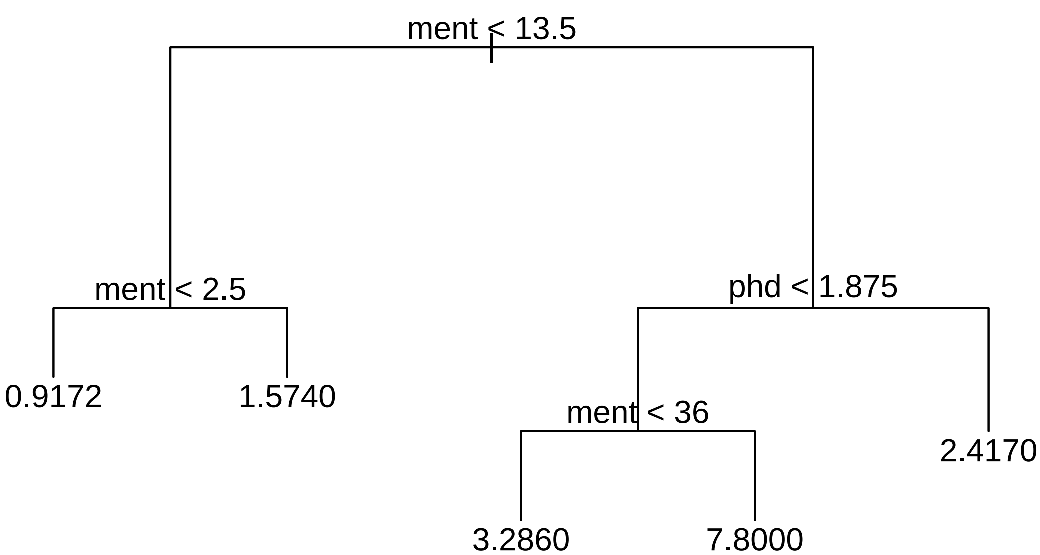
Part 3, Tree based methods

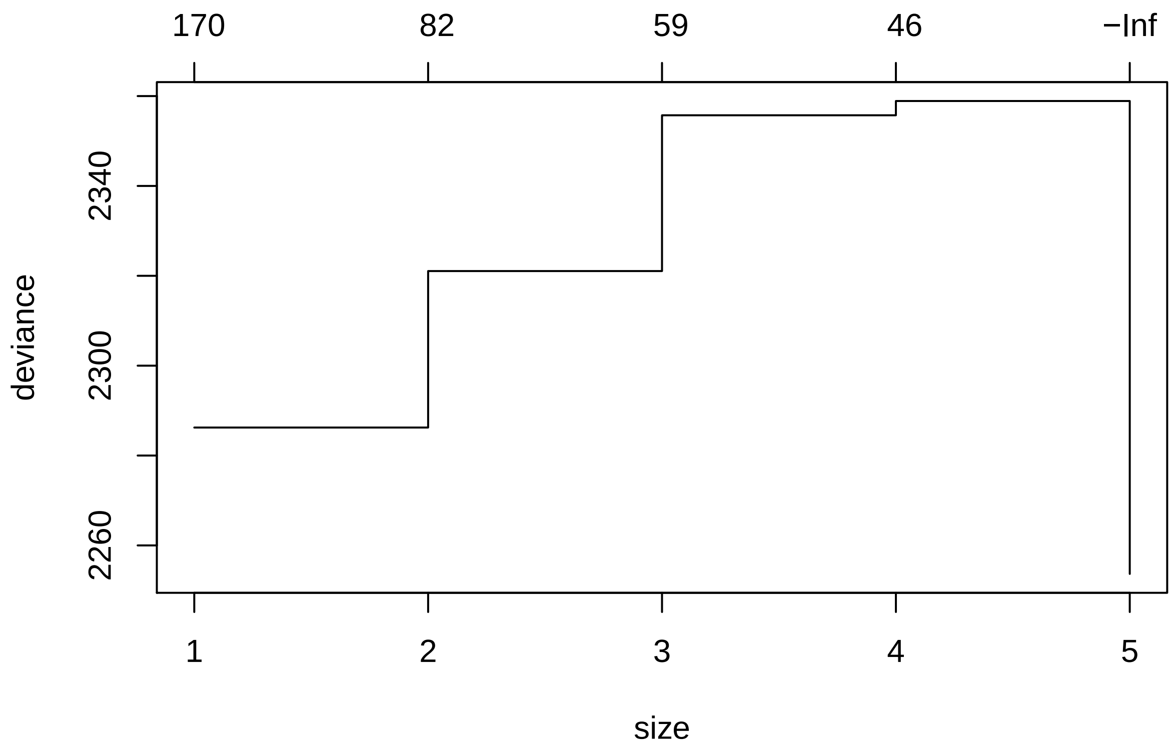
```
#Prepare the data
dataTrain2<-dataTrain
dataTrain2$fem<-(dataTrain2$fem=="Women")*1
dataTrain2$mar<-(dataTrain2$mar=="Married")*1

dataTest2<- dataTest
dataTest2$art<- log((dataTest2$art)+1)

dataTrain3<- dataTrain2
dataTrain3$art<- log((dataTrain3$art)+1)

dataTrain4<- dataTrain
dataTrain4$art<- log((dataTrain4$art)+1)

xTrain<-model.matrix(art~.,data=dataTrain)[,-1]
yTrain<-dataTrain$art
yTrain2<-log(yTrain+1)
```

3.1, CART

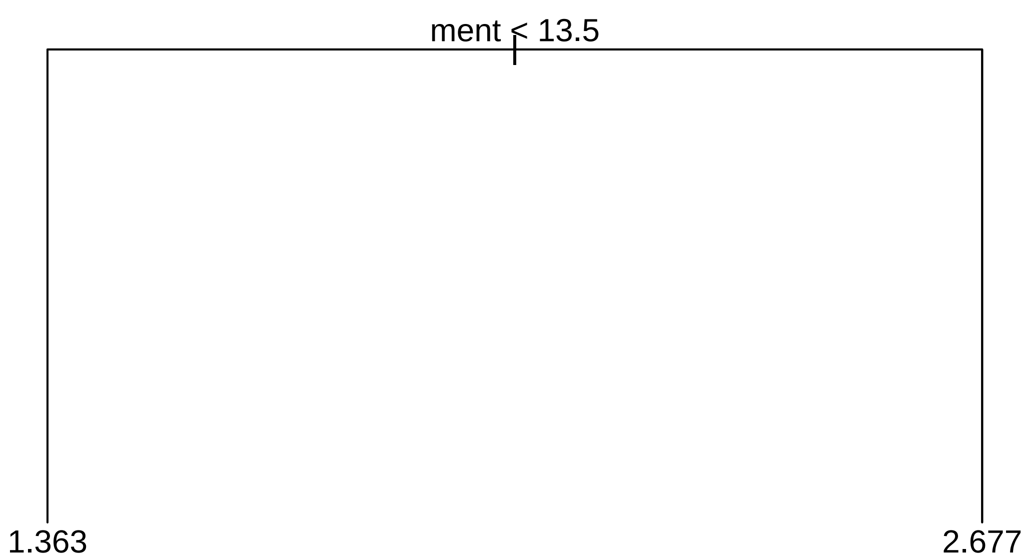
```
#Grow full tree
fullTree<-tree(art~.,data=dataTrain2)
#fullTree
#summary(fullTree) #Residual mean deviance:  3.08 = 1879 / 610, sum of squared error/(615-2)
plot(fullTree)
text(fullTree,pretty=0)
```




```
#Run CV
cvTree<-cv.tree(fullTree)
#cvTree
plot(cvTree)
```




```
#Use CV to prune tree
mod2<-prune.tree(fullTree,best=2)
plot(mod2)
text(mod2,pretty=0)
```

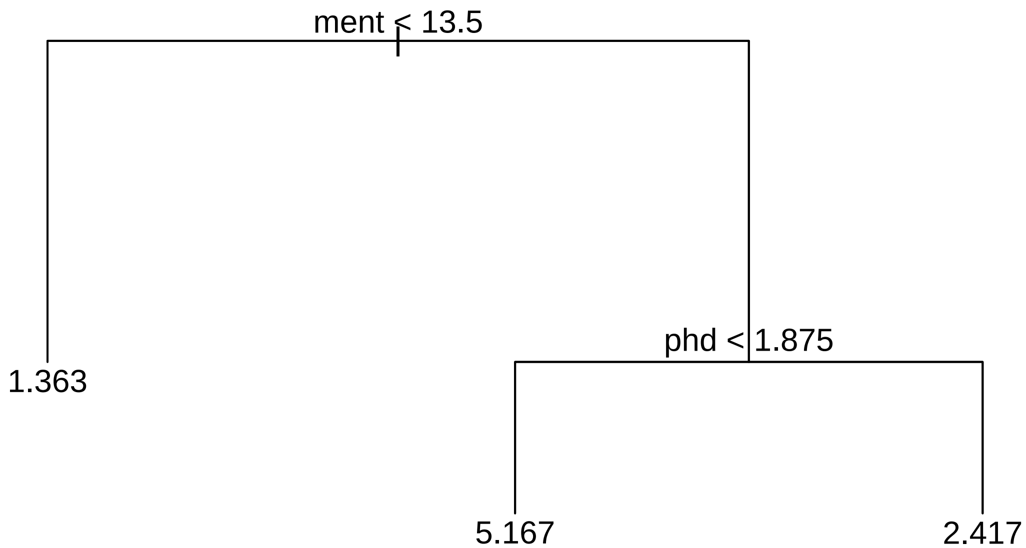



```
summary(mod2) #Residual mean deviance:  3.371 = 2067 / 613
```

```
##
## Regression tree:
## snip.tree(tree = fullTree, nodes = 2:3)
## Variables actually used in tree construction:
## [1] "ment"
## Number of terminal nodes:  2
```

```
## Residual mean deviance:  3.371 = 2067 / 613
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.6770 -1.3630 -0.3627  0.0000  0.6373 16.3200
```

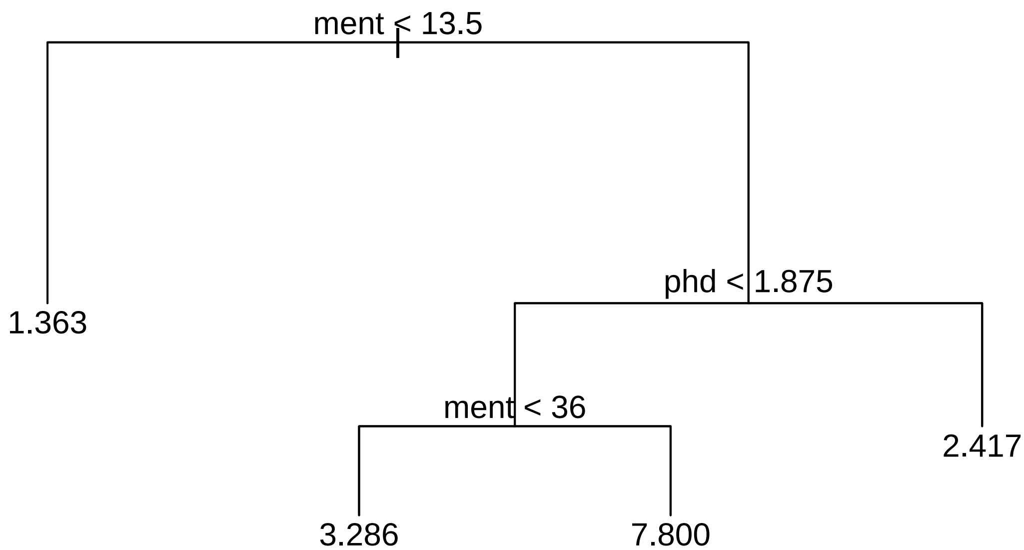
```r
MSEMod2<-mean((dataTrain$art-predict(mod2))^2)

mod3<-prune.tree(fullTree,best=3)
plot(mod3)
text(mod3,pretty=0)
```




```r
summary(mod3) #Residual mean deviance:  3.243 = 1984 / 612
```

```
##
## Regression tree:
## snip.tree(tree = fullTree, nodes = c(2L, 6L))
## Variables actually used in tree construction:
## [1] "ment" "phd"
## Number of terminal nodes:  3
## Residual mean deviance:  3.243 = 1984 / 612
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.1670 -1.3630 -0.3627  0.0000  0.6373 13.8300
```

```r
MSEMod3<-mean((dataTrain$art-predict(mod3))^2) #3.22672

mod4<-prune.tree(fullTree,best=4)
plot(mod4)
text(mod4,pretty=0)
```




```
summary(mod4) #Residual mean deviance:  3.151 = 1925 / 611
```

```
##
## Regression tree:
## snip.tree(tree = fullTree, nodes = 2L)
## Variables actually used in tree construction:
## [1] "ment" "phd"
## Number of terminal nodes:  4
## Residual mean deviance:  3.151 = 1925 / 611
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -6.8000 -1.3630 -0.3627  0.0000  0.6373 12.7100
```

```
MSEMod4<-mean((dataTrain$art-predict(mod4))^2)
```

Even the full tree only considers "mentor" (number of publications by the PhD student's mentor) and "PhD" (the prestige of the student's PhD program). So we do not expect this method to work well.

3.2, Random forest

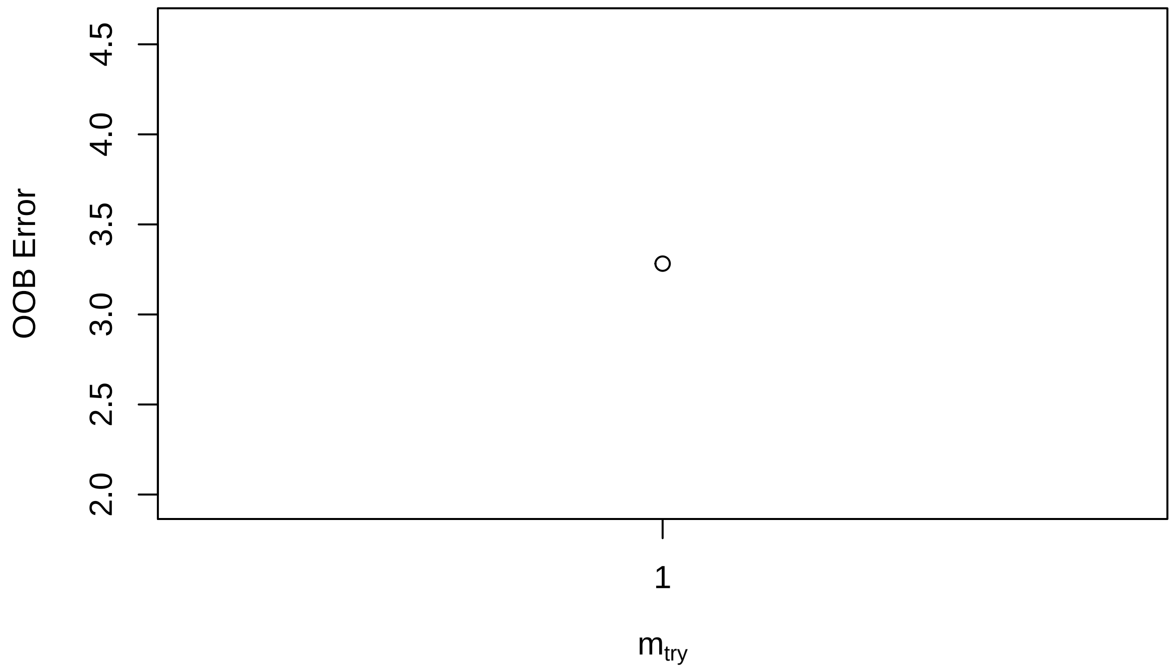
```
#Random forest with default mtry
modRf<-randomForest(art~.,mtry=1,data=dataTrain) #The default is mtry=p/3=1.
modRf
```

```
##
## Call:
##  randomForest(formula = art ~ ., data = dataTrain, mtry = 1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 3.344061
##                     % Var explained: 8.22
```

```
MSERf<-mean((dataTrain$art-predict(modRf))^2) #3.34406

#Tune mtry
```

```
modRfBestMtry<-tuneRF(xTrain,yTrain,stepFactor=1,improve=0)
```

```
## mtry = 1  OOB error = 3.282155
## Searching left ...
## Searching right ...
```




```
print(modRfBestMtry) #mtry=1 is the best based on out of bag error
```

```
##   mtry OOBError
## 1    1 3.282155
```

3.3, XG boost

```
#Default XG boost
modXGB<-xgboost(data=xTrain,label=yTrain,nrounds=2)
```

```
## [1]  train-rmse:1.911969
## [2]  train-rmse:1.709288
```

```
#Tune nround with the default parameters
params<-list(
        booster="gbtree",
        objective="reg:linear",
        eta=0.3,
        gamma=0,
        max_depth=6,
        min_child_weight=1,
        subsample=1,
        colsample_bytree=1
)

xgbCV<-xgb.cv(params=params,
```

```r
              data = xTrain,
              label= yTrain,
              nrounds=100,
              nfold=5,
              showsd=T,
              stratified=T,
              print_every_n=10,
              early_stop_round=20,
              maximize=F
)
```

```
## [1]  train-rmse:1.915956+0.065556    test-rmse:2.017608+0.332703
## [11] train-rmse:1.137338+0.048806    test-rmse:2.012689+0.276076
## [21] train-rmse:0.931552+0.035198    test-rmse:2.111717+0.285411
## [31] train-rmse:0.777405+0.020026    test-rmse:2.167888+0.300436
## [41] train-rmse:0.678207+0.022501    test-rmse:2.199230+0.315014
## [51] train-rmse:0.594917+0.027484    test-rmse:2.234491+0.316812
## [61] train-rmse:0.542921+0.030475    test-rmse:2.258721+0.316290
## [71] train-rmse:0.506680+0.022442    test-rmse:2.274018+0.321953
## [81] train-rmse:0.471382+0.018665    test-rmse:2.288732+0.323788
## [91] train-rmse:0.446186+0.018254    test-rmse:2.302405+0.325628
## [100]    train-rmse:0.431080+0.016182    test-rmse:2.307605+0.325112
```

```r
minTestRMSE=min(xgbCV$evaluation_log$test_rmse_mean)
minTestRMSEIndex=which.min(xgbCV$evaluation_log$test_rmse_mean)

#Alternative method to tune nrounds
modXGB.CV<-xgb.cv(data=xTrain,label=yTrain,nfold=5,nrounds=20)
```

```
## [1]  train-rmse:1.905512+0.069650    test-rmse:2.026296+0.318136
## [2]  train-rmse:1.689481+0.060030    test-rmse:1.946860+0.292776
## [3]  train-rmse:1.543210+0.052029    test-rmse:1.913599+0.272380
## [4]  train-rmse:1.441693+0.048028    test-rmse:1.918305+0.237305
## [5]  train-rmse:1.360571+0.047341    test-rmse:1.935169+0.226852
## [6]  train-rmse:1.297917+0.043551    test-rmse:1.953087+0.230017
## [7]  train-rmse:1.246186+0.047659    test-rmse:1.972470+0.239015
## [8]  train-rmse:1.204666+0.050808    test-rmse:1.965689+0.247839
## [9]  train-rmse:1.172656+0.050107    test-rmse:1.972537+0.257071
## [10] train-rmse:1.139686+0.048997    test-rmse:1.976760+0.252780
## [11] train-rmse:1.119055+0.049189    test-rmse:1.987049+0.262225
## [12] train-rmse:1.089031+0.042958    test-rmse:1.991923+0.266981
## [13] train-rmse:1.067113+0.039857    test-rmse:1.999963+0.270291
## [14] train-rmse:1.044686+0.033910    test-rmse:2.003126+0.270686
## [15] train-rmse:1.019185+0.026914    test-rmse:2.007930+0.266598
## [16] train-rmse:0.994014+0.021485    test-rmse:2.008174+0.259562
## [17] train-rmse:0.974893+0.016795    test-rmse:2.009966+0.260136
## [18] train-rmse:0.955394+0.012492    test-rmse:2.018277+0.258745
## [19] train-rmse:0.937368+0.015835    test-rmse:2.026153+0.260328
## [20] train-rmse:0.921733+0.013529    test-rmse:2.028869+0.263342
```

```r
minTestRMSE2=min(modXGB.CV$evaluation_log$test_rmse_mean)
minTestRMSEIndex2=which.min(modXGB.CV$evaluation_log$test_rmse_mean) #nrounds=3 is the best

modXGB3<-xgboost(data=xTrain,label=yTrain,nrounds=3)
```

```
## [1]  train-rmse:1.911969
## [2]  train-rmse:1.709288
## [3]  train-rmse:1.561789
```

```
MSEXGB<-mean((dataTrain$art-predict(modXGB3,xTrain))^2) #2.43918
```

```
#Tune the parameters using the MLR package
traintask<-makeRegrTask(data=dataTrain2,target="art")
testtask<-makeRegrTask(data=dataTest,target="art")
```

To do:

We will think about how to compare the tree based methods/KRLS to the GLM methods from Part 1 and ridge and LASSO from Part 2.

TESTING RESULTS:

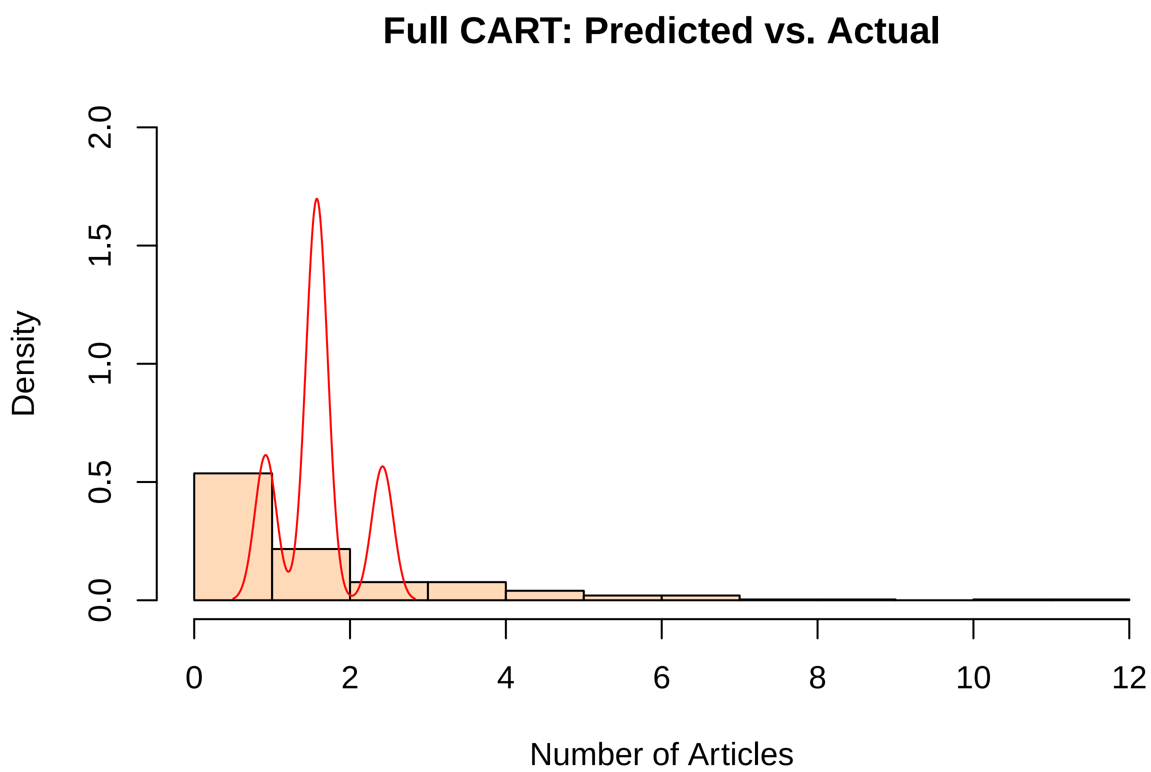
```
yTest <- dataTest[,1]
xTest3names <-names(as.data.frame(dataTest[,-1]))
xTest5 <- model.matrix(art~.,data=dataTest)[,-1]
xTest3 <- as.data.frame(xTest5)
names(xTest3) <- xTest3names
xTest4 <-dataTest[,-1]
```

FULL CART TEST MSE

```
yHatFULLCARTTest<- predict(fullTree,newdata = as.data.frame(xTest3))
MSEFullTest<-mean((yTest -yHatFULLCARTTest)^2)
sqrt(MSEFullTest)
```

```
## [1] 1.890329
```

```
hist(yTest,freq = F, ylim = c(0,2),col="peachpuff", main = "Full CART: Predicted vs. Actual", xlab = "Nu
lines(density(yHatFULLCARTTest), col = "red")
```

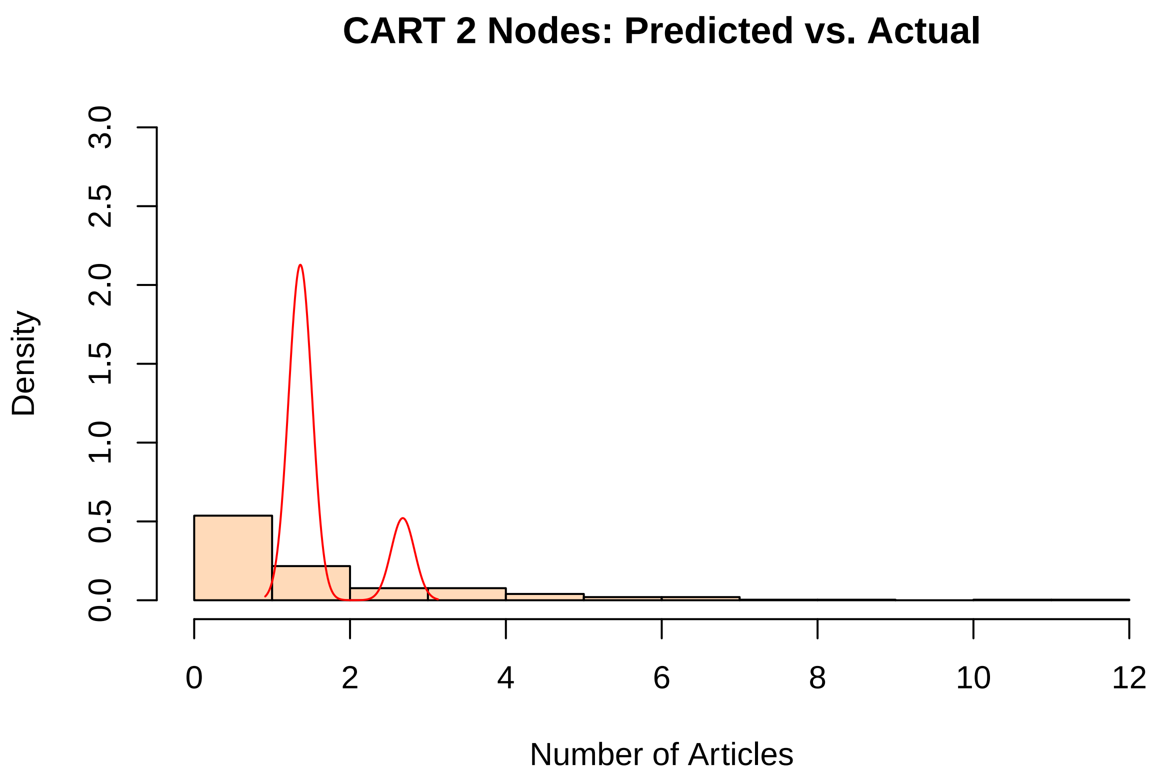


CART 2 Nodes TEST MSE

```
yHatMod2CARTTest<- predict(mod2,newdata = as.data.frame(xTest3))
MSEMod2Test<-mean((yTest -yHatMod2CARTTest)^2)
sqrt(MSEMod2Test)
```

```
## [1] 1.907779
```

```
hist(yTest,freq = F, ylim = c(0,3),col="peachpuff", main = "CART 2 Nodes: Predicted vs. Actual", xlab =
lines(density(yHatMod2CARTTest), col = "red")
```

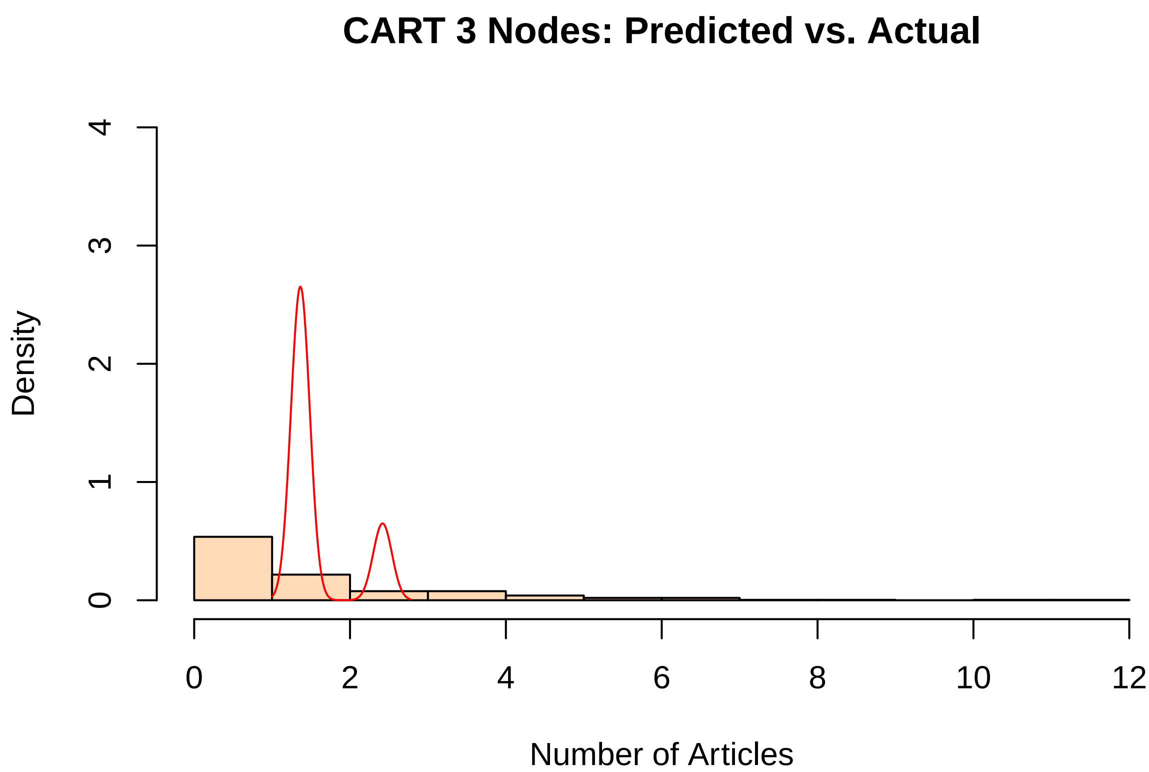


CART 3 Nodes TEST MSE

```
yHatMod3CARTTest<- predict(mod3,newdata = as.data.frame(xTest3))
MSEMod3Test<-mean((yTest -yHatMod3CARTTest)^2)
sqrt(MSEMod3Test)
```

```
## [1] 1.913087
```

```
hist(yTest,freq = F, ylim = c(0,4),col="peachpuff", main = "CART 3 Nodes: Predicted vs. Actual", xlab =
lines(density(yHatMod3CARTTest), col = "red")
```

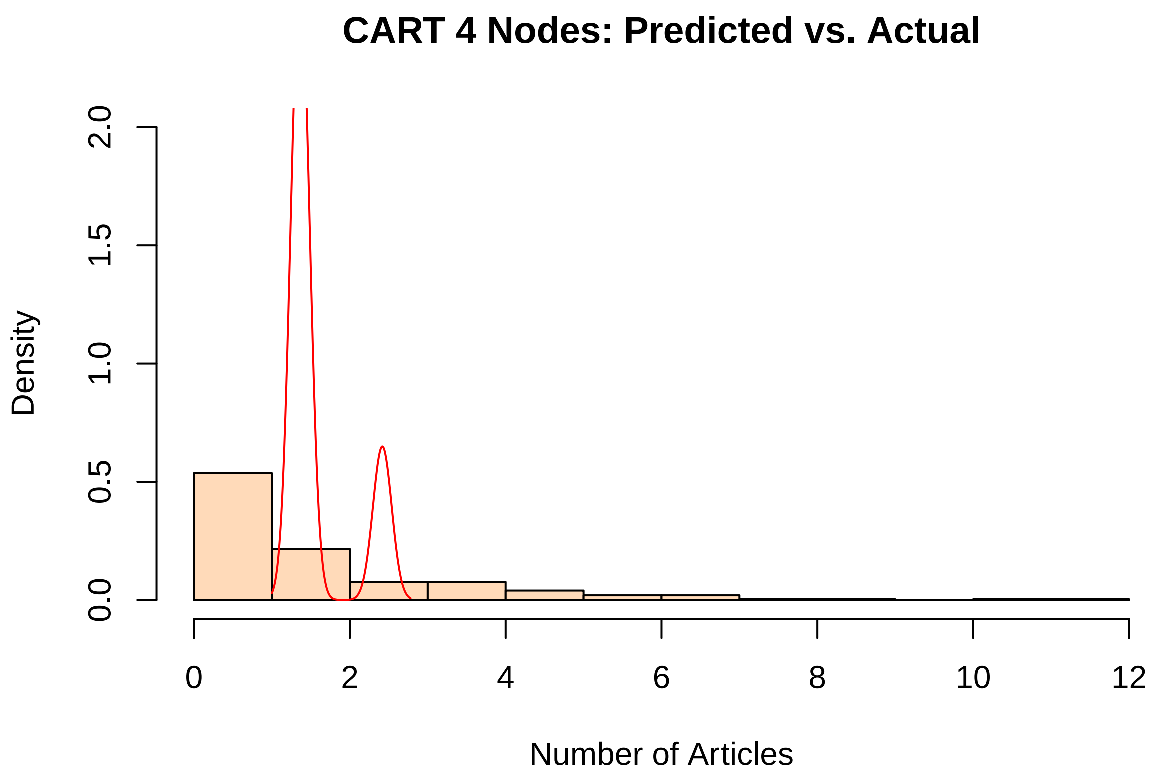


CART 4 Nodes TEST MSE

```
yHatMod4CARTTest<- predict(mod4,newdata = as.data.frame(xTest3))
MSEMod4Test<-mean((yTest -yHatMod4CARTTest)^2)
sqrt(MSEMod4Test)
```

```
## [1] 1.913087
```

```
hist(yTest,freq = F, ylim = c(0,2),col="peachpuff", main = "CART 4 Nodes: Predicted vs. Actual", xlab =
lines(density(yHatMod4CARTTest), col = "red")
```

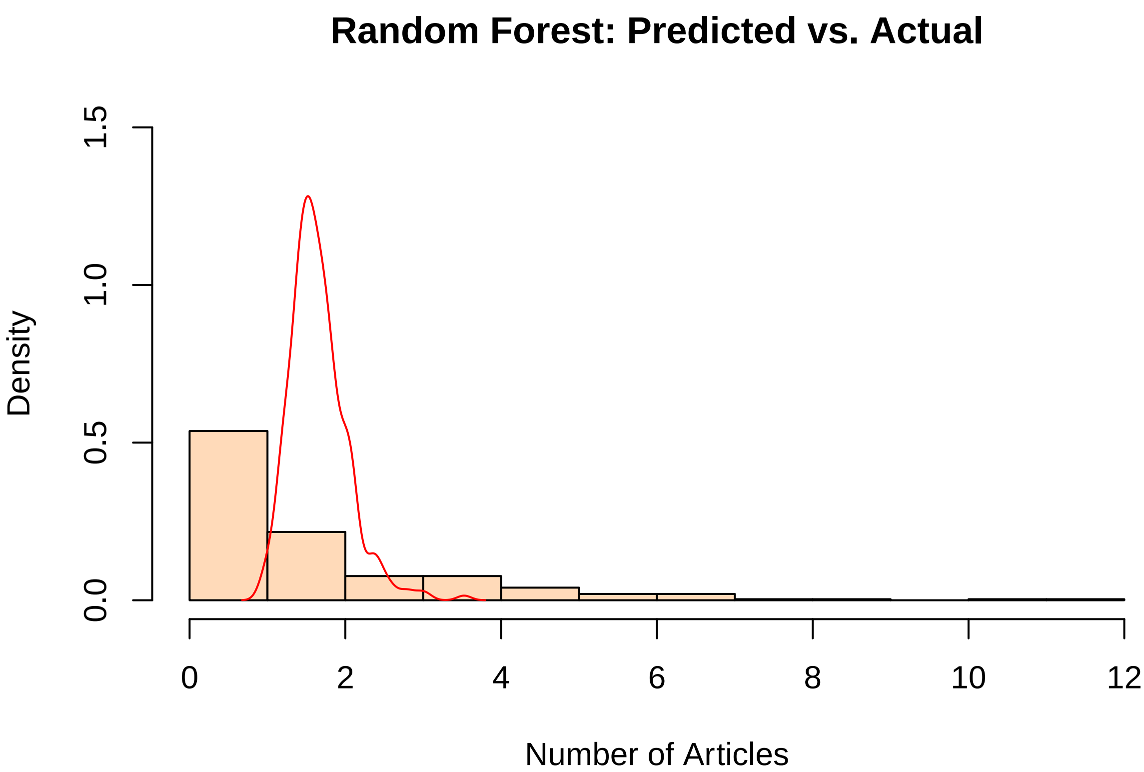


Random Forest TEST MSE

```
yHatRFTest<- predict(modRf,newdata = as.data.frame(xTest4))
MSERFTest<-mean((yTest -yHatRFTest)^2)
sqrt(MSERFTest)
```

```
## [1] 1.871652
```

```
hist(yTest,freq = F, ylim = c(0,1.5),col="peachpuff", main = "Random Forest: Predicted vs. Actual",, xla
lines(density(yHatRFTest), col = "red")
```



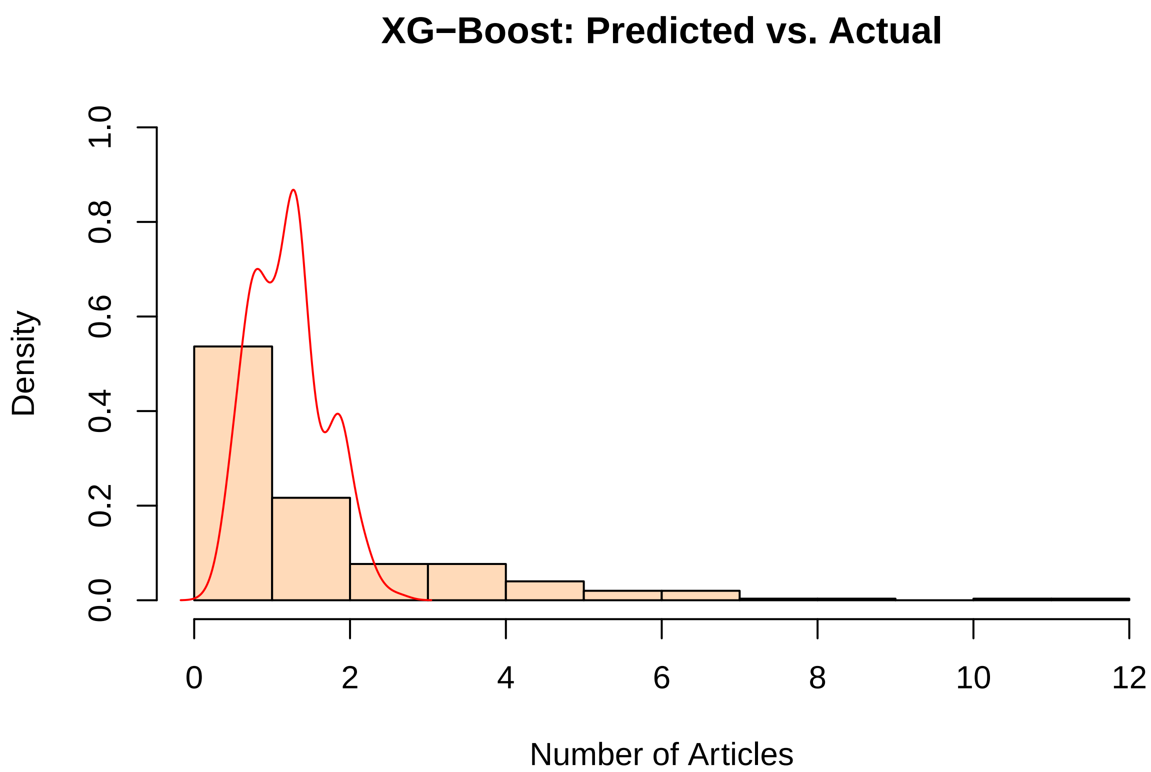
- Todo: tunning?

XGBoost TEST MSE

```
yHatXGTest<- predict(modXGB3,xTest5)
MSEXGTest<-mean((yTest -yHatXGTest)^2)
sqrt(MSEXGTest)
```

```
## [1] 1.985981
```

```
hist(yTest,freq = F, ylim = c(0,1),col="peachpuff", main = "XG-Boost: Predicted vs. Actual", xlab = "Nu
lines(density(yHatXGTest), col = "red")
```



- Todo: tunning?

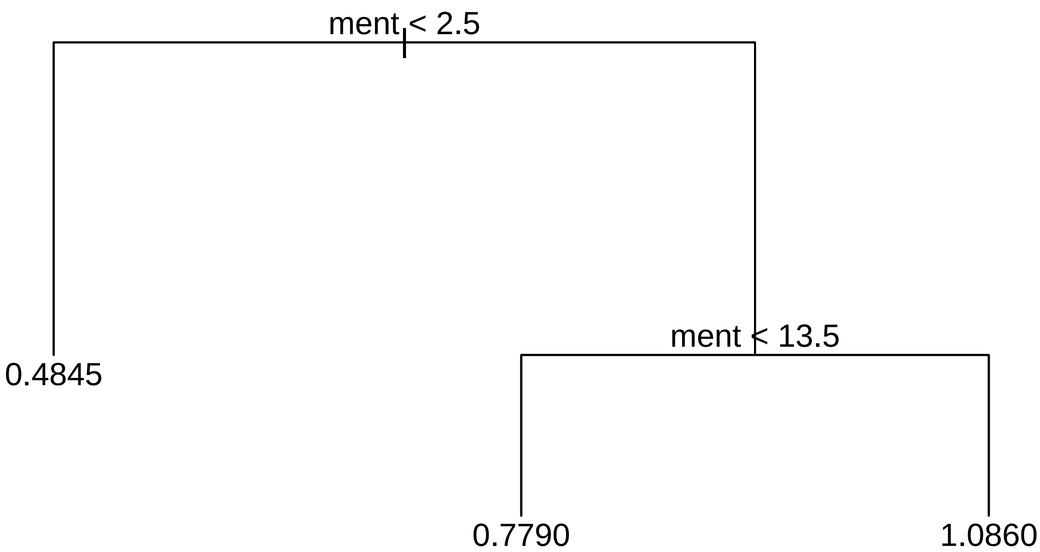
LLOOOGGG

LOOGGG!!!

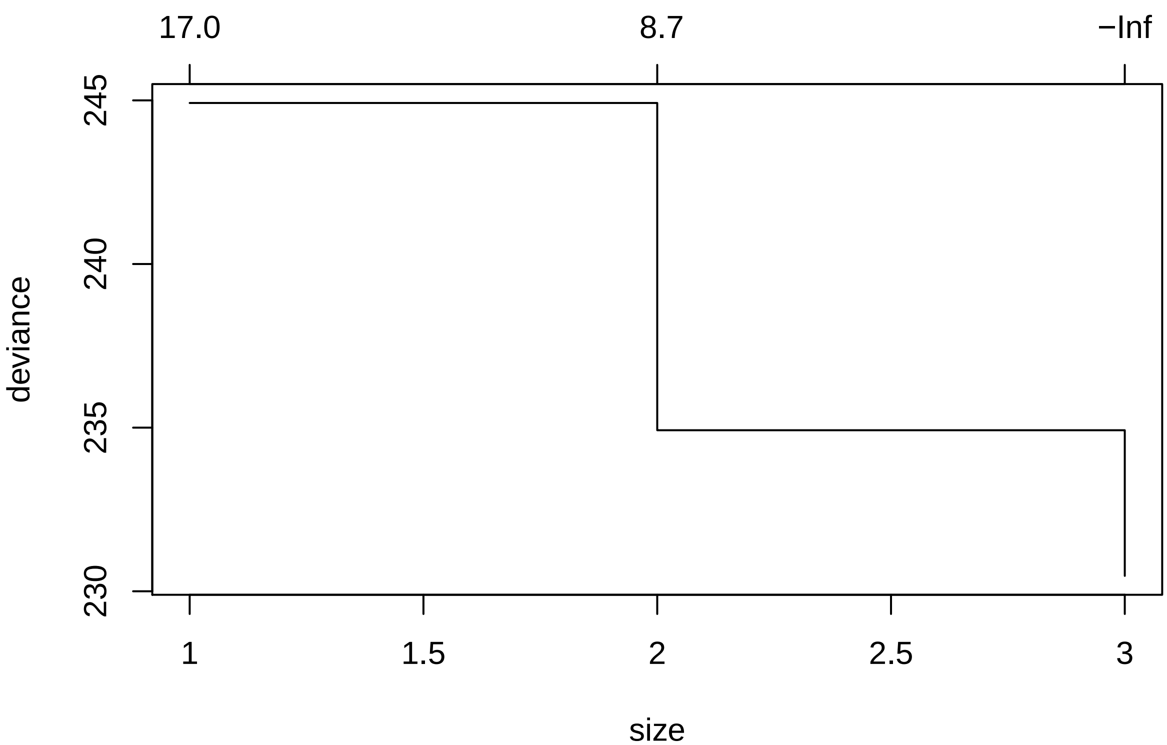
Part 3, Tree based methods

3.1, CART

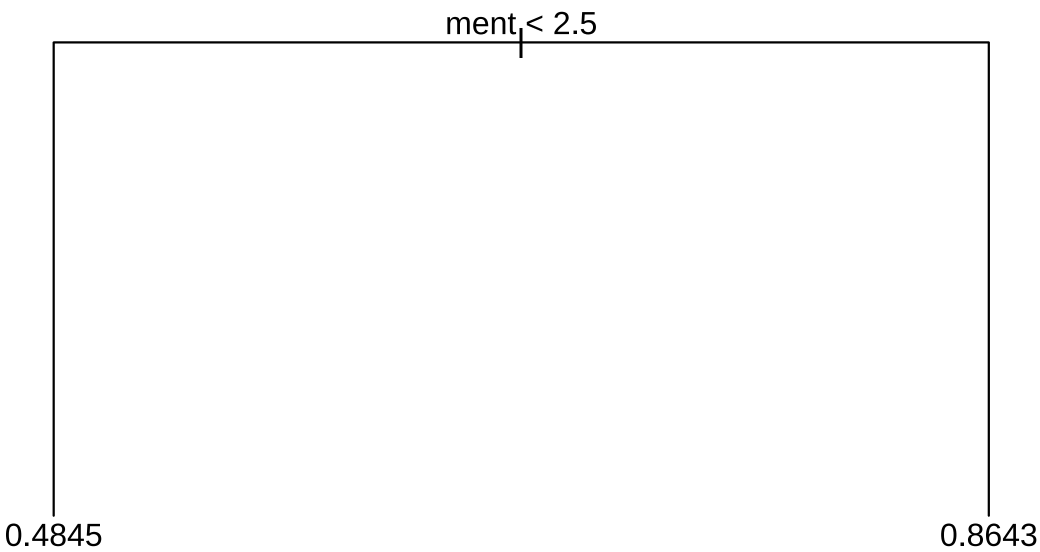
```
#Grow full tree
fullTree_2<-tree(art~.,data=dataTrain3)
#fullTree
#summary(fullTree) #Residual mean deviance:  3.08 = 1879 / 610, sum of squared error/(615-2)
plot(fullTree_2)
text(fullTree_2,pretty=0)
```




```r
#Run CV
cvTree_2<-cv.tree(fullTree_2)
#cvTree
plot(cvTree_2)
```



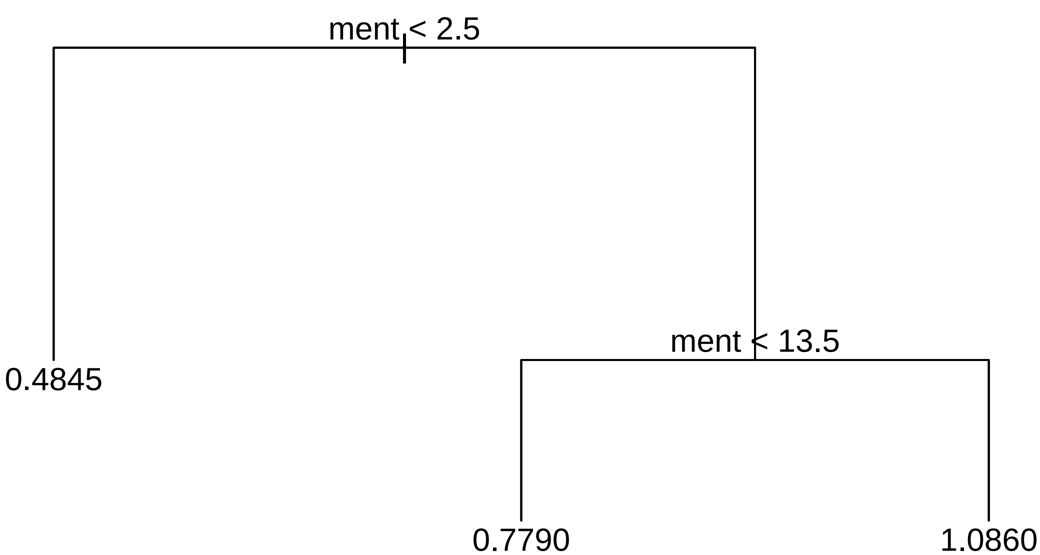

```r
#Use CV to prune tree
mod2_2<-prune.tree(fullTree_2,best=2)
plot(mod2_2)
text(mod2_2,pretty=0)
```




```r
summary(mod2_2) #Residual mean deviance:  3.371 = 2067 / 613
```

```
##
## Regression tree:
## snip.tree(tree = fullTree_2, nodes = 3L)
## Variables actually used in tree construction:
## [1] "ment"
## Number of terminal nodes:  2
## Residual mean deviance:  0.3608 = 221.2 / 613
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.8643 -0.4845 -0.1711  0.0000  0.5220  2.1310
```

```r
MSEMod2_2<-mean((dataTrain3$art-predict(mod2_2))^2)

mod3_2<-prune.tree(fullTree_2,best=3)
plot(mod3_2)
text(mod3_2,pretty=0)
```



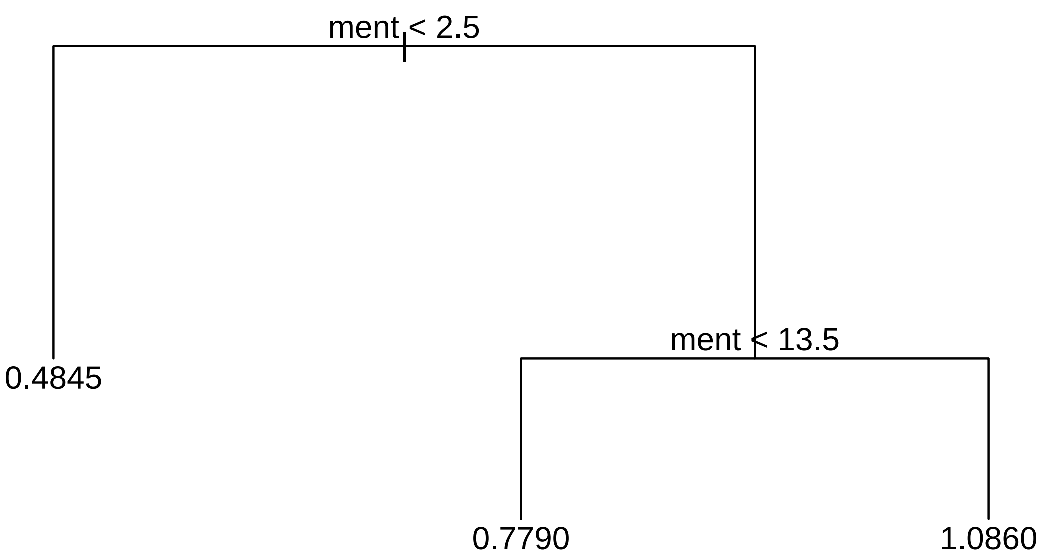
```r
summary(mod3_2) #Residual mean deviance:  3.243 = 1984 / 612
```

```
##
## Regression tree:
## tree(formula = art ~ ., data = dataTrain3)
## Variables actually used in tree construction:
## [1] "ment"
## Number of terminal nodes:  3
## Residual mean deviance:  0.3472 = 212.5 / 612
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.08600 -0.48450 -0.08589  0.00000  0.31960  1.90900
```

```r
MSEMod3_2<-mean((dataTrain3$art-predict(mod3_2))^2) #3.22672

mod4_2<-prune.tree(fullTree_2,best=4)
```

```
## Warning in prune.tree(fullTree_2, best = 4): best is bigger than tree size
```

```r
plot(mod4_2)
text(mod4_2,pretty=0)
```




```r
summary(mod4_2) #Residual mean deviance:  3.151 = 1925 / 611
```

```
##
## Regression tree:
## tree(formula = art ~ ., data = dataTrain3)
## Variables actually used in tree construction:
## [1] "ment"
## Number of terminal nodes:  3
## Residual mean deviance:  0.3472 = 212.5 / 612
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.08600 -0.48450 -0.08589  0.00000  0.31960  1.90900
```

```r
MSEMod4_2<-mean((dataTrain3$art-predict(mod4_2))^2)
```

Even the full tree only considers “mentor” (number of publications by the PhD student’s mentor) and “PhD” (the prestige of the student’s PhD program). So we do not expect this method to work well.

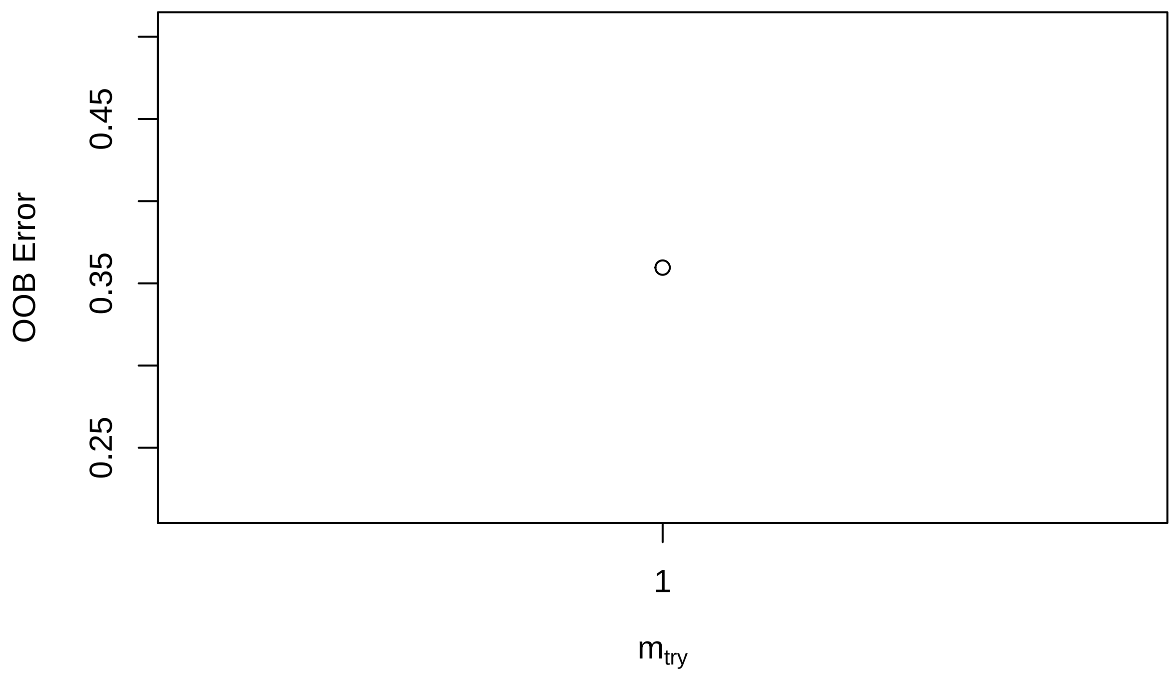
3.2, Random forest

```
#Random forest with default mtry
modRf_2<-randomForest(art~.,mtry=1,data=dataTrain4) #The default is mtry=p/3=1.
modRf_2
```

```
##
## Call:
##  randomForest(formula = art ~ ., data = dataTrain4, mtry = 1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.3554426
##                     % Var explained: 8.16
```

```
MSERf_2<-mean((dataTrain3$art-predict(modRf_2))^2) #3.34406

#Tune mtry
modRfBestMtry_2<-tuneRF(xTrain,yTrain2,stepFactor=1,improve=0)
```

```
## mtry = 1  OOB error = 0.3595766
## Searching left ...
## Searching right ...
```




```
print(modRfBestMtry_2) #mtry=1 is the best based on out of bag error
```

```
##   mtry  OOBError
## 1    1 0.3595766
```

3.3, XG boost

```
#Default XG boost
modXGB_2<-xgboost(data=xTrain,label=yTrain2,nrounds=2)
```

```
## [1]  train-rmse:0.605801
## [2]  train-rmse:0.556277
```

```
#Tune nround with the default parameters
params_2<-list(
        booster="gbtree",
        objective="reg:linear",
        eta=0.3,
        gamma=0,
        max_depth=6,
        min_child_weight=1,
        subsample=1,
        colsample_bytree=1
)

xgbCV_2<-xgb.cv(params=params_2,
              data = xTrain,
              label= yTrain2,
              nrounds=100,
              nfold=5,
              showsd=T,
              stratified=T,
              print_every_n=10,
              early_stop_round=20,
              maximize=F
)
```

```
## [1]  train-rmse:0.602187+0.009392    test-rmse:0.636846+0.042199
## [11] train-rmse:0.409399+0.014321    test-rmse:0.623970+0.039624
## [21] train-rmse:0.350934+0.014596    test-rmse:0.640372+0.045783
## [31] train-rmse:0.299993+0.010466    test-rmse:0.654282+0.048508
## [41] train-rmse:0.266967+0.010914    test-rmse:0.664216+0.047440
## [51] train-rmse:0.239271+0.008474    test-rmse:0.675552+0.045385
## [61] train-rmse:0.216417+0.007761    test-rmse:0.687830+0.050901
## [71] train-rmse:0.201253+0.007343    test-rmse:0.695008+0.050451
## [81] train-rmse:0.189680+0.008364    test-rmse:0.700830+0.050534
## [91] train-rmse:0.177858+0.008744    test-rmse:0.707624+0.050298
## [100]    train-rmse:0.171055+0.009271    test-rmse:0.712814+0.049319
```

```
minTestRMSE_2=min(xgbCV_2$evaluation_log$test_rmse_mean)
minTestRMSEIndex_2=which.min(xgbCV_2$evaluation_log$test_rmse_mean)

#Alternative method to tune nrounds
modXGB.CV_2<-xgb.cv(data=xTrain,label=yTrain2,nfold=5,nrounds=20)
```

```
## [1]  train-rmse:0.604106+0.006849    test-rmse:0.632180+0.030039
## [2]  train-rmse:0.555775+0.006262    test-rmse:0.619271+0.026049
## [3]  train-rmse:0.524129+0.008882    test-rmse:0.611526+0.022714
## [4]  train-rmse:0.497365+0.010638    test-rmse:0.609532+0.023101
## [5]  train-rmse:0.476681+0.010015    test-rmse:0.610875+0.024075
```

```
## [6]  train-rmse:0.460206+0.009044    test-rmse:0.615567+0.023559
## [7]  train-rmse:0.443949+0.005644    test-rmse:0.618917+0.025151
## [8]  train-rmse:0.432922+0.006406    test-rmse:0.623172+0.023104
## [9]  train-rmse:0.422838+0.007822    test-rmse:0.623158+0.024414
## [10] train-rmse:0.412611+0.008382    test-rmse:0.624326+0.025597
## [11] train-rmse:0.406799+0.008122    test-rmse:0.625755+0.024497
## [12] train-rmse:0.398750+0.009564    test-rmse:0.630098+0.026817
## [13] train-rmse:0.393773+0.009772    test-rmse:0.631825+0.025758
## [14] train-rmse:0.384937+0.006361    test-rmse:0.632957+0.025555
## [15] train-rmse:0.380014+0.008009    test-rmse:0.633602+0.025947
## [16] train-rmse:0.373763+0.010122    test-rmse:0.636993+0.028445
## [17] train-rmse:0.366460+0.008864    test-rmse:0.639824+0.028865
## [18] train-rmse:0.361214+0.009849    test-rmse:0.641063+0.029903
## [19] train-rmse:0.358086+0.010096    test-rmse:0.642051+0.030163
## [20] train-rmse:0.350893+0.008854    test-rmse:0.643491+0.030546
```

```r
minTestRMSE2_2=min(modXGB.CV_2$evaluation_log$test_rmse_mean)
minTestRMSEIndex2_2=which.min(modXGB.CV_2$evaluation_log$test_rmse_mean) #nrounds=3 is the best

modXGB3_2<-xgboost(data=xTrain,label=yTrain2,nrounds=3)
```

```
## [1]  train-rmse:0.605801
## [2]  train-rmse:0.556277
## [3]  train-rmse:0.522655
```

```r
MSEXGB_2<-mean((dataTrain3$art-predict(modXGB3_2,xTrain))^2) #2.43918
```

```r
#Tune the parameters using the MLR package
traintask_2<-makeRegrTask(data=dataTrain3,target="art")
testtask_2<-makeRegrTask(data=dataTest2,target="art")
```

To do:

We will think about how to compare the tree based methods/KRLS to the GLM methods from Part 1 and ridge and LASSO from Part 2.

TESTING RESULTS:

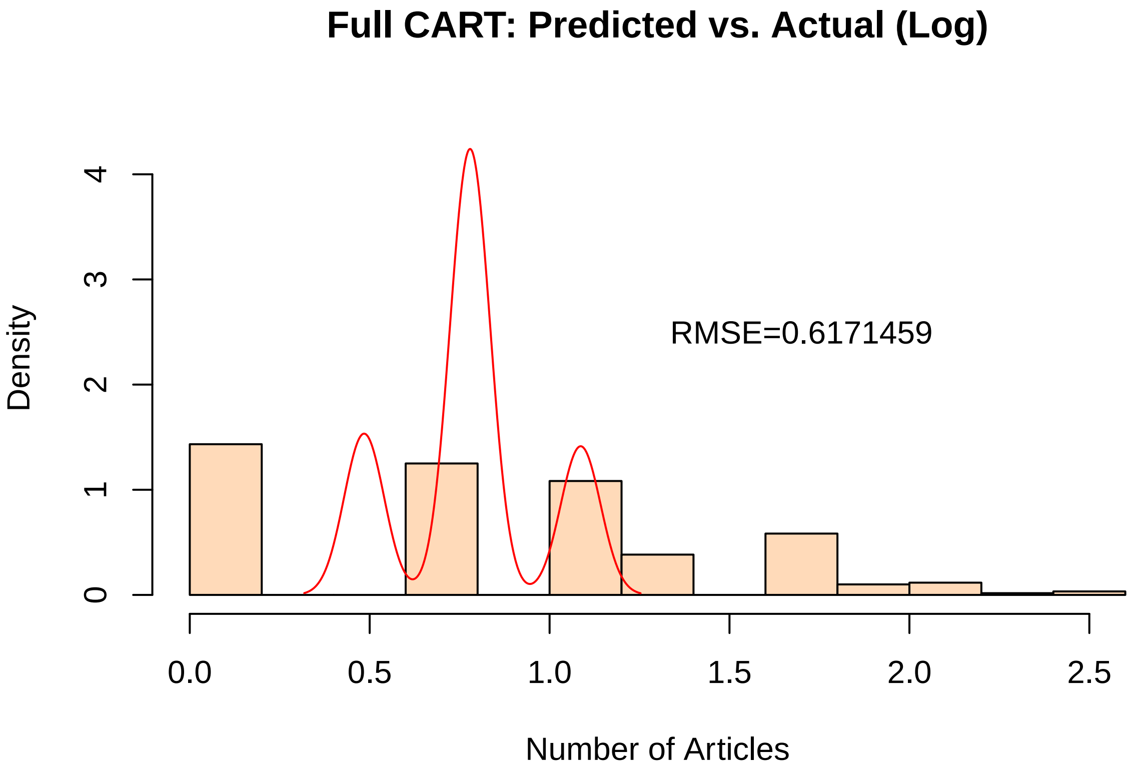
```r
yTest2_2 <- dataTest2[,1]
xTest3names_2 <-names(as.data.frame(dataTest2[,-1]))
xTest5_2 <- model.matrix(art~.,data=dataTest2)[,-1]
xTest3_2 <- as.data.frame(xTest5_2)
names(xTest3_2) <- xTest3names_2
xTest4_2 <-dataTest[,-1]
```

FULL CART TEST MSE

```r
yHatFULLCARTTest_2<- predict(fullTree_2,newdata = as.data.frame(xTest3_2))
MSEFullTest_2<-mean((yTest2_2 -yHatFULLCARTTest_2)^2)
sqrt(MSEFullTest_2)
```

```
## [1] 0.6171459
```

```r
hist(yTest2_2,freq = F, ylim = c(0,4.5),col="peachpuff", main = "Full CART: Predicted vs. Actual (Log)"
lines(density(yHatFULLCARTTest_2), col = "red")
text(1.7, y = 2.5, labels = "RMSE=0.6171459")
```

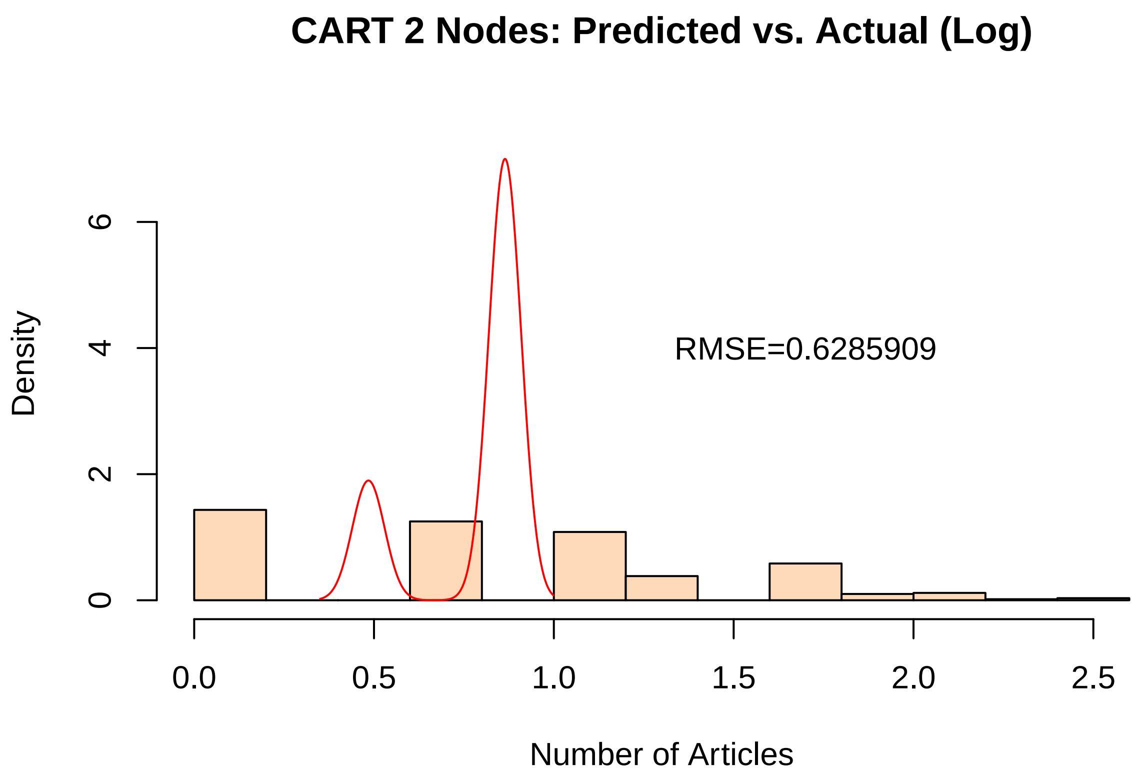


CART 2 Nodes TEST MSE

```r
yHatMod2CARTTest_2<- predict(mod2_2,newdata = as.data.frame(xTest3_2))
MSEMod2Test_2<-mean((yTest2_2 -yHatMod2CARTTest_2)^2)
sqrt(MSEMod2Test_2)
```

```
## [1] 0.6285909
```

```r
hist(yTest2_2,freq = F, ylim = c(0,7.5),col="peachpuff", main = "CART 2 Nodes: Predicted vs. Actual (Lo
lines(density(yHatMod2CARTTest_2), col = "red")
text(1.7, y = 4, labels = "RMSE=0.6285909")
```

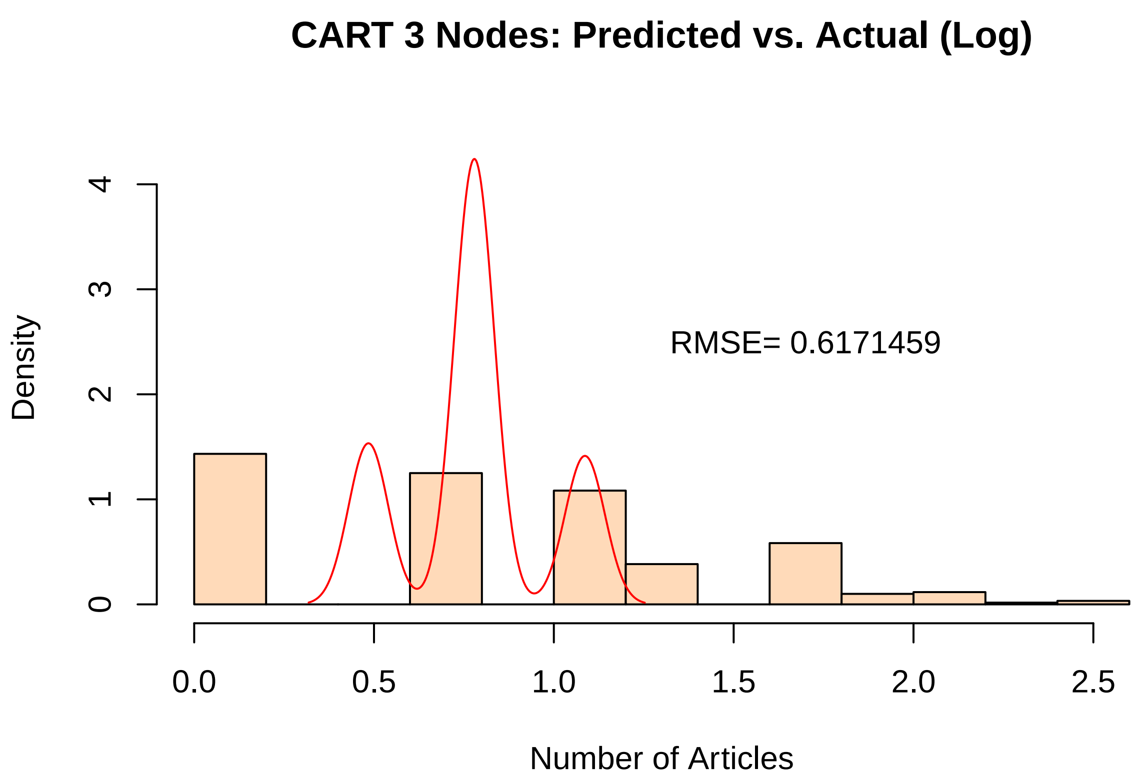


CART 3 Nodes TEST MSE

```
yHatMod3CARTTest_2<- predict(mod3_2,newdata = as.data.frame(xTest3_2))
MSEMod3Test_2<-mean((yTest2_2 -yHatMod3CARTTest_2)^2)
sqrt(MSEMod3Test_2)
```

```
## [1] 0.6171459
```

```
hist(yTest2_2,freq = F, ylim = c(0,4.5),col="peachpuff", main = "CART 3 Nodes: Predicted vs. Actual (Log
lines(density(yHatMod3CARTTest_2), col = "red")
text(1.7, y = 2.5, labels = "RMSE= 0.6171459")
```

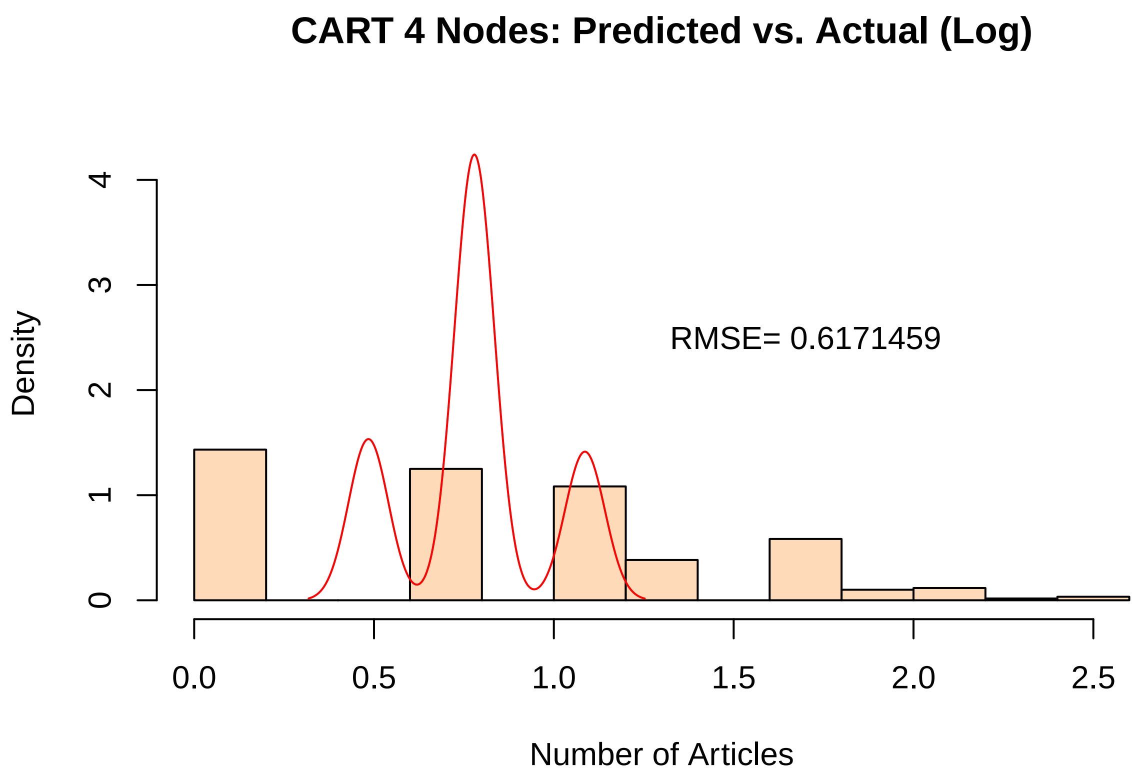


CART 4 Nodes TEST MSE

```
yHatMod4CARTTest_2<- predict(mod4_2,newdata = as.data.frame(xTest3_2))
MSEMod4Test_2<-mean((yTest2_2 -yHatMod4CARTTest_2)^2)
sqrt(MSEMod4Test_2)
```

```
## [1] 0.6171459
```

```
hist(yTest2_2,freq = F, ylim = c(0,4.5),col="peachpuff", main = "CART 4 Nodes: Predicted vs. Actual (Log
lines(density(yHatMod4CARTTest_2), col = "red")
text(1.7, y = 2.5, labels = "RMSE= 0.6171459")
```

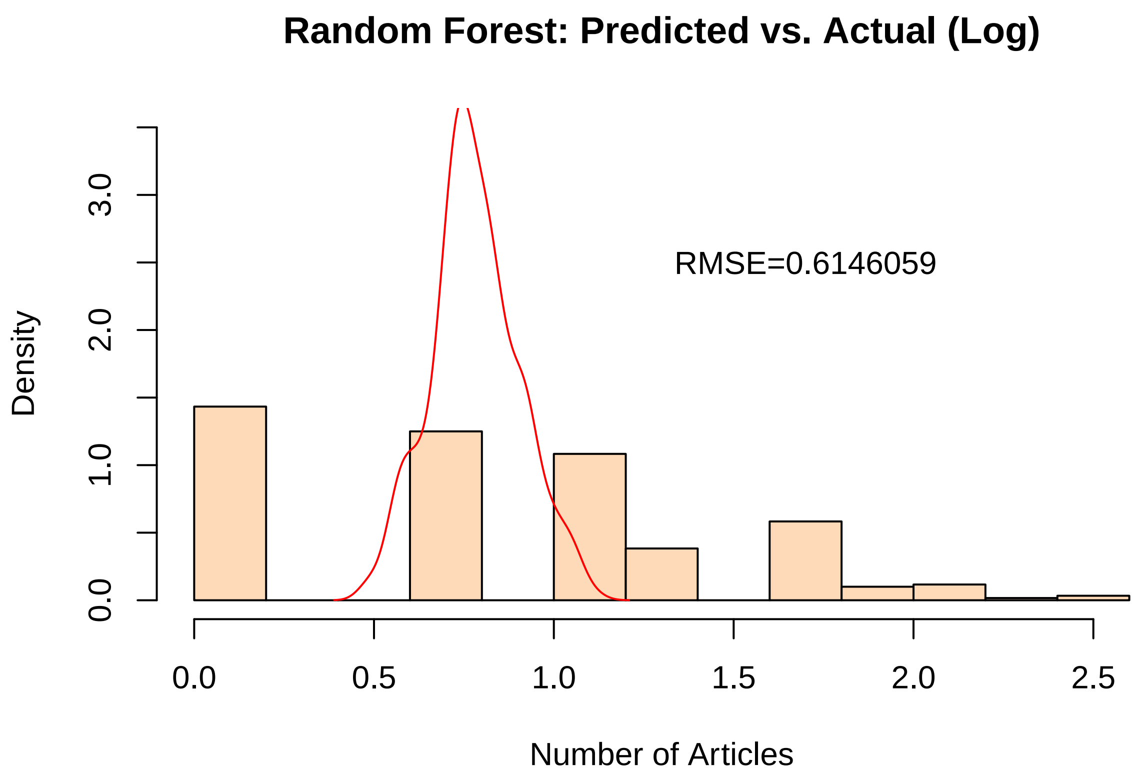


Random Forest TEST MSE

```
yHatRFTest_2<- predict(modRf_2,newdata = as.data.frame(xTest4_2))
MSERFTest_2<-mean((yTest2_2 -yHatRFTest_2)^2)
sqrt(MSERFTest_2)
```

```
## [1] 0.6158952
```

```
hist(yTest2_2,freq = F, ylim = c(0,3.5),col="peachpuff", main = "Random Forest: Predicted vs. Actual (Lo
lines(density(yHatRFTest_2), col = "red")
text(1.7, y = 2.5, labels = "RMSE=0.6146059")
```



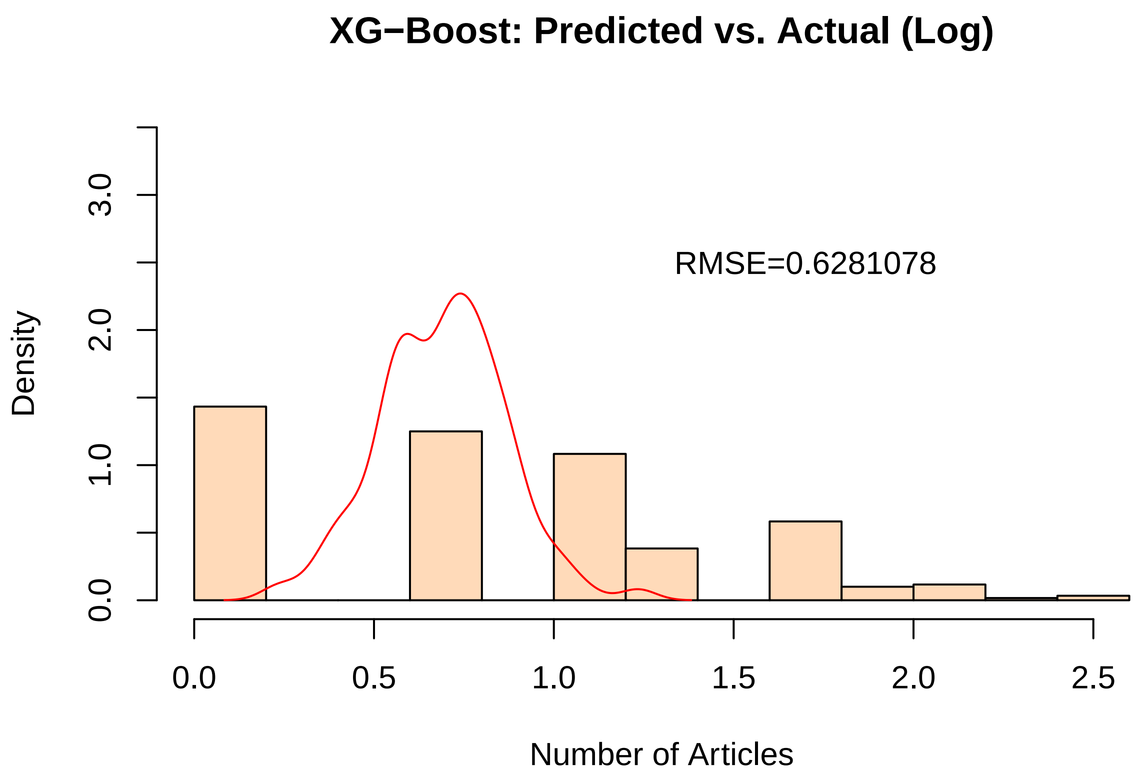
- Todo: tunning?

XGBoost TEST MSE

```
yHatXGTest_2<- predict(modXGB3_2,xTest5_2)
MSEXGTest_2<-mean((yTest2_2 -yHatXGTest_2)^2)
sqrt(MSEXGTest_2)
```

```
## [1] 0.6281078
```

```
hist(yTest2_2,freq = F, ylim = c(0,3.5),col="peachpuff", main = "XG-Boost: Predicted vs. Actual (Log)",
lines(density(yHatXGTest_2), col = "red")
text(1.7, y = 2.5, labels = "RMSE=0.6281078")
```



- Todo: tunning?