Decision tree hw9

Chi Hang(Philip) Cheung

2025-04-15

```
library(mlbench)
library(randomForest)
library(caret)
library(tidyverse)
library(party)
library(gbm)
library(Cubist)
library(rpart)
```

8.1)

Did the random forest model significantly use the uninformative predictors (V6 – V10)?

Ans: According to the varImp data, the model did not rank V6-V10 with much importance. V1-V5 are much more significant when compared to those.

```
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"
```

```
library(randomForest)
library(caret)
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)
rfImp1 <- varImp(model1, scale = FALSE)
rfImp1 %>% arrange(desc(Overall))
```

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

b) 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?

Ans: Compared to the original data, V1 was demoted from the importance rank. In addition, the duplicate1 that has high correlation with V1 is being ranked almost as important as V1. The effect of high correlation between predictors is affecting how V1 was being ranked.

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

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

```
set.seed(123)
model_2<- randomForest(y ~.,
                       data=simulated,
                       importance = TRUE, ntree = 1000)
new_imp<-varImp(model_2, scale=FALSE)
new_imp %>% arrange(desc(Overall))
```

```
##                Overall
## V4          6.84585204
## V2          6.17711385
## V1          5.62937024
## duplicate1  4.32662911
## V5          1.81782072
## V3          0.73469142
## V6          0.06863788
## V9         -0.04279069
## V7         -0.04750291
## V10        -0.04786506
## V8         -0.07015860
```

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 importance show the same pattern as the traditional random forest model?

Ans: The importance ranking did not change in both options. The ranking still follows the importance scale as the original model. The only difference is the slight changes in the numeric values of the rankings.

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

rank_conditional<- varimp(c_model, conditional = TRUE) %>% sort(decreasing=TRUE)
rank_unconditional<- varimp(c_model, conditional = FALSE)%>% sort(decreasing=TRUE)

#Combining all results into one dataframe
```

```
rank_comparsion<- cbind(new_imp, rank_conditional, rank_unconditional)

#Chaning colum name
names(rank_comparsion)[1]<-'original_ranking'
rank_comparsion
```

```
##            original_ranking rank_conditional rank_unconditional
## V1                5.62937024      5.973580188        7.392804129
## V2                6.17711385      4.801569808        6.093296844
## V3                0.73469142      1.947654295        5.085813474
## V4                6.84585204      1.915734317        4.428669947
## V5                1.81782072      1.018403069        1.864273405
## V6                0.06863788      0.027304648        0.015577318
## V7               -0.04750291      0.006510864       -0.007276374
## V8               -0.07015860      0.006153603       -0.013143895
## V9               -0.04279069      0.001293016       -0.024148669
## V10              -0.04786506     -0.001297792       -0.029442569
## duplicate1        4.32662911     -0.008865821       -0.029797156
```

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

Ans: The cubist model ranks V1-5 in the same order as the randomForest model while the boosted Tree model ranks V4 as the most important followed by V1, v2, V5, V3. From V6 to V10, they are still being regarded as no useful same as randomForest model.

Using the boosted model:

```
set.seed(123)
#Removing the duplicate1 column:
simulated<- simulated %>% select(-duplicate1)

#fit the boosted tree model:
og_gbm_model<-train(y~.,
                    data = simulated,
                    method = 'gbm',
                    distribution = "gaussian",
                    verbose = FALSE)

varImp(og_gbm_model)
```

```
## gbm variable importance
##
##      Overall
## V4  100.0000
## V1   99.5369
## V2   93.1077
## V5   45.6778
## V3   29.8281
## V7    2.4474
```

```
## V8    2.1192
## V6    1.5888
## V10   0.2352
## V9    0.0000
```

Using the Cubist model:

```
set.seed(123)
simulated_x<- simulated %>% select(-y)

cubist<- train(x=simulated_x, y=simulated$y,
               method='cubist')

varImp(cubist)
```

```
## cubist variable importance
##
##     Overall
## V1   100.00
## V2    75.69
## V4    68.06
## V3    58.33
## V5    55.56
## V6    15.28
## V8     0.00
## V10    0.00
## V7     0.00
## V9     0.00
```

8.2) Use a simulation to show tree bias with different granularities.

Ans: As shown in this example, the number of splits in the tree model for low, mid, and high are 591, 356, 53, respectively. The tree model has a bias to split data points that are continuous with high granularity. Fortunately, there are additional methods to combat this issue with ensemble, pruning, smoothing, and bootstrapping.

```
set.seed(3341)
#Repeat the process 1000 times:
n_iteration<- 1000

#Store the variable for the first split:
root_vars<- character(n_iteration)

#Main loop for repeating the modeling 1000 times:
for (i in 1:n_iteration){
#Define low, mid, high variance:
low<-sample(c('A','B'), 100, replace = TRUE)
mid<-sample(letters[1:5],100, replace = TRUE)
high<-rnorm(100)
y<- rnorm(100)
```

```
df<- data.frame(y=y,
                low=factor(low),
                mid=factor(mid),
                high=high)

test_model<- rpart(y~., data=df)

#Extract root node split variable:
root_var <- as.character(test_model$frame$var[1])
root_vars[i] <- root_var
}
#Counting which variable splits for which level of granularity:
table(root_vars)
```

```
## root_vars
## high  low  mid
##  591   53  356
```

8.3 a)

Ans: The model with high bagging and learning rate applies that there are less randomness in sample selection through the bagging process and higher model impact on the final model in the learning rate. Therefore, this model quickly identities and put a lot of emphasis on the top few predictors as shown on the plot. On the other hand, when the bagging and learning rate are low, the model is able to have more randomness and room for involving more predictor features into the final model and that is why we see more predictors are being considered as important. The former and prone to overfitting and the latter is prone to underfitting.

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

Ans: The model on the left with lower learning rate and bagging would be better suited for more predictive modeling. This is because the model less overfitted as compared to the one with more extreme learning and bagging parameters and therefore can analysis another new sample with better predictive performance.

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

Ans: Increasing the interaction depth allows the trees to grow deeper and therefore more specific to the training data. The effect might be similar to increasing the learning rate and bagging parameter to very high. The slope of the predictors will become steeper and there might more only a selected few predictors become more predominant.

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?

Ans: RandomForest is the most optimal model with the resampling and test set performance of 0.6556898 0.6579293 and 0.4948748 0.7456382 for RMSE and R^2, respectively.

```
library(AppliedPredictiveModeling)
data(ChemicalManufacturingProcess)
```

Train split the data:

```
set.seed(123)
#checking for missing data:
is.na(ChemicalManufacturingProcess) %>% sum() #106 missing data
```

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

```
#impute the data with knn:
imputed_data<- preProcess(ChemicalManufacturingProcess, method = 'knnImpute')

#apply to the dataset:
imputed_chem<- predict(imputed_data, ChemicalManufacturingProcess)
imputed_chem %>% is.na() %>% sum() # 0 missing data
```

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

```
#spliting data in 70-30:
chem_train_idx<- createDataPartition(imputed_chem$Yield, p = 0.7, list=FALSE)

#training dataset
chem_train_x<- imputed_chem[chem_train_idx, ] %>% select(-Yield)
chem_train_y<- imputed_chem$Yield[chem_train_idx]

#testing dataset:
chem_test_x<- imputed_chem[-chem_train_idx, ] %>% select(-Yield)
chem_test_y<- imputed_chem$Yield[-chem_train_idx]
```

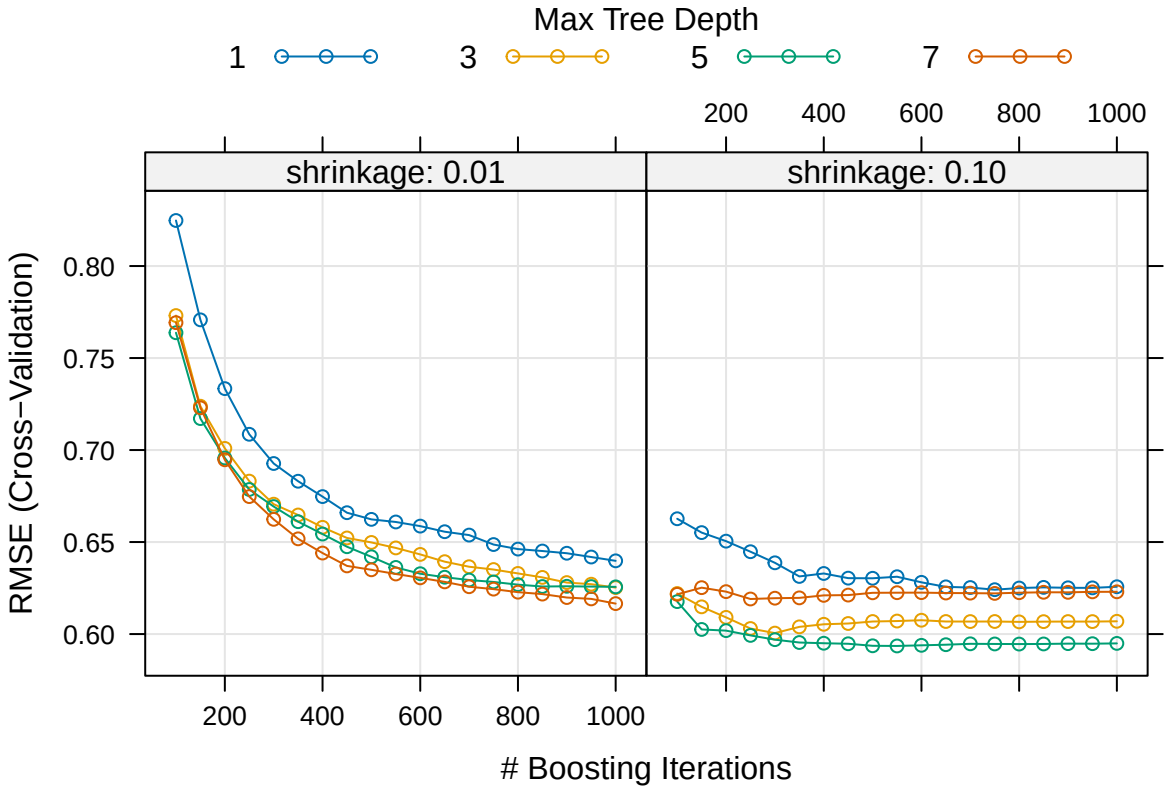
```
#Define CrossValidation parameter:
cv<- trainControl(method='cv', number = 10)
```

randomForest model:

```
set.seed(50)
#tuning mtry
rf_tune<-expand.grid(.mtry=(5:20))

rf_model<- train(x=chem_train_x,
                 y=chem_train_y,
                 method = 'rf',
                 tuneGrid = rf_tune,
                 trControl=cv,
                 ntree=1000)
```

Boosted Tree model

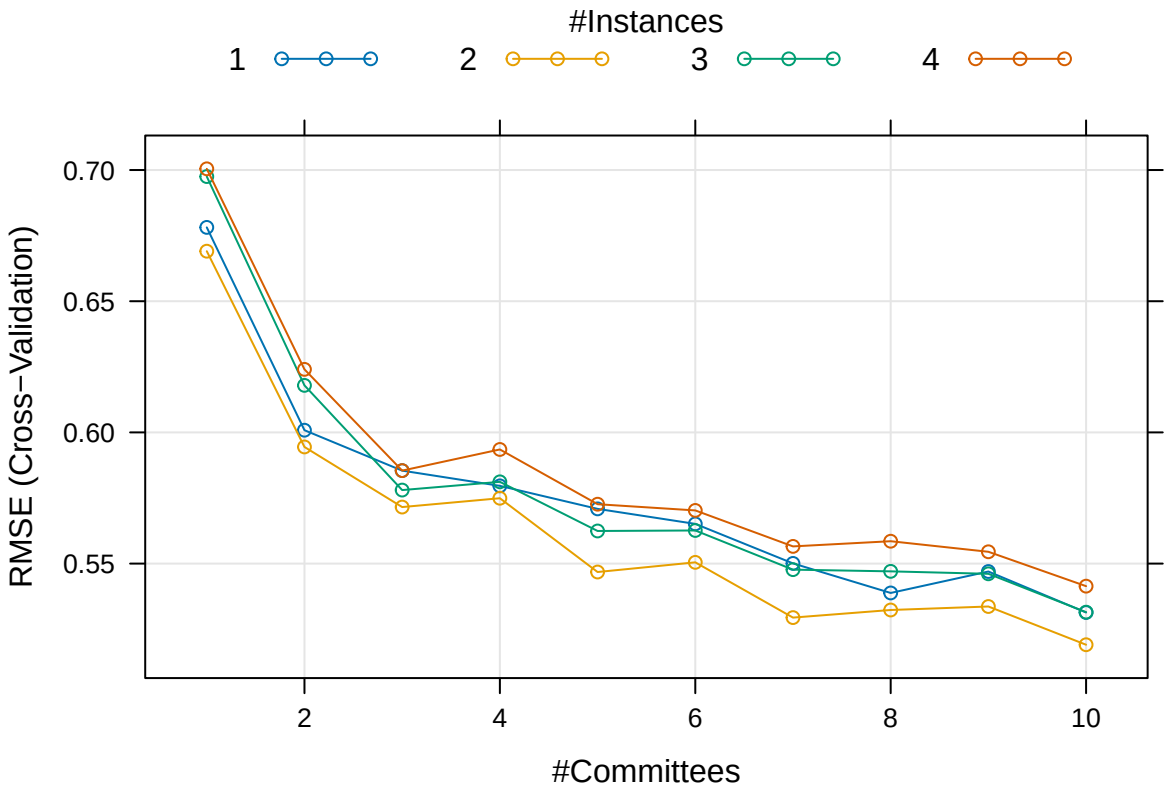
```
set.seed(50)
#define tuning parameters:
gbm_tune<- expand.grid(interaction.depth = seq(1, 7, by =2),
                       n.trees = seq(100, 1000, by =50),
                       shrinkage = c(0.01, 0.1),
                       n.minobsinnode = 10)

gbm_model<- train(x=chem_train_x,
                  y=chem_train_y,
                  method = 'gbm',
                  tuneGrid = gbm_tune,
                  trControl=cv,
                  verbose = FALSE)
plot(gbm_model)
```



Cubist:

```
set.seed(50)
cub_tune<-expand.grid(committees = 1:10,
                      neighbors = 1:4)
cub_model<- train(x=chem_train_x,
                  y=chem_train_y,
                  method = 'cubist',
                  trControl = cv,
                  tuneGrid = cub_tune)
```

```
plot(cub_model)
```



Compiling the prediction results:

```
rf_pred<- predict(rf_model, chem_test_x)
gbm_pred<- predict(gbm_model, chem_test_x)
cub_pred<- predict(cub_model, chem_test_x)

list(randomForest = postResample(rf_pred, chem_test_y),
     Boosted = postResample(gbm_pred, chem_test_y),
     cubist = postResample(cub_pred, chem_test_y))
```

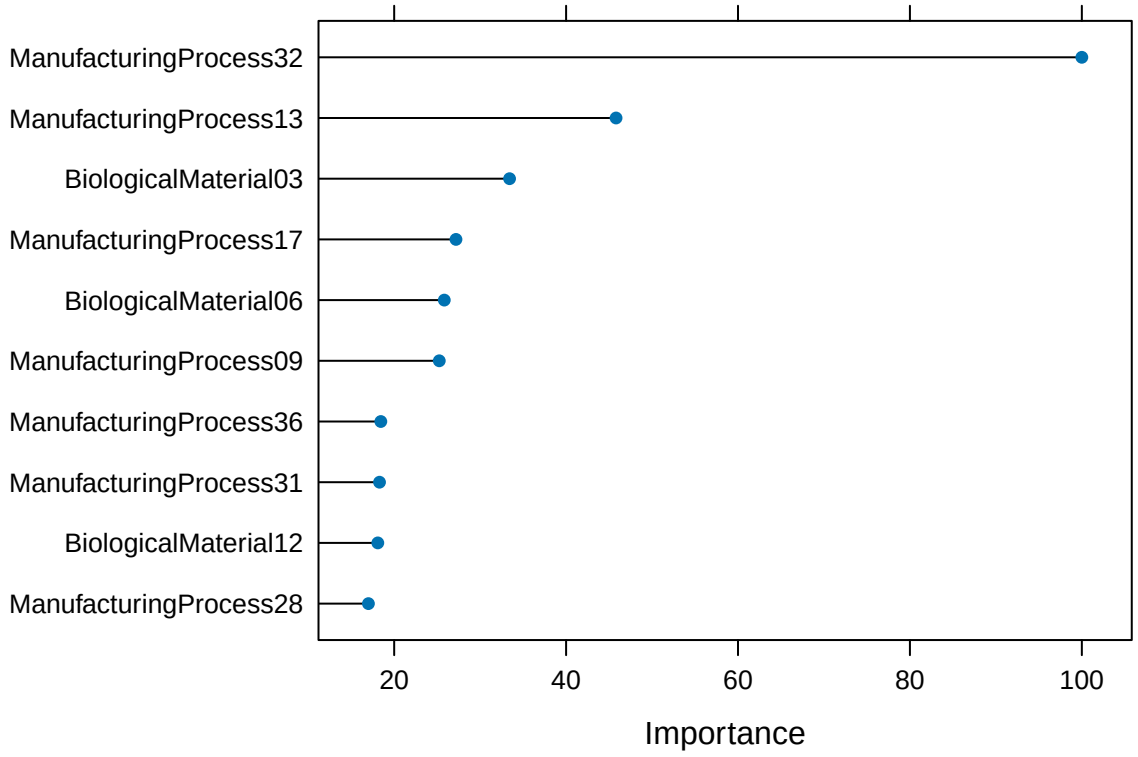
```
## $randomForest
##      RMSE  Rsquared       MAE
## 0.4945301 0.7537722 0.3894710
##
## $Boosted
##      RMSE  Rsquared       MAE
## 0.5707246 0.6443742 0.4609822
##
## $cubist
##      RMSE  Rsquared       MAE
## 0.6228166 0.5759597 0.4516156
```

(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?

Ans: MP32 is the most important predictor for the randomForest model followed by the other ones shown below. The Manufacturing processes dominate the importance list. Comparing the linear and non-linear models done in the previous homeworks, the important predictors are similar though the ordering might be slightly different.

```
plot(varImp(rf_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?

Ans: This view reveals interesting insights into how some of the predictors are related with each other. For example, MP32 is closely related to BM03–When both are high, the yield is consistently high. Similarly, when MP32, MP17, and BM03 are all low (with MP32 at different split point) the yield is consistent low. However, this is only a single tree, which can be prone to overfitting and bias. While it is more interpretable, a more robust understanding of predictor interactions would come from ensemble methods such as randomForest or gradient boosting, which aggregate over many trees and better capture complex relationships.

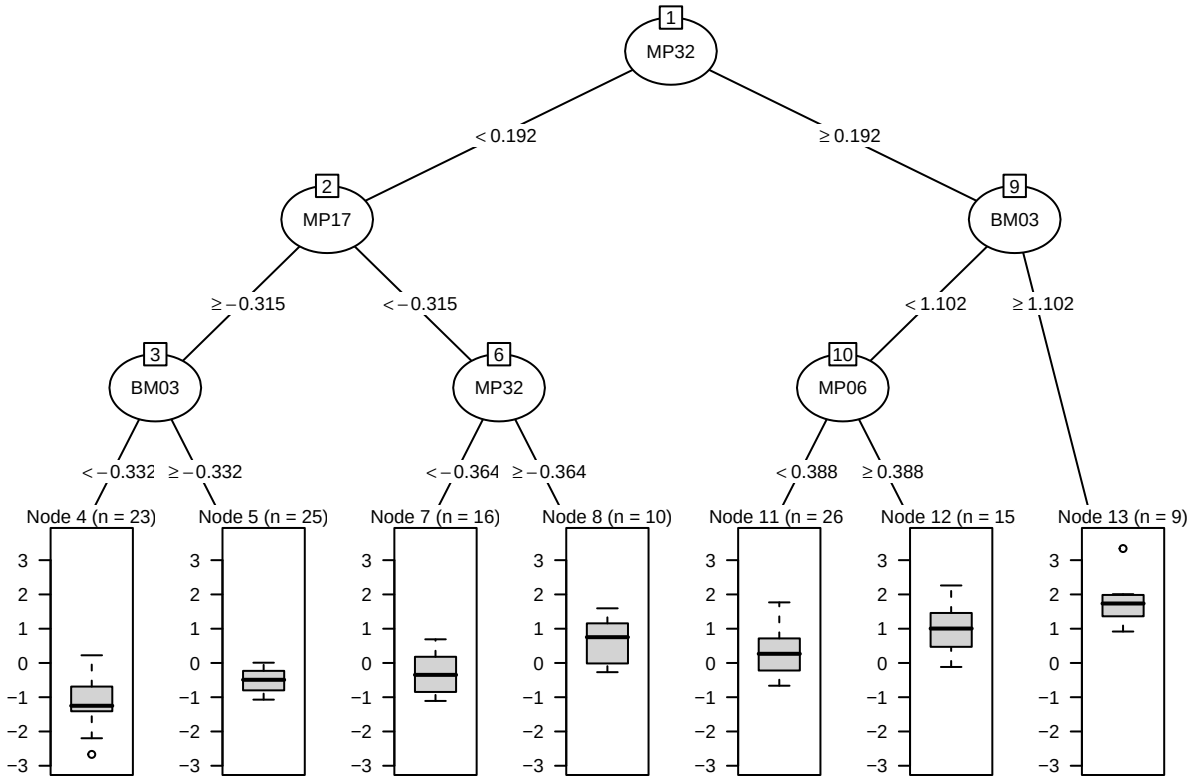
```
set.seed(23)

#combine
single_tree_model<- train(x=chem_train_x,
                   y=chem_train_y,
                   method = 'rpart2',
                   tuneLength = 10,
                   trControl = trainControl(method='cv', number=10))
```

```
## note: only 8 possible values of the max tree depth from the initial fit.
##  Truncating the grid to 8 .
```

```
single_tree<-single_tree_model$finalModel

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
```

```
party_tree<- as.party(single_tree)
plot(party_tree,
     ip_args = list(abbreviate=4),
     gp = gpar(fontsize = 7))
```


1
MP32
< 0.192
≥ 0.192
2
MP17
9
BM03
≥ −0.315
< −0.315
< 1.102
≥ 1.102
3
BM03
6
MP32
10
MP06
< −0.332
≥ −0.332
< −0.364
≥ −0.364
< 0.388
≥ 0.388
Node 4 (n = 23)
Node 5 (n = 25)
Node 7 (n = 16)
Node 8 (n = 10)
Node 11 (n = 26
Node 12 (n = 15
Node 13 (n = 9)
3
2
1
0
−1
−2
−3