Use weightlifting exercise datasets to predict how they will exercise

Coursera's Practical Machine Learning Final Project

Shengchu Wang

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Goal

The goal of the project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set.

- 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, and throwing the hips to the front.

Data Process

Load library

```
library(caret)
```

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

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

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Input training and testing datasets and training data summary

```
tPath <- "D:/00-OneDrive/GitHub/03-Coursera/PraticalML"
setwd(tPath)
set.seed(12521)
testing <- read.csv("pml-testing.csv")
training <- read.csv("pml-training.csv")
training$classe <- as.factor(training$classe)
dim(training)
```

```
## [1] 19622   160
```

```
# names(training)
table(training$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Delete the columns with high missing rate in the training data set

```
training <- training[colMeans(is.na(training)) <= 0.6]
dim(training)
```

```
## [1] 19622    93
```

```
missRate <- training %>%
  gather(col, value) %>%
  group_by(col) %>%
  summarize(missing_rate = mean(is.na(value)))
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
missRateD <- as.data.frame(missRate)
summary(missRateD)
```

```
##      col             missing_rate
##  Length:93          Min.   :0    
##  Class :character   1st Qu.:0    
##  Mode  :character   Median :0    
##                     Mean   :0    
##                     3rd Qu.:0    
##                     Max.   :0    
```

Now all columns in the training data set have no missing data

Remove non-predictor variables and near zero variables

```r
names(training)[1:10]
```

```
##  [1] "X"                    "user_name"            "raw_timestamp_part_1"
##  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"          
##  [7] "num_window"           "roll_belt"            "pitch_belt"          
## [10] "yaw_belt"
```

```r
training <- training[, -c(1:7)]
dim(training)
```

```
## [1] 19622    86
```

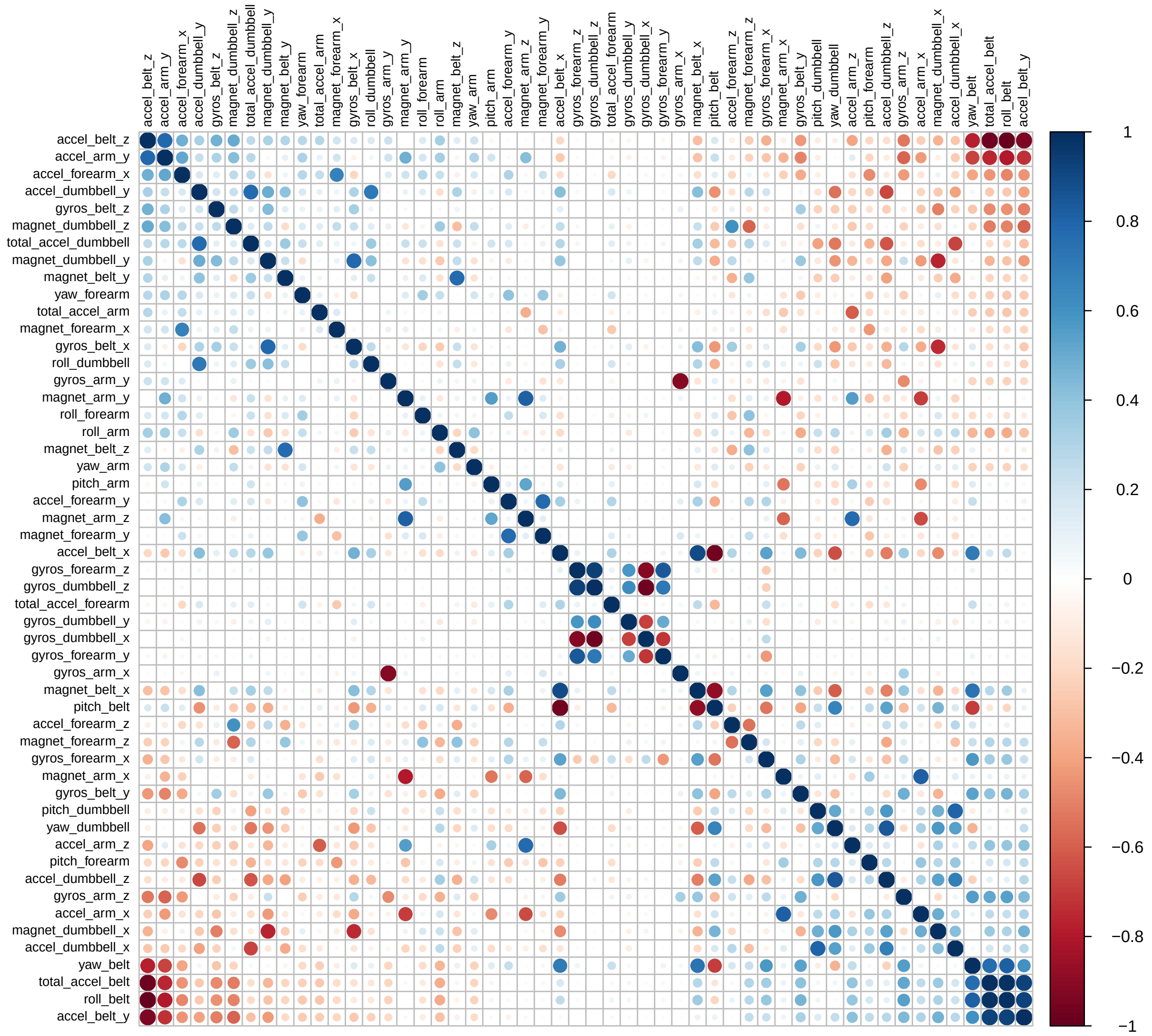
```r
near0var <- nearZeroVar(training)
training <- training[, -near0var]
dim(training)
```

```
## [1] 19622    53
```

There are 53 variables left in the training data set

Variable correlation analyis

```r
corrD <- cor(training[, -ncol(training)])
corrplot(corrD, order = "FPC", method = "circle", tl.cex = 0.7, tl.col = "black")
```



```
highCorr = findCorrelation(corrD, cutoff = 0.8)
highCorrNm <- names(training)[highCorr]
```

The high correlation variable list

accel_belt_z, roll_belt, accel_belt_y, accel_dumbbell_z, accel_belt_x, pitch_belt, accel_arm_x, accel_dumbbell_x, magnet_arm_y, gyros_forearm_y, gyros_dumbbell_x, gyros_dumbbell_z, gyros_arm_x

Build Prediction Model

Split data into train and valid

```
inTrain <- createDataPartition(training$classe, p = 0.6, list = FALSE)
train <- training[inTrain, ]
valid <- training[-inTrain, ]
msg <- paste0("train has ", nrow(train), " rows and valid has ", nrow(valid), " rows")
print(msg)
```

```
## [1] "train has 11776 rows and valid has 7846 rows"
```

Random Forest prediction modeling

```
rfControl <- trainControl(method = "cv", number = 3, verboseIter = FALSE)
rfMD <- train(classe ~ ., data=train, method = "rf", trControl = rfControl)
rfMD$finalModel
```

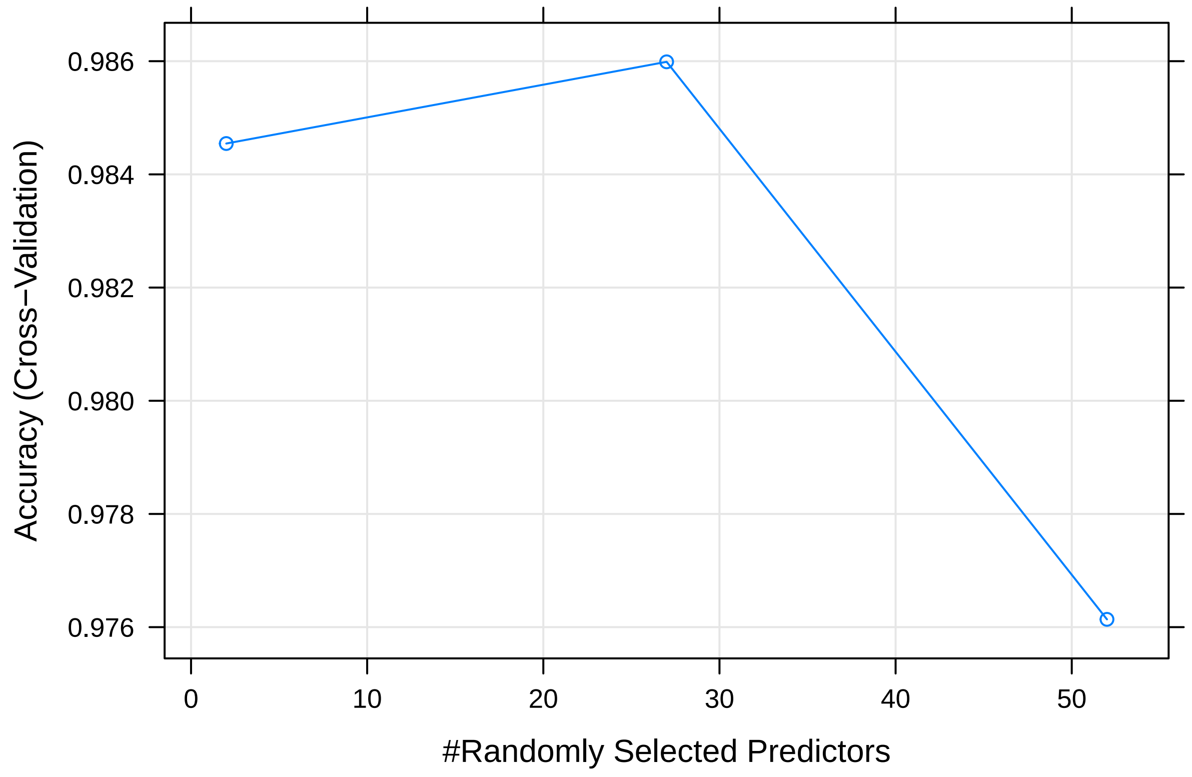
```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.96%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3341    5    2    0    0 0.002090800
## B   24 2240   14    1    0 0.017112769
## C    0   14 2032    8    0 0.010710808
## D    0    2   29 1898    1 0.016580311
## E    0    2    5    6 2152 0.006004619
```

```
rfPred <- predict(rfMD, newdata = valid)
rfconfus <- confusionMatrix(rfPred, valid$classe)
rfconfus
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2229   10    0    1    0
##          B    1 1502   12    0    1
##          C    1    6 1351   16    3
##          D    0    0    5 1269    7
##          E    1    0    0    0 1431
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9918
##                  95% CI : (0.9896, 0.9937)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9897
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9895   0.9876   0.9868   0.9924
## Specificity            0.9980   0.9978   0.9960   0.9982   0.9998
## Pos Pred Value         0.9951   0.9908   0.9811   0.9906   0.9993
## Neg Pred Value         0.9995   0.9975   0.9974   0.9974   0.9983
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1914   0.1722   0.1617   0.1824
## Detection Prevalence   0.2855   0.1932   0.1755   0.1633   0.1825
## Balanced Accuracy      0.9983   0.9936   0.9918   0.9925   0.9961
```

```r
rfAccuracy <- round(rfconfus$overall[1], 4)
rfErrRate <- 1.0-rfAccuracy
plot(rfMD)
```



The out of sample accuracy is high, therefore the out of sample error rate is low.

Accuracy = 0.9918 and Out of sample error = 0.0082

GBM prediction modeling

```
GBMcontrol <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
GBMmd  <- train(classe ~ ., data=train, method = "gbm", trControl = GBMcontrol, verbose = FALSE)
GBMmd$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 52 had non-zero influence.
```

```
print(GBMmd)
```

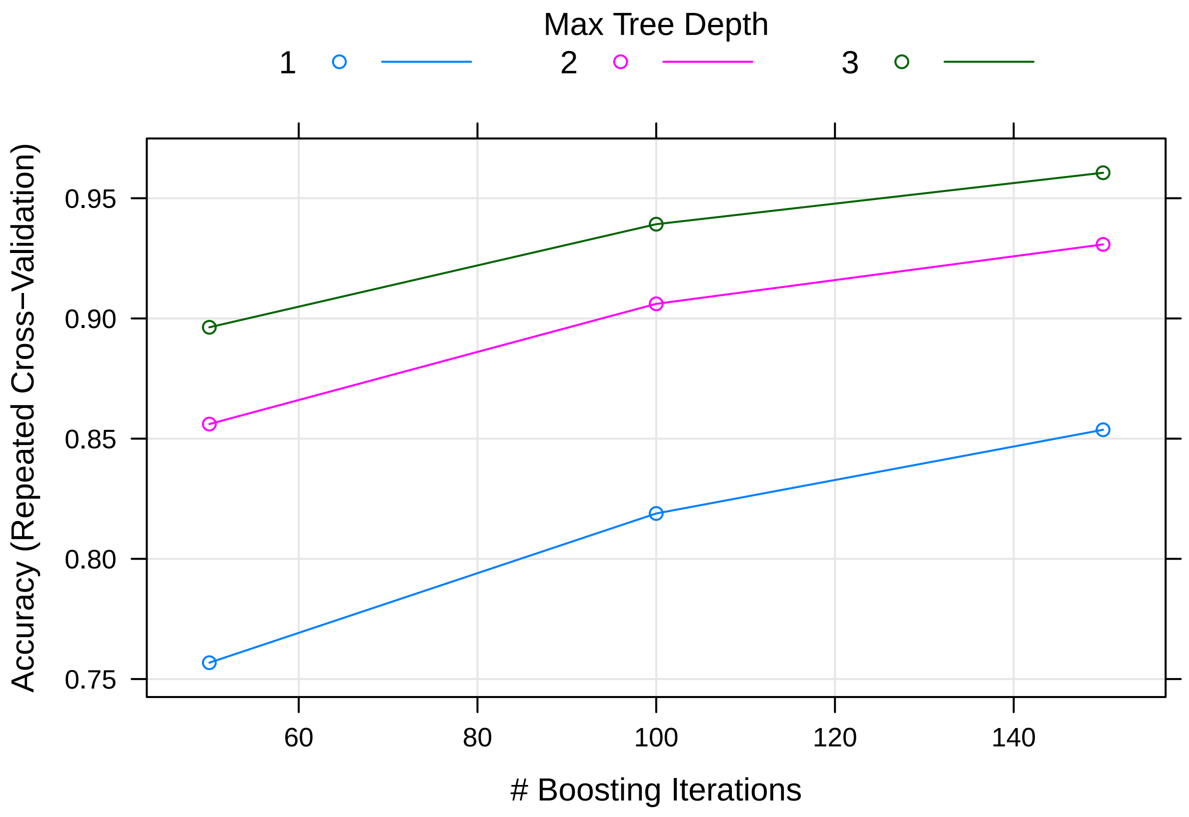
```
## Stochastic Gradient Boosting
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 9420, 9421, 9422, 9421, 9420
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7567928  0.6916987
##   1                  100      0.8188688  0.7707986
##   1                  150      0.8536856  0.8149162
##   2                   50      0.8560624  0.8176704
##   2                  100      0.9060801  0.8811286
##   2                  150      0.9307909  0.9124030
##   3                   50      0.8963137  0.8687723
##   3                  100      0.9391984  0.9230675
##   3                  150      0.9605982  0.9501509
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
GBMpred <- predict(GBMmd, newdata = valid)
GBMconfus <- confusionMatrix(GBMpred, valid$classe)
GBMconfus
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 2197   50    0    2    2
##          B   16 1425   54    4   19
##          C   13   38 1293   50   11
##          D    4    3   19 1220   26
##          E    2    2    2   10 1384
##
## Overall Statistics
##
##                Accuracy : 0.9583
##                  95% CI : (0.9537, 0.9626)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9473
##
##  Mcnemar's Test P-Value : 4.471e-12
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9843   0.9387   0.9452   0.9487   0.9598
## Specificity            0.9904   0.9853   0.9827   0.9921   0.9975
## Pos Pred Value         0.9760   0.9387   0.9203   0.9591   0.9886
## Neg Pred Value         0.9937   0.9853   0.9884   0.9900   0.9910
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2800   0.1816   0.1648   0.1555   0.1764
## Detection Prevalence   0.2869   0.1935   0.1791   0.1621   0.1784
## Balanced Accuracy      0.9874   0.9620   0.9639   0.9704   0.9786
```

```
GBMaccuracy <- round(GBMconfus$overall[1], 4)
GBMerrRate <- 1.0-GBMaccuracy
plot(GBMmd)
```



Compared with the random forest method, GBS has lower out-of-sample accuracy, so the out-of-sample error rate is higher.

Accuracy = 0.9583 and Out of sample error = 0.0417

Predict the test data

Therefore, the random forest prediction model is our choice for final prediction

```
finalPred <- predict(rfMD, newdata = testing)
finalPred
```

```
##  [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
```

End of the report

-
-
-