Machine Learning for Qualitative Activity Recognition

Debasmita Biswal

Introduction

Human activity recognition (HAR) is an active field of research in computer vision and machine learning, where the goal is to understand human behaviour from data captured by a variety of approaches such as cameras and wearable sensor devices (Ref:http://groupware.les.inf.puc-rio.br/work.jsf?p1=10335). HAR is a powerful tool in medical application arears that help enrich feature set in health studies and enhance personalization and effectiveness of health, wellness and fitness applications (Ref: https://arxiv.org/pdf/1607.04867.pdf). Velloso et al. reported a study on qualitative activity recognition, where the aim was to investigate the feasibility of automatically assessing the quality of weight-lifting exercises and providing real-time feedback to the athlete (Ref:http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf). The data for this specific study was collected from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. The participants were asked to perform weight lifts correctly and incorrectly in 5 different ways. The authors used machine learning and pattern recognition techniques to detect execution mistakes in an atumated fashion. The experimental details and data acquisition process is provided in the original paper (Ref:http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf). In this report, I have provided a detailed explnataion of how machine learning can be applied to predict the manner in which the participants performed the exercise (among the 5 different ways). The prediction model will be used to predict 20 different test cases provided in the test data set. All analyses reported here-in were performed using R, which is an open source programming language and software environment for statistical computing and graphics.

Get the Data

First, the training data was obtained from https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv and the test data was acquired from https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv. The code used to download the dataset is provided below:

```
url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(url, destfile = "training.csv", method = "curl") url <-
"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url, destfile = "final-test.csv", method = "curl")
```

The CSV files named, `training.csv` and `final-test.csv`, are used to save the training and test data set, respectively, in a local directory.

Data Processing and Analysis

Since the data set `training.csv` is stored in CSV form, the function `read.csv()` can be used to load the data into an R dataframe (`data1`). The `header=TRUE` argument is used to pass the column headers from `training.csv` to `read.csv()` function, and `na.strings=c("", "NA"))` argument is used to replace empty spaces with `NAs` (not available). In a similar way, the data set `final-test.csv` can be loaded into another data frame (`data2`).

```
data1 <- read.csv("~path/training.csv",header=TRUE, na.strings=c("", "NA"))
data2 <- read.csv("~path/final-test.csv", header=TRUE, na.strings=c("",
"NA"))
```

Using `ncol()` function it was found that `data1` and `data2` dataframes have 160 columns. While `data1` has `nrow(data1)`= 19622 rows, `data2` has `nrow(data2)`= 20 rows in total. The `str()` function in R provides a way to display the structure of R data structures. As a first step in data processing, the internal structure of the `data1` dataframe was investigated using `str(data1)` command. The first several lines of the output are as follows:

```
str(data1)

## 'data.frame':    19622 obs. of  160 variables:
##  $ X                        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name                : Factor w/ 6 levels "adelmo","carlitos",..: 2
2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1     : int  1323084231 1323084231 1323084231
1323084232 1323084232 1323084232 1323084232 1323084232 1323084232 1323084232
...
##  $ raw_timestamp_part_2     : int  788290 808298 820366 120339 196328
304277 368296 440390 484323 484434 ...
##  $ cvtd_timestamp           : Factor w/ 20 levels "02/12/2011 13:32",..: 9
9 9 9 9 9 9 9 9 9 ...
##  $ new_window               : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1
1 1 1 ...
##  $ num_window               : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt                : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42
1.43 1.45 ...
##  $ pitch_belt               : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13
8.16 8.17 ...
##  $ 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 ...
##  $ kurtosis_roll_belt       : Factor w/ 396 levels "-0.016850","-
0.021024",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_belt      : Factor w/ 316 levels "-0.021887","-
0.060755",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_belt        : Factor w/ 1 level "#DIV/0!": NA NA NA NA NA
NA NA NA NA NA ...
##  $ skewness_roll_belt       : Factor w/ 394 levels "-0.003095","-
0.010002",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_belt.1     : Factor w/ 337 levels "-0.005928","-
```

```
0.005960",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_belt       : Factor w/ 1 level "#DIV/0!": NA NA NA NA NA
NA NA NA NA NA ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 67 levels "-0.1","-0.2",..: NA NA
NA NA NA NA NA NA NA NA ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 67 levels "-0.1","-0.2",..: NA NA
NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 3 levels "#DIV/0!","0.00",..: NA NA
NA NA NA NA NA NA NA NA ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02
0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -
0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21
...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609
...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313
-312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128
-128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8
21.7 21.6 ...
##  $ 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 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
```

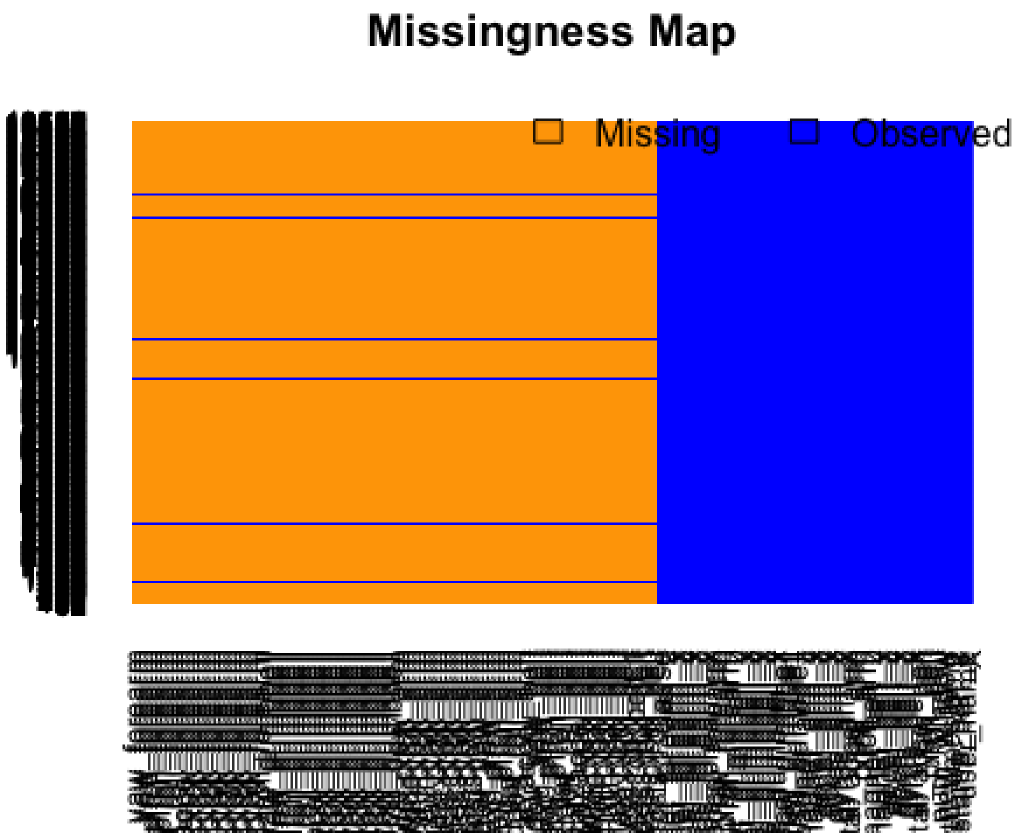
```
##  $ avg_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x            : num  0 0.02 0.02 0.02 0 0.02 0 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 -0.03 -0.03 ...
##  $ gyros_arm_z            : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -
0.02 ...
##  $ accel_arm_x            : int  -288 -290 -289 -289 -289 -289 -289 -289
-288 -288 ...
##  $ accel_arm_y            : int  109 110 110 111 111 111 111 111 109 110
...
##  $ accel_arm_z            : int  -123 -125 -126 -123 -123 -122 -125 -124
-122 -124 ...
##  $ magnet_arm_x           : int  -368 -369 -368 -372 -374 -369 -373 -372
-369 -376 ...
##  $ magnet_arm_y           : int  337 337 344 344 337 342 336 338 341 334
...
##  $ magnet_arm_z           : int  516 513 513 512 506 513 509 510 518 516
...
##  $ kurtosis_roll_arm      : Factor w/ 329 levels "-0.02438","-
0.04190",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_arm     : Factor w/ 327 levels "-0.00484","-
0.01311",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_arm       : Factor w/ 394 levels "-0.01548","-
0.01749",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_arm      : Factor w/ 330 levels "-0.00051","-
0.00696",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_arm     : Factor w/ 327 levels "-0.00184","-
0.01185",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_arm       : Factor w/ 394 levels "-0.00311","-
0.00562",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ max_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell          : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell         : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell           : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell : Factor w/ 397 levels "-0.0035","-0.0073",..:
NA NA NA NA NA NA NA NA NA NA ...
```

```
##  $ kurtosis_picth_dumbbell : Factor w/ 400 levels "-0.0163","-0.0233",..: 
NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 1 level "#DIV/0!": NA NA NA NA NA 
NA NA NA NA NA ...
##  $ skewness_roll_dumbbell  : Factor w/ 400 levels "-0.0082","-0.0096",..: 
NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_dumbbell : Factor w/ 401 levels "-0.0053","-0.0084",..: 
NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_dumbbell   : Factor w/ 1 level "#DIV/0!": NA NA NA NA NA 
NA NA NA NA NA ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 72 levels "-0.1","-0.2",..: NA NA 
NA NA NA NA NA NA NA NA ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 72 levels "-0.1","-0.2",..: NA NA 
NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

The `str()` function displays the number of rows (`obs`) and columns (`variables`) of the data frames, followed by a list of variables in separate lines. Each line begins with the name of the variable followed by class of the variable (e.g., variable `user_name` is of factor type (or categorical variable) and `num_window` is numeric type). A sequence of first few variable values are displayed in each line following the class of the variable. It is observed that the first 7 variables in `data1` data frame include information such as user identity, timestamp, and sliding windows used during feature extraction. These variables do not provide useful information and need to be excluded from the model. Furthermore, many variables in the data frame are observed to contain a sequence of missing values (`NA`s). In the data processing stage, it is rather important to find the missing data, which can have a big impact on data modeling. To this end, the `Amelia` package provides a useful visualization tool to get a quick summary of missing values in the data set. The command to plot missing data in `data1` is provided below:

```
# load library Amelia
library(Amelia)
# Use missmap function plot missing data
missmap(data1, col=c("orange", "blue"), legend=TRUE)
```

The missing plot reveals that a greater percentage of data is missing for many variables (orange color represents missing data). A summary of the number of missing observations for each of the variables in `data1` can also be found by using the command, `sapply(data1,function(x)  sum(is.na(x)))`. Considering the greater percentage of missing values (> 98%), data imputation techniques cannot be used for these variables. Therefore, those variables for which the number of `NA`s are greater than 19000 need to be excluded from `data1`. This was accomplished by using the `select()` function from the dplyr package. Similar data preprocessing was also applied to data2, and data2 was hold back to get an objective final evaluation of the best performing model.




```r
# load library dplyr
library(dplyr)
# select variables containing > 19000 NA's.
var <- colnames(data1)[colSums(is.na(data1)) > 19000]
# select a subset of training data, data1, by removing first 7 variables and those grouped under 'var'
subdata <- select(data1, -(1:7), -one_of(var))
```

As a result of cleaning, the number of variables in subdata was reduced to 53. The summary() function in R is then employed to obtain a five-number summary statistics of the numeric variables in subdata data frame.

```r
summary(subdata)

##    roll_belt        pitch_belt          yaw_belt       total_accel_belt
##  Min.   :-28.90   Min.   :-55.8000   Min.   :-180.00   Min.   : 0.00
##  1st Qu.:  1.10   1st Qu.:  1.7600   1st Qu.: -88.30   1st Qu.: 3.00
##  Median :113.00   Median :  5.2800   Median : -13.00   Median :17.00
##  Mean   : 64.41   Mean   :  0.3053   Mean   : -11.21   Mean   :11.31
```

```
##  3rd Qu.:123.00   3rd Qu.: 14.9000   3rd Qu.:  12.90   3rd Qu.:18.00
##  Max.   :162.00   Max.   : 60.3000   Max.   : 179.00   Max.   :29.00
##   gyros_belt_x        gyros_belt_y       gyros_belt_z
##  Min.   :-1.040000   Min.   :-0.64000   Min.   :-1.4600
##  1st Qu.:-0.030000   1st Qu.: 0.00000   1st Qu.:-0.2000
##  Median : 0.030000   Median : 0.02000   Median :-0.1000
##  Mean   :-0.005592   Mean   : 0.03959   Mean   :-0.1305
##  3rd Qu.: 0.110000   3rd Qu.: 0.11000   3rd Qu.:-0.0200
##  Max.   : 2.220000   Max.   : 0.64000   Max.   : 1.6200
##   accel_belt_x       accel_belt_y     accel_belt_z     magnet_belt_x
##  Min.   :-120.000   Min.   :-69.00   Min.   :-275.00   Min.   :-52.0
##  1st Qu.: -21.000   1st Qu.:  3.00   1st Qu.:-162.00   1st Qu.:  9.0
##  Median : -15.000   Median : 35.00   Median :-152.00   Median : 35.0
##  Mean   :  -5.595   Mean   : 30.15   Mean   : -72.59   Mean   : 55.6
##  3rd Qu.:  -5.000   3rd Qu.: 61.00   3rd Qu.:  27.00   3rd Qu.: 59.0
##  Max.   :  85.000   Max.   :164.00   Max.   : 105.00   Max.   :485.0
##  magnet_belt_y   magnet_belt_z       roll_arm         pitch_arm
##  Min.   :354.0   Min.   :-623.0   Min.   :-180.00   Min.   :-88.800
##  1st Qu.:581.0   1st Qu.:-375.0   1st Qu.: -31.77   1st Qu.:-25.900
##  Median :601.0   Median :-320.0   Median :   0.00   Median :  0.000
##  Mean   :593.7   Mean   :-345.5   Mean   :  17.83   Mean   : -4.612
##  3rd Qu.:610.0   3rd Qu.:-306.0   3rd Qu.:  77.30   3rd Qu.: 11.200
##  Max.   :673.0   Max.   : 293.0   Max.   : 180.00   Max.   : 88.500
##     yaw_arm          total_accel_arm  gyros_arm_x        gyros_arm_y
##  Min.   :-180.0000   Min.   : 1.00   Min.   :-6.37000   Min.   :-3.4400
##  1st Qu.: -43.1000   1st Qu.:17.00   1st Qu.:-1.33000   1st Qu.:-0.8000
##  Median :   0.0000   Median :27.00   Median : 0.08000   Median :-0.2400
##  Mean   :  -0.6188   Mean   :25.51   Mean   : 0.04277   Mean   :-0.2571
##  3rd Qu.:  45.8750   3rd Qu.:33.00   3rd Qu.: 1.57000   3rd Qu.: 0.1400
##  Max.   : 180.0000   Max.   :66.00   Max.   : 4.87000   Max.   : 2.8400
##   gyros_arm_z       accel_arm_x       accel_arm_y      accel_arm_z
##  Min.   :-2.3300   Min.   :-404.00   Min.   :-318.0   Min.   :-636.00
##  1st Qu.:-0.0700   1st Qu.:-242.00   1st Qu.: -54.0   1st Qu.:-143.00
##  Median : 0.2300   Median : -44.00   Median :  14.0   Median : -47.00
##  Mean   : 0.2695   Mean   : -60.24   Mean   :  32.6   Mean   : -71.25
##  3rd Qu.: 0.7200   3rd Qu.:  84.00   3rd Qu.: 139.0   3rd Qu.:  23.00
##  Max.   : 3.0200   Max.   : 437.00   Max.   : 308.0   Max.   : 292.00
##   magnet_arm_x     magnet_arm_y     magnet_arm_z    roll_dumbbell
##  Min.   :-584.0   Min.   :-392.0   Min.   :-597.0   Min.   :-153.71
##  1st Qu.:-300.0   1st Qu.:  -9.0   1st Qu.: 131.2   1st Qu.: -18.49
##  Median : 289.0   Median : 202.0   Median : 444.0   Median :  48.17
##  Mean   : 191.7   Mean   : 156.6   Mean   : 306.5   Mean   :  23.84
##  3rd Qu.: 637.0   3rd Qu.: 323.0   3rd Qu.: 545.0   3rd Qu.:  67.61
##  Max.   : 782.0   Max.   : 583.0   Max.   : 694.0   Max.   : 153.55
##  pitch_dumbbell     yaw_dumbbell      total_accel_dumbbell
##  Min.   :-149.59   Min.   :-150.871   Min.   : 0.00
##  1st Qu.: -40.89   1st Qu.: -77.644   1st Qu.: 4.00
##  Median : -20.96   Median :  -3.324   Median :10.00
##  Mean   : -10.78   Mean   :   1.674   Mean   :13.72
##  3rd Qu.:  17.50   3rd Qu.:  79.643   3rd Qu.:19.00
```

```
##  Max.   : 149.40   Max.   : 154.952   Max.   :58.00
##  gyros_dumbbell_x    gyros_dumbbell_y   gyros_dumbbell_z
##  Min.   :-204.0000   Min.   :-2.10000   Min.   : -2.380
##  1st Qu.:  -0.0300   1st Qu.:-0.14000   1st Qu.: -0.310
##  Median :   0.1300   Median : 0.03000   Median : -0.130
##  Mean   :   0.1611   Mean   : 0.04606   Mean   : -0.129
##  3rd Qu.:   0.3500   3rd Qu.: 0.21000   3rd Qu.:  0.030
##  Max.   :   2.2200   Max.   :52.00000   Max.   :317.000
##  accel_dumbbell_x  accel_dumbbell_y  accel_dumbbell_z  magnet_dumbbell_x
##  Min.   :-419.00   Min.   :-189.00   Min.   :-334.00   Min.   :-643.0
##  1st Qu.: -50.00   1st Qu.:  -8.00   1st Qu.:-142.00   1st Qu.:-535.0
##  Median :  -8.00   Median :  41.50   Median :  -1.00   Median :-479.0
##  Mean   : -28.62   Mean   :  52.63   Mean   : -38.32   Mean   :-328.5
##  3rd Qu.:  11.00   3rd Qu.: 111.00   3rd Qu.:  38.00   3rd Qu.:-304.0
##  Max.   : 235.00   Max.   : 315.00   Max.   : 318.00   Max.   : 592.0
##  magnet_dumbbell_y magnet_dumbbell_z  roll_forearm       pitch_forearm
##  Min.   :-3600     Min.   :-262.00   Min.   :-180.0000   Min.   :-72.50
##  1st Qu.:  231     1st Qu.: -45.00   1st Qu.:  -0.7375   1st Qu.:  0.00
##  Median :  311     Median :  13.00   Median :  21.7000   Median :  9.24
##  Mean   :  221     Mean   :  46.05   Mean   :  33.8265   Mean   : 10.71
##  3rd Qu.:  390     3rd Qu.:  95.00   3rd Qu.: 140.0000   3rd Qu.: 28.40
##  Max.   :  633     Max.   : 452.00   Max.   : 180.0000   Max.   : 89.80
##   yaw_forearm      total_accel_forearm gyros_forearm_x
##  Min.   :-180.00   Min.   :  0.00      Min.   :-22.000
##  1st Qu.: -68.60   1st Qu.: 29.00      1st Qu.: -0.220
##  Median :   0.00   Median : 36.00      Median :  0.050
##  Mean   :  19.21   Mean   : 34.72      Mean   :  0.158
##  3rd Qu.: 110.00   3rd Qu.: 41.00      3rd Qu.:  0.560
##  Max.   : 180.00   Max.   :108.00      Max.   :  3.970
##  gyros_forearm_y     gyros_forearm_z    accel_forearm_x   accel_forearm_y
##  Min.   : -7.02000   Min.   : -8.0900   Min.   :-498.00   Min.   :-632.0
##  1st Qu.: -1.46000   1st Qu.: -0.1800   1st Qu.:-178.00   1st Qu.:  57.0
##  Median :  0.03000   Median :  0.0800   Median : -57.00   Median : 201.0
##  Mean   :  0.07517   Mean   :  0.1512   Mean   : -61.65   Mean   : 163.7
##  3rd Qu.:  1.62000   3rd Qu.:  0.4900   3rd Qu.:  76.00   3rd Qu.: 312.0
##  Max.   :311.00000   Max.   :231.0000   Max.   : 477.00   Max.   : 923.0
##  accel_forearm_z   magnet_forearm_x  magnet_forearm_y magnet_forearm_z
##  Min.   :-446.00   Min.   :-1280.0   Min.   :-896.0   Min.   :-973.0
##  1st Qu.:-182.00   1st Qu.: -616.0   1st Qu.:   2.0   1st Qu.: 191.0
##  Median : -39.00   Median : -378.0   Median : 591.0   Median : 511.0
##  Mean   : -55.29   Mean   : -312.6   Mean   : 380.1   Mean   : 393.6
##  3rd Qu.:  26.00   3rd Qu.:  -73.0   3rd Qu.: 737.0   3rd Qu.: 653.0
##  Max.   : 291.00   Max.   :  672.0   Max.   :1480.0   Max.   :1090.0
##  classe
##  A:5580
##  B:3797
##  C:3422
##  D:3216
##  E:3607
##
```

The summary statistics include minimum, 1st quartile, median, mean, 3rd quartile and maximum values, and thus roughly depict the spread of a numeric variables's values in the data frame `subdata`. (These numeric variables include three-axis acclerometer, gyroscope and magnetometer readings as well as Euler angles (roll, pitch and yaw)). It can be observed from the summary statistics that the numeric variables have values in different ranges. Therefore,some data preprocessing of predictors should be performed before data modeling. The last variable in `subdata` is called `classe` and is a categorical variable. The summary statistics for a categorical variable displays all the labels and their associated values, e.g., A, B, C, D, and E are the five different labels in the `classe` variable. According to the original study, the classes A, B, C, D and E correspond to dumbbell biceps curl performed exactly as specified, throwing the elbows to the front, lifting the dumbbell only halfway, lowering the dumbbell only half-way, and throwing the hips to the front, respectively.

The goal of the current project is to predict this `classe` variable (outcome) based on predictors in the dataframe (`subdata`). This represents a multi-label classification problem.

Feature selection

In order to reduce the dimensionality (thereby reducing the chance of over-fitting) and for efficient model training (shorter training time), irrlevant and redundant information should be removed as much as possible form the main data set (`subdata`) during the process of feature subset selection. Irrelevant and redundant information are also known to adversely affect the performance of common machine learning algorithms. To remove redundant information, variables that are highly correlated to each other are removed by using the `cor()` and `findCorrelation()` functions from the `caret` package. First a correlation matrix is created from the first 52 variables in `subdata` dataframe using `cor()` function. Then `findCorrelation()` function returns a vector of integers corresponding to variables to remove to reduce pair-wise correlations.

Removing redundant variables

```
# load caret package
library(caret)
# create a correlation matrix from predictor variables (first 52 variables)
correlationMatrix <- cor(subdata[,1:52])
# identify variables that are highly correlated to each other (ideally >
0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.5)
# print the result
print(highlyCorrelated)

##  [1] 10  1  9 22  4  3 36  8  2 11 29 37 35 38 30 47 21 34 39 52 23 25 13
## [24] 48 24 15 32 46 31 33 18
```

To remove the highly correlated variables from `subdata`, the `select()` function from `dplyr` package was used.

```
# load library dplyr
library(dplyr)
# select all columns from subdata, exclusing those found in the vector,
"highlyCorrelated"
subdata.new <- select(subdata, -highlyCorrelated)
```

During the first step of feature subset selection, the number of variables was reduced from 53 (in `subdata`) to 22 (in `subdata.new`).

Ranking variables by importance

`subdata.new` data set was split to create a training and a testing data set by using the `createDataPartition()` function from `caret` package. The `createDataPartition()` function conducts data splits within groups of data, which is `classe` variable for the `subdata.new` set. During data splitting, 75% of data was used to train the model (`training.new` set) and the remainder 25% of data was used for prediction (`testing.new`) set. While the data in `training.new` set was used to train the data and estimate model parameters, the `testing.new` set was used to get an independent assessment of model efficiency.

```
inTrain <- createDataPartition(y=subdata.new$classe, p=0.75, list = FALSE)
# create training.new set from subdata.new
training.new <- subdata.new[inTrain, ]
# create testing.new set from subdata.new
testing.new <- subdata.new[-inTrain, ]
```

In order to rank variables by their relative importance in `subdata.new`, a decision tree model was used. The decision tree model has a built-in mechanism to report variable importance. The `caret` package was used to train the data with a decison model model (by using `train()` function). `varImp()` from `caret` package was then used to estimate the variable importance, which is printed and plotted.

```
# set seed to ensure repeatable results

seed <- 7

# library caret is already loaded to work space

# trainControl() function sets teh resampling technique to be used during
data modeling

control <- trainControl(method="repeatedcv", number=10, repeats=3)

# data preprocessing

preProcess=c("center", "scale")

set.seed(seed)
```

```
# fit a decision tree model (rpart) to the classe variable using all 22 variables in training.new

fit.cart <- train(classe ~., data=training.new, preProcess = preProcess, method="rpart", trControl=control)

# estimate and print variable importance

importance <- varImp(fit.cartnew, scale=FALSE)

# print relative importance of variables

print(importance)

# plot relative importance of variables

plot(importance)
```

The results show that 11 variables are the most important ones out of 22 variables in `training.new`. These variables are `gyros_belt_z`, `magnet_belt_y`, `pitch_forearm`, `roll_forearm`, `roll_dumbbell`, `roll_arm`, `yaw_arm`, `magnet_arm_z`, `magnet_forearm_x`, `total_accel_arm`, and `accel_forearm_z`. The name of these 11 variables was used to create subsets from `training.new` and `testing.new`, where these 11 variables and the outcome `classe` variable were retained in the new data frames.

```
# filter 12 variables from 22 in training.new to create new dataframe, training.newsub
training.newsub <- training.new[, c("gyros_belt_z", "magnet_belt_y", "pitch_forearm", "roll_forearm", "roll_dumbbell", "roll_arm", "yaw_arm", "magnet_arm_z", "magnet_forearm_x", "total_accel_arm", "accel_forearm_z","classe")]

# filter 12 variables from 22 in training.new to create new dataframe, training.newsub
testing.newsub <- testing.new[, c("gyros_belt_z", "magnet_belt_y", "pitch_forearm", "roll_forearm", "roll_dumbbell", "roll_arm", "yaw_arm", "magnet_arm_z", "magnet_forearm_x", "total_accel_arm", "accel_forearm_z","classe")]
```

In this second step of feature extraction, 11 predictors were filtered (out of 21 in `training.new`) and subset dataframes were generated (`training.newsub` and `testing.newsub`).

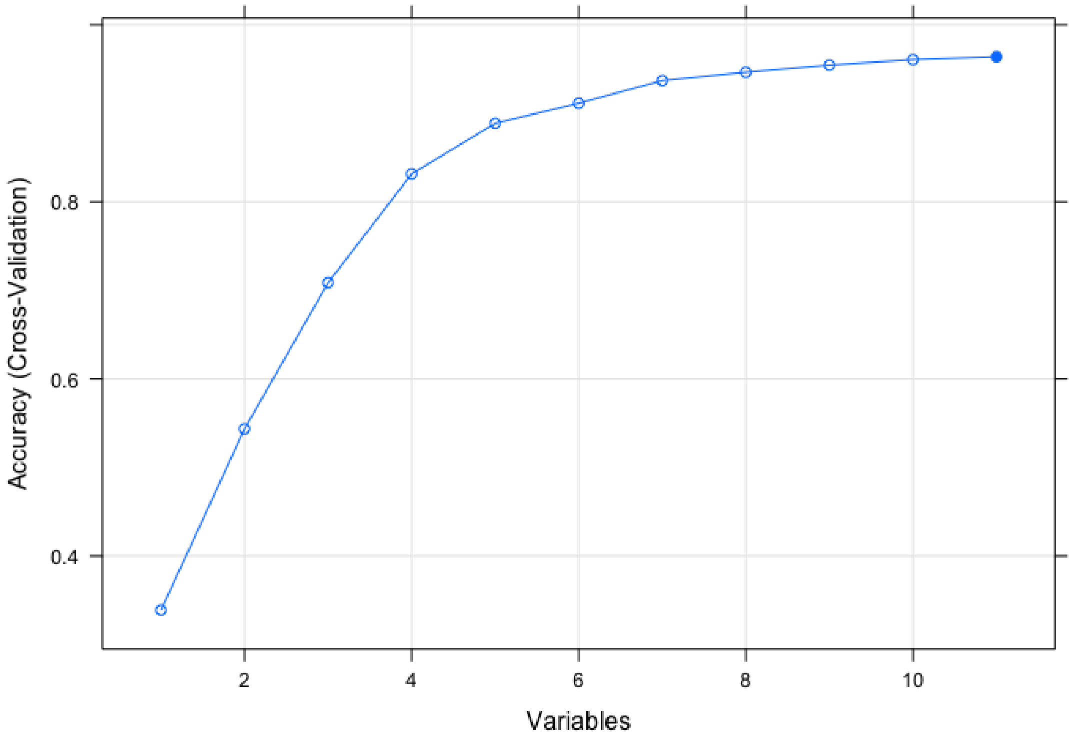
Automatic variable selection

Automatic feature selection methods can be used to identify the variables in a dataset that are required to build an accurate model. The `caret` package provides one of the popular feature selection method, namely recursive feature elimination of RFE. In this method, the algorithm first fits a model to all predictors. Each predictor is then ranked using its importance to the model and a performance profile for the predictors is generated. The RFE method was applied on `training.newsub`, where a random forest algorithm was used on each iteration to evaluate the model. The algorithm is configured to explore all possible subsets of the variables. All 11 predictors in `training.newsub` were selected. In the plot below, the accuracy of different variable subset sizes are shown. The name of the top five variables are printed.

```
# ensure repeatable results

set.seed(seed)

# define the control using a random forest selection function

control.new <- rfeControl(functions=rfFuncs, method="cv", number=10)

# run the RFE algorithm

results <- rfe(training.newsub[,1:11], training.newsub[,12], sizes=c(1:11),
rfeControl=control.new)

# summarize the results

print(results)

# plot the results

plot(results, type=c("g", "o"))
```

```
Recursive feature selection

Outer resampling method: Cross-Validated (10 fold)

Resampling performance over subset size:

 Variables Accuracy  Kappa AccuracySD  KappaSD Selected
         1   0.3390 0.1633   0.010106 0.012649
         2   0.5438 0.4184   0.025387 0.033199
         3   0.7088 0.6303   0.023170 0.028931
         4   0.8316 0.7864   0.008353 0.010638
         5   0.8887 0.8592   0.008031 0.010219
         6   0.9115 0.8881   0.007708 0.009803
         7   0.9370 0.9203   0.007032 0.008896
         8   0.9464 0.9322   0.006608 0.008357
         9   0.9544 0.9423   0.006638 0.008403
        10   0.9607 0.9503   0.005583 0.007062
        11   0.9638 0.9542   0.004539 0.005740        *

The top 5 variables (out of 11):
   roll_dumbbell, magnet_belt_y, pitch_forearm, roll_forearm, roll_arm
```

RFE method results summary



RFE method results

Based on the RFE method implemented for automatic feature selection, five predictors were filtered from `training.newsub` to create new subset, `training.newsubrfe`. Similarly a new subset was also created for `testing.newsub` by selecting the five predictors.

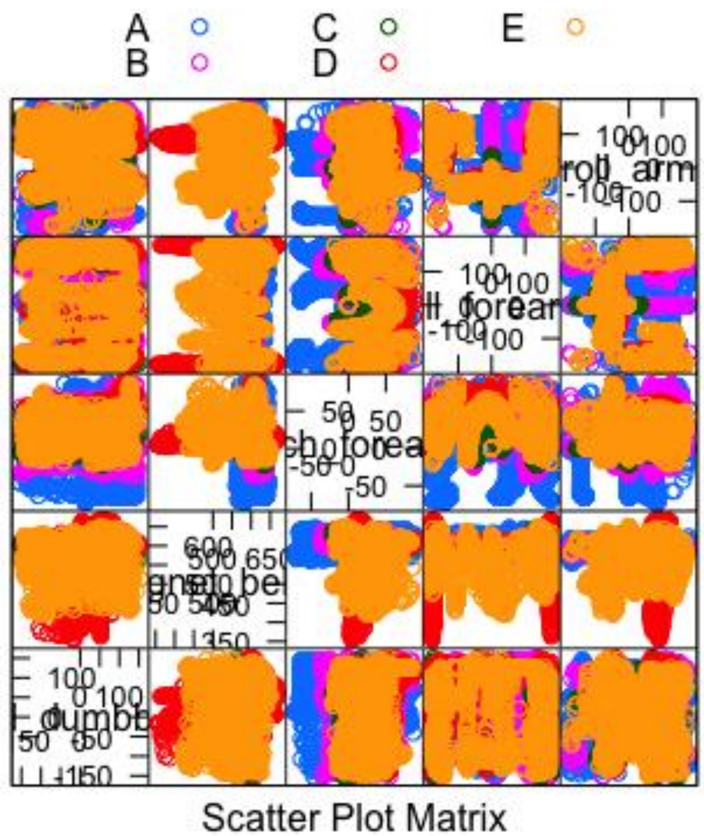
```
# select the five predictors from training.newsub to create a dataframe training.newsubrfe
training.newsubrfe <- training.newsub[, c("roll_dumbbell", "magnet_belt_y", "pitch_forearm", "roll_forearm", "roll_arm","classe")]
# select the five predictors from testing.newsub to create a dataframe testing.newsubrfe
testing.newsubrfe <- testing.newsub[, c("roll_dumbbell", "magnet_belt_y", "pitch_forearm", "roll_forearm", "roll_arm","classe")]
```

This concluded the three-step feature extraction process, where five important predictors were filtered from the original dataset to create a n new subset. This subset was then used during data modeling step.

Visualizing relationships among variables

To compare the distribution of the five predictors, a scatterplot matrix was generated that shows a grid of scatterplots where each variable is plotted against all other variables. It can be read by column or row, and each plot appears twice, allowing one to consider the spatial relationships from two perspectives. A scatterplot matrix is typically used to detect patterns among three or more variables. The class information from `classe` variable in the `training.newsubrfe` dataframe was included in the scatterplot matrix by coloring dots in each scatterplot by their class value (i.e., whether A, B, C, D or E). The command to create the scatterplot matrix from `training.newsubrfe` is shown below.

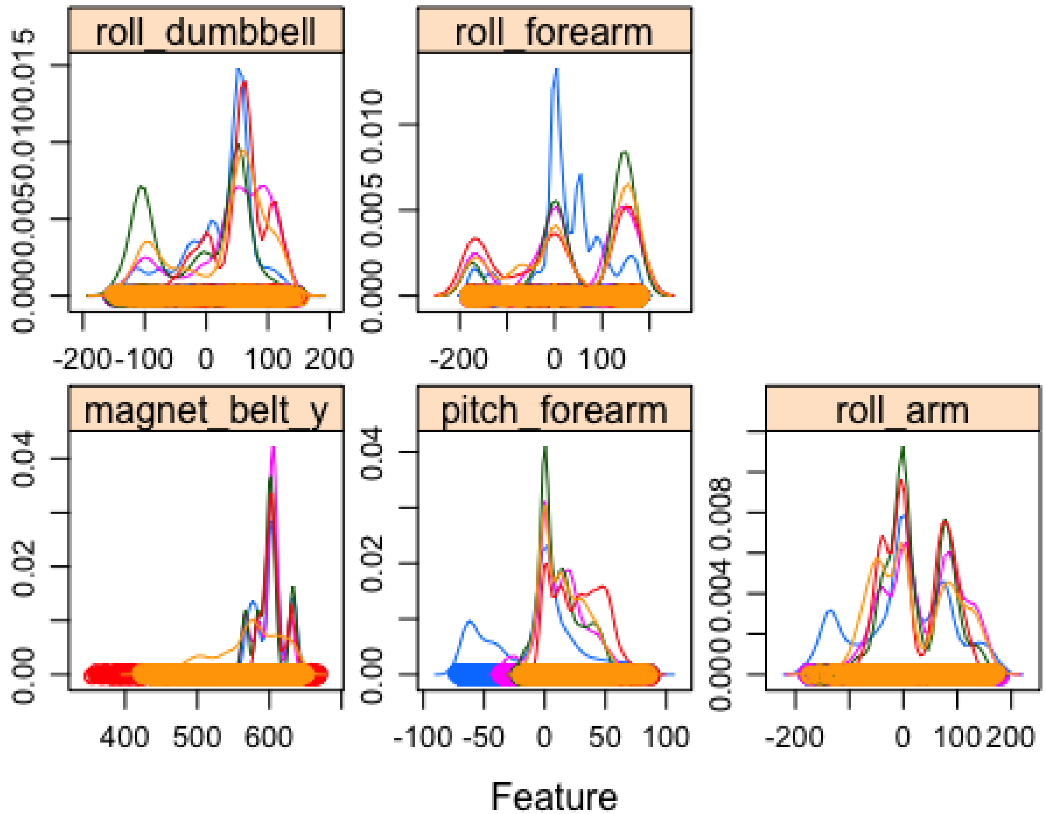
```
# create pair-wise plots of all 5 variables, dots colored by classe variable
featurePlot(x=training.newsubrfe[,1:5], y=training.newsubrfe[,6],
plot="pairs", auto.key=list(columns=3))
```



In the scatterplot matrix, the intersection of each row and column holds the scatterplot of the variables indicated by the row and column pair. The diagrams above and below the diagonal are transpositions since the x axis and y axis have been swapped. The scatterplot matrix plot shown above is not very clean. It can be seen that there is no clear distinction between data points separated by class label for different pair-wise plots and presence of severe overlapping. To probe further the overlapping of data points, density plot by class was also used to visualize the density distribution of each variable broken down by class value. Like the scatterplot matrix, the density plot by class can help see the separation/overlap of classes. It can also help to understand the overlap in class values for a variable. Here is the code for density plot by class:

```
# density plots for each variable by class value
x <- training.newsubrfe[,1:5]
y <- training.newsubrfe[,6]
```

```
scales <- list(x=list(relation="free"), y=list(relation="free"))
featurePlot(x=x, y=y, plot="density", scales=scales)
```



The density plots dispalys the severe overlappping between classes in the dataset. It is intuitive that traditional classification models will exhibit poor performance in predicting the `classe` variable for this multi-label classification problem. More sophisticated classification algorithms like random forest need to be used.

Data Modeling

Prediction of `classe` variable in this project represents a multi-label classification problem, for which three different models were tested:1. classification and regression tree (CART); 2. Bagging CART model; 3. Random forest. For each model, the default algorithm paramters were used as the `caret` package helps with estimating good defaults via automatic tuning functionality and the `tunelength` argument to the `train()` function. To avoid overfitting during training, resampling techniques are typically used. In this project, a 10-fold repeated cross validation with 3 repeats was used to find a robust estimate of the accuracy of the models. To have an honest comparison between models, it is important to ensure that each algorithm is evaluated on exactly the same splits of data. Therefore, a random number seed was assigned to a variable (`seed`), which can be used to reset the random number generator prior to training each of the algorithm. The evaluation metric `Accuracy` was used in the `train()` function for each model.

The code to fit the training data (`training.newsubrfe`) to the above-mentioned models is provided below.

```
# load caret package

library(caret)

control <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

seed <- 7

set.seed(seed)

preProcess=c("center", "scale")

# fit a decision tree model with rpart method from caret

fit.cartnewrfe <- train(classe ~., data=training.newsubrfe, preProcess =
preProcess, method="rpart", metric=metric, trControl=control)

# fit a decision tree model with Bagging CART method from caret package

fit.treebagrfe <- train(classe~., data=training.newsubrfe, preProcess =
preProcess, method="treebag",metric=metric,trControl=control)

# fit a random forst model with rf method from caret package

fit.rfrfe <- train(classe ~., data=training.newsubrfe, method="rf",
preProcess = preProcess, metric=metric, trControl=control)
```

The performance of the four models were compared and a quick summary of the results of the algorithms (in sample acuracy) were obtained as a table.

```
> results <- resamples(list(cart=fit.cartnewrfe, bagging=fit.treebagrfe, rf=fit.rfrfe))
> summary(results)

Call:
summary.resamples(object = results)

Models: cart, bagging, rf
Number of resamples: 30

Accuracy
          Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
cart    0.4752  0.4907 0.5088 0.5088  0.5245 0.5411    0
bagging 0.8533  0.8622 0.8688 0.8692  0.8774 0.8844    0
rf      0.8756  0.8840 0.8896 0.8903  0.8961 0.9082    0

Kappa
          Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
cart    0.3206  0.3398 0.3734 0.3697  0.3954 0.4163    0
bagging 0.8144  0.8257 0.8341 0.8346  0.8451 0.8540    0
rf      0.8428  0.8533 0.8606 0.8612  0.8687 0.8840    0
```

In-sample accuracy comparison

Comparison of accuracy for the three models reveal that bagging and random forest algorithms perform better in predicting the outcome.A random forest model exhibits best performance (89%) among the three methods and the decision tree models has lowest performance (50%). For further investigation, the trained models were used to predict outcome of the testing (`testing.newsubrfe`) set. The predicted outcome is then compared to the actual outcome and out-of-sample accuracy were computed for various models.

Here is the predicted outcome for decision tree model:

```
> pred.cartrfe <- predict(fit.cartnewrfe, testing.newsubrfe)
> confusionMatrix(pred.cartrfe, testing.newsubrfe$classe)
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1161  353  372  208  191
         B   61  270   51   96   95
         C   96  200  393  187  190
         D   47  123    1  241   36
         E   30    3   38   72  389

Overall Statistics

               Accuracy : 0.5004
                 95% CI : (0.4863, 0.5145)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3529
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.8323  0.28451  0.45965  0.29975  0.43174
Specificity            0.6797  0.92339  0.83379  0.94951  0.96428
Pos Pred Value         0.5081  0.47120  0.36867  0.53795  0.73120
Neg Pred Value         0.9107  0.84322  0.87962  0.87365  0.88289
Prevalence             0.2845  0.19352  0.17435  0.16395  0.18373
Detection Rate         0.2367  0.05506  0.08014  0.04914  0.07932
Detection Prevalence   0.4659  0.11684  0.21737  0.09135  0.10848
Balanced Accuracy      0.7560  0.60395  0.64672  0.62463  0.69801
```

out-of sample accuracy with decision tree model

Here is the predicted outcome for bagging decision tree model:

```
> pred.treebagrfe <- predict(fit.treebagrfe, testing.newsubrfe)
> confusionMatrix(pred.treebagrfe, testing.newsubrfe$classe)
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1285   60   31   13   10
         B   63  786   47   25   29
         C   20   56  729   52   29
         D   12   28   34  701   17
         E   15   19   14   13  816

Overall Statistics

               Accuracy : 0.8803
                 95% CI : (0.8709, 0.8893)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8486
 Mcnemar's Test P-Value : 0.09797

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9211   0.8282   0.8526   0.8719   0.9057
Specificity            0.9675   0.9585   0.9612   0.9778   0.9848
Pos Pred Value         0.9185   0.8274   0.8228   0.8851   0.9304
Neg Pred Value         0.9686   0.9588   0.9686   0.9750   0.9789
Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
Detection Rate         0.2620   0.1603   0.1487   0.1429   0.1664
Detection Prevalence   0.2853   0.1937   0.1807   0.1615   0.1788
Balanced Accuracy      0.9443   0.8934   0.9069   0.9248   0.9452
```

out-of sample accuracy with bagging decision tree model

Here is the predicted outcome for random forest model:

```
> pred.rfrfe <- predict(fit.rfrfe, testing.newsubrfe)
> confusionMatrix(pred.rfrfe, testing.newsubrfe$classe)
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1310   67   26    7   11
         B   42  774   42   14   26
         C   21   61  752   48   32
         D   16   29   26  724   14
         E    6   18    9   11  818

Overall Statistics

               Accuracy : 0.8927
                 95% CI : (0.8837, 0.9013)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8643
 Mcnemar's Test P-Value : 1.018e-05

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9391   0.8156   0.8795   0.9005   0.9079
Specificity            0.9684   0.9686   0.9600   0.9793   0.9890
Pos Pred Value         0.9219   0.8619   0.8228   0.8949   0.9490
Neg Pred Value         0.9756   0.9563   0.9742   0.9805   0.9795
Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
Detection Rate         0.2671   0.1578   0.1533   0.1476   0.1668
Detection Prevalence   0.2898   0.1831   0.1864   0.1650   0.1758
Balanced Accuracy      0.9537   0.8921   0.9198   0.9399   0.9484
```

out-of sample accuracy with random forest model

The out-of-sample accuracy for the classification tree (CART), bagging CART, and Random forest models were found to be 50%, 88%, and 89%, respectively. Typically, in-sample accuracy is expected to be greater than the out-of-sample accuracy. While it is observed that the out-of-sample accuracy is more or less similar to the in-sample accuracy, the Random forest is the best model with respect to prediction.

In a further investigation, the random forest model was also used to fit training data from `subdata` dataframe (after splitting the data frame to training and testing set, similar to splitting subdata.new dataframe shown previously). The `subdata` dataframe contains 53 variables.

Here is the predicted outcome for random forest model when fittied to dataset with 53 variables:

```
> pred4 <- predict(fit.rf, testing)
> confusionMatrix(pred4, testing$classe)
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1394    4    0    0    0
         B    0  942   13    0    0
         C    0    3  838    9    1
         D    1    0    4  794    1
         E    0    0    0    1  899

Overall Statistics

               Accuracy : 0.9925
                 95% CI : (0.9896, 0.9947)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9905
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9993   0.9926   0.9801   0.9876   0.9978
Specificity            0.9989   0.9967   0.9968   0.9985   0.9998
Pos Pred Value         0.9971   0.9864   0.9847   0.9925   0.9989
Neg Pred Value         0.9997   0.9982   0.9958   0.9976   0.9995
Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
Detection Rate         0.2843   0.1921   0.1709   0.1619   0.1833
Detection Prevalence   0.2851   0.1947   0.1735   0.1631   0.1835
Balanced Accuracy      0.9991   0.9947   0.9885   0.9930   0.9988
```

Random forest model prediction

Interestingly, the random forest models exhibited 99% accuracy. Therefore, it can be concluded that the variables that were excluded from the data set (during feature extraction process) are useful to build a more accurate model. With an impressive 99% out-of-sample accuracy, the random forest model was then used for a final evaluation against the 20 test cases in the `data2` data frame (not used during model training). The trained random forest model exhibited 100% accuracy in predicting the outcome (i.e., the classe variable).

Conclusion

In this project, three different classification models were tested to predict outcome, which is the quality of execution of weight lifting exercises. The Random forest model was found to be the best model during training and cross-validation step. When this model was used to predict the outcome for the 20 test cases, 100% accuracy was observed.