Team 2 - Homework Two

Assignment 3: KJ 8.1-8.3; KJ 8.7

Juliann McEachern

11/8/19

Dependencies

```
# Forecast libraries
libraries("mlbench", "AppliedPredictiveModeling", "party", "randomForest",
    "caret")

# Formatting Libraries
libraries("default", "knitr", "kableExtra", "tidyverse")

# Plotting Libraries
libraries("ggplot2", "grid", "ggfortify")
```

(1) Kuhn & Johnson 8.1

Recreate the simulated data from Exercise 7.2:

```
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)?

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

The random forest model predominantly favored the predictor variables V1-V5, with the exception of V3. The uninformative predictors (V6-10) all had significantly lower overall importance.

Table 1: Variable Importance for Random Forest Model1

	V1	V4	V2	V5	V3	V6	V7	V10	V9	V8
Overall	8.73	7.62	6.42	2.02	0.76	0.17	-0.01	-0.07	-0.1	-0.17

(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:

```r
simulated$duplicate1 <- simulated$V1 + rnorm(200) * 0.1
model2 <- randomForest(y ~ ., data = simulated, importance = T,
    ntree = 1000)
rfImp2 <- varImp(model2, scale = F)
```

The correlation of the simulated V1 and duplicate variable was 0.9460206. Adding a highly correlated value, such as this duplicate, decreased the overall variable importance of V1.

Table 2: Variable Importance for Random Forest Model2

	V4	V2	V1	duplicate1	V5	V3	V6	V10	V9	V7	V8
Overall	7.05	6.07	5.69	4.28	1.87	0.63	0.14	0.03	0.01	-0.01	-0.04

(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?

The `cforest` approach shows the same pattern as the traditional random forest model. Both models have similar overall importance for V4,V1, and V2, however the importance of variable V5 decreases slightly with the `cforest` approach. Like `rforest`, the `cforest` model also decreases the overall importance of V1 when the duplicate variable is evaluated in the model. The overall importance decreases more with the `cforest` method.

Table 3: Variable Importance for Conditional Inference Trees Models

	V4	V2	duplicate1	V1	V5	V3	V7	V6	V10	V9	V8
Original	6.66	5.19	NA	5.50	1.21	0.03	-0.01	-0.01	-0.01	0.00	0.00
Duplicate	6.05	4.82	1.92	1.92	1.06	0.02	0.01	0.00	-0.01	-0.01	-0.02

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

We repeated this process a final time using a gradient boosted model. We used a simple model with 1000 trees The boosted trees approach also picked V1-V5 to be the variables with the most influence. Again, we see V5 and V3 have lower importance than V1, V2, and V4. The duplicate value in the `gbm` model also responds similarly to our previous approaches. The importance of V1 and the duplicate variable both decrease. Unlike the other models, however, the `gbm` model only slightly lowers the importance of V1 and the duplicate only has half the level of importance, compared to the original.

Table 4: Variable Relative Influence of Gradient Boosted Models

	V4	V2	V1	V3	V5	duplicate1	V7	V6	V10	V9	V8
Original	25.10	19.60	23.43	9.10	11.22	NA	2.94	2.53	1.59	2.4	2.09
Duplicate	24.94	19.43	16.85	10.28	10.10	8.16	3.16	2.18	1.96	1.5	1.46

(2) Kuhn & Johnson 8.2

Use a simulation to show tree bias with different granularities.

Basic regression trees split predictor variables into small groupings based on the response variable. According to the text, "predictors with a higher number of distinct values are favored over more granular predictors." The simulation below helps us visualize this tree bias among different granularities.

In our simulation:

- V1 contains 100 random samples of 2 numbers
- V2 contains 100 random samples of 100 numbers
- V3 contains 100 random samples of 1000 numbers
- V4 contains 100 random samples of 5000 numbers

Our response variable, y, was derived using the formula: $y = V1 * V2 * V3 + e$, where e is a random error.

Table 5: Head of Simulation Table

V1	V2	V3	V4	y
1	13	946	493	12298.61
2	47	151	1164	14195.20
1	17	749	2128	12732.55
2	55	344	3788	37840.56
2	59	645	2281	76109.04
1	87	477	3513	41498.56

Despite V1-V3 having equal weight in our model, V3 had the highest overall variable importance. This helps illustrate the bias in trees as V1 and V2 were both of much smaller granularity.

	Overall
V2	32.829698
V3	30.738404
V1	25.108499
V4	4.484885

(3) Kuhn & 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:

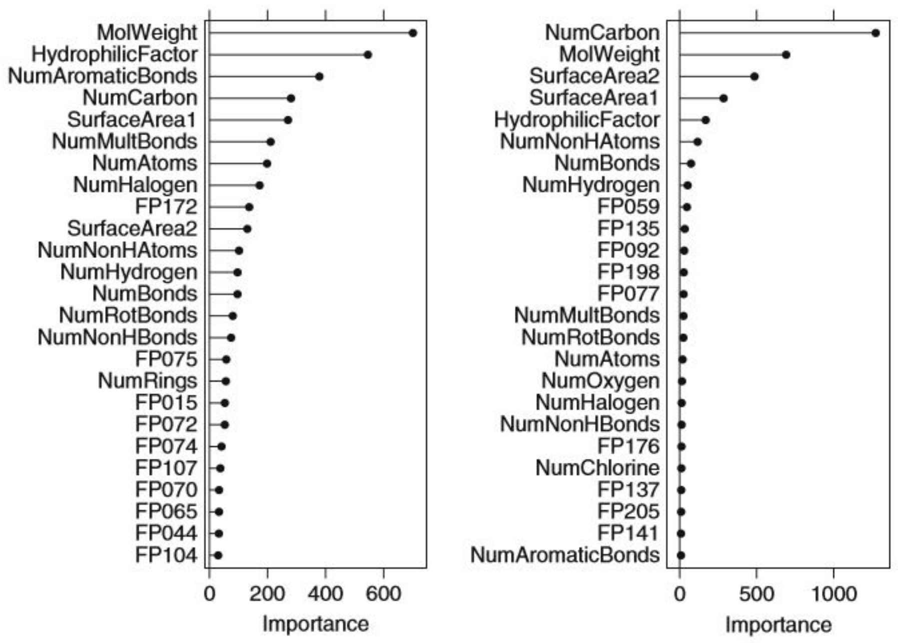


Fig. 8.24: A comparison of variable importance magnitudes for differing values of the bagging fraction and shrinkage parameters. Both tuning parameters are set to 0.1 in the *left* figure. Both are set to 0.9 in the *right* figure

(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?

Stochastic gradient boosting distributes data differently than other classification trees. Boosted trees contribute to the model unequally and have a distribution structure dependent on past trees and tree depth. The dependency aspect of boosted trees creates a much steeper cut off in variable importance than other tree models. From the text, we know that the learning rate is the fraction of the current predicted value added to the previous iteration's predicted value. And, bagging reduces the variance of a prediction. Thus, the bagging and shrinking parameters would greatly affect the influence of boosted trees on the predictor variable importance.

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

The higher bagging and learning rate tuning parameter of 0.9 could make the prediction model more stable and more accurate. We would have to run tests to ensure that the increased value did not over-fit the data to our model.

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

Increased interaction depth would increase the spread of variable importance as each tree would have more features to be considered during the splitting process.

(4) Kuhn & 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?

Resampling & Test Performance

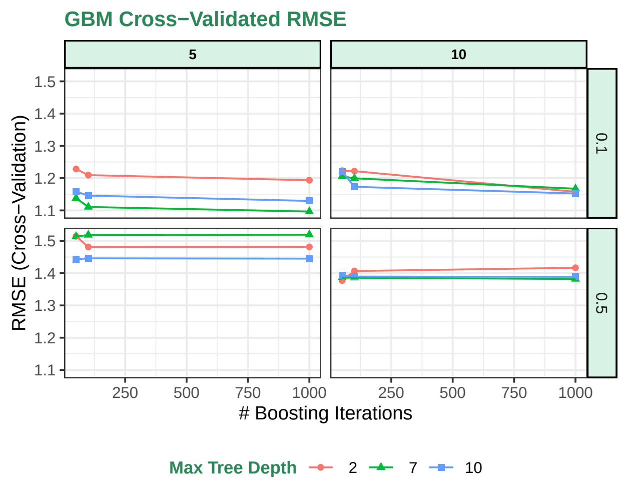
For this question, we evaluated random forest, gradient boosted, and cubist models on our chemical manufacturing process data. We found that the cubist model gave the best results, however the gradient boosted performed similiarly.

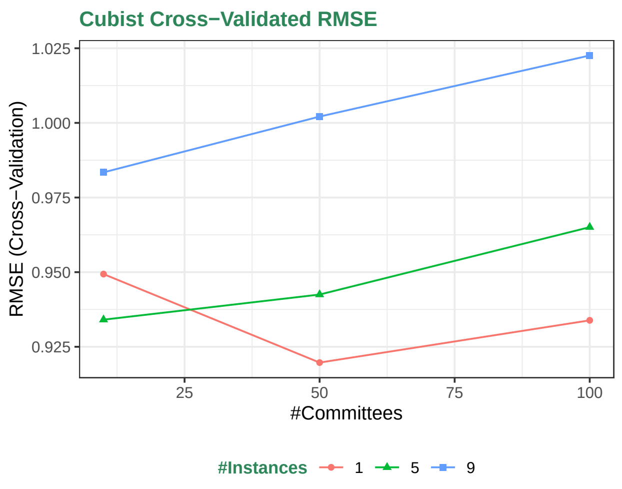
Table 6: Model Performance

	RMSE	RSquared	MAE
rfTrain	1.1225	0.6288	0.8835
rfTest	1.1553	0.8461	0.8699
gbmTrain	1.0964	0.6171	0.8875
gbmTest	1.0916	0.7947	0.8108
cbTrain	**0.9197**	**0.7503**	**0.6608**
cbTest	**1.0916**	**0.7947**	**0.8108**

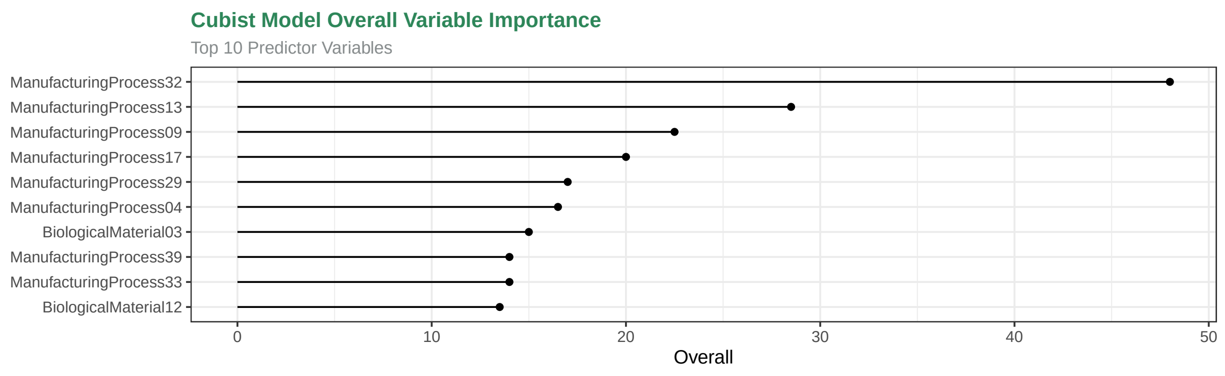
RMSE Plot:

We compared the RMSE metric between our cubist and gradient boosted below and determined the cubist was our optimal model.





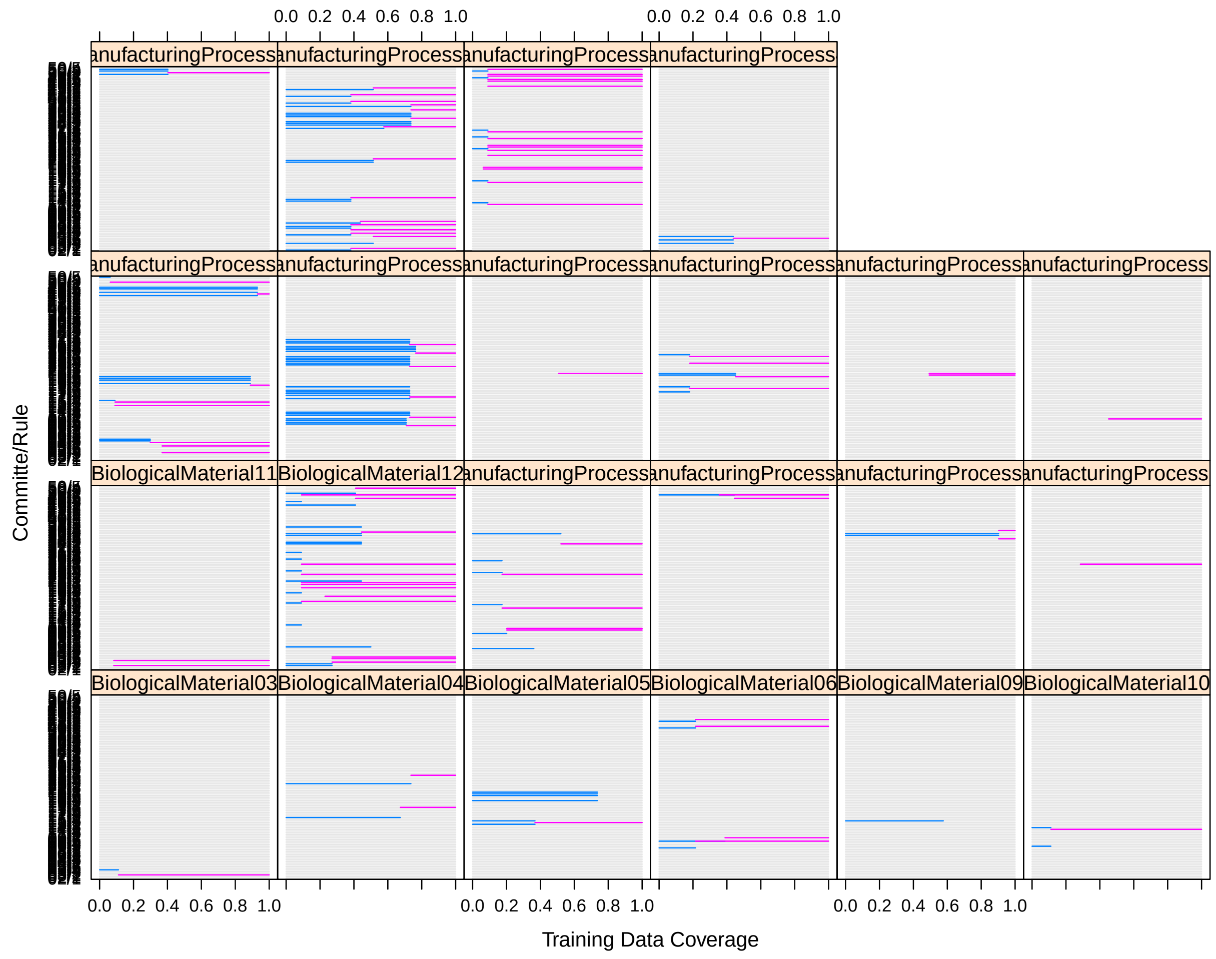
(b). Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?



Manufacturing processes dominate the top 10 list of important predictors for the cubist model. The high presence of manufacturing is inline with our non-linear models, whereas our optimal linear model favored biological materials.

(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?

I was unable to find a package which plotted the terminal nodes of the cubist package. The final model summary, however, provides the rules used for the distriubution of yield in the terminal nodes. The cubist model also allows you to view the splits as a dotplot shown below. For each split, a panel is created for each variable to show how the committee/rule was applied. I have researched the cubist method and can not find a way (ie party package) to nicely plot / view what this question is asking. I will leave this answer for now and I am hoping to work as a team to find a better solution :)



R Code

```
# Libraries
library(mlbench)
library(AppliedPredictiveModeling)
library(party)
library(randomForest)
library(caret)
library(tidyverse)
library(gbm)
library(mice)
library(recipes)
library(Cubist)

# Set Seed
set.seed(200)

# (8.1)
simulated <- mlbench.friedman1(200, sd = 1)
simulated <- cbind(simulated$x, simulated$y)
```

```r
simulated <- as.data.frame(simulated)
colnames(simulated)[ncol(simulated)] <- "y"

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

# (8.1b)
simulated$duplicate1 <- simulated$V1 + rnorm(200) * 0.1
model2 <- randomForest(y ~ ., data = simulated, importance = T,
    ntree = 1000)
rfImp2 <- varImp(model2, scale = F)

# (8.1c) Rename variables for ease of reference
sim_original <- select(simulated, -duplicate1)
sim_duplicate <- simulated

model3 <- cforest(y ~ ., data = sim_original, controls = cforest_unbiased(ntree = 1000))
cfImp3 <- varimp(model3, conditional = T)
cfImp3df <- as.data.frame(cfImp3) %>% rownames_to_column("Variable") %>%
    rename(Overall = "cfImp3")
model4 <- cforest(y ~ ., data = sim_duplicate, controls = cforest_unbiased(ntree = 1000))
cfImp4 <- varimp(model4, conditional = T)
cfImp4df <- as.data.frame(cfImp4) %>% rownames_to_column("Variable") %>%
    rename(Overall = "cfImp4")
cfImpTbl <- right_join(cfImp3df, cfImp4df, by = "Variable") %>%
    column_to_rownames("Variable") %>% rename(Original = Overall.x,
    Duplicate = Overall.y)

# (8.1d)
model5 <- gbm(y ~ ., data = sim_original, distribution = "gaussian",
    n.trees = 1000, cv.folds = 5)
gbmImp5 <- summary(model5, plot = F) %>% mutate(var = as.character(var))
gbmImp5df <- as.data.frame(gbmImp5) %>% arrange(desc(rel.inf))
model6 <- gbm(y ~ ., data = sim_duplicate, distribution = "gaussian",
    n.trees = 1000, cv.folds = 5)
gbmImp6 <- summary(model6, plot = F) %>% mutate(var = as.character(var))
gbmImp6df <- as.data.frame(gbmImp6) %>% arrange(desc(rel.inf))
gbmImpTbl <- right_join(gbmImp5df, gbmImp6df, by = "var") %>%
    column_to_rownames("var") %>% rename(Original = rel.inf.x,
    Duplicate = rel.inf.y)

# (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)

# (8.7)
# FOR FINAL HW SUBMISSION, DO NOT REPEAT STEPS: JUST CALL
# VARIABLES FROM PRIOR ASSIGNMENT.

data(ChemicalManufacturingProcess)
CMP_Impute <- mice(ChemicalManufacturingProcess, m = 5, printFlag = F)
CMP_DF <- mice::complete(CMP_Impute, 2)

# Create Partition for Train/Test Splits
trainingRows <- createDataPartition(CMP_DF$Yield, p = 0.8, list = FALSE)

# Split Train/Test Data
train <- CMP_DF[trainingRows, ]
test <- CMP_DF[-trainingRows, ]

# Pre-Process Recipe
rec <- recipes::recipe(CMP_DF, Yield ~ .)
rec <- rec %>% step_nzv(all_predictors(), options = list(freq_cut = 95/5,
    unique_cut = 10))
prep_rec = prep(rec, training = CMP_DF)
CMP_DF_TF = bake(prep_rec, CMP_DF)

# Create Partition for Train/Test Splits
trainingRows <- createDataPartition(CMP_DF_TF$Yield, p = 0.8,
    list = FALSE)

# Split Train/Test Data
train <- CMP_DF_TF[trainingRows, ]
test <- CMP_DF_TF[-trainingRows, ]

# (8.7a) Controls
tr_control <- trainControl(method = "cv", number = 10, repeats = 10)
# Random Foreset
rfModel <- train(Yield ~ ., data = train, method = "rf", importance = T,
    trControl = tr_control, tuneLength = 5)
rfTrainMetric <- data.frame(RMSE = rfModel$results$RMSE, RSquared = rfModel$results$Rsquared,
    MAE = rfModel$results$MAE) %>% filter(RMSE == min(RMSE))
rfPred <- predict(rfModel, newdata = test)
rfPerf <- postResample(pred = rfPred, obs = test$Yield)
rfPlot <- ggplot(rfModel) + labs(title = "Random Forest Model Cross-Validated RMSE Profile") +
    theme(legend.position = "bottom")

# Gradient Boosted
gbmGrid <- expand.grid(n.trees = c(50, 100, 1000), interaction.depth = c(2,
    7, 10), shrinkage = c(0.1, 0.5), n.minobsinnode = c(5, 10))
gbmModel <- train(Yield ~ ., data = train, method = "gbm", trControl = tr_control,
    tuneGrid = gbmGrid, tuneLength = 5)
gbmTrainMetric <- data.frame(RMSE = gbmModel$results$RMSE, RSquared = gbmModel$results$Rsquared,
    MAE = gbmModel$results$MAE) %>% filter(RMSE == min(RMSE))
gbmPred <- predict(gbmModel, newdata = test)
gbmPerf <- postResample(pred = gbmPred, obs = test$Yield)
gbmPlot <- ggplot(gbmModel) + labs(title = "GBM Cross-Validated RMSE") +
    theme(legend.position = "bottom")
```

```
# Cubist
cbGrid <- expand.grid(committees = c(10, 50, 100), neighbors = c(1,
    5, 9))
cbModel <- train(Yield ~ ., data = train, method = "cubist",
    trControl = tr_control, tuneGrid = cbGrid, tuneLength = 5)
cbTrainMetric <- data.frame(RMSE = cbModel$results$RMSE, RSquared = cbModel$results$Rsquared,
    MAE = cbModel$results$MAE) %>% filter(RMSE == min(RMSE))
cbPred <- predict(gbmModel, newdata = test)
cbPerf <- postResample(pred = gbmPred, obs = test$Yield)
cbPlot <- ggplot(cbModel) + labs(title = "Cubist Cross-Validated RMSE") +
    theme(legend.position = "bottom")

# Metrics Table
performance <- rbind(rfTrain = rfTrainMetric, rfTest = rfPerf,
    gbmTrain = gbmTrainMetric, gbmTest = gbmPerf, cbTrain = cbTrainMetric,
    cbTest = cbPerf)

# (8.7b)
cbImp <- varImp(cbModel, scale = F)
cbImpPlot <- cbImp$importance %>% as.data.frame() %>% rownames_to_column("Variable") %>%
    top_n(10, Overall) %>% ggplot(aes(x = reorder(Variable, Overall),
    y = Overall)) + geom_point() + geom_segment(aes(x = Variable,
    xend = Variable, y = 0, yend = Overall)) + coord_flip() +
    labs(y = "Overall", x = "", title = "Cubist Model Overall Variable Importance",
        subtitle = "Top 10 Predictor Variables")

# (8.7c)
cbDotPlot <- dotplot(cbModel$finalModel)
```