STA 521 HW6

William Tirone

The Honor Code:

❗ Important

(a) Please state the names of people who you worked with for this homework. You can also provide your comments about the homework here.
Eli Gnesin, Natalie Smith, Alonso Guererro
(b) Please type/write the following sentences yourself and sign at the end. We want to make it extra clear that nobody cheats even unintentionally.
I hereby state that all of my solutions were entirely in my words and were written by me. I have not looked at another student's solutions and I have fairly credited all external sources in this write up.

1 True or False

1. FALSE, it is built on a random set of m predictors.
2. FALSE, the trees in boosting are grown sequentially.
3. FALSE, the hyperparameters are the number of trees, the number of features/covariates, and the type of tree.
4. FALSE, they can handle discrete or continuous-valued covariates.
5.

FALSE, the forest makes the model less interpretable since it's a combination of the individual trees.

6. TRUE, a single tree could potentially overfit and learning slowly can help avoid this (ISL p. 346).

7. TRUE, since random forest performs bootstrapping it is less susceptible to noise in the data. Check lecture 23 p. 21 / ESL 10.6.

8. FALSE, a deep tree can overfit and lead to higher variance.

9. FALSE, in BART trees are grown successively on the original data (lecture 23 slide 29).

10. FALSE, AdaBoost is iterative.

11. TRUE, deeper trees have higher variance and a higher chance to overfit in general.

12. FALSE, it can overfit if you add too many trees.

2)

handwritten

3)

handwritten

4) (8.4.9)

a)

```
set.seed(123)
OJ.train = sample_n(OJ,800)
OJ.test = OJ[-as.integer(rownames(OJ.train)), ]
```

b)

Training Error Rate: 0.165.

Terminal Nodes: 8

```
t1 = tree(Purchase ~ ., data=OJ.train)
summary(t1)
```

```
Classification tree:
tree(formula = Purchase ~ ., data = OJ.train)
Variables actually used in tree construction:
[1] "LoyalCH"   "PriceDiff"
Number of terminal nodes:  8
Residual mean deviance:  0.7625 = 603.9 / 792
Misclassification error rate: 0.165 = 132 / 800
```

c)

The terminal node below indicates the split criteria, number of observations, deviance, y label, and the proportion of observations that that either MM or the other labels in the node.

```
 9) LoyalCH > 0.0356415 114  108.90 MM ( 0.18421 0.81579 )
```

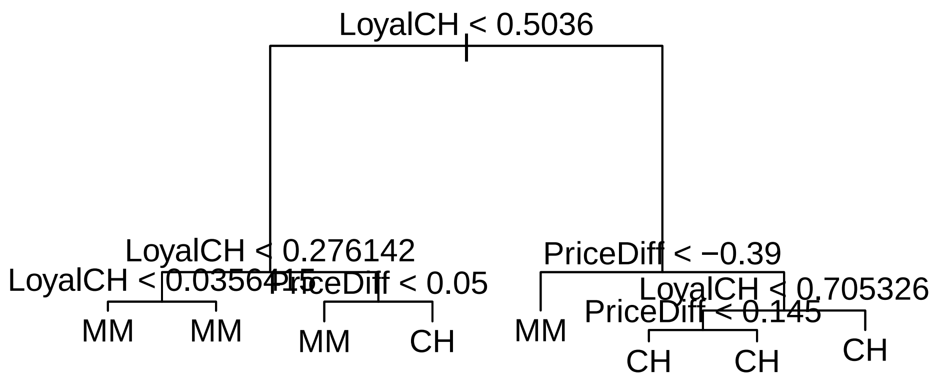
```
t1
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1071.00 CH ( 0.60875 0.39125 )
   2) LoyalCH < 0.5036 350  415.10 MM ( 0.28000 0.72000 )
     4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
       8) LoyalCH < 0.0356415 56   10.03 MM ( 0.01786 0.98214 ) *
       9) LoyalCH > 0.0356415 114  108.90 MM ( 0.18421 0.81579 ) *
     5) LoyalCH > 0.276142 180  245.20 MM ( 0.42222 0.57778 )
      10) PriceDiff < 0.05 74   74.61 MM ( 0.20270 0.79730 ) *
      11) PriceDiff > 0.05 106  144.50 CH ( 0.57547 0.42453 ) *
   3) LoyalCH > 0.5036 450  357.10 CH ( 0.86444 0.13556 )
     6) PriceDiff < -0.39 27   32.82 MM ( 0.29630 0.70370 ) *
     7) PriceDiff > -0.39 423  273.70 CH ( 0.90071 0.09929 )
      14) LoyalCH < 0.705326 130  135.50 CH ( 0.78462 0.21538 )
        28) PriceDiff < 0.145 43   58.47 CH ( 0.58140 0.41860 ) *
        29) PriceDiff > 0.145 87   62.07 CH ( 0.88506 0.11494 ) *
      15) LoyalCH > 0.705326 293  112.50 CH ( 0.95222 0.04778 ) *
```

d)

Interpretation: starting from the top, we can follow the edges to the terminal nodes to find the label for the training observations. For example, LoyalCH < 0.0356415 predicts a label of *MM* and LoyalCH > 0.5036, PriceDiff > -0.39, and LoyalCH > 0.705326 predicts a label of *CH*.

```
plot(t1)
text(t1)
```



e)

The test error rate for t1 is 0.1851

```
t1.pred = predict(t1, OJ.test, type='class')
table(t1.pred, OJ.test$Purchase)
```

```
t1.pred  CH  MM
     CH 150  34
     MM  16  70
```

```
# 1 minus correct classification
1 - (150 + 70)/270
```

```
[1] 0.1851852
```

f)

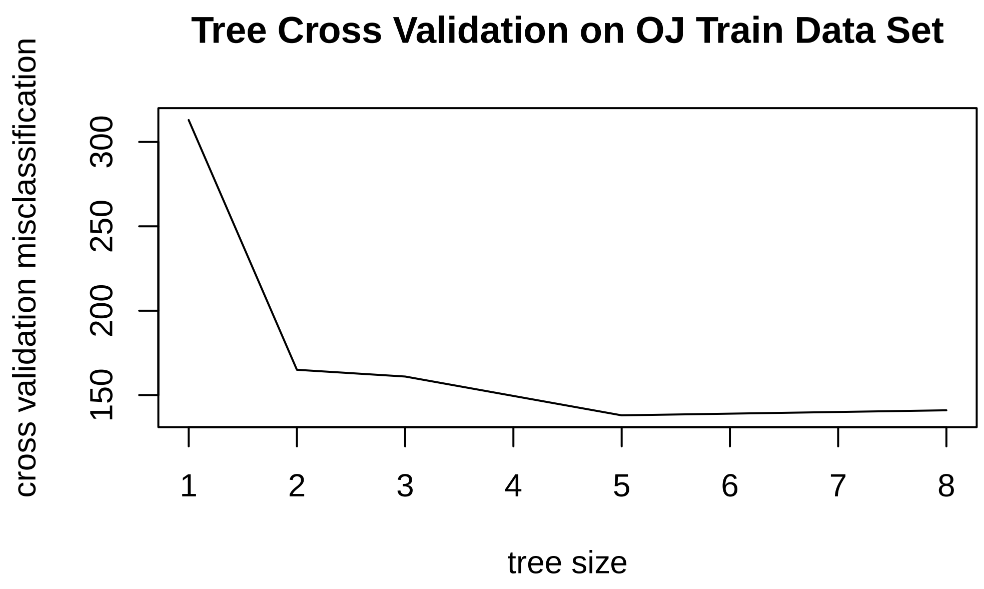
A tree with size 5 minimizes the misclassification error at 138.

```
cv.t1 = cv.tree(t1, FUN = prune.misclass)
cv.t1
```

```
$size
[1] 8 5 3 2 1

$dev
[1] 141 138 161 165 313

$k
[1] -Inf    0    8   11  154

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

g)

```
x = cv.t1$size
y = cv.t1$dev

plot(x,y,type='l',
     xlab='tree size',
     ylab = 'cross validation misclassification',
     main='Tree Cross Validation on OJ Train Data Set')
```



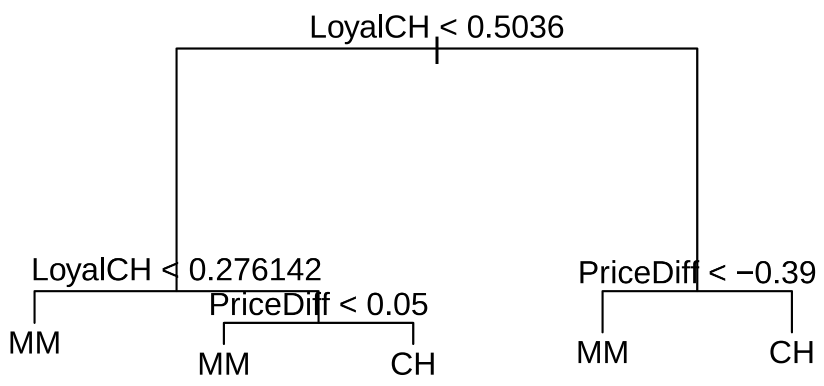
h)

A tree with size 5 minimizes the misclassification error at 138.

i)

Creating a tree with 5 nodes, the optimal number of nodes:

```
prune.t1 = prune.misclass(t1, best = 5)
plot(prune.t1)
text(prune.t1)
```



j)

The pruned tree has a training misclassification error rate of 0.165, which is the same as the unpruned version in this case.

```
summary(prune.t1)
```

```
Classification tree:
snip.tree(tree = t1, nodes = c(4L, 7L))
Variables actually used in tree construction:
[1] "LoyalCH"   "PriceDiff"
Number of terminal nodes:  5
Residual mean deviance:  0.826 = 656.6 / 795
Misclassification error rate: 0.165 = 132 / 800
```

k)

Since the pruned tree had the same training error rate, it is not surprising that it has the same error rate at 0.1851852.

```
t1.prune.pred = predict(prune.t1, OJ.test, type='class')
table(t1.prune.pred, OJ.test$Purchase)
```

```
t1.prune.pred  CH  MM
           CH 150  34
           MM  16  70
```

```
1 - (150 + 70) / 270
```

```
[1] 0.1851852
```

5) (8.4.10)

a)

```
Hitters = Hitters |>
  filter(!is.na(Hitters$Salary)) |>
  mutate(log_salary = log(Salary))
```

b)

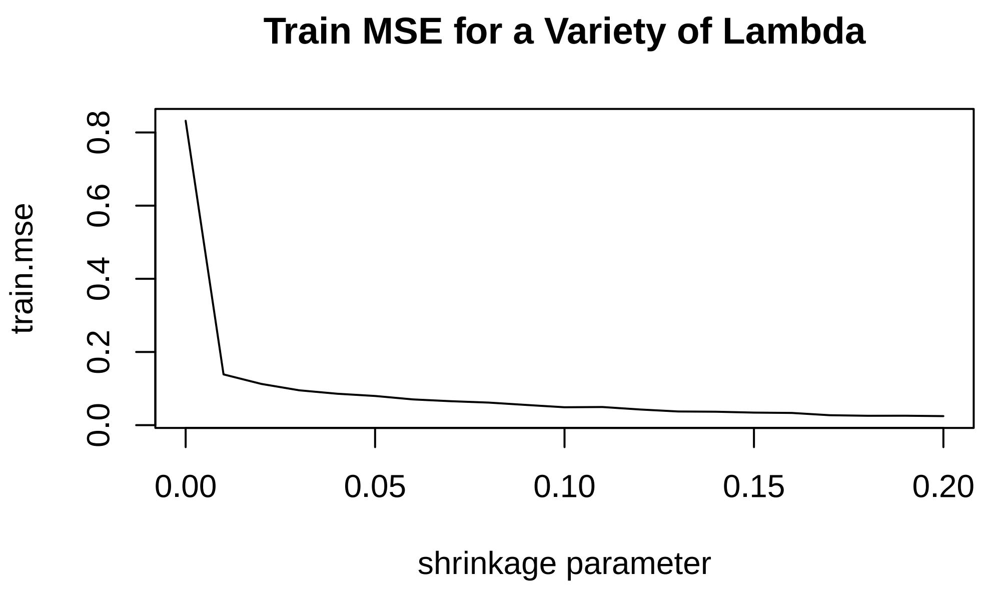
```
hit.train = Hitters[0:200,-19]
hit.test = Hitters[200 : dim(Hitters)[1] - 200,-19]
```

c)

```
set.seed(123)

shrink = seq(0,.2,0.01)

train.mse = c()

for (i in shrink) {

  boost.hit = gbm(log_salary  ~ .,
                 data=hit.train,
                 distribution = "gaussian",
                 n.trees = 1000,
                 shrinkage = i)

  y.hat = predict(boost.hit)
  mse = mean((y.hat - hit.train$log_salary)^2)
  train.mse = c(train.mse,mse)
}
```

```
Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...
Using 1000 trees...
```

```r
plot(shrink, train.mse,
     xlab = "shrinkage parameter",
     main = "Train MSE for a Variety of Lambda",
     type='l')
```

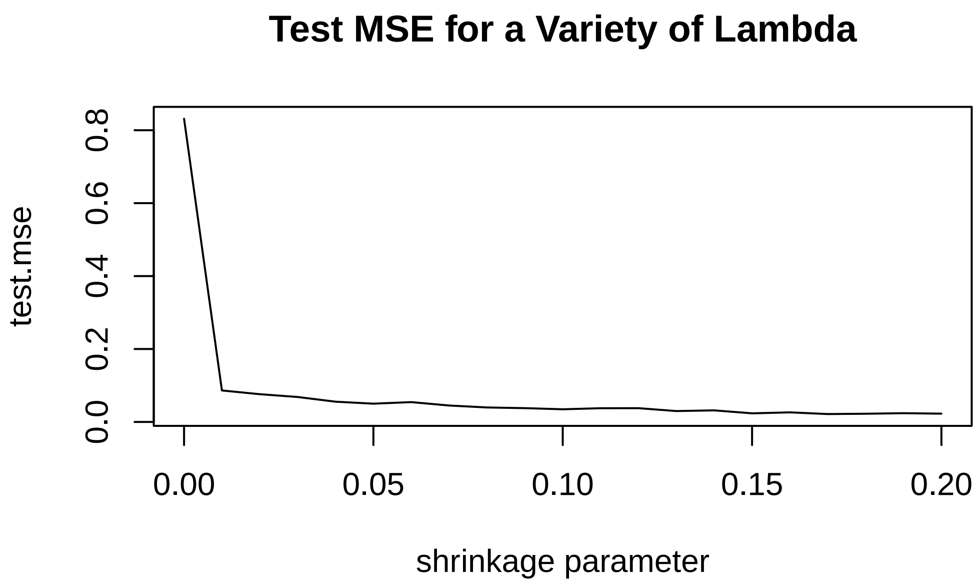


d)

```r
test.mse = c()

for (i in shrink) {

  boost.hit = gbm(log_salary  ~ .,
                  data=hit.train,
                  distribution = "gaussian",
                  n.trees = 1000,
                  shrinkage = i)

  y.hat = predict(boost.hit,hit.test)
  mse = mean((y.hat - hit.test$log_salary)^2)
  test.mse = c(test.mse,mse)
}
```

```
Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...

Using 1000 trees...
```

```r
plot(shrink, test.mse,
     xlab = "shrinkage parameter",
     main = "Test MSE for a Variety of Lambda",
     type='l')
```



e)

Using standard linear regression, we get an MSE of 0.3150123 which exceeds the test MSE of almost all choices of λ in the boosting example from part d.

With ridge regression, using $\lambda = .0024$ results in an MSE of 0.4062, which is still higher than linear regression and much higher than the boosting results.

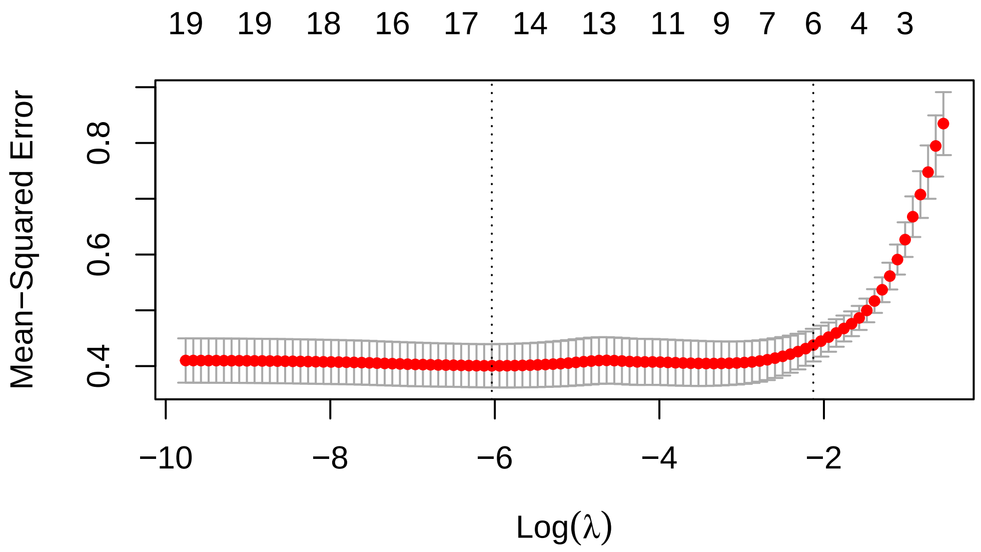
```r
# let's try OLS

lm.hit = lm(log_salary  ~ .,
            data=hit.train)

lm.hit.pred = predict(lm.hit,hit.test)
lm.hit.mse = mean((lm.hit.pred - hit.test$log_salary)^2)

lm.hit.mse
```

```
[1] 0.3150123
```

```
# trying ridge
ridge.hit = glmnet(data.matrix(hit.train[,-20]),
                   hit.train$log_salary)

cv.ridge = cv.glmnet(data.matrix(hit.train[,-20]),
                     hit.train$log_salary)

min.lambda = cv.ridge$lambda.min
print(min.lambda)
```

```
[1] 0.002389207
```

```
plot(cv.ridge)
```



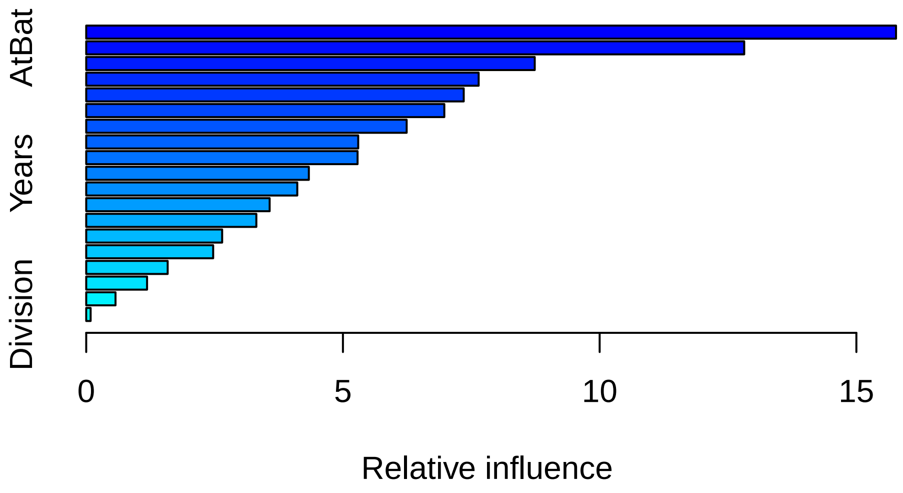
```
cv.ridge
```

```

Call:  cv.glmnet(x = data.matrix(hit.train[, -20]), y = hit.train$log_salary)

Measure: Mean-Squared Error

     Lambda Index Measure      SE Nonzero
min 0.00239    60  0.4005 0.03885      16
1se 0.11891    18  0.4376 0.02927       6
```

f)

With $\lambda = 1$, CRuns and Assists have a high amount of relative influence, and the rest have less than 8 relative influence.

```
# fitting a boosted model with lambda = 1
boost.hit = gbm(log_salary  ~ .,
              data=hit.train,
              distribution = "gaussian",
              n.trees = 1000,
              shrinkage = 1)

summary(boost.hit)
```



```
              var     rel.inf
CHits       CHits 15.77207016
AtBat       AtBat 12.81524681
PutOuts   PutOuts  8.73573664
Assists   Assists  7.64352883
RBI           RBI  7.35236712
Walks       Walks  6.97460518
CHmRun     CHmRun  6.24099045
Hits         Hits  5.29768315
Runs         Runs  5.28513322
Years       Years  4.33610497
CWalks     CWalks  4.11071176
CRBI         CRBI  3.57278190
HmRun       HmRun  3.31373209
```

```
CRuns         CRuns  2.64755156
Errors       Errors  2.47265446
CAtBat       CAtBat  1.58676154
League       League  1.18531884
NewLeague NewLeague  0.57068248
Division   Division  0.08633883
```

g)

The test MSE is 0.2096 with the bagging approach, which is not as good as boosting but better than linear and ridge regression.

```
bag.hit = randomForest(log_salary ~ ., data = hit.train,
                       mtry = 19)
y.hat.bag = predict(bag.hit, hit.test)
mse = mean((y.hat.bag - hit.test$log_salary)^2)
mse
```

```
[1] 0.02136258
```

References:

https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf