Problem Set 3

Rei Bertoldi

2/16/2020

Decision Trees

Question 1

```
set.seed(10)
nes <- read_csv("nes2008.csv")
p <- length(nes) - 1
lambda <- seq(0.0001, 0.04, 0.001)
```

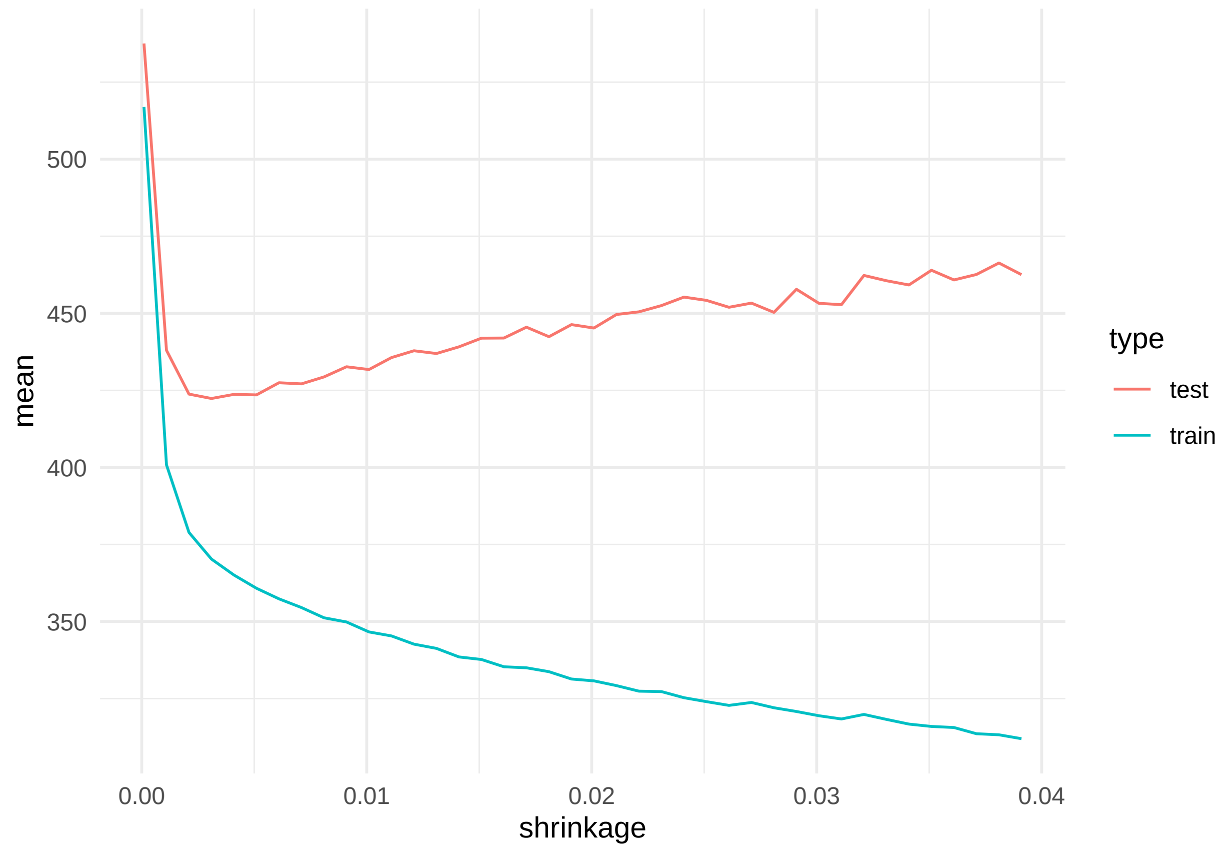
Question 2

```
set.seed(10)
train_nes <- sample(1:1807,size=1355)
train <- nes[train_nes,]
test <- nes[-train_nes,]
```

Question 3

```
df_out <- tibble()
for (i in lambda){
  boost_nes <- gbm(biden ~ .,
                   data = nes[train_nes,],
                   distribution="gaussian",
                   n.trees = 1000,
                   interaction.depth = 4,
                   shrinkage = i)
  preds_train <- predict(boost_nes, newdata=train,
                         n.trees = 1000)
  preds_test <- predict(boost_nes, newdata=test,
                   n.trees = 1000)
  train_mean <- mean((preds_train - train$biden)^2)
  test_mean <- mean((preds_test - test$biden)^2)
  train_out <- as.data.frame(train_mean) %>%
    mutate(shrinkage = i, type = "train") %>%
    rename(mean = train_mean)
  test_out <- as.data.frame(test_mean) %>%
    mutate(shrinkage = i, type = "test") %>%
    rename(mean = test_mean)
  df <- bind_rows(train_out, test_out)
  df_out <- bind_rows(df_out, df)
}
```

```
ggplot(df_out) +
  geom_line(aes(x = shrinkage, y = mean, group = type, color = type)) +
  theme_minimal()
```



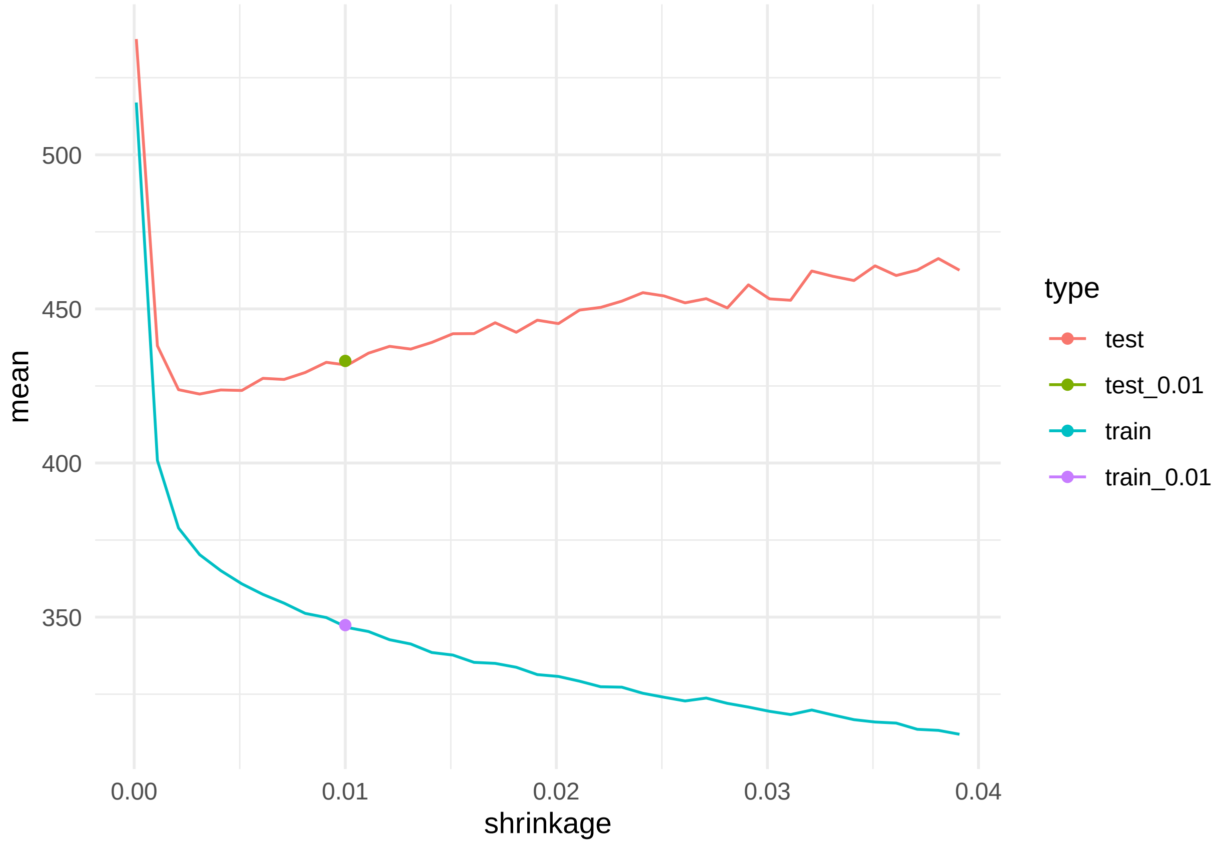
Question 4

```
boost_nes_2 <- gbm(biden ~ .,
                 data = nes[train_nes,],
                 distribution="gaussian",
                 n.trees = 1000,
                 interaction.depth = 4,
                 shrinkage = 0.01)
preds_train_2 <- predict(boost_nes_2, newdata=train,
                       n.trees = 1000)
preds_test_2 <- predict(boost_nes_2, newdata=test,
                      n.trees = 1000)
train_mean_2 <- mean((preds_train_2 - train$biden)^2)
test_mean_2 <- mean((preds_test_2 - test$biden)^2)
train_out_2 <- as.data.frame(train_mean_2) %>%
  mutate(shrinkage = 0.01, type = "train_0.01") %>%
  rename(mean = train_mean_2)
test_out_2 <- as.data.frame(test_mean_2) %>%
  mutate(shrinkage = 0.01, type = "test_0.01") %>%
  rename(mean = test_mean_2)
mean_0.01 <- bind_rows(train_out_2, test_out_2)
```

```
test_out_2
```

```
##       mean shrinkage      type
## 1 433.1164      0.01 test_0.01
```

```
ggplot() +
  geom_line(data = df_out,
            aes(x = shrinkage, y = mean, group = type, color = type)) +
  geom_point(data = mean_0.01,
             aes(x = shrinkage, y = mean, group = type, color = type)) +
  theme_minimal()
```



Looking at the plot above, we can see that the mse is slightly higher but essentially the same to the mse values we gathered in question 3. This means that we are not necessarily gaining much accuracy when tuning lambda. However, looking at the plot we can also notice that then lambda is very small or fairly large, the larger the mse.

Question 5

```
set.seed(10)
train_nes_bagging <- nes %>%
  sample_frac(0.75)
test_nes_bagging <- nes %>%
  setdiff(train_nes_bagging)

bag_nes <- bagging(
  formula = biden ~ .,
  data = train_nes_bagging,
  nbagg = 100,
  coob = TRUE
)
bag_pred <- predict(bag_nes, newdata=test_nes_bagging)
mean((bag_pred - test_nes_bagging$biden)^2)
```

```
## [1] 466.8401
```

Question 6

```
set.seed(10)
rf_nes <- randomForest(biden ~ .,
             data = nes,
             subset = train_nes)
rf_pred <- predict(rf_nes, newdata=test)
mean((rf_pred - test$biden)^2)
```

```
## [1] 429.7925
```

Question 7

```
lin_nes <- lm(biden ~ ., data = train)
lin_pred <- predict(lin_nes, newdata=test)
mean((lin_pred - test$biden)^2)
```

```
## [1] 426.0333
```

Question 8

Linear regression has the smaller mse at 426.0333426. Since it has the smallest mse it is probably is the best fit.

Support Vector Machines

Question 1

```
set.seed(10)
train_sample <- sample(1:nrow(OJ),size=800)
train_oj <- OJ[train_sample, ]
test_oj <- OJ[-train_sample, ]
```

Question 2

```
set.seed(10)
svm_oj <- svm(Purchase ~ .,
    data = train_oj,
    kernel = "linear",
    cost = 0.01)
summary(svm_oj)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_oj, kernel = "linear", cost = 0.01)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  451
##
##  ( 226 225 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There are 451 support vectors, 226 in one class and 225 in the other,

Question 3

```r
pred_train_svm <- predict(svm_oj, train_oj)
train_matrix <-
  confusionMatrix(
    pred_train_svm, train_oj$Purchase, dnn = c("Prediction", "Reference"))
table(train_oj$Purchase, pred_train_svm)
```

```
##     pred_train_svm
##       CH  MM
##   CH 431  60
##   MM  84 225
```

```r
1 - train_matrix$overall[1]
```

```
## Accuracy
##     0.18
```

```r
pred_test_svm <- predict(svm_oj, test_oj)
test_matrix <-
  confusionMatrix(
    pred_test_svm, test_oj$Purchase, dnn = c("Prediction", "Reference"))
table(test_oj$Purchase, pred_test_svm)
```

```
##     pred_test_svm
##       CH  MM
##   CH 143  19
##   MM  23  85
```

```r
1 - test_matrix$overall[1]
```

```
##  Accuracy
## 0.1555556
```

Question 4

```r
set.seed(10)
tune_oj <- tune(svm,
               Purchase ~ .,
               data = train_oj,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 10, 50, 100, 500, 1000)))

summary(tune_oj)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.18
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.18250 0.04090979
## 2 1e-01 0.18000 0.03736085
## 3 1e+00 0.18250 0.03545341
## 4 1e+01 0.18000 0.03782269
## 5 5e+01 0.18250 0.03736085
## 6 1e+02 0.18375 0.03537988
## 7 5e+02 0.18375 0.03775377
## 8 1e+03 0.18375 0.04084609
```

```r
tuned_model <- tune_oj$best.model
summary(tuned_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train_oj,
##     ranges = list(cost = c(0.01, 0.1, 1, 10, 50, 100, 500, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  367
##
##  ( 185 182 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Question 5

```
pred_train_tuned <- predict(tuned_model, train_oj)
train_matrix_tuned <-
  confusionMatrix(
    pred_train_tuned, train_oj$Purchase, dnn = c("Prediction", "Reference"))
table(train_oj$Purchase, pred_train_tuned)
```

```
##     pred_train_tuned
##       CH  MM
##   CH 430  61
##   MM  79 230
```

```
1 - train_matrix_tuned$overall[1]
```

```
## Accuracy
##    0.175
```

```
pred_test_tuned <- predict(tuned_model, test_oj)
test_matrix_tuned <-
  confusionMatrix(
    pred_test_tuned, test_oj$Purchase, dnn = c("Prediction", "Reference"))
table(test_oj$Purchase, pred_test_tuned)
```

```
##     pred_test_tuned
##       CH  MM
##   CH 145  17
##   MM  21  87
```

```
1 - test_matrix_tuned$overall[1]
```

```
##  Accuracy
## 0.1407407
```

The test and training error rates in the tuned model are slightly lower than the model from question 2. In question 2, the training error rate was 0.18 and the test error rate was 0.1555556. In the tuned model, the training error rate was 0.175 and the test error rate was 0.1407407. The error rates in the tuned model are lower, but only slightly lower. The tuned model preformed only slightly better than the model from question 2. Depending on the context and how much we value accuracy, even out to the third decimal place, we may just use the model from question 2 since the tuned model was much more computationally expensive.