STATS 101C - Statistical Models and Data Mining - Homework 6

Darren Tsang, Discussion 4B

Produced on Saturday, Nov. 21 2020 @ 05:03:13 PM

Question 1 (Exercise 2 from Section 5.4)

Part A

The probability that the first bootstrap observation is not the jth observation from the original sample is $1 - \frac{1}{n}$. The probability that it is the jth observation is $\frac{1}{n}$ because each element is equally as likely to be chosen. Thus, we just do $1 - \frac{1}{n}$ to obtain the probability that is is not.

Part B

The same answer and reasoning as seen in Part A since the position in the bootstrap sample does not affect the probability of which observation is chosen.

Part C

We know that the probability that the jth observation is not in the first, second, etc bootstrap observation is $1 - \frac{1}{n}$. Thus, if there are n spots, the probability that the jth observation is not in any of the spots (so the jth observation isn't in the bootstrap sample at all) is just $(1 - \frac{1}{n})^n$.

Part D

$$1 - (1 - 1/5)^5 = .67232$$

Part E

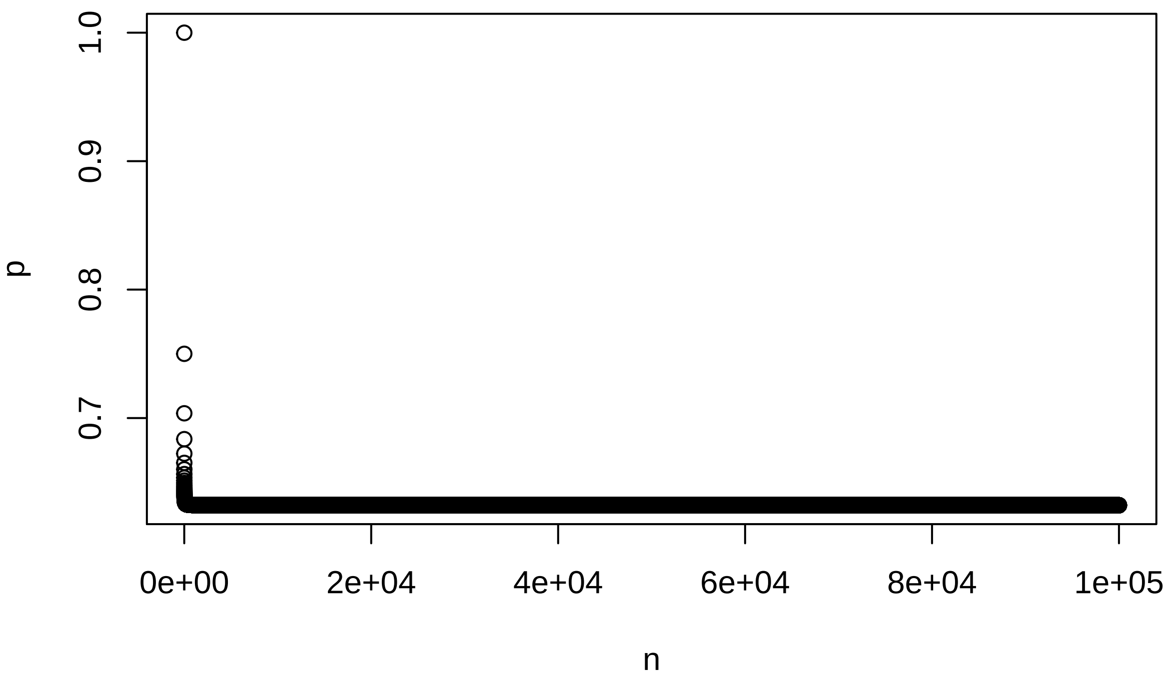
$$1 - (1 - 1/100)^{100} = .63397$$

Part F

$$1 - (1 - 1/10000)^{10000} = .63214$$

Part G

```r
n <- 1:100000
p <- 1 - (1 - 1/(n))^(n)
plot(p, xlab = 'n')
```




```r
min(p)
```

```
## [1] 0.6321224
```

We notice that the probability drops really quickly and begins to stabilize around .63.

Part H

```r
set.seed(999)
store <- rep(NA, 10000)
for (i in 1:10000) {
  store[i] <- sum(sample(1:100, rep=TRUE) == 4) > 0
}
mean(store)
```

```
## [1] 0.6376
```

```r
1 - (1 - 1/(100))^(100)
```

```
## [1] 0.6339677
```

We see that the empirical rate is slightly higher than the actual rate (0.6376 vs 0.6339677).

Question 2 (Exercise 10 from Section 8.4)

```
library(ISLR)
library(gbm)
library(glmnet)
library(randomForest)
library(caret)
```

```
data(Hitters)
```

Part A

```
Hitters <- Hitters[!is.na(Hitters$Salary),]
Hitters$Salary <- log(Hitters$Salary)
```

Part B

```r
hitters.training <- Hitters[1:200, ]
hitters.testing <- Hitters[-(1:200), ]

dim(hitters.training)
```

```
## [1] 200  20
```

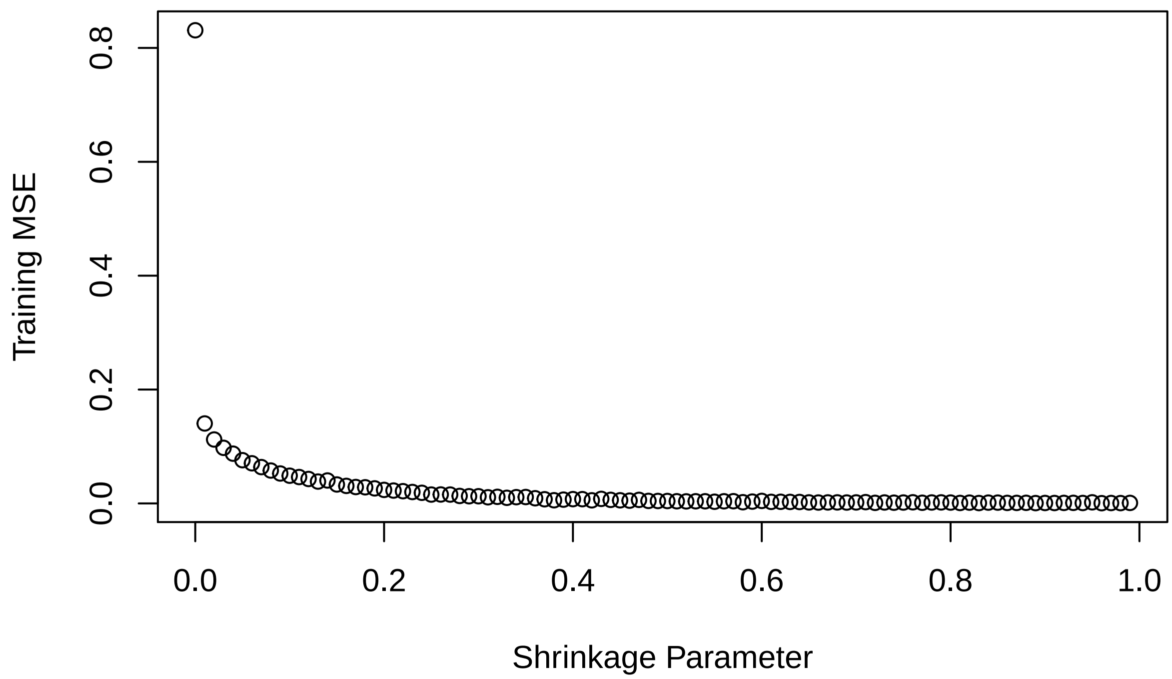
```r
dim(hitters.testing)
```

```
## [1] 63 20
```

Part C

```r
set.seed(999)
alpha.grid <- seq(.000001, 1, by = .01)

train.mse <- rep(NA, length(alpha.grid))
test.mse <- rep(NA, length(alpha.grid))
for (i in seq(1, length(alpha.grid))){
  boost.temp <- gbm(Salary ~ .,
                      data = hitters.training,
                      distribution = "gaussian",
                      n.trees = 1000,
                      shrinkage = alpha.grid[i])
  train.predicted <- predict(boost.temp, hitters.training, n.trees = 1000)
  test.predicted <- predict(boost.temp, hitters.testing, n.trees = 1000)
  train.mse[i] <- mean((train.predicted - hitters.training$Salary)^2)
  test.mse[i] <- mean((test.predicted - hitters.testing$Salary)^2)
}
```

```r
plot(alpha.grid, train.mse,
     xlab = "Shrinkage Parameter",
     ylab = "Training MSE")
```




```r
min(train.mse)
```

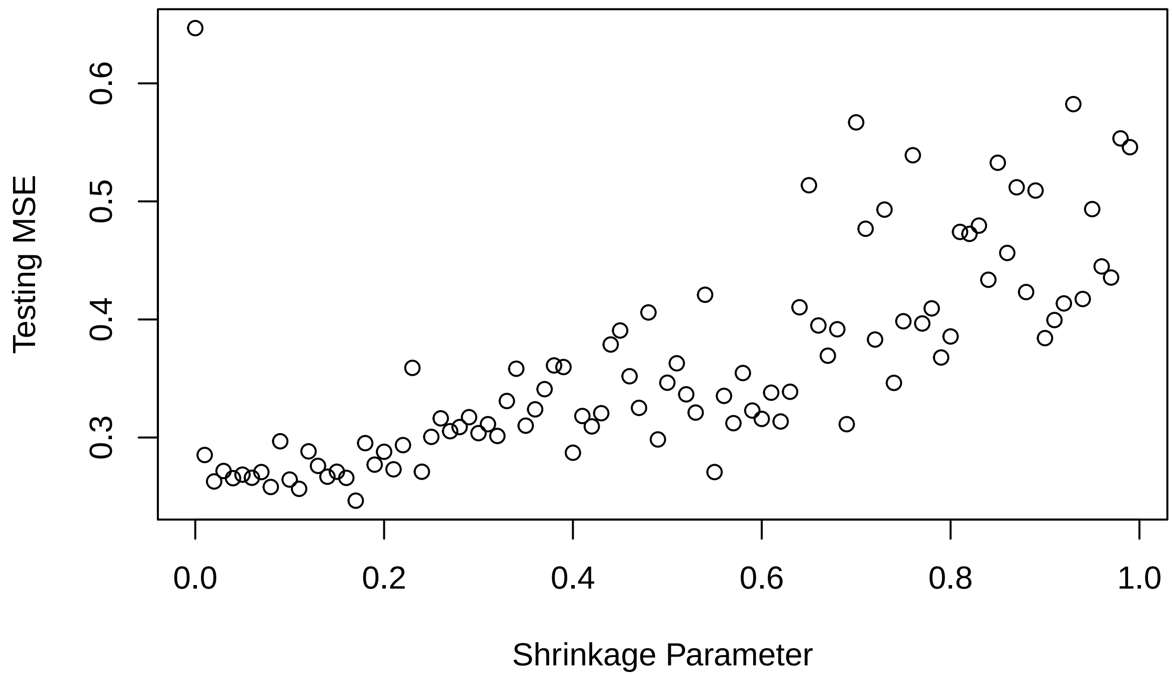
```
## [1] 0.0004535139
```

```r
alpha.grid[which.min(train.mse)]
```

```
## [1] 0.960001
```

Part D

```
plot(alpha.grid, test.mse,
     xlab = "Shrinkage Parameter",
     ylab = "Testing MSE")
```




```
min(test.mse)
```

```
## [1] 0.2465204
```

```
alpha.grid[which.min(test.mse)]
```

```
## [1] 0.170001
```

```
best.boosted <- gbm(Salary ~ .,
                    data = hitters.training,
                    distribution = "gaussian",
                    n.trees = 1000,
                    shrinkage = alpha.grid[which.min(test.mse)])
```

Part E

```r
lm.model <- lm(Salary ~ ., data = hitters.training)

lm.predictions <- predict(lm.model, newdata = hitters.testing)
lm.test.mse <- mean((lm.predictions - hitters.testing$Salary)^2)
lm.test.mse
```

```
## [1] 0.4917959
```

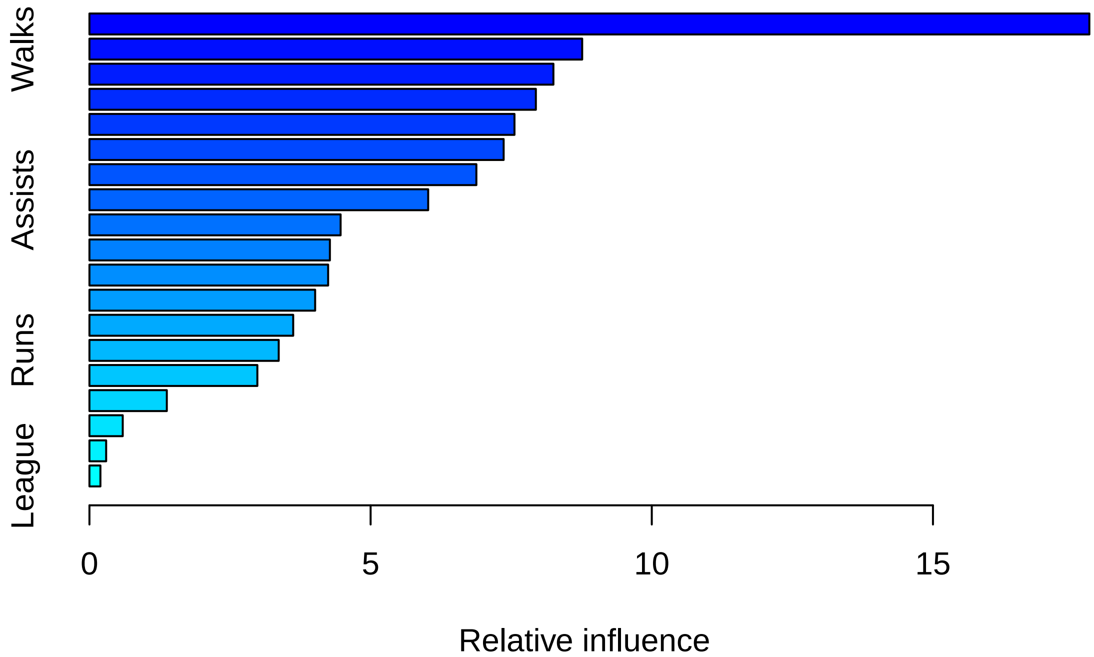
```r
grid <- 10^seq(10, -2, length = 100)
x <- model.matrix(Salary ~ ., hitters.training)[, -1]
y <- hitters.training$Salary

ridge.model <- glmnet(x, y,
                      family = "gaussian",
                      alpha = 0,
                      lambda = grid,
                      standardize = TRUE)

set.seed(999)
ridge.cv.output <- cv.glmnet(x, y, family = "gaussian", alpha = 0,
                             lambda = grid, standardize = TRUE,
                             nfolds = 10)

ridge.best.lambda.cv <- ridge.cv.output$lambda.min

ridge.predictions <- predict(ridge.model,
                             s = ridge.best.lambda.cv,
                             newx = model.matrix(Salary ~ ., hitters.testing)[,-1])

ridge.test.mse <- mean((ridge.predictions - hitters.testing$Salary)^2)
ridge.test.mse
```

```
## [1] 0.4645472
```

The test MSE for boosting, linear regression, and ridge regression are 0.2465204, 0.4917959, and 0.4645472, respectively. We can see that using boosting yielded us the lowest test MSE out of all three methods.

Part F

```
summary(best.boosted)
```

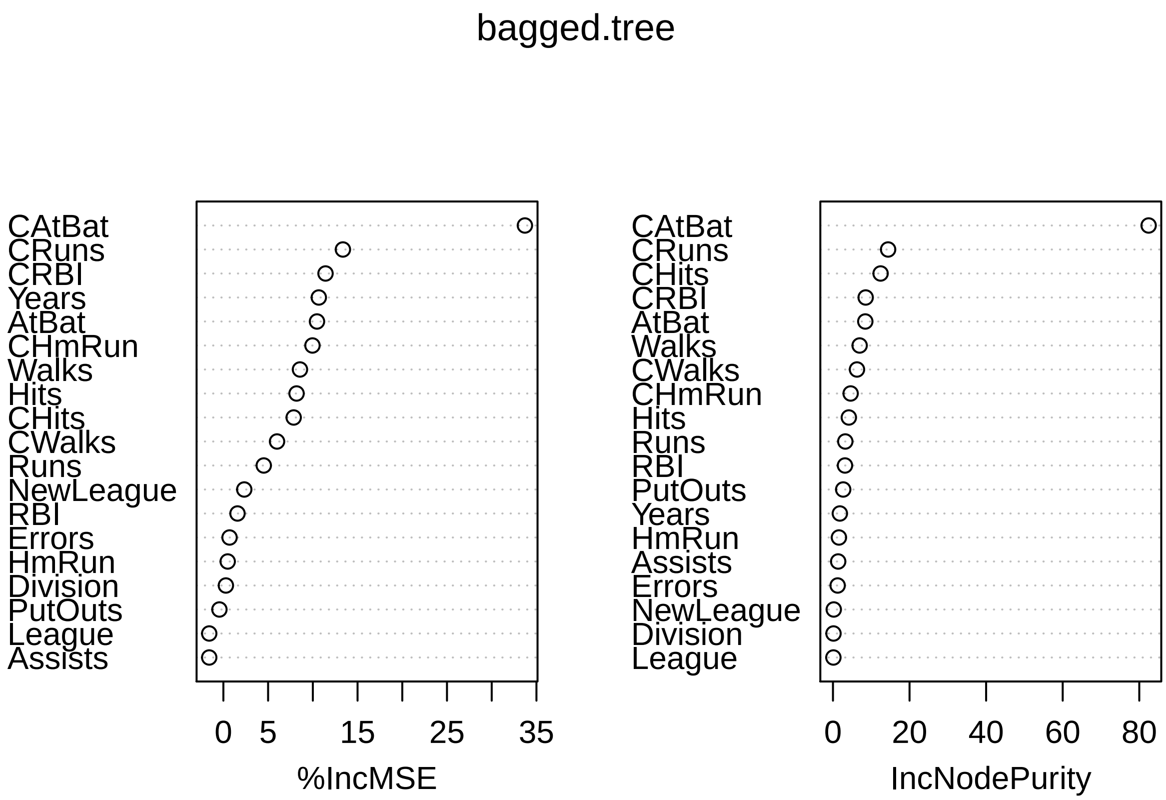



```
##                 var    rel.inf
## CAtBat       CAtBat 17.7814528
## Walks         Walks  8.7623928
## CWalks       CWalks  8.2509715
## PutOuts     PutOuts  7.9388886
## CRuns         CRuns  7.5583932
## CRBI           CRBI  7.3659814
## Years         Years  6.8809469
## Assists     Assists  6.0240156
## CHmRun       CHmRun  4.4665944
## HmRun         HmRun  4.2757269
## RBI             RBI  4.2446482
## Hits           Hits  4.0136771
## AtBat         AtBat  3.6233743
## Runs           Runs  3.3658857
## Errors       Errors  2.9860672
## CHits         CHits  1.3760289
## Division   Division  0.5921576
## NewLeague NewLeague  0.2975151
## League       League  0.1952818
```

The most important variables in the boosted model are 'CAtBat', 'Walks', 'CWalks', 'PutOuts', and 'CRuns'.

Part G

```r
bagged.tree <- randomForest(Salary ~ .,
                            data = hitters.training,
                            mtry = dim(hitters.training)[2] - 1,
                            ntree = 500,
                            importance = T)
varImpPlot(bagged.tree)
```




```r
bagged.tree.predictions <- predict(bagged.tree, hitters.testing)
bagged.tree.mse <- mean((bagged.tree.predictions - hitters.testing$Salary)^2)
bagged.tree.mse
```

```
## [1] 0.2323669
```

Using bagging, the test MSE is 0.2323669, which is not better than boosting. The most important predictors when using bagging are 'CAtBat', 'CRuns', 'CRBI', 'Years', and 'AtBat'. It is interesting to note that just cause a variable is important for bagging, that does not mean that variable will be important for boosting. For example, 'AtBat' is important for bagging, but not so important for boosting.