Machine Learning Project - Body Activity Data

Willem Abrie

2023-02-05

Project Brief

The following is mostly quoted from the *Practical Machine Learning* course project instructions or the information links contained therein:

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. 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, the goal is 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.

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

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience.

(Read more:)[http:/groupware.les.inf.puc-rio.br/har#literature#ixzz4TjrBbK00] (Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.)[http://web.archive.org/web/20161217164008/http://groupware.les.inf.puc-rio.br/work.jsf?p1=11201]

Executive Summary

This project was used as a data science learning opporutinty. The data was explored and plotted, then wrangled and modelled. Variables with little or no data in them were excluded from the analysis. A validation set was set aside using 25% of the testing data.

Various models were trained and compared, including LDA, CART and RF models. The most successful was a tuned random forest which resulted in 100% out-of-sample accuracy.

Key Learnings

- Don't include index or time stamp variables when training decision trees. This allows them to find patterns that won't be present in future data.
- Skewed variables, high correlation between some variables and outliers in the data doesn't necessarily pose a problem for classification models. It wasn't a problem in this case.
- Optimising decision tree tuning parameters can make a big difference in model run time.
- Investigating model plots and variable importance after models have been trained is an important step in the process and can lead to identification of model weaknesses

System Info

The following version of RStudio was used: RStudio 2022.07.2 Build 576. For more info on packages see the session info in the Appendix.

Data Gathering

Data was downloaded directly from the links that were provided with the assignment.

```r
rm(list = ls())
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
dim(training)
```

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

```r
testing <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
dim(testing)
```

```
## [1]  20 160
```

Data Exploration

First we explored the data to understand it a bit better. Some select bits are shown below as a sample.

```
set.seed(100)
sampleData <- training[c(sample(x = 1:dim(training)[1], size = 5)), c(1:9, 159:160)]
print("A random sample of rows to show what the data looks like..a lot of sensor output variables have been excluded...")
```

```
## [1] "A random sample of rows to show what the data looks like..a lot of sensor output variables have been excluded..."
```

```
sampleData
```

```
##           X user_name raw_timestamp_part_1 raw_timestamp_part_2
## 16887 16887   charles           1322837934               808273
## 3430   3430    jeremy           1322673048               882659
## 3696   3696    jeremy           1322673060               746720
## 3052   3052  carlitos           1323084259               168289
## 11159 11159    jeremy           1322673117               462788
##         cvtd_timestamp new_window num_window roll_belt pitch_belt
## 16887 02/12/2011 14:58         no        146    134.00       8.89
## 3430  30/11/2011 17:10         no        413      1.25       4.72
## 3696  30/11/2011 17:11         no        424      0.98       4.14
## 3052  05/12/2011 11:24         no        341      1.67       7.97
## 11159 30/11/2011 17:11         no        476     -0.23       7.34
##       magnet_forearm_z classe
## 16887              236      E
## 3430               682      A
## 3696               505      A
## 3052               511      A
## 11159              746      C
```

```
print("Let's see how many of the observations correspond to each class of the excercise, grouped by participant.")
```

```
## [1] "Let’s see how many of the observations correspond to each class of the excercise, grouped by participant."
```

```
table(training$user_name, training$classe)
```

```
##
##               A    B    C    D    E
##   adelmo   1165  776  750  515  686
##   carlitos  834  690  493  486  609
##   charles   899  745  539  642  711
##   eurico    865  592  489  582  542
##   jeremy   1177  489  652  522  562
##   pedro     640  505  499  469  497
```

It is unclear if each observation corresponds to a full repetition of the exercise or whether multiple observations can form part of a single exercise repetition. I.e. we want to figure out how the data is set up with the timestamps and window labelling.

We'll construct some plots to help clear this up. But first let's do some processing to set some things straight..

We combine the two separate data sets that were provided so that we perform the same processing on both. We'll split them again before training an algorithm:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#combine data
training1 <- training %>%
        mutate(cat = "train")
testing1 <- testing %>%
        mutate(classe = NA)  %>%
```

```
        mutate(cat = "test") %>%
        select(-problem_id)

print("Dimensions of training and testing set:")
```

```
## [1] "Dimensions of training and testing set:"
```

```
dim(training1)
```

```
## [1] 19622   161
```

```
dim(testing1)
```

```
## [1]  20 161
```

```
traintest <- rbind(training1,testing1)
```

We get the variables in the right classes and in useful formats:

```
#Convert non-numeric variables to factors
traintest$user_name <- factor(traintest$user_name)
traintest$classe <- factor(traintest$classe)

#Convert the two Unix epoch time stamps to microsecond POSIXct times
#first pad the second UNIX time stamp variable with zeros so all entries have six digits

traintest <- traintest %>%
        mutate(raw_timestamp_part_2 = stringr::str_pad(string = raw_timestamp_part_2,
                                                       width = 6, pad = "0")) %>%
        mutate(time = as.POSIXct(as.numeric(paste0(raw_timestamp_part_1, raw_timestamp_part_2))*1e-6,
                                origin="1970-01-01"), .before = roll_belt) %>%
        select(-c(raw_timestamp_part_1, raw_timestamp_part_2)) %>%
        relocate(classe, .before = roll_belt)

#set options to see microseconds
options(digits.secs = 6)

print("Having a useful time stamp variable allows us to easily show information summaries like:")
```

```
## [1] "Having a useful time stamp variable allows us to easily show information summaries like:"
```

```
traintest %>%
        filter(cat == "train") %>%
        group_by(user_name) %>%
        summarise(StartTime=min(time), EndTime=max(time), Duration = EndTime-StartTime)
```

```
## # A tibble: 6 x 4
##   user_name StartTime                  EndTime                    Duration
##   <fct>     <dttm>                     <dttm>                     <drtn>
## 1 adelmo    2011-12-02 23:32:52.092295 2011-12-02 23:35:45.492326 2.890001 mins
## 2 carlitos  2011-12-05 21:23:51.788290 2011-12-05 21:25:56.664311 2.081267 mins
## 3 charles   2011-12-03 00:56:48.428308 2011-12-03 00:59:22.832342 2.573401 mins
## 4 eurico    2011-11-29 00:13:25.734802 2011-11-29 00:15:30.770669 2.083931 mins
## 5 jeremy    2011-12-01 03:10:25.254730 2011-12-01 03:12:46.022810 2.346135 mins
## 6 pedro     2011-12-06 00:22:48.872380 2011-12-06 00:24:41.568295 1.878265 mins
```

```
traintest %>%
        filter(cat == "train", user_name == "adelmo") %>%
        group_by(user_name, classe) %>%
        summarise(StartTime=min(time), EndTime=max(time), Duration = EndTime-StartTime)
```

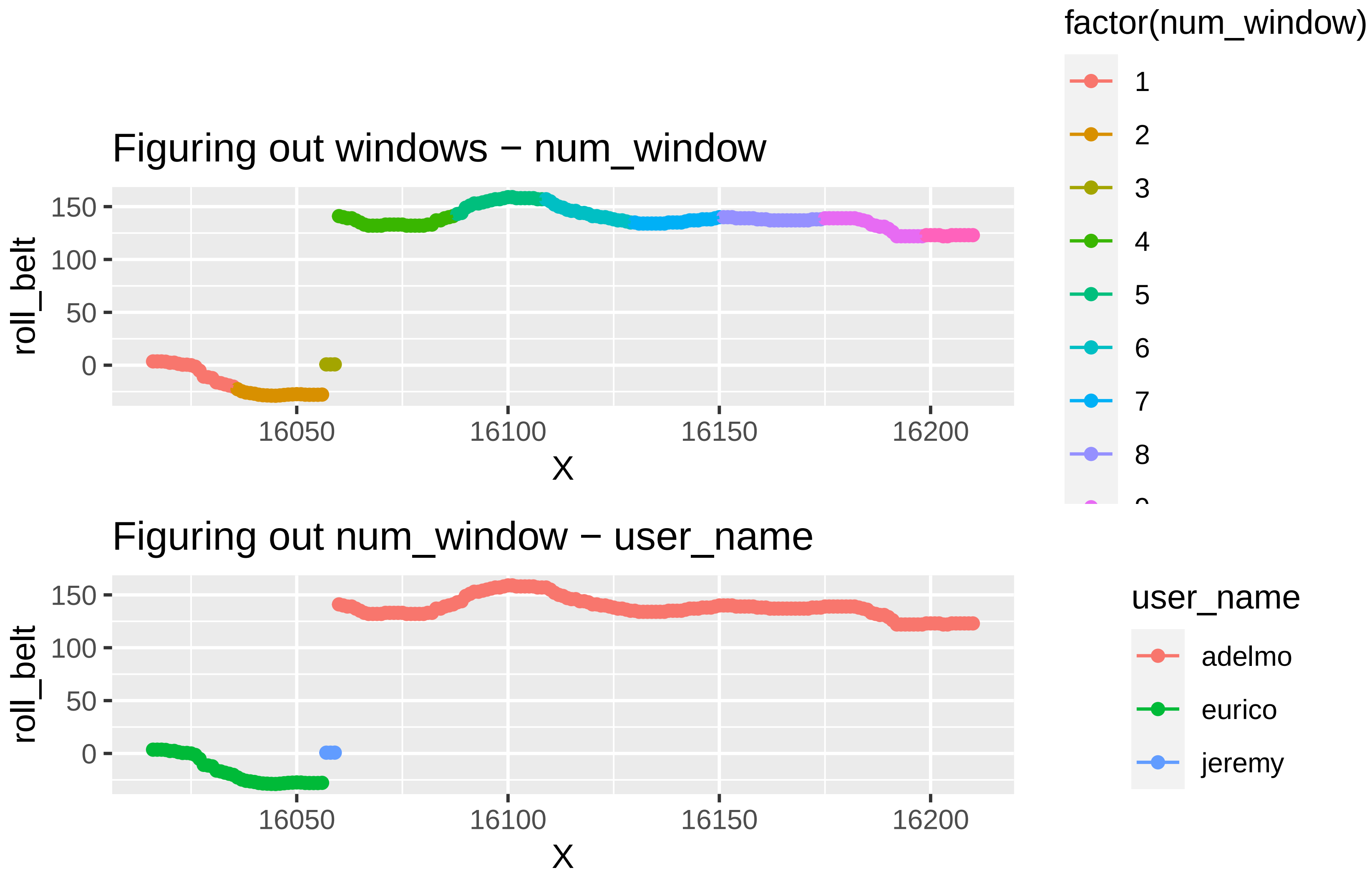
```
## `summarise()` has grouped output by 'user_name'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 5 x 5
## # Groups:   user_name [1]
##   user_name classe StartTime                  EndTime                    Durat~1
##   <fct>     <fct>  <dttm>                     <dttm>                     <drtn>
## 1 adelmo    A      2011-12-02 23:32:52.092295 2011-12-02 23:33:44.752298 52.660~
## 2 adelmo    B      2011-12-02 23:33:44.868370 2011-12-02 23:34:18.520397 33.652~
## 3 adelmo    C      2011-12-02 23:34:18.540395 2011-12-02 23:34:51.300323 32.759~
## 4 adelmo    D      2011-12-02 23:34:51.332291 2011-12-02 23:35:14.952371 23.620~
## 5 adelmo    E      2011-12-02 23:35:15.060390 2011-12-02 23:35:45.492326 30.431~
## # ... with abbreviated variable name 1: Duration
```

From the tables above we can see that each participant was only transmitting data for a couple of minutes and less than a minute was spent doing the lifts in each category. Each participant was logged one at a time and it took a number of days to get them all in for their short sessions.

We do some plots to inspect how the data windows were set up, referring to the *num_window* and *new_window* variables:

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   0.3.5
## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.1     v forcats 0.5.2
## v readr   2.1.3
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```



We can see from the tables above that the data was captured over a number of days, one participant at a time. Then the data was patched together in “windows” to more or less correspond to each participants signal data from classe A to D.

Judging by the type of plot shown above for the first 10 windows, it seems from the continuity of the data that multiple observations (data frame rows) read together give the motion signature of the class of lift for the relevant participant. No databook or units of measure were given with the data, but the accelerometer data entries seem to be instantaneous measurements and discrete time points and not average values for the lift.

The X variable seems to be some sort of index variable but the data arrangement by X is scattered and it doesn’t immediately seem fully useful for ordering.

The “windows” that the observations are grouped in also don’t immediately seem very useful. Once again the ordering seems haphazard. We suspect we can ingore the variables *X*, *num_window* and *new_window* in our classification training.

We notice that in the data set it appears we have lots of missing values, some character variables that will have no predictive value and some character variables that should be used as factors.

Some data cleaning is in order before we start inspecting data plots.

Data Wrangling

The columns with little or no information were dropped. The classe and user name variables were converted to factors. To apply the same processing to the testing data, we combined the data first and split it out again at the end.

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
#Step 1: clean data by removing columns with no data
#exclude the time variable for consideration in this step since the POSIX format doesnt work here
traintest_remove1 <- subset(traintest, select = -time)
#check for no data
traintest_remove1 <- traintest_remove1[sapply(traintest_remove1, function(x) all(x == "" || is.na(x)))]

print(paste("the following ", dim(traintest_remove1)[2], " variables have no data: " ))
```

```
## [1] "the following  100  variables have no data: "
```

```
colnames(traintest_remove1)
```

```
##   [1] "kurtosis_roll_belt"       "kurtosis_picth_belt"
##   [3] "kurtosis_yaw_belt"        "skewness_roll_belt"
##   [5] "skewness_roll_belt.1"     "skewness_yaw_belt"
##   [7] "max_roll_belt"            "max_picth_belt"
##   [9] "max_yaw_belt"             "min_roll_belt"
##  [11] "min_pitch_belt"           "min_yaw_belt"
##  [13] "amplitude_roll_belt"      "amplitude_pitch_belt"
##  [15] "amplitude_yaw_belt"       "var_total_accel_belt"
```

```
## [17] "avg_roll_belt"            "stddev_roll_belt"
## [19] "var_roll_belt"            "avg_pitch_belt"
## [21] "stddev_pitch_belt"        "var_pitch_belt"
## [23] "avg_yaw_belt"             "stddev_yaw_belt"
## [25] "var_yaw_belt"             "var_accel_arm"
## [27] "avg_roll_arm"             "stddev_roll_arm"
## [29] "var_roll_arm"             "avg_pitch_arm"
## [31] "stddev_pitch_arm"         "var_pitch_arm"
## [33] "avg_yaw_arm"              "stddev_yaw_arm"
## [35] "var_yaw_arm"              "kurtosis_roll_arm"
## [37] "kurtosis_picth_arm"       "kurtosis_yaw_arm"
## [39] "skewness_roll_arm"        "skewness_pitch_arm"
## [41] "skewness_yaw_arm"         "max_roll_arm"
## [43] "max_picth_arm"            "max_yaw_arm"
## [45] "min_roll_arm"             "min_pitch_arm"
## [47] "min_yaw_arm"              "amplitude_roll_arm"
## [49] "amplitude_pitch_arm"      "amplitude_yaw_arm"
## [51] "kurtosis_roll_dumbbell"   "kurtosis_picth_dumbbell"
## [53] "kurtosis_yaw_dumbbell"    "skewness_roll_dumbbell"
## [55] "skewness_pitch_dumbbell"  "skewness_yaw_dumbbell"
## [57] "max_roll_dumbbell"        "max_picth_dumbbell"
## [59] "max_yaw_dumbbell"         "min_roll_dumbbell"
## [61] "min_pitch_dumbbell"       "min_yaw_dumbbell"
## [63] "amplitude_roll_dumbbell"  "amplitude_pitch_dumbbell"
## [65] "amplitude_yaw_dumbbell"   "var_accel_dumbbell"
## [67] "avg_roll_dumbbell"        "stddev_roll_dumbbell"
## [69] "var_roll_dumbbell"        "avg_pitch_dumbbell"
## [71] "stddev_pitch_dumbbell"    "var_pitch_dumbbell"
## [73] "avg_yaw_dumbbell"         "stddev_yaw_dumbbell"
## [75] "var_yaw_dumbbell"         "kurtosis_roll_forearm"
## [77] "kurtosis_picth_forearm"   "kurtosis_yaw_forearm"
## [79] "skewness_roll_forearm"    "skewness_pitch_forearm"
## [81] "skewness_yaw_forearm"     "max_roll_forearm"
## [83] "max_picth_forearm"        "max_yaw_forearm"
## [85] "min_roll_forearm"         "min_pitch_forearm"
## [87] "min_yaw_forearm"          "amplitude_roll_forearm"
## [89] "amplitude_pitch_forearm"  "amplitude_yaw_forearm"
## [91] "var_accel_forearm"        "avg_roll_forearm"
## [93] "stddev_roll_forearm"      "var_roll_forearm"
```

```
##  [95] "avg_pitch_forearm"        "stddev_pitch_forearm"
##  [97] "var_pitch_forearm"        "avg_yaw_forearm"
##  [99] "stddev_yaw_forearm"       "var_yaw_forearm"
```

```
#Step 2: remove variables that have too many NA, ignore time variable and variables identified in
#subset relevant columns
traintest_remove2 <- traintest[,!names(traintest) %in% c("time", colnames(traintest_remove1))]
#test for % NA's
traintest_remove2 <- traintest_remove2[sapply(traintest_remove2, function(x) sum(is.na(x))/dim(traintest)[1] > 0.30) ]

print(paste("the following ", dim(traintest_remove2)[2], " variables have mostly NA's: " ))
```

```
## [1] "the following  0  variables have mostly NA's: "
```

```
colnames(traintest_remove2)
```

```
## character(0)
```

```
#percentNA <- sapply(traintest_remove2, function(x) sum(is.na(x))/dim(traintest)[1])
#df_NAs <- data.frame("Variable" = colnames(traintest_remove2), "Percent NA" = percentNA)
#df_NAs

#Step 3: remove variables with near-zero variance. be careful not to remove the classe variable and ignore variables from previous two ste
nzv <- nearZeroVar(traintest[!names(traintest) %in% c("time", colnames(traintest_remove1), colnames(traintest_remove2))])
traintest_remove3 <- traintest[,nzv]
print(paste("the following ", dim(traintest_remove3)[2], " variables have near-zero variance: " ))
```

```
## [1] "the following  2  variables have near-zero variance: "
```

```
colnames(traintest_remove3)
```

```
## [1] "new_window"  "var_yaw_arm"
```

```
#Step 4: after inspection, remove further useless variables
traintest_remove4 <- subset(traintest, select = c(cvtd_timestamp, num_window))
print(paste("after inspection, the following ", dim(traintest_remove4)[2], " useless predictors will be removed: "))
```

```
## [1] "after inspection, the following  2  useless predictors will be removed: "
```

```
colnames(traintest_remove4)
```

```
## [1] "cvtd_timestamp" "num_window"
```

```
#remove duplicates (some columns could have fit more than one criteria)

traintest_remove <- unique(c(colnames(traintest_remove1), colnames(traintest_remove2), colnames(traintest_remove3), colnames(traintest_rem
print(paste(length(traintest_remove), "variables will be removed."))
```

```
## [1] "103 variables will be removed."
```

```
#remove all the unwanted variables to get a model sub-set
traintest_mss <- traintest[!names(traintest) %in% traintest_remove]
print("The remaining variables for modelling are: ")
```

```
## [1] "The remaining variables for modelling are: "
```

```
str(traintest_mss)
```

```
## 'data.frame':    19642 obs. of  57 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 ...
##  $ time                : POSIXct, format: "2011-12-05 21:23:51.788290" "2011-12-05 21:23:51.808297" ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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 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 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ 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 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
```

```
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ cat                 : chr  "train" "train" "train" "train" ...
```

Split off the test set again before imputing any missing data or any training algorithms are applied.

```
#split off train and test set
train <- traintest_mss %>%
        filter(cat == "train") %>%
        select(-cat)
test <- traintest_mss %>%
        filter(cat == "test") %>%
        select(-c(cat, classe))

#summary(traintest)
print("New dimensions of training and testing set:")
```

```
## [1] "New dimensions of training and testing set:"
```

```
dim(train)
```

```
## [1] 19622    56
```

```
dim(test)
```

```
## [1] 20 55
```

The classe variable was removed for the small 20-observation testing set, since this information was not provided. The classe predictions were submitted as a course quizz and graded through a web form.

Check for remaining missing values and impute - treat testing and training set separatedly

```
#First remove rows (if any) where the outcome variable is missing, unless they can be figured out by patterns in the data
dim(train[is.na(train$classe),])[1]
```

```
## [1] 0
```

```r
#Mark any further missing values, especially empty characters.

missingVals_train <- sapply(train, function(x) sum(is.na(x)))
print("Training set: There as so many NAs per column:")
```

```
## [1] "Training set: There as so many NAs per column:"
```

```r
missingVals_train[missingVals_train>0]
```

```
## named integer(0)
```

```r
missingVals_test <- sapply(test, function(x) sum(is.na(x)))
print("Training set: There as so many NAs per column:")
```

```
## [1] "Training set: There as so many NAs per column:"
```

```r
missingVals_test[missingVals_test>0]
```

```
## named integer(0)
```

There are no further missing values to deal with in the train or test set.

Visualising the Data

A typical plot of the data looks like this:

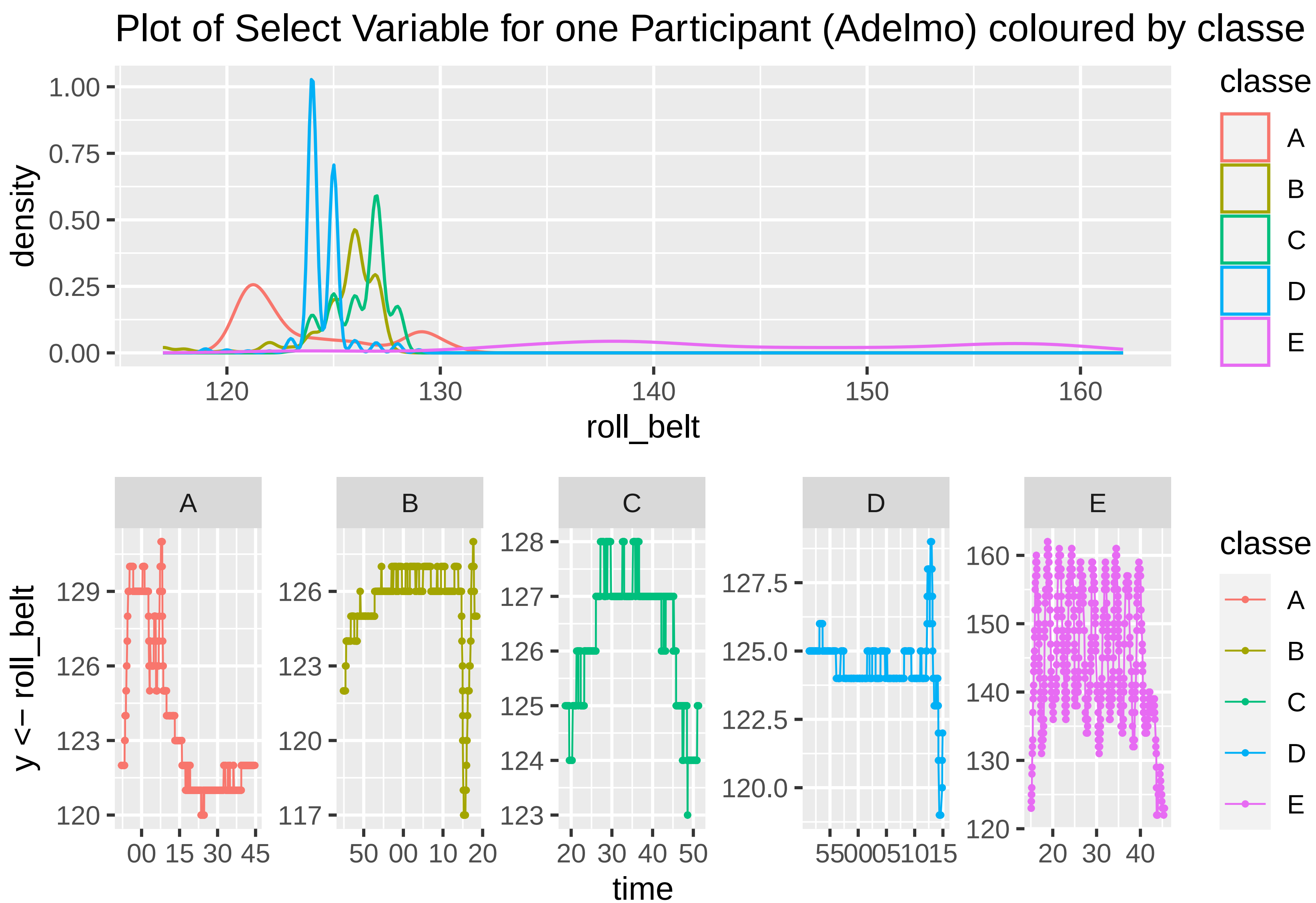
```
library(patchwork)

density1 <- train %>%
        filter(user_name == "adelmo") %>%
        group_by(classe) %>%
        ggplot(mapping = aes(x = roll_belt, colour = classe)) +
        geom_density()+
        ggtitle("Plot of Select Variable for one Participant (Adelmo) coloured by classe")
points1 <- train %>%
        filter(user_name == "adelmo") %>%
        group_by(classe) %>%
        ggplot(mapping = aes(x = time, y <- roll_belt, colour = classe)) +
        geom_point(size=0.5)+
        geom_line(size=0.3)+
        facet_wrap(vars(classe), nrow = 1, scales = "free" )
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

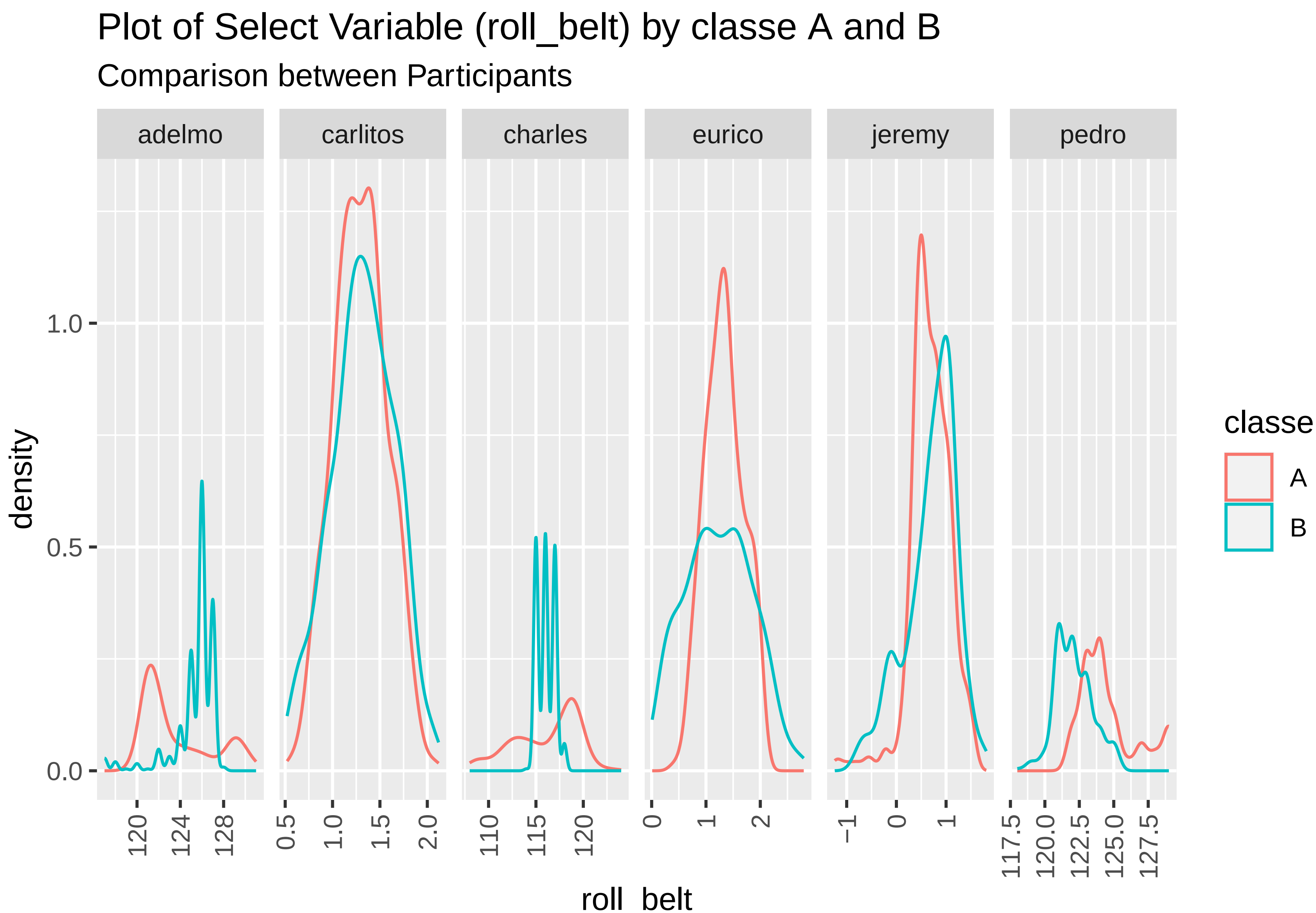
```
        ggtitle("Plot of Select Variable for one Participant coloured by classe")
```

```
## $title
## [1] "Plot of Select Variable for one Participant coloured by classe"
##
## attr(,"class")
## [1] "labels"
```

```
density1 / points1
```


Plot of Select Variable for one Participant (Adelmo) coloured by classe
density
roll_belt
classe
A
B
C
D
E
y <− roll_belt
time

```
train %>%
        filter(classe == c("A", "B")) %>%
        group_by(classe) %>%
        ggplot(mapping = aes(x = roll_belt, colour = classe)) +
        geom_density()+
        ggtitle("Plot of Select Variable (roll_belt) by classe A and B", subtitle = "Comparison between Participants")+
        facet_wrap(vars(user_name), nrow = 1, scales = "free_x")+
        theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

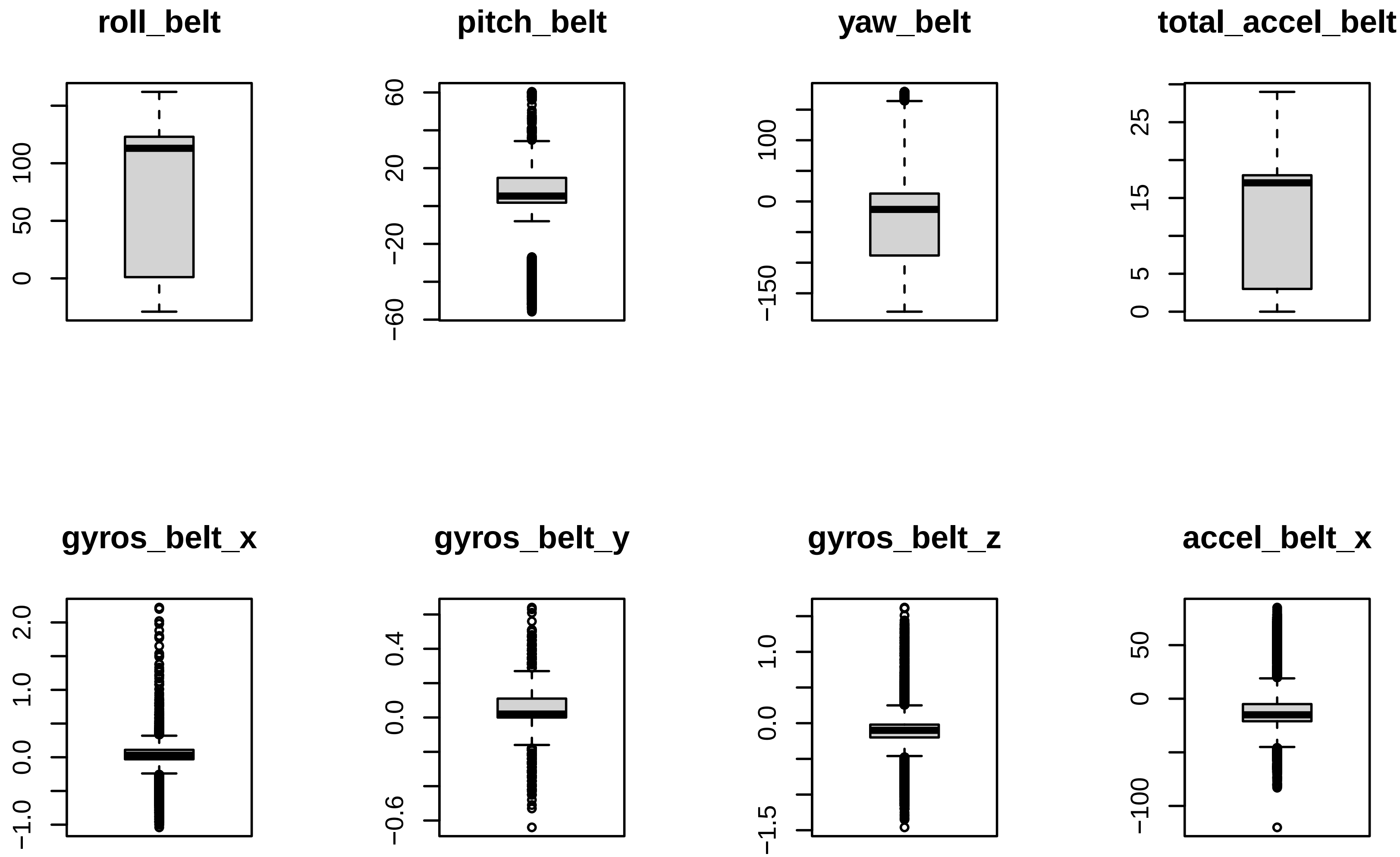


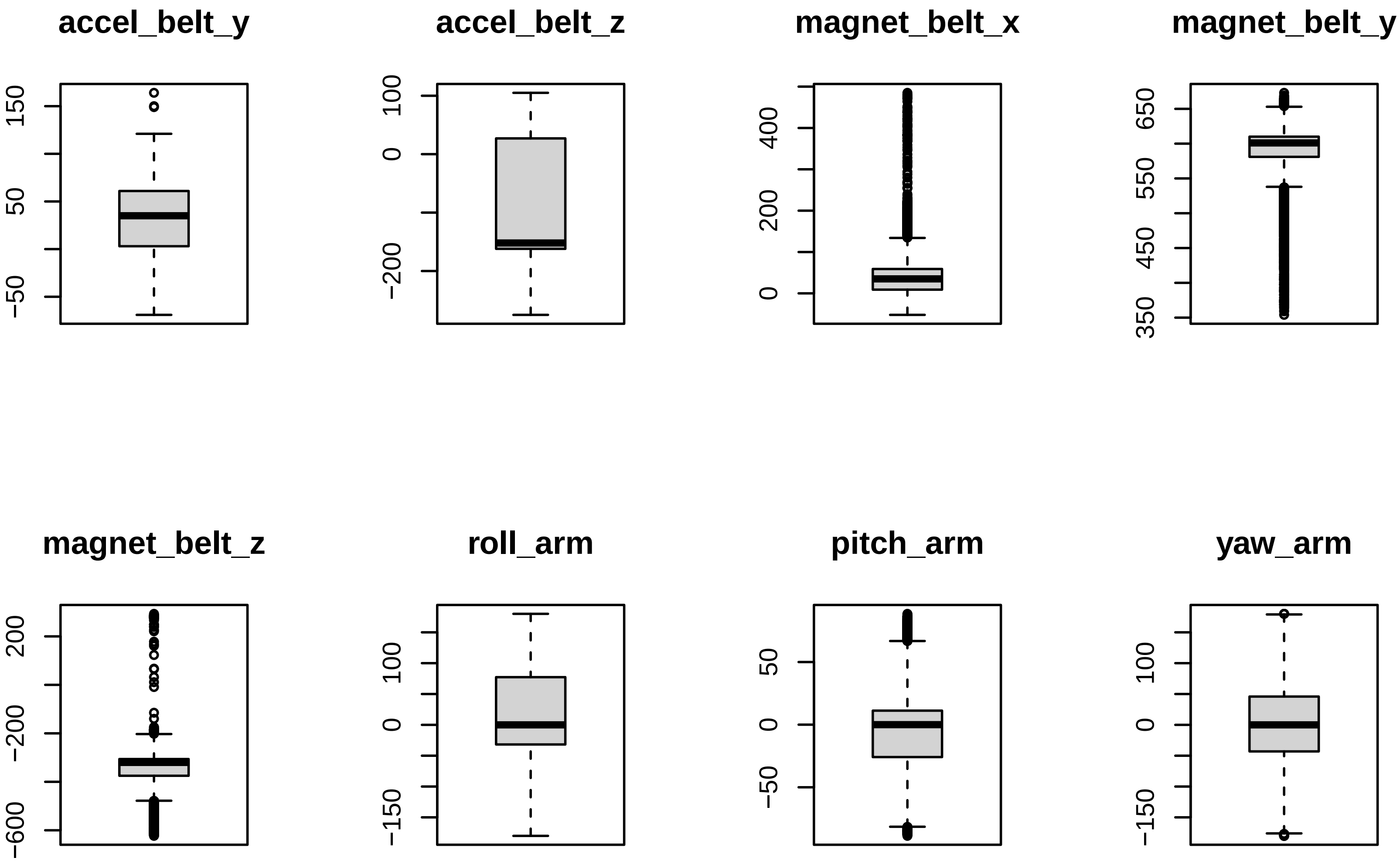
Plot of Select Variable (roll_belt) by classe A and B
Comparison between Participants
adelmo
carlitos
charles
eurico
jeremy
pedro
density
1.0
0.5
0.0
120
124
128
0.5
1.0
1.5
2.0
110
115
120
0
1
2
−1
0
1
117.5
120.0
122.5
125.0
127.5
roll_belt
classe
A
B

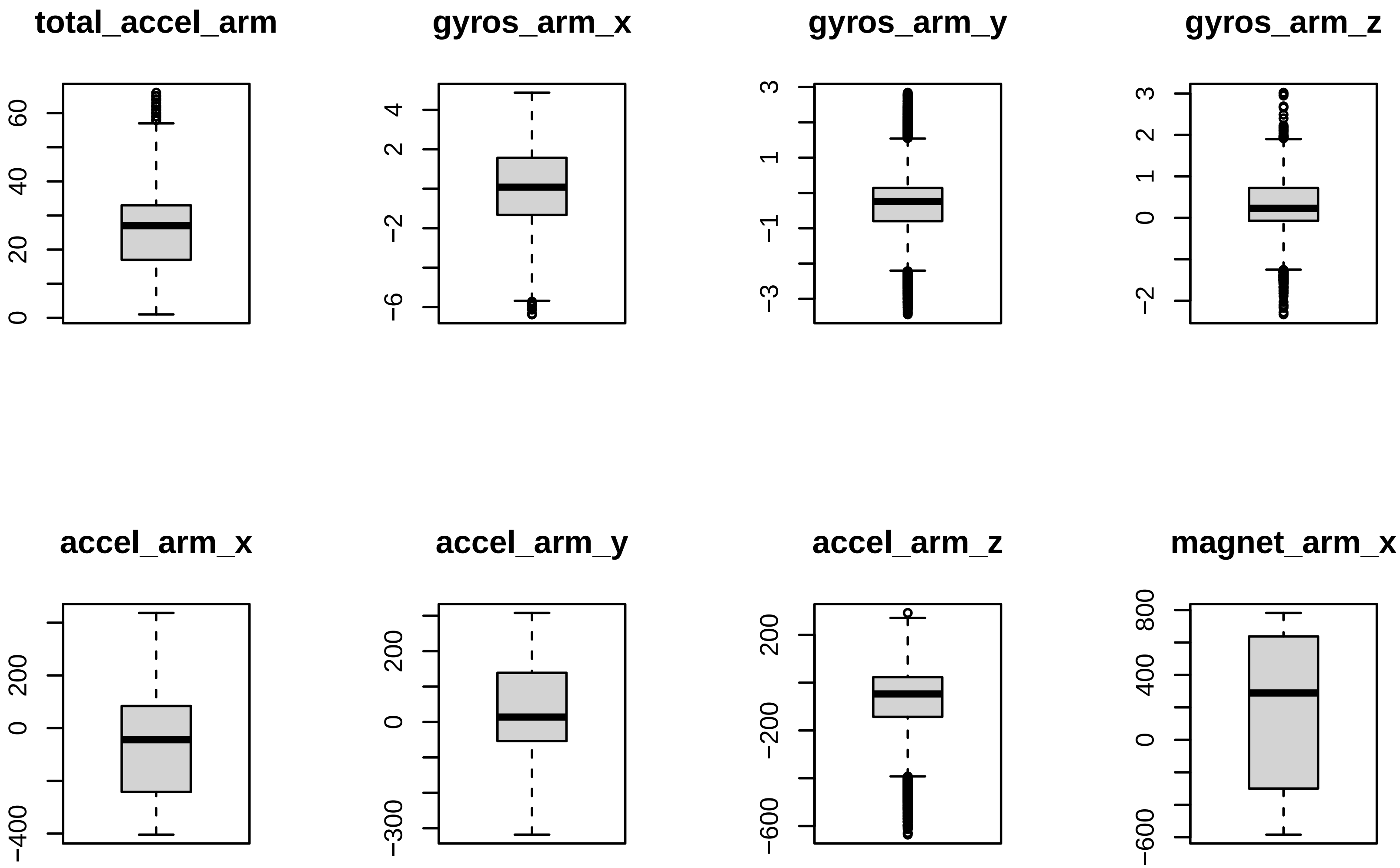
```
#Boxplot of each attribute
col_first_var <- grep("roll_belt", names(train))
col_last_var <- dim(train)[2]

nrows<- 2
ncols<- 4

num_plotsets <- ceiling((col_last_var - col_first_var)/(nrows*ncols))
num_plotsets
```

```
## [1] 7
```

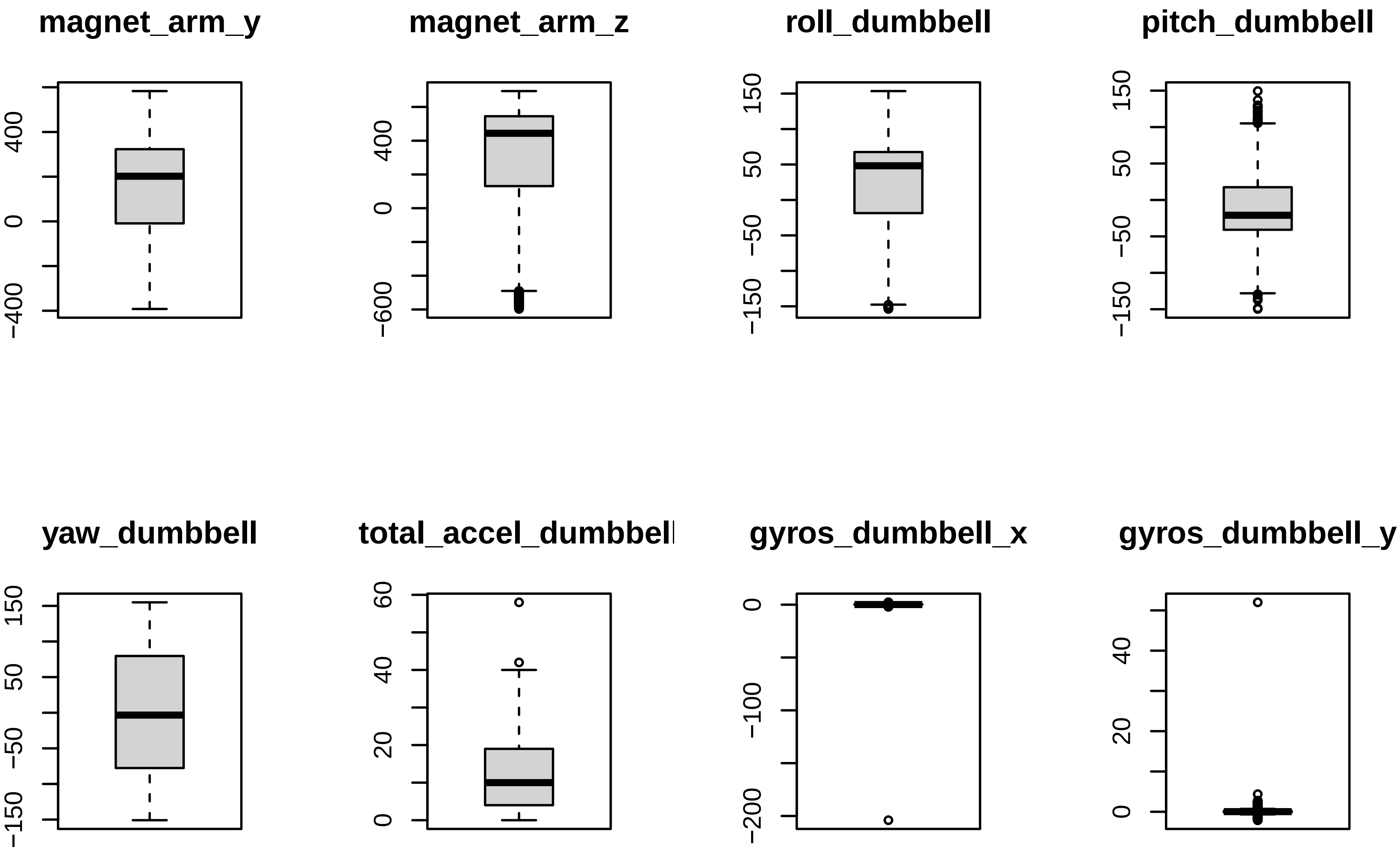
```
first <- col_first_var
last <- first + nrows*ncols - 1

for (j in 1:num_plotsets) {

        par(mfrow=c(nrows, ncols))

        for(i in first:last) {
            boxplot(train[,i], main=names(train)[i])
        }

        first <- last+1
        last <- first + nrows*ncols - 1
        if (last > dim(train)[2]) { last = dim(train)[2] }
}
```


roll_belt
0
50
100
pitch_belt
−60
−20
20
60
yaw_belt
−150
0
100
total_accel_belt
0
5
15
25
gyros_belt_x
−1.0
0.0
1.0
2.0
gyros_belt_y
−0.6
0.0
0.4
gyros_belt_z
−1.5
0.0
1.0
accel_belt_x
−100
0
50

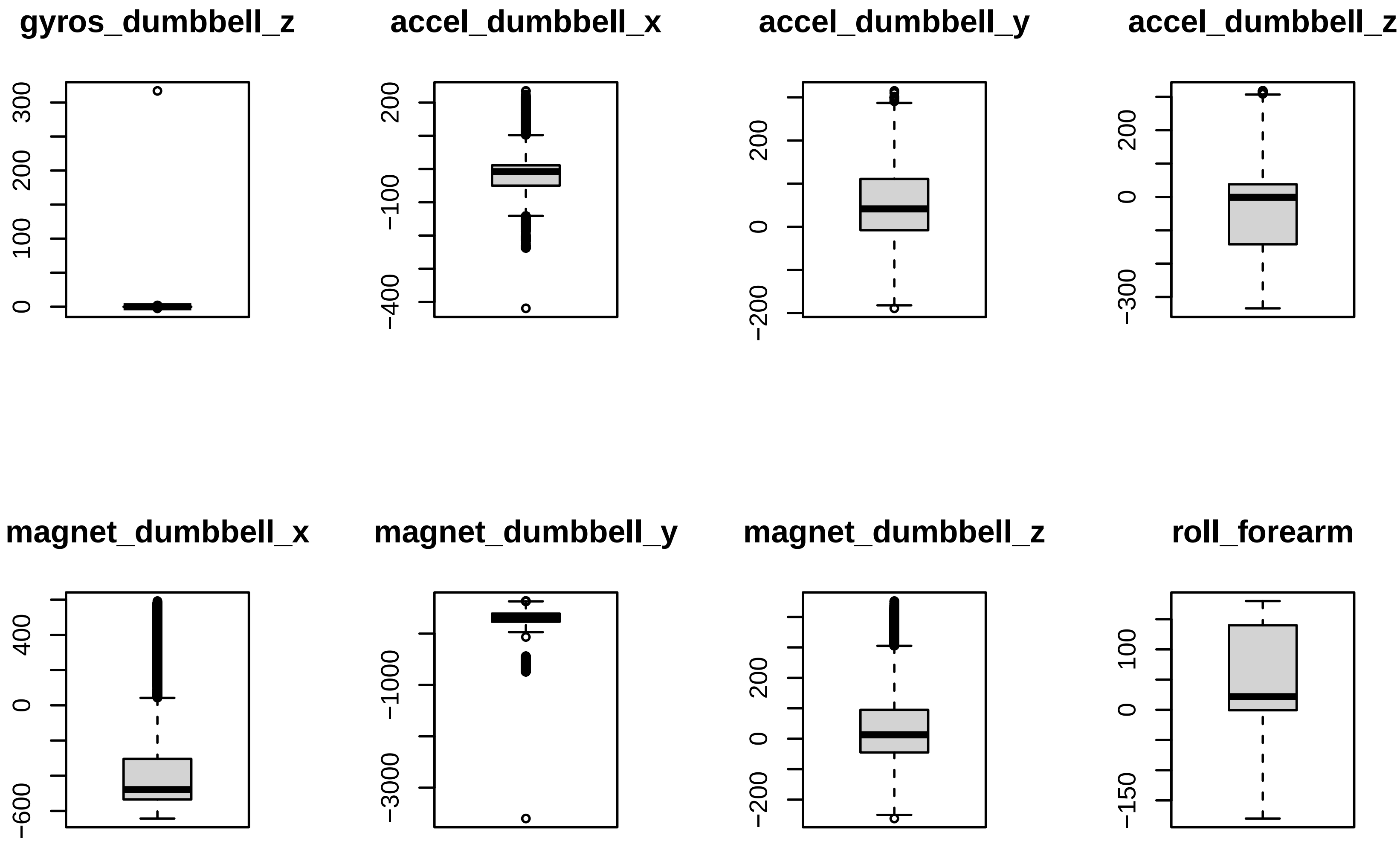

accel_belt_y
150
50
−50
accel_belt_z
100
0
−200
magnet_belt_x
400
200
0
magnet_belt_y
650
550
450
350
magnet_belt_z
200
−200
−600
roll_arm
100
0
−150
pitch_arm
50
0
−50
yaw_arm
100
0
−150



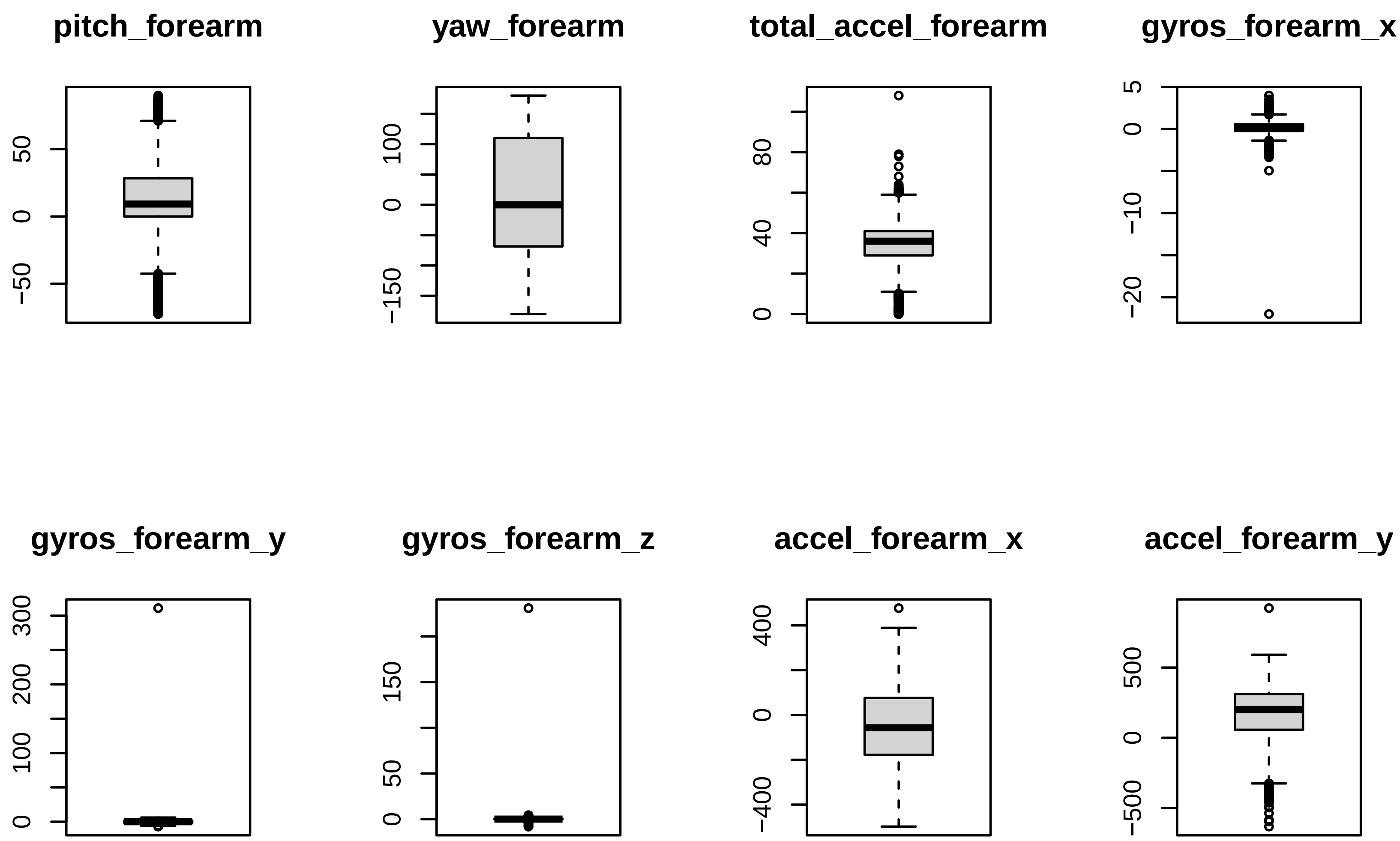
total_accel_arm
0
20
40
60
gyros_arm_x
−6
−2
2
4
gyros_arm_y
−3
−1
1
3
gyros_arm_z
−2
0
1
2
3
accel_arm_x
−400
0
200
accel_arm_y
−300
0
200
accel_arm_z
−600
−200
200
magnet_arm_x
−600
0
400
800

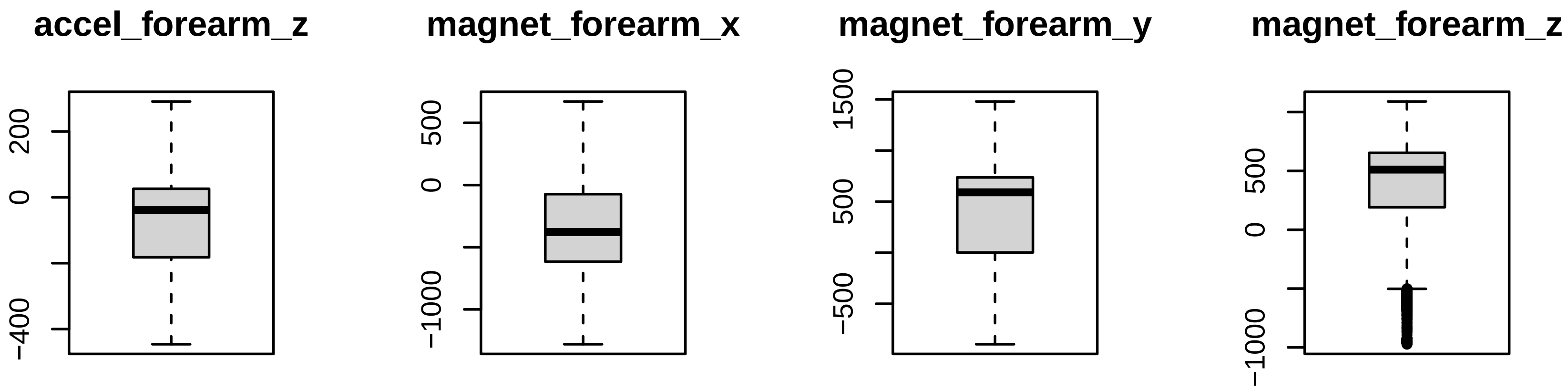


magnet_arm_y
−400
0
400
magnet_arm_z
−600
0
400
roll_dumbbell
−150
−50
50
150
pitch_dumbbell
−150
−50
50
150
yaw_dumbbell
−150
−50
50
150
total_accel_dumbbell
0
20
40
60
gyros_dumbbell_x
−200
−100
0
gyros_dumbbell_y
0
20
40

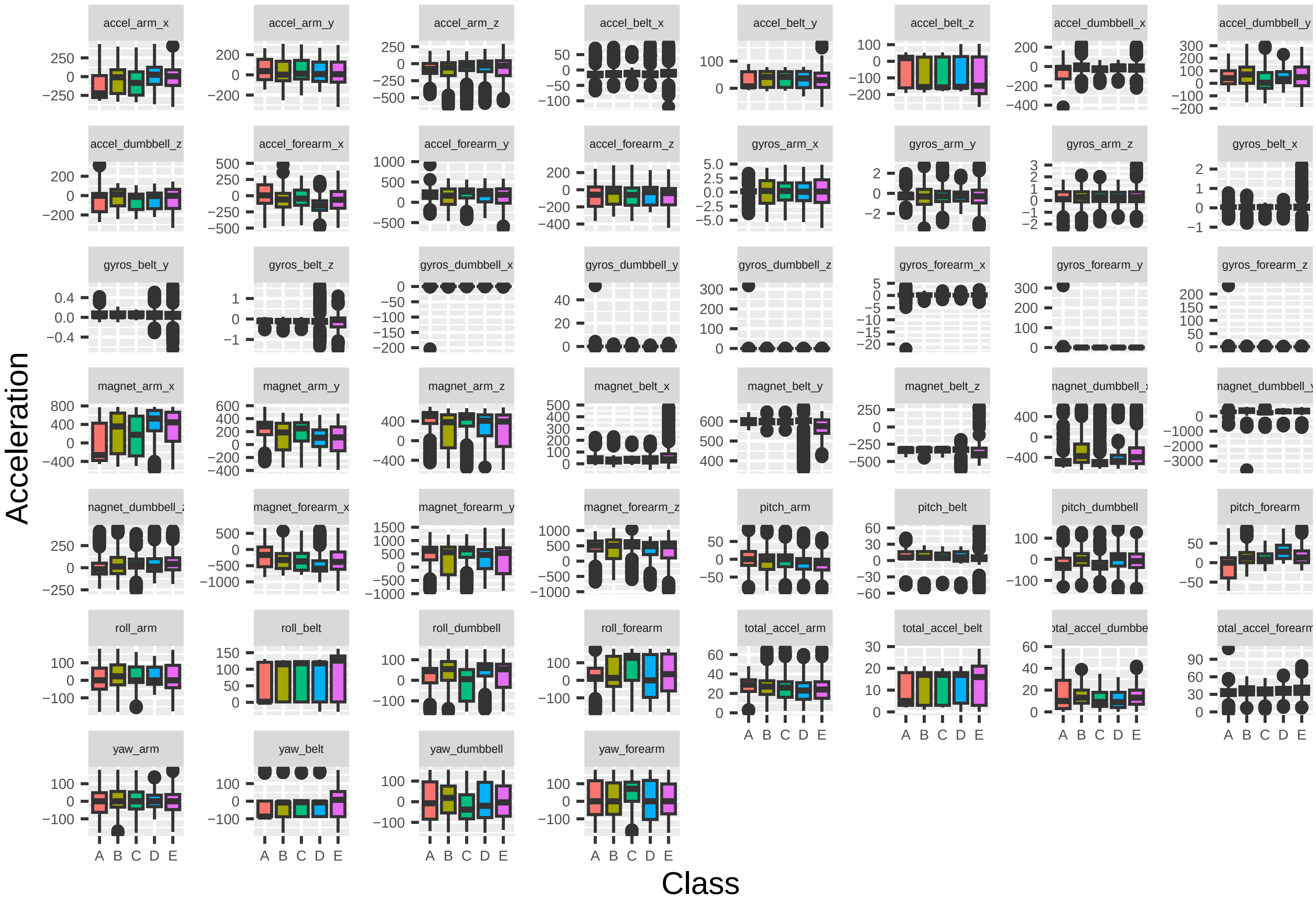


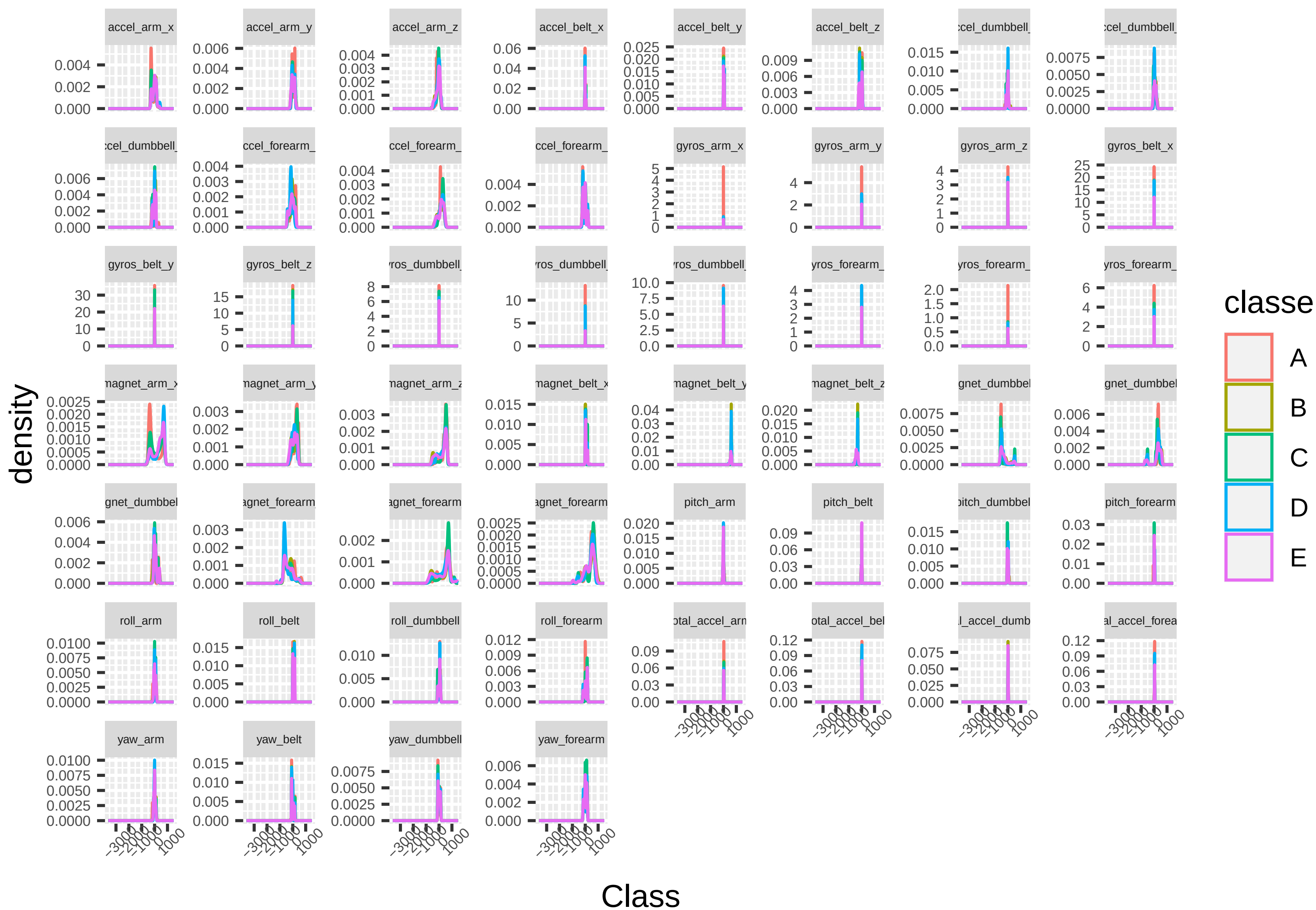
gyros_dumbbell_z
0
100
200
300
accel_dumbbell_x
−400
−100
200
accel_dumbbell_y
−200
0
200
accel_dumbbell_z
−300
0
200
magnet_dumbbell_x
−600
0
400
magnet_dumbbell_y
−3000
−1000
magnet_dumbbell_z
−200
0
200
roll_forearm
−150
0
100


pitch_forearm
50
0
−50
yaw_forearm
100
0
−150
total_accel_forearm
80
40
0
gyros_forearm_x
5
0
−10
−20
gyros_forearm_y
300
200
100
0
gyros_forearm_z
150
50
0
accel_forearm_x
400
0
−400
accel_forearm_y
500
0
−500

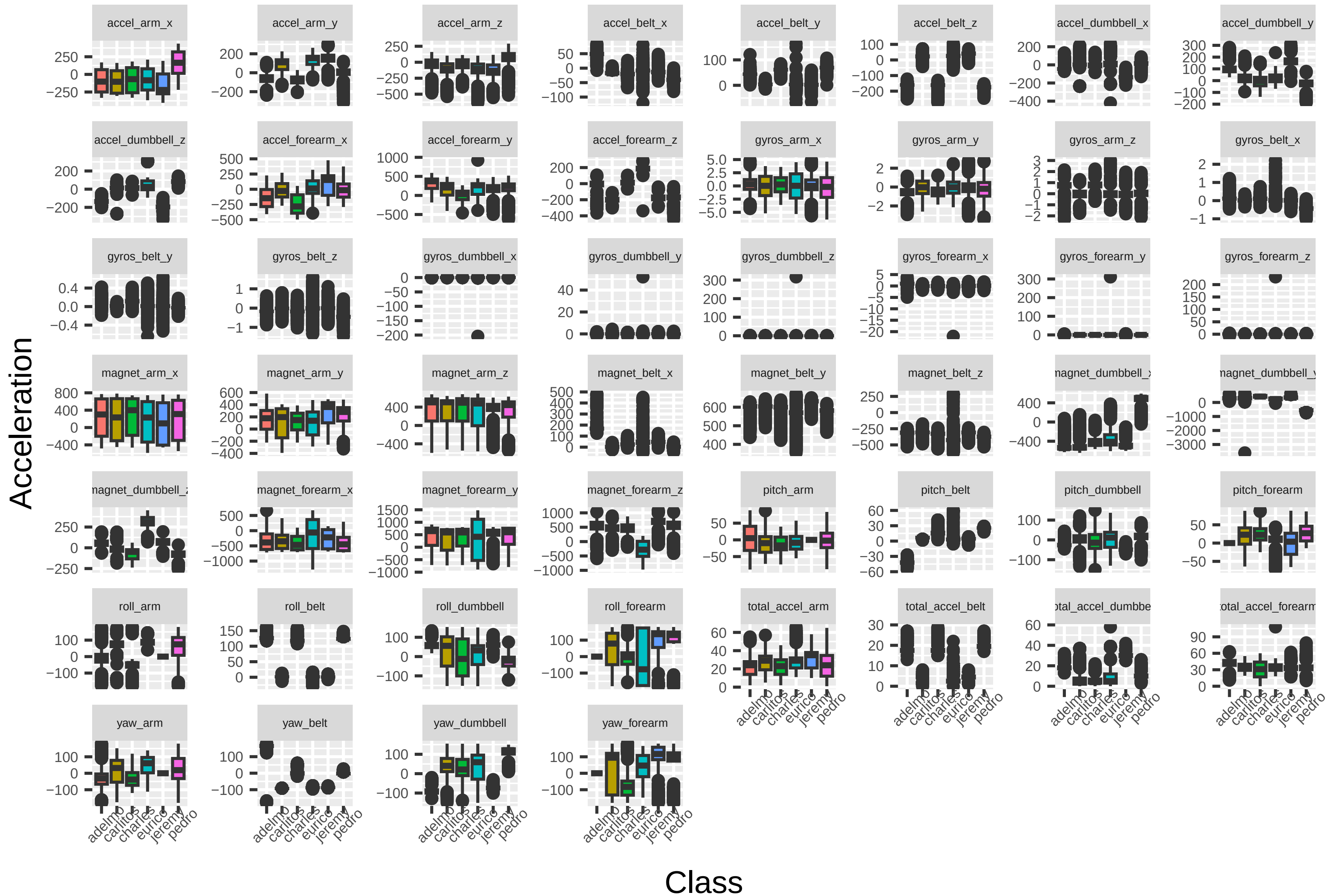

accel_forearm_z
200
0
−400
magnet_forearm_x
500
0
−1000
magnet_forearm_y
1500
500
−500
magnet_forearm_z
500
0
−1000

```
## Warning: package 'corrplot' was built under R version 4.2.2

## corrplot 0.92 loaded
```

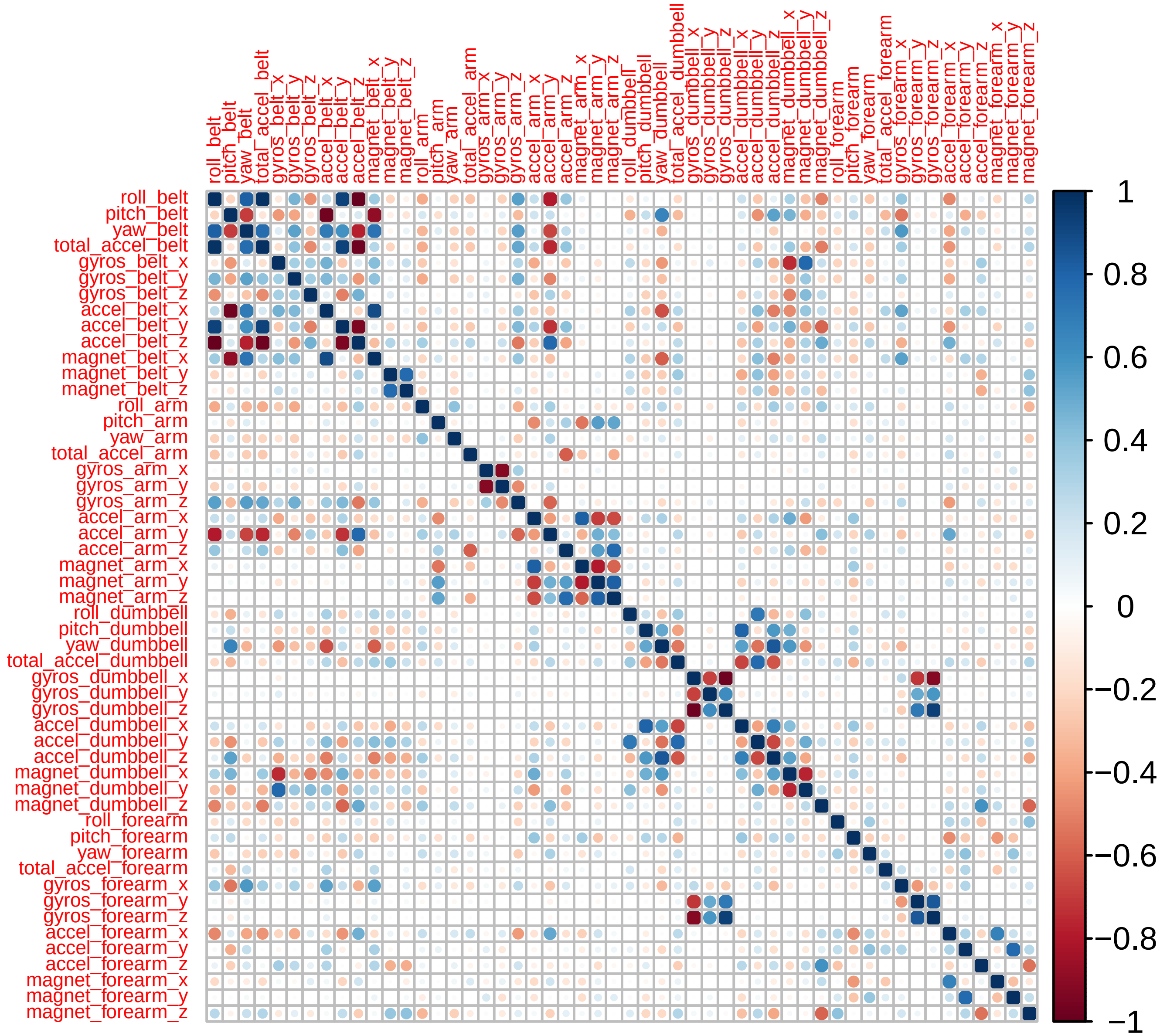

Acceleration
Class
A B C D E
accel_arm_x
accel_arm_y
accel_arm_z
accel_belt_x
accel_belt_y
accel_belt_z
accel_dumbbell_x
accel_dumbbell_y
accel_dumbbell_z
accel_forearm_x
accel_forearm_y
accel_forearm_z
gyros_arm_x
gyros_arm_y
gyros_arm_z
gyros_belt_x
gyros_belt_y
gyros_belt_z
gyros_dumbbell_x
gyros_dumbbell_y
gyros_dumbbell_z
gyros_forearm_x
gyros_forearm_y
gyros_forearm_z
magnet_arm_x
magnet_arm_y
magnet_arm_z
magnet_belt_x
magnet_belt_y
magnet_belt_z
magnet_forearm_x
magnet_forearm_y
magnet_forearm_z
pitch_arm
pitch_belt
pitch_dumbbell
pitch_forearm
roll_arm
roll_belt
roll_dumbbell
roll_forearm
total_accel_arm
total_accel_belt
yaw_arm
yaw_belt
yaw_dumbbell
yaw_forearm



classe
A
B
C
D
E
density
Class
accel_arm_x
accel_arm_y
accel_arm_z
accel_belt_x
accel_belt_y
accel_belt_z
gyros_arm_x
gyros_arm_y
gyros_arm_z
gyros_belt_x
gyros_belt_y
gyros_belt_z
pitch_arm
pitch_belt
pitch_forearm
roll_arm
roll_belt
roll_dumbbell
roll_forearm
yaw_arm
yaw_belt
yaw_dumbbell
yaw_forearm



Acceleration
Class
accel_arm_x
accel_arm_y
accel_arm_z
accel_belt_x
accel_belt_y
accel_belt_z
accel_dumbbell_x
accel_dumbbell_y
accel_dumbbell_z
accel_forearm_x
accel_forearm_y
accel_forearm_z
gyros_arm_x
gyros_arm_y
gyros_arm_z
gyros_belt_x
gyros_belt_y
gyros_belt_z
gyros_dumbbell_x
gyros_dumbbell_y
gyros_dumbbell_z
gyros_forearm_x
gyros_forearm_y
gyros_forearm_z
magnet_arm_x
magnet_arm_y
magnet_arm_z
magnet_belt_x
magnet_belt_y
magnet_belt_z
pitch_arm
pitch_belt
pitch_dumbbell
pitch_forearm
roll_arm
roll_belt
roll_dumbbell
roll_forearm
total_accel_arm
total_accel_belt
yaw_arm
yaw_belt
yaw_dumbbell
yaw_forearm
adelmo
carlitos
charles
eurico
jeremy
pedro



Looking through various plots we notice some variables have significant skewness and many variables have lots of outliers. We'll investigate what difference it makes to remove outliers and impute values under various strategies. There are some variables with high correlations but we anticipate this won't be a problem for most of our classification algorithms.

Model Training - Fit various models

A validation set was split off from the training data.

```
library(caret)
set.seed(100)
part <- createDataPartition(train$classe, p = 3/4, list = FALSE)
train <- train[part, ]
validate <- train[-part, ]
```

Remove outliers from training set. Using KNN imputation under the caret train() function preprocessing commands delivered terrible results. So instead we imputed mean values after grouping by user_name and classe.

We specify quantiles to control the definition of outliers.

```
out_ind <- 0
missing <- 0
train_no_outliers <- train

for (i in col_first_var:col_last_var) {
        below <- quantile(train_no_outliers[, i], 0.02) #remove bottom 2%
        above <- quantile(train_no_outliers[, i], 0.98) #remove top 2%

        out_ind <- which(train_no_outliers[, i] < below | train_no_outliers[, i] > above)

        train_no_outliers[out_ind, i] = NA
        missing[i-1] = sum(is.na(train_no_outliers[, i])) #count the amount of NA's for that variable

        #replace values with averages by group (courtesy of)[https://stackoverflow.com/questions/55345593/impute-missing-data-with-mean-by
        train_no_outliers[i] <- ave(train_no_outliers[, i], train_no_outliers$user_name,
                                    train_no_outliers$classe,
                                    FUN = function(x) ifelse(is.na(x), mean(x,na.rm=TRUE), x))
}

print("Amount of values that were identified as outliers, removed, and imputed - per variable:")
```

```
## [1] "Amount of values that were identified as outliers, removed, and imputed - per variable:"
```

```
missing
```

```
##  [1]   0  NA  NA 581 549 445 374 548 560 574 534 539 569 558 540 584 559 589 573
## [20] 534 578 579 561 540 578 580 560 570 578 569 590 590 310 580 576 565 589 576
## [39] 558 571 585 584 431 587 570 495 580 579 581 576 585 534 588 575 585
```

```
print("Check if any NA's are left after imputation command:")
```

```
## [1] "Check if any NA's are left after imputation command:"
```

```
ouliers_remaining <- sapply(train_no_outliers, function(x) sum(is.na(x)))
ouliers_remaining[ouliers_remaining>0]
```

```
## named integer(0)
```

```
# redo box and whisker plots for each attribute by classe
#fp_box_byClasse <- featurePlot(x= train_no_outliers[,5:56], y=train_no_outliers$classe, plot="box", scales=scales)
#fp_box_byClasse
```

We trained a random forest on the remaining training data. It was very slow. Some tuning of the most important model parameters (mtry and ntree) was performed to reduce the computational time (on a slow machine) from 2.5 hr to 20.4 s. The latter included a repeated cross validation of 3 folds and 3 repeats.

```
library(caret)
tcont <- trainControl(method = "repeatedcv", number=3, repeats=3)

#CART model
set.seed(100)
start.time1 <- Sys.time()
grid <- expand.grid(.cp=c(0.01,0.05,0.1))
model_cart <- train(classe ~ ., method="rpart", data = train, tuneGrid=grid, trControl=tcont)
end.time1 <- Sys.time()
time.taken1 <- end.time1 - start.time1

time.taken1
```

```
## Time difference of 7.509096 secs
```

```r
print(model_cart)
```

```
## CART
##
## 14718 samples
##    55 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9812, 9811, 9813, 9811, 9812, 9813, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.01  0.9997962  0.9997422
##   0.05  0.9997962  0.9997422
##   0.10  0.9997962  0.9997422
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.1.
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

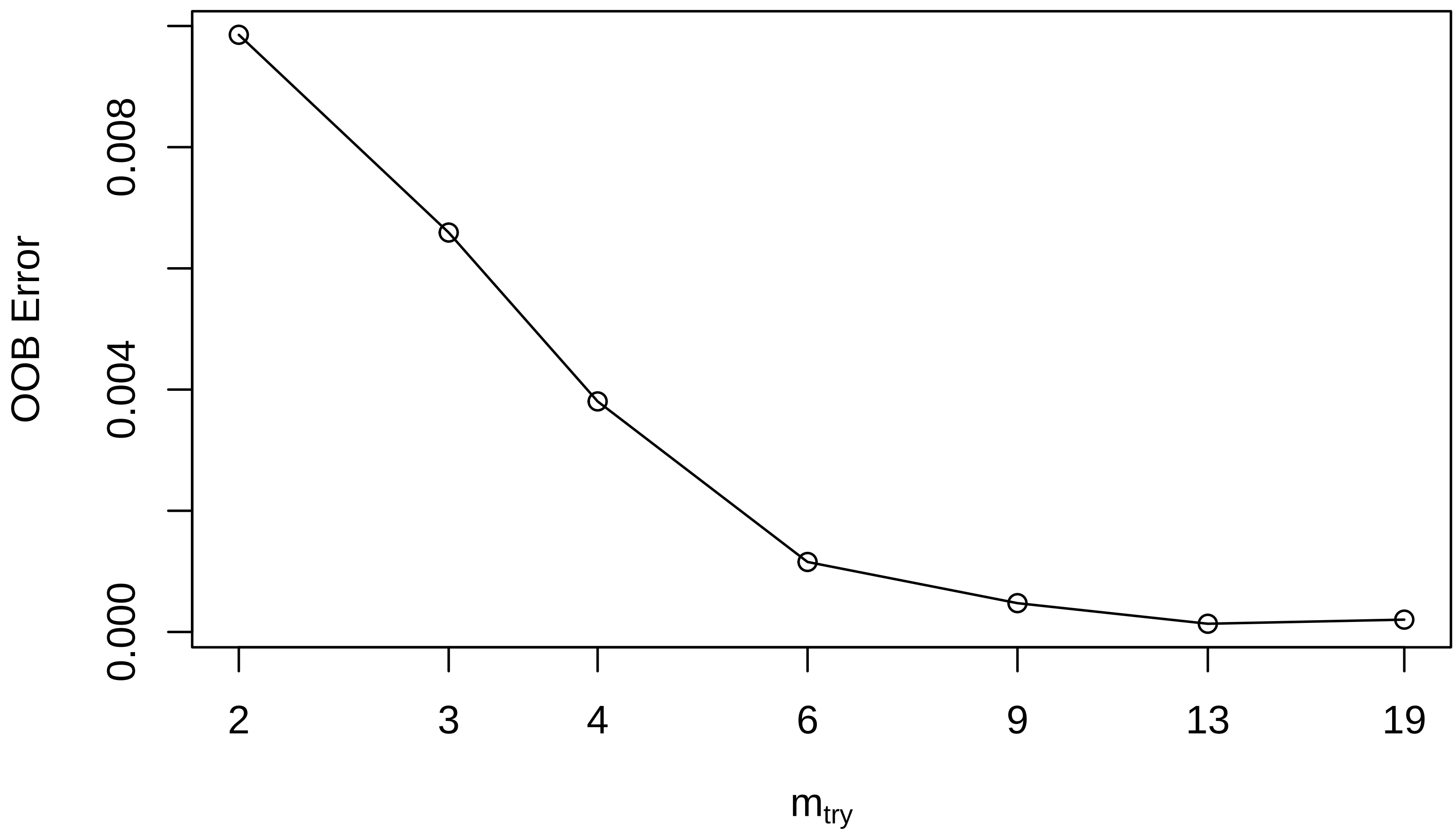
```
##
## Attaching package: 'randomForest'
```

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

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
#use tuneRF() function from the randomForest package to find the optimal value for mtry
tune_mtry <- tuneRF(subset(train, select = -classe), train$classe, mtryStart = 2, stepFactor=1.5, ntreeTry = 20, improve = 0.01)
```

```
## mtry = 2  OOB error = 0.99%
## Searching left ...
## Searching right ...
## mtry = 3     OOB error = 0.66%
## 0.3311708 0.01
## mtry = 4     OOB error = 0.38%
## 0.4227196 0.01
## mtry = 6     OOB error = 0.12%
## 0.6964079 0.01
## mtry = 9     OOB error = 0.05%
## 0.5882633 0.01
## mtry = 13    OOB error = 0.01%
## 0.7142663 0.01
## mtry = 19    OOB error = 0.02%
## -0.4998981 0.01
```


OOB Error
0.000
0.004
0.008
2
3
4
6
9
13
19
m_{try}

```r
print(tune_mtry)
```

```
##        mtry     OOBError
## 2.OOB     2 0.0098545603
## 3.OOB     3 0.0065910172
## 4.OOB     4 0.0038048648
## 6.OOB     6 0.0011551267
## 9.OOB     9 0.0004756081
## 13.OOB   13 0.0001358973
## 19.OOB   19 0.0002038320
```

```r
tunegrid <- expand.grid(.mtry=13)

#manual iterations were performed to find the lowest value for ntree without having a noticeable effect on the resulting accuracy.
set.seed(100)
start.time2 <- Sys.time()

rf_tuned <- train(classe ~ ., method = "rf", data = train, ntree = 20, tuneGrid = tunegrid, preProcess = c("center", "scale"),trControl =

end.time2 <- Sys.time()
time.taken2 <- end.time2 - start.time2

time.taken2
```

```
## Time difference of 17.76809 secs
```

```r
print(rf_tuned)
```

```
## Random Forest
##
## 14718 samples
##    55 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9812, 9811, 9813, 9811, 9812, 9813, ...
```

```
## Resampling results:
##
##   Accuracy   Kappa
##   0.9999774  0.9999714
##
## Tuning parameter 'mtry' was held constant at a value of 13
```

```r
#now see what difference it makes to use the dataset with outliers removed
set.seed(100)
start.time3 <- Sys.time()

rf_tuned_no_outliers <- train(classe ~ ., method = "rf", data = train_no_outliers, ntree = 20, tuneGrid = tunegrid, preProcess = c("center

end.time3 <- Sys.time()
time.taken3 <- end.time3 - start.time3

time.taken3
```

```
## Time difference of 16.90107 secs
```

```r
print(rf_tuned_no_outliers)
```

```
## Random Forest
##
## 14718 samples
##    55 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9812, 9811, 9813, 9811, 9812, 9813, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9999774  0.9999714
##
## Tuning parameter 'mtry' was held constant at a value of 13
```

```r
# Linear Discriminant Analysis
set.seed(100)
start.time4 <- Sys.time()

model_lda <- train(classe ~ ., method = "lda", data = train, metric="Accuracy", trControl=tcont)

end.time4 <- Sys.time()
time.taken4 <- end.time4 - start.time4

time.taken4
```

```
## Time difference of 5.717247 secs
```

```r
print(model_lda)
```

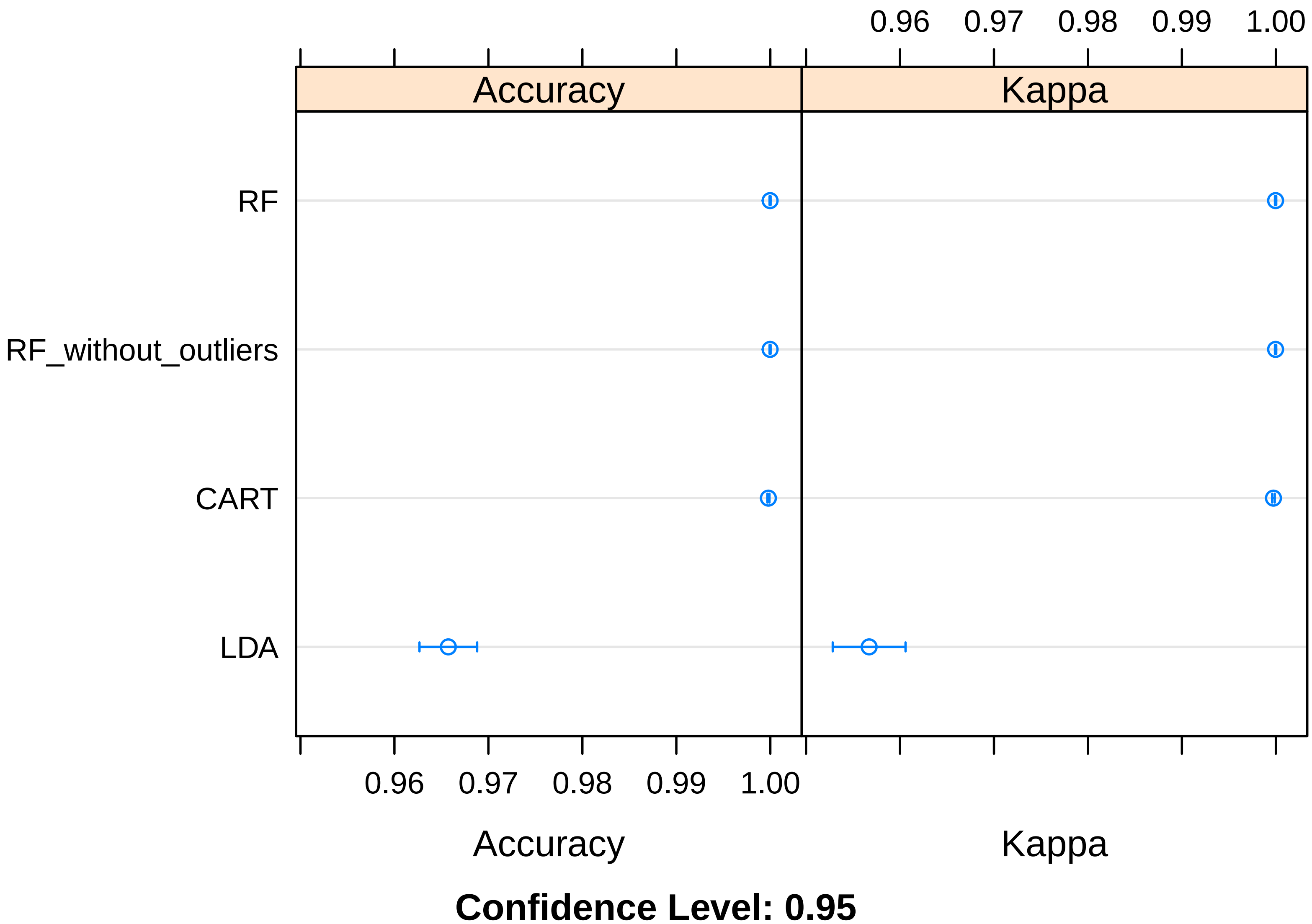
```
## Linear Discriminant Analysis
##
## 14718 samples
##    55 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9812, 9811, 9813, 9811, 9812, 9813, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9657334  0.9567175
```

```r
#show model comparison based on resampling accuracy results:
results <- resamples(list(CART = model_cart, RF=rf_tuned, RF_without_outliers=rf_tuned_no_outliers, LDA = model_lda))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
```

```
## Models: CART, RF, RF_without_outliers, LDA
## Number of resamples: 9
##
## Accuracy
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## CART                0.9995923 0.9997961 0.9997962 0.9997962 0.9997962 1.0000000
## RF                  0.9997962 1.0000000 1.0000000 0.9999774 1.0000000 1.0000000
## RF_without_outliers 0.9997962 1.0000000 1.0000000 0.9999774 1.0000000 1.0000000
## LDA                 0.9600408 0.9626987 0.9665647 0.9657334 0.9673935 0.9714635
##                     NA's
## CART                   0
## RF                     0
## RF_without_outliers    0
## LDA                    0
##
## Kappa
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## CART                0.9994843 0.9997421 0.9997422 0.9997422 0.9997422 1.0000000
## RF                  0.9997422 1.0000000 1.0000000 0.9999714 1.0000000 1.0000000
## RF_without_outliers 0.9997422 1.0000000 1.0000000 0.9999714 1.0000000 1.0000000
## LDA                 0.9495376 0.9528818 0.9577749 0.9567175 0.9588183 0.9639456
##                     NA's
## CART                   0
## RF                     0
## RF_without_outliers    0
## LDA                    0
```

```
dotplot(results)
```



At this stage the CART and RF models look slightly better than the LDA.

Validation

The models were validated by predicting on the validation set and comparing to the reference values in the classe variable.

```
set.seed(100)
print("Results on test set prediction for CART: single decision tree")
```

```
## [1] "Results on test set prediction for CART: single decision tree"
```

```
predClasse1 <- predict(object = model_cart, newdata = subset(validate, select = -c(classe)))
result1 <- confusionMatrix(validate$classe, predClasse1)
result1$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1053    0    0    0    0
##          B    0  713    0    0    0
##          C    0    0  638    0    0
##          D    0    0    0  600    0
##          E    0    0    0    0  675
```

```
set.seed(100)
print("Results on test set prediction for rf_tuned: tuned random forest with outliers included")
```

```
## [1] "Results on test set prediction for rf_tuned: tuned random forest with outliers included"
```

```
predClasse2 <- predict(object = rf_tuned, newdata = subset(validate, select = -c(classe)))
result2 <- confusionMatrix(validate$classe, predClasse2)
result2$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1053    0    0    0    0
##          B    0  713    0    0    0
##          C    0    0  638    0    0
##          D    0    0    0  600    0
##          E    0    0    0    0  675
```

```r
set.seed(100)
print("Results on test set prediction for rf_tuned_no_outliers:
      tuned random forest with outliers excluded and replaced by grouped means")
```

```
## [1] "Results on test set prediction for rf_tuned_no_outliers: \n      tuned random forest with outliers excluded and replaced by groupe
```

```r
predClasse3 <- predict(object = rf_tuned_no_outliers, newdata = subset(validate, select = -c(classe)))
result3 <- confusionMatrix(validate$classe, predClasse3)
result3$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1053    0    0    0    0
##          B    0  713    0    0    0
##          C    0    0  638    0    0
##          D    0    0    0  600    0
##          E    0    0    0    0  675
```

```r
set.seed(100)
print("Results on test set prediction for model_lda: linear discriminant analyis model")
```

```
## [1] "Results on test set prediction for model_lda: linear discriminant analyis model"
```

```r
predClasse4 <- predict(object = model_lda, newdata = subset(validate, select = -c(classe)))
result4 <- confusionMatrix(validate$classe, predClasse4)
result4$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1019   34    0    0    0
##          B    1  676   36    0    0
##          C    0    0  605   33    0
##          D    0    0    2  593    5
##          E    0    0    0   21  654
```

```r
compare_validation_result <- data.frame(Model = c("CART", "RF", "RF_less_Outliers", "LDA"),
                                        Accuracy = c(result1$overall[1], result2$overall[1],
                                                     result3$overall[1], result4$overall[1]))
print("Out of sample accuracies compared:")
```

```
## [1] "Out of sample accuracies compared:"
```

```r
compare_validation_result
```

```
##              Model  Accuracy
## 1             CART 1.0000000
## 2               RF 1.0000000
## 3 RF_less_Outliers 1.0000000
## 4              LDA 0.9641207
```

The simple CART model did surprisingly wel, considering it didn't require any time to tune for computational time like the random forest. The out of sample error for the CART and both rf models were excellent, with a 100% accuracy. The lda model performed well with an accuracy of 96.4% and it was just as easy as the CART model. The accuracy dropped a further 3% when using the data where outliers were removed and imputed.

Variable Importance

For interest, we have a look at which variables turned out to be important in our random forest and CART models.

```r
i_scores1 <- varImp(rf_tuned, scale = TRUE)$importance %>%
        as.data.frame() %>%
        arrange(desc(Overall))
print("Variable Imporantance for RF")
```

```
## [1] "Variable Imporantance for RF"
```

```r
i_scores1
```

```
##                           Overall
## X                    1.000000e+02
## time                 1.660604e+01
```

```
## roll_belt            9.716082e+00
## pitch_forearm        7.029036e+00
## magnet_dumbbell_y    4.724534e+00
## magnet_belt_y        4.208407e+00
## magnet_dumbbell_z    4.175090e+00
## pitch_belt           4.147591e+00
## yaw_belt             4.017768e+00
## accel_belt_z         3.959863e+00
## roll_forearm         2.364047e+00
## roll_dumbbell        2.301107e+00
## accel_dumbbell_y     1.937848e+00
## magnet_dumbbell_x    1.836417e+00
## gyros_belt_z         1.784848e+00
## magnet_belt_z        1.751046e+00
## total_accel_belt     1.741161e+00
## accel_forearm_x      1.600793e+00
## accel_arm_x          1.581666e+00
## roll_arm             1.369002e+00
## accel_dumbbell_x     1.311150e+00
## total_accel_dumbbell 1.259796e+00
## yaw_arm              1.148757e+00
## magnet_belt_x        1.101259e+00
## magnet_forearm_z     9.977080e-01
## magnet_forearm_x     9.841842e-01
## accel_dumbbell_z     9.766517e-01
## magnet_forearm_y     8.168276e-01
## pitch_dumbbell       6.886734e-01
## accel_forearm_z      6.620836e-01
## magnet_arm_z         6.433059e-01
## magnet_arm_y         6.397048e-01
## yaw_dumbbell         5.613608e-01
## accel_arm_z          5.183410e-01
## pitch_arm            3.715298e-01
## gyros_dumbbell_y     3.208767e-01
## yaw_forearm          3.178295e-01
## gyros_arm_y          3.166019e-01
## user_nameeurico      3.083847e-01
## total_accel_arm      2.947667e-01
## accel_arm_y          2.781035e-01
```

```
## accel_belt_y         2.521526e-01
## gyros_belt_y         2.444723e-01
## magnet_arm_x         2.206297e-01
## gyros_belt_x         2.121614e-01
## gyros_arm_x          1.918124e-01
## accel_belt_x         1.349826e-01
## gyros_forearm_x      1.294423e-01
## gyros_forearm_y      1.180082e-01
## accel_forearm_y      1.000764e-01
## gyros_forearm_z      9.199403e-02
## gyros_dumbbell_x     8.500987e-02
## total_accel_forearm  8.396068e-02
## gyros_dumbbell_z     7.603695e-02
## user_namepedro       5.010247e-02
## gyros_arm_z          3.761283e-02
## user_namecharles     2.151594e-02
## user_namejeremy      6.871519e-03
## user_namecarlitos    0.000000e+00
```

```
i_scores2 <- varImp(model_cart)$importance %>%
        as.data.frame() %>%
        arrange(desc(Overall))
print("Variable Imporantance for CART model")
```

```
## [1] "Variable Imporantance for CART model"
```

```
i_scores2
```

```
##                         Overall
## X                    100.000000
## roll_belt             27.119461
## accel_belt_z          16.575285
## magnet_belt_y         14.534881
## total_accel_belt       9.219841
## pitch_forearm          6.826777
## roll_dumbbell          3.753260
## magnet_dumbbell_y      3.106431
## accel_forearm_x        2.832714
```

```
## pitch_dumbbell         2.736820
## user_namecarlitos      0.000000
## user_namecharles       0.000000
## user_nameeurico        0.000000
## user_namejeremy        0.000000
## user_namepedro         0.000000
## time                   0.000000
## pitch_belt             0.000000
## yaw_belt               0.000000
## gyros_belt_x           0.000000
## gyros_belt_y           0.000000
## gyros_belt_z           0.000000
## accel_belt_x           0.000000
## accel_belt_y           0.000000
## magnet_belt_x          0.000000
## magnet_belt_z          0.000000
## roll_arm               0.000000
## pitch_arm              0.000000
## yaw_arm                0.000000
## total_accel_arm        0.000000
## gyros_arm_x            0.000000
## gyros_arm_y            0.000000
## gyros_arm_z            0.000000
## accel_arm_x            0.000000
## accel_arm_y            0.000000
## accel_arm_z            0.000000
## magnet_arm_x           0.000000
## magnet_arm_y           0.000000
## magnet_arm_z           0.000000
## yaw_dumbbell           0.000000
## total_accel_dumbbell   0.000000
## gyros_dumbbell_x       0.000000
## gyros_dumbbell_y       0.000000
## gyros_dumbbell_z       0.000000
## accel_dumbbell_x       0.000000
## accel_dumbbell_y       0.000000
## accel_dumbbell_z       0.000000
## magnet_dumbbell_x      0.000000
## magnet_dumbbell_z      0.000000
```

```
## roll_forearm           0.000000
## yaw_forearm            0.000000
## total_accel_forearm    0.000000
## gyros_forearm_x        0.000000
## gyros_forearm_y        0.000000
## gyros_forearm_z        0.000000
## accel_forearm_y        0.000000
## accel_forearm_z        0.000000
## magnet_forearm_x       0.000000
## magnet_forearm_y       0.000000
## magnet_forearm_z       0.000000
```

And which ones were not…

```r
print("variables not used in the RF model")
```

```
## [1] "variables not used in the RF model"
```

```r
rownames(i_scores1)[i_scores1$Overall == 0]
```

```
## [1] "user_namecarlitos"
```

```r
print("variables not used in the CART model")
```

```
## [1] "variables not used in the CART model"
```

```r
rownames(i_scores2)[i_scores2$Overall == 0]
```

```
##  [1] "user_namecarlitos"    "user_namecharles"     "user_nameeurico"
##  [4] "user_namejeremy"      "user_namepedro"       "time"
##  [7] "pitch_belt"           "yaw_belt"             "gyros_belt_x"
## [10] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [13] "accel_belt_y"         "magnet_belt_x"        "magnet_belt_z"
## [16] "roll_arm"             "pitch_arm"            "yaw_arm"
## [19] "total_accel_arm"      "gyros_arm_x"          "gyros_arm_y"
## [22] "gyros_arm_z"          "accel_arm_x"          "accel_arm_y"
```

```
## [25] "accel_arm_z"          "magnet_arm_x"         "magnet_arm_y"
## [28] "magnet_arm_z"         "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_z"    "roll_forearm"
## [40] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [43] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_y"
## [46] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [49] "magnet_forearm_z"
```

We note that both models shown have a top ranking of the X (index) variable. Interesting that the CART model didn't use the user_name variables or the time variable. And only 10 of the remaining ones. The trends must have been strong across participants for the model to pick the activity without really caring about who did it.

The RF was much more complicated - using almost all of the variables to some extent.

Post-modelling plotting

For interest, lets look at a graphic representation of the CART model.

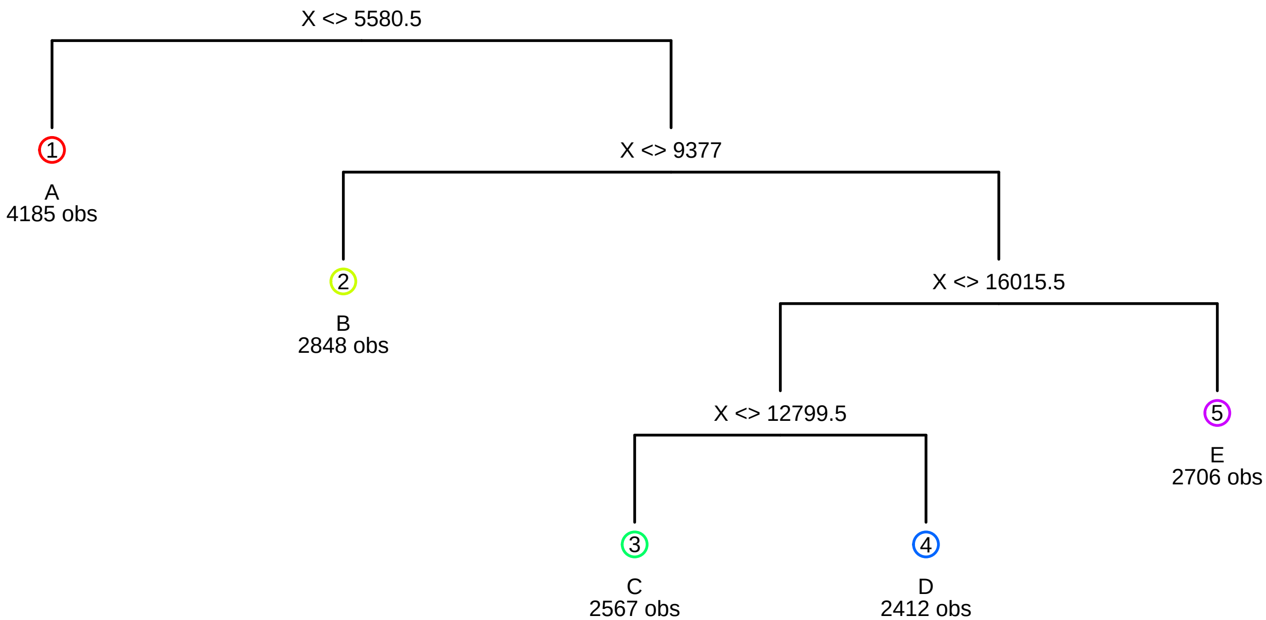
```
library(maptree)
```

```
## Warning: package 'maptree' was built under R version 4.2.2
```

```
## Loading required package: cluster
```

```
## Loading required package: rpart
```

```
draw.tree(model_cart$finalModel,cex=0.5)
```



From the plot above it is clear that the model figured out that the classe variable was arranged in chunks along the X variable. ALthough this model will work for the purpose of passing this assignment, if we give it new data that has similar data but arranged differently by idex the model will struggle. We should retrain the models without giving them the index or the time. This would be more useful.

Let's see if we can still attain good accuracy:

```
#New CART model
set.seed(100)

grid <- expand.grid(.cp=c(0.01,0.05,0.1))
model_cart_mod <- train(classe ~ ., method="rpart", data = subset(train, select = -c(X, time, user_name)),
                        tuneGrid=grid, trControl=tcont)

print("Results on validation set prediction for revised CART: single decision tree")
```

```
## [1] "Results on validation set prediction for revised CART: single decision tree"
```

```
set.seed(100)
predClasse_CART_mod <- predict(object = model_cart_mod, newdata = subset(validate, select = -c(X, time, user_name, classe)))
result_CART_mod <- confusionMatrix(validate$classe, predClasse_CART_mod)
result_CART_mod$table
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 963  29  27  11  23
##          B 136 420  76  53  28
##          C   6  48 544  28  12
##          D  57  18  89 404  32
##          E  20  42  90  36 487
```

```
result_CART_mod$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.659690e-01   7.026887e-01   7.519436e-01   7.795706e-01   3.212830e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   3.055850e-44
```

```r
#New LDA model
set.seed(100)

model_lda_mod <- train(classe ~ ., method = "lda", data = subset(train, select = -c(X, time, user_name)),
                       metric="Accuracy", trControl=tcont)

print("Results on validation set prediction for revised LDA")
```

```
## [1] "Results on validation set prediction for revised LDA"
```

```r
set.seed(100)
predClasse_lda_mod <- predict(object = model_lda_mod, newdata = subset(validate, select = -c(X, time, user_name, classe)))
result_lda_mod <- confusionMatrix(validate$classe, predClasse_lda_mod)
result_lda_mod$table
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 859  34  84  73   3
##          B 111 458  86  22  36
##          C  64  65 411  79  19
##          D  22  21  86 443  28
##          E  20 127  61  58 409
```

```r
result_lda_mod$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.012775e-01   6.219790e-01   6.861961e-01   7.160385e-01   2.924708e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   2.649821e-31
```

```r
#New RF model


tunegrid <- expand.grid(.mtry=9)
model_rf_mod <- train(classe ~ ., method = "rf", data = subset(train, select = -c(X, time, user_name)),
                      ntree = 20, tuneGrid = tunegrid, preProcess = c("center", "scale"),trControl = tcont )

print(model_rf_mod)
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9812, 9813, 9811, 9813, 9811, 9812, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9873851  0.9840413
##
## Tuning parameter 'mtry' was held constant at a value of 9
```

```r
print("Results on validation set prediction for revised RF")
```

```
## [1] "Results on validation set prediction for revised RF"
```

```r
set.seed(100)
predClasse_rf_mod <- predict(object = model_rf_mod, newdata = subset(validate, select = -c(X, time, user_name, classe)))
result_rf_mod <- confusionMatrix(validate$classe, predClasse_rf_mod)
result_rf_mod$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1053    0    0    0    0
##          B    0  713    0    0    0
##          C    0    0  638    0    0
##          D    0    0    0  600    0
##          E    0    0    0    0  675
```

```r
result_rf_mod$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      1.0000000      1.0000000      0.9989978      1.0000000      0.2862191
```

```
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
compare_validation_result_mod <- data.frame(Model = c("CART", "LDA", "RF"), Accuracy = as.numeric(c(result_CART_mod$overall[1], result_lda
print("Out of sample accuracies compared: ")
```

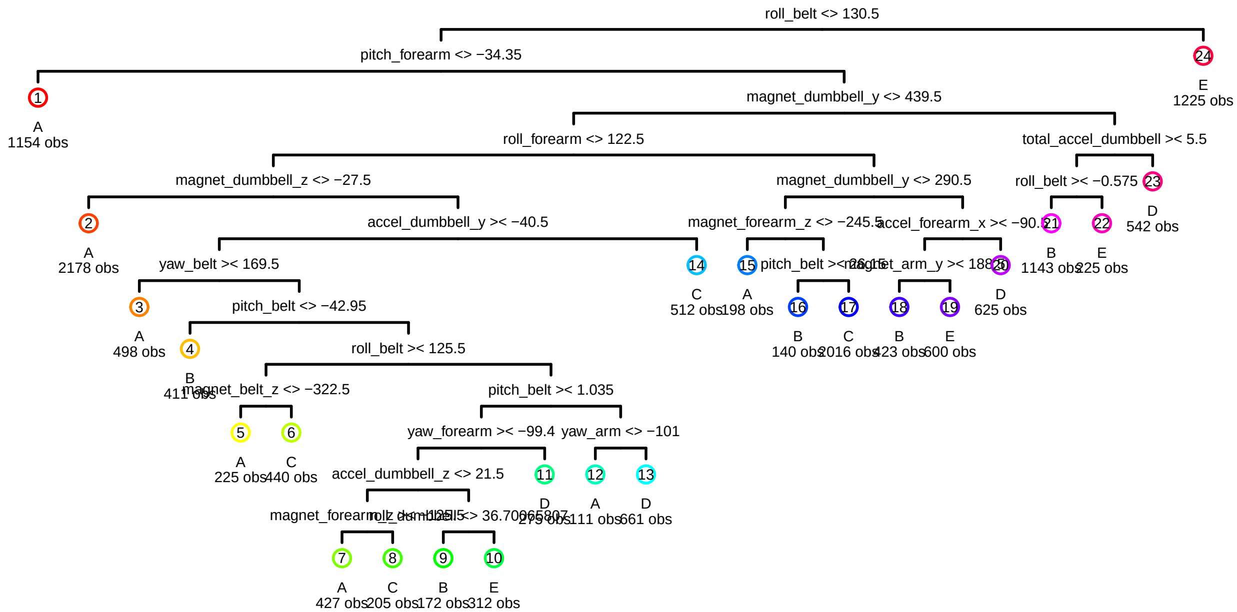
```
## [1] "Out of sample accuracies compared: "
```

```
compare_validation_result_mod
```

```
##   Model  Accuracy
## 1  CART 0.7659690
## 2   LDA 0.7012775
## 3    RF 1.0000000
```

The revised results show that without training on the time or index or participant name data, the LDA model suffered a significant hit in accuracy but the random forest was still spot on, making it the clear winner in this case.

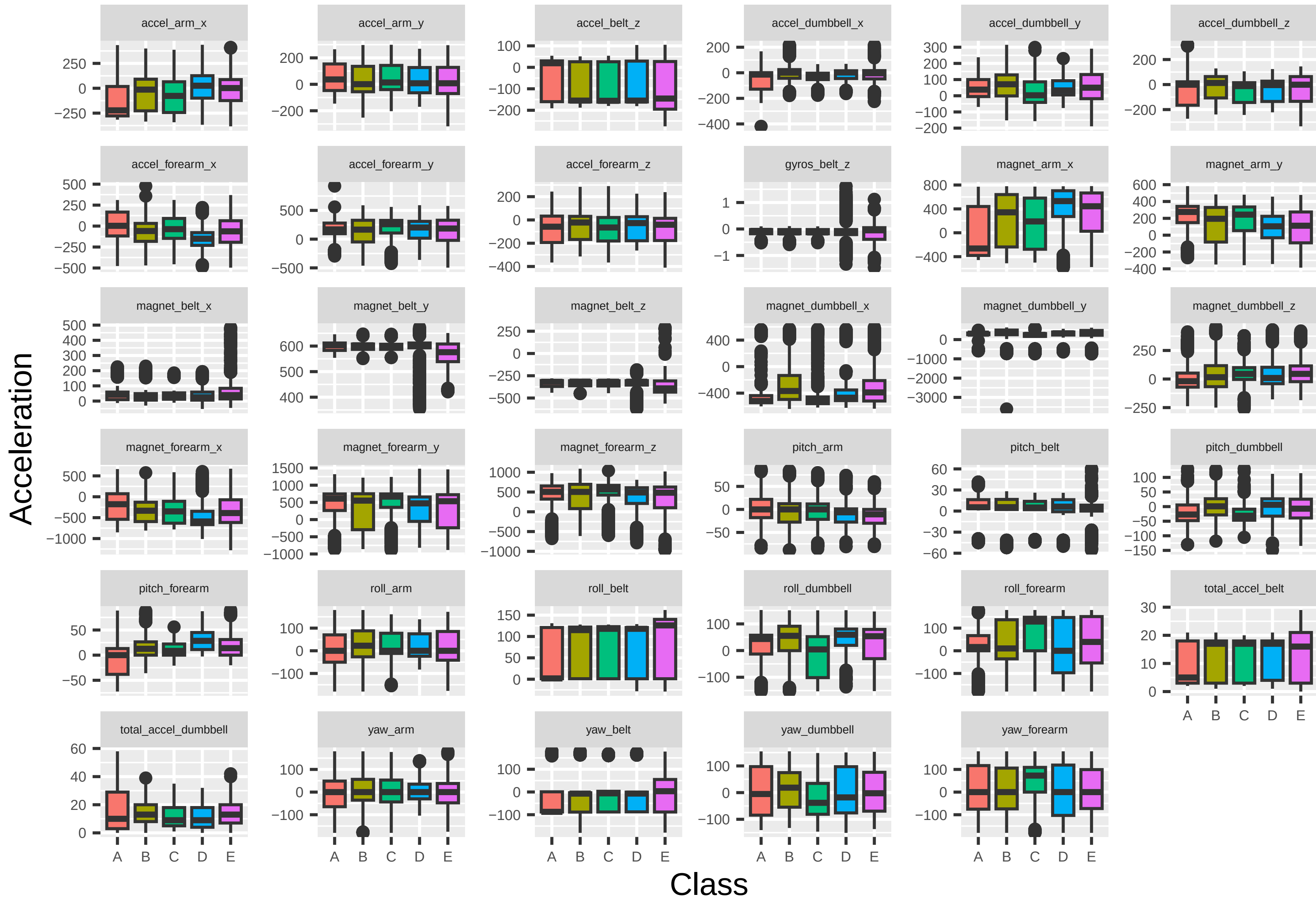
For kicks, let's see how the variable importance and tree graphic changed for the CART model:


roll_belt <> 130.5
pitch_forearm <> −34.35
magnet_dumbbell_y <> 439.5
roll_forearm <> 122.5
total_accel_dumbbell >< 5.5
magnet_dumbbell_z <> −27.5
magnet_dumbbell_y <> 290.5
roll_belt >< −0.575
accel_dumbbell_y >< −40.5
magnet_forearm_z <> −245.5
yaw_belt >< 169.5
pitch_belt <> −42.95
roll_belt >< 125.5
magnet_belt_z <> −322.5
pitch_belt >< 1.035
yaw_forearm >< −99.4
yaw_arm <> −101
accel_dumbbell_z <> 21.5
A 1154 obs
A 2178 obs
A 498 obs
B 411 obs
A 225 obs
C 440 obs
A 427 obs
C 205 obs
B 172 obs
E 312 obs
D 275 obs
A 111 obs
D 661 obs
C 512 obs
A 198 obs
B 140 obs
C 2016 obs
B 423 obs
E 600 obs
D 625 obs
B 1143 obs
E 225 obs
D 542 obs
E 1225 obs

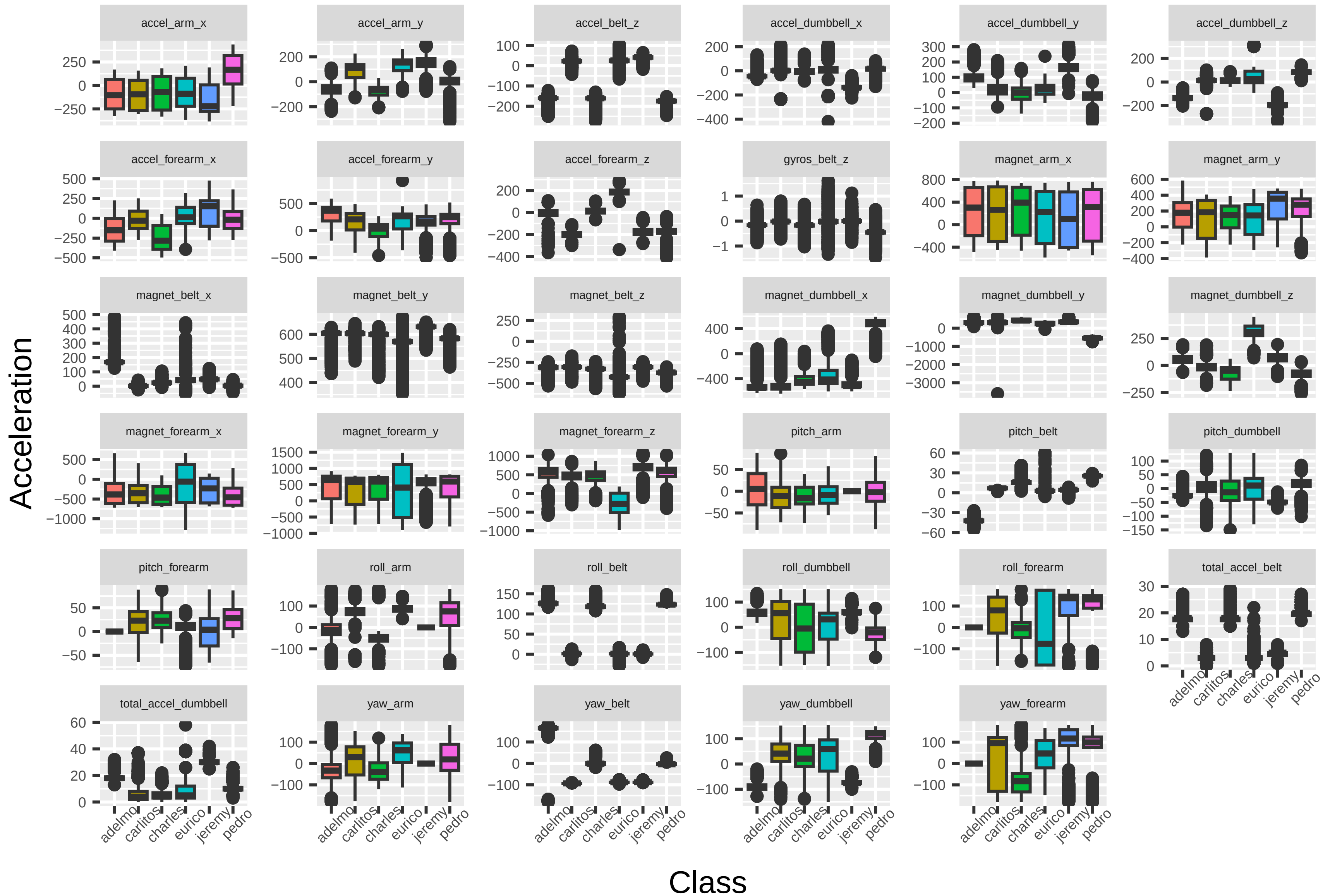
```
## [1] "Variable Imporantance for revised CART model"

##                          Overall
## roll_belt             100.000000
## pitch_forearm          93.521359
## yaw_belt               83.753915
## roll_forearm           76.225829
## magnet_dumbbell_z      72.488083
## pitch_belt             65.026351
## magnet_dumbbell_y      51.941003
## accel_dumbbell_y       51.458445
## accel_belt_z           42.753580
## yaw_arm                39.402090
## roll_dumbbell          37.468487
## accel_forearm_x        35.763103
## magnet_forearm_z       30.990547
## magnet_belt_y          29.949410
## magnet_dumbbell_x      27.780560
## accel_arm_x            27.271228
## total_accel_belt       26.020243
## magnet_belt_z          25.255013
## accel_dumbbell_z       24.207992
## magnet_arm_x           19.989004
## total_accel_dumbbell   14.203555
## magnet_forearm_x       14.096076
## gyros_belt_z           11.815115
## magnet_arm_y           11.524404
## yaw_forearm            11.203601
## roll_arm               11.063536
## magnet_forearm_y       10.287327
## accel_forearm_y         9.604683
## accel_forearm_z         8.932615
## pitch_dumbbell          5.328364
## yaw_dumbbell            3.812131
## pitch_arm               3.564180
## accel_dumbbell_x        3.232252
## accel_arm_y             3.231147
## magnet_belt_x           1.958694
## gyros_belt_x            0.000000
```

```
## gyros_belt_y           0.000000
## accel_belt_x           0.000000
## accel_belt_y           0.000000
## total_accel_arm        0.000000
## gyros_arm_x            0.000000
## gyros_arm_y            0.000000
## gyros_arm_z            0.000000
## accel_arm_z            0.000000
## magnet_arm_z           0.000000
## gyros_dumbbell_x       0.000000
## gyros_dumbbell_y       0.000000
## gyros_dumbbell_z       0.000000
## total_accel_forearm    0.000000
## gyros_forearm_x        0.000000
## gyros_forearm_y        0.000000
## gyros_forearm_z        0.000000
```

```
## [1] "variables used in the revised CART model"
```



accel_arm_x
accel_arm_y
accel_belt_z
accel_dumbbell_x
accel_dumbbell_y
accel_dumbbell_z
accel_forearm_x
accel_forearm_y
accel_forearm_z
gyros_belt_z
magnet_arm_x
magnet_arm_y
magnet_belt_x
magnet_belt_y
magnet_belt_z
magnet_dumbbell_x
magnet_dumbbell_y
magnet_dumbbell_z
magnet_forearm_x
magnet_forearm_y
magnet_forearm_z
pitch_arm
pitch_belt
pitch_dumbbell
pitch_forearm
roll_arm
roll_belt
roll_dumbbell
roll_forearm
total_accel_belt
total_accel_dumbbell
yaw_arm
yaw_belt
yaw_dumbbell
yaw_forearm
Acceleration
Class
A B C D E



accel_arm_x
accel_arm_y
accel_belt_z
accel_dumbbell_x
accel_dumbbell_y
accel_dumbbell_z
accel_forearm_x
accel_forearm_y
accel_forearm_z
gyros_belt_z
magnet_arm_x
magnet_arm_y
magnet_belt_x
magnet_belt_y
magnet_belt_z
magnet_dumbbell_x
magnet_dumbbell_y
magnet_dumbbell_z
magnet_forearm_x
magnet_forearm_y
magnet_forearm_z
pitch_arm
pitch_belt
pitch_dumbbell
pitch_forearm
roll_arm
roll_belt
roll_dumbbell
roll_forearm
total_accel_belt
total_accel_dumbbell
yaw_arm
yaw_belt
yaw_dumbbell
yaw_forearm
adelmo
carlitos
charles
eurico
jeremy
pedro
Acceleration
Class

Conclusion

The random forest decision tree was computationally expensive when un-tuned. Tuning was very effective at reducing the computational time. The accuracy was excellent. The attempt at improving accuracy further by trying to eliminated and replace outliers didn't add any value but some lessons were learnt in the process. The LDA and CART models were simple but not as effective.

Training models without the time or index variables still has a good result and will deliver a model that will be more useful for subsequent predictions of similar body sensor data.

Appendix

Session Info

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_Australia.utf8  LC_CTYPE=English_Australia.utf8
## [3] LC_MONETARY=English_Australia.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_Australia.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] maptree_1.4-8       rpart_4.1.16        cluster_2.1.3
##  [4] plotly_4.10.1       reprtree_0.6        plotrix_3.8-2
##  [7] tree_1.0-42         devtools_2.4.5      usethis_2.1.6
## [10] randomForest_4.7-1.1 corrplot_0.92      caret_6.0-93
## [13] lattice_0.20-45     patchwork_1.1.2     forcats_0.5.2
## [16] stringr_1.4.1       purrr_0.3.5         readr_2.1.3
## [19] tidyr_1.2.1         tibble_3.1.8        ggplot2_3.4.0
## [22] tidyverse_1.3.2     dplyr_1.0.10
##
## loaded via a namespace (and not attached):
##   [1] googledrive_2.0.0    colorspace_2.0-3     ellipsis_0.3.2
##   [4] class_7.3-20         fs_1.5.2             rstudioapi_0.14
##   [7] proxy_0.4-27         listenv_0.8.0        farver_2.1.1
##  [10] remotes_2.4.2        prodlim_2019.11.13   fansi_1.0.3
##  [13] lubridate_1.9.0      xml2_1.3.3           codetools_0.2-18
##  [16] splines_4.2.0        cachem_1.0.6         knitr_1.40
```

```
##  [19] pkgload_1.3.2       jsonlite_1.8.3       pROC_1.18.0
##  [22] broom_1.0.1         dbplyr_2.2.1         shiny_1.7.3
##  [25] compiler_4.2.0      httr_1.4.4           backports_1.4.1
##  [28] lazyeval_0.2.2      assertthat_0.2.1     Matrix_1.5-3
##  [31] fastmap_1.1.0       gargle_1.2.1         cli_3.3.0
##  [34] later_1.3.0         prettyunits_1.1.1    htmltools_0.5.3
##  [37] tools_4.2.0         gtable_0.3.1         glue_1.6.2
##  [40] reshape2_1.4.4      Rcpp_1.0.9           cellranger_1.1.0
##  [43] vctrs_0.5.0         nlme_3.1-157         iterators_1.0.14
##  [46] timeDate_4021.106   gower_1.0.0          xfun_0.32
##  [49] ps_1.7.2            globals_0.16.2       rvest_1.0.3
##  [52] timechange_0.1.1    mime_0.12            miniUI_0.1.1.1
##  [55] lifecycle_1.0.3     googlesheets4_1.0.1  future_1.29.0
##  [58] MASS_7.3-56         scales_1.2.1         ipred_0.9-13
##  [61] hms_1.1.2           promises_1.2.0.1     parallel_4.2.0
##  [64] yaml_2.3.6          memoise_2.0.1        stringi_1.7.8
##  [67] foreach_1.5.2       e1071_1.7-12         pkgbuild_1.3.1
##  [70] hardhat_1.2.0       lava_1.7.0           rlang_1.0.6
##  [73] pkgconfig_2.0.3     evaluate_0.18        htmlwidgets_1.5.4
##  [76] recipes_1.0.3       labeling_0.4.2       processx_3.8.0
##  [79] tidyselect_1.2.0    parallelly_1.32.1    plyr_1.8.8
##  [82] magrittr_2.0.3      R6_2.5.1             profvis_0.3.7
##  [85] generics_0.1.3      DBI_1.1.3            pillar_1.8.1
##  [88] haven_2.5.1         withr_2.5.0          survival_3.3-1
##  [91] nnet_7.3-17         future.apply_1.10.0  modelr_0.1.10
##  [94] crayon_1.5.2        utf8_1.2.2           urlchecker_1.0.1
##  [97] tzdb_0.3.0          rmarkdown_2.18       grid_4.2.0
## [100] readxl_1.4.1        data.table_1.14.4    callr_3.7.3
## [103] ModelMetrics_1.2.2.2 reprex_2.0.2        digest_0.6.29
## [106] xtable_1.8-4        httpuv_1.6.6         stats4_4.2.0
## [109] munsell_0.5.0       viridisLite_0.4.1    sessioninfo_1.2.2
```