Team 2 - Homework Two

Assignment 3: KJ 8.1-8.3; KJ 8.7

Vinicio Haro

DATE

Dependencies

```
# Forecast libraries
libraries("mlbench", "AppliedPredictiveModeling", "party",
    "gbm", "Cubist")
# Regression libraries
libraries("randomForest", "caret", "rpart", "impute",
    "partykit")
# Formatting Libraries
libraries("default", "knitr", "kableExtra", "dplyr",
    "sqldf")
# 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)?

The code to the RF model has been provided to us through the literature.

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

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

Lets 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 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.

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

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

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

Table 1: Random Forest Variable Importance

	Overall
V1	8.7322354
V2	6.4153694
V3	0.7635918
V4	7.6151188
V5	2.0235246
V6	0.1651112
V7	-0.0059617
V8	-0.1663626
V9	-0.0952927
V10	-0.0749448

Did the random forest model significantly use the uninformative predictors (V6 – V10)? The most important feature was V1 with an importance of over 8 followed by V2. Variables V6 through V10 are not as significant as v1-V5.

> (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:

Table 2: Random Forest Variable Importance With New Variable

	Overall
V1	8.6895807
V2	6.4296506
V3	0.7471133
V4	7.6869946
V5	2.3690477
V6	0.1099727
V7	0.0308325
V8	-0.1077942
V9	-0.1281411
V10	0.0489594

By adding a highly correlated variable, we see that V1 importance drops by roughly 2 measures.

> (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 3: Unconditional CForest Model: Variable Importance

	x
V1	7.9163421
V2	6.1631759
V3	0.2768109
V4	7.5306237
V5	1.9909976
V6	0.0642007
V7	0.1624573
V8	-0.0686124
V9	-0.1156031
V10	-0.0358464

Table 4: Conditional CForest Model: Variable Importance

	x
V1	6.2335814
V2	5.3910906
V3	0.1410907
V4	6.2496285
V5	1.4071458
V6	-0.1874603
V7	-0.0349666
V8	-0.3524608
V9	-0.3676360
V10	-0.3055672

We see that adding a conditional or non conditional clause for cForest does not have an effect on the importance of variables V1-V6. We some changes in variable importance closer to the upper bound. What is going on in the background? When we set conditional to true,we compute feature importance by creating permutations built on covariates associated with features of interest. The clause has a backend threshold where it only grabs features with a compliment of the p value greater than the threshold (determined on the backend). This generated score is analagous to beta coefficients for regression models.

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

Let's try a gradient boosting tree. Why does this tree matter? It is computationaly fast. On a high level, the GB tree starts by taking the average across the target label. We then compute residuals where the residual is simply the difference between the actual value and the predicted value. A decision tree is built on these residuals where each leaf node will contain some prediction along with the residual value. There are some cases where there are more residuals than leaves but on the backend, the GB tree function will compute the average residuals in these cases. We then of course predict new target values and compute new residuals. This process can be repeated until convergece or a tree limit is defined.

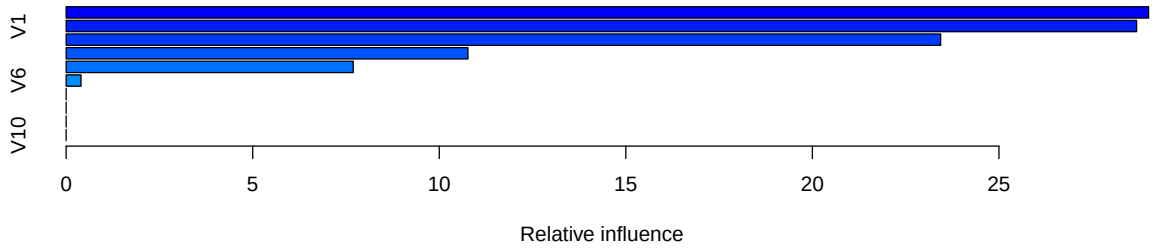


Table 5: Gradient Boosting Tree

	var	rel.inf
V4	V4	29.0111902
V1	V1	28.6900508
V2	V2	23.4378987
V5	V5	10.7687292
V3	V3	7.6935092
V6	V6	0.3986219
V7	V7	0.0000000
V8	V8	0.0000000
V9	V9	0.0000000
V10	V10	0.0000000

The summary returns the variable name along with a measure of influence on the target variable. From our GBM tree, we can see v4 is the most influential. V1 and correlated duplicate1 are also much more influential. V6-V10 does not break the top half of our list of influential variables.

The next model we want to try is the cubist model. The cubist model is a rather unique variation on trees. Each leaf in the tree 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 splis. It should be noted that each intermediate step between leafs also contain a linear model. Predictions are made via linear models on the on the terminal node.

Table 6: Cubist Model Regression Variable Importance

	Overall
y	50
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0

V2 is the most important variable closely followed by V1. We still do not see variables V6-V10 increasing in

importance.

(2) Kuhn & Johnson 8.2

Use a simulation to show tree bias with different granularities. We need to understand the purpose of this problem. The literature covers some key elements when it comes to tree bias based off certain conditions. We know that if we have a variable with many distinct values, then it is said that this variable has a low variance. These types of variables are magnetic to the response variable vs other predictors that have a higher variance. This greatly affects the splits in any tree system and in some cases, splits are done on noise. Typically we try to avoid cases where noise takes the role of the root node.

Let's create a dataset where our predictors will vary in terms of unique values.We then run rpart.

Table 7: Tree Bias

	Overall
X1	0.4226645
X2	0.8863903

(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:

(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? There are some mechanics that occur on the right hand model that we should point out. First, we see that there is a relationship between the learning rate and how "greedy" the model gets. Recall that a greedy model tries to find the optimal point at each iteration. The relationship states that as the learning rate approaches one, the overall "greed" of the model increases. As a consequence of increased "greed" , the model tends to no longer identify as many variables related to the label variable. We should also address the bagging fraction. When the bagging fraction increases, we tend to see more usage of data in the model. An analogy is if you consider data to be the fuel of the model (engine),then more of it is consumed as the bagging fraction increases.This also reduces the number of predictors considered to be important.

(b). Which model do you think would be more predictive of other samples? We need to consider how altering the parameter values will affect our models. If we were to increase the values in the parameters, we should expect to see model performance decrease, therefore it is safe to say that the left hand model will be more predictive of our other samples.

(c). How would increasing interaction depth affect the slope of predictor importance for either model in Fig.8.24? The literature states that there is a relationship between interaction depth and feature importance. As we see tree depth increase, we expect a more uniform spread of variable importance across all variables. Visually, this makes the lines bigger in the importance figures shown in 8.2.4.

(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?

Build Models We will build a standard model for CART, Random Forest, GBM, and Cubist. We already looked at the background of some of the models featured here.

> (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?

Table 8: R Squared Comparisons

r_squared	model
0.36	CART
0.55	RF
0.56	GBM
0.64	Cubist

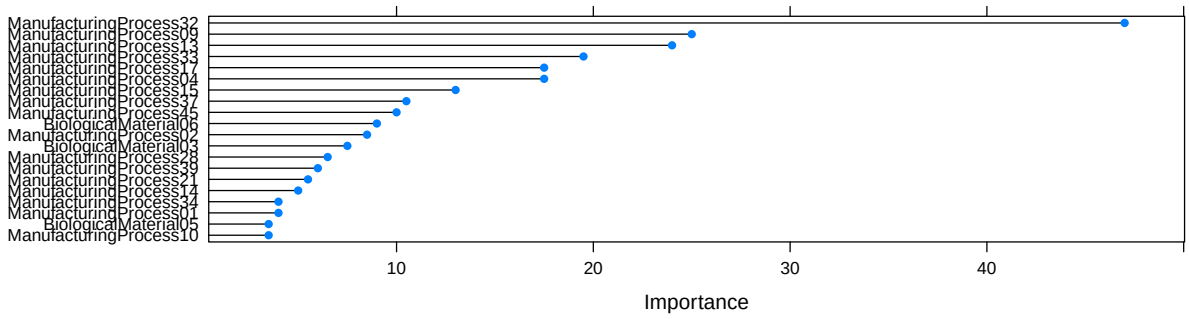
Our initial performance inspection should start by checking the variability captured in the data. We generate the r squared value for each model and can observe that the cubist model is the highest performing with 70 percent of the data variability captured in the data. CART captures the least amount of data variability out of our four models.

Table 9: R Squared Comparisons (Test)

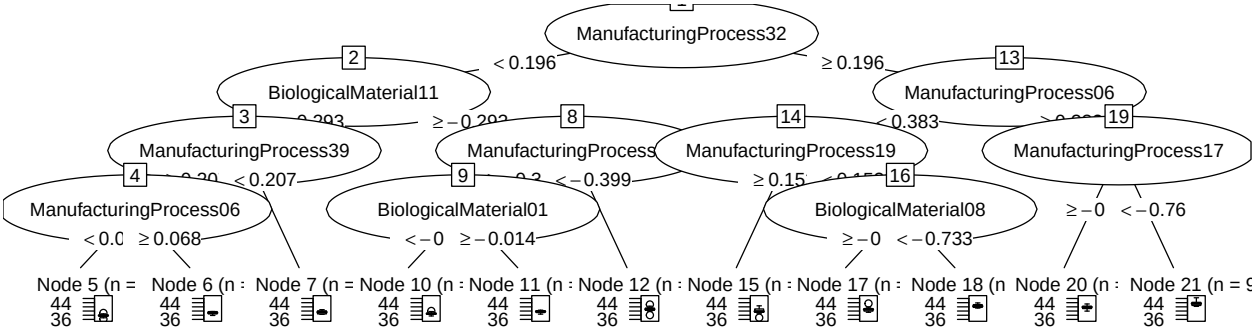
model	RMSE	Rsquared	MAE
CART	1.3775897	0.5074503	0.9737224
RF	1.0167800	0.7770856	0.7270947
GBM	0.9902281	0.7272257	0.7668699
Cubist	0.8313616	0.8165322	0.5784957

We also examine model performance on the test data. We want to see the predictive performance of each model. Based on performance agaist the test data, Cubist model still has the highest RSquared in addition to the lowest RMSE and MAE. I believe ths cubist model outperforms any of the other models from chapter 6 and 7.

Now we test variable importance for our optimal model.



(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?



CART is our most optimal single tree. We can use partykit to visualuze our tree. Manufacturing Process 32 becomes our root node. Accordig to our tree, when MP 32 is greater, this results in an increased yield.

Manufacturing processes 32 and 13 are at the top, with higher values of process 32 being associated with larger yields. Lower values of process 32 are associated with smaller yields. However, a lower values of process 32 may be counter-acted with a corresponding lower value of process 13.

R Code

```
# insert code here (8.1a)
rf_mod <- randomForest(y ~ ., data = simulated, importance = TRUE,
    ntree = 1000)
rf_imp <- varImp(rf_mod, scale = FALSE)

dt <- as.matrix(rf_imp)

# (8.1b) code
rf_mod2 <- randomForest(y ~ ., data = simulated, importance = TRUE,
    ntree = 1000)
rf_imp2 <- varImp(rf_mod2, scale = FALSE)

dt <- as.matrix(rf_imp2)

dt <- as.matrix(rf_imp2)


# (8.1c) code
library(party)

rf_mod3 <- cforest(y ~ ., data = simulated)

# (8.1d) code
gbm_mod <- gbm(y ~ ., data = simulated, distribution = "gaussian")

# (8.2) code
set.seed(100)
X1 <- rep(1:2, each = 100)
X2 <- rnorm(200, mean = 0, sd = 2)
```

```r
# X3 <- sample(0:100 / 100, 200, replace = T)
Y <- X1 + rnorm(200, mean = 0, sd = 4)
sim <- data.frame(Y = Y, X1 = X1, X2 = X2)

rpart_mod <- rpart(Y ~ ., data = sim)

dt4 <- as.data.frame(as.matrix(varImp(rpart_mod)))

# (8.3a) (8.3b) (8.3c)

# (8.7a) code
data(ChemicalManufacturingProcess)
cd <- ChemicalManufacturingProcess



predictors <- subset(cd, select = -Yield)
yield <- subset(cd, select = "Yield")

set.seed(517)
trainingRows <- createDataPartition(yield$Yield, p = 0.8,
    list = FALSE)

x_train <- predictors[trainingRows, ]
y_train <- yield[trainingRows, ]

x_test <- predictors[-trainingRows, ]
y_test <- yield[-trainingRows, ]

# Pre-process trainPredictors and apply to
# trainPredictors and testPredictors
pp <- preProcess(x_train, method = c("BoxCox", "center",
    "scale", "knnImpute"))
transform_x_train <- predict(pp, x_train)
transform_x_test <- predict(pp, x_test)

# Identify and remove NZV
nz_predictors <- nearZeroVar(transform_x_train)
transform_x_train <- transform_x_train[-nz_predictors]
transform_x_test <- transform_x_test[-nz_predictors]

# Identify and remove highly correlated predictors
pred_corr = cor(transform_x_train)
high_corr <- findCorrelation(pred_corr)
transform_x_train <- transform_x_train[, -high_corr]
transform_x_test <- transform_x_test[, -high_corr]

# Set-up trainControl boot method (CV takes way too
# long)
set.seed(517)
tune_param <- trainControl(method = "boot", number = 25)

# CART
```

```r
set.seed(614)
rpart_param <- expand.grid(maxdepth = seq(1, 10, by = 1))
rpart_tune <- train(x = transform_x_train, y = y_train,
    method = "rpart2", metric = "Rsquared", tuneGrid = rpartGrid,
    trControl = tune_param)

# RF
set.seed(614)
rf_param <- expand.grid(mtry = seq(2, 38, by = 3))
rf_tune <- train(x = transform_x_train, y = y_train,
    method = "rf", tuneGrid = rf_param, metric = "Rsquared",
    importance = TRUE, trControl = tune_param)

# GBM
set.seed(614)
gbm_param <- expand.grid(interaction.depth = seq(1,
    6, by = 1), n.trees = c(25, 50, 100, 200), shrinkage = c(0.01,
    0.05, 0.1, 0.2), n.minobsinnode = c(5, 10, 15))
gbm_tune <- train(x = transform_x_train, y = y_train,
    method = "gbm", metric = "Rsquared", verbose = FALSE,
    tuneGrid = gbm_param, trControl = tune_param)

# cubist
set.seed(614)
cubist_param <- expand.grid(committees = c(1, 5, 10,
    20, 50, 100), neighbors = c(0, 1, 3, 5, 7))
cubist_tune <- train(x = transform_x_train, y = y_train,
    method = "cubist", verbose = FALSE, metric = "Rsquared",
    tuneGrid = cubist_param, trControl = tune_param)

# (8.7b) code
rpart_metric <- as.data.frame(round(rpart_tune$results$Rsquared[best(rpart_tune$results,
    "Rsquared", maximize = TRUE)], 2))
colnames(rpart_metric) <- "r_squared"
row.names(rpart_metric) <- "CART"

rf_metric <- as.data.frame(round(rf_tune$results$Rsquared[best(rf_tune$results,
    "Rsquared", maximize = TRUE)], 2))
colnames(rf_metric) <- "r_squared"
row.names(rf_metric) <- "RF"

gbm_metric <- as.data.frame(round(gbm_tune$results$Rsquared[best(gbm_tune$results,
    "Rsquared", maximize = TRUE)], 2))
colnames(gbm_metric) <- "r_squared"
row.names(gbm_metric) <- "GBM"

cubist_metric <- as.data.frame(round(cubist_tune$results$Rsquared[best(cubist_tune$results,
    "Rsquared", maximize = TRUE)], 2))
colnames(cubist_metric) <- "r_squared"
row.names(cubist_metric) <- "Cubist"

library(sqldf)
metrics <- sqldf("
```

```
  select r_squared, 'CART' as model from rpart_metric
  union all
  select r_squared, 'RF' as model from rf_metric
  union all
  select r_squared, 'GBM' as model from gbm_metric
  union all
  select r_squared, 'Cubist' as model from cubist_metric
              ")

metrics %>% kable(caption = "R Squared Comparisons") %>%
    kable_styling() %>% row_spec()

# code
rpart_pred <- predict(rpart_tune, newdata = transform_x_test)
rpart_res <- data.frame()
rpart_res <- rbind(rpart_res, t(postResample(pred = rpart_pred,
    obs = y_test)))

rf_pred <- predict(rf_tune, newdata = transform_x_test)
rf_res <- data.frame()
rf_res <- rbind(rf_res, t(postResample(pred = rf_pred,
    obs = y_test)))

gbm_pred <- predict(gbm_tune, newdata = transform_x_test)
gbm_res <- data.frame()
gbm_res <- rbind(gbm_res, t(postResample(pred = gbm_pred,
    obs = y_test)))

cubist_pred <- predict(cubist_tune, newdata = transform_x_test)
cubist_res <- data.frame()
cubist_res <- rbind(cubist_res, t(postResample(pred = cubist_pred,
    obs = y_test)))

test_metric <- sqldf("
  select 'CART' as model, RMSE, Rsquared, MAE from rpart_res
  union all
  select 'RF' as model, RMSE, Rsquared, MAE from rf_res
  union all
  select 'GBM' as model, RMSE, Rsquared, MAE from gbm_res
  union all
  select 'Cubist' as model, RMSE, Rsquared, MAE from cubist_res
  ")
# code
cubist_imp <- varImp(cubist_tune, scale = FALSE)
plot(cubist_imp, top = 20, scales = list(y = list(cex = 0.8)))

# (8.7c)
library(partykit)
plot(as.party(rpart_tune$finalModel), gp = gpar(fontsize = 11))
```