5:

(Requires library(AppliedPredictiveModeling), which I included in libraries at the top.)

Reading data:

```
data(ChemicalManufacturingProcess)
dim(ChemicalManufacturingProcess)
```

```
## [1] 176  58
```

Imputation:

```
data_imputed <- mice(ChemicalManufacturingProcess, m = 1, method = "pmm", print = F) %>% complete()
```

```
## Warning: Number of logged events: 135
```

Checking the data:

```
any(is.na(data_imputed))
```

```
## [1] FALSE
```

At this point, the data is imputed. I'll proceed to Box-Cox it.

```
# Initially identifying columns, whose skewness are not less than 1.
transform_cols <- c()

for(i in seq(from = 1, to = length(data_imputed), by = 1)) {
  if(abs(skewness(data_imputed[, i])) >= 1) {
    transform_cols <- append(transform_cols, names(data_imputed[i]))
  }
}

# Applying Box-cox.
lambda <- NULL
data_imputed_2 <- data_imputed

for (i in 1:length(transform_cols)) {
  lambda[transform_cols[i]] <- BoxCox.lambda(abs(data_imputed[, transform_cols[i]]))

  data_imputed_2[c(transform_cols[i])] <- BoxCox(data_imputed[transform_cols[i]], lambda[transform_cols
}
```

At this point, the data is pre-processed. The pre-processed data is stored in the variable data_imputed_2.

So, I'll proceed to split the data into train and test in 80/20 ratio.

```
set.seed(200)

split_index <- createDataPartition(data_imputed_2$Yield, p = 0.8, list = FALSE)
X_train <- data_imputed_2[split_index, ]
```

```
y_train <- data_imputed_2$Yield[split_index]

X_test <- data_imputed_2[-split_index, ]
y_test <- data_imputed_2$Yield[-split_index]
```

At this point, the data is split and we can proceed to create 4 models gbm, rpart, cubist, rf.

```
set.seed(200)

grid <- expand.grid(n.trees = c(50, 75, 100, 150), interaction.depth = c(1, 5, 10, 12), shrinkage = c(0

gbmModel <- train(x = X_train, y = y_train, method = 'gbm', tuneGrid = grid, verbose = F)

gbmModel
```

```
## Stochastic Gradient Boosting
##
## 144 samples
##  58 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  RMSE       Rsquared
##   0.01        1                  5               50      1.3334748  0.8412630
##   0.01        1                  5               75      1.1647161  0.8760667
##   0.01        1                  5              100      1.0285470  0.8987778
##   0.01        1                  5              150      0.8212163  0.9300194
##   0.01        1                 10               50      1.3328735  0.8432358
##   0.01        1                 10               75      1.1638989  0.8768405
##   0.01        1                 10              100      1.0265276  0.8991673
##   0.01        1                 10              150      0.8198850  0.9292761
##   0.01        1                 12               50      1.3346774  0.8432172
##   0.01        1                 12               75      1.1661653  0.8747017
##   0.01        1                 12              100      1.0289114  0.8977553
##   0.01        1                 12              150      0.8210018  0.9269524
##   0.01        5                  5               50      1.1524660  0.9739079
##   0.01        5                  5               75      0.9251154  0.9765199
##   0.01        5                  5              100      0.7485161  0.9783904
##   0.01        5                  5              150      0.5073922  0.9808669
##   0.01        5                 10               50      1.1678124  0.9572306
##   0.01        5                 10               75      0.9489222  0.9599672
##   0.01        5                 10              100      0.7810422  0.9618883
##   0.01        5                 10              150      0.5595947  0.9638283
##   0.01        5                 12               50      1.1838570  0.9470583
##   0.01        5                 12               75      0.9695403  0.9504364
##   0.01        5                 12              100      0.8047813  0.9526929
##   0.01        5                 12              150      0.5888558  0.9554000
##   0.01       10                  5               50      1.1353258  0.9785228
##   0.01       10                  5               75      0.9051400  0.9794628
##   0.01       10                  5              100      0.7284467  0.9803381
```

```
##   0.01       10                5          150      0.4926996  0.9814914
##   0.01       10               10           50      1.1684580  0.9567983
##   0.01       10               10           75      0.9499491  0.9602434
##   0.01       10               10          100      0.7814004  0.9619300
##   0.01       10               10          150      0.5600177  0.9639247
##   0.01       10               12           50      1.1817895  0.9477460
##   0.01       10               12           75      0.9683293  0.9505827
##   0.01       10               12          100      0.8034433  0.9523328
##   0.01       10               12          150      0.5883808  0.9549341
##   0.01       12                5           50      1.1344439  0.9786862
##   0.01       12                5           75      0.9041795  0.9798329
##   0.01       12                5          100      0.7273214  0.9804992
##   0.01       12                5          150      0.4910085  0.9813899
##   0.01       12               10           50      1.1678821  0.9578659
##   0.01       12               10           75      0.9493406  0.9603834
##   0.01       12               10          100      0.7807940  0.9620672
##   0.01       12               10          150      0.5605604  0.9640273
##   0.01       12               12           50      1.1839320  0.9466312
##   0.01       12               12           75      0.9696768  0.9503426
##   0.01       12               12          100      0.8060800  0.9520609
##   0.01       12               12          150      0.5898923  0.9550377
##   0.10        1                5           50      0.2936907  0.9798040
##   0.10        1                5           75      0.2443969  0.9831820
##   0.10        1                5          100      0.2366008  0.9840742
##   0.10        1                5          150      0.2331726  0.9843386
##   0.10        1               10           50      0.3572961  0.9659589
##   0.10        1               10           75      0.3348996  0.9678647
##   0.10        1               10          100      0.3283857  0.9686789
##   0.10        1               10          150      0.3253717  0.9689037
##   0.10        1               12           50      0.3953389  0.9576123
##   0.10        1               12           75      0.3763772  0.9591645
##   0.10        1               12          100      0.3692784  0.9598912
##   0.10        1               12          150      0.3672561  0.9598050
##   0.10        5                5           50      0.2362221  0.9833287
##   0.10        5                5           75      0.2300090  0.9841540
##   0.10        5                5          100      0.2269624  0.9847007
##   0.10        5                5          150      0.2239413  0.9851870
##   0.10        5               10           50      0.3318170  0.9676847
##   0.10        5               10           75      0.3253921  0.9687554
##   0.10        5               10          100      0.3233826  0.9690254
##   0.10        5               10          150      0.3201334  0.9696156
##   0.10        5               12           50      0.3705607  0.9598513
##   0.10        5               12           75      0.3636040  0.9605946
##   0.10        5               12          100      0.3594370  0.9612124
##   0.10        5               12          150      0.3586719  0.9612741
##   0.10       10                5           50      0.2357496  0.9838457
##   0.10       10                5           75      0.2301403  0.9846773
##   0.10       10                5          100      0.2265160  0.9852456
##   0.10       10                5          150      0.2238453  0.9856151
##   0.10       10               10           50      0.3284927  0.9686056
##   0.10       10               10           75      0.3212666  0.9694754
##   0.10       10               10          100      0.3190434  0.9699138
##   0.10       10               10          150      0.3166766  0.9702861
##   0.10       10               12           50      0.3764311  0.9592786
```

```
##   0.10       10                12                75       0.3715844  0.9593189
##   0.10       10                12               100       0.3685894  0.9597218
##   0.10       10                12               150       0.3677138  0.9598439
##   0.10       12                 5                50       0.2337229  0.9837290
##   0.10       12                 5                75       0.2271879  0.9847674
##   0.10       12                 5               100       0.2245629  0.9852606
##   0.10       12                 5               150       0.2225059  0.9855353
##   0.10       12                10                50       0.3334563  0.9671855
##   0.10       12                10                75       0.3273561  0.9680333
##   0.10       12                10               100       0.3246826  0.9685213
##   0.10       12                10               150       0.3237135  0.9688295
##   0.10       12                12                50       0.3797383  0.9579441
##   0.10       12                12                75       0.3733181  0.9584547
##   0.10       12                12               100       0.3724105  0.9585117
##   0.10       12                12               150       0.3718095  0.9586126
##   0.50        1                 5                50       0.4811783  0.9326124
##   0.50        1                 5                75       0.4908077  0.9308890
##   0.50        1                 5               100       0.4892604  0.9312377
##   0.50        1                 5               150       0.4923780  0.9300853
##   0.50        1                10                50       0.5119388  0.9209790
##   0.50        1                10                75       0.5130523  0.9199351
##   0.50        1                10               100       0.5176026  0.9184488
##   0.50        1                10               150       0.5201601  0.9188004
##   0.50        1                12                50       0.5521536  0.9079652
##   0.50        1                12                75       0.5605528  0.9054944
##   0.50        1                12               100       0.5649505  0.9045156
##   0.50        1                12               150       0.5682331  0.9031914
##   0.50        5                 5                50       0.3236012  0.9691355
##   0.50        5                 5                75       0.3234816  0.9691670
##   0.50        5                 5               100       0.3234727  0.9691512
##   0.50        5                 5               150       0.3234535  0.9691497
##   0.50        5                10                50       0.4307592  0.9454819
##   0.50        5                10                75       0.4328607  0.9450810
##   0.50        5                10               100       0.4321453  0.9452323
##   0.50        5                10               150       0.4317943  0.9453308
##   0.50        5                12                50       0.4842247  0.9304511
##   0.50        5                12                75       0.4881811  0.9294782
##   0.50        5                12               100       0.4898547  0.9291629
##   0.50        5                12               150       0.4907817  0.9289711
##   0.50       10                 5                50       0.2921298  0.9751192
##   0.50       10                 5                75       0.2913356  0.9752850
##   0.50       10                 5               100       0.2903032  0.9754518
##   0.50       10                 5               150       0.2898354  0.9755256
##   0.50       10                10                50       0.4256034  0.9467556
##   0.50       10                10                75       0.4244085  0.9469672
##   0.50       10                10               100       0.4248693  0.9467769
##   0.50       10                10               150       0.4244551  0.9468050
##   0.50       10                12                50       0.4741830  0.9330151
##   0.50       10                12                75       0.4768105  0.9324154
##   0.50       10                12               100       0.4776544  0.9322679
##   0.50       10                12               150       0.4777986  0.9322813
##   0.50       12                 5                50       0.2729111  0.9780828
##   0.50       12                 5                75       0.2723561  0.9781446
##   0.50       12                 5               100       0.2719424  0.9782056
```

```
##   0.50       12                  5              150      0.2717809  0.9782278
##   0.50       12                 10               50      0.4048474  0.9515717
##   0.50       12                 10               75      0.4064685  0.9512835
##   0.50       12                 10              100      0.4065397  0.9512387
##   0.50       12                 10              150      0.4064305  0.9512563
##   0.50       12                 12               50      0.4694591  0.9333812
##   0.50       12                 12               75      0.4696353  0.9333073
##   0.50       12                 12              100      0.4701573  0.9332432
##   0.50       12                 12              150      0.4700692  0.9332664
##   MAE
##   0.9972588
##   0.8402138
##   0.7181118
##   0.5443798
##   0.9956266
##   0.8384022
##   0.7155531
##   0.5420117
##   0.9979716
##   0.8405037
##   0.7173288
##   0.5404612
##   0.9228345
##   0.7271286
##   0.5732171
##   0.3582141
##   0.9199582
##   0.7241621
##   0.5713189
##   0.3578076
##   0.9243019
##   0.7286976
##   0.5751984
##   0.3626416
##   0.9106590
##   0.7129780
##   0.5591873
##   0.3462158
##   0.9201677
##   0.7255527
##   0.5715318
##   0.3583674
##   0.9231881
##   0.7282906
##   0.5742744
##   0.3619506
##   0.9100881
##   0.7122995
##   0.5582796
##   0.3445451
##   0.9199518
##   0.7248053
##   0.5714098
##   0.3587380
```

```
##   0.9250083
##   0.7301212
##   0.5767313
##   0.3638481
##   0.1506491
##   0.1301999
##   0.1339858
##   0.1378815
##   0.1801545
##   0.1818028
##   0.1888550
##   0.1985780
##   0.2069172
##   0.2135533
##   0.2203016
##   0.2313061
##   0.1170914
##   0.1207641
##   0.1228793
##   0.1250904
##   0.1729033
##   0.1820807
##   0.1876451
##   0.1940836
##   0.2022892
##   0.2122843
##   0.2180688
##   0.2260743
##   0.1135625
##   0.1174501
##   0.1193070
##   0.1226151
##   0.1711917
##   0.1800427
##   0.1864034
##   0.1937695
##   0.2072316
##   0.2163768
##   0.2227239
##   0.2307909
##   0.1116458
##   0.1155826
##   0.1177711
##   0.1207893
##   0.1741870
##   0.1841861
##   0.1892288
##   0.1968267
##   0.2089648
##   0.2192821
##   0.2267146
##   0.2344446
##   0.3700828
##   0.3803340
```

```
##   0.3799930
##   0.3836187
##   0.3936825
##   0.3970440
##   0.4022951
##   0.4048503
##   0.4224319
##   0.4297305
##   0.4354237
##   0.4399689
##   0.2327603
##   0.2331205
##   0.2331770
##   0.2332689
##   0.3165785
##   0.3192043
##   0.3198351
##   0.3204009
##   0.3564340
##   0.3599776
##   0.3618885
##   0.3633809
##   0.1927364
##   0.1927686
##   0.1927875
##   0.1927866
##   0.3060413
##   0.3076266
##   0.3088848
##   0.3093541
##   0.3514483
##   0.3555437
##   0.3572342
##   0.3586812
##   0.1870112
##   0.1871933
##   0.1871713
##   0.1871566
##   0.2949677
##   0.2964810
##   0.2970528
##   0.2973129
##   0.3430971
##   0.3450868
##   0.3456186
##   0.3461288
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  12, shrinkage = 0.1 and n.minobsinnode = 5.
```

```
set.seed(200)

rpartModel <- train(x = X_train, y = y_train, method = 'rpart', tuneLength = 10, control = rpart.contro
```

```
rpartModel
```

```
## CART
##
## 144 samples
##  58 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   cp           RMSE       Rsquared   MAE
##   0.001423809  0.6258925  0.8825406  0.4908525
##   0.002905583  0.6258925  0.8825406  0.4908525
##   0.007468569  0.6258925  0.8825406  0.4908525
##   0.012989601  0.6258925  0.8825406  0.4908525
##   0.014539435  0.6258925  0.8825406  0.4908525
##   0.016023658  0.6258925  0.8825406  0.4908525
##   0.025976811  0.6258925  0.8825406  0.4908525
##   0.076351752  0.6977620  0.8516766  0.5425167
##   0.132924167  0.9588948  0.7189421  0.7603646
##   0.688667676  1.4093097  0.6481983  1.1574424
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.02597681.
```

For some strange reason, all the RMSE values in rpart are coming to be the same. This is affecting the downstream analysis.

```
set.seed(200)

cubistModel <- train(x = X_train, y = y_train, method = 'cubist')

cubistModel
```

```
## Cubist
##
## 144 samples
##  58 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE          Rsquared  MAE
##    1          0          1.075242e-06  1         9.259229e-07
##    1          5          1.075242e-06  1         9.259229e-07
##    1          9          1.075242e-06  1         9.259229e-07
##   10          0          1.075242e-06  1         9.259229e-07
##   10          5          1.075242e-06  1         9.259229e-07
```

```
##   10          9          1.075242e-06  1         9.259229e-07
##   20          0          1.075242e-06  1         9.259229e-07
##   20          5          1.075242e-06  1         9.259229e-07
##   20          9          1.075242e-06  1         9.259229e-07
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 1 and neighbors = 0.
```

For some strange reason, all the RMSE values in cubist are coming to be the same. This is affecting the downstream analysis.

```
set.seed(200)

rfModel <- train(x = X_train, y = y_train, method = 'rf', tuneLength = 10)

rfModel
```

```
## Random Forest
##
## 144 samples
##  58 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    2    1.1516486  0.6764331  0.9102573
##    8    0.8479032  0.8326831  0.6365960
##   14    0.6808467  0.8942933  0.4859623
##   20    0.5596000  0.9282616  0.3790907
##   26    0.4649420  0.9493735  0.2984883
##   33    0.3776602  0.9641055  0.2274781
##   39    0.3265075  0.9716687  0.1870854
##   45    0.2774839  0.9783160  0.1512584
##   51    0.2452358  0.9820982  0.1298669
##   58    0.2212162  0.9848909  0.1171890
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 58.
```

At this point all the models were run, and we are in a position to run the

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

Let's first assemble all the models.

```
samp <- resamples(list(GradientBoosting = gbmModel, SingleTree = rpartModel, Cubist = cubistModel, Rando

summary(samp)
```

```
##
## Call:
## summary.resamples(object = samp)
##
## Models: GradientBoosting, SingleTree, Cubist, RandomForest
## Number of resamples: 25
##
## MAE
##                            Min.      1st Qu.       Median         Mean
## GradientBoosting 6.550404e-02 9.949971e-02 1.169630e-01 1.207893e-01
## SingleTree       4.116227e-01 4.694628e-01 4.833305e-01 4.908525e-01
## Cubist           7.965088e-07 8.861835e-07 9.392632e-07 9.259229e-07
## RandomForest     7.460975e-02 9.194196e-02 1.097528e-01 1.171890e-01
##                       3rd Qu.         Max. NA's
## GradientBoosting 1.455831e-01 1.694806e-01    0
## SingleTree       5.308505e-01 5.609267e-01    0
## Cubist           9.566087e-07 1.040089e-06    0
## RandomForest     1.371516e-01 1.903360e-01    0
##
## RMSE
##                            Min.      1st Qu.       Median         Mean
## GradientBoosting 7.725253e-02 1.912669e-01 2.254779e-01 2.225059e-01
## SingleTree       4.914666e-01 5.759979e-01 6.190318e-01 6.258925e-01
## Cubist           9.368946e-07 1.051214e-06 1.088301e-06 1.075242e-06
## RandomForest     9.808555e-02 1.540783e-01 2.234715e-01 2.212162e-01
##                       3rd Qu.         Max. NA's
## GradientBoosting 2.601699e-01 4.138932e-01    0
## SingleTree       6.900160e-01 7.477231e-01    0
## Cubist           1.102150e-06 1.182744e-06    0
## RandomForest     2.523397e-01 4.325661e-01    0
##
## Rsquared
##                       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## GradientBoosting 0.9618363 0.9822896 0.9849449 0.9855353 0.9914567 0.9974372
## SingleTree       0.8336897 0.8636901 0.8777723 0.8825406 0.9016210 0.9323672
## Cubist           1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## RandomForest     0.9559919 0.9805769 0.9836985 0.9848909 0.9920629 0.9970188
##                  NA's
## GradientBoosting    0
## SingleTree          0
## Cubist              0
## RandomForest        0
```

Now performance testing, we already have the models. All we have to do is predict on test and compare the RMSE.

```
perf_testing <- function(models, testData, testTarget) {
  method <- c()
  df <- data.frame()
  for(model in models){
    method <- c(method, model$method)
    pred <- predict(model, newdata = testData)
    df <- rbind(df, t(postResample(pred = pred, obs = testTarget)))
  }
```

```
  row.names(df) <- method
  return(df)
}

models <- list(gbmModel, rpartModel, cubistModel, rfModel)

performance <- perf_testing(models, X_test, y_test)

performance[order(performance$RMSE),]
```

```
##                RMSE  Rsquared          MAE
## cubist 1.086017e-06 1.0000000 9.393692e-07
## gbm    4.720618e-01 0.9501244 1.600934e-01
## rf     4.996562e-01 0.9436454 1.692558e-01
## rpart  7.667523e-01 0.8567573 5.002906e-01
```

In order of performance, cubit is the best, followed by gbm, rf and rpart.

(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?

We found Cubist to be the optimal. So, we'll check the varImp() of cubistModel.

```
varImp(cubistModel)
```

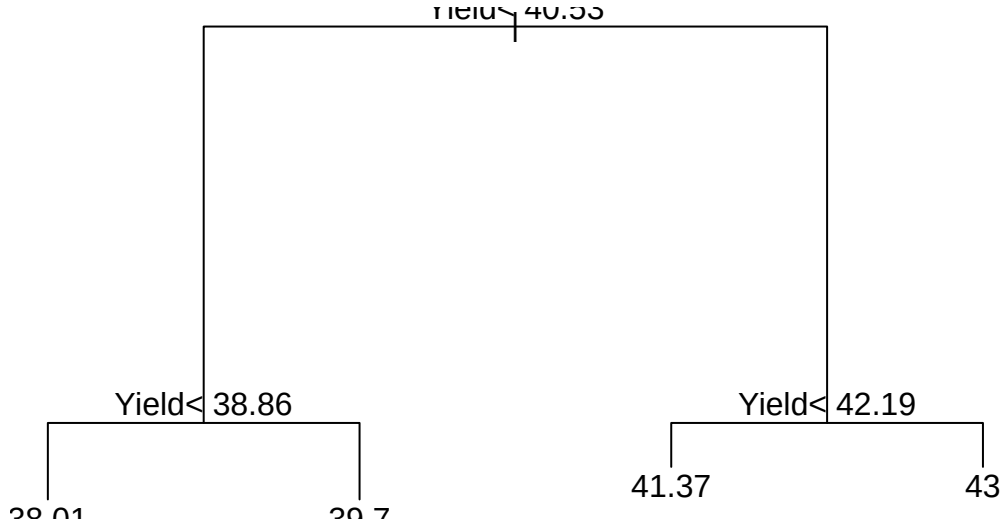
```
## cubist variable importance
##
##   only 20 most important variables shown (out of 58)
##
##                         Overall
## Yield                       100
## BiologicalMaterial11          0
## ManufacturingProcess45        0
## ManufacturingProcess37        0
## ManufacturingProcess06        0
## BiologicalMaterial12          0
## ManufacturingProcess25        0
## ManufacturingProcess15        0
## ManufacturingProcess36        0
## ManufacturingProcess32        0
## ManufacturingProcess08        0
## ManufacturingProcess27        0
## ManufacturingProcess41        0
## ManufacturingProcess31        0
## ManufacturingProcess13        0
## BiologicalMaterial01          0
## ManufacturingProcess26        0
## ManufacturingProcess21        0
## BiologicalMaterial10          0
## ManufacturingProcess30        0
```

We observed above that the RMSE values in cubist were same. That was very weird. It affected the importance of the predictors. Only ManufacturingProcess02 has an Overall of 100. The rest are all 0.

(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?

The plot is as follows:

```
plot(rpartModel$finalModel)
text(rpartModel$finalModel)
```



Marker: 624-10