Applied Predictive Modeling

DATA 624 Group 2 Homework

John Grando, Kyle Gilde, Jonathan Hernandez, Albert Gilharry, Neil Hwang

January 27, 2019

Table of Contents

# Overview

**For each response, and commentary, provided by us, the text will be bolded, as it appears here**

# Chapter 8 Regression Trees and Rule-Based Models

## Question 8.1

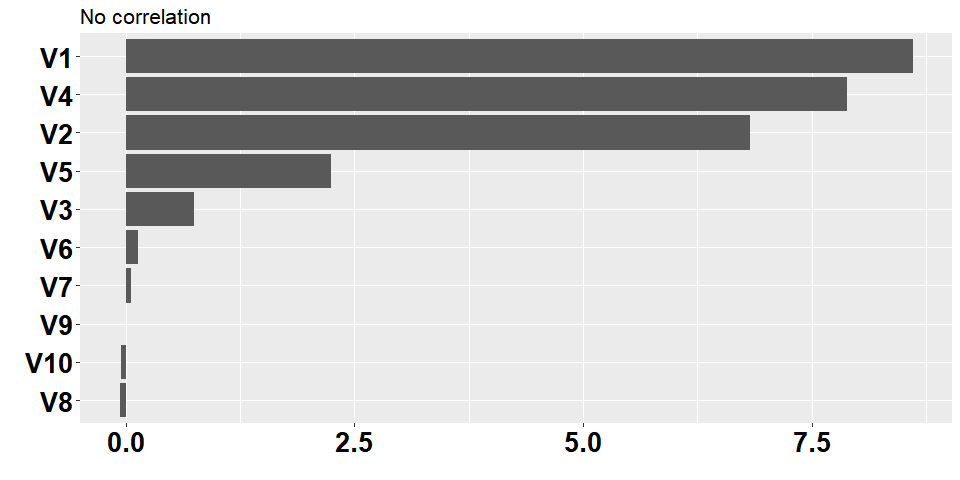
Recreate the simulated data from Exercise 7.2:

set.seed(200)  
simulated1 <- mlbench.friedman1(200, sd = 1)  
simulated1 <- cbind(simulated1$x, simulated1$y)  
simulated1 <- as.data.frame(simulated1)  
colnames(simulated1)[ncol(simulated1)] <- "y"

1. 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)?

**It did not. The plot below shows that they are the 5 least informative variables.**

set.seed(200)  
model1 <- randomForest(y ~ ., data = simulated1, importance = T, ntree = 1000)  
  
rfImp1 <-   
 varImp(model1, scale = FALSE) %>%   
 mutate(Var = rownames(.)) %>%   
 arrange(desc(Overall)) %>%   
 dplyr::select(Var, Overall)  
  
ggplot(rfImp1) +   
 geom\_col(aes(x = reorder(Var, Overall), y = Overall)) +  
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(size=16)) +   
 ggtitle("No correlation")



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

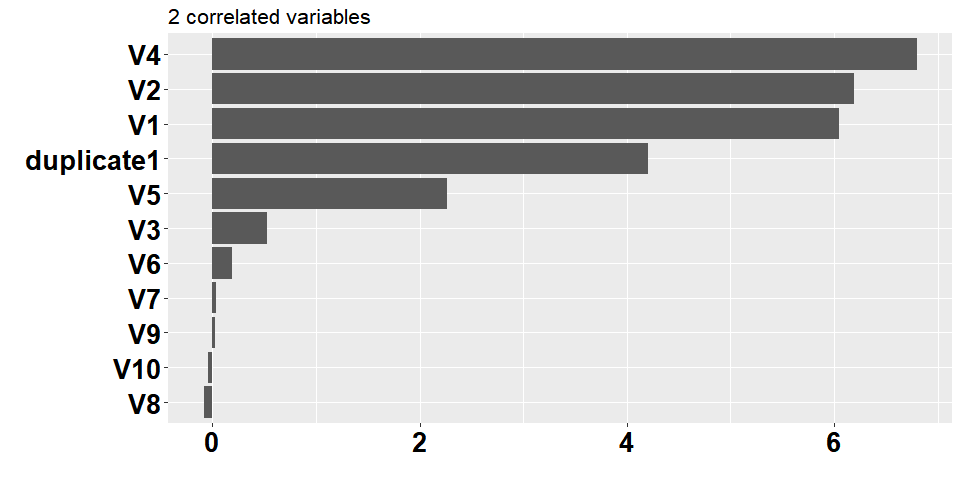
set.seed(200)  
simulated2 <-   
 simulated1 %>%   
 mutate(duplicate1 = V1 + rnorm(200) \* .1)  
  
cor(simulated2$duplicate1, simulated2$V1)

## [1] 0.9497025

Fit another random forest model to these data. Did the importance score for V1 change?

**Yes, the importance score did decrease for V1 from 8.61 to 6.80, and the highly correlated variable is the next important at 4.21**

set.seed(200)  
model2 <- randomForest(y ~ ., data = simulated2,   
 importance = TRUE,  
 ntree = 1000)  
  
rfImp2 <- varImp(model2, scale = FALSE) %>%   
 mutate(Var = rownames(.)) %>%   
 arrange(desc(Overall)) %>%   
 dplyr::select(Var, Overall)  
  
  
ggplot(rfImp2) +   
 geom\_col(aes(x = reorder(Var, Overall), y = Overall)) +  
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(size=16)) +   
 ggtitle("2 correlated variables")



What happens when you add another predictor that is also highly correlated with V1?

# add another correlated variable  
set.seed(5)  
simulated3 <-   
 simulated2 %>%   
 mutate(duplicate2 = V1 + rnorm(200) \* .1)  
  
cor(simulated3$duplicate2, simulated3$V1)

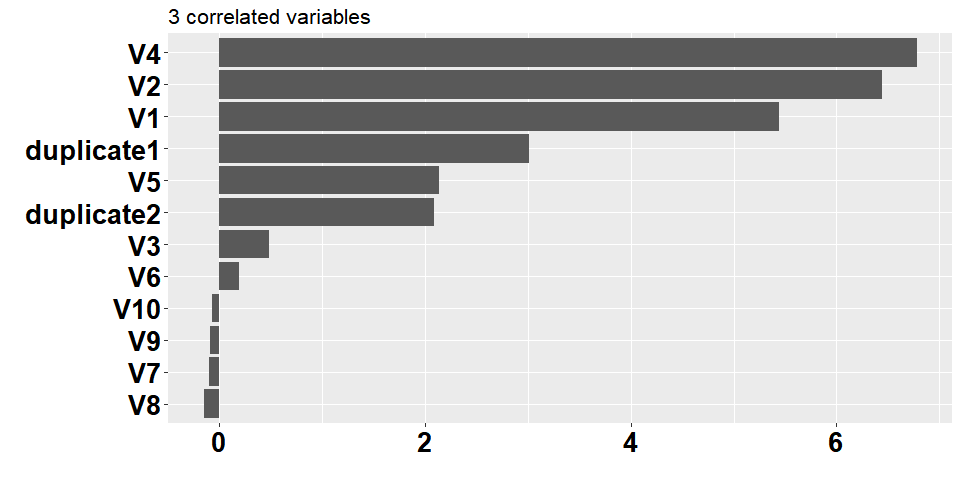
## [1] 0.9412195

**From the table below, we can see that adding another highly correlated variable further diluted the importance scores of V1 and duplicate1.**

**As KJ note on p180, when variables are extremely correlated, the “choice of which to use in a sp0lit is somewhat random.”**

**Having highly correlated variables made the correct interpretation of predictor importance difficult. Using the model with 3 highly correlated variables, one gets the impression that V1, duplicate1 and duplicate2 are of only moderate importance to the model. However, without the other 2 variables, one would understand that the signal coming from V1 is actually most important to the model.**

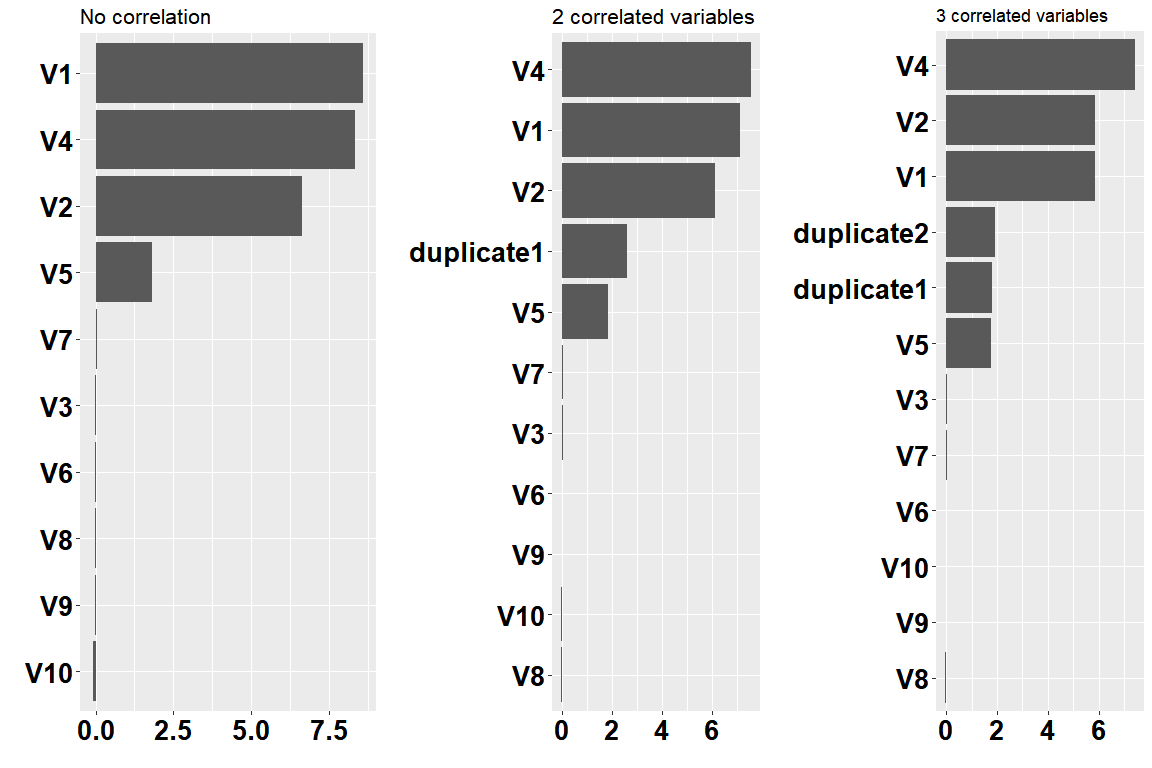
model3 <- randomForest(y ~ ., data = simulated3, importance = TRUE, ntree = 1000)  
  
rfImp3 <- varImp(model3, scale = FALSE) %>%  
 mutate(Var = rownames(.)) %>%  
 arrange(desc(Overall)) %>%  
 dplyr::select(Var, Overall)  
  
ggplot(rfImp3) +   
 geom\_col(aes(x = reorder(Var, Overall), y = Overall)) +  
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(size=16)) +   
 ggtitle("3 correlated variables")



1. 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?

**Yes, in the 3 plots below, the same pattern is displayed - although perhaps to a lesser to degree. The importance of V1 becomes more diluted when more highly correlated variables are present.**

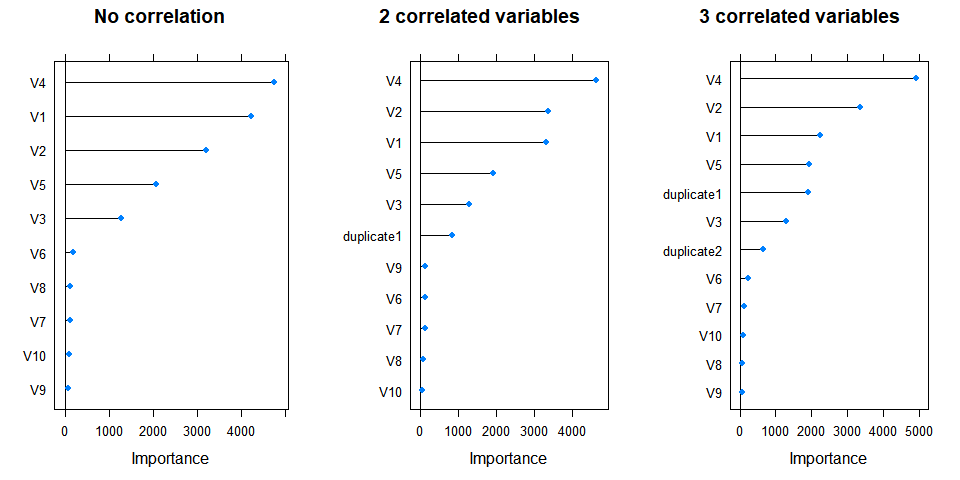
crf1 <- cforest(y ~ ., data = simulated1, controls = cforest\_unbiased(ntree = 1000))  
crf2 <- cforest(y ~ ., data = simulated2, controls = cforest\_unbiased(ntree = 1000))  
crf3 <- cforest(y ~ ., data = simulated3, controls = cforest\_unbiased(ntree = 1000))  
  
crf\_imp <- data.frame(  
 crf3 = varimp(crf3),  
 crf1 = c(varimp(crf1), NA, NA),   
 crf2 = c(varimp(crf2), NA)  
 ) %>%   
 dplyr::select(sort(names(.))) %>%   
 mutate(Var = row.names(.))   
  
gridExtra::grid.arrange(  
 ggplot(na.omit(crf\_imp)) +   
 geom\_col(aes(x = reorder(Var, crf1), y = crf1)) +  
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(size=16)) +   
 ggtitle("No correlation"),  
 ggplot(na.omit(crf\_imp[, c("Var", "crf2")])) +   
 geom\_col(aes(x = reorder(Var, crf2), y = crf2)) +   
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(size=16)) +  
 ggtitle("2 correlated variables"),  
 ggplot(crf\_imp) +   
 geom\_col(aes(x = reorder(Var, crf3), y = crf3)) +   
 labs(x = "", y = "") +   
 coord\_flip() +   
 theme(axis.text = element\_text(size=20, color="black", face = "bold"), plot.title = element\_text(16)) +  
 ggtitle("3 correlated variables"),  
 ncol = 3  
)



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

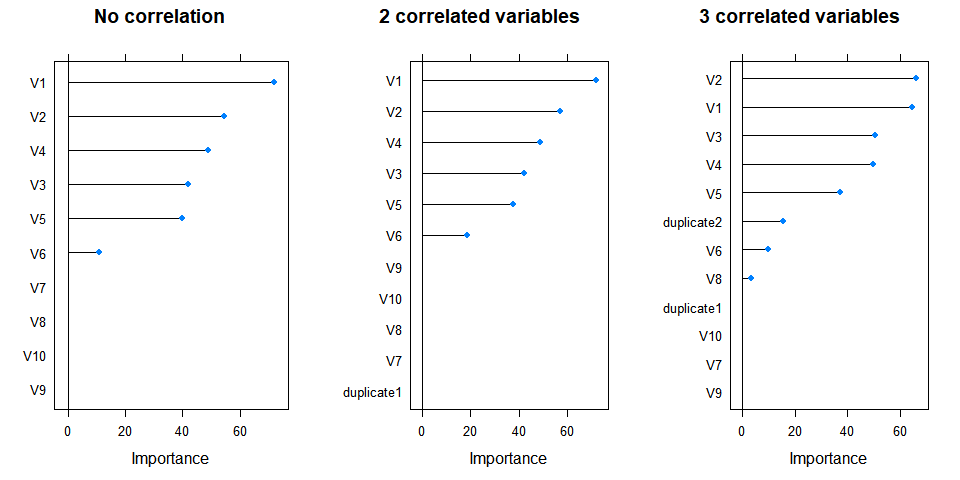
**Interestingly, V1 is only the second most important variable in the first non-correlated dataset. When we add the first highly correlated variable, V1’s position does decline some, but not as much as the first random forest model. When the second highly correlated variable is added, V1’s importance remains relatively unchanged.**

gbm1 <- train(y ~ ., data = simulated1, method = "gbm", verbose = F)  
gbm2 <- train(y ~ ., data = simulated2, method = "gbm", verbose = F)  
gbm3 <- train(y ~ ., data = simulated3, method = "gbm", verbose = F)  
  
  
gridExtra::grid.arrange(  
 plot(varImp(gbm1, scale = F), main = "No correlation"),  
 plot(varImp(gbm2, scale = F), main = "2 correlated variables"),  
 plot(varImp(gbm3, scale = F), main = "3 correlated variables"),  
 ncol = 3  
)



**From the plots below, the Cubist model’s variable importance is the least affected by the addition highly correlated variables. V1’s variable importance remains largely unchanged When the first highly correlated variable is added. When the second one is added, V1 does drop a position in importance, but its score only marginally declines.**

cubist1 <- train(y ~ ., data = simulated1, method = "cubist", control = Cubist::cubistControl(seed = 1))  
cubist2 <- train(y ~ ., data = simulated2, method = "cubist", control = Cubist::cubistControl(seed = 1))  
cubist3 <- train(y ~ ., data = simulated3, method = "cubist", control = Cubist::cubistControl(seed = 1))  
  
  
gridExtra::grid.arrange(  
 plot(varImp(cubist1, scale = F), main = "No correlation"),  
 plot(varImp(cubist2, scale = F), main = "2 correlated variables"),  
 plot(varImp(cubist3, scale = F), main = "3 correlated variables"),  
 ncol = 3  
)



## Question 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:

1. 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?
2. Which model do you think would be more predictive of other samples?
3. How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

## Question 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:

data(ChemicalManufacturingProcess)  
  
set.seed(5)  
# add mape to the summary functions  
mapeSummary <- function (data,  
 lev = NULL,  
 model = NULL) {  
 c(MAPE = MLmetrics::MAPE(data$pred, data$obs) \* 100,  
 RMSE = MLmetrics::RMSE(data$pred, data$obs),  
 Rsquared = summary(lm(pred ~ obs, data))$r.squared)  
}  
  
# preprocess for trees, impute missing  
chem\_preprocess <- preProcess(ChemicalManufacturingProcess, method = c("bagImpute"))  
chem\_df <- predict(chem\_preprocess, ChemicalManufacturingProcess)  
  
# train-test partition  
training\_rows <- createDataPartition(chem\_df$Yield, p =.8, list = F)  
x\_train <- chem\_df[training\_rows, ]  
x\_test <- chem\_df[-training\_rows, ]  
y\_test <- chem\_df[-training\_rows, "Yield"]

#### Linear Model ####  
  
(chem\_preprocess\_lm <- preProcess(ChemicalManufacturingProcess, method = c("nzv", "corr", "bagImpute"))) # "center", "scale",

## Created from 152 samples and 58 variables  
##   
## Pre-processing:  
## - bagged tree imputation (47)  
## - ignored (0)  
## - removed (11)

chem\_df\_lm <- predict(chem\_preprocess\_lm, ChemicalManufacturingProcess)  
  
x\_train\_lm <- chem\_df\_lm[training\_rows, ]  
x\_test\_lm <- chem\_df\_lm[-training\_rows, ]  
y\_test\_lm <- chem\_df\_lm[-training\_rows, "Yield"]  
  
  
sum(summary(chem\_df\_lm[, "Yield"]) == 0)

## [1] 0

set.seed(5)  
lm\_control <- trainControl(  
 method = "cv",  
 number = 5, #num\_cvs,  
 allowParallel = T,  
 # verboseIter = T,  
 savePredictions = "final",  
 summaryFunction = mapeSummary  
 )  
  
lm\_model <- train(Yield ~ .,   
 data = x\_train\_lm,  
 method = "glmnet",  
 trControl = lm\_control,  
 tuneLength = 10  
 )  
  
(lm\_model$bestTune)

## alpha lambda  
## 57 0.6 0.08159858

lm\_y\_pred <- predict(lm\_model, x\_test\_lm)  
  
  
# sources: http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/#elastic-net

#### CART Model ####  
  
set.seed(5)  
cart\_control <- trainControl(  
 method = "cv",  
 number = 5, #num\_cvs,  
 allowParallel = T,  
 # verboseIter = T,  
 savePredictions = "final",  
 summaryFunction = mapeSummary  
 )  
  
cart\_model <- train(Yield ~ .,   
 data = x\_train,  
 method = "rpart",  
 metric="RMSE",  
 trControl = cart\_control,  
 tuneLength = 10  
 )  
  
(cart\_model$bestTune)

## cp  
## 9 0.08949137

cart\_y\_pred <- predict(cart\_model, x\_test)  
# set.seed(5)  
cart\_control <- trainControl(  
 method = "cv",  
 number = 5, #num\_cvs,  
 allowParallel = T,  
 # verboseIter = T,  
 savePredictions = "final",  
 summaryFunction = mapeSummary  
 )  
  
cart\_model <- train(Yield ~ .,   
 data = x\_train,  
 method = "rpart",  
 metric="RMSE",  
 trControl = cart\_control,  
 tuneLength = 10  
 )  
  
(cart\_model$bestTune)

## cp  
## 6 0.02210552

cart\_y\_pred <- predict(cart\_model, x\_test)  
  
# http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/141-cart-model-decision-tree-essentials/#loading-required-r-packages

#### Random Forest Model ####  
set.seed(5)  
rf\_control <- trainControl(  
 method = "cv",  
 number = 5, #num\_cvs,  
 allowParallel = T,  
 # verboseIter = T,  
 savePredictions = "final",  
 summaryFunction = mapeSummary  
 )  
  
rf\_grid <- expand.grid(mtry = seq(5, 30, 5))  
  
rf\_model <- train(Yield ~ .,   
 data = x\_train,  
 method = "rf",  
 metric="RMSE",  
 trControl = rf\_control,  
 tuneGrid = rf\_grid,  
 ntree = 1000  
 )  
  
(rf\_model$bestTune)

## mtry  
## 4 20

rf\_y\_pred <- predict(rf\_model, x\_test)

#### Extreme Gradient Boosting Trees ####  
set.seed(5)  
xgb\_control <- trainControl(method = "cv",  
 number = 5, #num\_cvs,  
 allowParallel = T,  
 # verboseIter = T,  
 savePredictions = "final",  
 summaryFunction = mapeSummary  
 )  
  
xgb\_grid <- expand.grid(nrounds=250,  
 eta = c(0.025, .05, .1), #.05, #   
 max\_depth = c(5, 10, 20), #5, #  
 colsample\_bytree = seq(.25, 1, .25), #.25, #  
 gamma = 0,  
 min\_child\_weight = 0,  
 subsample = 1  
 )  
  
xgb\_model <- train(Yield ~ .,   
 data = x\_train,  
 method = "xgbTree",  
 trControl = xgb\_control,  
 tuneGrid = xgb\_grid  
 )  
  
xgb\_y\_pred <- predict(xgb\_model, x\_test)

1. Which tree-based regression model gives the optimal resampling and test set performance?

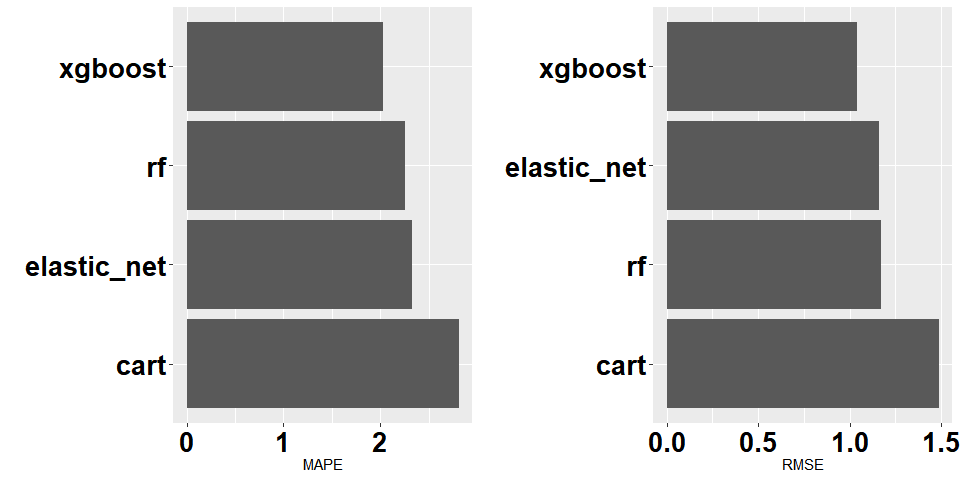
measure\_cv\_results <- function(train\_obj){  
 # measures the best cv results  
 colMeans(train\_obj$resample[, -4])  
}  
  
measure\_test\_results <- function(y\_pred, y\_test){  
 # measures the test set results  
 c(MAPE = MLmetrics::MAPE(y\_pred, y\_test) \* 100,  
 RMSE = MLmetrics::RMSE(y\_pred, y\_test))  
}  
  
  
# collect all the train model objects  
all\_models <- list(elastic\_net = lm\_model, cart = cart\_model, rf = rf\_model, xgboost = xgb\_model)  
  
# calculate the cv results  
all\_cv\_results <-   
 lapply(all\_models, measure\_cv\_results) %>%   
 data.frame() %>%   
 t() %>%   
 data.frame() %>%   
 tibble::rownames\_to\_column("model")  
  
# collect all the y predictions  
all\_y\_preds <- list(elastic\_net = lm\_y\_pred, cart = cart\_y\_pred, rf = rf\_y\_pred, xgboost = xgb\_y\_pred)  
  
# calculate the test results  
all\_test\_results <-   
 lapply(all\_y\_preds, function(x) measure\_test\_results(x, y\_test)) %>%   
 data.frame() %>%   
 t() %>%   
 data.frame() %>%   
 tibble::rownames\_to\_column("model")  
  
all\_cv\_results

## model MAPE RMSE Rsquared  
## 1 elastic\_net 2.326555 1.160337 0.6362241  
## 2 cart 2.809892 1.485834 0.4321730  
## 3 rf 2.252216 1.173212 0.6356945  
## 4 xgboost 2.024899 1.041066 0.7064779

all\_test\_results

## model MAPE RMSE  
## 1 elastic\_net 2.856225 2.0470192  
## 2 cart 2.725632 1.4398441  
## 3 rf 1.907353 0.9860895  
## 4 xgboost 2.177444 1.1745131

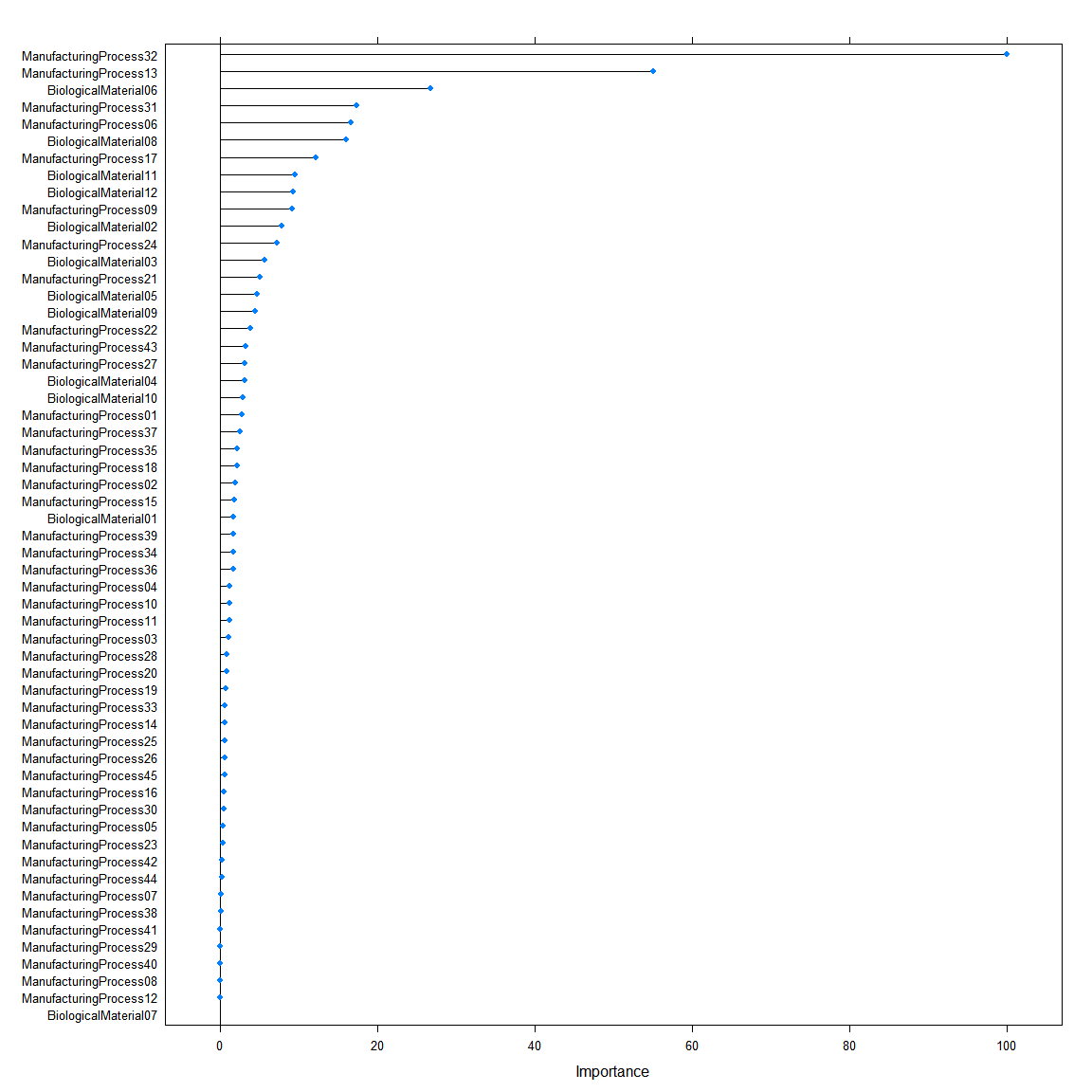
gridExtra::grid.arrange(  
 ggplot(all\_cv\_results) + geom\_col(aes(x = reorder(model, -MAPE), y = MAPE)) + coord\_flip() + xlab("") + theme(axis.text = element\_text(size=20, color="black", face = "bold")),  
 ggplot(all\_cv\_results) + geom\_col(aes(x = reorder(model, -RMSE), y = RMSE)) + coord\_flip() + xlab("") + theme(axis.text = element\_text(size=20, color="black", face = "bold")),  
 ncol = 2  
)



# all\_cv\_results[all\_cv\_results$model == "elastic\_net", "RMSE"] <- NA  
# all\_test\_results[all\_test\_results$model == "elastic\_net", "RMSE"] <- NA  
  
# y\_test\_lm

1. Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list?

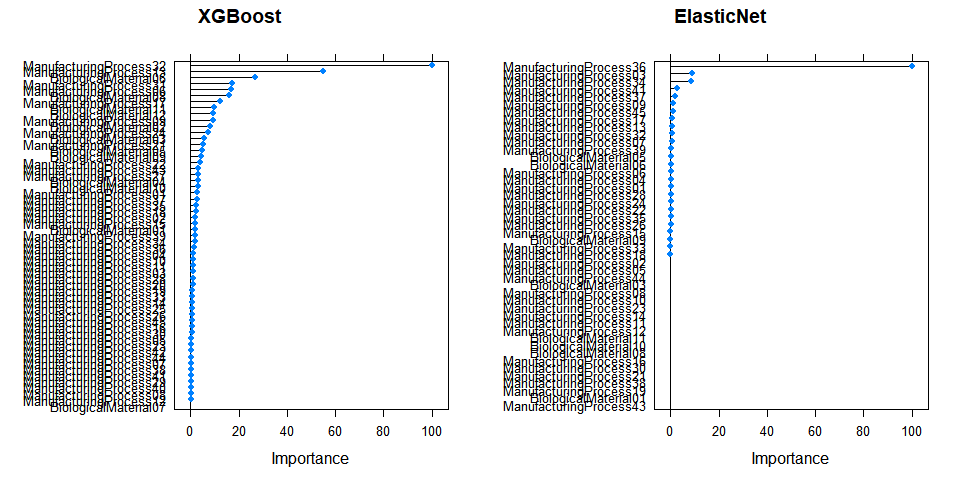
xgb\_var\_imp <- varImp(xgb\_model)  
plot(xgb\_var\_imp)



# character\_cols <- names(flux.in.data)[sapply(flux.in.data, is.character)]  
# flux.in.data[, (character\_cols) := lapply(.SD, function(x) iconv(x, to = "UTF-8")), .SDcols = character\_cols]

How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?

gridExtra::grid.arrange(  
 plot(xgb\_var\_imp, main = "XGBoost"),  
 plot(varImp(lm\_model), main = "ElasticNet"),  
 ncol = 2  
)



1. 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?