Qualitative Activity Recognition

Wesley Padilla

2/14/2021

Synopsis

The Qualitative Activity Recognition of Weight Lifting Exercises study (available at http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf) investigates three aspects that pertain to qualitative activity recognition - specifying correct execution, detecting execution mistakes, and providing feedback to the user. People regularly quantify how much of a particular activity they do but they rarely quantify how well they do it. In this project we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants who were asked to perform barbell lifts correctly and incorrectly in 5 different ways. Our objective is to develop a machine learning model to predict the manner in which they did the exercise.

The "classe" variable in the training set is what we will predict. Participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

Data Sources

The training data for this project are available here:
https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:
https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

Data Processing

```r
# Load libraries
library(caret)
```

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

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

```r
library(Hmisc)
```

```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```r
library(Amelia)
```

```
## Loading required package: Rcpp

## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.6, built: 2019-11-24)
## ## Copyright (C) 2005-2021 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```r
library(VIM)
```

```
## Loading required package: colorspace

## Loading required package: grid

## Loading required package: data.table

## VIM is ready to use.
##  Since version 4.0.0 the GUI is in its own package VIMGUI.
##
##           Please use the package to use the new (and old) GUI.

## Suggestions and bug-reports can be submitted at: https://github.com/alexkowa/VIM/issues

##
## Attaching package: 'VIM'

## The following object is masked from 'package:datasets':
##
##     sleep
```

```r
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:VIM':
##
##     wine
```

```r
library(rpart)
library(rpart.plot)
library(RColorBrewer)
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:rattle':
##
##     importance
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(AppliedPredictiveModeling)
```

```
# Load Data and assigned NA to missing values
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training <- read.csv(url(trainURL), header = TRUE, as.is = TRUE, sep = ',', na.strings = c("", "NA", "#
testing <- read.csv(url(trainURL), header = TRUE, as.is = TRUE, sep = ',', na.strings = c("", "NA", "#D
training$classe <- as.factor(training$classe)

dim(training)
```

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

```
dim(testing)
```

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

R output confirms there are 19,622 observations and 160 variables in the training and test data sets.

Data Cleaning

Many models will fail in cases where predictors have a single unique value (also known as "zero-variance predictors"). Therefore, using the nearZeroVar function to remove near zero-variance predictors.

```
nzv <- nearZeroVar(training)
training <- training[, -nzv]
testing  <- testing[, -nzv]
dim(training)
```

```
## [1] 19622   124
```

```
dim(testing)
```

```
## [1] 19622   124
```

Removing near zero-variance predictors reduced the variable count from 160 to 124.

After running STR function, you can see the number of variables filled with NA values. Therefore, removing those variables.

```
training <- training[, colSums(is.na(training)) == 0]
testing <- testing[, colSums(is.na(testing)) == 0]
dim(training)
```

```
## [1] 19622    59
```

```r
dim(testing)
```

```
## [1] 19622    59
```

Removing NA variables reduced the variable count to 59.

Several variables will not be required if we are trying to predict the exercise class based on accelerometer. Top six variables are not needed for the prediction therefore removing them from the dataset.

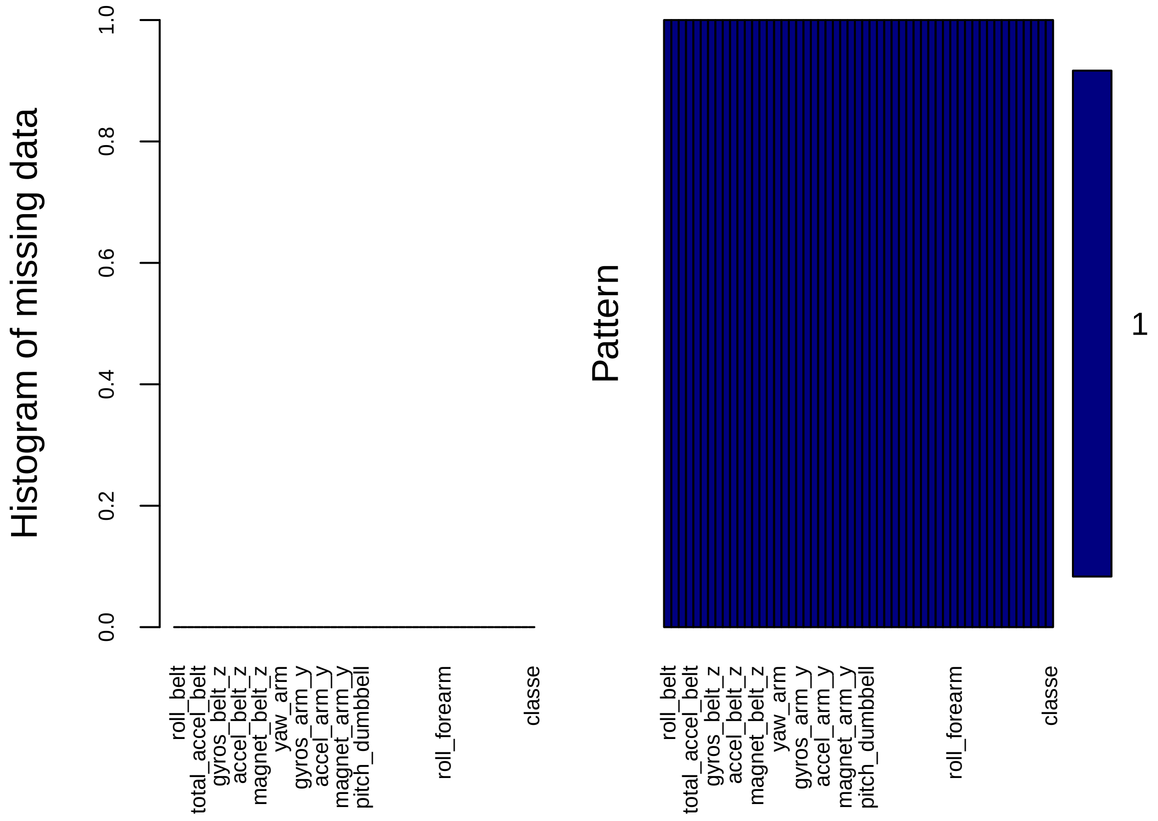
```r
# Remove variables not required from the datasets
training <- training[, -c(1:6)]
testing <- testing[, -c(1:6)]
dim(training)
```

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

```r
dim(testing)
```

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

Using library(VIM) to check if additional cleaning is required. Algorithm below plots NAs for each variable. Upon inspection, it looks like missing values are not an issue anymore.



```
##
##  Variables sorted by number of missings:
##              Variable Count
##             roll_belt     0
##            pitch_belt     0
##              yaw_belt     0
##      total_accel_belt     0
##          gyros_belt_x     0
##          gyros_belt_y     0
##          gyros_belt_z     0
##          accel_belt_x     0
##          accel_belt_y     0
##          accel_belt_z     0
##         magnet_belt_x     0
##         magnet_belt_y     0
##         magnet_belt_z     0
##              roll_arm     0
##             pitch_arm     0
##               yaw_arm     0
##       total_accel_arm     0
##           gyros_arm_x     0
##           gyros_arm_y     0
##           gyros_arm_z     0
##           accel_arm_x     0
##           accel_arm_y     0
##           accel_arm_z     0
##          magnet_arm_x     0
##          magnet_arm_y     0
##          magnet_arm_z     0
##         roll_dumbbell     0
##        pitch_dumbbell     0
##          yaw_dumbbell     0
##  total_accel_dumbbell     0
##      gyros_dumbbell_x     0
##      gyros_dumbbell_y     0
##      gyros_dumbbell_z     0
##      accel_dumbbell_x     0
##      accel_dumbbell_y     0
##      accel_dumbbell_z     0
##     magnet_dumbbell_x     0
##     magnet_dumbbell_y     0
##     magnet_dumbbell_z     0
##          roll_forearm     0
##         pitch_forearm     0
##           yaw_forearm     0
##   total_accel_forearm     0
##       gyros_forearm_x     0
##       gyros_forearm_y     0
##       gyros_forearm_z     0
##       accel_forearm_x     0
##       accel_forearm_y     0
##       accel_forearm_z     0
##      magnet_forearm_x     0
##      magnet_forearm_y     0
##      magnet_forearm_z     0
##                classe     0
```

Preprocessing Variables

```r
v <- which(lapply(training, class) %in% "numeric")

preObj <-preProcess(training[,v],method=c('knnImpute', 'center', 'scale'))
trainAdj <- predict(preObj, training[,v])
trainAdj$classe <- training$classe

testAdj <-predict(preObj,testing[,v])
```

Data Splitting/Cross Validation

Using function createDataPartition to create balanced splits of the data. If the y argument to this function is a factor, the random sampling occurs within each class and would preserve the overall class distribution of the data. I decided to use a 75/25 % split of the data.

```r
set.seed(998)
inTrain <- createDataPartition(y = trainAdj$classe,
                                  p = .75,
                                  list = FALSE,
                                  times = 1)
trainClass <- trainAdj[inTrain,]
testClass <- trainAdj[-inTrain,]
dim(trainClass)
```

```
## [1] 14718    28
```

```r
dim(testClass)
```

```
## [1] 4904   28
```

Machine Learning Prediction Models

Using two models - decision trees with Classification and Regression Trees (CART) (using rpart function) and random forest (rf).

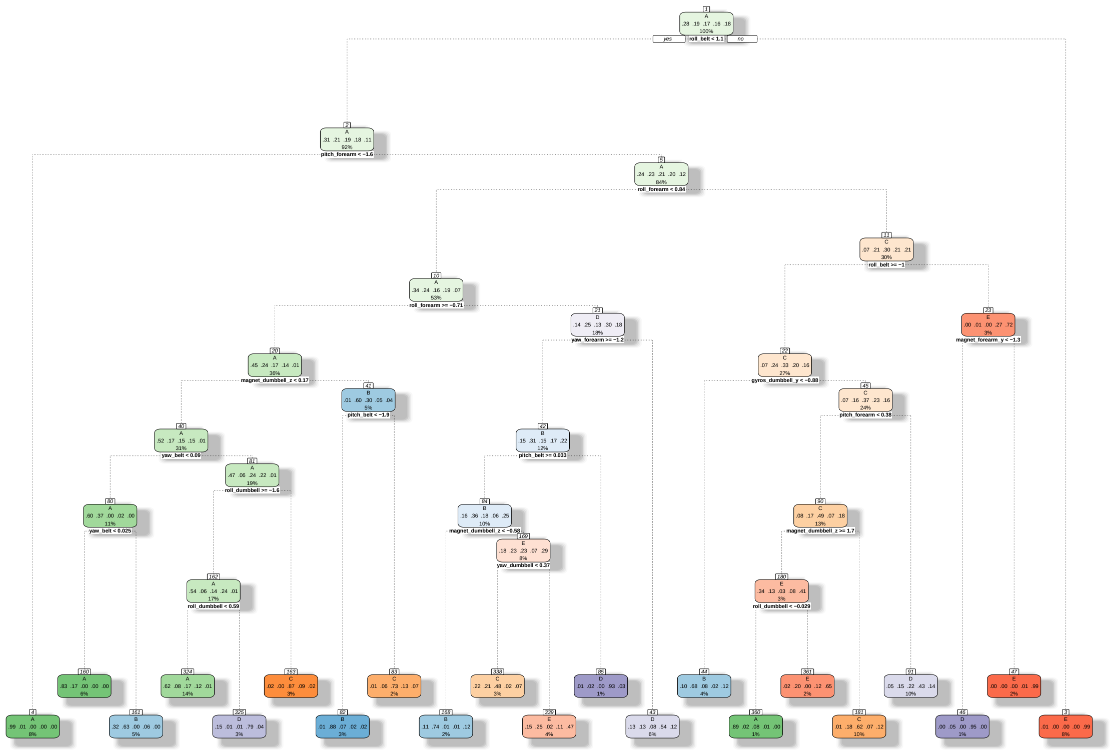
Model #1: CART

Using CART (by default it uses 10-fold CV (cross validation)).

```r
modFit_CART <- rpart(classe~., data = trainClass, method = "class")
```

Using fancyRpartPlt function from the rattle package to plot the decision tree.

```r
fancyRpartPlot(modFit_CART)
```


Rattle 2021−Feb−14 13:35:02 wesleypadilla

```r
prediction_CART <- predict(modFit_CART, trainClass, type = "class")
confusionMatrix(prediction_CART, trainClass$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3335  323  360  241   14
##          B  334 1557   89   65  124
##          C  139  399 1686  178  245
##          D  269  359  419 1822  344
##          E  108  210   13  106 1979
##
## Overall Statistics
##
##                Accuracy : 0.7052
##                  95% CI : (0.6978, 0.7126)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6276
##
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7969   0.5467   0.6568   0.7554   0.7313
## Specificity            0.9109   0.9484   0.9209   0.8870   0.9636
## Pos Pred Value         0.7805   0.7178   0.6369   0.5671   0.8191
## Neg Pred Value         0.9186   0.8971   0.9270   0.9487   0.9409
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2266   0.1058   0.1146   0.1238   0.1345
## Detection Prevalence   0.2903   0.1474   0.1798   0.2183   0.1642
## Balanced Accuracy      0.8539   0.7476   0.7889   0.8212   0.8475
```

Results are very poor as reflected in the overall statistics.

Model #2: Random Forest

Second model should perform much better than CART. Random forest models are popular due to their high accuracy rate.

```
modFit_rf <- randomForest(classe~., data = trainClass)
```

```
prediction_rf <- predict(modFit_rf, trainClass, type = "class")
confusionMatrix(prediction_rf, trainClass$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 4185    0    0    0    0
##          B    0 2848    0    0    0
##          C    0    0 2567    0    0
##          D    0    0    0 2412    0
##          E    0    0    0    0 2706
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  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   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1839
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1839
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Results and Final Model Selection

Comparing overall statistics between the two models, the best performance comes from the random forest model. Therefore, we will use it as the final model. Using resampling, we can estimate the standard error of performance.

```r
prediction_rf2 <- predict(modFit_rf, testClass, type = "class")
confusionMatrix(prediction_rf2, testClass$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    3    0    0    0
##          B    1  944    3    0    0
##          C    0    2  843    6    0
##          D    0    0    7  798    1
##          E    0    0    2    0  900
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9925, 0.9967)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9947   0.9860   0.9925   0.9989
## Specificity            0.9991   0.9990   0.9980   0.9980   0.9995
## Pos Pred Value         0.9979   0.9958   0.9906   0.9901   0.9978
## Neg Pred Value         0.9997   0.9987   0.9970   0.9985   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1925   0.1719   0.1627   0.1835
## Detection Prevalence   0.2849   0.1933   0.1735   0.1644   0.1839
## Balanced Accuracy      0.9992   0.9969   0.9920   0.9953   0.9992
```

Performance results are still high using the test data set, therefore, I am confident this model can predict out of sample observations with a high degree of accuracy. Performance was better using in-sample data.

Out of sample error is the error you predict with the validation data set. As you can see below, it is very low at 0.0051 or 0.51%.

```r
missed_classification <- function(values, prediction) {
    sum(prediction != values)/length(values)
}
OS_error_rate <- missed_classification(testClass$classe, prediction_rf2)
OS_error_rate
```

```
## [1] 0.005097879
```