Applied Predictive Modeling - Kuhn and Johnson

Data 624 - Predictive Analytics

16 December 2019

Group Members:

Sang Yoon (Andy) Hwang

Contents

Getting Started

Dependencies

```
# Predicitve Modeling
libraries("AppliedPredictiveModeling", "mice", "caret", "tidyverse", "impute", "pls",
    "caTools", "mlbench", "randomForest", "party", "gbm", "Cubist", "rpart")
# Formatting Libraries
libraries("default", "knitr", "kableExtra", "gridExtra", "sqldf", "tibble")
# Plotting Libraries
libraries("ggplot2", "grid", "ggfortify", "rpart.plot")

# Data Wrangling
library(AppliedPredictiveModeling)
library(mice)
library(caret)
library(tidyverse)
library(pls)
library(caTools)
library(mlbench)
library(stringr)

# Formatting
library(default)
library(knitr)
library(kableExtra)

# Plotting
library(ggplot2)
library(grid)
library(ggfortify)
library(gridExtra)
```

Assignment 1

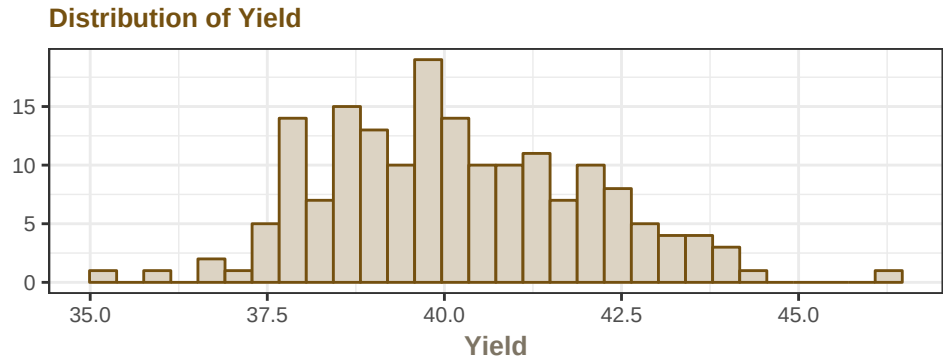
Kuhn and Johnson 6.3

A chemical manufacturing process for a pharmaceutical product was discussed in Sect.1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1% will boost revenue by approximately one hundred thousand dollars per batch:

(a). Start R and use these commands to load the data:

```
data("ChemicalManufacturingProcess")
```

The matrix `processPredictors` contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. Yield contains the percent yield for each run. Using a histogram, we examined the distribution of `Yield` and found that the response variable appears unimodal with a normal distribution.



(b). A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

`ManufacturingProcess03` has the largest volume of missing data followed by `ManufacturingProcess11`. Given that each variable is missing less than 25% of its data, we should impute that data. For our purposes, we will use MICE (Multiple Imputation with Chained Equations) methods.

MICE is based on the Gibbs sampler, a Markov chain that uses Monte Carlo methods. MICE iteratively draws estimates of missing values and parameters related to the distribution of said variables. Chained equations are generally faster than the Monte Carlo-based Gibbs sampler. MICE defaults to 5 imputations and predictive mean matching (PMM), the latter of which does a better job at keeping non-linear relationships within individual variables.

In addition to imputing using MICE, we also drop variables that have near zero variance - though this only removes one variable, and we still include it as a process step per the textbook's specifications.

Following imputation there is no more missing data in the set. We examined other imputation methods such as KNN but found no significant difference in summary statistics.

Table 1: Variables with Missing Values

Predictor	n	Predictor	n
ManufacturingProcess03	15	ManufacturingProcess02	3
ManufacturingProcess11	10	ManufacturingProcess06	2
ManufacturingProcess10	9	ManufacturingProcess01	1
ManufacturingProcess25	5	ManufacturingProcess04	1
ManufacturingProcess26	5	ManufacturingProcess05	1
ManufacturingProcess27	5	ManufacturingProcess07	1
ManufacturingProcess28	5	ManufacturingProcess08	1
ManufacturingProcess29	5	ManufacturingProcess12	1
ManufacturingProcess30	5	ManufacturingProcess14	1
ManufacturingProcess31	5	ManufacturingProcess22	1
ManufacturingProcess33	5	ManufacturingProcess23	1
ManufacturingProcess34	5	ManufacturingProcess24	1
ManufacturingProcess35	5	ManufacturingProcess40	1
ManufacturingProcess36	5	ManufacturingProcess41	1

(c). Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

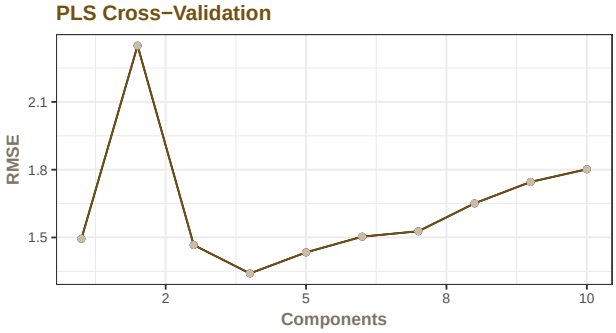
We will build a PLS model also known as partial least squares. PLS is a statistical method that fits a linear regression model by projecting the feature variables and response variable to some new space using a mapping function. PLS has certain advantages over other methods, including being more robust to multicollinearity and the issues attending it.

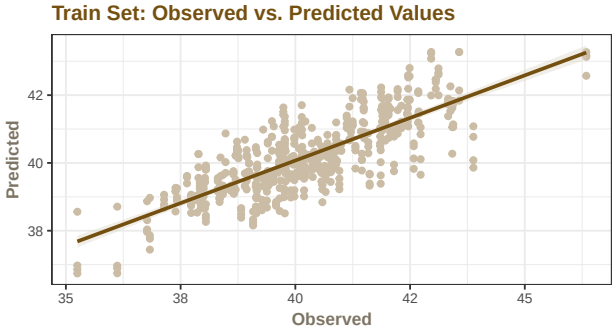
We partitioned the data into an 80-20 train-test split and preprocessed it by centering and scaling. We built a standard PLS model and evaluated the root mean summary areas to determine the optimal number of components to select. Tuning of the model yields the following performance:

Table 2: PLS Performance Metrics on Training Subset

RMSE	Rsquared	MAE
1.3408	0.5375	1.0339

The Baseline PLS model generates an RMSE of 1.34 and accounts for 53.75% of variance, as highlighted in the charts below:





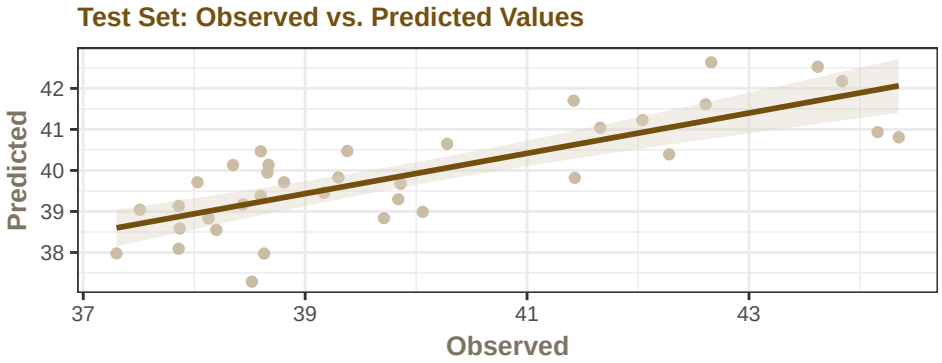
(d). Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

The R^2 is greater for the test data, with 62% of the data variance explained. The RMSE has also dropped from the training results of 1.34 to the test result of 1.31. There is also a slight increased in the MAE.

Table 3: PLS Performance Metrics on Test Subset

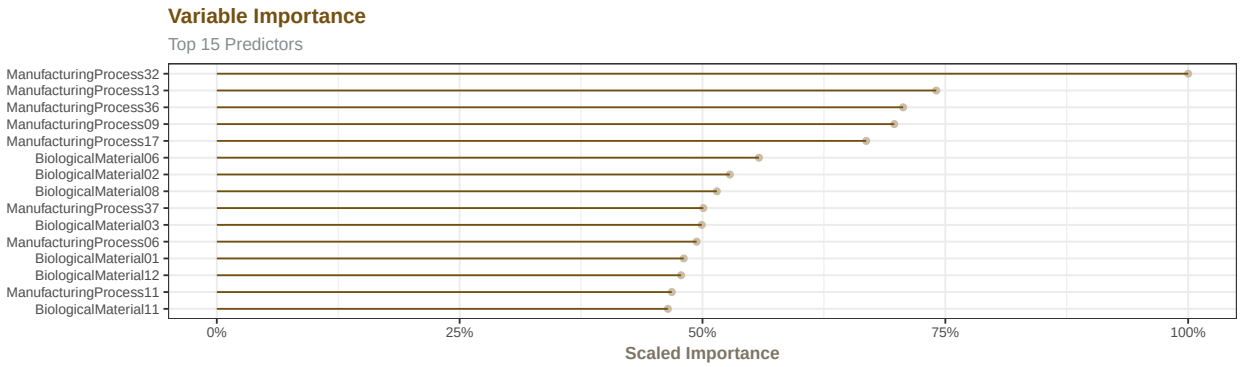
RMSE	Rsquared	MAE
1.3115	0.6183	1.0715

Deviation in the line fitted to predicted vs. observed values below suggested that the selected linear model may not provide the best predictions for `Yield`.



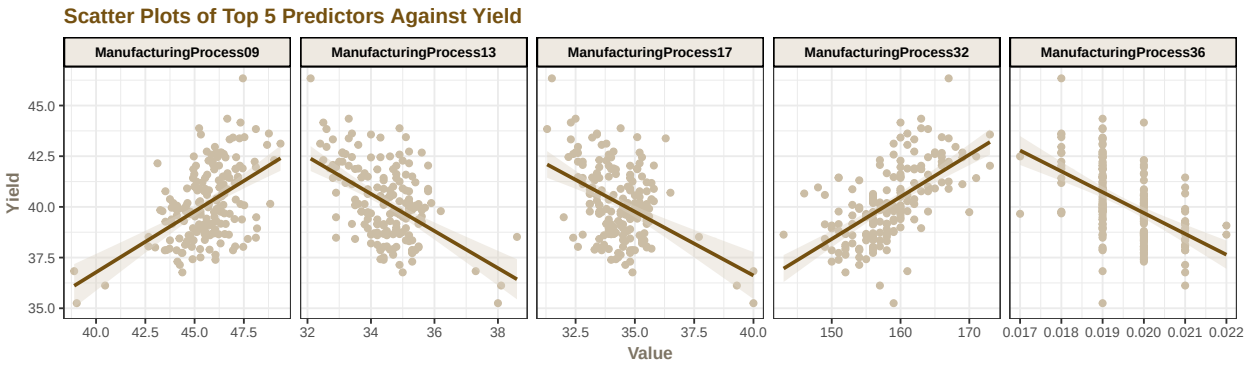
(e). Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

VarImp identifies the variables by their importance. `ManufacturingProcess32` is the most important predictor overall as well as within the group of other Manufacturing Process variables. `BiologicalMaterial06` ranked second and was the most important variable amongst the BiologicalMaterial group. The variable importance rankings are a mix; amonst the top 15 predictors, 7 are BiologicalMaterials variabels and 8 are ManufacturingProcess variables.



(f). Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?

A scatter plot helps to visualize relationship between our top five important predictors against the response variable `Yield`. All but `ManufacturingProcess32` show a moderate positive, linear relationship with `Yield`.



We further examined this relationship by analyzing the five response variables most correlated with with the `Yield`, of which `ManufacturingProcess32` was at the fore.

From a business point of view, the aim is to increase yield as this is connected to revenue. While we don't have specific into the mechanics of each manufacturing process, we can use this knowledge to adjust the processes to emulate the highest yield outputs.

Table 4: Variable Correlation with Yield

Variable	Yield
ManufacturingProcess32	0.6083
ManufacturingProcess09	0.5035
ManufacturingProcess17	-0.4258
ManufacturingProcess36	-0.5014
ManufacturingProcess13	-0.5037

Assignment 2

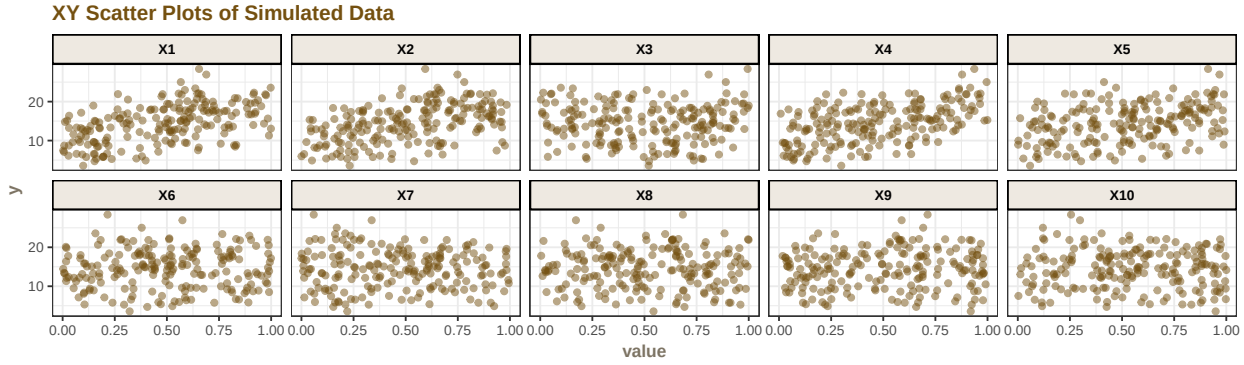
Kuhn and Johnson 7.2

Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data: $y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0,\sigma^2)$; where the x values are random variables uniformly distributed between $[0, 1]$ (there are also 5 other non-informative variables also created in the simulation).
The package 'mlbench' contains a function called 'mlbench.friedman1' that simulates these data. We convert the 'x' data from a matrix to a data frame. One reason is that this will give the columns names. The 'testData' code creates a list with a vector 'y' and a matrix of predictors 'x'. It also simulates a large test set to estimate the true error rate with good precision:

```
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)

## We convert the 'x' data from a matrix to a data frame One reason is that this
## will give the columns names.

trainingData$x <- data.frame(trainingData$x)
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)
```



(a). Tune several models on these data. For example:

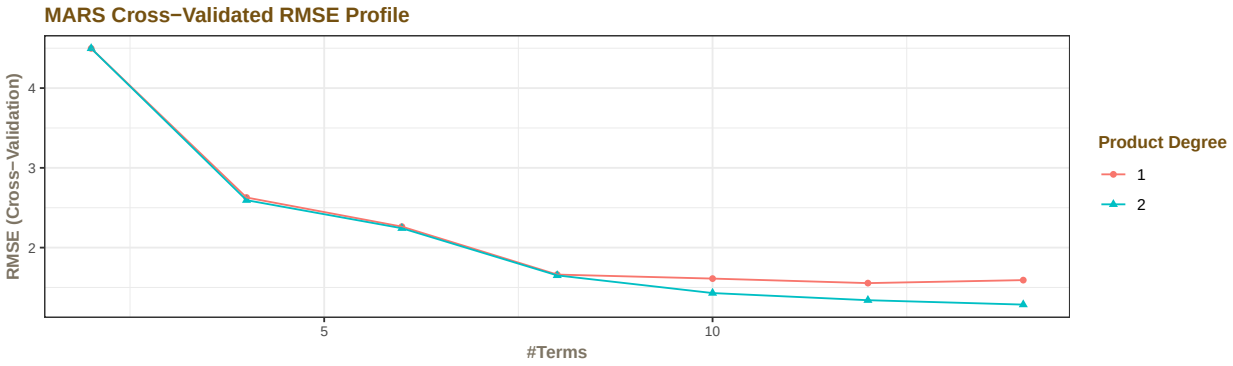
```
knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center",
    "scale"), tuneLength = 10)
knnModel
knnPred <- predict(knnModel, newdata = testData$x)
postResample(pred = knnPred, obs = testData$y)
```

Model 1-MARS Regression:

MARS (Multivariate Adaptive Regression Splines) is a non-parametric regression technique that automatically captures non-linearity and interaction between predictors. The basic MARS model takes the following form:

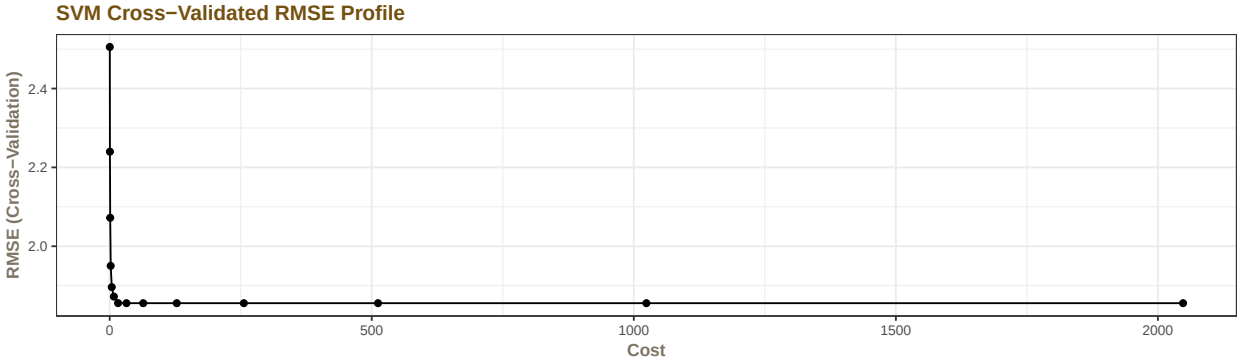
$$\hat{f} = \sum_{i=1}^{k} c_i B_i(x)$$

The model computes the sum of basis functions B multiplied by constant coefficients C. The basis function can either be a constant, a hinge function, or a product of hinge functions. By definition, a hinge function is a piecewise function that converges at a point known as a knot.



Model 2 SVM:

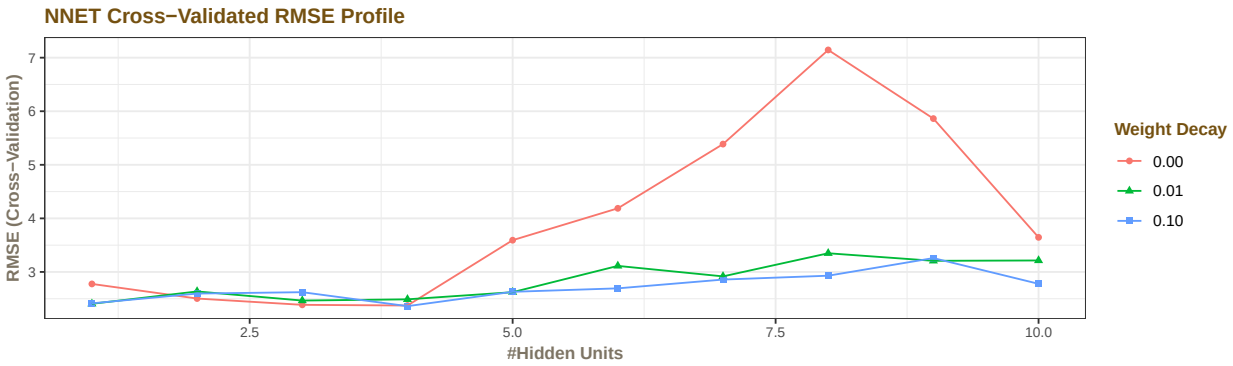
SVM (Support Vector Machies) is a method that can be applied to classification and regression tasks. SVM creates a hyperplane in n dimensional space which acts as a margin-maximizing classification boundary, either linear or nonlinear. This boundary classifies information from a feature space.



Model 3 NNET:

NNET (Neural Networks) is a method inspired by biological neuron systems. It uses a system of nodes that parallel the way neurons work, and is ideal for capturing non-linear relationships that would otherwise be complicated in most multiple linear regression models. NNET evolves internally based on the calculated weights of each input. The basic structure is shown below:

$$Y = \sum (weight \quad * \quad input) \quad + \quad bias$$



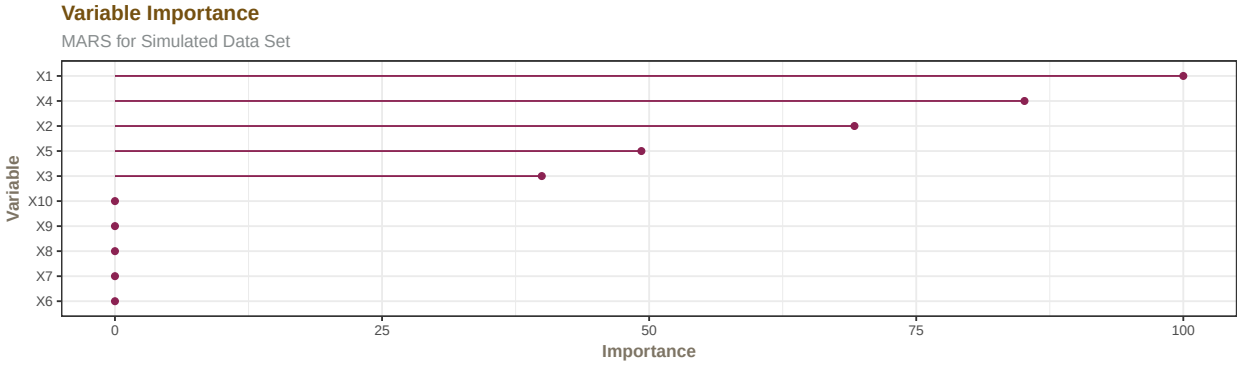
(b). Which models appear to give the best performance? Does MARS select the informative predictors (those named X1-X5)?

MARS appears to give the best performance based on RMSE, R^2, and MAE on test set. The performance metrics below show how other models stack up against the MARS model.

Table 5: Model Performance

	RMSE	RSquared	MAE
knnTrain	3.5962	3.5962	3.5962
knnTest	3.1751	0.6786	2.5443
MARSTrain	**4.4979**	**0.9331**	**3.7465**
MARSTest	**1.1723**	**0.9449**	**0.9325**
SVMTrain	2.5054	0.8600	1.9897
SVMTest	2.0702	0.8262	1.5725
NNETTrain	7.1443	0.7829	4.0248
NNETTest	2.2434	0.8042	1.6997

The chart below demonstrates that variables `X1` through `X5` are all important, with `X1` the most important. It is very likely that the lack of contribution from variables `X6` through `X10` bolster the models R^2 and RMSE performance, and that noise from these variables did not reduce the predictive strength of this model as it does in small quantities in the other three models.



Kuhn and Johnson 7.5

Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several nonlinear regression models.

We prepared data identically as in 6.3, imputing, removing near zero variance features, and preproecessing. Please refer to 6.3 for a more detailed look at the EDA involved with this data set.

We tuned KNN, NNET, MARS, and SVM models using specifications from the textbook.

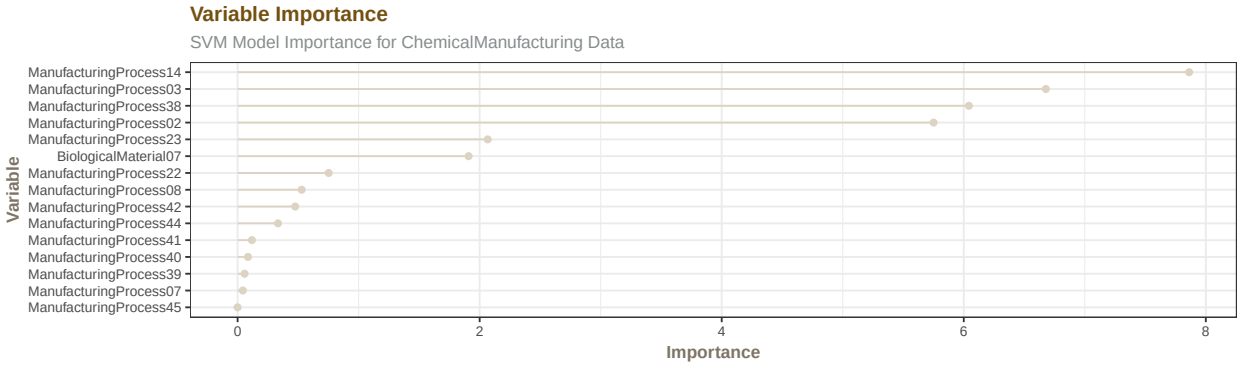
(a). Which nonlinear regression model gives the optimal resampling and test set performance?

SVM using a radial basis kernel outperformed the other models across all performance metrics, followed by MARS regression. In part b, we will address what variables are dominant in our SVM model.

Table 6: Model Performance on ChemicalManufacturing Data

	RMSE	RSquared	MAE
knnTrain	1.4808	0.4340	1.1588
knnTest	1.4414	0.5216	1.2076
MARSTrain	1.4473	0.6539	1.1317
MARSTest	1.2657	0.5836	1.0164
SVMTrain	**1.3613**	**0.6278**	**1.1264**
SVMTest	**1.1888**	**0.6533**	**0.9179**
NNETTrain	9.6023	0.3722	5.8435
NNETTest	1.7301	0.4157	1.4542

(b). Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?



For the SVM model, `ManufacturingProcess` variables dominate the importance rankings, with `ManufacturingProcess14` at the top. On the other hand, for the linear model `ManufacturingProcess32` was the most important variable with a number of `BiologicalProcess` variables within the top 10.

(c). Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Table 7: Correlation

VALUE	Yield
Yield	1.0000
ManufacturingProcess14	-0.0103
ManufacturingProcess38	-0.0865
ManufacturingProcess03	-0.1227
ManufacturingProcess37	-0.1593
ManufacturingProcess02	-0.1947

We examined the top 5 predictors identified as the most important variables before the importance measure dropped. There are some pretty clear differences in the data which might explain both the overall poor performance of the linear models as well as the improved significance of `ManufacturingProcess` variables in the non-linear models.

Of the `ManfuacturingProcess` variables, they appear to be either tight clusters or discrete values which predict an array of possible `Yield`. This is directly opposed the definition of linearly separable data base on earlier examination of correlation plots.

Assignment 3

Kuhn and Johnson 8.1

Recreate the simulated data from Exercise 7.2:

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

The code for the RF model is provided by the textbook.

What is the code actually doing? We should dive into the theory. The importance is calculated with the following formula:

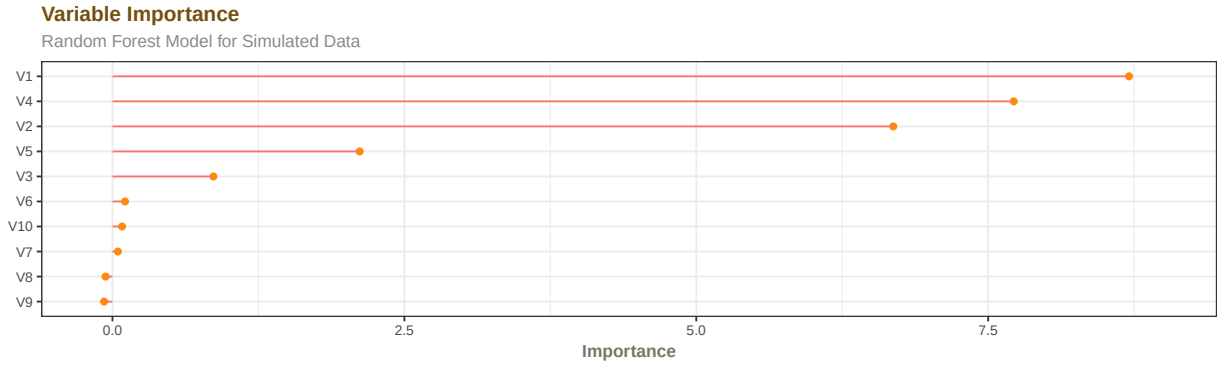
$$ni_j = W_{left(j)}C_{left(j)} - W_{right(j)}C_{right(j)}$$

Let's deconstruct the theory. ni_j stands for node importance. w_j is the weighted number of samples reaching node j. c_j is the impurity value of node j. $left(j)$ is the child node from the left split on node j and $right(j)$ is the child node from right split on node j. Once ni_j is calculated, the importance for each variable on a decision tree is calculated with formula I. Formula I is then normalized as shown as formula II. The final feature importance is the average over all trees. We simply find the sum of the features importance value and then divide by all trees T, shown in formula III.

$$I)\ fi_j = \frac{(\sum_{j\ node\ split\ on\ i})ni_j}{(\sum_{all\ nodes})ni_k}$$

$$II)\ \parallel fi_j \parallel = \frac{(fi_j)}{\sum_{all\ features} fi_j}$$

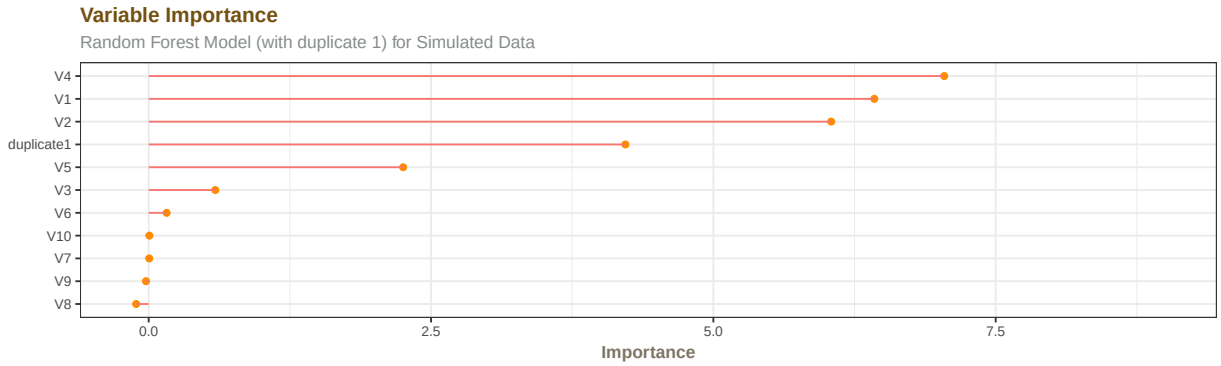
$$III)\ RFfi_j = \frac{(\sum_{all\ trees})\parallel fi_j \parallel}{T}$$



Variables `V6` through `V10` were some of the least important variables, all with a measure less than 1. Variables `V1` through `V5` were the most important, with variable `V1` in the top spot.

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

We add a feature that has a strong correlation of .93 with existing feature `V1` which we'll call `duplicate 1`



Note that `V1` decreased importance by roughly 2 measures. `V4` is now the most important predictor. It looks like the importance score for `V1` was partly absorbed by new predictor which underestimates true importance of `V1` - the score sum of `V1` and `duplicate1` are similar to the `V1` score in (a). It makes sense as `duplicate1` contains almost the same information as `V1`.

(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?

Table 8: Conditional vs Unconditional CForest Model: Variable Importance

features	RF	CF	RF.cor	CF.cor	CF.cond	CF.cor.cond
duplicate1	NA	NA	4.2213	7.6647	NA	1.3087
V1	8.7065	10.0562	6.4252	2.1014	3.3242	0.2776
V2	6.6871	7.7244	6.0433	6.8085	4.7155	3.9022
V3	0.8648	0.0203	0.5894	0.0189	0.0223	0.0149
V4	7.7190	10.6930	7.0444	9.8405	5.9425	5.5863
V5	2.1172	2.1893	2.2531	2.4539	0.7729	0.8364
V6	0.1072	0.0335	0.1589	0.0398	0.0027	-0.0023
V7	0.0465	0.0732	0.0042	0.0115	0.0188	0.0181
V8	-0.0588	-0.0501	-0.1113	-0.0208	-0.0015	-0.0061
V9	-0.0717	-0.0265	-0.0242	-0.0293	-0.0083	-0.0099
V10	0.0822	0.0060	0.0062	-0.0153	-0.0107	0.0102

We performed both `varimp(, conditional = T)` and `varimp(, conditional = F)` to compare `varimp` of `cforest` in terms of permutation importance and conditional permutation importance.

1. RF vs CF Given that no correlated term is added, the importance pattern is similar except for the fact that V4 is now the most important feature in CF.

2. RF vs CF (with correlated term added) Given that the correlated term is added, the importance score for `duplicate1` is much smaller in CF. This is the pinpoint difference between importance based on Gini coefficient (decision tree) and permutation test using p-value (conditional inference tree).

3. CF conditional vs CF with correlated term added and conditional When `conditional = T`, we perform conditional permutation test for measuring feature importance instead. Note that `duplicate1` has even smaller importance in `CF.cor.cond`

than in `CF.cor`. For `CF.cor.cond`, notice `V1` became 3rd most important feature when it was 2nd most important for `CF.cor`. This is because conditional permutation helps to uncover the spurious correlation between `V1` and `duplicate1`.

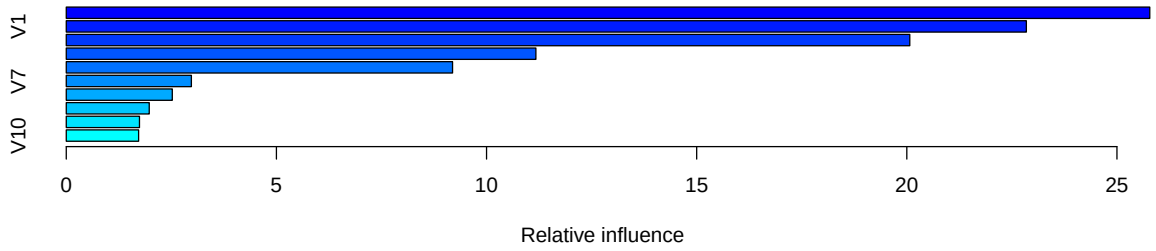
In summary, we learned that CF model surpresses the importance score of `duplicate1`, which helps to maintain the importance of `V1`. When `conditional = TRUE` in `varimp` for CF model, the importance score of `duplicate1` is even smaller.

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

Extracting the relative importance from a GBM object requires the use of the native model summary. The summary returns the variable name along with a measure of its influence on the target variable.

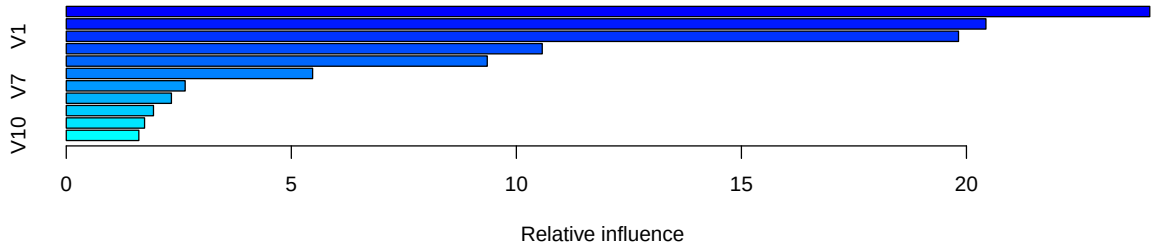
From the GBM tree, we can see `V4` is the most influential. `V1` and correlated `duplicate1` are also much more influential. `V6` through `V10` do not break the top half of our list of influential variables.

GBM Without Duplicate



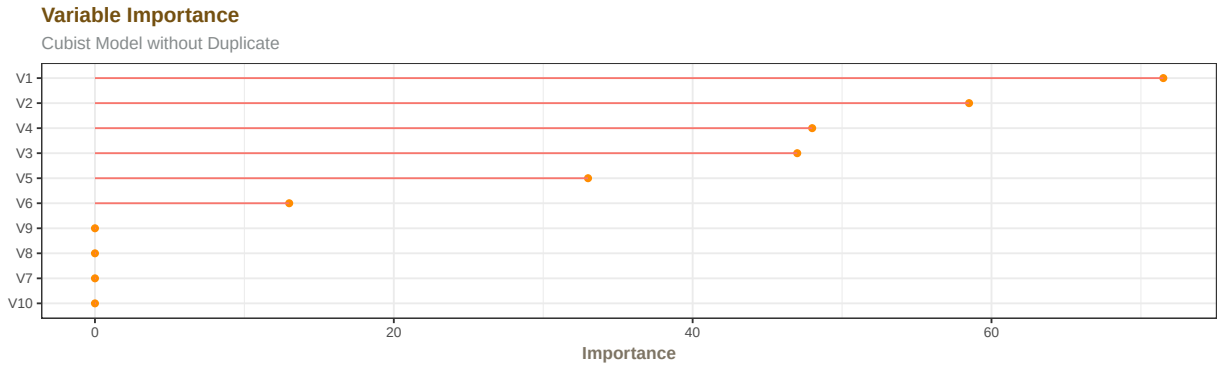

```
    var   rel.inf
V4   V4 25.780913
V1   V1 22.844642
V2   V2 20.070827
V5   V5 11.174466
V3   V3  9.191411
V7   V7  2.976852
V6   V6  2.523050
V9   V9  1.974093
V8   V8  1.742326
V10 V10  1.721421
```

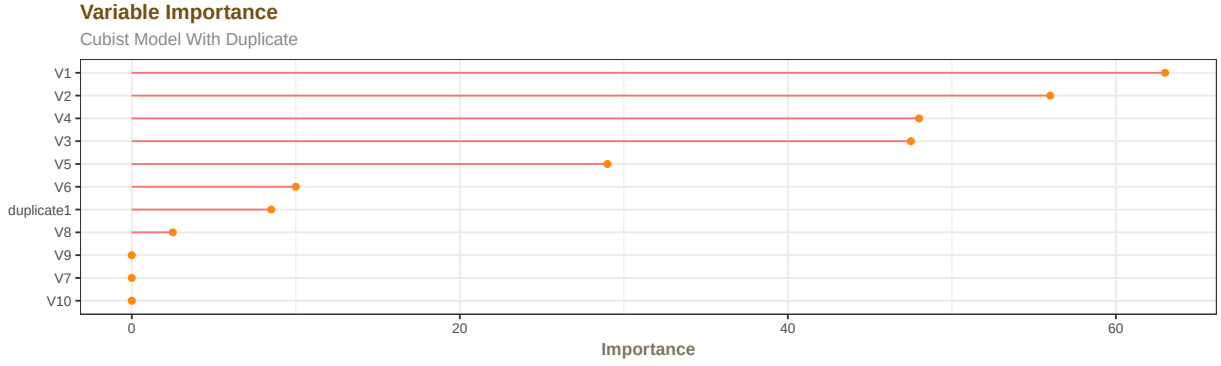
GBM With Duplicate



	var	rel.inf
V4	V4	24.072901
V2	V2	20.432881
V1	V1	19.823637
V5	V5	10.574969
V3	V3	9.353386
duplicate1	duplicate1	5.473609
V7	V7	2.641898
V6	V6	2.336609
V8	V8	1.937914
V9	V9	1.739826
V10	V10	1.612370

The Cubist model is a rather unique variation on trees. Each leaf in the tree - and each intermediate step between leafs - contains a linear regression model. Every layer in the tree alters the predicitons used within each leaf contained model. In other words, the selection of predictors in leaf n is based on the previous splits. Predictions are made via linear models on the terminal node.

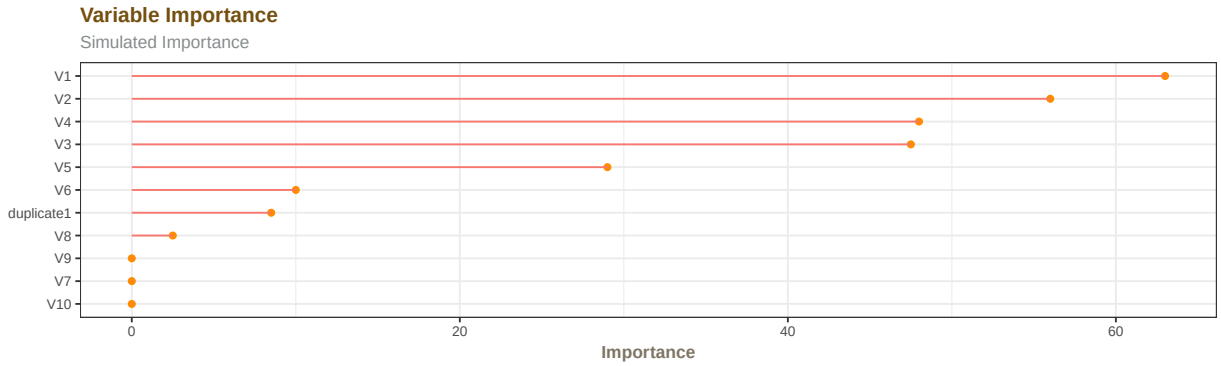




Kuhn and Johnson 8.2

Use a simulation to show tree bias with different granularities.

Basic regression trees split predictor variables into small groups based on response variable. According to the literature, "predictors with a higher number of distinct values are favored over more granular predictors."



Kuhn and Johnson 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?

Because the model on the right has both high learning rates and high bagging fractions, it does two things: first, it uses 90% of the variables in each tree, second, it uses 90% of the error for a given model.

Imagine a ten-variable model, with a bagging fraction of .9 every tree would have nine of the ten trees in it and only one variable would be different from the set of 9 trees in the first model each time. In this case, most of the possible break-points would be the same from tree to tree and different initial splits would only be possible when the dominant variable is removed. As a result, these modles would likely make the same first few decisions each time. Given that with a .9 learning rate 90% of the error is added in from each tree, in addition to consistently choosing from one or a few initial splits the models are also maximizing the contributions of those splits.

Given these learning rates and bagging fractions, the trees contributing to the Stochastic Boosted Tree in this example will make the same decisions repeatedely, yielding very few unique decisions overall. Lack of variation in initial choices means that the number of paths the learning can take is limited, and with a high learning rate a core set of variables is selected early on from the similarly built trees. In essence, the model never has the opportunity to evaluate other possible variables because the greediness of the model makes the same first few choices every time.

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

The .9 / .9 model would likely overfit the training data, because the variation in values within those most important variables may not reflect the general population. This model will be tuned to choose from sample members following the samples' distributions along the most important features at the expense of recognizing potential splits in other features which might be more common in the general population.

With a learning rate of .1 and a bagging fraction of .1, the left model is more likely to build truly weak predictors, from smallers sets of variables, consider more distinct breaking points, and therefore extend better to wild data not fully described by the first few variables in the importance summarise_layers.

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

Increasing the tree depth would affect both models, but differently.

For the model with bagging fraction and learning rate equal to .1, increasing the number of nodes in the tree would likely increase the importance of less important variables, creating a less polynomial slope. The reason for this is that making more decisions means giving weight to the variables and values where those decisions are made.

For the model with bagging fraction and learning rate equal to .9, increasing the depth would likely not change the slope of less important lower variables, just adding a new variable or two to the top.

In this model with high bagging fraction and learning rate we will still be making most of the same decisions from tree to tree. Increases in importance will come from nodes added downstream, and these are likely to be the same from tree to tree. As a result, you might find added importance low on the scale for those variables upon which the new nodes break, or for variables higher on the scale with a second or third break. This might lead to a reshuffling of upper nodes and a slight increase of one or two less important variables, but the overall slope quickly going to zero will be constrained by the high bagging fraction and learning rate.

Kuhn and Johnson 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:

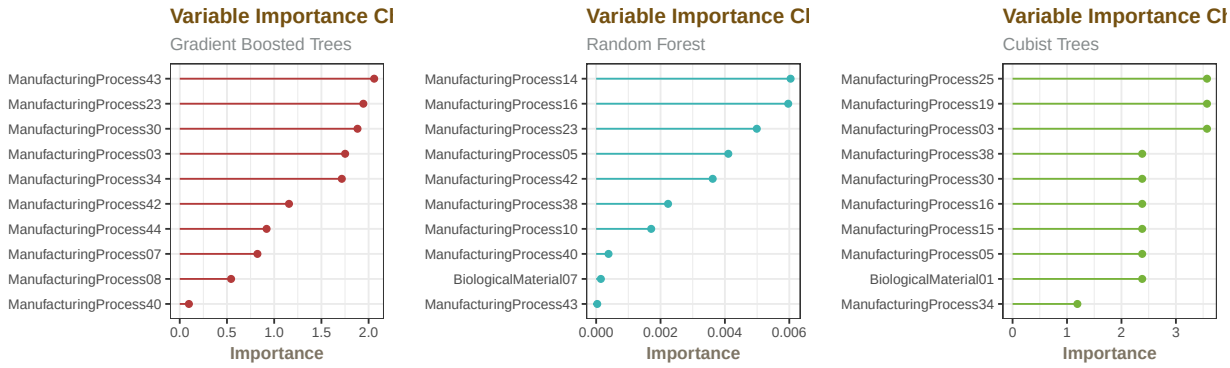
(a). Which tree-based regression model gives the optimal resampling and test set performance?

Table 9: Tree Model Performance on ChemicalManufacturing Data

	RMSE	RSquared	MAE
GBM	1.4669	0.5860	1.1585
GBMTest	1.1637	0.6722	0.9276
RFTrain	1.2344	0.5498	0.9307
RFTest	1.1169	0.7363	0.9037
CubistTrain	**0.9882**	**0.7519**	**0.7678**
CubistTest	**1.0700**	**0.7074**	**0.8232**

The Cubist model is the optimal based on the R^2 value. Cubist has a lower RMSE both on the train and test data across the different selected tree models.

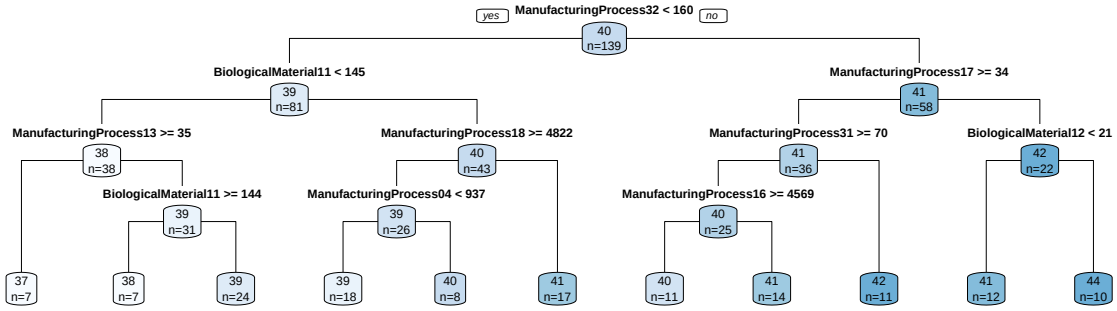
(b). Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?



In all the models the `ManufacturingProcess` variables dominate the list, with slight differences in position and influence.

Comparing the top 10 variables in each model reveals some strong differences. The Boosted Tree model sees a rather linear drop-off of importance through a list of exclusively `ManufacturingProcess` variables. The Random Forest model falls off slower in the first five variables but becomes rapidly linear; all but the last are `ManufacturingProcess` variables. The Cubist tree depreciates differently at seemingly discrete levels; other than the first and last variables, all are `ManufacturingProcess` variables.

(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?



Based on this regression tree, the differences between `ManufacturingProcess` and `BiologicalMaterials` is that the formerdifferentiate more observations at each break and so break first. While this increases node purity quickly, the final decisions seem to be based rather wholly on the latter variables. Accordingly, looking only at `ManufacturingProcess` variables or prunning too early could lead to overfitting.

R Script

```
## Dependencies
# SOURCE DEFAULT SETTINGS
### UNIVERSAL DATA SOURCING & DEFAULT SETTINGS FOR PROJECT
library(knitr)
library(kableExtra)
library(default)
library(ggplot2)
library(easypackages)
library(formatR)

# Load .rd file
setwd('~/GitHub/CUNY_DATA_624/Homework-Two')
load(file = "hw2.rds")

# SOURCE DEFAULT SETTINGS
source('~/GitHub/CUNY_DATA_624/Homework-Two/defaults.R')
# SOURCE HW ANSWERS
#source('~/GitHub/CUNY_DATA_624/Homework-Two/answers_final.R')
#load(file = "hw2nnet_mod2.rds")
load(file = "hw2.rds")

# SOURCE HW ANSWERS

# DEPENDENCIES
# Predicitve Modeling
libraries('AppliedPredictiveModeling',
          'mice',
          'caret',
          'tidyverse',
          'impute','pls','caTools','mlbench',
          'randomForest','party','gbm','Cubist','rpart')
# Formatting Libraries
libraries('default', 'knitr',
          'kableExtra','gridExtra','sqldf')
# Plotting Libraries
libraries('ggplot2', 'grid',
          'ggfortify','rpart.plot')


# Data Wrangling
library(AppliedPredictiveModeling); library(mice);
library(caret); library(tidyverse); library(pls);
library(caTools); library(mlbench); library(stringr);
# Formatting
library(default); library(knitr); library(kableExtra);
# Plotting
library(ggplot2); library(grid); library(ggfortify)
```

```r
# THEME COLORS
dark_gold <- "#745010"
medium_gold <- "#cbbda5"
light_gold <- "#dcd3c3"

# SET SEED

set.seed(58677)

# ASSIGNMENT 1
# KJ 6.3
data("ChemicalManufacturingProcess")

# (6.3a)
Plt_CMP.Yield <-ggplot(ChemicalManufacturingProcess, aes(x = Yield))+
  geom_histogram(color=light_gold, fill = light_gold)+
  scale_x_continuous(labels = scales::number_format(accuracy = .1))+
  labs(title="Distribution of Yield") +
  theme_bw()+theme(plot.title = element_text(color="#745010",
                                             size=10, face="bold"),
                   axis.title.y = element_blank())

# (6.3b)

# Total NA Values
#na_table<- table(is.na(ChemicalManufacturingProcess))
CMP_na <- ChemicalManufacturingProcess %>%
  select(-Yield) %>%
  summarise_all(funs(sum(is.na(.)))) %>%
  t() %>% as.data.frame() %>%
  rownames_to_column("Predictor") %>%
  filter(V1 > 0) %>%
  arrange(desc(V1)) %>%
  rename(n=V1)

CMP_na_left <-CMP_na %>%
  slice(1:14);

CMP_na_right <- CMP_na %>%
  slice(15:28);

CMP.total_na <- cbind(CMP_na_left, ` `=" ", CMP_na_right)

# use mice w/ default settings to impute missing data
miceImp <- mice(ChemicalManufacturingProcess, printFlag = FALSE)

# add imputed data to original data set
CMP_DF <-mice::complete(miceImp)

# (6.3c)

#code
set.seed(58677)   #  set seed to ensure you always have same random numbers generated
```

```
sample = sample.split(CMP_DF, SplitRatio = 0.80)
# splits the data in the ratio mentioned in SplitRatio.
# After splitting marks these rows as logical TRUE and the the remaining are marked as logical FALSE

chem_train =subset(CMP_DF,sample ==TRUE) # creates a training dataset named train1
# with rows which are marked as TRUE

chem_test=subset(CMP_DF, sample==FALSE)

#code
pls_model <- plsr(Yield~., data=chem_train,
                  method = 'kernelpls',
                  scale = TRUE,
                  center = TRUE)


pls_model2 <- plsr(Yield~., data=chem_train,
                   method = 'kernelpls',
                   scale = TRUE,
                   center = TRUE,
                   ncomp =41)


#  Train Metrics
train_eval=data.frame('obs' = chem_train$Yield, 'pred' =pls_model$fitted.values)
colnames(train_eval) <- c('obs', 'pred')

# (6.3d)
#code
# #Test Predictions & Metrics
pls2_pred<- predict(pls_model2, chem_test, ncomp=41)

pls2test_eval=data.frame('obs' = chem_test$Yield, 'pred' =pls2_pred)

colnames(pls2test_eval) <- c('obs', 'pred')



caret::defaultSummary(pls2test_eval)%>%
  kable(caption="PLS Performance Metrics on Test Subset") %>%
  kable_styling()# %>% row_spec()

eval_plot <- ggplot(pls2test_eval, aes(obs, pred)) +
  labs(title="Observed vs. Predicted Results for Test Data",
       subtitle="Partial Least Squares Model")+
  geom_point()+
  coord_flip()

CMP.sample = sample.split(CMP_DF, SplitRatio = 0.80)
# splits the data in the ratio mentioned in SplitRatio.
# After splitting marks these rows as logical TRUE and the the remaining
# are marked as logical FALSE
CMP.train = subset(CMP_DF, CMP.sample ==TRUE)
```

```r
# creates a training dataset named train1 with rows which are marked as TRUE
CMP.test = subset(CMP_DF, CMP.sample==FALSE)

# Train model
CMP.pls.fit <- train(Yield~., data=CMP.train, method = 'pls',
                     preProcess=c('zv', 'nzv', 'center', 'scale'),
                     trControl = trainControl(method = "cv", number = 5,
                                              savePredictions = T),
                     tuneLength=10)
CMP.pls.fit.obs_vs_pred <- cbind(Observed = CMP.pls.fit$finalModel$model$.outcome,
                                 Predicted = CMP.pls.fit$finalModel$fitted.values) %>%
  as.data.frame()
Plt_CMP.fit.obs_vs_pred <- ggplot(CMP.pls.fit.obs_vs_pred, aes(Observed, Predicted)) +
  geom_point(color=medium_gold) +
  geom_smooth(method="lm", color=dark_gold, fill=light_gold)+
  scale_x_continuous(labels = scales::number_format(accuracy = 1))+
  scale_y_continuous(labels = scales::number_format(accuracy = 1))+
  labs(title="Train Set: Observed vs. Predicted Values")+
  theme_bw()+
  theme()

#  Train Metrics
CMP.pls.train.perf <- CMP.pls.fit$results %>%
  as.data.frame() %>%
  filter(RMSE==min(RMSE)) %>%
  select(RMSE, Rsquared, MAE)

Plt_CMP.RMSE <- ggplot(CMP.pls.fit) +
  geom_line(color=dark_gold) +
  geom_point(color=medium_gold)
+theme_bw()+
  theme()+
  labs(title="PLS Cross-Validation", y="RMSE", x="Components")+
  scale_x_continuous(labels = scales::number_format(accuracy = 1))

# (6.3d)
## Test Predictions & Metrics
CMP.pls.pred <- predict(CMP.pls.fit, CMP.test)
CMP.pls.test.perf <- postResample(pred = CMP.pls.pred, obs = CMP.test$Yield) %>%
  t() %>%
  as.data.frame()

CMP.pls.test.obs_vs_pred <- cbind(Predicted = CMP.pls.pred, Observed = CMP.test$Yield)%>%
  as.data.frame()

Plt_CMP.test.obs_vs_pred <- ggplot(CMP.pls.test.obs_vs_pred, aes(Observed, Predicted)) +
  geom_point(color=medium_gold) +
  geom_smooth(method="lm", color=dark_gold, fill=light_gold)+
  labs(title="Test Set: Observed vs. Predicted Values")+
  scale_x_continuous(labels = scales::number_format(accuracy = 1))+
  scale_y_continuous(labels = scales::number_format(accuracy = 1))+
  theme_bw()+
  theme(plot.title = element_text(color="#745010", size=10, face="bold"))
```

```r
# (6.3e)
CMP.pls.imp <- caret::varImp(CMP.pls.fit, scale=T)

CMP.pls.imp.df <- CMP.pls.imp$importance %>%
  as.data.frame() %>%
  rownames_to_column("Variable")%>%
  arrange(desc(Overall))

Plt_CMP.pls.imp <- CMP.pls.imp.df %>%
  top_n(15, Overall) %>%
  ggplot(aes(x=reorder(Variable, Overall), y=Overall)) +
  geom_point(colour = medium_gold) +
  geom_segment(aes(x=Variable,xend=Variable,y=0,yend=Overall),colour = dark_gold) +
  labs(title="Variable Importance",
       subtitle="Top 15 Predictors",
       x="Variable",
       y="Scaled Importance")+
  scale_y_continuous(labels = function(x) paste0(x, "%"))+
  coord_flip()+
  theme_bw()+
  theme(axis.title.y = element_blank())

# (6.3f)
# Scatter Plot Comparison
CMP.varImp.top5 <- CMP.pls.imp.df %>%
  top_n(5, Overall)

CMP_DF.gather <- CMP_DF %>%
  gather(Variable, Value, -Yield)

Plt_CMP.Scatter <- CMP_DF.gather %>%
  filter(Variable %in% CMP.varImp.top5$Variable) %>%
  ggplot(aes(x=Value, y=Yield)) +
  geom_point(color=medium_gold)+
  geom_smooth(method = "lm", color=dark_gold, fill=light_gold)+
  labs(title="Scatter Plots of Top 5 Predictors Against Yield")+
  facet_wrap(~Variable, scales = "free_x", nrow = 1)+
  theme_bw()+
  theme()

# Correlation
CMP_DF.subset <- CMP_DF[(names(CMP_DF) %in% c(CMP.varImp.top5$Variable, "Yield"))]
CMP_DF.corr <-as.data.frame(as.matrix(cor(CMP_DF.subset)))
CMP_DF.corr.tbl <- CMP_DF.corr %>%
  select(Yield) %>%
  rownames_to_column('Variable') %>%
  filter(Variable!="Yield") %>%
  arrange(desc(Yield))

# ASSIGNMENT 2
# KJ 7.2; KJ 7.5
# The package `mlbench` contains a function called `mlbench.friedman1`
# that simulates these data:
```

```r
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)
## We convert the 'x' data from a matrix to a data frame
## One reason is that this will give the columns names.
trainingData$x <- data.frame(trainingData$x)
## Look at the data using
#featurePlot(trainingData$x, trainingData$y)
## or other methods.
## This creates a list with a vector 'y' and a matrix
## of predictors 'x'. Also simulate a large test set to
## estimate the true error rate with good precision:
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)


#over ride set seed from sample data simulation
set.seed(58677)

#KNN provided by literature
knnModel <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10)
knnModel
knnPred <- predict(knnModel, newdata = testData$x)
## The function 'postResample' can be used to get the test set performance values
postResample(pred = knnPred, obs = testData$y)

# instructions from text
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)
trainingData$x <- data.frame(trainingData$x)
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)
#featurePlot(trainingData$x, trainingData$y)

# created ggplot instead of featurePlot()

Sim.featurePlot <- trainingData %>%
  as.data.frame() %>%
  gather(x, value, -y) %>%
  mutate(x = str_replace(x, "x.","")) %>%
  arrange(desc(x)) %>%
  mutate(x = as.factor(x))
Sim.featurePlot$x <- factor(Sim.featurePlot$x,
                            levels = c("X1","X2", "X3", "X4",
                                       "X5", "X6", "X7", "X8", "X9","X10"))
Plt_Sim.featurePlot <- ggplot(Sim.featurePlot, aes(value, y)) +
  geom_point(color=dark_gold, alpha=.5)+
  facet_wrap(~ x, nrow=2, scales = "fixed")+
  theme_bw()+theme()+
  labs(title="XY Scatter Plots of Simulated Data")
```

```r
# revert seed back to our set group number:
set.seed(58677)

# (7.2a)
#mars
marsGrid <- expand.grid(degree =1:2, nprune=seq(2,14,by=2))
hw2mars_mod <- train(x = trainingData$x,
                     y = trainingData$y,
                     method='earth',
                     tuneGrid = marsGrid,
                     trControl = trainControl(method = "cv"))

hw2mars_modplot<- ggplot(hw2mars_mod)+theme_bw()+
  theme()+
  labs(title="MARS Cross-Validated RMSE Profile")

#svm
hw2svm_mod <- train(x = trainingData$x,
                    y = trainingData$y,
                    method='svmRadial',
                    tuneLength = 14,
                    trControl = trainControl(method = "cv"))
#hw2svm_mod$finalModel
hw2svm_modplot<- ggplot(hw2svm_mod)+theme_bw()+
  theme()+
  labs(title="SVM Cross-Validated RMSE Profile")


##nnet (Taken from Andy)
# hyperparameter tuning for nnet
nnetGrid <- expand.grid(.size = c(1:10), .decay = c(0, 0.01, .1))

hw2nnet_mod <- train(trainingData$x,
                     trainingData$y,
                     method = "nnet",
                     tuneGrid = nnetGrid,
                     trControl = trainControl(method="cv"),
                     ## Automatically standardize data prior to modeling and prediction
                     preProc = c("center", "scale"),
                     linout = TRUE,
                     trace = FALSE,
                     MaxNWts = 10 * (ncol(trainingData$x) + 1)  + 10 +  1,
                     maxit = 500)

hw2nnet_modplot<- ggplot(hw2nnet_mod)+theme_bw()+
  theme()+labs(title="NNET Cross-Validated RMSE Profile")


# (7.2b)
#knn pred given to us
hw2marsPred <- predict(hw2mars_mod, newdata = testData$x)
hw2svmPred <- predict(hw2svm_mod, newdata = testData$x)
hw2nnetPred <- predict(hw2nnet_mod, newdata = testData$x)
```

```
knn_performance <- postResample(pred = knnPred, obs = testData$y)
hw2marsPerf <- postResample(pred = hw2marsPred, obs = testData$y)
hw2svmPerf <- postResample(pred = hw2svmPred, obs = testData$y)
hw2nnetPerf <- postResample(pred = hw2nnetPred, obs = testData$y)

hw2.2.bperformance_table <- rbind("knnTrain"=c("RMSE"=max(knnModel$results$RMSE),
                                               "RSquared"=max(knnModel$results$RMSE),
                                               "MAE"=max(knnModel$results$RMSE)),
                                  "knnTest"=knn_performance,
                                  "MARSTrain"=c("RMSE"=max(hw2mars_mod$results$RMSE),
                                                "RSquared"=max(hw2mars_mod$results$Rsquared),
                                                "MAE"=max(hw2mars_mod$results$MAE)),
                                  "MARSTest"=hw2marsPerf,
                                  "SVMTrain"=c(max(hw2svm_mod$results$RMSE),
                                               max(hw2svm_mod$results$Rsquared),
                                               max(hw2svm_mod$results$MAE)),
                                  "SVMTest"=hw2svmPerf,
                                  "NNETTrain"=c(max(hw2nnet_mod$results$RMSE),
                                                max(hw2nnet_mod$results$Rsquared),
                                                max(hw2nnet_mod$results$MAE)),
                                  "NNETTest"=hw2nnetPerf) %>%
  kable(caption="Model Performance", digits=4) %>%
  kable_styling() %>%
  row_spec() %>%
  row_spec(row=3:4, background ="#d9f2e6")

#hw2marsImp <- varImp(hw2mars_mod)
#hw2marsImptbl <- hw2marsImp$importance %>%
##kable(caption="MARS Model - Variable Importance", digits=2) %>% kable_styling()

hw2marsImp <- caret::varImp(hw2mars_mod)
#hw2marsImp<-hw2marsImp%>%
#    mutate(Variable = row.names(hw2marsImp))%>%
#    remove_rownames()%>%
#    select(Variable, Overall)%>%
#    arrange(desc(Overall))

hw2marsImp<-hw2marsImp$importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  arrange(Overall) %>%
  mutate(rowname = forcats::fct_inorder(rowname ))

hwmarsimp_plot <- ggplot(head(hw2marsImp, 15),
                         aes(x=reorder(rowname, Overall), y=Overall)) +
  geom_point(colour = 'violetred4') +
  geom_segment(aes(x=rowname,xend=rowname,y=0,yend=Overall),colour = 'violetred4') +
  labs(title="Variable Importance",
       subtitle="MARS for Simulated Data Set", x="Variable", y="Importance")+
  coord_flip()+
  theme_bw()+
  theme()
```

```r
# (7.5a)
##knn
hw2knn_mod2 <- train(Yield~.,
                     data=chem_train,
                     method = "knn",
                     preProc = c("center", "scale"),
                     tuneLength = 10)

#nnet
nnetGrid_75 <- expand.grid(.decay = c(0, 0.01, .1),
                           .size = c(1:10),
                           .bag = FALSE)

hw2nnet_mod2 <- train(Yield~.,
                      data=chem_train,
                      method = "avNNet",
                      tuneGrid = nnetGrid_75,
                      preProc = c("center", "scale"),
                      linout = TRUE,
                      trace = FALSE,
                      MaxNWts = 10 * (ncol(chem_train) + 1) + 5 + 1,
                      maxit = 500)

#MARS
# Define the candidate models to test
marsGrid_75 <- expand.grid(.degree = 1:2, .nprune = 2:38)

hw2mars_mod2 <- train(Yield~.,
                      data=chem_train,
                      method = "earth",
                      tuneGrid = marsGrid_75,
                      trControl = trainControl(method = "cv"))

#SVM
hw2svm_mod2 <- train(Yield~.,
                     data=chem_train,
                     method = "svmRadial",
                     preProc = c("center", "scale"),
                     tuneLength = 14,
                     trControl = trainControl(method = "cv"))


#model performances
#knn pred given to us
hw2knnPred2 <- predict(hw2knn_mod2, newdata = chem_test)
hw2nnetPred2 <- predict(hw2nnet_mod2, newdata = chem_test)
hw2marsPred2 <- predict(hw2mars_mod2, newdata = chem_test)
hw2svmPred2 <- predict(hw2svm_mod2, newdata = chem_test)

hw2knnPerf2 <- postResample(pred = hw2knnPred2, obs = chem_test$Yield)
hw2marsPerf2 <- postResample(pred = hw2marsPred2, obs = chem_test$Yield)
hw2svmPerf2 <- postResample(pred = hw2svmPred2, obs = chem_test$Yield)
hw2nnetPerf2 <- postResample(pred = hw2nnetPred2, obs = chem_test$Yield)
```

```
hw2.2.cperformance_table <- rbind("knnTrain"=c("RMSE"=max(hw2knn_mod2$results$RMSE),
                                               "RSquared"=max(hw2knn_mod2$results$Rsquared),
                                               "MAE"=max(hw2knn_mod2$results$MAE)),
                                  "knnTest"=hw2knnPerf2,
                                  "MARSTrain"=c("RMSE"=max(hw2mars_mod2$results$RMSE),
                                                "RSquared"=max(hw2mars_mod2$results$Rsquared),
                                                "MAE"=max(hw2mars_mod2$results$MAE)),
                                  "MARSTest"=hw2marsPerf2,
                                  "SVMTrain"=c(max(hw2svm_mod2$results$RMSE),
                                               max(hw2svm_mod2$results$Rsquared),
                                               max(hw2svm_mod2$results$MAE)),
                                  "SVMTest"=hw2svmPerf2,
                                  "NNETTrain"=c(max(hw2nnet_mod2$results$RMSE),
                                                max(hw2nnet_mod2$results$Rsquared),
                                                max(hw2nnet_mod2$results$MAE)),
                                  "NNETTest"=hw2nnetPerf2) %>%
  kable(caption="Model Performance on ChemicalManufacturing Data", digits=4) %>%
  kable_styling() %>%
  row_spec() %>%
  row_spec(row=5:6, background ="#d9f2e6")

# (7.5b)
#hw2svmImp2 <- varImp(hw2svm_mod2)
#hw2svmImptbl2 <- hw2svmImp2$importance %>% kable(caption="SVM Model - Variable Importance", digits=2)

hw2svmImp2 <- caret::varImp(hw2svm_mod2)

hw2svmImp2<-hw2svmImp2$importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  arrange(Overall) %>%
  mutate(rowname = forcats::fct_inorder(rowname ))

hwsvmimp_plot2 <- ggplot(head(hw2svmImp2, 15), aes(x=reorder(rowname, Overall), y=Overall)) +
  geom_point(colour = light_gold) +
  geom_segment(aes(x=rowname,xend=rowname,y=0,yend=Overall),colour = light_gold) +
  labs(title="Variable Importance",
       subtitle="SVM Model Importance for ChemicalManufacturing Data",
       x="Variable",
       y="Importance")+
  coord_flip()+
  theme_bw()+
  theme()

# (7.5c)
#alterate apprach (use plot importance to identify top few important features)
hw2imp <- CMP_DF %>%select(Yield,
                           ManufacturingProcess14,
                           ManufacturingProcess02,
                           ManufacturingProcess03,
                           ManufacturingProcess38,
                           ManufacturingProcess37 )
```

```r
hw2cor_pre_df<- as.data.frame(as.matrix(cor(hw2imp)))

hw2cor_df<-tibble::rownames_to_column(hw2cor_pre_df, "VALUE")

hw2cor_df2<-sqldf("select VALUE, Yield from hw2cor_df order by Yield desc")%>%
  kable(caption="Correlation") %>%
  kable_styling()


# ASSIGNMENT 3
# KJ 8.1-8.3; KJ 8.7

# (8.1a)
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"

# revert seed back to our set group number:
set.seed(58677)

model1 <- randomForest(y ~ ., data = simulated,
                       importance = TRUE,
                       ntree = 1000)

rfImp1 <- varImp(model1, scale = FALSE)

rfImp1.df <-tibble::rownames_to_column(as.data.frame(as.matrix(varImp(model1,
                                                                      scale = FALSE))),
                                       "VALUE")    #as.data.frame(as.matrix(varimp(hw3_rfmodel3)))

#colnames(hw3_rfdt3) <- c("Value","Importance")

rfImp1plot<-ggplot(rfImp1.df, aes(x=reorder(VALUE, Overall), y=Overall)) +
  geom_point(color ='darkorange') +
  geom_segment(aes(x=VALUE,
                   xend=VALUE,
                   y=0,yend=Overall,
                   color = 'darkorange')) +
  labs(title="Variable Importance",
       subtitle="Random Forest Model for Simulated Data",
       x="",
       y="Importance")+
  coord_flip()+
  theme_bw()+
  theme(legend.position = "none") +
  scale_y_continuous(limits = c(-0.15, 9))

# (8.1b)
simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1

hw3model2 <- randomForest(y ~ ., data = simulated,
```

```
                          importance = TRUE,
                          ntree = 1000)
rfImp2 <- varImp(hw3model2, scale = FALSE)

rfImp2.df <-tibble::rownames_to_column(as.data.frame(as.matrix(varImp(hw3model2,
                                                                      scale = FALSE))),
                                       "VALUE")    #as.data.frame(as.matrix(varimp(hw3_rfmodel3)))

rfImp2plot<-ggplot(rfImp2.df, aes(x=reorder(VALUE, Overall), y=Overall)) +
  geom_point(color ='darkorange') +
  geom_segment(aes(x=VALUE,xend=VALUE,y=0,yend=Overall, color = 'darkorange')) +
  labs(title="Variable Importance",
       subtitle="Random Forest Model (with duplicate 1) for Simulated Data",
       x="",
       y="Importance")+
  coord_flip()+
  theme_bw()+
  theme(legend.position = "none") +
  scale_y_continuous(limits = c(-0.15, 9))



#hw3rf2_imp_table<- rfImp2%>%
# kable(caption="Random Forest Variable Importance on Simulated Dataset") %>%
#   kable_styling()


# (8.1c)
#hw3_rfmodel3 <- cforest(y ~ ., data=simulated)
#rfdt3 <-as.data.frame(as.matrix(varimp(rf_mod3)))
#hw3_rfdt3 <-tibble::rownames_to_column(as.data.frame(as.matrix(varimp(hw3_rfmodel3))), "VALUE")    #as

#colnames(hw3_rfdt3) <- c("Value","Importance")

#hw3_rfdt3a <- hw3_rfdt3[order(-hw3_rfdt3$Importance),]

#hw3_rfdt3.tbl <- hw3_rfdt3a  %>%
# kable(caption="Unconditional CForest Model: Variable Importance") %>%
#   kable_styling() %>%
#   row_spec()

#hw3_rfdt3b <-  tibble::rownames_to_column(as.data.frame(as.matrix(varimp(hw3_rfmodel3,conditional=T)))
# "VALUE")
#as.data.frame(as.matrix(varimp(hw3_rfmodel3, conditional=T)))

#colnames(hw3_rfdt3b) <- c("Value","Importance")

#hw3_rfdt3ba <- hw3_rfdt3b[order(-hw3_rfdt3b$Importance),]

#hw3_rfdt3b.tbl<-hw3_rfdt3ba  %>%
# kable(caption="Conditional CForest Model: Variable Importance") %>%
#   kable_styling() %>%
#   row_spec()
```

```r
# Now remove correlated predictor
simulated$duplicate1 <- NULL
bagCtrl <- cforest_control(mtry = ncol(simulated) - 1)
baggedTree <- party::cforest(y ~ ., data = simulated, controls = bagCtrl)
cfImp <- party::varimp(baggedTree, conditional = T)
#cfImp <- kable(sort(cfImp, decreasing = TRUE))
cfImp1 <- party::varimp(baggedTree, conditional = F)
#cfImp1 <- kable(sort(cfImp1, decreasing = TRUE))
# Keep correlated predictor
simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
bagCtrl <- cforest_control(mtry = ncol(simulated) - 1)
baggedTree <- party::cforest(y ~ ., data = simulated, controls = bagCtrl)
cfImp2 <- party::varimp(baggedTree, conditional = T)
#cfImp2 <- kable(sort(cfImp2, decreasing = TRUE))
cfImp22 <- party::varimp(baggedTree, conditional = F)
#cfImp22 <- kable(sort(cfImp22, decreasing = TRUE))
simulated$duplicate1 <- NULL

a <- data.frame(features = rownames(rfImp1), RF = rfImp1[,1])
b <- data.frame(features = rownames(rfImp2), RF.cor = rfImp2[,1])
c <- data.frame(features = names(cfImp), CF.cond = cfImp)
d <- data.frame(features = names(cfImp1), CF = cfImp1)
e <- data.frame(features = names(cfImp2), CF.cor.cond = cfImp2)
f <- data.frame(features = names(cfImp22), CF.cor = cfImp22)
aa <- merge(a,d, all=T)
bb <- merge(b,f,all=T)
cc <- merge(c,e,all=T)
dd <-merge(aa,bb,all=T)
hw3final_df <- merge(dd,cc,all=T)
hw3final_df <- rbind(hw3final_df[-3,], hw3final_df[3,])
rownames(hw3final_df) <- c(1:11)

hw3final_dfb<-hw3final_df%>%
  kable(caption="Conditional vs Unconditional CForest Model: Variable Importance") %>%
  kable_styling() %>%
  row_spec()


# (8.1d)

#GBM
gbmModel_nodup <- gbm(y ~ ., data = simulated,
                      distribution = "gaussian",
                      n.trees=1000)

#gbmModel_nodupb <-caret::varImp(gbmModel_nodup)    #as.data.frame(as.matrix(varimp(hw3_rfmodel3)))


simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
gbmModel_wdup <- gbm(y ~ ., data = simulated,
                     distribution = "gaussian",
                     n.trees=1000)
```

```
#gbmModel_wdupb <-varImp(gbmModel_wdup)   #as.data.frame(as.matrix(varimp(hw3_rfmodel3)))


simulated$duplicate1 <- NULL

#Cubist
cubistMod_nodup <- cubist(simulated[-11],
                          simulated$y,
                          committees = 100)

cubistMod_nodupb <-varImp(cubistMod_nodup)

cubistMod_nodupb.df <-
  tibble::rownames_to_column(as.data.frame(as.matrix(varImp(cubistMod_nodup))), "VALUE") #as.data.frame

cubistMod_nodupbplot<-ggplot(cubistMod_nodupb.df,
                             aes(x=reorder(VALUE, Overall),y=Overall)) +
  geom_point(color ='darkorange') +
  geom_segment(aes(x=VALUE,
                   xend=VALUE,
                   y=0,
                   yend=Overall,
                   color = 'darkorange')) +
  labs(title="Variable Importance",
       subtitle="Cubist Model without Duplicate",
       x="",
       y="Importance")+coord_flip()+
  theme_bw()+
  theme(legend.position = "none")



simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
cubistMod_wdup <- cubist(simulated[-11],
                         simulated$y,
                         committees = 100)

cubistMod_wdupb <-varImp(cubistMod_wdup)

cubistMod_wdupb.df <-tibble::rownames_to_column(as.data.frame(as.matrix(varImp(cubistMod_wdup))),
"VALUE")    #as.data.frame(as.matrix(varimp(hw3_rfmodel3)))

cubistMod_wdupbplot<-ggplot(cubistMod_wdupb.df,
                            aes(x=reorder(VALUE, Overall),
                                y=Overall)) +
  geom_point(color ='darkorange') +
  geom_segment(aes(x=VALUE,xend=VALUE,y=0,yend=Overall, color = 'darkorange')) +
  labs(title="Variable Importance",
       subtitle="Cubist Model With Duplicate",
       x="",
       y="Importance")+
  coord_flip()+
  theme_bw()+
```

```
  theme(legend.position = "none")



simulated$duplicate1 <- NULL



# (8.2)

random_predictor <- data.frame(V1=sample(1:2, 100, replace=TRUE),
                               V2=sample(1:100, 100, replace=TRUE),
                               V3=sample(1:1000, 100, replace=TRUE),
                               V4=sample(1:5000, 100, replace=TRUE))
sim_df <- random_predictor %>% mutate(y=V1*V2*V3+rnorm(100))
sim_rf <- randomForest(y ~ ., data = sim_df, importance = TRUE, ntree = 1000)
sim_varImp <- varImp(sim_rf, scale=T)

sim_varImp.df <-tibble::rownames_to_column(as.data.frame(as.matrix(varImp(sim_rf))),
                                           "VALUE")
#as.data.frame(as.matrix(varimp(hw3_rfmodel3)))

sim_varImp.plot<-ggplot(cubistMod_wdupb.df, aes(x=reorder(VALUE, Overall),
                                                y=Overall)) +
  geom_point(color ='darkorange') +
  geom_segment(aes(x=VALUE,xend=VALUE,y=0,yend=Overall, color = 'darkorange')) +
  labs(title="Variable Importance",
       subtitle="Simulated Importance",
       x="",
       y="Importance")+ coord_flip()+theme_bw()+theme(legend.position = "none")


# (8.3a)

#No code needed for this problem

# (8.3b)

#No code needed for this problem

# (8.3c)

#No code needed for this problem


# (8.7a)
#GBM
gbmGrid_87 <- expand.grid(interaction.depth = seq(1, 7, by = 2),
                          shrinkage = c(0.01, 0.1),
                          n.trees = seq(100, 1000, by = 50),
                          n.minobsinnode = 10)

gbmTune_87 <- train(Yield~.,
```

```
                    data=chem_train,
                    method = "gbm",
                    verbose = FALSE,
                    tuneGrid = gbmGrid_87)
#gbmTune_87
#plot(gbmTune_87)
#min(gbmTune_87$results$RMSE, na.rm = TRUE)
# 0.01       7                   850     1.254821  0.5470138  0.9589064
gbmPred_87 <- predict(gbmTune_87, newdata = chem_test)
gb_test_87 <- postResample(pred = gbmPred_87, obs = chem_test$Yield)

# Random Forest

rf_87_grid <- expand.grid(mtry= seq(100, 1000, by=50))

rfTune_87 <-  train(Yield~.,
                    data=chem_train,
                    method = 'rf',
                    ntree = 50,
                    tuneGrid = rf_87_grid)
#plot(rfTune_87)
#min(rfTune_87$results$RMSE, na.rm = TRUE)
rfPred_87 <- predict(rfTune_87, newdata = chem_test)
rf_test_87 <- postResample(pred = rfPred_87, obs = chem_test$Yield)
rf_87<-randomForest(Yield~.,
                    data=chem_train,
                    importance = TRUE,
                    ntree = 900)
#  Cubist
set.seed(58677)
cb_grid_87 <- expand.grid(committees = c(25:45),
                          neighbors = c(1, 3, 5 ))
cbTune_87 <- train(Yield~.,
                   data=chem_train,
                   method = "cubist",
                   metric="RMSE",
                   na.action = na.pass,
                   tuneGrid = cb_grid_87,
                   trControl = trainControl(method = 'cv'))
#plot(cbTune_87)
#min(cbTune_87$results$RMSE, na.rm = TRUE)
cbPred_87 <- predict(cbTune_87, newdata = chem_test)
cb_test_87 <- postResample(pred = cbPred_87, obs = chem_test$Yield)


hw2.3.dperformance_table <- rbind("GBM"=c("RMSE"=max(gbmTune_87$results$RMSE),
                                          "RSquared"=max(gbmTune_87$results$Rsquared),
                                          "MAE"=max(gbmTune_87$results$MAE)),
                                  "GBMTest"=gb_test_87,

                                  "RFTrain"=c("RMSE"=max(rfTune_87$results$RMSE),
                                              "RSquared"=max(rfTune_87$results$Rsquared),
                                              "MAE"=max(rfTune_87$results$MAE)),
```

```
                                  "RFTest"=rf_test_87,

                                  "CubistTrain"=c(max(cbTune_87$results$RMSE),
                                                  max(cbTune_87$results$Rsquared),
                                                  max(cbTune_87$results$MAE)),
                                  "CubistTest"=cb_test_87) %>% kable(caption="Tree Model Performance on
  kable_styling() %>%
  row_spec() %>% row_spec(row=5:6, background ="#d9f2e6")



# (8.7b)

#Boosted Model
import <-varImp(gbmTune_87)
import <- as.data.frame(import$importance) %>%
  rownames_to_column("Variable") %>%
  filter(Overall>0)%>%arrange(Overall)
boost<- ggplot(import[1:10,],
               aes(x=reorder(Variable, Overall),
                   y=Overall)) +
  geom_point(color ='#b33a3a') +
  geom_segment(aes(x=Variable,
                   xend=Variable,
                   y=0,
                   yend=Overall),
               color = '#b33a3a') +
  labs(title="Variable Importance Chemical Manufacturing",
       subtitle="Gradient Boosted Trees",
       x="",
       y="Importance")+coord_flip()+theme_bw()+theme()
#rf Model
import <- varImp(rf_87, scale = FALSE)
import <- as.data.frame(import) %>%
  rownames_to_column("Variable") %>%
  filter(Overall>0)%>%arrange(Overall)
random <- ggplot(import[1:10,],
                 aes(x=reorder(Variable, Overall),
                     y=Overall)) +
  geom_point(color ='#3ab3b3') +
  geom_segment(aes(x=Variable,
                   xend=Variable,
                   y=0,
                   yend=Overall),
               color = '#3ab3b3') +
  labs(title="Variable Importance Chemical Manufacturing",
       subtitle="Random Forest",
       x="",
       y="Importance")+coord_flip()+theme_bw()+theme()
# Cubist Model
import <-varImp(cbTune_87)
import <- as.data.frame(import$importance) %>%
  rownames_to_column("Variable") %>%
```

```
  filter(Overall>0)%>%arrange(Overall)
cube<- ggplot(import[1:10,],
              aes(x=reorder(Variable, Overall),
                  y=Overall)) +
  geom_point(color ='#77b33a') +
  geom_segment(aes(x=Variable,
                   xend=Variable,
                   y=0,
                   yend=Overall),
               color = '#77b33a') +
  labs(title="Variable Importance Chemical Manufacturing",
       subtitle="Cubist Trees",
       x="",
       y="Importance")+coord_flip()+theme_bw()+theme()


# (8.7c)

rpartTune <- train(Yield~.,
                   data=chem_train,
                   method = "rpart2",
                   tuneLength = 10,
                   trControl = trainControl(method = "cv"))
#plot(rpartTune)
best_rpart <- rpart(Yield~., data =chem_train,
                    control = rpart.control(maxdepth = 4))
#decision_plot <- rpart.plot(best_rpart,
# type = 1,
# extra = 1)


## Libraries for .rmd file

# Predicitve Modeling
libraries('AppliedPredictiveModeling',
          'mice',
          'caret',
          'tidyverse',
          'impute',
          'pls',
          'caTools',
          'mlbench',
          'randomForest',
          'party',
          'gbm',
          'Cubist',
          'rpart')

# Formatting Libraries
libraries('default',
          'knitr',
          'kableExtra',
          'gridExtra',
```

```
          'sqldf',
          'tibble')

# Plotting Libraries
libraries('ggplot2',
          'grid',
          'ggfortify',
          'rpart.plot')

# Data Wrangling
library(AppliedPredictiveModeling);
library(mice);
library(caret);
library(tidyverse);
library(pls);
library(caTools);
library(mlbench);
library(stringr)

# Formatting
library(default);
library(knitr);
library(kableExtra);

# Plotting
library(ggplot2);
library(grid);
library(ggfortify);
library(gridExtra)


# Assignment 1
## kj-6.3a
data("ChemicalManufacturingProcess")

## kj-6.3a-plot
Plt_CMP.Yield

## kj-6.3b
CMP.total_na %>% kable(caption="Variables with Missing Values",
                       booktabs=T) %>%
  kable_styling(full_width = F,
                position = "center") %>%
  column_spec(3, bold = T,
              border_left = F,
              border_right = F,
              width = '0.1cm') %>%
  row_spec()

## kj-6.3c
CMP.pls.train.perf %>%
  kable(caption="PLS Performance Metrics on Training Subset",
        booktabs=T) %>%
```

```
  kable_styling() %>%
  row_spec()

## kj-6.3c2
grid.arrange(Plt_CMP.RMSE, Plt_CMP.fit.obs_vs_pred, ncol=2)

## kj-6.3d-1
CMP.pls.test.perf %>%
  kable(caption="PLS Performance Metrics on Test Subset",
        booktabs=T) %>%
  kable_styling() %>%
  row_spec()

## kj-6.3d-2

## kj-6.3e
Plt_CMP.pls.imp

## kj-6.3f-1
Plt_CMP.Scatter

## kj-6.3f-2
CMP_DF.corr.tbl %>%
  kable(caption="Variable Correlation with Yield",
        booktabs=T) %>%
  kable_styling(full_width = F) %>%
  row_spec()


# Assignment 2
## kj-7.2-ex1
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)

## We convert the 'x' data from a matrix to a data frame
## One reason is that this will give the  columns names.
trainingData$x <- data.frame(trainingData$x)
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)

## kj-7.2-ex3, echo=F
Plt_Sim.featurePlot

## kj-7.2-ex4, echo=T, eval=F
knnModel <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10)
knnModel
knnPred <- predict(knnModel, newdata = testData$x)
postResample(pred = knnPred, obs = testData$y)
```

```
###Model 1-MARS Regression:
## kj-7.2-1
hw2mars_modplot

###Model 2 SVM:
## kj-7.2-2
hw2svm_modplot

###Model 3 NNET:
## kj-7.2-3
hw2nnet_modplot

## kj-7.2-4
hw2.2.bperformance_table
#%>% kable(caption="Model Performance", digits=4)
#%>% kable_styling()
#%>% row_spec() %>% row_spec(row=3:4,background ="#d9f2e6")

## kj-7.2-4b
hwmarsimp_plot

## kj-7.5a
hw2.2.cperformance_table

## kj-7.5b
hwsvmimp_plot2

## kj-7.5c
hw2cor_df2


# Assignment 3
## kj-8.1
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"

## kj-8.1a
hw3model1 <- randomForest(y ~ ., data = simulated,
                          importance = TRUE,
                          ntree = 1000)
rfImp1 <- varImp(model1, scale = FALSE)
rfImp1plot

## kj-8.1b-ex
simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1

## kj-8.1b
#code
rfImp2plot
```

```
## kj-8.1c
#code
hw3final_dfb

## kj-8.1d
summary(gbmModel_nodup)

## kj-8.1da
summary(gbmModel_wdup)

## kj-8.1db
cubistMod_nodupbplot;cubistMod_wdupbplot

## kj-8.2
sim_varImp.plot

## kj-8.7a
hw2.3.dperformance_table

## kj-8.7b
grid.arrange(boost, random, cube, ncol=3)

##kj-8.7c
rpart.plot(best_rpart,
           type = 1,
           extra = 1)
```