Biden Sentament and OJ Trees and SVMs

Sarah Gill

2/15/2020

Decision Trees

1. Set up the data and store some things for later use:

```
set.seed(13579) #global set seed did not seem to work! setting in every chunk
nes2008_df <- read_csv("data/nes2008.csv")
```

```
## Parsed with column specification:
## cols(
##   biden = col_integer(),
##   female = col_integer(),
##   age = col_integer(),
##   educ = col_integer(),
##   dem = col_integer(),
##   rep = col_integer()
## )
```

```
p <- subset(nes2008_df,select = -c(biden))
lambda <- seq(from = 0.0001, to = 0.04, by = 0.001)
```

2. Create a training set consisting of 75% of the observations, and a test set with all remaining obs.

```
set.seed(13579)
split <- initial_split(nes2008_df, prop = 0.75)

train <- training(split)

test <- testing(split)
```

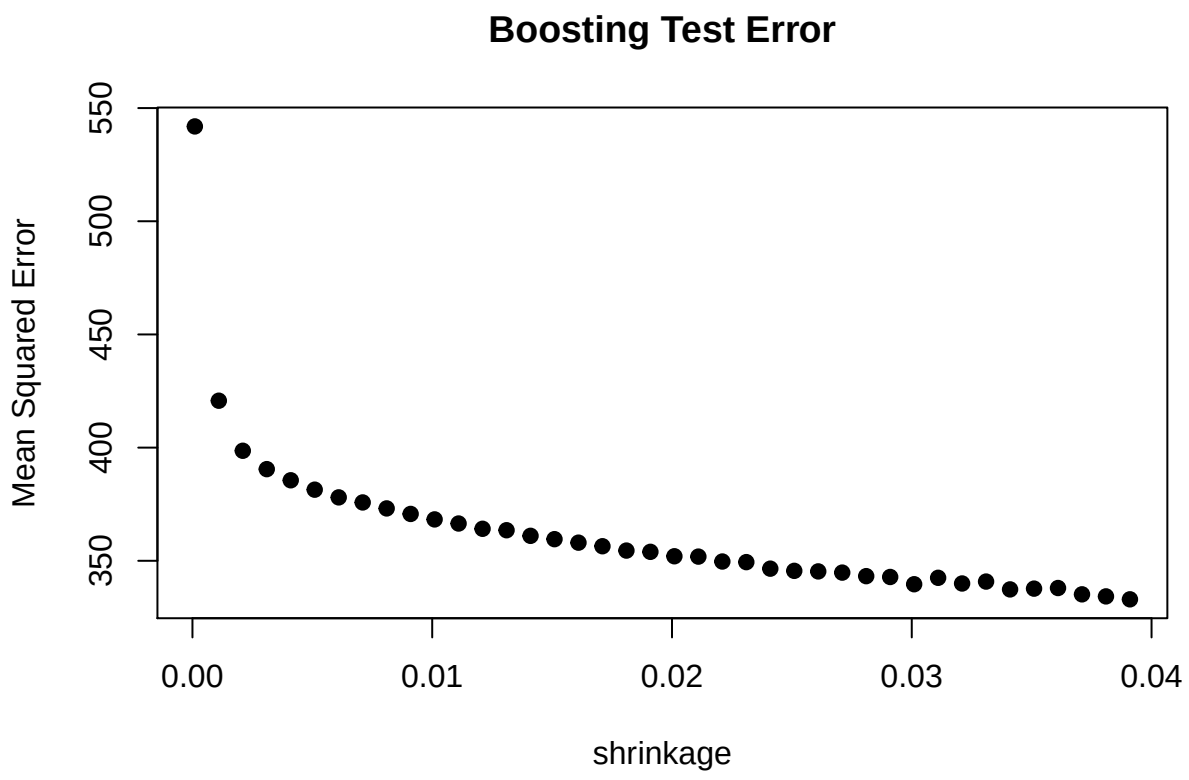
3. Create empty objects to store training and testing MSE, and then write a loop to perform boosting on the training set with 1,000 trees for the pre-defined range of values of the shrinkage parameter, lambda.

```
set.seed(13579)
mses_train <- c()
mses_test <- c()

for (i in lambda){

boost.biden <- gbm(biden ~ .,
                   data=train,
                   distribution="gaussian", #assuming distribution is gaussian
                   n.trees=1000, #do it 10000 times
```

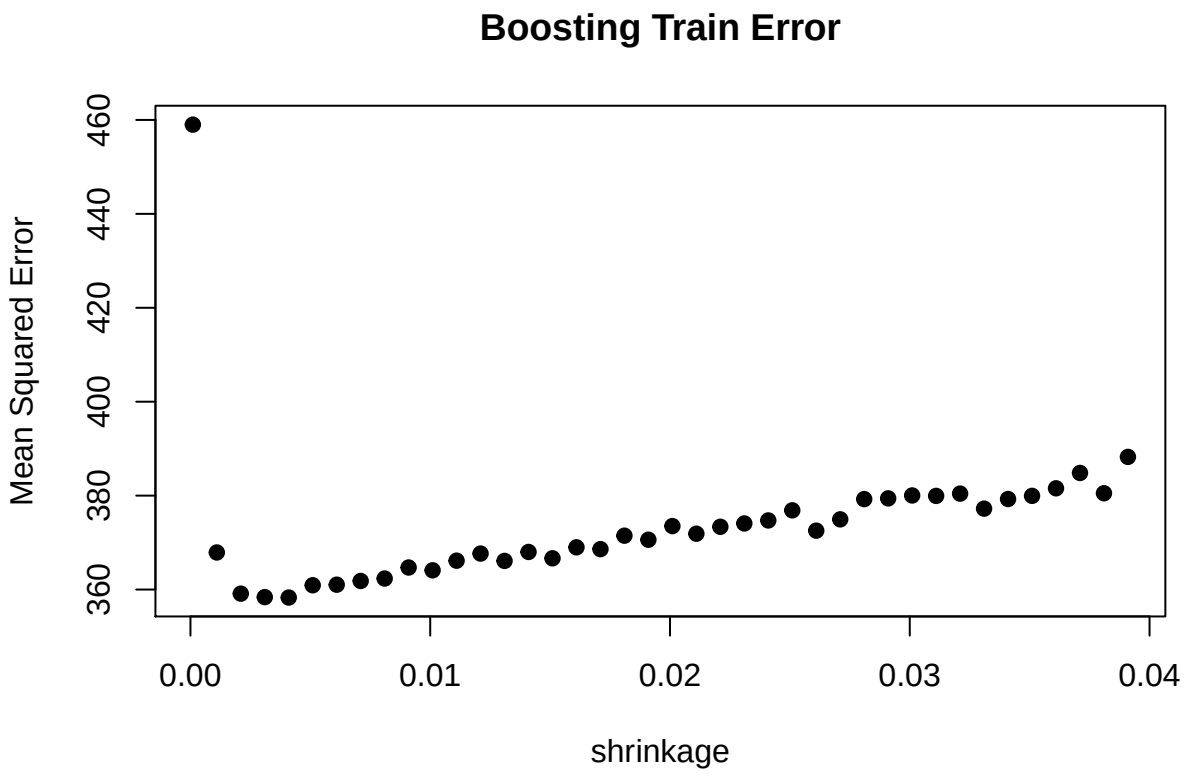
```
                    shrinkage=i,
                    interaction.depth = 4) #d, how many nodes to have


preds_tr = predict(boost.biden, newdata=train, n.trees = 1000)
preds_te = predict(boost.biden, newdata=test, n.trees = 1000)
                # shrinkage = lambda)

mse_train = MSE(y_pred=preds_tr, y_true=train$biden)
mses_train <- append(mses_train, mse_train)

mse_test = MSE(y_pred=preds_te, y_true=test$biden)
mses_test <- append(mses_test, mse_test)
}
```

Then, plot the training set and test set MSE across shrinkage values.

```
plot(lambda, mses_train,
     pch=19,
     ylab="Mean Squared Error",
     xlab="shrinkage",
     main="Boosting Test Error")
```



```r
plot(lambda, mses_test,
     pch=19,
     ylab="Mean Squared Error",
     xlab="shrinkage",
     main="Boosting Train Error")
```



4. (10 points) The test MSE values are insensitive to some precise value of lambda as long as its small enough. Update the boosting procedure by setting lambda equal to 0.01 (but still over 1000 trees). Report the test MSE and discuss the results. How do they compare?

```r
set.seed(13579)
boost.biden <- gbm(biden ~ .,
                   data=train,
                   distribution="gaussian", #assuming distribution is gaussian
                   n.trees=1000, #do it 10000 times
                   shrinkage=0.01, #penalize here, this is a calssic value
                   interaction.depth = 4) #d, how many nodes to have
#deapth must be smaller than number of features
#2-5 is pretty standard for d

#preds = predict(boost.biden, newdata=test, n.trees = 1000)

mse_test_0.01 = MSE(predict(boost.biden, newdata=test, n.trees = 1000), test$biden)
mse_test_0.01
```

```
## [1] 364.0391
```

mse_test at shrinkage = 0.01 is 364.0391

```
range(unlist(mses_test))
```

```
## [1] 358.2985 458.9984
```

```
mean(unlist(mses_test))
```

```
## [1] 373.8601
```

mse on our test portion of the data, using shrinkage from 0.001 to 0.04 range from: 358.2985 to 458.9984 with a mean of 373.8601

The mse from a shrinkage of 0.01, is between the minimum and mean. This shows that we are not gaining only a small amount of additional accuracy by tuning the shrinkage parameter. Further, note that the smaller lambdas do better than larger lambdas on the test set (after a point), but worse on the training set. This is a reminder of the importance of using test set error measurements

5. Now apply bagging to the training set. What is the test set MSE for this approach?

```
set.seed(13579)
biden_bag1 <- bagging(
  formula = biden ~ .,
  data = train,
  nbagg = 100,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)

mse_test_bag = MSE(predict(biden_bag1, newdata=test),test$biden)
mse_test_bag
```

```
## [1] 493.0998
```

```
#site https://bradleyboehmke.github.io/HOML/bagging.html
```

MSE for bagged is 493.0998

6. Now apply random forest to the training set. What is the test set MSE for this approach?

```
set.seed(13579)
rf_biden <- randomForest(biden ~ ., data = train)

mse_test_rf = MSE(y_pred = predict(rf_biden, newdata=test), y_true = test$biden)
mse_test_rf
```

```
## [1] 359.458
```

MSE from random forrest is 359.458

7. Now apply linear regression to the training set. What is the test set MSE for this approach?

```
set.seed(13579)
lm_biden <- lm(biden ~ ., data = train)

mse_lm <- mean((test$biden - predict(lm_biden, test)) ^ 2)
mse_lm
```

```
## [1] 364.6943
```

MSE for linear regression is 364.694

8. Compare test errors across all fits. Discuss which approach generally fits best and how you concluded this.

```
mses <- matrix(c(min(unlist(mses_test)), mse_test_0.01, mse_test_bag, mse_test_rf, mse_lm), ncol=5, byr

colnames(mses) <- c( "boost(.0001-.04)", "boost(.01)","bagged","random forrest", "lm")
rownames(mses) <- c('mse')
as.table(mses)
```

```
##     boost(.0001-.04) boost(.01)   bagged random forrest       lm
## mse         358.2985   364.0391 493.0998       359.4580 364.6943
```

```
#site: https://www.cyclismo.org/tutorial/R/types.html#tables
```

```
boost(.0001-.04) boost(.01)   bagged random forrest       lm
```

mse 358.2985 364.0391 493.0998 359.4580 364.6943

The best fit appears to be the minimum found among using the shrinkage coefficient 0.0001-0.04 in boosting, with an mse of about 358.30. Next best is using random forrest with a test mse of about 359.46, followed by boosting with shrinkage at 0.01, test mse of about 364.04. The simple linear model is close behind with a test mse of 364.69. Bagging seems to be the worst fit with a test mse of about 493.1. Note that the order of these vary somewhat with random draws, with the linear model sometimes performing best. However bagging seems to always be the worst fit.

Support Vector Machines

1. Create a training set with a random sample of size 800, and a test set containing the remaining observations.

```
set.seed(13579)
#OJ
samples=sample(1:nrow(OJ),800)
train_oj <- OJ[samples, ]
test_oj <- OJ[-samples, ]
```

2. Fit a support vector classifier to the training data with cost = 0.01, with Purchase as the response and all other features as predictors. Discuss the results.

```r
set.seed(13579)
svm_oj <- svm(Purchase ~ .,
              data = train_oj,
              kernel = "linear",
              cost = 0.01,
              scale = TRUE); summary(svm_oj)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_oj, kernel = "linear",
##     cost = 0.01, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  444
##
##  ( 223 221 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Using a cost of 0.01 and a linear hyper-plane, the model uses about 444 support vectors out of the 800 observations in the training set (the number of support vectors verys with random draw). 223 were CH and 221 were MM. This seems high, possibly indicating a high level of overlap in these classes

3. (5 points) Display the confusion matrix for the classification solution,

```r
table(pred = predict(svm_oj,
                     newdata = train_oj), true = train_oj$Purchase)
```

```
##     true
## pred  CH  MM
##   CH 418  76
##   MM  68 238
```

Report both the training and test set error rates.

Training set: True Positive for Citrus Hill: 418 True Positive for Minute Maid: 238 Predicted to be CH but really MM: 76 Predicted to be MM but really CH: 68

```r
#error rate
#1 - ((correct)/(total))
1 - ((418+238)/(800))
```

```
## [1] 0.18
```

```
#(1-mean(predict(svm_oj, test_oj) == test$Purchase))
```

Training set error rate of about 18%

```
table(pred = predict(svm_oj,
                     newdata = test_oj), true = test_oj$Purchase)
```

```
##     true
## pred  CH  MM
##   CH 152  27
##   MM  15  76
```

Test set: True Positive for Citrus Hill: 152 True Positive for Minute Maid: 76 Predicted to be CH but really MM: 27 Predicted to be MM but really CH: 15

```
1 - ((152+76)/(270))
```

```
## [1] 0.1555556
```

Test set error rate of about 15%

The error rate is a little bit lower for the test set than for the training set. This is a little odd, the training set is data that the learner has not seen before and we generally expect it to do slightly worse on it, however this can be explained by the random draw. (my global seed did not set correctly so I have seen several outcomes. Usually the test error rate has been just a little bit higher than the training set error rate)

4. Find an optimal cost in the range of 0.01 to 1000 (specific range values can vary; there is no set vector of range values you must use).

```
set.seed(13579)
tune_c <- tune(svm,
               Purchase ~ .,
               data = train_oj,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)))


#mannually put in tuning grid: ranges
#varying costs at each fit ????

# CV errors
summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
```

```
##  cost
##  1000
##
## - best performance: 0.18125
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.18500 0.04779877
## 2 1e-01 0.18250 0.03343734
## 3 1e+00 0.18250 0.04090979
## 4 5e+00 0.18375 0.03866254
## 5 1e+01 0.18250 0.04090979
## 6 1e+02 0.18250 0.03961621
## 7 1e+03 0.18125 0.03875224
```

```
# best?
tuned_model <- tune_c$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, 5, 10, 100, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1000
##       gamma:  0.05555556
##
## Number of Support Vectors:  348
##
##  ( 172 176 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Among the costs that we checked: 1000 is "best" (this time!)

(note I am having some trouble with my global set.seed, so I saw several outcomes of this. Most often 0.1 is "best" however the one time that 10 was "best" the error rate was lowest overall. (the one time that I saw 100 as "best" the error rate was nearly the same as it usually is for 0.1, the same was the case when I saw 5 as the "best"). With this seed I am seeing 1000 as "best". This just shows that the random draws can impact overall fit, however it looks like the draw for 10 being best was a draw in the tail, as potentially is this draw)

5. Display the confusion matrix for the classification solution

```
table(pred = predict(tuned_model,
                     newdata = train_oj), true = train_oj$Purchase)
```

```
##     true
## pred  CH  MM
##   CH 424  75
##   MM  62 239
```

Training set: True Positive for Citrus Hill: 424 True Positive for Minute Maid: 239 Predicted to be CH but really MM: 75 Predicted to be MM but really CH: 62

```
1-((239+424)/(800))
```

```
## [1] 0.17125
```

The error rate on the training set is about 17% (slightly better than before optimizing)

```
table(pred = predict(tuned_model,
                     newdata = test_oj), true = test_oj$Purchase)
```

```
##     true
## pred  CH  MM
##   CH 152  22
##   MM  15  81
```

Test set: True Positive for Citrus Hill: 152 True Positive for Minute Maid: 81 Predicted to be CH but really MM: 22 Predicted to be MM but really CH: 15

Report both the training and test set error rates.

```
1-((152+81)/(270))
```

```
## [1] 0.137037
```

The error rate on the test set is about 14%, this is a slight improvement over the error rate of about 15% before we tuned our cost hyper-parameter.

```
((152+81)/(270))
```

```
## [1] 0.862963
```

Thus, our optimally tuned classifier was correct about 86% of the time and incorrect about 14% of the time, when predicting purchasing on a set of data that it had not seen before (but that was initially collected in the same way as the data that it had seen, thus any biases in real data collection are expected to be the same between test and training sets). This is pretty good. Depending on what we wanted to use the classifier for we may care more about different measures of correct and incorrect classification, but overall looking at how many our classifier got correct compared to the total sample is helpful in assessing the classifier. And on this metric, this classifier is good. Note that this classifier does not do all that much better after being tuned than before. However, computing power is cheap so the small gains from optimizing are likely worth it.