Machine Learning PSET 3

Krish Suchak

Decision Trees

1. Setup

```r
set.seed(45)
survey <- read.csv('nes2008.csv')
feats <- names(survey)[-1]
p <- length(feats)
lambda <- seq(0.0001, 0.04, 0.001)
```

2. Train and Test Splits

```r
samples <- sample(nrow(survey), .75 * nrow(survey), replace = FALSE)
train <- survey[samples, ]
test <- survey[-samples, ]
```

3. Plot Train/Test MSEs after Boosting

```r
library(gbm)
```

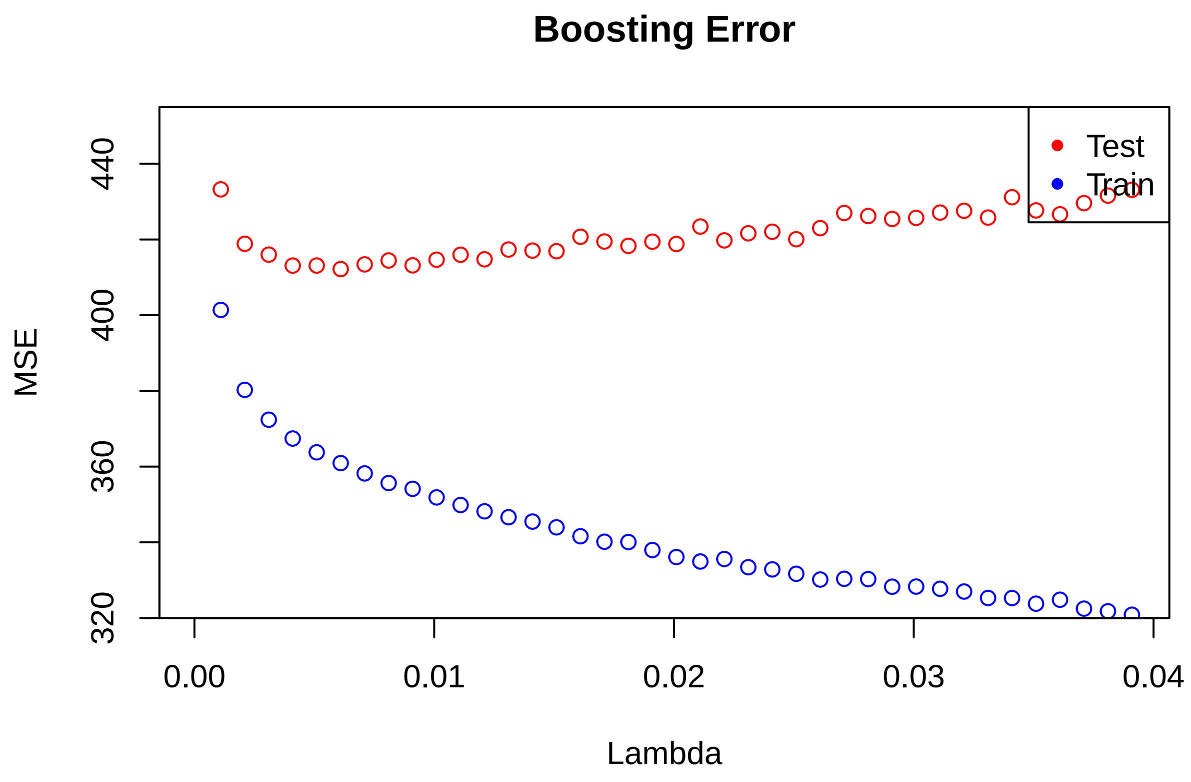
```
## Loaded gbm 2.1.5
```

```r
train_mse <- c()
test_mse <- c()

for (l in lambda) {
  boost <- gbm(biden ~ .,
               data = train,
               distribution = "gaussian",
               n.trees = 1000,
               shrinkage = l,
               interaction.depth = 4)
  train_preds <- predict(boost, newdata = train, n.trees = 1000)
  test_preds <- predict(boost, newdata = test, n.trees = 1000)
  train_mse <- append(train_mse, mean((train_preds - train$biden)^2))
  test_mse <- append(test_mse, mean((test_preds - test$biden)^2))

}
```

```r
plot(lambda, train_mse, col = 'blue',
     ylab = 'MSE', xlab = 'Lambda', main = 'Boosting Error', ylim = c(325, 450))
points(lambda, test_mse, col = 'red')
legend(x='topright', legend=c('Test', 'Train'), col=c('red', 'blue'), pch = 20)
```



4. MSE at lambda = 0.01

```
boost <- gbm(biden ~ .,
             data = train,
             distribution = "gaussian",
             n.trees = 1000,
             shrinkage = 0.01,
             interaction.depth = 4)
pred <- predict(boost, newdata = test, n.trees = 1000)
mse <- mean((pred - test$biden)^2)
mse
```

```
## [1] 416.5368
```

```
min(test_mse)
```

```
## [1] 412.1839
```

These values are quite comparable (test MSE at lambda = 0.01 and minimum test MSE value in boosting). La

5. Bagging

```
library(ipred)
bagg <- bagging(biden ~ .,
                data = train)
```

```
pred <- predict(bagg, newdata = test)
mse = mean((pred - test$biden)^2)
mse
```

```
## [1] 418.6089
```

This test MSE is larger than the one from previous parts (less accurate model).

6. Random Forest

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
rf <- randomForest(biden ~ ., data = train)
pred <- predict(rf, newdata = test)
mse <- mean((pred - test$biden)^2)
mse
```

```
## [1] 421.5164
```

This test MSE is similar to the one achieved through bagging.

7. Linear Regression

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

```
## [1] 416.2959
```

8. Model Comparison

The linear regression yielded a test MSE smaller than the bagging and random forest models but larger than a boosted model with a low learning rate. Therefore, the boosted model is the best approach.

SVMs

1. Setup

```
library(ISLR)
oj <- OJ
oj$Purchase <- as.factor(oj$Purchase)
set.seed(45)
samples <- sample(nrow(oj), 800, replace = FALSE)
train <- oj[samples,]
test <- oj[-samples,]
```

2. SVM Classifier

```r
library(e1071)
clf <- svm(Purchase ~ .,
           data = train,
           kernel = 'linear',
           cost = 0.01)
summary(clf)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  434
##
##  ( 216 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The SVM model has 434 support vectors with 216 belonging to Citrus Hill and 211 belonging to Minute Maid. We set our cost value to 0.01, possibly resulting in the large number of support vectors. Our two classifications are "CH" and "MM" (possible values of the response variable "Purchase").

3. Confusion Matrices

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin
```

```r
confusionMatrix(data=predict(clf, train), reference=train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 427  74
##          MM  62 237
##
##                Accuracy : 0.83
##                  95% CI : (0.8021, 0.8554)
##     No Information Rate : 0.6112
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6398
##
##  Mcnemar's Test P-Value : 0.3456
##
##             Sensitivity : 0.8732
##             Specificity : 0.7621
##          Pos Pred Value : 0.8523
##          Neg Pred Value : 0.7926
##              Prevalence : 0.6112
##          Detection Rate : 0.5337
##    Detection Prevalence : 0.6262
##       Balanced Accuracy : 0.8176
##
##        'Positive' Class : CH
##
```

```
    confusionMatrix(data=predict(clf, test), reference=test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 147  33
##         MM  17  73
##
##                Accuracy : 0.8148
##                  95% CI : (0.7633, 0.8593)
##     No Information Rate : 0.6074
##     P-Value [Acc > NIR] : 1.693e-13
##
##                   Kappa : 0.6011
##
##  Mcnemar's Test P-Value : 0.03389
##
##             Sensitivity : 0.8963
##             Specificity : 0.6887
##          Pos Pred Value : 0.8167
##          Neg Pred Value : 0.8111
##              Prevalence : 0.6074
##          Detection Rate : 0.5444
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 0.7925
##
##        'Positive' Class : CH
##
```

Train error rate = 100% - 83.0% = 17.0% Test error rate = 100% - 81.5% = 18.5%

The small difference in train and test errors imply that our SVM model minimize both bias and variance well.

4. Optimal Cost Value

```
library(e1071)
tuned <- tune(svm,
              Purchase ~ .,
              data = train,
              kernel = 'linear',
              ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 250, 500, 1000)))
tuned_model <- tuned$best.model
summary(tuned_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 250, 500, 1000)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  339
##
##  ( 169 170 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The optimal cost value is 1 (according to the tuner), yielding a SVM classifier with 339 support vectors.

5. Performance of Optimal Model

```
confusionMatrix(data=predict(tuned_model, train), reference=train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 426  72
##         MM  63 239
##
##                Accuracy : 0.8312
##                  95% CI : (0.8035, 0.8566)
##     No Information Rate : 0.6112
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.643
##
##  Mcnemar's Test P-Value : 0.4911
```

```
##
##             Sensitivity : 0.8712
##             Specificity : 0.7685
##          Pos Pred Value : 0.8554
##          Neg Pred Value : 0.7914
##              Prevalence : 0.6112
##          Detection Rate : 0.5325
##    Detection Prevalence : 0.6225
##       Balanced Accuracy : 0.8198
##
##        'Positive' Class : CH
##
```

```
confusionMatrix(data=predict(tuned_model, test), reference=test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 148  29
##         MM  16  77
##
##                Accuracy : 0.8333
##                  95% CI : (0.7834, 0.8758)
##     No Information Rate : 0.6074
##     P-Value [Acc > NIR] : 6.513e-16
##
##                   Kappa : 0.6428
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 0.9024
##             Specificity : 0.7264
##          Pos Pred Value : 0.8362
##          Neg Pred Value : 0.8280
##              Prevalence : 0.6074
##          Detection Rate : 0.5481
##    Detection Prevalence : 0.6556
##       Balanced Accuracy : 0.8144
##
##        'Positive' Class : CH
##
```

Train error rate = 100% - 83% = ~17% Test error rate = 100% - 83% = ~17%

Our new model with an optimal cost value of 1 yields lower train and test errors than the model with a cost value of 0.01 and negligible difference in train and test errors imply that our SVM model minimize both bias and variance well (even better than the previous model).