Data 624_Exercise_HW9

Jiaxin Zheng

2025-04-19

Kuhn and Johnson, Chapter 8

Exercise 8.1 Recreate the simulated data from Exercise 7.2:

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.4.3
```

```r
library(ipred)
```

```
## Warning: package 'ipred' was built under R version 4.4.2
```

```r
library(party)
```

```
## Warning: package 'party' was built under R version 4.4.3
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
## Warning: package 'mvtnorm' was built under R version 4.4.2

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 4.4.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.4.3

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 4.4.3

##
## Attaching package: 'party'

## The following object is masked from 'package:dplyr':
##
##     where
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.3

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.4.3

## Loaded gbm 2.2.2

## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```r
library(Cubist)
```

```
## Warning: package 'Cubist' was built under R version 4.4.3

## Loading required package: lattice
```

```r
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.4.3
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.4.2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin
```

```r
library(partykit)
```

```
## Warning: package 'partykit' was built under R version 4.4.3

## Loading required package: libcoin

## Warning: package 'libcoin' was built under R version 4.4.3

##
## Attaching package: 'partykit'

## The following objects are masked from 'package:party':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner, node_surv,
##     node_terminal, varimp
```

```r
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)?

- Answer: No. But V1-V5 are more significant compare to V6-V10.

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

rfImp1 <- varImp(model1, scale = FALSE)

rfImp1
```

```
##          Overall
## V1   8.732235404
## V2   6.415369387
## V3   0.763591825
## V4   7.615118809
## V5   2.023524577
## V6   0.165111172
## V7  -0.005961659
## V8  -0.166362581
## V9  -0.095292651
## V10 -0.074944788
```

```r


simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
cor(simulated$duplicate1, simulated$V1)
```

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

```
## [1] 0.9460206
```

For example: 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?

- Answer: Yes, the importance score for V1 is change, the score is decreased. And other significant score is change too.

```r
model2<- randomForest(y ~.,
                      data = simulated,
                      importance = TRUE,
                      ntree = 1000)

rfImp2 <- varImp(model2, scale = FALSE)

rfImp2
```

```
##                Overall
## V1          5.69119973
## V2          6.06896061
## V3          0.62970218
## V4          7.04752238
## V5          1.87238438
## V6          0.13569065
## V7         -0.01345645
## V8         -0.04370565
## V9          0.00840438
## V10         0.02894814
## duplicate1  4.28331581
```

(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?

- Yes, all the scores are decreased again.

```r
cforestModel <- cforest(y ~ .,
        data = simulated)

cf <- data.frame(Overall = varimp(cforestModel, conditional= TRUE)) %>%
  arrange(desc(Overall))

cf
```

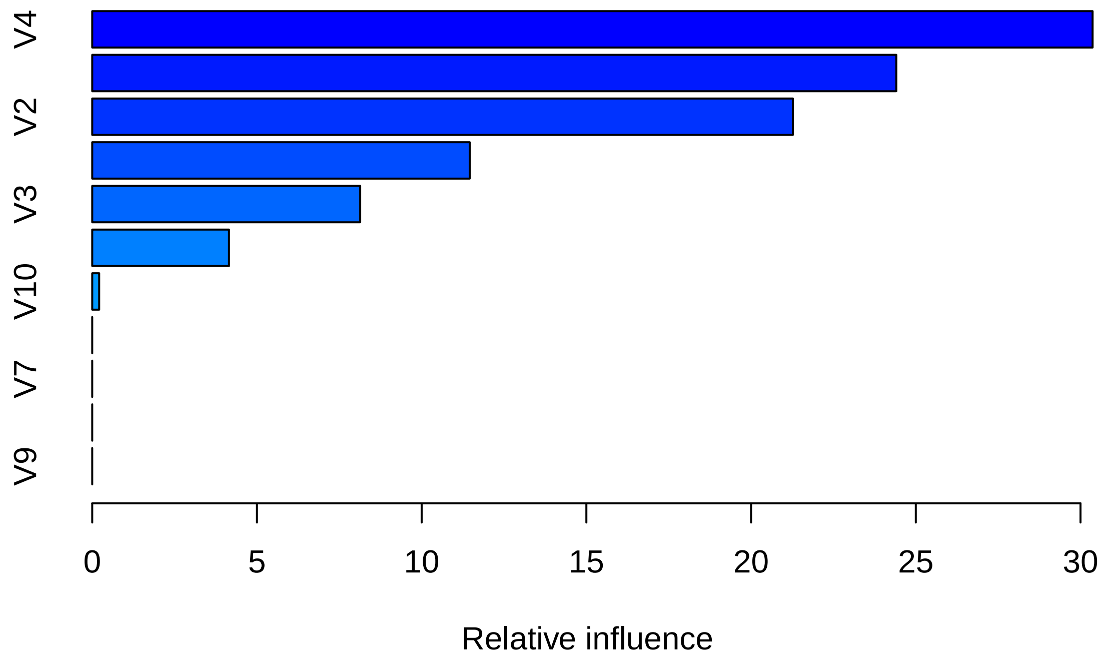
```
##                Overall
## V4          5.78489850
## V2          5.22819343
## V1          3.33065561
## duplicate1  2.78120557
## V5          1.39002706
## V3          0.04936459
## V6         -0.07029802
## V9         -0.11458990
## V7         -0.15477759
## V10        -0.17347857
## V8         -0.35461203
```

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

- Answer: No, the pattern are not the same. In Boosted trees,V4 has higher important scores. In Cubist V2 has higher important score. But V6 -V10 still not significant compare to others.

```
# boosted trees
boosted <- gbm(y ~.,
               data = simulated,
               distribution = "gaussian")

summary.gbm(boosted)
```



Boosted Trees

```
##                   var    rel.inf
## V4                 V4 30.3669149
## V1                 V1 24.4090151
## V2                 V2 21.2680313
## V5                 V5 11.4582221
## V3                 V3  8.1351552
## duplicate1 duplicate1  4.1515584
## V10               V10  0.2111029
## V6                 V6  0.0000000
## V7                   V7  0.0000000
## V8                   V8  0.0000000
## V9                   V9  0.0000000
```

```
# cubist

cubistModel <- train(y ~ .,
                     data = simulated,
                     method = "cubist")

varImp(cubistModel)
```

Cubist

```
## cubist variable importance
##
##            Overall
## V2          100.00
## V1           89.52
## V4           80.65
## V3           67.74
## duplicate1   59.68
## V5           50.00
## V6           25.00
## V8            0.00
## V10           0.00
## V9            0.00
## V7            0.00
```

Exercise 8.2

Use a simulation to show tree bias with different granularities.

- This simulation shows the bias of decision trees toward variables with higher granularity, even when the response is random.

```
set.seed(123)
simulated <- data.frame(
  x1 = rnorm(100),
  x2 = runif(100),
  x3 = sample(letters[1:3], 100, replace = TRUE),
  y = rnorm(100)
)

simulated$x3 <- as.factor(simulated$x3)

rpartTree <- rpart(y ~ ., data = simulated)
varImp(rpartTree)
```

```
##       Overall
## x1 0.50069936
## x2 0.29440594
## x3 0.04284495
```

Exercise 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?

- Answer: The model on the left side's both parameters are 0.1. When the bagging and learning rate are low, the importance of the predictors is more evenly distributed, and each individual tree gets trained, which prevents overfitting. The model on the right side's both parameters are set up at 0.9, which can cause overemphasis on a few predictors and may cause overfitting.

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

- Answer: The model have lower parameters(left side model) is more predictive.

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

- Answer: Increasing interaction depth will reduce the dominance of the top predictors and spread the importance over the predictors. The slope of the predictor plot would flatten.

Exercise 8.7 Refer to Exercises 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several tree-based models:

```r
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.4.3
```

```r
data(ChemicalManufacturingProcess)
```

```r
sum(is.na(ChemicalManufacturingProcess))
```

```
## [1] 106
```

```r
## impute missing values
imputed_df <- preProcess(ChemicalManufacturingProcess, "knnImpute")
imp_df <- predict(imputed_df, ChemicalManufacturingProcess)
```

```r
sum(is.na(imp_df))
```

```
## [1] 0
```

```r
# split the data in 80/20
set.seed(123)
train_index <- createDataPartition(imp_df$Yield, p = 0.8, list=FALSE)

train_data <- imp_df[train_index, ]
test_data  <- imp_df[-train_index, ]

# separate predictors and response
X_train <- train_data[, -1]  # Remove Yield column
y_train <- train_data$Yield

X_test <- test_data[, -1]
y_test <- test_data$Yield
```

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

- Answer: The Cubist model, it has higher R^2 0.7591132, and lower RMSE 0.4905725

```r
set.seed(123)
rf_model <- randomForest(x = X_train,
                         y = y_train,
                         importance = TRUE,
                         ntree = 1000)

rf_pred <- predict(rf_model, X_test)
postResample(rf_pred, y_test)
```

Random Forest Model

```
##      RMSE  Rsquared       MAE
## 0.6713856 0.5793669 0.5236411
```

```r
gbm_tune<- expand.grid(interaction.depth = seq(1, 7, by =2),
                       n.trees = seq(100, 1000, by =50),
                       shrinkage = c(0.01, 0.1),
                       n.minobsinnode = 10)
set.seed(123)
```

```r
gbm_model <- train(x = X_train,
                   y = y_train,
                   method = 'gbm',
                   tuneGrid = gbm_tune,
                   trControl = trainControl (method = "cv"),
                   verbose = FALSE)


gbm_pred <- predict(gbm_model, X_test)
postResample(gbm_pred, y_test)
```

Boosted Trees

```
##      RMSE  Rsquared       MAE
## 0.5698392 0.6760074 0.4627192
```

```r


set.seed(123)
cubist_model <- train(x = X_train,
                      y = y_train,
                      method = "cubist",
                      trControl = trainControl (method = "cv"))

cubist_pred <- predict(cubist_model, X_test)
postResample(cubist_pred, y_test)
```

Cubist

```
##      RMSE  Rsquared       MAE
## 0.4905725 0.7591132 0.3876428
```

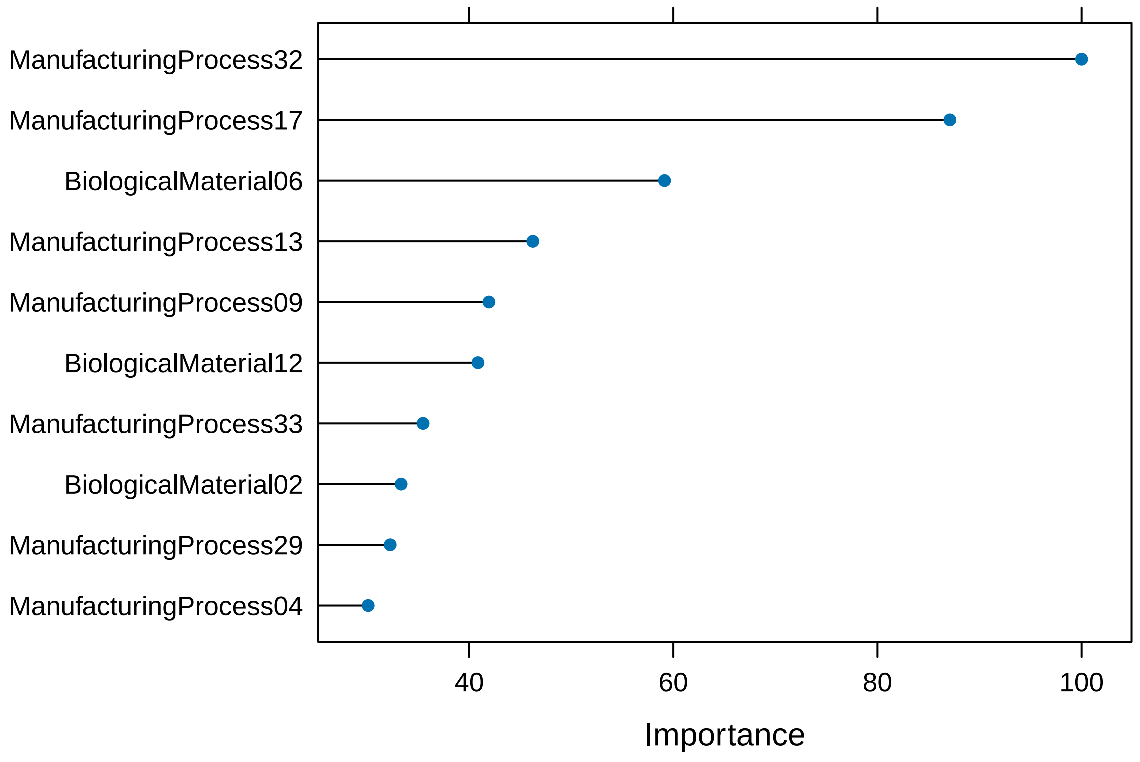
```r
list(RandomForest = postResample(rf_pred, y_test),
     Boosted = postResample(gbm_pred, y_test),
     Cubist = postResample(cubist_pred, y_test))
```

```
## $RandomForest
##      RMSE  Rsquared       MAE
## 0.6713856 0.5793669 0.5236411
##
## $Boosted
##      RMSE  Rsquared       MAE
## 0.5698392 0.6760074 0.4627192
##
## $Cubist
##      RMSE  Rsquared       MAE
## 0.4905725 0.7591132 0.3876428
```

(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?

- Answer: ManufacturingProcess32 is most important predictor in the model, and fellow by ManufacturingProcess17 and BiologicalMaterial06. Comparing the previous homework, ManufacturingProcess32 still top important predictor. The important predictors are similar.

```r
plot(varImp(cubist_model), top=10)
```



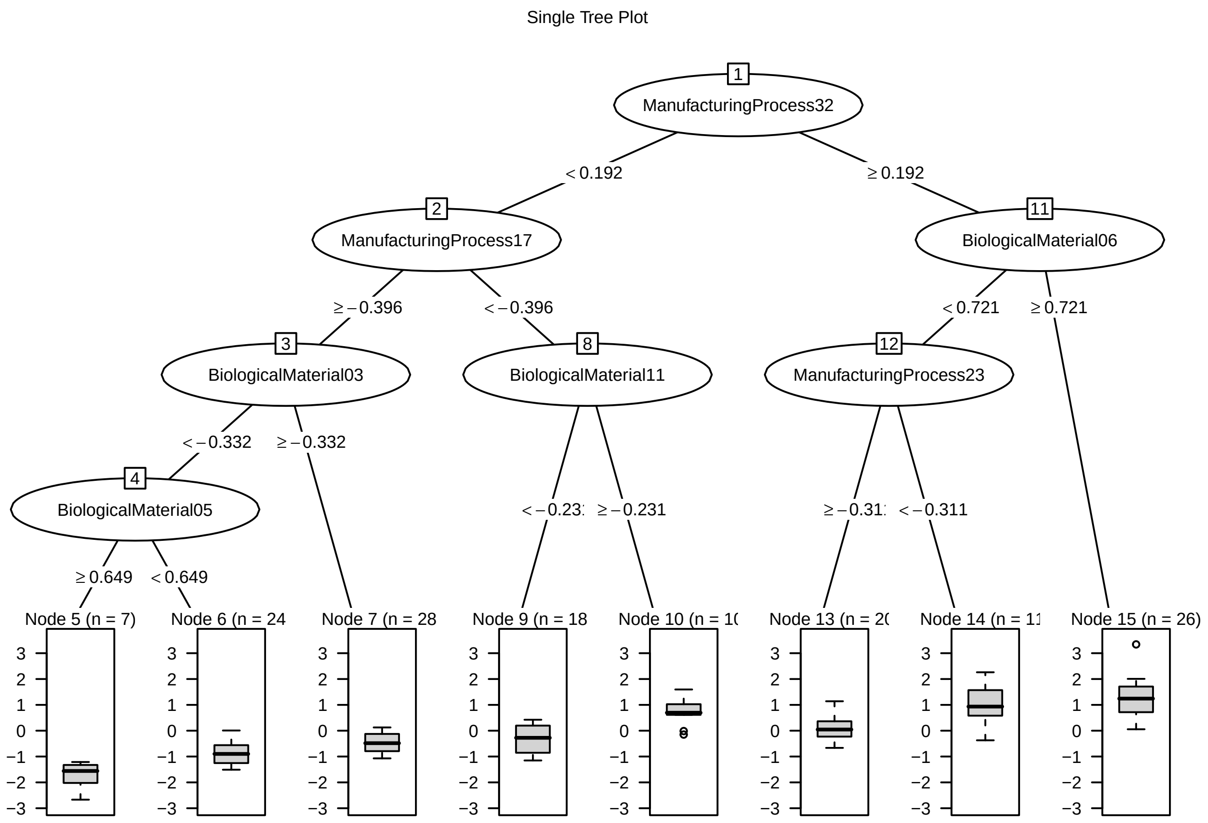
(c) Plot the optimal single tree with the distribution of yield in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield?

- Answer: Yes, this model show the relationship between each predictor in the tree.

```r
set.seed(1234)
treemodel <- train(x = X_train,
                   y = y_train,
                   method = "rpart",
                   trControl = trainControl("cv", number = 10),
                   tuneLength = 10)
```

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

```r
finaltreemodel <- treemodel$finalModel
tree_party <- as.party(finaltreemodel)
plot(tree_party, main = "Single Tree Plot", gp = gpar(fontsize = 7))
```


Single Tree Plot
1 ManufacturingProcess32
< 0.192
≥ 0.192
2 ManufacturingProcess17
11 BiologicalMaterial06
≥ −0.396
< −0.396
< 0.721
≥ 0.721
3 BiologicalMaterial03
8 BiologicalMaterial11
12 ManufacturingProcess23
< −0.332
≥ −0.332
4 BiologicalMaterial05
< −0.231
≥ −0.231
≥ −0.311
< −0.311
≥ 0.649
< 0.649
Node 5 (n = 7)
Node 6 (n = 24
Node 7 (n = 28
Node 9 (n = 18
Node 10 (n = 10
Node 13 (n = 20
Node 14 (n = 11
Node 15 (n = 26)