Practical Machine Learning Prediction Assignment

J.Erickson

2/26/2021

Summary

This work will explore and predict how well a person is able to do a unilateral dumbbell bicep Curl. Information was taken from subjects using accelerometers on a belt, forearm, arm, and dumbell. Each subject was asked to lift the dumbell 5 ways, 4 of the ways had incorrect form. The data about which form was performed was stored in the "classe" variable. The training and test data for this project were taken from: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har.

The classes are broken down as follows: - Class A: exactly according to the specification - Class B: throwing the elbows to the front - Class C: lifting the dumbbell only halfway - Class D: lowering the dumbbell only halfway - Class E: throwing the hips to the front

Loading Data

The training and test data were downloaded and stored. Any values marked as "#DIV/0!" were replaced with NA's. The first 8 columns were also removed since this information did not provide any value for predictions.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
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
```

```
download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv','trainingdata.csv'
trainingdata <- read.csv("trainingdata.csv", sep=",", header=TRUE, na.strings=c("NA", "#DIV/0!"), strin
download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv','testingdata.csv',m
testingdata <- read.csv("testingdata.csv", sep=",", header=TRUE, na.strings=c("NA", "#DIV/0!"), strings
# Remove first 8 columns- unnecessary information
trainingdata <- trainingdata[,-(1:8)]
testingdata <- testingdata[,-(1:8)]
```

Explore/Partition

In order to create tidy data, the variables with a large portion of NA's were removed. This left 52 variables to use in the model. The data was then portioned out giving 75% of the observations to the training set and 25% to the validation set.

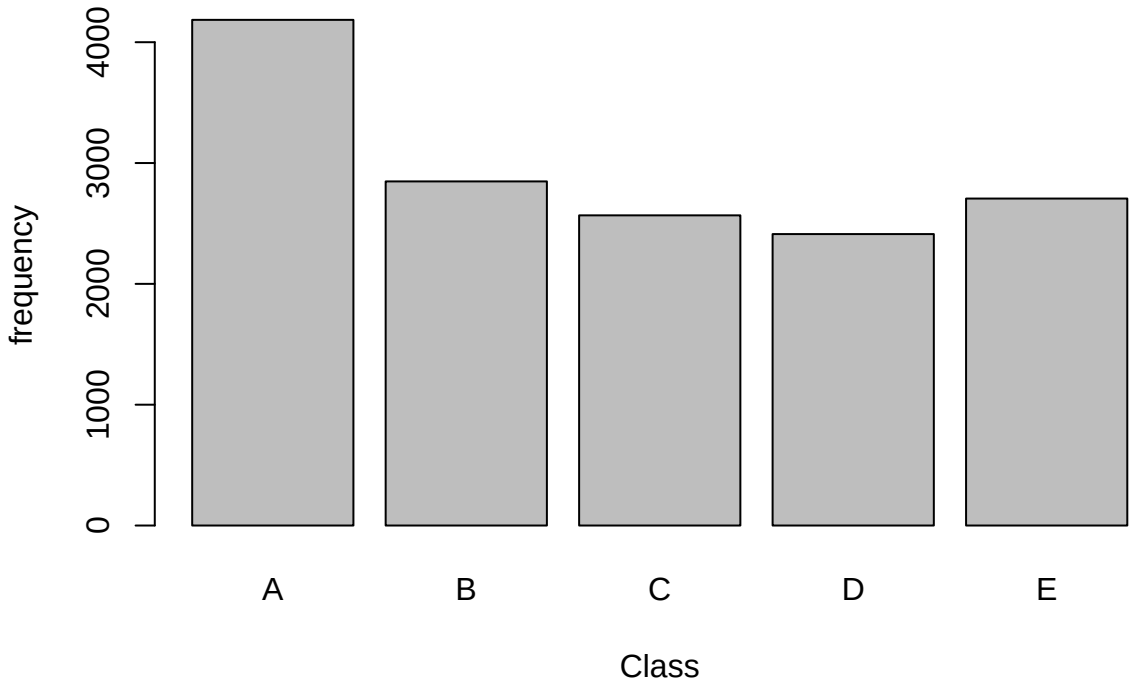
```
set.seed(12345)
# If number of na's in a column is <95% of the total number of rows, keep variable column
trainingdata <- trainingdata[ , colSums(is.na(trainingdata)) < nrow(trainingdata)*.95]
inTrain  <- createDataPartition(trainingdata$classe, p=0.75, list=FALSE)
training <- trainingdata[inTrain,]
validation <- trainingdata[-inTrain,]
# reassign to x and y to avoid slowing down caret() with model syntax
y <- training[,52]
x <- training[,-52]
str(training)
```

```
## 'data.frame':    14718 obs. of  52 variables:
##  $ pitch_belt          : num  8.07 8.07 8.07 8.07 8.06 8.16 8.17 8.2 8.21 8.2 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.03 0.02 0.02 0 ...
##  $ gyros_belt_y        : num  0 0 0 0.02 0 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 0 0 -0.02 0 ...
##  $ accel_belt_x        : int  -21 -22 -20 -21 -21 -20 -21 -22 -22 -21 ...
##  $ accel_belt_y        : int  4 4 5 2 4 2 4 4 4 2 ...
##  $ accel_belt_z        : int  22 22 23 24 21 24 22 21 21 22 ...
##  $ magnet_belt_x       : int  -3 -7 -2 -6 0 1 -3 -3 -8 -1 ...
##  $ magnet_belt_y       : int  599 608 600 600 603 602 609 606 598 597 ...
##  $ magnet_belt_z       : int  -313 -311 -305 -302 -312 -312 -308 -309 -310 -310 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -129 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22 21.7 21.6 21.4 21.4 21.4 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0 0.02 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 0 0 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0 0 -0.02 -0.02 -0.02 -0.03 -0.03 ...
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -288 -288 -287 -288 -289 ...
##  $ accel_arm_y         : int  109 110 110 111 111 109 110 111 111 111 ...
##  $ accel_arm_z         : int  -123 -125 -126 -123 -122 -122 -124 -124 -124 -124 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -374 -369 -369 -376 -372 -371 -374 ...
##  $ magnet_arm_y        : int  337 337 344 337 342 341 334 338 331 342 ...
##  $ magnet_arm_z        : int  516 513 513 506 513 518 516 509 523 510 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.8 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.5 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0.02 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 0 0 0 0 -0.02 -0.02 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -233 -234 -232 -235 -234 -234 -234 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 47 48 48 48 47 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -270 -269 -269 -270 -269 -268 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -554 -558 -549 -558 -552 -554 -554 ...
##  $ magnet_dumbbell_y   : int  293 296 298 292 294 292 291 302 295 294 ...
```

```
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -68 -66 -65 -69 -69 -68 -63 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28 27.9 27.7 27.7 27.2 27.2 27.2 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.9 -63.9 -63.9 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -151 -151 -151 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.03 0.02 0 0 0 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 0 -0.02 0 0 0 -0.02 -0.02 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 -0.02 -0.03 -0.02 -0.02 -0.03 -0.03 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 193 193 190 193 193 192 ...
##  $ accel_forearm_y     : int  203 203 204 206 203 204 205 205 202 201 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -215 -214 -215 -215 -214 -214 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -17 -9 -16 -22 -15 -14 -16 ...
##  $ magnet_forearm_y    : num  654 661 658 655 660 653 656 655 659 656 ...
##  $ magnet_forearm_z    : num  476 473 469 473 478 476 473 472 478 472 ...
##  $ classe              : chr  "A" "A" "A" "A" ...
```

Here a plot shows the amount of times each class was recorded. Class A, the only correct form, is shown to be most frequent in the training data.

```
plot(factor(training$classe), xlab="Class", ylab="frequency")
```



Fit Model

Because there are so many obervations to consider, a random forest model was used to build randomized trees and average those predictions. This results in probabilities of each class across all trees and should be an overall good predictor. In order to make processing of the model quicker, TrainControl was set to use cross-validation, 5 for the number of folds (k-fold), and allowParallel set to TRUE to use the created cluster for parallel processing. A random forest model was then created with the training data.

```
# Configure parallel processing for speed
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

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

```
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
fitControl <- trainControl(method="cv", number=5, allowParallel=TRUE)
fit <- train(x, y, method="rf", data=trainingdata, trControl=fitControl)
# shutdown cluster
stopCluster(cluster)
registerDoSEQ()
```

```
## Random Forest
##
## 14718 samples
##    51 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11775, 11774, 11774, 11775, 11774
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9920505  0.9899431
##   26    0.9915748  0.9893410
##   51    0.9862753  0.9826367
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Validation

The validation data was then used with the model. Accuracy is shown to be over 99% with an error under 1%. The frequency of each class is shown in the confusion matrix table. This high accuracy with the validation data shows that random forest method creates an accurate model.

```
modValid <- predict(fit, validation)
CM<-confusionMatrix(modValid,as.factor(validation$classe))
print(CM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    2    0    0    0
##          B    0  947    6    0    0
##          C    0    0  849   15    0
##          D    0    0    0  784    0
##          E    0    0    0    5  901
##
## Overall Statistics
##
##                Accuracy : 0.9943
##                  95% CI : (0.9918, 0.9962)
```

```
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9928
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9979   0.9930   0.9751   1.0000
## Specificity            0.9994   0.9985   0.9963   1.0000   0.9988
## Pos Pred Value         0.9986   0.9937   0.9826   1.0000   0.9945
## Neg Pred Value         1.0000   0.9995   0.9985   0.9951   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1931   0.1731   0.1599   0.1837
## Detection Prevalence   0.2849   0.1943   0.1762   0.1599   0.1847
## Balanced Accuracy      0.9997   0.9982   0.9946   0.9876   0.9994
```

Prediction

Finally the model was used on the test data (20 observations) to predict the class performed. The predictions for each class are shown in the table below.

```
final<- predict(fit, testingdata)
print(final)
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

```
table(final)
```

```
## final
## A B C D E
## 7 8 1 1 3
```