Intro to Machine learning

Problem Set 3

Jingming Wei(Mark)

2/17/2020

```
# set working directory
setwd("/Users/Mark/Desktop/Intro to Machine Learning/HWK3")

#load some libraries
library(ggplot2)
library(gbm)
```

```
## Loaded gbm 2.1.5
```

Q1 Decision Trees

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

```
set.seed(1)
NES <- read.csv("/Users/Mark/Documents/GitHub/problem-set-3/data/nes2008.csv")
attach(NES)
p <- length(names(NES)) - 1
lambda <- seq(0.0001, 0.04, by = 0.001)
```

2. Train test split

```
set.seed(1)
train.index <- sample(1:nrow(NES), size=nrow(NES)*0.75, replace=FALSE)
train <- NES[train.index, ]
test <- NES[-train.index,]
```

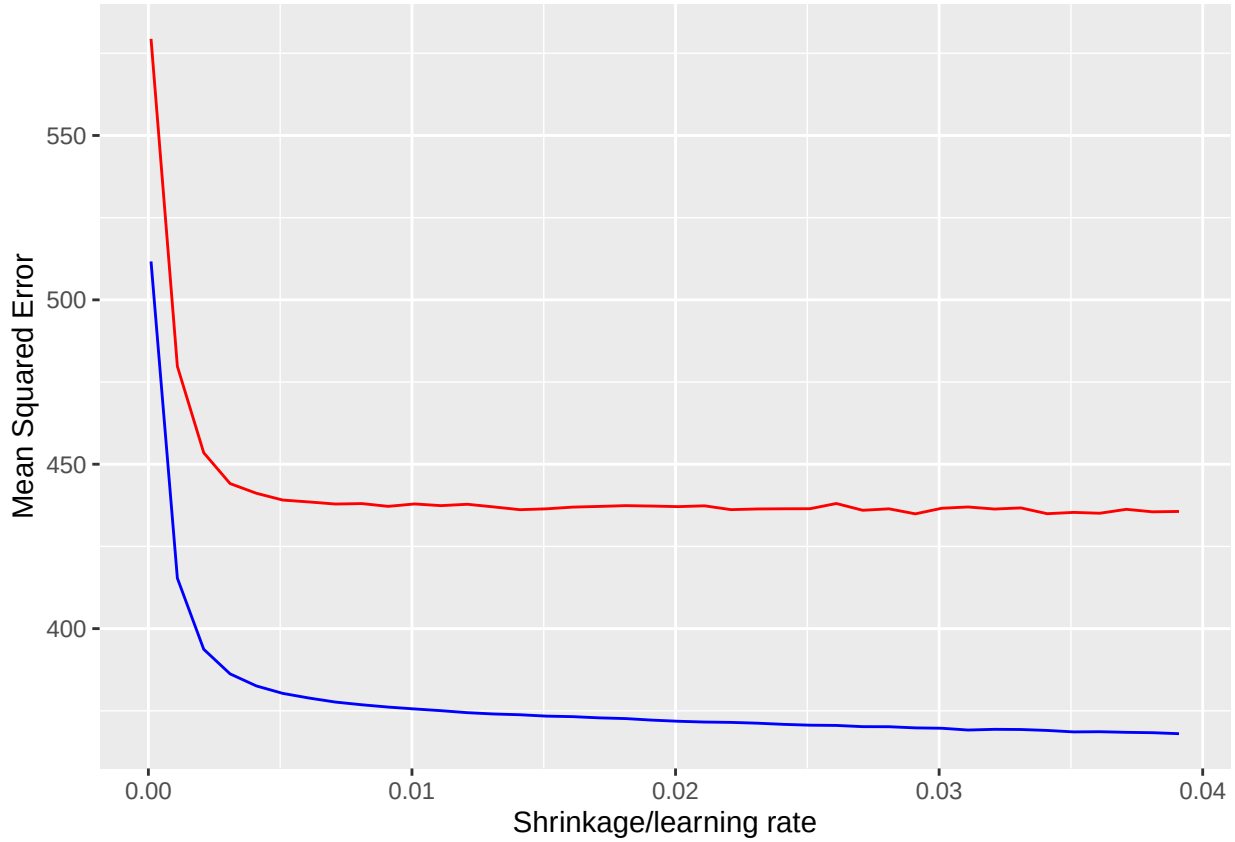
3. Training set and test set MSE across shrinkage values

```r
set.seed(1)
train.mse <- list()
test.mse <- list()
for (l in lambda){
  boost.tree=gbm(biden~female+age+educ+dem+rep,data=train,
                 distribution="gaussian",n.trees=1000, shrinkage=l)

  biden.train.pred <- predict(boost.tree,newdata=train, n.trees=1000)
  tr.mse <- mean((biden.train.pred - train$biden)^2)
  train.mse <- c(train.mse, tr.mse)

  biden.test.pred <- predict(boost.tree,newdata=test, n.trees=1000)
  te.mse <- mean((biden.test.pred - test$biden)^2)
  test.mse <- c(test.mse, te.mse)
}

mse.data <- data.frame(cbind(lambda, train.mse=unlist(train.mse),
                             test.mse=unlist(test.mse)))
ggplot(data = mse.data, aes(x=lambda)) +
  geom_line(aes(y = train.mse), color = "blue") +
  geom_line(aes(y = test.mse), color = "red") +
  xlab('Shrinkage/learning rate') +
  ylab('Mean Squared Error')
```



The blue line is the training set, and the red line is the test set.

4. Setting lambda = 0.01

```
set.seed(1)
boost.tree2=gbm(biden~female+age+educ+dem+rep,
                data=train,distribution="gaussian",
                n.trees=1000, shrinkage=0.01)
biden.boost.pred2 <- predict(boost.tree2,newdata=test, n.trees=1000)
te.mse2 <- mean((biden.boost.pred2 - test$biden)^2)
te.mse2
[1] 437.6405
```

The test MSE is 437.6405 by setting learning rate as 0.01. We can find the correpsonding point in the above graph. It sits in the flat part of the line, suggesting that the test MSE is insensitive to the change of learning rate.

5. Bagging

```
library(randomForest)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'
The following object is masked from 'package:ggplot2':

    margin
set.seed(1)
bag <- randomForest(biden~., data = train, ntree = 1000, mtry = p)
biden.bag.pred <- predict(bag, newdata=test, n.trees=1000)
te.mse3 <- mean((biden.bag.pred - test$biden)^2)
te.mse3
[1] 526.0503
```

The test MSE is 526.0503 for bagging.

6. Random Forest

```
set.seed(1)
rf <- randomForest(biden~., data = train, ntree = 1000, mtry = sqrt(p))
biden.rf.pred <- predict(boost.tree, newdata=test, n.trees=1000)
te.mse4 <- mean((biden.rf.pred - test$biden)^2)
te.mse4
[1] 435.6413
```

The test MSE is 435.6413 for the random forest by setting mtry = sqrt(p).

7. Linear Regression

```
lm.fit <- lm(biden~., data = train)
biden.lm.pred <- predict(lm.fit, newdata=test)
te.mse5 <- mean((biden.lm.pred - test$biden)^2)
te.mse5
[1] 437.1178
```

The test MSE is 437.1178.

8. Compare test errors The test MSEs are summarized as follows.

```
summary <- rbind(boosted.tree=te.mse2, bagging=te.mse3,
                 random.forest=te.mse4, linear.reg=te.mse5)
colnames(summary) <- c("test.MSE")
summary
              test.MSE
boosted.tree  437.6405
bagging       526.0503
random.forest 435.6413
linear.reg    437.1178
```

1) Bagging is the worst: As we can see, bagging performs the worst. This is probably due to the fact that in bagging, we grow highly correlated trees and thus averaging many correlated trees doesn't substantially reduce variance. But for random forest, since we are only considering a subset of all possible precitors in a single split, we decorrelate the trees in this way. Thus, averaging uncorrelated trees should see a substantial reduction in variance. It's hard to compare boosted trees, linear regressions, and random forest in this case, since their test MSEs don't differ by much. The results might be subject to our split of the training and the test set.

2) Random forest vs. Boosted trees: In general, if we compare boosted trees and random forests, random forests builds each tree independently while gradient boosting builds one tree at a time. Random forests combine results at the end of the process (by taking average) while gradient boosting combines results along the way. If we tune all the three parameters carefully for boosting carefully, boosted trees in general outperform random forests if we don't have a lot of noice in the data.

3) Linear regression: The main difference between the linear regression and the tree models is the flexibility. In linear regression, we restrict a linear relationship between the response and the predictors. If the true relationship is linear, than linear regression should perform the best. Tree models might suffer overfitting due to their high flexibility. But if the relationship is rather complex, allowing higher flexibility might reduce bias. Simply put, linear regressions might have high bias with low variance while tree models might have low bias and large variance. We need to balance the bias variance tradoff in unclear situations.

Q2 Support Vector Machines

1. Create a training set with a random sample of size 800

```
library(ISLR)
set.seed(100)
train.index <- sample(1:nrow(OJ), size=800, replace=FALSE)
tr <- OJ[train.index, ]
te <- OJ[-train.index,]
```

2. Fit a support vector classifier to the training data with cost = 0.01

```
attach(OJ)
library(e1071)
svm.fit=svm(Purchase~., data=tr, kernel="linear", cost=0.01, scale=FALSE)
summary(svm.fit)
```

```
Call:
svm(formula = Purchase ~ ., data = tr, 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
```

There were 623 support vectors, 312 in one class and 311 in the other.

3. Confusion matrix

```
#Train confusion
tr.confusion <- table(true = tr$Purchase,
                      pred = predict(svm.fit,
                                     newdata = tr))
tr.err <- (tr.confusion[1,2]+tr.confusion[2,1])/sum(tr.confusion)
tr.confusion; tr.err
    pred
true  CH  MM
  CH 466  22
  MM 177 135
[1] 0.24875
```

We can see from the above that the training error rate is 0.2488.

```
#Test confusion
te.confusion <- table(true = te$Purchase,
                      pred = predict(svm.fit,
                                     newdata = te))
te.err <- (te.confusion[1,2]+te.confusion[2,1])/sum(te.confusion)
te.confusion; te.err
    pred
true  CH  MM
  CH 157   8
  MM  63  42
[1] 0.262963
```

We can see from the above that the test error rate is 0.2630.

4. Tuning parameter

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

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.16875

- Detailed performance results:
   cost   error dispersion
1 1e-02 0.17500 0.04639804
2 1e-01 0.16875 0.03875224
3 1e+00 0.17375 0.04101575
4 1e+01 0.17000 0.03782269
5 1e+02 0.17125 0.04332131
6 1e+03 0.17250 0.04199868

tuned_model <- tune_c$best.model
summary(tuned_model)

Call:
best.tune(method = svm, train.x = Purchase ~ ., data = tr, ranges = list(cost = c(0.01,
    0.1, 1, 10, 100, 1000)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  343

 ( 172 171 )


Number of Classes:  2

Levels:
 CH MM
```

As we can see, the optimal cost is 0.1.

5. Confusion matrix and error rates using cost = 0.1

```
#Train confusion
tr.optimal.confusion <- table(true = tr$Purchase,
                              pred = predict(tuned_model,
                                             newdata = tr))
tr.optimal.err <- (tr.optimal.confusion[1,2]+tr.optimal.confusion[2,1])/sum(tr.optimal.confusion)
tr.optimal.confusion; tr.optimal.err
    pred
true  CH  MM
  CH 427  61
  MM  63 249
[1] 0.155
```

```
#Test confusion
te.optimal.confusion <- table(true = te$Purchase,
                              pred = predict(tuned_model,
                                             newdata = te))
te.optimal.err <- (te.optimal.confusion[1,2]+te.optimal.confusion[2,1])/sum(te.optimal.confusion)
te.optimal.confusion; te.optimal.err
    pred
true  CH  MM
  CH 140  25
  MM  27  78
[1] 0.1925926
```

Using cost = 0.1, we get the training set error rate as 0.1550, and the test set error rate as 0.1926. Both of the error rates our lower than those of SVM with cost = 0.01. Therefore, out tuned model outperforms the model with arbitrarily chosen cost. Tuning in general will yield more reliable results. In general, a small cost creates a large margin and allows more misclassifications. On the other hand, a large cost creates a narrow margin and permits fewer misclassifications. But a too large cost might yield overfitting.