Prediction Assignment Writeup

Glenn Matias

August 6, 2019

Introduction

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).

Preliminary loading of requirements

Load the required libraries

```
suppressWarnings(suppressMessages(library(randomForest)))
suppressWarnings(suppressMessages(library(caret)))
suppressWarnings(suppressMessages(library(skimr)))
suppressWarnings(suppressMessages(library(dplyr)))
```

Read and load the training set

```
train <- read.csv("datasets/pml-training.csv")
```

Cleaning

Use skimr package to check for missing data

```
suppressWarnings(suppressMessages(skim_to_wide(train)))
```

```
## # A tibble: 160 x 16
##    type  variable missing complete n     n_unique top_counts ordered mean
##    <chr> <chr>    <chr>   <chr>    <chr> <chr>    <chr>      <chr>   <chr>
##  1 fact~ amplitu~ 0       19622    19622 4        "        ~ FALSE   <NA>
##  2 fact~ amplitu~ 0       19622    19622 3        "        ~ FALSE   <NA>
##  3 fact~ amplitu~ 0       19622    19622 3        "        ~ FALSE   <NA>
##  4 fact~ classe   0       19622    19622 5        "        ~ FALSE   <NA>
##  5 fact~ cvtd_ti~ 0       19622    19622 20       "28/: 149~ FALSE   <NA>
##  6 fact~ kurtosi~ 0       19622    19622 328      "        ~ FALSE   <NA>
##  7 fact~ kurtosi~ 0       19622    19622 317      "        ~ FALSE   <NA>
##  8 fact~ kurtosi~ 0       19622    19622 401      "        ~ FALSE   <NA>
##  9 fact~ kurtosi~ 0       19622    19622 323      "        ~ FALSE   <NA>
## 10 fact~ kurtosi~ 0       19622    19622 330      "        ~ FALSE   <NA>
## # ... with 150 more rows, and 7 more variables: sd <chr>, p0 <chr>,
## #   p25 <chr>, p50 <chr>, p75 <chr>, p100 <chr>, hist <chr>
```

Set missing values to 0

```
train[is.na(train)] <- 0
```

Remove unneccesary features from the dataset

```
train = subset(train, select = -c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timesta
```

Using the skimr command earlier 'skim_to_wide', we were able to see features that are not numeric in datatype. We can also see the datatypes of the features by running the following code

```
# sapply(train, class)
```

We now convert the datatypes of the features that are not set to numeric to be numeric

```
{
    train$kurtosis_roll_belt = as.numeric(train$kurtosis_roll_belt)
    train$kurtosis_yaw_belt = as.numeric(train$kurtosis_yaw_belt)
    train$skewness_roll_belt = as.numeric(train$skewness_roll_belt)
    train$skewness_roll_belt.1 = as.numeric(train$skewness_roll_belt.1)
    train$skewness_yaw_belt = as.numeric(train$skewness_yaw_belt)
    train$max_yaw_belt = as.numeric(train$max_yaw_belt)
    train$min_yaw_belt = as.numeric(train$min_yaw_belt)
    train$amplitude_yaw_belt = as.numeric(train$amplitude_yaw_belt)
    train$kurtosis_roll_arm = as.numeric(train$kurtosis_roll_arm)
    train$kurtosis_picth_arm = as.numeric(train$kurtosis_picth_arm)
    train$kurtosis_yaw_arm = as.numeric(train$kurtosis_yaw_arm)
    train$skewness_roll_arm = as.numeric(train$skewness_roll_arm)
    train$skewness_pitch_arm = as.numeric(train$skewness_pitch_arm)
    train$skewness_yaw_arm = as.numeric(train$skewness_yaw_arm)
    train$kurtosis_roll_dumbbell = as.numeric(train$kurtosis_roll_dumbbell)
    train$kurtosis_picth_dumbbell = as.numeric(train$kurtosis_picth_dumbbell)
    train$kurtosis_yaw_dumbbell = as.numeric(train$kurtosis_yaw_dumbbell)
    train$skewness_roll_dumbbell = as.numeric(train$skewness_roll_dumbbell)
    train$skewness_pitch_dumbbell = as.numeric(train$skewness_pitch_dumbbell)
    train$skewness_yaw_dumbbell = as.numeric(train$skewness_yaw_dumbbell)
    train$max_yaw_dumbbell = as.numeric(train$max_yaw_dumbbell)
    train$min_yaw_dumbbell = as.numeric(train$min_yaw_dumbbell)
    train$amplitude_yaw_dumbbell = as.numeric(train$amplitude_yaw_dumbbell)
    train$kurtosis_roll_forearm = as.numeric(train$kurtosis_roll_forearm)
    train$kurtosis_picth_forearm = as.numeric(train$kurtosis_picth_forearm)
    train$kurtosis_yaw_forearm = as.numeric(train$kurtosis_yaw_forearm)
    train$skewness_roll_forearm = as.numeric(train$skewness_roll_forearm)
    train$skewness_pitch_forearm = as.numeric(train$skewness_pitch_forearm)
    train$skewness_yaw_forearm = as.numeric(train$skewness_yaw_forearm)
    train$max_yaw_forearm = as.numeric(train$max_yaw_forearm)
    train$min_yaw_forearm = as.numeric(train$min_yaw_forearm)
    train$amplitude_yaw_forearm = as.numeric(train$amplitude_yaw_forearm)
    train$kurtosis_picth_belt = as.numeric(train$kurtosis_picth_belt)
}
```

Splitting of the dataset

We split the given data with the known Y into two datasets; for training and for testing. The training set will have 80% of the dataset while the test set will have the remaining 20% of the dataset.

```
inTrain     <- createDataPartition(train$classe, p = 0.8, list = FALSE)
inTraining  <- train[inTrain,]
inTest      <- train[-inTrain,]
```

Allocation

For faster training and to maximize available hardware, we set number of cores to be used in training to the max number of cores available in the computer - 1. We deducted it by 1 core so that there will not be any processor throttling. Also, by not exhausting all the available cores, we can still use the computer while the model is in training.

Load the required library

```
suppressWarnings(suppressMessages(library(doParallel)))
```

Determine available CPU cores and deduct it by 1 core

```
ncores <- makeCluster(detectCores() - 1)
```

Set R to use ncores

```
registerDoParallel(cores=ncores)
getDoParWorkers() # 3
```

```
## [1] 3
```

Training

For this paper, we will be using Random Forest in predicting our Y variable.

Set cross validation number of folds and number times it is to be repeated

```
control <- trainControl(method='repeatedcv', number=5, repeats=3)
```

Train the Random forest:
1. Set number of trees to 100
2. Set metric as Accuracy since we will be using categorical outcomes
3. Normalize and scale the data
4. Save the model
5. Saved model can be used for minimizing time consumed in training the Random Forest.

```
if (!file.exists("coursera_machine_learning_model.RData")) {

model <- train(classe ~ ., data = inTraining, method = "rf", ntree = 100, metric = "Accuracy", preProce
)

save(model, file = "coursera_machine_learning_model.RData")

} else {
    load(file = "coursera_machine_learning_model.RData", verbose = TRUE)
}
```

```
## Loading objects:
##   model
```

Evaluating

Let's now see our training result

```
print(model, digits=4)
```

```
## Random Forest
##
## 15699 samples
##   154 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (154), scaled (154)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 12560, 12559, 12559, 12559, 12559, 12560, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##     2   0.8210    0.7699
##    78   0.9967    0.9959
##   154   0.9933    0.9915
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 78.
```

Let's now try our model to our test dataset

```
pred <- predict(model, newdata=inTest)
```

Show the result of the testing

```
confusionMatrix(pred, inTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    0    0    0    0
##          B    0  759    0    0    0
##          C    0    0  684    1    0
##          D    0    0    0  642    0
##          E    0    0    0    0  721
##
## Overall Statistics
##
##                Accuracy : 0.9997
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.9997
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   0.9984   1.0000
## Specificity            1.0000   1.0000   0.9997   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   0.9985   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   0.9997   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1744   0.1637   0.1838
## Detection Prevalence   0.2845   0.1935   0.1746   0.1637   0.1838
## Balanced Accuracy      1.0000   1.0000   0.9998   0.9992   1.0000
```

Based on the confusion matrix, we can say that our model did a pretty good job at classifying the test dataset.

Let's now check the estimated OOB estimate of error rate of our model

```
model$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 100, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 0.25%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4461    2    0    0    1 0.000672043
## B    5 3028    4    0    1 0.003291639
## C    0    8 2730    0    0 0.002921841
## D    0    0   10 2561    2 0.004663817
## E    0    1    0    5 2880 0.002079002
```

As observed we only have an error rate of 0.25%, an impressive score!

Prediction

Read the quiz set / unknown Y dataset

```
validate <- read.csv("datasets/pml-testing.csv")
```

Do the necessary data cleaning as performed to the training set

```
validate[is.na(validate)] <- 0


validate = subset(validate, select = -c(problem_id, X, user_name, raw_timestamp_part_1, raw_timestamp_pa
```

```r
{
    validate$kurtosis_roll_belt = as.numeric(validate$kurtosis_roll_belt)
    validate$kurtosis_yaw_belt = as.numeric(validate$kurtosis_yaw_belt)
    validate$skewness_roll_belt = as.numeric(validate$skewness_roll_belt)
    validate$skewness_roll_belt.1 = as.numeric(validate$skewness_roll_belt.1)
    validate$skewness_yaw_belt = as.numeric(validate$skewness_yaw_belt)
    validate$max_yaw_belt = as.numeric(validate$max_yaw_belt)
    validate$min_yaw_belt = as.numeric(validate$min_yaw_belt)
    validate$amplitude_yaw_belt = as.numeric(validate$amplitude_yaw_belt)
    validate$kurtosis_roll_arm = as.numeric(validate$kurtosis_roll_arm)
    validate$kurtosis_picth_arm = as.numeric(validate$kurtosis_picth_arm)
    validate$kurtosis_yaw_arm = as.numeric(validate$kurtosis_yaw_arm)
    validate$skewness_roll_arm = as.numeric(validate$skewness_roll_arm)
    validate$skewness_pitch_arm = as.numeric(validate$skewness_pitch_arm)
    validate$skewness_yaw_arm = as.numeric(validate$skewness_yaw_arm)
    validate$kurtosis_roll_dumbbell = as.numeric(validate$kurtosis_roll_dumbbell)
    validate$kurtosis_picth_dumbbell = as.numeric(validate$kurtosis_picth_dumbbell)
    validate$kurtosis_yaw_dumbbell = as.numeric(validate$kurtosis_yaw_dumbbell)
    validate$skewness_roll_dumbbell = as.numeric(validate$skewness_roll_dumbbell)
    validate$skewness_pitch_dumbbell = as.numeric(validate$skewness_pitch_dumbbell)
    validate$skewness_yaw_dumbbell = as.numeric(validate$skewness_yaw_dumbbell)
    validate$max_yaw_dumbbell = as.numeric(validate$max_yaw_dumbbell)
    validate$min_yaw_dumbbell = as.numeric(validate$min_yaw_dumbbell)
    validate$amplitude_yaw_dumbbell = as.numeric(validate$amplitude_yaw_dumbbell)
    validate$kurtosis_roll_forearm = as.numeric(validate$kurtosis_roll_forearm)
    validate$kurtosis_picth_forearm = as.numeric(validate$kurtosis_picth_forearm)
    validate$kurtosis_yaw_forearm = as.numeric(validate$kurtosis_yaw_forearm)
    validate$skewness_roll_forearm = as.numeric(validate$skewness_roll_forearm)
    validate$skewness_pitch_forearm = as.numeric(validate$skewness_pitch_forearm)
    validate$skewness_yaw_forearm = as.numeric(validate$skewness_yaw_forearm)
    validate$max_yaw_forearm = as.numeric(validate$max_yaw_forearm)
    validate$min_yaw_forearm = as.numeric(validate$min_yaw_forearm)
    validate$amplitude_yaw_forearm = as.numeric(validate$amplitude_yaw_forearm)
    validate$kurtosis_picth_belt = as.numeric(validate$kurtosis_picth_belt)
}
```

Predict!

```r
print(predict(model, newdata=validate))
```

```
##  [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
```

We are now done with our small experiment!