PS3

Joanna Zhang

2/17/2020

Decision Trees

1.

```r
set.seed(135)
nesdata <- read_csv("data/nes2008.csv")
shrinkage <- seq(from = 0.0001, to = 0.04, by = 0.001)
p <- ncol(nesdata) - 1
```

2. Create training set

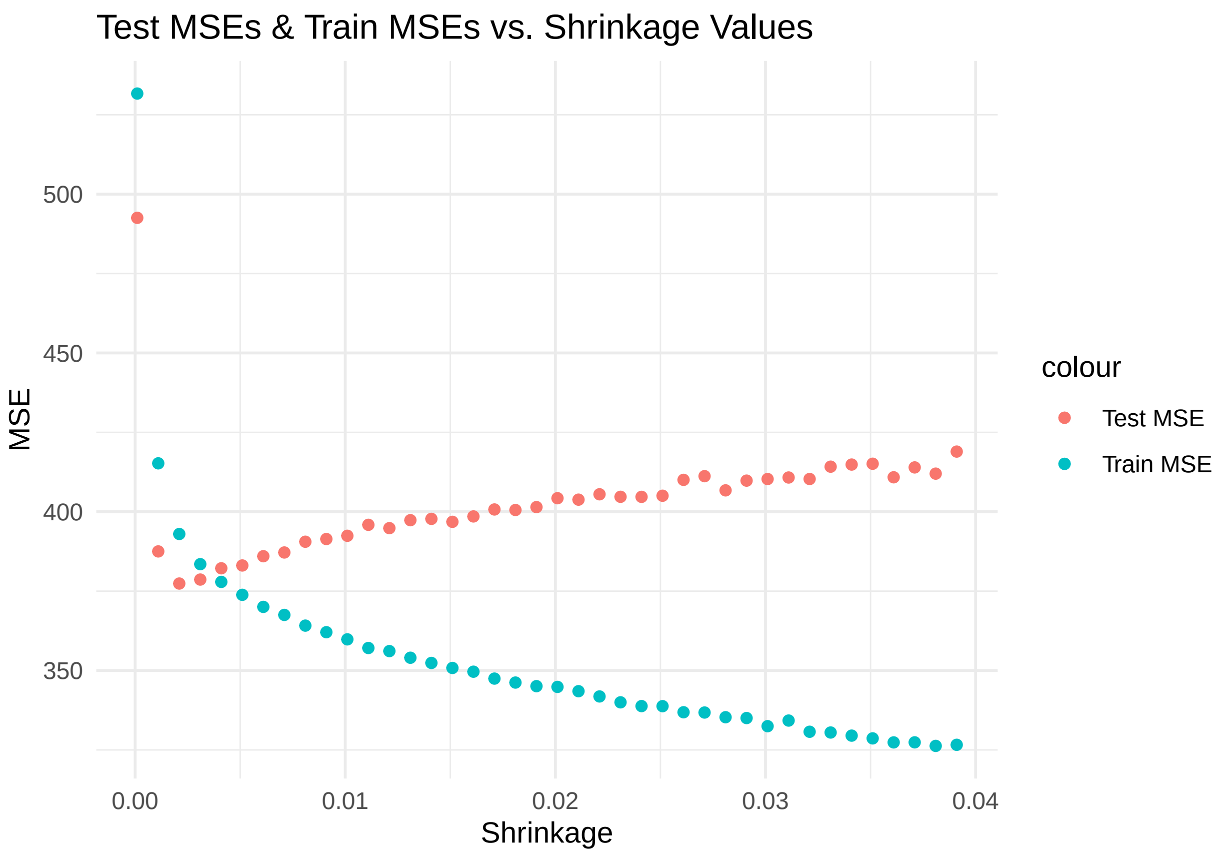
```r
# ramdomly select 75% of the observations
train = sample(1:nrow(nesdata), size = 0.75*nrow(nesdata))
```

3. Boosting

```r
trainingMSE <- vector()
testingMSE <- vector()
# loop over all lamda values
for (i in shrinkage){
boost.nes <- gbm(biden ~ ., data = nesdata[train,],
                    distribution="gaussian", n.trees=1000,
                    shrinkage=i, interaction.depth = 4)
trainpreds = predict(boost.nes, newdata = nesdata[train,],
                  n.trees = 1000)
testpreds = predict(boost.nes, newdata = nesdata[-train,],
                  n.trees = 1000)
trainmse <-  mean((trainpreds - nesdata[train,]$biden)^2)
testmse <-  mean((testpreds - nesdata[-train,]$biden)^2)
# store the MSEs in a vector
trainingMSE <-append(trainingMSE, trainmse)
testingMSE <- append(testingMSE, testmse)
}

MSEdf <- data.frame(Shrinkage = shrinkage, TrainMSE = trainingMSE, TestMSE = testingMSE)

MSEdf %>% ggplot() +
  geom_point(aes(Shrinkage, TrainMSE, color = 'Train MSE'))+
  geom_point(aes(Shrinkage, TestMSE, color = 'Test MSE'))+
  theme_minimal()+
  labs( y = "MSE", title = 'Test MSEs & Train MSEs vs. Shrinkage Values')
```



4.

```
set.seed(135)
boost.nes <- gbm(biden ~ ., data = nesdata[train,],
                    distribution="gaussian", n.trees=1000,
                    shrinkage= 0.01, interaction.depth = 4)
trainpreds = predict(boost.nes, newdata = nesdata[train,],
                  n.trees = 1000)
testpreds = predict(boost.nes, newdata = nesdata[-train,],
                  n.trees = 1000)
trainmse <-  mean((trainpreds - nesdata[train,]$biden)^2)
testmse <-  mean((testpreds - nesdata[-train,]$biden)^2)

trainmse
```

```
## [1] 359.9355
```

```
testmse
```

```
## [1] 392.1324
```

When shrinkage is 0.01, we get a training MSE of 360 and a test MSE of 392. The training MSE is lower than MSE, and this is normal because the observations in the test set were not used to train the model. The test MSE we get from a shrinkage of 0.01 is not optimal. As we see in the graph in (3), the test MSE reaches its lowest point at a shrinkage of about 0.003. Also the test MSE is pretty stable when lamda is between 0.002 and 0.1, this implies that the test MSE is not so sensitive to small lamda values.

5. Bagging

```r
set.seed(135)
library(rpart)
library(ipred)
bag <- bagging(biden ~ ., nbagg = 25, data = nesdata[train,], coob=T)

bag_pred <- predict(bag, newdata = nesdata[-train,])
bag_mse <- mean((bag_pred - nesdata[-train,]$biden)^2)
bag_mse
```

```
## [1] 377.7899
```

The bagging approach generates a MSE of 377.44.

6. Random Forest

```r
set.seed(135)
rf <- randomForest(biden ~ .,data = nesdata,subset = train, ntree = 1000)

rf_pred <- predict(rf, newdata = nesdata[-train,])
rf_mse <- mean((rf_pred - nesdata[-train,]$biden)^2)
rf_mse
```

```
## [1] 380.4552
```

The random forest model generates a MSE of 384.

7. Linear Regression

```r
linear_mod <- lm(biden ~., data = nesdata, subset = train)
modmse <- augment(linear_mod, newdata = nesdata[-train,]) %>%
  mse(truth = biden, estimate = .fitted)
modmse
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mse     standard        379.
```

The random forest model generates a MSE of 378.

8. Discussion

Of all the methods, the bagging model produces the smallest test MSE (377), which implies that the bagging model could be the best model. But note that the MSEs of the models may change once we reset the random process. It is possible that once we reset the seeds, the bagging model no longer generates the lowest MSE. Additionally, the linear model generates an MSE of 378, which is very close to the MSE of the bagging model. For simplicity and better interpretation, the linear regression model could be more preferable than the bagging model.

Support Vector Machines

1.

```r
set.seed(135)
data(OJ)
train = sample(1:nrow(OJ), size = 800)
trainOJ <- OJ[train,]
testOJ <- OJ[-train,]
```

2.

```r
svmOJ <- svm(Purchase ~ ., data = trainOJ,
             kernel = "linear",  cost = 0.01,
             scale = FALSE)
summary(svmOJ)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  623
##
##  ( 312 311 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The support vector classifier has 615 support vectors. 309 of the support vectors are from the Citrus Hill group, and 306 of the vectors are from the Minute Maid group. The number of support vectors is relatively large since we only have 800 observations. Once we use a larger cost this number may get smaller.

3.

```r
table(predicted = predict(svmOJ, newdata = testOJ), true = testOJ$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 144  30
##        MM  33  63
```

```
OJtrainpreds = predict(svmOJ, newdata = trainOJ)

train_err <- trainOJ %>%
  mutate(estimate = OJtrainpreds) %>%
  accuracy(truth = Purchase, estimate = estimate)
train_err<- 1 - train_err$.estimate[[1]]

OJtestpreds = predict(svmOJ, newdata = testOJ)
test_err <- testOJ %>%
  mutate(estimate = OJtestpreds) %>%
  accuracy(truth = Purchase, estimate = estimate)
test_err<- 1 - test_err$.estimate[[1]]

train_err
```

```
## [1] 0.245
```

```
test_err
```

```
## [1] 0.2333333
```

4.

```
set.seed(123)
tune_c <- tune(svm, Purchase ~ ., data = trainOJ, kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)))
summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.18
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.18750 0.05590170
## 2 1e-01 0.18125 0.05535554
## 3 1e+00 0.18000 0.06072479
## 4 5e+00 0.18125 0.05870418
## 5 1e+01 0.18125 0.05870418
## 6 1e+02 0.18500 0.05361903
## 7 1e+03 0.18625 0.06079622
```

```
tuned_best <- tune_c$best.model
tuned_best
```

```
##
## Call:
```

```
## best.tune(method = svm, train.x = Purchase ~ ., data = trainOJ,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  352
```

The optimal cost is 1.

5.

```r
optimal<- svmOJ <- svm(Purchase ~ ., data = trainOJ,
             kernel = "linear",  cost = 1,
             scale = FALSE)

summary(optimal)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, kernel = "linear",
##     cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  370
##
##  ( 186 184 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
trainpreds_opt = predict(optimal, newdata = trainOJ)

train_err_opt <- trainOJ %>%
  mutate(estimate = trainpreds_opt) %>%
  accuracy(truth = Purchase, estimate = estimate)
train_err_opt <- 1 - train_err_opt$.estimate[[1]]

testpreds_opt = predict(optimal, newdata = testOJ)
test_err_opt <- testOJ %>%
  mutate(estimate = testpreds_opt) %>%
  accuracy(truth = Purchase, estimate = estimate)
```

```r
test_err_opt<- 1 - test_err_opt$.estimate[[1]]

train_err_opt
```

```
## [1] 0.1725
```

```r
test_err_opt
```

```
## [1] 0.1407407
```

```r
table(predicted = predict(optimal, newdata = testOJ), true = testOJ$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 157  18
##        MM  20  75
```

The train error and the test error using the optimal cost are 17% and 14%, respectively. Compared to the previous classifiers, the optimal classifier produces smaller test and training errors. We have improved the test error by about 8%. I think an improvement of 8% is pretty significant. But note that compared to the classifier with cost = 0.01, the optimal classifier has more Citrus Hill misclassified as Minute Maid, but fewer Minute Maid misclassified as Citrus Hill.