Weight Lifting Exercies

Dan Allbaugh

October 14, 2019

Executive Summary

Machine learning attempts to predict body movement from accelorometers while performing bicep curls to determine how well the user is executing the movement.

Loading packages

Loading in all package dependencies.

```
suppressMessages(library(caret))
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
suppressMessages(library(dplyr))
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
suppressMessages(library(ggplot2))
suppressMessages(library(parallel))
suppressMessages(library(doParallel))
```

```
## Warning: package 'doParallel' was built under R version 3.5.3
```

```
## Warning: package 'foreach' was built under R version 3.5.3
```

```
## Warning: package 'iterators' was built under R version 3.5.3
```

```
suppressMessages(library(gridExtra))
```

```
## Warning: package 'gridExtra' was built under R version 3.5.3
```

Data Processing

Data for this article comes from Weight Lifting Exercies Dataset (http://groupware.les.inf.puc-rio.br/har).

First we store the URL for training and testing .csv files. Then look to see if a data directory exists and create one if not. Next we check if the .csv files have already been downloaded and files already exist. If these things are not done the code will execute. Finally testing and training variables are created for analysis.

```
setwd("C:/Users/da13287/Desktop/RStudio Tests/Session 8/homework3/Weight_Liftin
g")
trainurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.c
sv"
testurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.cs
v"
if(!dir.exists("./data")) {dir.create("./data")}
trainfile <- paste0(getwd(),"/data/","pml_training.csv")
testfile <- paste0(getwd(),"/data/","pml_testing.csv")
if(!file.exists(trainfile)) {download.file(trainurl, trainfile)}
if(!file.exists(testfile)) {download.file(testurl, testfile)}
training <- read.csv(trainfile)
testing <- read.csv(testfile)
```

Exploring Data

Below is the code used to explore the data. Results are hidden because of the output length. What code reveals is the training set consists of 19,622 rows and 160 columns. There are 1,287,472 ‘NA’ values. In each column with an ‘NA’ it appears consistently 19,216 times. The testing set has 20 rows and 160 columns. ‘NA’ appears 2,000 times and each time it appears in a column it appears 20 times. Given the proportion of NAs for columns in which they appear I have removed all columns with ‘NAs’ in both data sets which left 55 dependent variables.

```r
dim(training)
dim(testing)
str(training)
str(testing)
#removing columns 1:5 which includes row number, user name and timestamp which
won't be relevant
#depdendent variables
training <- training[, 6:length(training)]
testing <- testing[, 6:length(testing)]
#count NAs
table(is.na (training)); table(is.na (testing))
#counts NAs column wise
sapply(training, function(x) sum(is.na(x)))
sapply(testing, function(x) sum(is.na(x)))
#doing this shows that NAs appears in consistent numbers throughout columns so
plan to remove
#those columns

#creating data frame to see consistency of NAs between training and testing set
s
traindf <- data.frame(sapply(training, function(x) sum(is.na(x))))
testdf <- data.frame(sapply(testing, function(x) sum(is.na(x))))
dfcombine <-cbind(names(training),traindf, testdf)
dfcombine <- mutate(dfcombine, total = dfcombine$sapply.training..function.x..s
um.is.na.x... + dfcombine$sapply.testing..function.x..sum.is.na.x...)
#left only with columns that do not contain any NAs in either training or testi
ng set
dfcombinesub <- subset(dfcombine, total ==0)[,1]
dfcombinesub <- as.character(dfcombinesub)
#creating new training and testing variables keeping only columns which have n
o NAs
#keeping the 'classe' column in training set; no 'classe' column in testing
training <- training[, dfcombinesub]
#change all non factor variables into numeric to have consistency between train
ing and testing set
training <- training %>% mutate_at(vars(1:54), as.numeric)
testingoriginal <- testing[, dfcombinesub[1:length(dfcombinesub)-1]]
#change all non factor variables into numeric to have consistency between train
ing and testing set
testingoriginal <- testingoriginal %>% mutate_at(vars(1:54), as.numeric)
#testingoriginal <- mutate(testingoriginal, classe=as.factor(""))
```

Creating Training, Testing and Validation Models

The original test data does not have classe column because evenutally we will use this data to predict the classe based on the dependent variables. To build and validate models we need to split the original training data into a testing, training and validation group to measure various model accuracies.

```
set.seed(13)
#build the data set with independent variable as classe
inbuild <- createDataPartition(y=training$classe, p=0.7, list=FALSE)
#set aside a validation set
validation <- training[-inbuild, ]
builddata <- training[inbuild,]
#use the remaining build data to create a testing a training set
intrain <- createDataPartition(y=builddata$classe, p=0.7, list=FALSE)
training <- builddata[intrain,]
testing <- builddata[-intrain,]
```

After splitting the original training set we are left with 9,619 obersations for training, 4,188 for testing and 5,885 for validation.

Build Machine Learning Models

Now we are able to build various machine learning models and test accuracy. Model training can be computationaly expensive, particularly with randomized and iterative methods such as random forest and boosting. To speed up performance we will be using a parallel processing with the techinque outlined here (https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-randomForestPerformance.md). To begin parallel processing we first will configure parallel processing and then configuring a trainControl object.

```
set.seed(13)
# set up training run for x / y syntax because model format performs poorly
x <- training[,-55]
y <- training[,55]
#configure parallel processing
cluster <- makeCluster(detectCores()-1)# convention to leave 1 core for OS
registerDoParallel(cluster)
# configure trainControl object
fitControl <- trainControl(method="cv", number=5, allowParallel = TRUE)
```

Random Forest

Using this approach we acheive and accuracy of 99.4%.

```
#develop training model
modfitrf <- train(x,y, method="rf", trControl=fitControl)
predrf <- predict(modfitrf, newdata=testing)
confusionMatrix(testing$classe, predrf)$overall[1]
```

```
##  Accuracy
## 0.9936863
```

```
confusionMatrix(testing$classe, predrf)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1169    1    0    0    1
##          B    4  790    3    0    0
##          C    0    6  712    0    0
##          D    0    0    8  667    0
##          E    0    0    0    3  754
```

Boosting

Using this approach we acheive and accuracy of 98.3%.

```
#develop training model
modfitboost <- train(x,y, method="gbm", trControl=fitControl, verbose=FALSE)
predboost <- predict(modfitboost, newdata=testing)
confusionMatrix(testing$classe, predboost)$overall[1]
```

```
##  Accuracy
## 0.9844585
```

```
confusionMatrix(testing$classe, predboost)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1166    5    0    0    0
##          B   11  781    5    0    0
##          C    0    7  707    4    0
##          D    0    3   11  660    1
##          E    0    5    2   10  740
```

Linear Discriminant Analysis

Using this approach we acheive and accuracy of 71.4%.

```
#develop training model
modfitlda <- train(classe~., method="lda", data=training, trControl=fitControl)
predlda <- predict(modfitlda, newdata=testing)
confusionMatrix(testing$classe, predlda)$overall[1]
```

```
##  Accuracy
## 0.7139388
```

```
confusionMatrix(testing$classe, predlda)$table
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 988  42  65  72   4
##          B 116 518  99  27  37
##          C  69  80 484  62  23
##          D  39  29  82 501  24
##          E  38 126  77  67 449
```

Plotting predictions

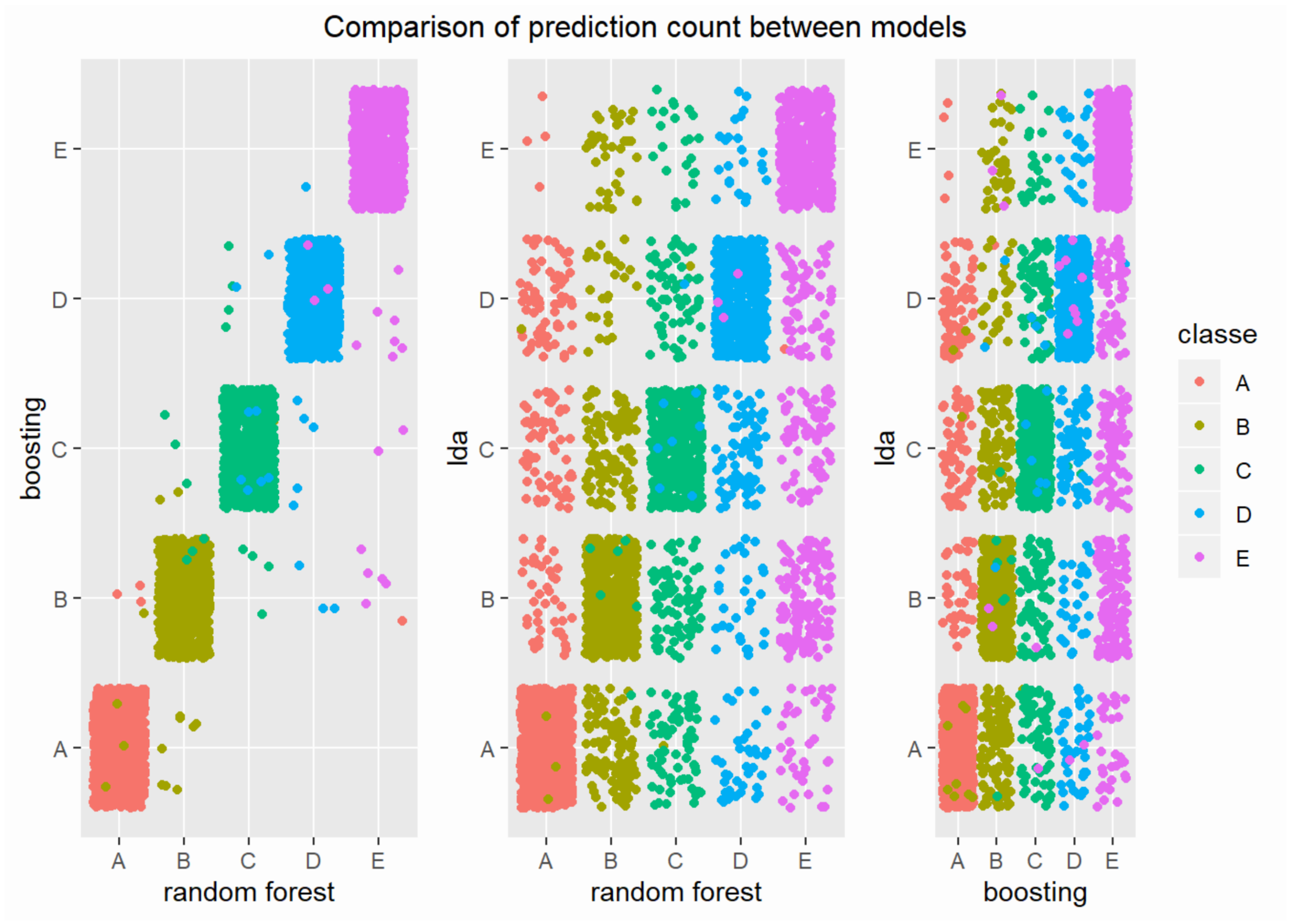
The joint accuracy of the random forest and boosting approaches is apparant when the three predicting methods are plotted against each other.

```
#compare random forest to boosting
g1 <- ggplot(testing, aes(predrf, predboost)) +
        geom_jitter(aes(col=classe), show.legend = F) +
        labs(y="boosting",
        x="random forest")

g2 <- ggplot(testing, aes(predrf, predlda)) +
    geom_jitter(aes(col=classe), show.legend = F) +
    labs(y="lda",
         x="random forest")

g3 <- ggplot(testing, aes(predboost, predlda)) +
        geom_jitter(aes(col=classe), show.legend = T) +
        labs(y="lda",
        x="boosting")

grid.arrange(g1, g2, g3, nrow=1, top = "Comparison of prediction count between
models")
```



combined models

Given the accuracy of both random forest and boosting we will create a combined model which produces an accuracy of 99.4%.

```
#build a new dataset that consists of predictions from rf and boost
predDF <- data.frame(predrf, predboost, classe=testing$classe)
predx <- predDF[,-3]
predy <- predDF[, 3]
combModFit <- train(classe~., method="rf", data=predDF, trControl=fitControl)
combModPred <- predict(combModFit, newdata=testing)
confusionMatrix(testing$classe, combModPred)$overall[1]
```

```
##  Accuracy
## 0.9936863
```

```
confusionMatrix(testing$classe, combModPred)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1169    1    0    0    1
##          B    4  790    3    0    0
##          C    0    6  712    0    0
##          D    0    0    8  667    0
##          E    0    0    0    3  754
```

De-register parallel processing cluster.

Shutting down the cluster by calling the stopCluster() and registerDoSEQ() functions. The registerDoSEQ() function is required to force R to return to single threaded processing.

```
#de-register parallel processing cluster
stopCluster(cluster)
registerDoSEQ()
```

Final Validation

Considering the accuracy for the combined model was not materially better than the stand alone random forest model we are selecting random forest at the machine learning model to make predictions for the validation model previously set aside.

```
predrfvalidation <- predict(modfitrf, newdata=validation)
confusionMatrix(validation$classe, predrfvalidation)$overall[1]
```

```
##  Accuracy
## 0.9964316
```

```
confusionMatrix(validation$classe, predrfvalidation)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1670    3    0    0    1
##          B    2 1132    4    1    0
##          C    0    5 1021    0    0
##          D    0    0    3  961    0
##          E    0    0    0    2 1080
```

The model performs admirably, achieving an accuracy of 99.6% which makes us feels confident this model wil work well to predict values on the quiz.

Predictions

Using the random forest model we are now able to make predictions about the 20 cases from the original testing data to use as answers for the course project prediction quiz.

```
predrffinal <- predict(modfitrf, newdata=testingoriginal)
answers <- data.frame(predrffinal)
answers
```

```
##    predrffinal
## 1            B
## 2            A
## 3            B
## 4            A
## 5            A
## 6            E
## 7            D
## 8            B
## 9            A
## 10           A
## 11           B
## 12           C
## 13           B
## 14           A
## 15           E
## 16           E
## 17           A
## 18           B
## 19           B
## 20           B
```