# Stats 503 Homework 4

*Xingwen Wei*

*March 23, 2021*

## Q1

By the definition of conditional expectation,

$$\begin{aligned} f^*(x) &= argmin_{f(x)} E_{Y|x}(e^{-Yf(x)}) \\ &= argmin_{f(x)} e^{-f(x)} P(Y=1|x) + e^{f(x)} P(Y=-1|x) \end{aligned}$$

We can find argmin f(x) by seeting the derivative of above equation to zero.

$$\begin{aligned} \frac{d}{df} e^{-f(x)} P(Y=1|x) + e^{f(x)} P(Y=-1|x) &= 0 \\ -e^{-f(x)} P(Y=1|x) + e^{f(x)} P(Y=-1|x) &= 0 \\ -P(Y=1|x) + e^{2f(x)} P(Y=-1|x) &= 0 \\ e^{2f(x)} &= \frac{P(Y=1|x)}{P(Y=-1|x)} \\ f(x) &= \frac{1}{2} ln(\frac{P(Y=1|x)}{P(Y=-1|x)}) \end{aligned}$$

Thus, we find that the minimizer of population exponential loss is one-half of the log odds.

## Q2

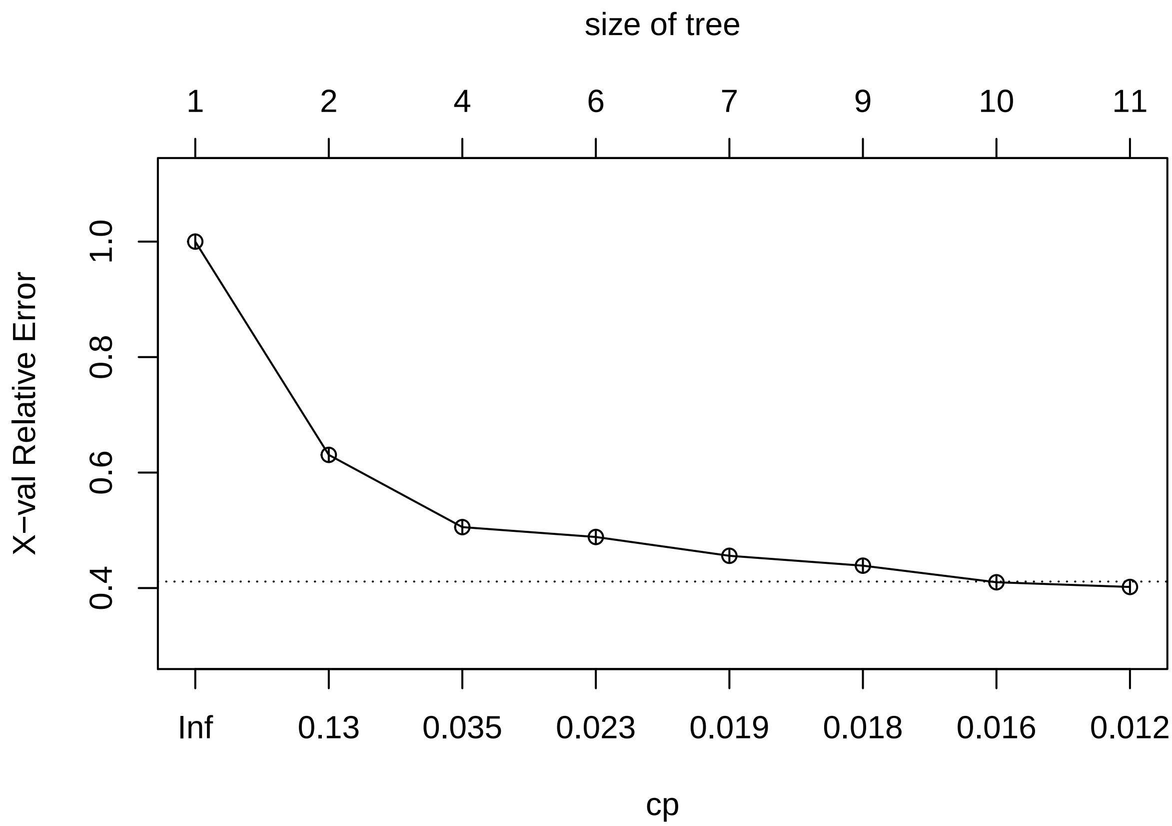
### a

We use cross validation to find optimal cp to build a tree.

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.6.3
```

```
# Read in data
train = read.csv("C:/Users/xingw/Desktop/503/stats503/hw4/bank_marketing_train.csv")
tree1_cv = rpart(deposit ~ ., data=train, parms=list(split='gini'), method='class')
plotcp(tree1_cv)
```




The summary of test result is as following.

```
tree1 = rpart(deposit ~ ., data=train, parms=list(split='gini'), method='class', cp=0.016)
test = read.csv("C:/Users/xingw/Desktop/503/stats503/hw4/bank_marketing_test.csv")
tree1_pred = predict(tree1, test, type='class')
t = table(tree1_pred, test$deposit)
t
```

```
##
## tree1_pred   no  yes
##        no  1295  160
##        yes  450 1444
```

```
test_stats <- function(t){
  tm = (t[2, 1] + t[1, 2])/nrow(test)
  nm = t[2, 1] / (t[1, 1] + t[2, 1])
  ym = t[1, 2] / (t[1, 2] + t[2, 2])
  tree_m = matrix(c(tm, nm, ym), ncol=3, byrow=TRUE)
  colnames(tree_m) = c('Total Misclassification', 'No Misclassification', 'Yes Misclassification')
  rownames(tree_m) = 'Percentage'
  tree_t = as.table(tree_m)
  tree_t
}
```

```
test_stats(t)
```

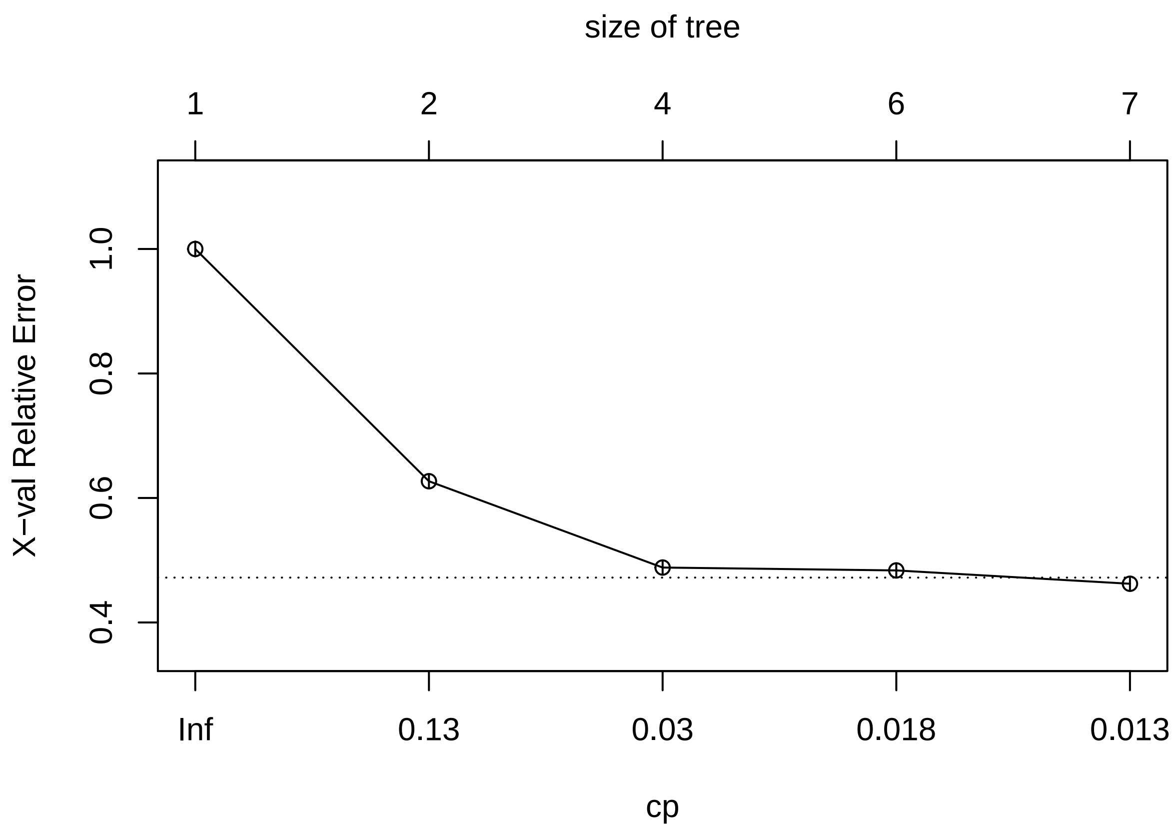
```
##            Total Misclassification No Misclassification
## Percentage              0.18214392           0.25787966
```

```
##            Yes Misclassification
## Percentage            0.09975062
```

## b

We limit the maximum terminal nodes = 8 by constraining on the depth = 3. And use cross validation to find optimal cp.

```
tree2_cv = rpart(deposit ~ ., data=train, parms=list(split='gini'), method='class', control=list(maxdep
plotcp(tree2_cv)
```

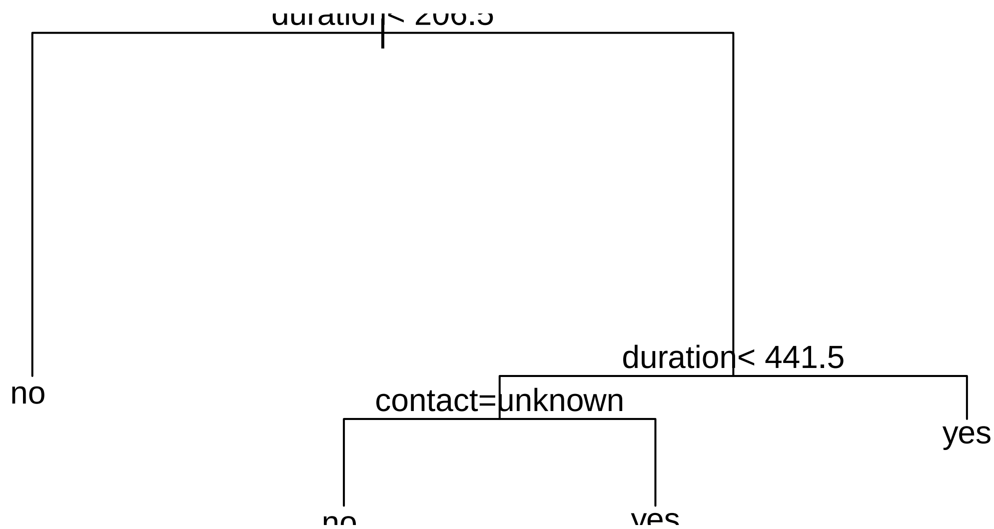



The variables used are ‘duration’ and ‘contact’.

```
tree2_cv = rpart(deposit ~ ., data=train, parms=list(split='gini'), method='class', control=list(maxdep
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.6.3
```

```
plot(tree2_cv)
text(tree2_cv, pretty=0)
```




#c

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

```
## randomForest 4.6-14
```

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

```r
rf = randomForest(deposit ~ ., data=train, importance=TRUE, ntree=500)
```

The test summary is as following.

```r
rf_pred = predict(rf, test)
t = table(rf_pred, test$deposit)
t
```

```
##
## rf_pred   no  yes
##     no  1426  170
##     yes  319 1434
```

```r
test_stats(t)
```

```
##            Total Misclassification No Misclassification
## Percentage               0.1460137            0.1828080
##            Yes Misclassification
## Percentage             0.1059850
```

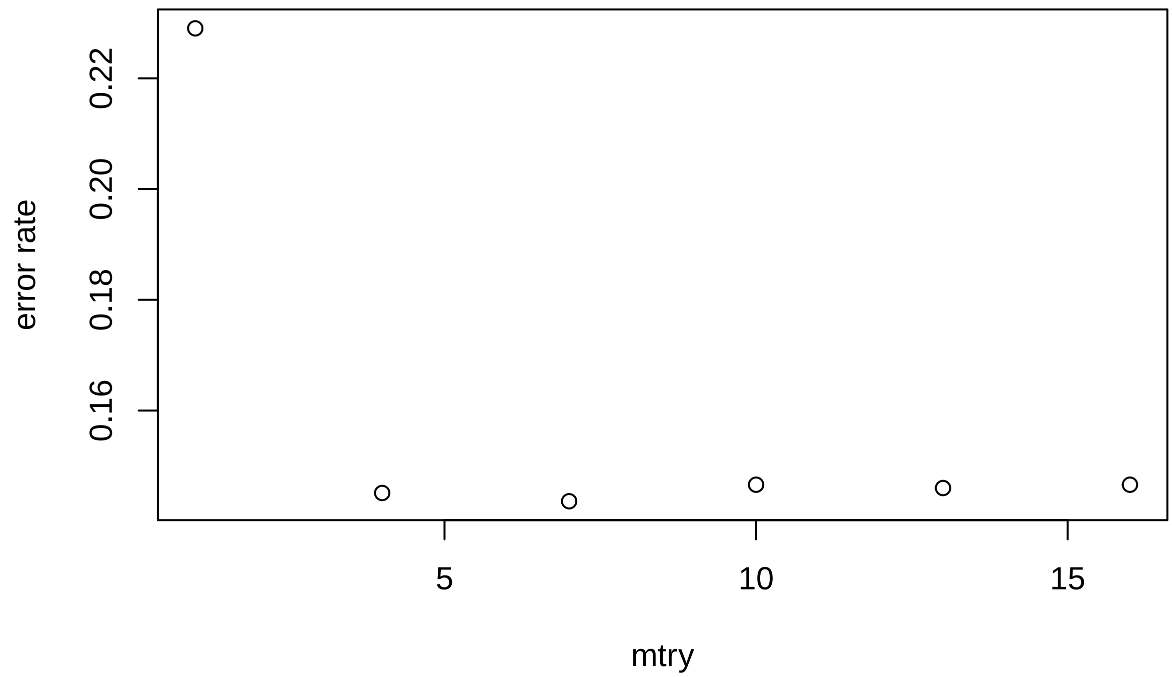
We can use the variable-importance to interpret their effectiveness in the tree. We find 'duration' is the most important variable.

```
importance(rf)
```

```
##                    no         yes MeanDecreaseAccuracy MeanDecreaseGini
## age        32.0461320  17.9414777           36.5823523       265.171237
## job        25.4410564   4.1179847           21.1896793       225.632472
## marital     4.9632489  12.2266116           12.3871933        62.123996
## education  14.7899008   3.5574261           13.9305097        79.211295
## default    -0.7708862  -0.1028719           -0.4724245         4.139095
## balance    13.4677535   9.6957039           16.3629675       286.146621
## housing    36.0066103  29.2058379           43.3888101       107.273552
## loan        3.1947622  11.5233714           11.0177009        28.872358
## contact    46.7283264  18.7933215           50.8543172       133.464686
## day        48.7492436   9.8469179           46.6453263       257.635368
## month      99.6555449  42.8992700          113.1710462       501.334540
## duration  188.3606712 242.8684626          266.6776122      1338.181515
## campaign   11.2635186  15.9854442           19.8920487       110.344457
## pdays      22.4197280  19.6503147           29.3815541       139.958432
## previous   17.0776866  11.5327041           17.9515681        92.140042
## poutcome   53.3066648   7.5181712           42.3389462       203.686435
```

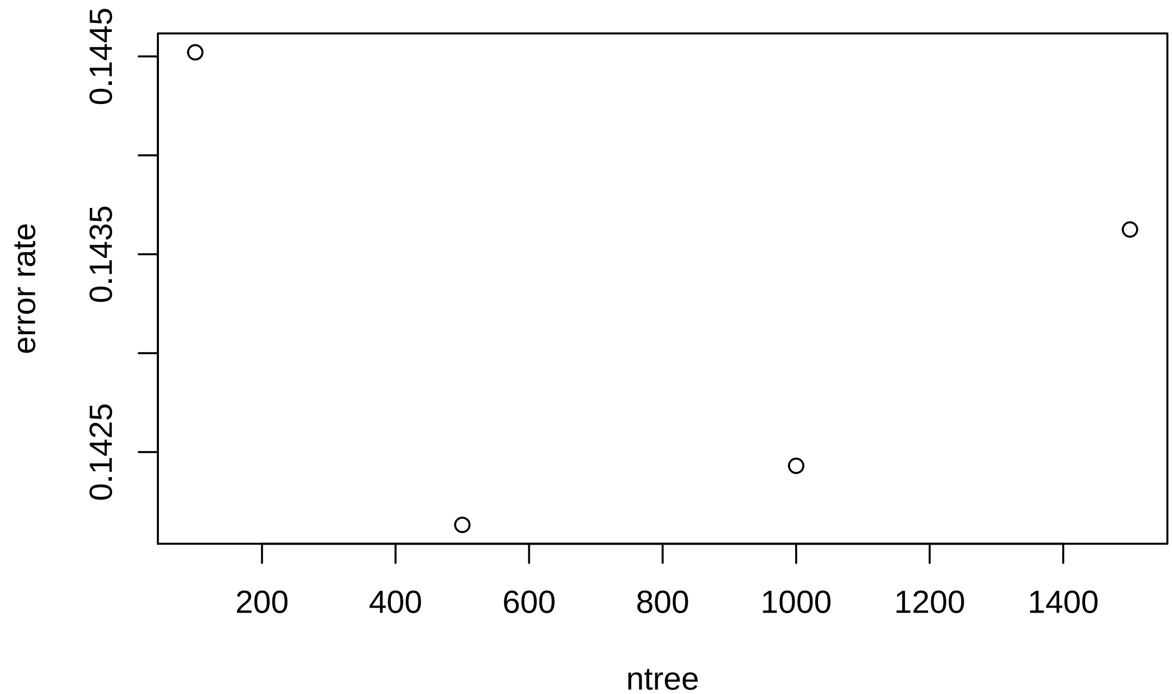
As long as mtry is greater than four, the error rate is about the same.

```
mt = c(1, 4, 7, 10, 13, 16)
error = rep(NA, 6)
for(i in 1:6){
  rf_t = randomForest(deposit ~ ., data=train, importance=TRUE, ntree=500, mtry=mt[i])
  rf_t_pred = predict(rf_t, test, type='class')
  error[i] = mean(rf_t_pred!=test$deposit)
}
plot(x=mt, y=error, xlab='mtry', ylab='error rate')
```



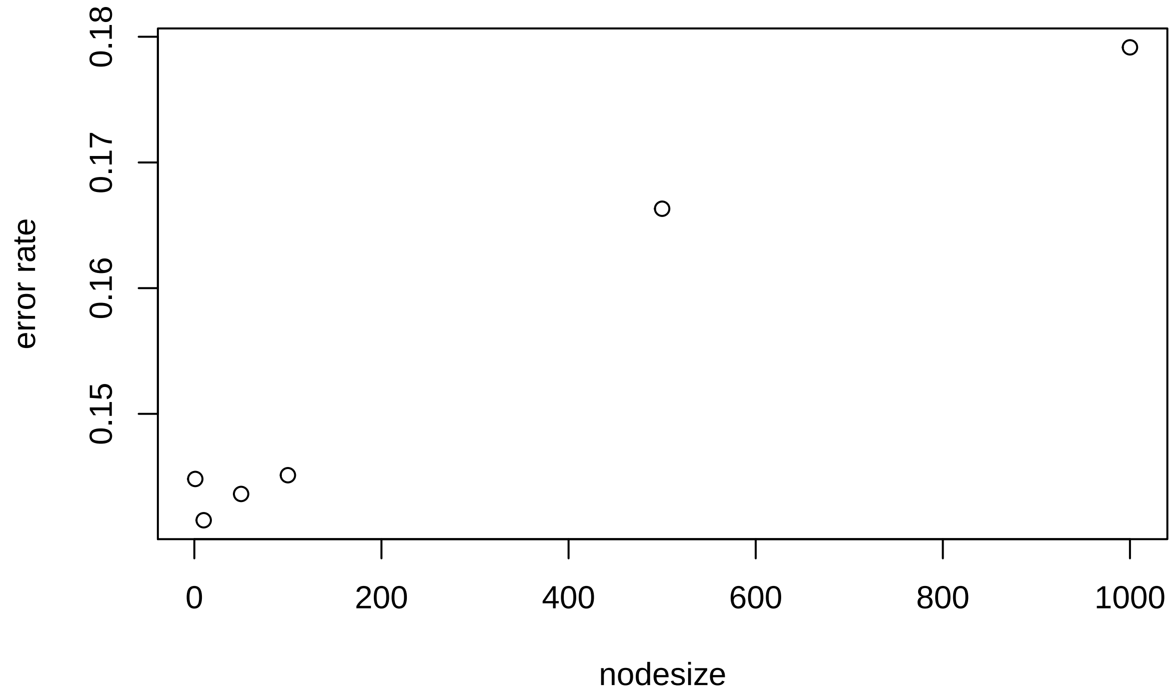

We find that the test error is smallest when ntree is around 1000. Although the error rate is higher when ntree is 1500, we believe this is caused by the realization of this particular test sample. We believe in general, the more trees the better if computation power is not a concern.

```
nt = c(100, 500, 1000, 1500)
error = rep(NA, 4)
for(i in 1:4){
  rf_t = randomForest(deposit ~ ., data=train, importance=TRUE, ntree=nt[i])
  rf_t_pred = predict(rf_t, test, type='class')
  error[i] = mean(rf_t_pred!=test$deposit)
}
plot(x=nt, y=error, xlab='ntree', ylab='error rate')
```




We find the test error is smallest when nodesize is around 50. We believe this is a paired tuning parameter with ntree. The smaller nodesize is, the less the bias becomes for each subtree. While the larger ntree is, the less the variance becomes for the forest. This plot indicates that we will receive a smallest test error with nodesize 50 when we have 500 trees in this particular realization of sample.

```
ns = c(1, 10, 50, 100, 500, 1000)
error = rep(NA, 6)
for(i in 1:6){
  rf_t = randomForest(deposit ~ ., data=train, importance=TRUE, ntree=500, nodesize=ns[i])
  rf_t_pred = predict(rf_t, test, type='class')
  error[i] = mean(rf_t_pred!=test$deposit)
}
plot(x=ns, y=error, xlab='nodesize', ylab='error rate')
```



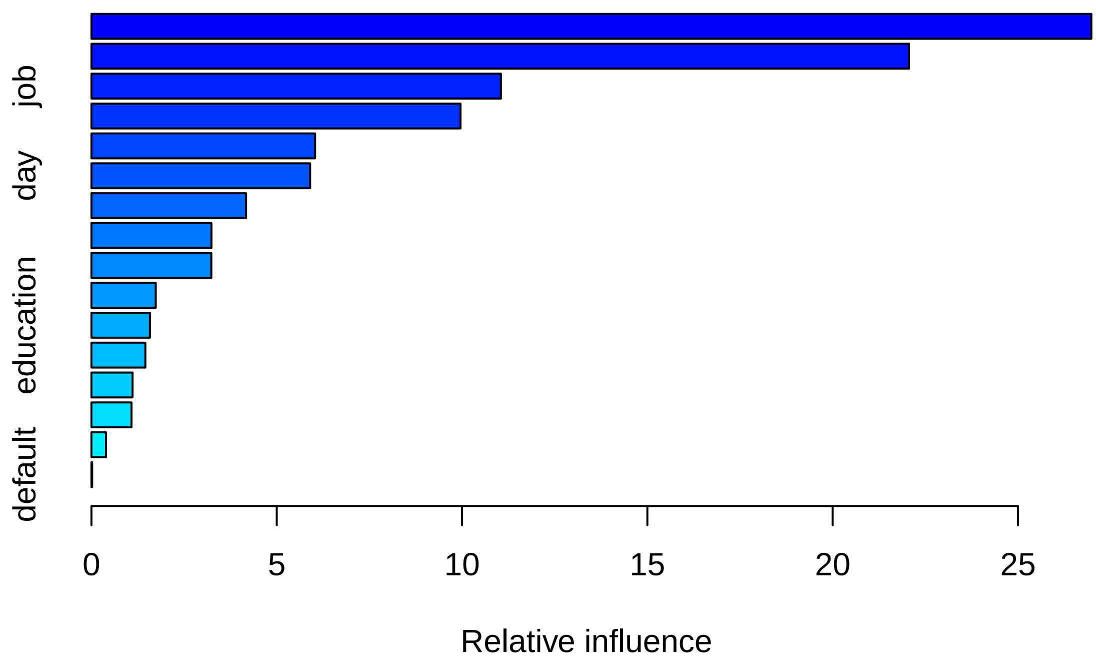

#d

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.3
```

```
## Loaded gbm 2.1.8
```

```
train$deposit = ifelse(train$deposit == 'yes', 1, 0)
test$deposit = ifelse(test$deposit == 'yes', 1, 0)
boost = gbm(deposit ~ ., data=train, distribution='adaboost', n.trees=5000, interaction.depth = 3, shrin
summary(boost)
```




```
##                   var     rel.inf
## duration     duration 26.97937387
## month           month 22.05963071
## job               job 11.04822292
## balance       balance  9.95822706
## age               age  6.03520105
## day               day  5.90141266
## poutcome     poutcome  4.17248839
## contact       contact  3.23753064
## pdays           pdays  3.23492245
## housing       housing  1.73571726
## education education  1.57879605
## campaign     campaign  1.45464155
## marital       marital  1.10963113
## previous     previous  1.08142416
## loan             loan  0.39367033
## default       default  0.01910977
```

The test summary is as following.

```
boost_pred_response = predict(boost, test, n.trees=5000, type='response')
boost_pred = ifelse(boost_pred_response>0.5, 1, 0)
t = table(boost_pred, test$deposit)
t
```

```
##
## boost_pred    0    1
##          0 1477  231
```

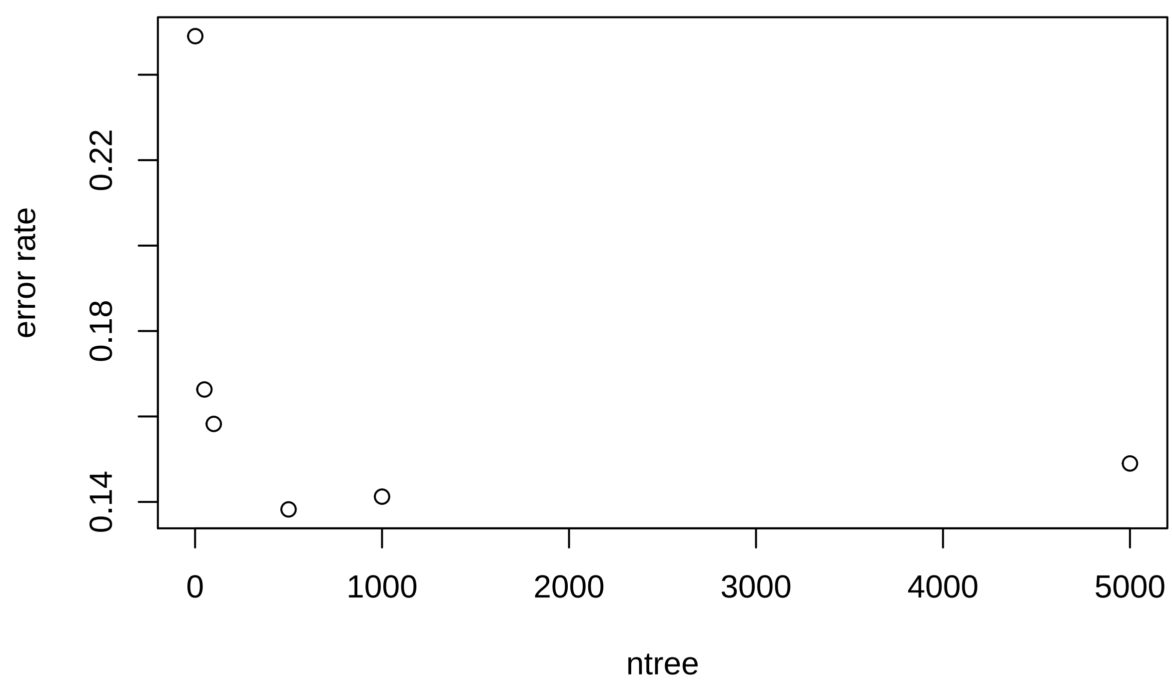
```
##          1  268 1373
```

```r
test_stats(t)
```

```
##            Total Misclassification No Misclassification
## Percentage               0.1489997            0.1535817
##            Yes Misclassification
## Percentage             0.1440150
```

We find the test error smallest when n.trees is around 500. This indicates that we would have learned enough at 500th iteration and the following iterations may not be necessary.

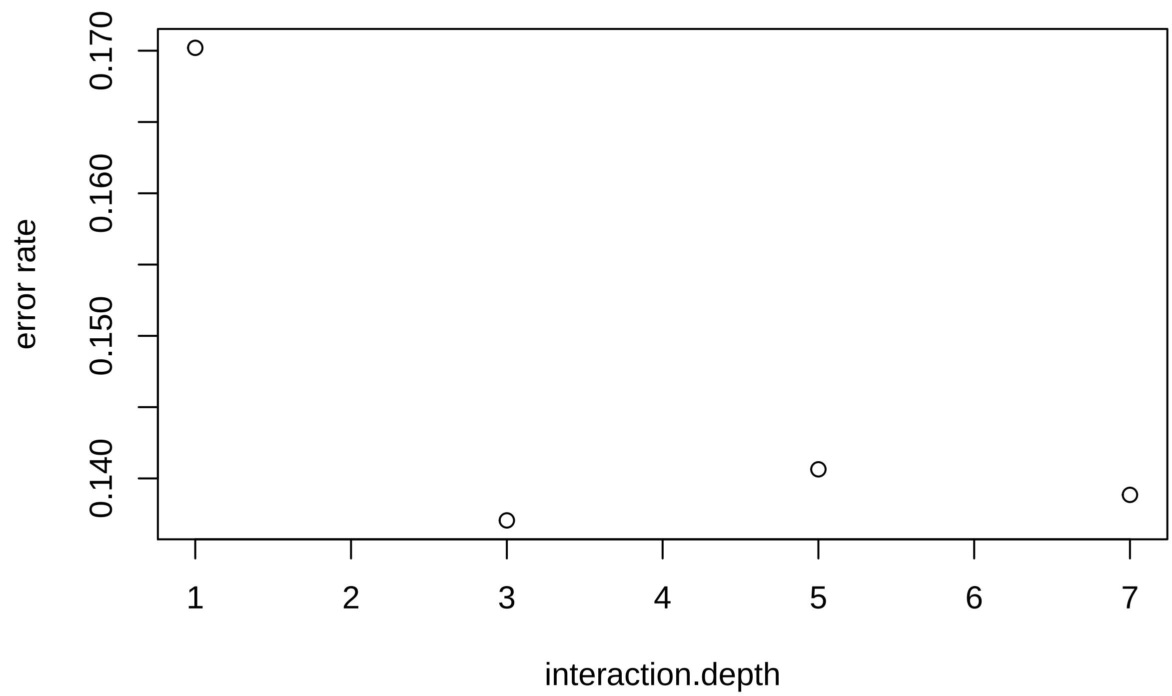
```r
nt = c(1, 50, 100, 500, 1000, 5000)
error = rep(NA, 6)
for(i in 1:6){
  boost_t_response = predict(boost, test, n.trees=nt[i], type='response')
  boost_t_pred = ifelse(boost_t_response>0.5, 1, 0)
  error[i] = mean(boost_t_pred!=test$deposit)
}
plot(x=nt, y=error, xlab='ntree', ylab='error rate')
```




We find the test error is smallest when interaction.depth is around 3. When the interaction.depth is higher, the model complexity becomes higher, so as the variance. We believe a interaction.depth is around 3 may be the best choice in this sample.

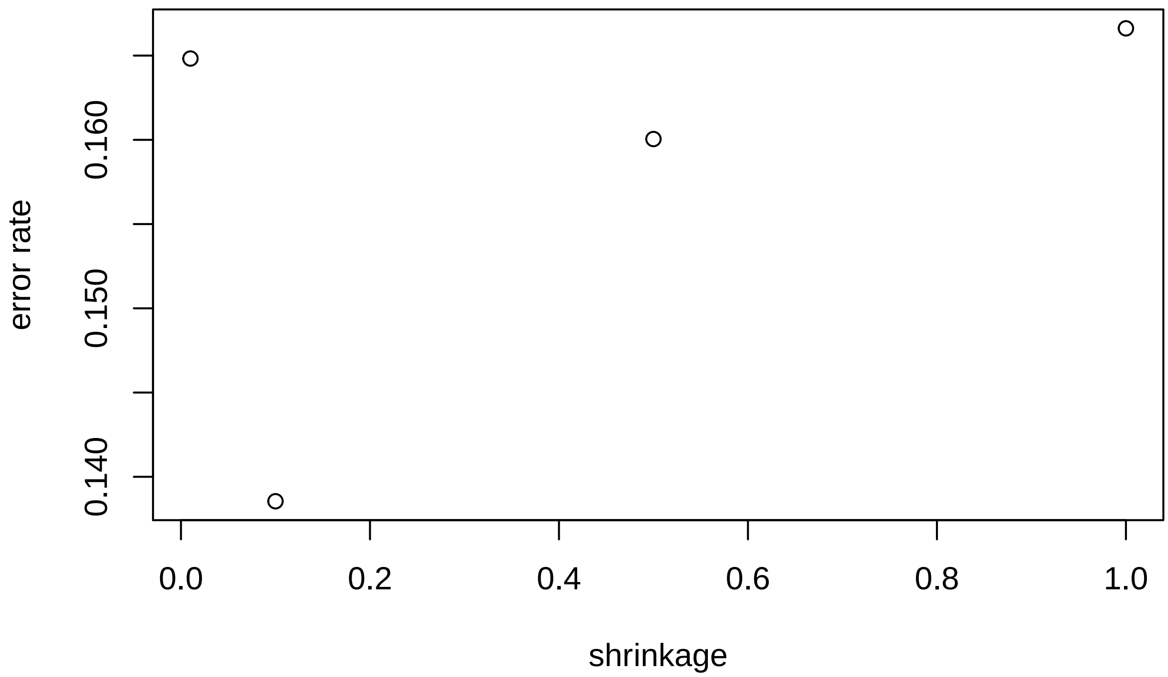
```r
id = c(1, 3, 5, 7)
error = rep(NA, 4)
for(i in 1:4){
  boost = gbm(deposit ~ ., data=train, distribution='adaboost', n.trees=500, interaction.depth = id[i],
```

```
  boost_t_response = predict(boost, test, n.trees=500, type='response')
  boost_t_pred = ifelse(boost_t_response>0.5, 1, 0)
  error[i] = mean(boost_t_pred!=test$deposit)
}
plot(x=id, y=error, xlab='interaction.depth', ylab='error rate')
```




We find the test error is smallest when the shrinkage is around 0.1. In general, the shrinkage is paired with the n.trees, where shrinkage controls the learning rate while n.trees controls how many times to learn. In our Adaboost model, we find that shrinkage = 0.1 is a good match with n.trees = 500 for this sample.

```
sh = c(1, 0.5, 0.1, 0.01)
error = rep(NA, 4)
for(i in 1:4){
  boost = gbm(deposit ~ ., data=train, distribution='adaboost', n.trees=500, interaction.depth = 3, shri
  boost_t_response = predict(boost, test, n.trees=500, type='response')
  boost_t_pred = ifelse(boost_t_response>0.5, 1, 0)
  error[i] = mean(boost_t_pred!=test$deposit)
}
plot(x=sh, y=error, xlab='shrinkage', ylab='error rate')
```


0.140
0.150
0.160
error rate
0.0
0.2
0.4
0.6
0.8
1.0
shrinkage