Homework 3

Aaron Politsky

October 15, 2015

9.1

Load libraries

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
## Loading required package: survival
## Loading required package: lattice
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
```

I will be parallelizing to save some time, using my machine's 4 cores when possible.

```
## Loading required package: iterators
```

Let's import the data and split it into train, validation, and test sets:

```
# import the data
used.cars <- read.csv("UsedCars.csv")

# Split the data into train, validate, and test subsets.
n <- nrow(used.cars)
n1 <- floor(n/2)
n2 <- floor(n/4)
n3 <- n - n1 - n2

set.seed(14) # Choosing 14 for consistency with slides
ii <- sample(1:n, n)
train <- used.cars[ii[1:n1],]
val   <- used.cars[ii[n1+1:n2],]
test  <- used.cars[ii[n1+n2+1:n3],]
```

Using only Mileage and Year

Trees

```
#################
# Single Trees
big.mileage.year.tree <-
  rpart(price ~ mileage + year, data=train, method="anova",
        control=rpart.control(minsplit = 5, cp = 0.00005))
```

```
index.of.best.cp <- which.min(big.mileage.year.tree$cptable[,"xerror"])
best.cp <- big.mileage.year.tree$cptable[index.of.best.cp, "CP"]
best.size <- big.mileage.year.tree$cptable[index.of.best.cp, "nsplit"] + 1

# prune to best tree size
mileage.year.tree <- prune(big.mileage.year.tree, cp=best.cp)

# Prediction
pred.val.tree.mileage.year <- predict(mileage.year.tree, newdata=val)
```

Random Forests

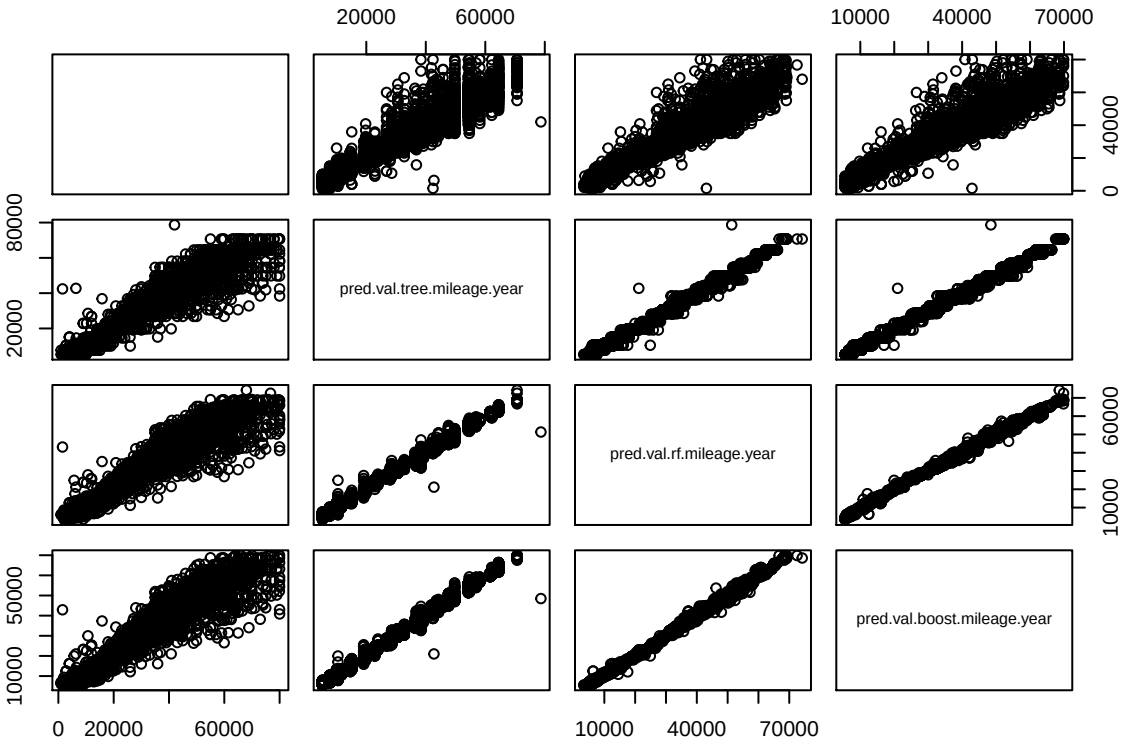
```
# try out different parameters
m.try <- c(1,2) # random forests, bagging
n.tree <- c(100, 500)

# create a data frame of each combo of these options
random.forest.params <- expand.grid(data.frame(cbind(m.try, n.tree)))

train.mileage.year <- train[,c("price", "mileage", "year")]
val.mileage.year <- val[,c("price", "mileage", "year")]

# parallelize the plyr loop
rf.list.mileage.year <-
  # m.try is only valid for 1 or 2 here
  dlply(random.forest.params, .(m.try, n.tree), function(x) {
    randomForest(price ~ mileage + year, data=train.mileage.year,
                 mtry=x$m.try, ntree=x$n.tree,
                 xtest=val.mileage.year[,c("mileage", 'year')], ytest=val$price)
  }, .parallel=T)


# generate in-bag and oob loss: rmse
rf.perf.mileage.year <-
  ldply(rf.list.mileage.year, c(
    # here we make use of the oob predicted values generated in our forest call
    olrf=function(rf) {sqrt(mean((val$price - rf$test$predicted)^2))},
    ilrf=function(rf) {sqrt(mean((train$price - rf$predicted)^2))}
  ))

# which settings were the best?
best.rf.perf.mileage.year <- rf.perf.mileage.year[which.min(rf.perf.mileage.year$olrf),]

# the best forest object
best.rf.mileage.year <- rf.list.mileage.year[[which.min(rf.perf.mileage.year$olrf)]]

# predict:
pred.val.rf.mileage.year <- best.rf.mileage.year$test$predicted
```

Boosting

```
depth <- c(4,10)
n.tree <- c(1000, 4000)
lambda <- c(.1, .001)
boost.params <- expand.grid(data.frame(cbind(n.tree, lambda, depth)))

boost.list.mileage.year <-
  dlply(boost.params, .(depth, n.tree, lambda), .parallel=T, function(x) {
    gbm(price ~ mileage + year,
        data=train.mileage.year,
        distribution="gaussian",
        interaction.depth=x$depth,
        n.trees=x$n.tree,
        shrinkage=x$lambda)
  })

# evaluate using min RMSE
boost.perf.mileage.year <-
  ldply(boost.list.mileage.year, c(
    ilb = function(b) {sqrt(mean((train$price - b$fit)^2))},
    olb = function(b) {sqrt(mean((val$price -
                                    predict(b,
                                            newdata=val.mileage.year,
                                            n.trees=b$n.trees))^2))}
  ))

# best settings
best.boost.perf.mileage.year <-
  boost.perf.mileage.year[which.min(boost.perf.mileage.year$olb),]

# best model
best.boost.mileage.year <-
  boost.list.mileage.year[[which.min(boost.perf.mileage.year$olb)]]

# predict
pred.val.boost.mileage.year <- predict(best.boost.mileage.year,
                                       newdata=val.mileage.year,
                                       n.trees=best.boost.mileage.year$n.trees)
```

Compare fits

```
# compare all models
pairs(cbind(val$price,
            pred.val.tree.mileage.year,
            pred.val.rf.mileage.year,
            pred.val.boost.mileage.year))
```




```
cor(cbind(val$price,
          pred.val.tree.mileage.year,
          pred.val.rf.mileage.year,
          pred.val.boost.mileage.year))
```

```
##                                   pred.val.tree.mileage.year
##                             1.0000000                  0.9516598
## pred.val.tree.mileage.year  0.9516598                  1.0000000
## pred.val.rf.mileage.year    0.9546822                  0.9960494
## pred.val.boost.mileage.year 0.9557101                  0.9964157
##                             pred.val.rf.mileage.year
##                                            0.9546822
## pred.val.tree.mileage.year                 0.9960494
## pred.val.rf.mileage.year                   1.0000000
## pred.val.boost.mileage.year                0.9988047
##                             pred.val.boost.mileage.year
##                                               0.9557101
## pred.val.tree.mileage.year                    0.9964157
## pred.val.rf.mileage.year                      0.9988047
## pred.val.boost.mileage.year                   1.0000000
```

That was nice, but let's get to it with all covariates:

Fitting models using all of our covariates

Trees

```
big.tree <-
  rpart(price ~ ., data=train, method="anova",
        control=rpart.control(minsplit = 5, cp = 0.00005))

index.of.best.cp <- which.min(big.tree$cptable[,"xerror"])
best.cp <- big.tree$cptable[index.of.best.cp, "CP"]
best.size <- big.tree$cptable[index.of.best.cp, "nsplit"] + 1

# prune to best tree size
tree <- prune(big.tree, cp=best.cp)

# Predict
pred.val.tree <- predict(tree, newdata=val)
```

Random Forests

```
# Forests & Bagging

# try out different parameters
m.try <- c(floor(sqrt(ncol(used.cars)-1)), ncol(used.cars)-1) # random forests, bagging
n.tree <- c(100, 500)
# create a data frame of each combo of these options
random.forest.params <- expand.grid(data.frame(cbind(m.try, n.tree)))

# for each combo of options, fit trees on train while validating on val
rf.list <-
  dlply(random.forest.params, .(m.try, n.tree), function(x) {
    randomForest(price ~ ., data=train,
                 mtry=x$m.try, ntree=x$n.tree,
                 xtest=val[,names(val)!="price"], ytest=val$price)
  }, .parallel=T)

# generate in-bag and oob loss: rmse
rf.perf <-
  ldply(rf.list, c(
    olrf=function(rf) {sqrt(mean((val$price - rf$test$predicted)^2))},
    ilrf=function(rf) {sqrt(mean((train$price - rf$predicted)^2))}
  ))

best.rf.perf <- rf.perf[which.min(rf.perf$olrf),]
print(best.rf.perf)
```

```
##   m.try n.tree     olrf     ilrf
## 2     3    500 4152.532 4177.288
```

```
best.rf <- rf.list[[which.min(rf.perf$olrf)]]

# OOB Predictions
pred.val.rf <- best.rf$test$predicted
```

Boosting

```
depth <- c(4,10)
n.tree <- c(1000, 4000)
lambda <- c(.1, .001)
boost.params <- expand.grid(data.frame(cbind(n.tree, lambda, depth)))

boost.list <-
  dlply(boost.params, .(depth, n.tree, lambda), .parallel=T, function(x) {
    gbm(price ~ .,
        data=train,
        distribution="gaussian",
        interaction.depth=x$depth,
        n.trees=x$n.tree,
        shrinkage=x$lambda)
  })

# evaluate performance
boost.perf <-
  ldply(boost.list, c(
    ilb = function(b) {sqrt(mean((train$price - b$fit)^2))},
    olb = function(b) {sqrt(mean((val$price -
                                  predict(b,
                                          newdata=val,
                                          n.trees=b$n.trees))^2))}
  ))

best.boost <- boost.list[[which.min(boost.perf$olb)]]
best.boost.perf <- boost.perf[which.min(boost.perf$olb),]

# Predict using
pred.val.boost <- predict(best.boost, newdata=val, n.trees=best.boost$n.trees)

best.boost <- boost.list[[which.min(boost.perf$olb)]]
best.boost.perf <- boost.perf[which.min(boost.perf$olb),]

# Predict using
pred.val.boost <- predict(best.boost, newdata=val, n.trees=best.boost$n.trees)
```

And for fun, Multiple Regression

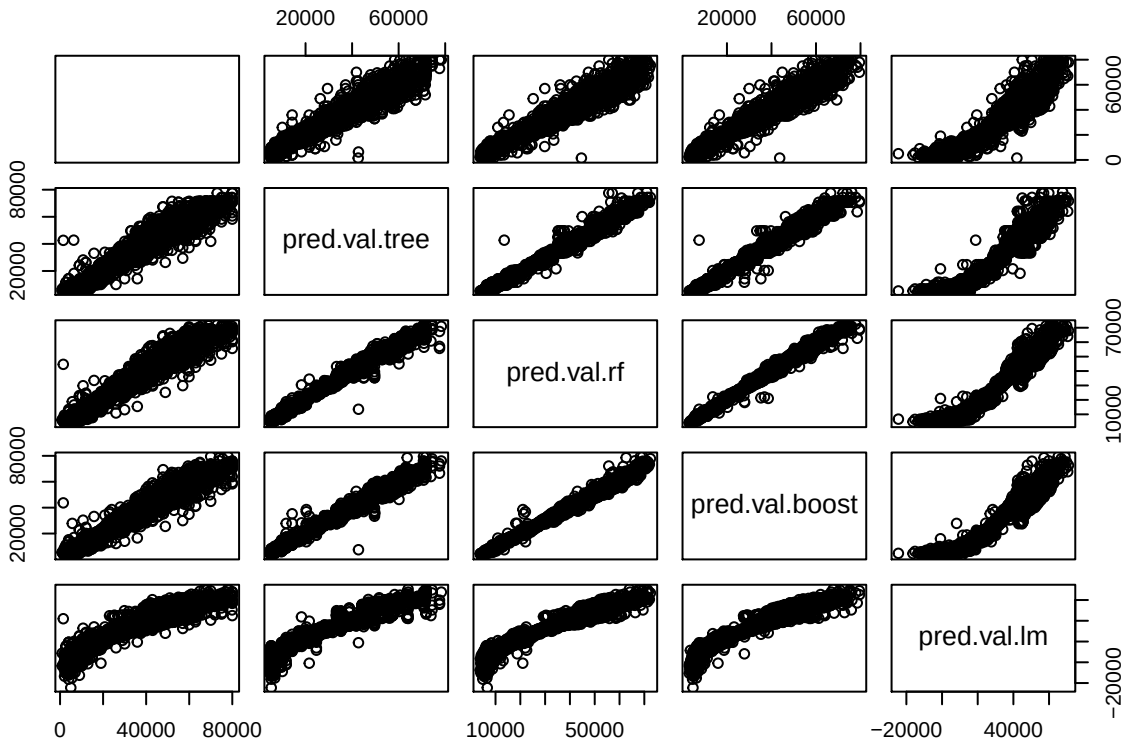
```
# Multiple Regression
lm.fit <- lm(price ~ ., data=train)
lm.in.err <- sqrt(mean((train$price - lm.fit$fitted.values)^2))
pred.val.lm <- predict.lm(lm.fit, newdata=val)
```

```
## Warning in predict.lm(lm.fit, newdata = val): prediction from a rank-
## deficient fit may be misleading
```

```
lm.out.err <- sqrt(mean((val$price - pred.val.lm)^2))
```

Compare our Models

```
pairs(cbind(val$price, pred.val.tree, pred.val.rf, pred.val.boost, pred.val.lm))
```




```
cor(cbind(val$price, pred.val.tree, pred.val.rf, pred.val.boost, pred.val.lm))
```

```
##                          pred.val.tree pred.val.rf pred.val.boost
##                1.0000000     0.9691736   0.9737190      0.9742146
## pred.val.tree  0.9691736     1.0000000   0.9929692      0.9911924
## pred.val.rf    0.9737190     0.9929692   1.0000000      0.9963349
## pred.val.boost 0.9742146     0.9911924   0.9963349      1.0000000
## pred.val.lm    0.9499203     0.9637120   0.9725462      0.9698697
##                pred.val.lm
##                  0.9499203
## pred.val.tree    0.9637120
## pred.val.rf      0.9725462
## pred.val.boost   0.9698697
## pred.val.lm      1.0000000
```

I like the Random Forest model, because it produces a pretty good fit, and I can parallelize its growth.

Retrain the chosen model on combined Train + Validation set

```
##############################
#  Now let's combine the train and validation set and test our chosen methods
train.val <- rbind(train, val)


# refit random forest
refit.rf <-
  foreach(ntree=rep(best.rf.perf$n.tree/NUM.CORES, NUM.CORES),
          .combine=combine,
          .multicombine=T,
          .packages='randomForest') %dopar%
  randomForest(price ~ ., data=train.val, mtry=best.rf.perf$m.try, ntree=ntree,
               xtest=test[,names(test) != "price"], ytest=test$price)

# Our out of sample RMSE is:
oob.rmse <- sqrt(mean((test$price - refit.rf$test$predicted)^2))
```

Our out of sample RMSE for this model is 4246.3127028.