DATA 624 - Homework 9

OMER OZEREN

Table of Contents

Exercises 8.1, 8.2, 8.3, and 8.7 in Kuhn and Johnson.

```
library(knitr)
library(tidyr)
library(AppliedPredictiveModeling)
```

```r
library(mlbench)
library(ggplot2)
library(mice)
library(caret)
library(Cubist)
library(gbm)
library(ipred)
library(party)
library(partykit)
library(randomForest)
library(rpart)
library(vip)
library(dplyr)
library(rattle)
data(solubility)
```

8.1 Recreate the simulated data from Exercise 7.2

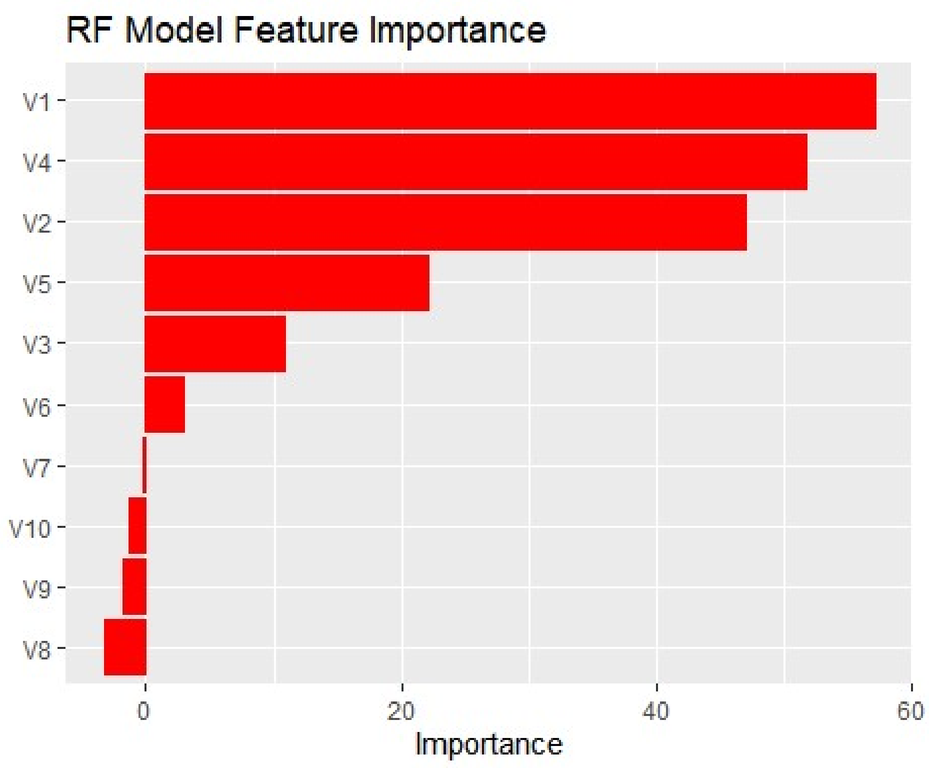
Load data

```r
library(mlbench)
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: Did the random forest model significantly use the uninformative predictors ($V6 - V10$)?

```r
rf_model1 <- randomForest(y ~ ., data = simulated,
                       importance = TRUE,
                       ntree = 1000)
rf_Imp1 <- rf_model1$importance
vip(rf_model1, color = 'red', fill='red') +
  ggtitle('RF Model Feature Importance')

## Warning in vip.default(rf_model1, color = "red", fill = "red"): Arguments
## `width`, `alpha`, `color`, `fill`, `size`, and `shape` have all been
deprecated
## in favor of the new `mapping` and `aesthetics` arguments. They will be
removed
## in version 0.3.0.
```



The graph above shows feature importances that resulted from Random Forest. According to graph above, Random forest model didn't use the uninformative predictors $(V6 - V10)$

b)

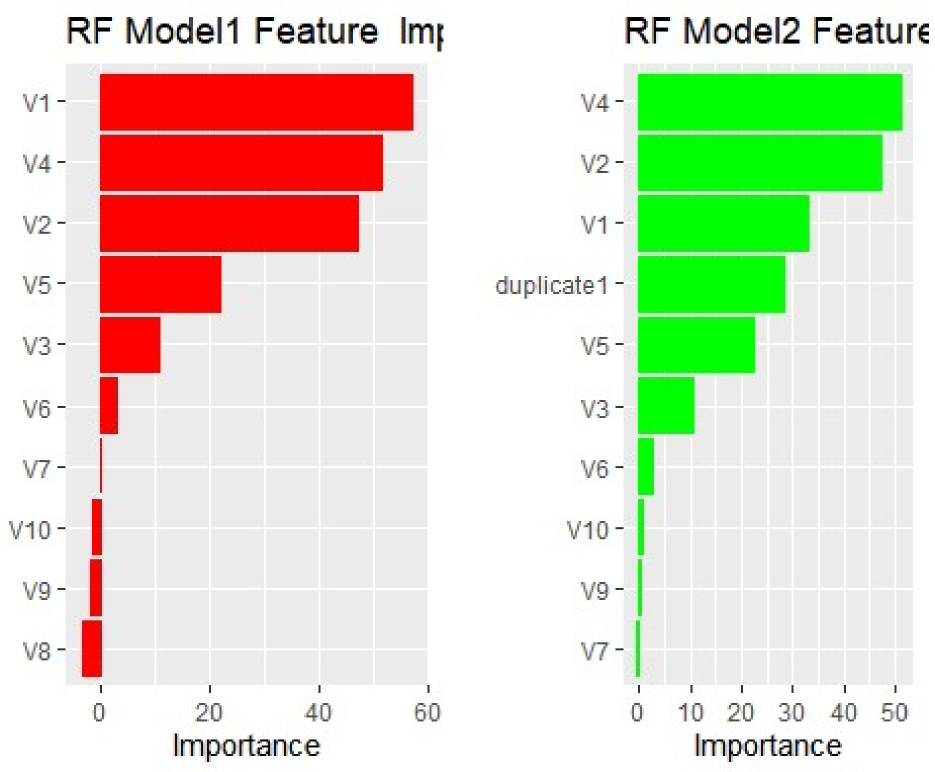
Now add an additional predictor that is highly correlated with one of the informative predictors. For example:

```
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

```
rf_model2 <- randomForest(y ~ ., data = simulated,
                       importance = TRUE,
                       ntree = 1000)
rfImp2 <- varImp(rf_model2, scale = FALSE)
grid.arrange(vip(rf_model1, color = 'red', fill='red') +
  ggtitle('RF Model1 Feature  Importance'), vip(rf_model2, color = 'green',
fill='green') +
  ggtitle('RF Model2 Feature  Importance'), ncol = 2)
```

```
## Warning in vip.default(rf_model1, color = "red", fill = "red"): Arguments
## `width`, `alpha`, `color`, `fill`, `size`, and `shape` have all been
deprecated
## in favor of the new `mapping` and `aesthetics` arguments. They will be
removed
## in version 0.3.0.

## Warning in vip.default(rf_model2, color = "green", fill = "green"):
Arguments
## `width`, `alpha`, `color`, `fill`, `size`, and `shape` have all been
deprecated
## in favor of the new `mapping` and `aesthetics` arguments. They will be
removed
## in version 0.3.0.
```

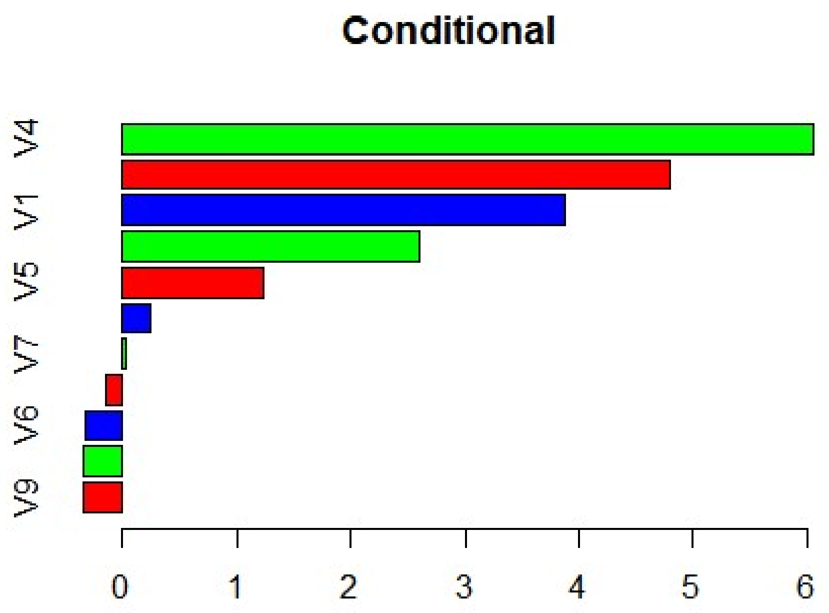


Adding high correlated feature in model, makes change in feature importances for model.The first model says that most important feature is V1 but adding additional highly correlated variable, made V4 the most important variable.

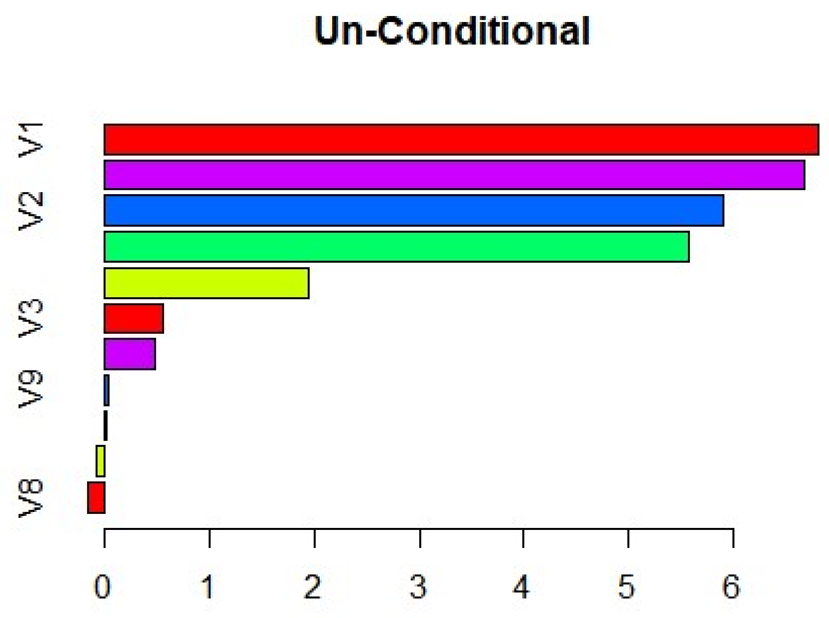
c)

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?

```r
rf_model3 <- cforest(y ~ ., data = simulated, ntree = 100)
cfImp3 <- varimp(rf_model3, conditional = TRUE)
cfImp4 <- varimp(rf_model3, conditional = FALSE)
barplot(sort(cfImp3),horiz = TRUE, main = 'Conditional', col = rainbow(3))
```



```r
barplot(sort(cfImp4),horiz = TRUE, main = 'Un-Conditional', col = rainbow(5))
```



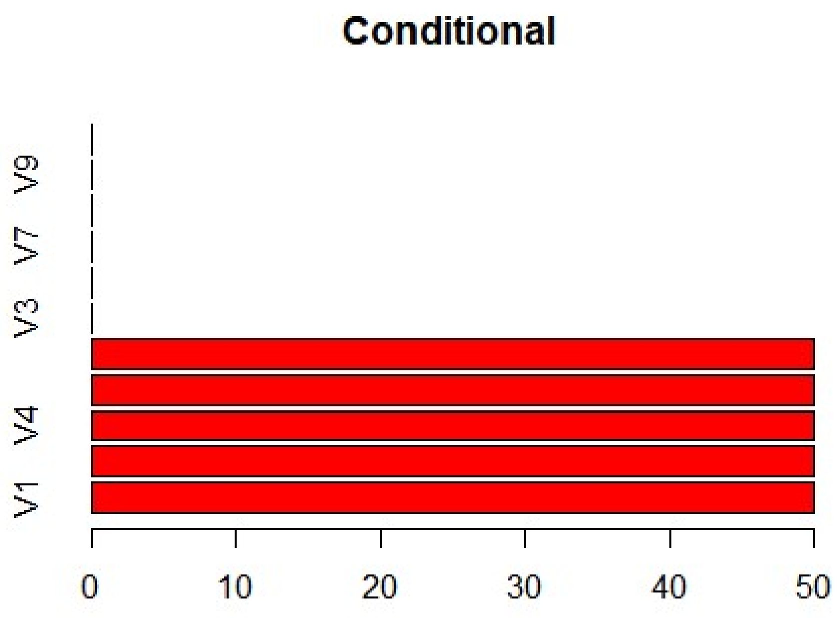
The graph above shows that conditional and un-conditional inference tress also ingnores $V6 - V10$

d)

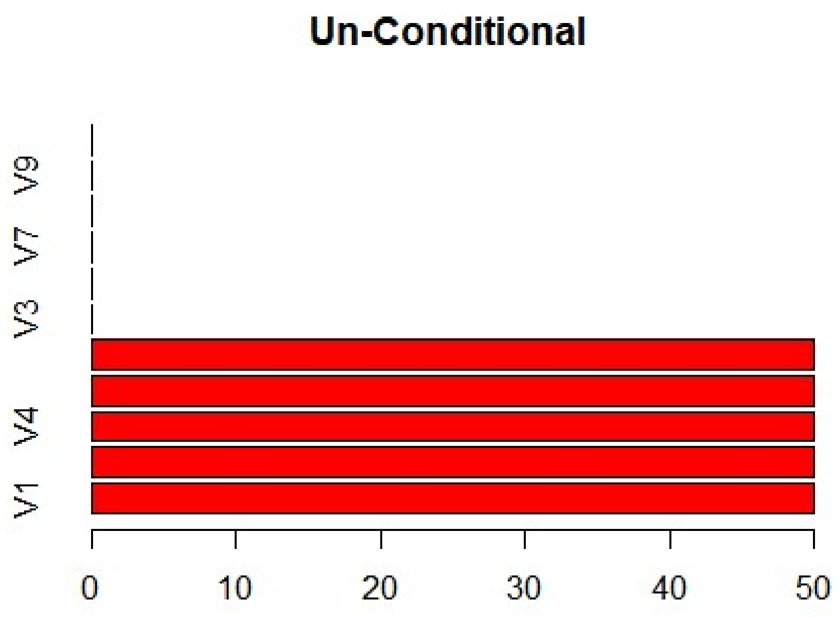
Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

First, I will use Cubist model

```
cubist_model <- cubist(x = simulated[, names(simulated)[names(simulated) !=
'y']],
                 y = simulated[,c('y')])

cubist_Imp4 <- varImp(cubist_model, conditional = TRUE) # Conditional
variable importance
cubist_Imp5 <- varImp(cubist_model, conditional = FALSE) # Un-conditional
variable importance
barplot((t(cubist_Imp4)),horiz = TRUE, main = 'Conditional', col =
rainbow(3))
```

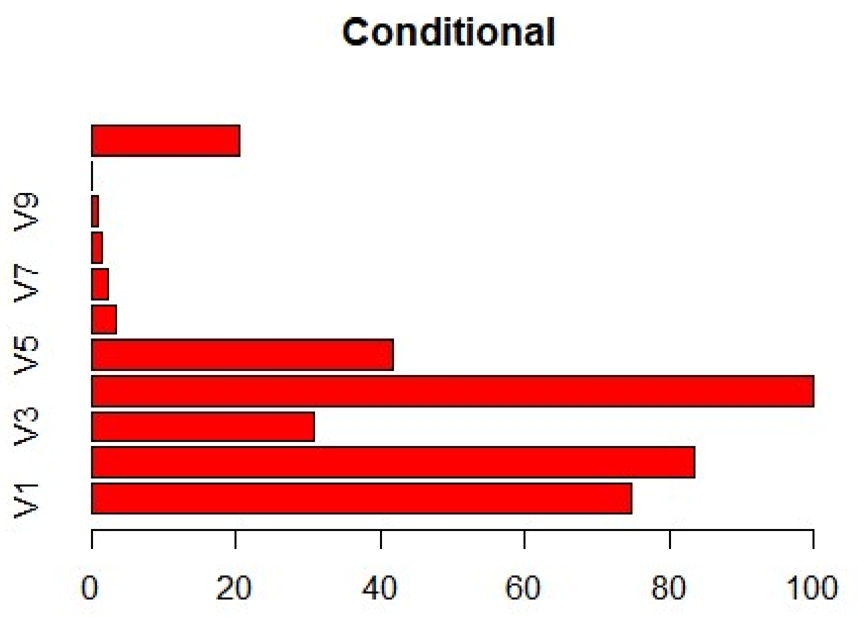


```
barplot((t(cubist_Imp5)),horiz = TRUE, main = 'Un-Conditional', col =
rainbow(5))
```

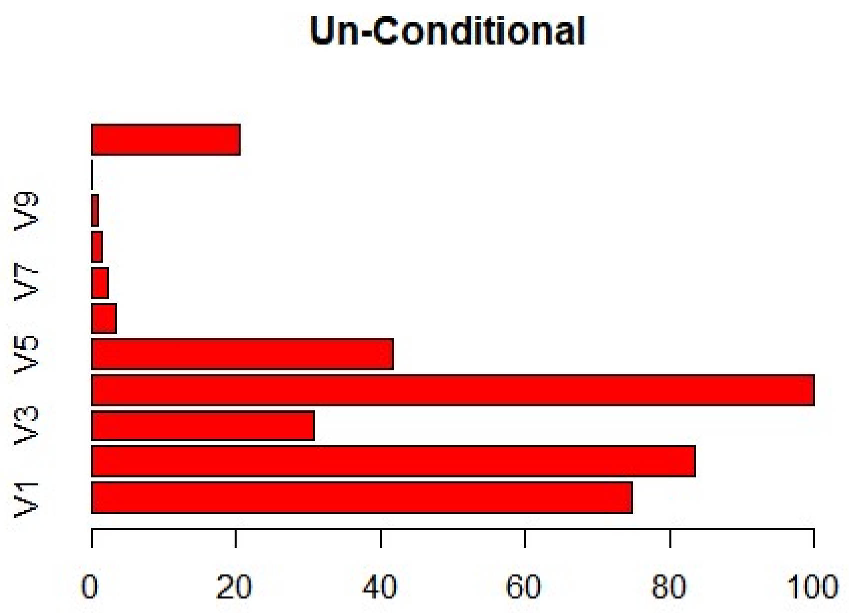


Next, I will use Boosted Trees

```
grid_params = expand.grid(interaction.depth = seq(1,5, by=2), n.trees =
seq(100, 150, by = 10), shrinkage = 0.1, n.minobsinnode = 5)
cubist_model <- train(y ~ ., data = simulated, tuneGrid = grid_params,
verbose = FALSE, method = 'gbm' )
cubist_Imp4 <- varImp(cubist_model, conditional = TRUE) # Conditional
variable importance
cubist_Imp5 <- varImp(cubist_model, conditional = FALSE) # Un-conditional
variable importance
barplot((t(cubist_Imp4$importance)),horiz = TRUE, main = 'Conditional', col =
rainbow(3))
```



```
barplot((t(cubist_Imp5$importance)),horiz = TRUE, main = 'Un-Conditional',
col = rainbow(5))
```

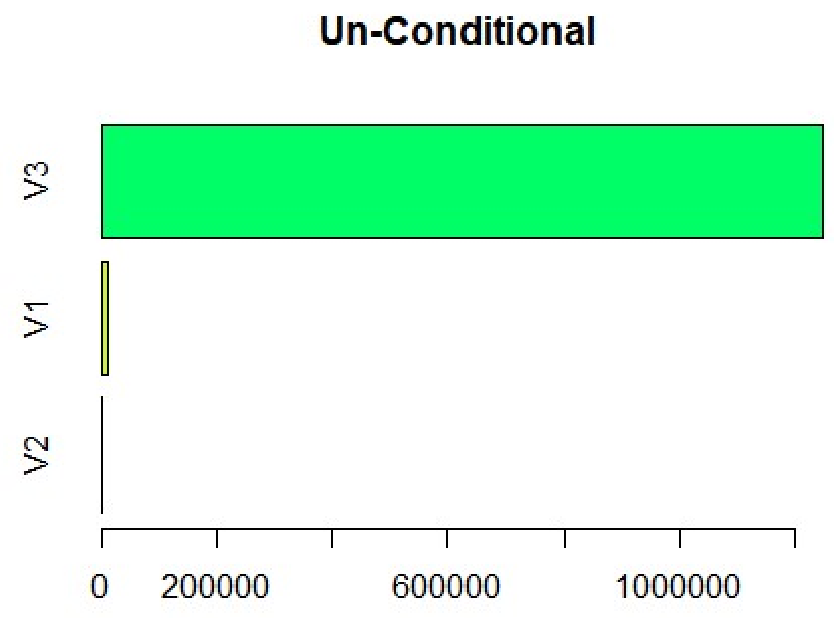


The feature importances for cubist model and GBM model with Conditional and Un-conditional is the same.

8.2

Use a simulation to show tree bias with different granularities.

```
V1 <- runif(500, 2,500)
V2 <- rnorm(500, 2,10)
V3 <- rnorm(500, 1,1000)
y <- V2 + V3
df <- data.frame(V1, V2, V3, y)
test_model <- cforest(y ~ ., data = df, ntree = 10)
test_model_imp <- varimp(test_model, conditional = FALSE)
barplot(sort(test_model_imp),horiz = TRUE, main = 'Un-Conditional', col =
rainbow(5))
```



Random Forest model's the most important variable is significantly V3 based on using function of y <- V2 + V3

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.

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 learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.The learning rate may be the most important hyperparameter when configuring your neural network. Therefore it is vital to know how to investigate the effects of the learning rate on model performance and to build an intuition about the dynamics of the learning rate on model behavior.The larger learning rates result in rapid changes and require fewer training epochs.(References :https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/)

The model on right have significant learning rate.The model on right is trained high fraction rate (0.9).It takes more variables to learn from it.

b)

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

I believe model on left would be more predictive of other sample. I also believe that model on right will face the overfitting problem.I think that using bagging models with weak learners would predict better.

c)

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

The increase in depth in tree, will result the tree to grow deeper.The model would consider more and more variable to consider in the final tree model.

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
library(AppliedPredictiveModeling)
data(ChemicalManufacturingProcess)

# using Knn imputation
knn_model <- preProcess(ChemicalManufacturingProcess, "knnImpute")
df <- predict(knn_model, ChemicalManufacturingProcess)
df <- df %>%
  select_at(vars(-one_of(nearZeroVar(., names = TRUE))))
in_train <- createDataPartition(df$Yield, times = 1, p = 0.8, list = FALSE)

#split data in train and test samples
train_df <- df[in_train, ]
test_df <- df[-in_train, ]
```

I’ll use multiple tree-based models

Bagged Model

```r
set.seed(42)
bag_setup = bagControl(fit = ctreeBag$fit, predict = ctreeBag$pred, aggregate
= ctreeBag$aggregate)
bag_model_fit <- train(Yield ~ ., data = train_df, method="bag", bagControl =
bag_setup,
                   center = TRUE,
                   scale = TRUE,
                   trControl = trainControl("cv", number = 10),importance =
"permutation",
                   tuneLength = 25)
```

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

```
bag_pred <- predict(bag_model_fit, test_df)
bag_results<- merge(bag_model_fit$results, bag_model_fit$bestTune)
bag_model_fit
```

```
## Bagged Model
##
## 144 samples
##  56 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 131, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.6982059  0.5355203  0.5677121
##
## Tuning parameter 'vars' was held constant at a value of 56
```

Random Forest

```
set.seed(42)
rf_model_fit <- train(Yield ~ ., data = train_df, method = "ranger",
                      scale = TRUE,
                      trControl = trainControl("cv", number = 10),importance =
"permutation",
                      tuneLength = 25)
rf_pred <- predict(rf_model_fit, test_df)
rf_results<- merge(rf_model_fit$results, rf_model_fit$bestTune)
rf_model_fit
```

```
## Random Forest
##
## 144 samples
##  56 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 131, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##    2    variance    0.6596347  0.6730199  0.5461635
##    2    extratrees  0.7069935  0.6445118  0.5784647
##    4    variance    0.6221818  0.6890184  0.5145952
##    4    extratrees  0.6623573  0.6782966  0.5476618
##    6    variance    0.6069993  0.7089790  0.5021309
##    6    extratrees  0.6439441  0.6913187  0.5332596
##    8    variance    0.6034978  0.7051654  0.4979139
##    8    extratrees  0.6321382  0.7005991  0.5240254
##   11    variance    0.6041013  0.7007864  0.4888491
##   11    extratrees  0.6261832  0.7015360  0.5181244
##   13    variance    0.5987025  0.7019262  0.4843624
##   13    extratrees  0.6145560  0.7093223  0.5066966
##   15    variance    0.5955820  0.7054603  0.4808433
##   15    extratrees  0.6153537  0.7075823  0.5071409
##   17    variance    0.5915535  0.7076097  0.4769376
##   17    extratrees  0.6078224  0.7115051  0.5024403
##   20    variance    0.5945331  0.7020151  0.4785689
##   20    extratrees  0.6039336  0.7166329  0.4991691
##   22    variance    0.5935498  0.7049929  0.4758162
##   22    extratrees  0.6079235  0.7052497  0.5025346
##   24    variance    0.5942722  0.7031048  0.4764206
##   24    extratrees  0.6009275  0.7201858  0.4965043
##   26    variance    0.5975124  0.6957064  0.4772120
##   26    extratrees  0.6042247  0.7054541  0.4983234
##   29    variance    0.5934303  0.6955083  0.4720059
##   29    extratrees  0.5938772  0.7238096  0.4876019
##   31    variance    0.5984898  0.6902514  0.4752357
##   31    extratrees  0.6014699  0.7112300  0.4951885
##   33    variance    0.5920592  0.7011292  0.4711088
##   33    extratrees  0.5935499  0.7181202  0.4851416
##   35    variance    0.5941304  0.6959748  0.4726392
##   35    extratrees  0.5973220  0.7133100  0.4893577
##   38    variance    0.5951624  0.6970735  0.4716350
##   38    extratrees  0.5971571  0.7103849  0.4884069
##   40    variance    0.5954807  0.6909799  0.4720197
##   40    extratrees  0.5945167  0.7182807  0.4875036
##   42    variance    0.5934469  0.6891524  0.4700760
##   42    extratrees  0.5946703  0.7143891  0.4857360
##   44    variance    0.5987754  0.6879898  0.4725805
##   44    extratrees  0.5950121  0.7115564  0.4848409
##   47    variance    0.5988108  0.6907673  0.4723850
##   47    extratrees  0.5963543  0.7074772  0.4866569
##   49    variance    0.6026037  0.6813637  0.4750228
##   49    extratrees  0.5976771  0.7058332  0.4890629
##   51    variance    0.6023486  0.6825048  0.4745954
##   51    extratrees  0.6001544  0.7018232  0.4885167
##   53    variance    0.6023119  0.6832649  0.4740204
##   53    extratrees  0.5943415  0.7120494  0.4879861
##   56    variance    0.6031821  0.6848129  0.4748161
##   56    extratrees  0.5926617  0.7099710  0.4832255
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 17, splitrule = variance
##  and min.node.size = 5.
```

GBM Model with Grid Search Parameters

```
grid_params <- 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))
gmb_model_fit<- train(Yield ~.,
                  data = train_df,
                  method = 'gbm',
                  tuneGrid = grid_params,
                  verbose = FALSE)
gmb_pred <- predict(gmb_model_fit, test_df)
gmb_results<- merge(gmb_model_fit$results, gmb_model_fit$bestTune)
gmb_model_fit
```

```
## Stochastic Gradient Boosting
##
## 144 samples
##  56 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      0.8905525
0.4646386
##   0.01        1                  5              100      0.8201710
0.5080546
##   0.01        1                 10               50      0.8899421
0.4662205
##   0.01        1                 10              100      0.8169397
0.5051338
##   0.01        5                  5               50      0.8315637
0.5580368
##   0.01        5                  5              100      0.7398220
0.5785746
##   0.01        5                 10               50      0.8335716
0.5450531
##   0.01        5                 10              100      0.7489084
0.5596447
##   0.01       10                  5               50      0.8203798
0.5769731
##   0.01       10                  5              100      0.7292240
0.5858597
##   0.01       10                 10               50      0.8340794
0.5463490
##   0.01       10                 10              100      0.7494200
0.5601704
```

```
##   0.10        1                   5                50       0.6751552
0.5717954
##   0.10        1                   5               100       0.6574689
0.5872348
##   0.10        1                  10                50       0.6819994
0.5571468
##   0.10        1                  10               100       0.6644514
0.5716992
##   0.10        5                   5                50       0.6375177
0.6062945
##   0.10        5                   5               100       0.6234699
0.6215074
##   0.10        5                  10                50       0.6495247
0.5947947
##   0.10        5                  10               100       0.6432539
0.6015066
##   0.10       10                   5                50       0.6325042
0.6146587
##   0.10       10                   5               100       0.6225700
0.6258944
##   0.10       10                  10                50       0.6476617
0.5953512
##   0.10       10                  10               100       0.6389381
0.6035732
##   0.20        1                   5                50       0.6782735
0.5524938
##   0.20        1                   5               100       0.6742231
0.5595049
##   0.20        1                  10                50       0.6771706
0.5552471
##   0.20        1                  10               100       0.6706262
0.5658950
##   0.20        5                   5                50       0.6620531
0.5716892
##   0.20        5                   5               100       0.6606302
0.5735395
##   0.20        5                  10                50       0.6558359
0.5823877
##   0.20        5                  10               100       0.6498455
0.5899516
##   0.20       10                   5                50       0.6522168
0.5868506
##   0.20       10                   5               100       0.6511136
0.5880604
##   0.20       10                  10                50       0.6700937
0.5641587
##   0.20       10                  10               100       0.6689084
0.5678719
##   MAE
##   0.7100256
##   0.6530620
##   0.7086569
##   0.6505222
##   0.6615056
##   0.5860643
##   0.6615865
##   0.5920153
##   0.6514434
##   0.5741771
##   0.6619891
##   0.5913029
##   0.5323869
##   0.5114674
##   0.5308315
##   0.5112949
##   0.4952583
##   0.4828053
##   0.5027599
##   0.4956537
##   0.4843069
##   0.4762804
##   0.4972859
##   0.4888663
##   0.5254858
##   0.5205232
##   0.5266903
##   0.5176813
##   0.5130905
##   0.5111049
##   0.5075953
##   0.5037803
##   0.5011052
##   0.5023596
##   0.5229251
##   0.5233873
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 100, interaction.depth
=
##  10, shrinkage = 0.1 and n.minobsinnode = 5.
```

Let’s review **in-sample** results of eact tree-based model

```
insample_results <- data.frame(rbind(getTrainPerf(bag_model_fit),
                                     getTrainPerf(rf_model_fit),
                                     getTrainPerf(gmb_model_fit)))
row.names(insample_results) <- c("Bagging", "RandomForest","GBM")
insample_results
```

```
##             TrainRMSE TrainRsquared  TrainMAE method
## Bagging     0.6982059     0.5355203 0.5677121    bag
## RandomForest 0.5915535     0.7076097 0.4769376 ranger
## GBM         0.6225700     0.6258944 0.4762804    gbm
```

Let's review **out-sample** results of eact tree-based model

```
outsample_results <- data.frame(rbind(postResample(pred = bag_pred, obs =
test_df$Yield),
                        postResample(pred = rf_pred, obs = test_df$Yield),
                        postResample(pred = gmb_pred, obs = test_df$Yield)))
row.names(outsample_results) <- c("Bagging", "RandomForest","GBM")
outsample_results

##                   RMSE  Rsquared       MAE
## Bagging      0.6818692 0.5222643 0.5343107
## RandomForest 0.5851765 0.6707843 0.4448205
## GBM          0.7136320 0.4873264 0.5477102
```

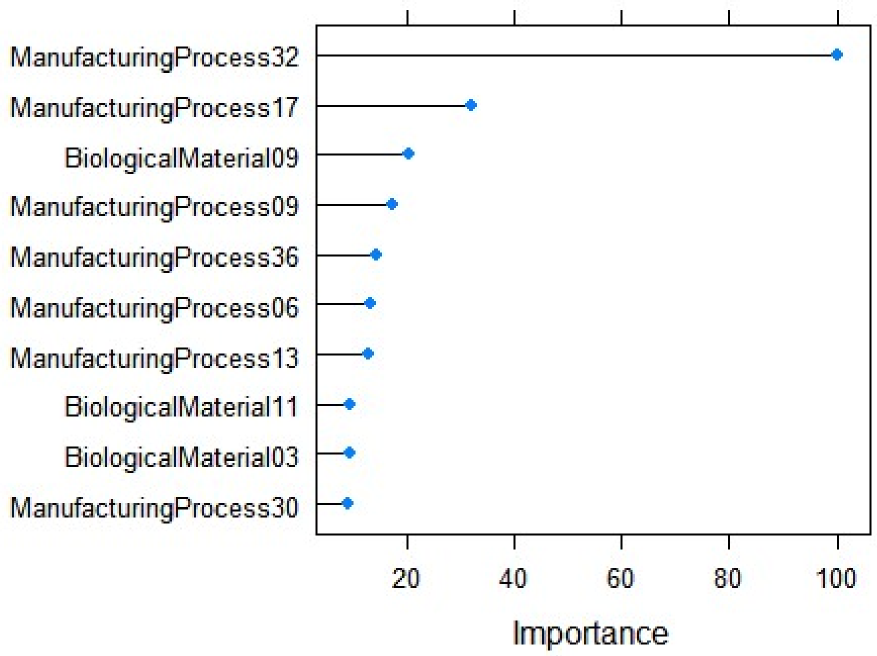
a) Which tree-based regression model gives the optimal resampling and test set performance?

The best model in insample test is GBM model (0.64) and The best model in outsample test is also GBM model (0.57)

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?

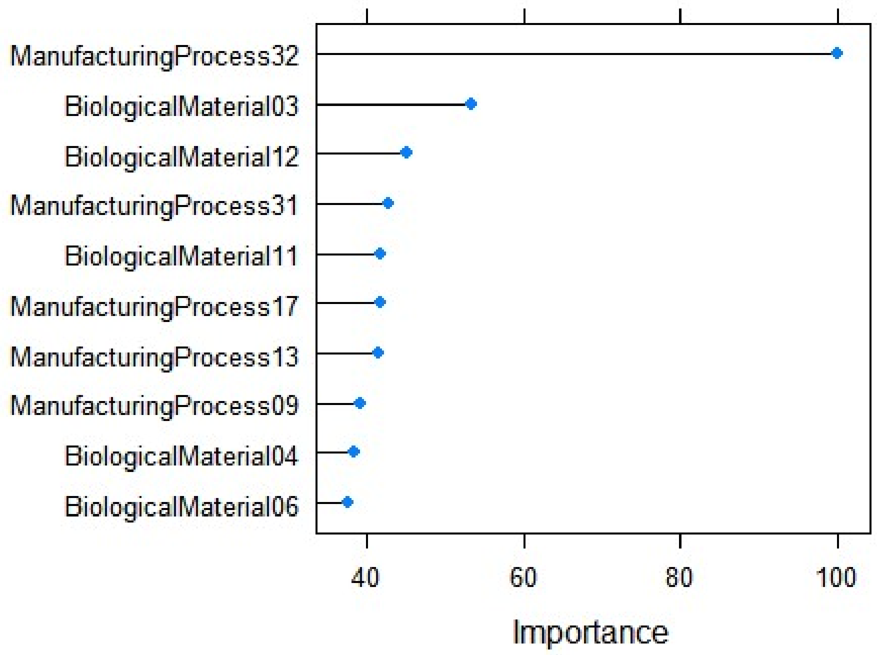
```
plot(varImp(gmb_model_fit),
     top=10,
     main="Feature Importance with GBM Model")
```

Feature Importance with GBM Model

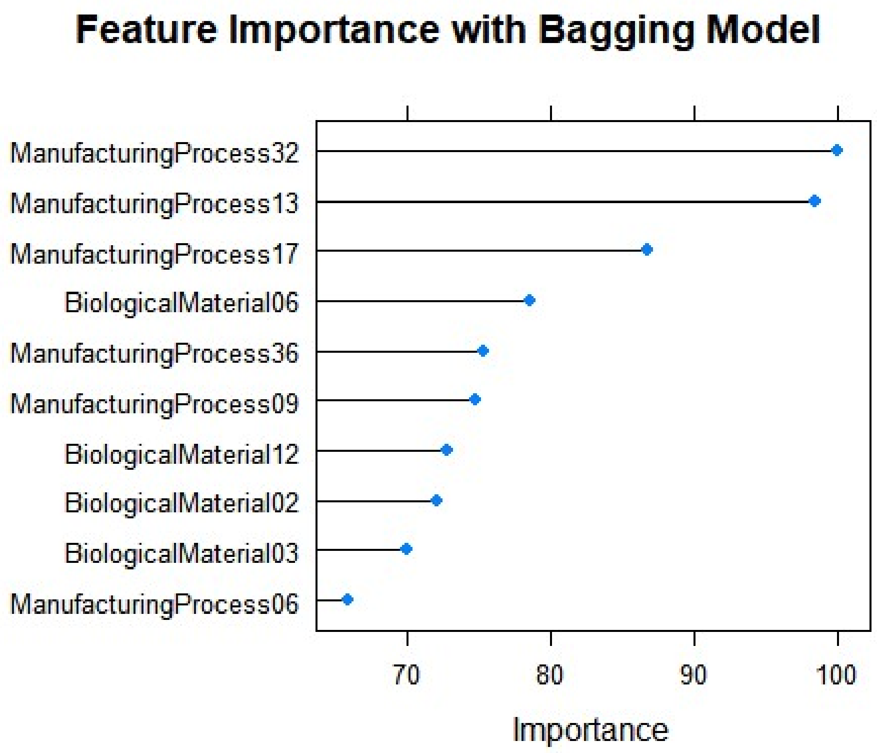


```r
plot(varImp(rf_model_fit),
     top=10,
     main="Feature Importance with Random Forest Model")
```

Feature Importance with Random Forest Model



```r
plot(varImp(bag_model_fit),
     top=10,
     main="Feature Importance with Bagging Model")
```

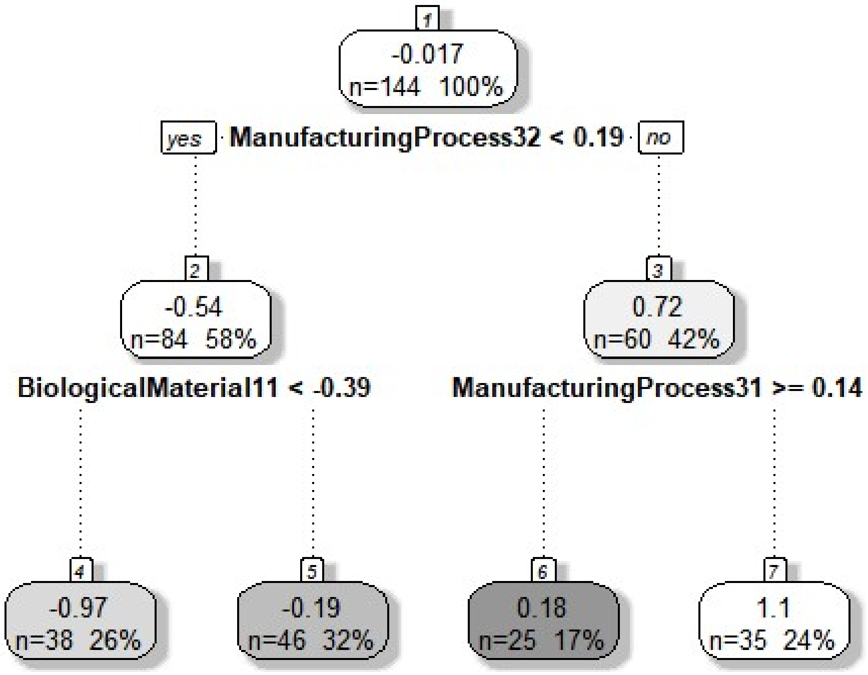


The plots above indicate that best model (GBM)'s most importantant variable is Process32 and remaning majority variables are belong to manufactoring category.The three tree-based model says thatAManufactoringProcess32 is significantly most important variable.

Part 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?

```r
set.seed(42)
library(rpart)
model <- train(Yield ~ ., data = train_df, method = "rpart",
                  trControl = trainControl("cv", number = 10),
                  tuneLength = 25)

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

fancyRpartPlot(model$finalModel, sub="", palettes=c("Greys", "Oranges"))
```



I found fancyRpartPlot from website https://rdrr.io/cran/rattle/man/fancyRpartPlot.html. The plot looks like does its job.The plot indicates that majority feature importances are dominated by manufacturing .