Predictions using the Human Activity Recognition Dataset

Csaba Iranyi

The Question

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it.

In this project, I will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants.
Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions:

Movement type	**Class**
Exactly according to the specification	Class A
Throwing the elbows to the front	Class B
Lifting the dumbbell only halfway	Class C
Lowering the dumbbell only halfway	Class D
Throwing the hips to the fron	Class E

Based on a dataset provide by Human Activity Recognition (HAR) http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har) I will try:

- to train and evaluate some multi-class classification predictive models using the training dataset with 159 features and one label (classe),
- to use the trained prediction models to predict 20 different test cases (what exercise was performed) from the testing dataset.

Getting data

```
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE, fig.width=10, fig.height=5)
options(width=120)

library(caret)
library(randomForest)
library(ggcorrplot)
library(ggpubr)
```

Downloading of traning and testing datasets.

```
# Downloading training dataset
training.data <- read.table(file = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                            header = TRUE,
                            sep = ",",
                            encoding = "ISO_8859-1")

# Downloading training dataset
testing.data <- read.table(file = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                           header = TRUE,
                           sep = ",",
                           encoding = "ISO_8859-1")
```

Exploratory data analysis

Examining the dimensions and the features of the training dataset.

```
# Number of features
cat("Number of features in training dataset:", length(names(training.data)) - 1, "\n")
```

```
## Number of features in training dataset: 159
```

```
cat("Number of features in testing dataset:", length(names(training.data)) - 1, "\n")
```

```
## Number of features in testing dataset: 159
```

```
# Number of rows
cat("Number of observations in training dataset:", nrow(training.data), "\n")
```

```
## Number of observations in training dataset: 19622
```

```
cat("Number of observations in testing dataset:", nrow(testing.data), "\n")
```

```
## Number of observations in testing dataset: 20
```

```
# Create summary of features (and label)
feature.summary <- data.frame(row.names = 1:length(names(training.data)))
feature.summary$feature.index <- 1:length(names(training.data))
feature.summary$feature.name <- names(training.data)
feature.summary$type <- sapply(training.data, class)
feature.summary$NA.rows <- sapply(training.data, function(x) sum(is.na(x)))
feature.summary$NA.rows.percent <- sapply(training.data, function(x) round(sum(is.
na(x)) / length(x) * 100.0, digits = 2))
feature.summary$empty.rows <- sapply(training.data, function(x) length(x[x == ""])
)
feature.summary$empty.rows.percent <- sapply(training.data, function(x) round(leng
th(x[x == ""]) / length(x) * 100.0, digits = 2))
feature.summary$unique.values <- sapply(training.data, function(x) length(unique(x
)))
feature.summary[, -1]
```

```
##                 feature.name    type NA.rows NA.rows.percent empty.rows empty.r
ows.percent unique.values
## 1                          X integer       0            0.00          0
0.00         19622
## 2                  user_name  factor       0            0.00          0
0.00             6
## 3       raw_timestamp_part_1 integer       0            0.00          0
0.00           837
## 4       raw_timestamp_part_2 integer       0            0.00          0
0.00         16783
## 5            cvtd_timestamp  factor       0            0.00          0
0.00            20
## 6                new_window  factor       0            0.00          0
0.00             2
## 7                num_window integer       0            0.00          0
0.00           858
## 8                 roll_belt numeric       0            0.00          0
0.00          1330
## 9                pitch_belt numeric       0            0.00          0
0.00          1840
## 10                 yaw_belt numeric       0            0.00          0
0.00          1957
## 11          total_accel_belt integer       0            0.00          0
0.00            29
## 12        kurtosis_roll_belt  factor       0            0.00      19216
97.93           397
## 13       kurtosis_picth_belt  factor       0            0.00      19216
97.93           317
## 14         kurtosis_yaw_belt  factor       0            0.00      19216
97.93             2
## 15        skewness_roll_belt  factor       0            0.00      19216
97.93           395
## 16      skewness_roll_belt.1  factor       0            0.00      19216
97.93           338
```

```
## 17         skewness_yaw_belt  factor       0            0.00      19216
97.93            2
## 18            max_roll_belt numeric   19216           97.93      19216
97.93          196
## 19           max_picth_belt integer   19216           97.93      19216
97.93           23
## 20             max_yaw_belt  factor       0            0.00      19216
97.93           68
## 21            min_roll_belt numeric   19216           97.93      19216
97.93          185
## 22           min_pitch_belt integer   19216           97.93      19216
97.93           17
## 23             min_yaw_belt  factor       0            0.00      19216
97.93           68
## 24       amplitude_roll_belt numeric   19216           97.93      19216
97.93          149
## 25      amplitude_pitch_belt integer   19216           97.93      19216
97.93           14
## 26        amplitude_yaw_belt  factor       0            0.00      19216
97.93            4
## 27      var_total_accel_belt numeric   19216           97.93      19216
97.93           66
## 28            avg_roll_belt numeric   19216           97.93      19216
97.93          192
## 29         stddev_roll_belt numeric   19216           97.93      19216
97.93           70
## 30            var_roll_belt numeric   19216           97.93      19216
97.93           97
## 31           avg_pitch_belt numeric   19216           97.93      19216
97.93          215
## 32        stddev_pitch_belt numeric   19216           97.93      19216
97.93           44
## 33           var_pitch_belt numeric   19216           97.93      19216
97.93           64
## 34             avg_yaw_belt numeric   19216           97.93      19216
97.93          241
## 35           stddev_yaw_belt numeric   19216           97.93      19216
97.93           59
## 36             var_yaw_belt numeric   19216           97.93      19216
97.93          146
## 37             gyros_belt_x numeric       0            0.00          0
0.00           140
## 38             gyros_belt_y numeric       0            0.00          0
0.00            69
## 39             gyros_belt_z numeric       0            0.00          0
0.00           169
## 40             accel_belt_x integer       0            0.00          0
0.00           164
## 41             accel_belt_y integer       0            0.00          0
0.00           143
## 42             accel_belt_z integer       0            0.00          0
0.00           299
## 43            magnet_belt_x integer       0            0.00          0
0.00           327
```

```
## 44             magnet_belt_y integer       0             0.00          0
0.00          298
## 45             magnet_belt_z integer       0             0.00          0
0.00          457
## 46                   roll_arm numeric       0             0.00          0
0.00         2654
## 47                  pitch_arm numeric       0             0.00          0
0.00         3087
## 48                    yaw_arm numeric       0             0.00          0
0.00         2876
## 49            total_accel_arm integer       0             0.00          0
0.00           66
## 50              var_accel_arm numeric   19216            97.93      19216
97.93          396
## 51               avg_roll_arm numeric   19216            97.93      19216
97.93          331
## 52            stddev_roll_arm numeric   19216            97.93      19216
97.93          331
## 53               var_roll_arm numeric   19216            97.93      19216
97.93          331
## 54              avg_pitch_arm numeric   19216            97.93      19216
97.93          331
## 55           stddev_pitch_arm numeric   19216            97.93      19216
97.93          331
## 56              var_pitch_arm numeric   19216            97.93      19216
97.93          331
## 57                avg_yaw_arm numeric   19216            97.93      19216
97.93          331
## 58             stddev_yaw_arm numeric   19216            97.93      19216
97.93          328
## 59                var_yaw_arm numeric   19216            97.93      19216
97.93          328
## 60               gyros_arm_x numeric       0             0.00          0
0.00          643
## 61               gyros_arm_y numeric       0             0.00          0
0.00          376
## 62               gyros_arm_z numeric       0             0.00          0
0.00          248
## 63               accel_arm_x integer       0             0.00          0
0.00          777
## 64               accel_arm_y integer       0             0.00          0
0.00          537
## 65               accel_arm_z integer       0             0.00          0
0.00          792
## 66              magnet_arm_x integer       0             0.00          0
0.00         1339
## 67               magnet_arm_y integer       0             0.00          0
0.00          872
## 68               magnet_arm_z integer       0             0.00          0
0.00         1265
## 69          kurtosis_roll_arm  factor       0             0.00      19216
97.93          330
## 70         kurtosis_picth_arm  factor       0             0.00      19216
97.93          328
```

```
## 71          kurtosis_yaw_arm  factor       0            0.00      19216
97.93           395
## 72         skewness_roll_arm  factor       0            0.00      19216
97.93           331
## 73        skewness_pitch_arm  factor       0            0.00      19216
97.93           328
## 74          skewness_yaw_arm  factor       0            0.00      19216
97.93           395
## 75             max_roll_arm numeric   19216           97.93      19216
97.93           291
## 76            max_picth_arm numeric   19216           97.93      19216
97.93           264
## 77                max_yaw_arm integer   19216           97.93      19216
97.93            52
## 78             min_roll_arm numeric   19216           97.93      19216
97.93           279
## 79            min_pitch_arm numeric   19216           97.93      19216
97.93           291
## 80                min_yaw_arm integer   19216           97.93      19216
97.93            39
## 81        amplitude_roll_arm numeric   19216           97.93      19216
97.93           307
## 82       amplitude_pitch_arm numeric   19216           97.93      19216
97.93           295
## 83         amplitude_yaw_arm integer   19216           97.93      19216
97.93            52
## 84             roll_dumbbell numeric       0            0.00          0
0.00         16523
## 85            pitch_dumbbell numeric       0            0.00          0
0.00         16040
## 86              yaw_dumbbell numeric       0            0.00          0
0.00         16381
## 87    kurtosis_roll_dumbbell  factor       0            0.00      19216
97.93           398
## 88   kurtosis_picth_dumbbell  factor       0            0.00      19216
97.93           401
## 89     kurtosis_yaw_dumbbell  factor       0            0.00      19216
97.93             2
## 90    skewness_roll_dumbbell  factor       0            0.00      19216
97.93           401
## 91   skewness_pitch_dumbbell  factor       0            0.00      19216
97.93           402
## 92     skewness_yaw_dumbbell  factor       0            0.00      19216
97.93             2
## 93         max_roll_dumbbell numeric   19216           97.93      19216
97.93           339
## 94        max_picth_dumbbell numeric   19216           97.93      19216
97.93           340
## 95          max_yaw_dumbbell  factor       0            0.00      19216
97.93            73
## 96         min_roll_dumbbell numeric   19216           97.93      19216
97.93           333
## 97        min_pitch_dumbbell numeric   19216           97.93      19216
97.93           357
```

```
## 98          min_yaw_dumbbell  factor       0            0.00      19216
97.93           73
## 99   amplitude_roll_dumbbell numeric   19216           97.93      19216
97.93          388
## 100 amplitude_pitch_dumbbell numeric   19216           97.93      19216
97.93          384
## 101   amplitude_yaw_dumbbell  factor       0            0.00      19216
97.93            3
## 102     total_accel_dumbbell integer       0            0.00          0
0.00           43
## 103       var_accel_dumbbell numeric   19216           97.93      19216
97.93          385
## 104        avg_roll_dumbbell numeric   19216           97.93      19216
97.93          398
## 105     stddev_roll_dumbbell numeric   19216           97.93      19216
97.93          392
## 106        var_roll_dumbbell numeric   19216           97.93      19216
97.93          392
## 107       avg_pitch_dumbbell numeric   19216           97.93      19216
97.93          398
## 108    stddev_pitch_dumbbell numeric   19216           97.93      19216
97.93          392
## 109       var_pitch_dumbbell numeric   19216           97.93      19216
97.93          392
## 110         avg_yaw_dumbbell numeric   19216           97.93      19216
97.93          398
## 111      stddev_yaw_dumbbell numeric   19216           97.93      19216
97.93          392
## 112         var_yaw_dumbbell numeric   19216           97.93      19216
97.93          392
## 113         gyros_dumbbell_x numeric       0            0.00          0
0.00           241
## 114         gyros_dumbbell_y numeric       0            0.00          0
0.00           278
## 115         gyros_dumbbell_z numeric       0            0.00          0
0.00           206
## 116         accel_dumbbell_x integer       0            0.00          0
0.00           425
## 117         accel_dumbbell_y integer       0            0.00          0
0.00           466
## 118         accel_dumbbell_z integer       0            0.00          0
0.00           410
## 119        magnet_dumbbell_x integer       0            0.00          0
0.00          1128
## 120        magnet_dumbbell_y integer       0            0.00          0
0.00           844
## 121        magnet_dumbbell_z numeric       0            0.00          0
0.00           676
## 122            roll_forearm numeric       0            0.00          0
0.00          2176
## 123           pitch_forearm numeric       0            0.00          0
0.00          2915
## 124             yaw_forearm numeric       0            0.00          0
0.00          1991
```

```
## 125    kurtosis_roll_forearm  factor       0            0.00      19216
97.93           322
## 126   kurtosis_picth_forearm  factor       0            0.00      19216
97.93           323
## 127     kurtosis_yaw_forearm  factor       0            0.00      19216
97.93             2
## 128    skewness_roll_forearm  factor       0            0.00      19216
97.93           323
## 129   skewness_pitch_forearm  factor       0            0.00      19216
97.93           319
## 130     skewness_yaw_forearm  factor       0            0.00      19216
97.93             2
## 131         max_roll_forearm numeric   19216           97.93      19216
97.93           272
## 132        max_picth_forearm numeric   19216           97.93      19216
97.93           156
## 133         max_yaw_forearm  factor       0            0.00      19216
97.93            45
## 134         min_roll_forearm numeric   19216           97.93      19216
97.93           270
## 135        min_pitch_forearm numeric   19216           97.93      19216
97.93           172
## 136         min_yaw_forearm  factor       0            0.00      19216
97.93            45
## 137   amplitude_roll_forearm numeric   19216           97.93      19216
97.93           294
## 138  amplitude_pitch_forearm numeric   19216           97.93      19216
97.93           184
## 139    amplitude_yaw_forearm  factor       0            0.00      19216
97.93             3
## 140      total_accel_forearm integer       0            0.00          0
0.00            70
## 141        var_accel_forearm numeric   19216           97.93      19216
97.93           400
## 142         avg_roll_forearm numeric   19216           97.93      19216
97.93           323
## 143      stddev_roll_forearm numeric   19216           97.93      19216
97.93           321
## 144         var_roll_forearm numeric   19216           97.93      19216
97.93           321
## 145        avg_pitch_forearm numeric   19216           97.93      19216
97.93           325
## 146     stddev_pitch_forearm numeric   19216           97.93      19216
97.93           324
## 147        var_pitch_forearm numeric   19216           97.93      19216
97.93           325
## 148         avg_yaw_forearm numeric   19216           97.93      19216
97.93           325
## 149       stddev_yaw_forearm numeric   19216           97.93      19216
97.93           323
## 150         var_yaw_forearm numeric   19216           97.93      19216
97.93           323
## 151         gyros_forearm_x numeric       0            0.00          0
0.00          298
```

```
## 152          gyros_forearm_y numeric       0            0.00          0
0.00           741
## 153          gyros_forearm_z numeric       0            0.00          0
0.00           307
## 154          accel_forearm_x integer       0            0.00          0
0.00           794
## 155          accel_forearm_y integer       0            0.00          0
0.00          1003
## 156          accel_forearm_z integer       0            0.00          0
0.00           580
## 157         magnet_forearm_x integer       0            0.00          0
0.00          1524
## 158         magnet_forearm_y numeric       0            0.00          0
0.00          1872
## 159         magnet_forearm_z numeric       0            0.00          0
0.00          1683
## 160                   classe  factor       0            0.00          0
0.00             5
```

Lot of features has either NA or empty values in great percent.

Cleaning data

Removing useless features

Removing constant features and features with NA or empty rows from the training and testing data as well.

```
# Removing useless features
#  - Constant features
#  - Features with NA rows
#  - Features with empty rows
useless.feature.indices <- feature.summary[feature.summary$unique.values == 1 | fe
ature.summary$NA.rows != 0 | feature.summary$empty.rows !=0, 1]
cat("Number of removable useless features:", length(useless.feature.indices), "\n"
)
```

```
## Number of removable useless features: 100
```

```
training.data <- training.data[, -useless.feature.indices]
testing.data <- testing.data[, -useless.feature.indices]
```

Removing unnecessary features

Removing the sequence feature (X) and all time related features () from the training and testing data.

```
# Removing unnecessary features
unnecessary.feature.indices <- grep("timestamp|X", names(training.data))
cat("Number of removable unnecessary features:", length(unnecessary.feature.indice
s), "\n")
```

```
## Number of removable unnecessary features: 4
```

```
training.data <- training.data[, -unnecessary.feature.indices]
testing.data <- testing.data[, -unnecessary.feature.indices]
```

Converting features

Converting all features to numeric type (except the label column).

```
# Converting all features to numeric type (except the label column)
label.column <- training.data$classe
training.data <- data.frame(data.matrix(training.data))
training.data$classe <- label.column
testing.data <- data.frame(data.matrix(testing.data))

# Number of usable features
cat("Number of usable features:", length(names(training.data)) - 1, "\n")
```

```
## Number of usable features: 55
```

Splitting data

The downloaded test dataset (20 observations) is the the ultimate validation set (one time scoring). Splitting up the downloaded training dataset into a cross-validating (25%) and training dataset (75%).

```
# Initialize RNG
set.seed(333)

# Crete training/CV datasets
partition.indices <- createDataPartition(y = training.data$classe, p = 0.75, list
= FALSE)
cv.data <- training.data[-partition.indices,]
training.data <- training.data[partition.indices,]

cat("Number of rows in training dataset:", nrow(training.data), "\n")
```

```
## Number of rows in training dataset: 14718
```

```
cat("Number of rows in cross-validation dataset:", nrow(cv.data), "\n")
```

```
## Number of rows in cross-validation dataset: 4904
```

```
cat("Number of rows in testing dataset:", nrow(testing.data), "\n")
```

```
## Number of rows in testing dataset: 20
```

Feature engineering

Finding feature correlations

Calculating feature correlations with the outcome (classe).

```
# Label (outcome) column index
label.index <- which(names(training.data) == "classe")

# Create summary of usable features
feature.set <- training.data[,-label.index]
feature.summary <- data.frame(row.names = 1:length(names(feature.set)))
feature.summary$feature.index <- 1:length(names(feature.set))
feature.summary$feature.name <- names(feature.set)
feature.summary$type <- sapply(feature.set, class)
feature.summary$unique <- sapply(feature.set, function(x) length(unique(x)))
feature.summary$mean <- sapply(feature.set, function(x) round(mean(x), digits = 4)
)
feature.summary$sum <- sapply(feature.set, function(x) round(sum(x), digits = 4))
feature.summary$sd <- sapply(feature.set, function(x) round(sd(x), digits = 4))
feature.summary$cor <- sapply(feature.set, function(x) round(abs(cor(x, as.numeric
(training.data$classe))), digits = 4))
feature.summary <- feature.summary[order(feature.summary$cor, decreasing = TRUE),]
feature.summary[, -1]
```

```
##             feature.name    type unique      mean          sum       sd    cor
## 44        pitch_forearm numeric   2681   10.7231   157822.40  28.0715 0.3437
## 27         magnet_arm_x numeric   1323  193.8237  2852697.00 443.2437 0.2987
## 15        magnet_belt_y numeric    288  593.7373  8738625.00  35.0887 0.2817
## 28         magnet_arm_y numeric    857  156.1741  2298570.00 201.4303 0.2543
## 24          accel_arm_x numeric    767  -59.7171  -878917.00 182.5694 0.2403
## 50      accel_forearm_x numeric    785  -62.6650  -922303.00 181.1984 0.1874
## 53     magnet_forearm_x numeric   1463 -313.3946 -4612542.00 345.9312 0.1768
## 18            pitch_arm numeric   2858   -4.6924   -69062.66  30.7253 0.1759
## 16        magnet_belt_z numeric    437 -345.4974 -5085030.00  64.7346 0.1712
## 46  total_accel_forearm numeric     67   34.7744   511809.00  10.0465 0.1565
## 29         magnet_arm_z numeric   1258  306.6023  4512572.00 326.5414 0.1519
## 42    magnet_dumbbell_z numeric    665   45.9814   676754.60 140.1708 0.1482
## 20      total_accel_arm numeric     66   25.5113   375476.00  10.4827 0.1201
## 54     magnet_forearm_y numeric   1836  378.7137  5573907.88 510.6675 0.1162
## 37     accel_dumbbell_x numeric    412  -28.3435  -417160.00  66.9324 0.1131
## 31       pitch_dumbbell numeric  12315  -10.7792  -158648.36  36.8117 0.0867
## 17             roll_arm numeric   2440   17.8867   263257.02  72.8711 0.0842
## 25          accel_arm_y numeric    527   32.5860   479601.00 109.7806 0.0838
## 13         accel_belt_z numeric    293  -72.6519 -1069290.00 100.3117 0.0756
## 7      total_accel_belt numeric     28   11.3036   166367.00   7.7427 0.0745
## 40    magnet_dumbbell_x numeric   1076 -328.5024 -4834898.00 339.7504 0.0626
## 39     accel_dumbbell_z numeric    401  -37.4433  -551090.00 109.1693 0.0620
## 4              roll_belt numeric   1149   64.4274   948242.40  62.6976 0.0600
## 19               yaw_arm numeric   2668   -0.6943   -10218.06  71.3012 0.0517
## 30        roll_dumbbell numeric  12662   23.8458   350962.71  70.2060 0.0497
## 45          yaw_forearm numeric   1827   18.7492   275950.32 103.2239 0.0462
```

```
## 55     magnet_forearm_z numeric   1629  391.7553  5765854.91 369.5289 0.0430
## 26          accel_arm_z numeric    770  -70.8377 -1042589.00 134.2652 0.0420
## 33 total_accel_dumbbell numeric     42   13.6253   200537.00  10.2235 0.0411
## 35     gyros_dumbbell_y numeric    270    0.0484      711.83   0.6457 0.0340
## 51      accel_forearm_y numeric    983  162.6670  2394133.00 200.8818 0.0304
## 43         roll_forearm numeric   1967   33.4958   492990.89 108.2365 0.0247
## 3            num_window numeric    858  431.6262  6352674.00 248.7035 0.0245
## 14        magnet_belt_x numeric    308   55.3338   814403.00  63.8920 0.0220
## 9          gyros_belt_y numeric     69    0.0394      579.38   0.0781 0.0216
## 22          gyros_arm_y numeric    369   -0.2621    -3857.72   0.8546 0.0214
## 8          gyros_belt_x numeric    131   -0.0059      -87.48   0.2085 0.0190
## 32         yaw_dumbbell numeric  12571    2.1753    32015.52  82.4508 0.0185
## 47      gyros_forearm_x numeric    278    0.1564     2302.05   0.6523 0.0178
## 6              yaw_belt numeric   1829  -11.4666  -168765.07  94.9211 0.0151
## 1             user_name numeric      6    3.3462    49250.00   1.6933 0.0125
## 48      gyros_forearm_y numeric    718    0.0790     1162.18   3.3551 0.0125
## 11         accel_belt_x numeric    159   -5.6843   -83662.00  29.5720 0.0111
## 49      gyros_forearm_z numeric    292    0.1544     2273.09   1.9938 0.0100
## 23          gyros_arm_z numeric    237    0.2690     3959.77   0.5549 0.0099
## 10         gyros_belt_z numeric    161   -0.1299    -1912.44   0.2416 0.0094
## 36     gyros_dumbbell_z numeric    199   -0.1215    -1788.73   2.6337 0.0083
## 34     gyros_dumbbell_x numeric    236    0.1548     2278.90   1.7272 0.0082
## 21          gyros_arm_x numeric    638    0.0547      805.09   2.0017 0.0073
## 5            pitch_belt numeric   1727    0.4171     6139.26  22.3177 0.0065
## 12         accel_belt_y numeric    140   30.2068   444583.00  28.5974 0.0055
## 38     accel_dumbbell_y numeric    452   52.1879   768102.00  80.6613 0.0053
## 52      accel_forearm_z numeric    570  -54.9934  -809393.00 138.6856 0.0046
## 41    magnet_dumbbell_y numeric    828  220.8063  3249827.00 327.1579 0.0031
## 2            new_window numeric      2    1.0195    15005.00   0.1383 0.0001
```

Selecting 33 (manually choosen number) features with best correlations with outcome.

```
# The most relevant features (best correlations with outcome)
most.correlated.outcome.feature.indices <- feature.summary$feature.index[1:33]
cat("Relevant features:", feature.summary$feature.name[1], "...", feature.summary$
feature.name[33], "\n")
```

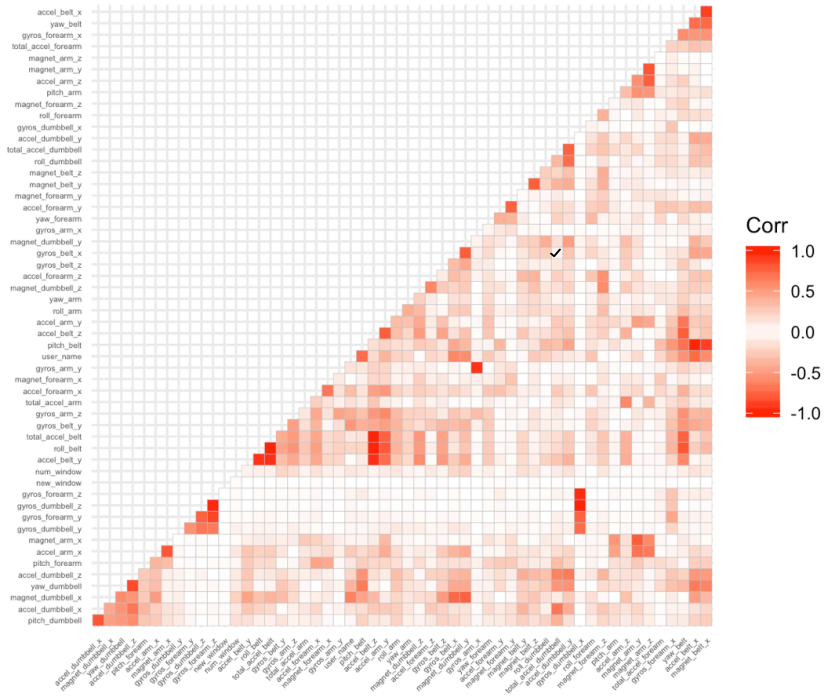
```
## Relevant features: pitch_forearm ... num_window
```

Searching for duplicated (lineary dependent) features to reduce pair-wise correlations.

```
# Duplicated feature
feature.correlation.matrix <- cor(training.data[, -label.index])
duplicated.feature.indices <- findCorrelation(feature.correlation.matrix, cutoff =
0.9, exact = TRUE)
cat("Removable duplicated features:", paste(names(training.data)[duplicated.featur
e.indices], collapse = ", "), "\n")
```

```
## Removable duplicated features: accel_belt_z, roll_belt, accel_belt_y, accel_bel
t_x, gyros_dumbbell_x, gyros_dumbbell_z, gyros_arm_x
```

```
# Plot correlation matrix
ggcorrplot(feature.correlation.matrix, hc.order = TRUE, type = "lower", colors = c
("red", "white", "red"), insig = "blank", tl.cex = list(size = 4))
```



Training and scoring models

Creating a summary dataset to store result of trained models.

```
# Summary of models
model.summary <- data.frame(row.names = 1:6, stringsAsFactors = TRUE)

# Initialize RNG
set.seed(333)
```

Setting options of random forest.

```
# Number of trees
number.trees <- 50
```

Random Forest: using all features

```
# Random Forest: all features
start.time <- Sys.time()
model.all.features <- randomForest(
  x = training.data[, -label.index],
  y = training.data$classe,
  xtest = cv.data[, -label.index],
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.all.features$ntree,
  features = ncol(training.data) - 1,
  model = "All features",
  training.accuracy = round((1 - sum(model.all.features$confusion[,"class.error"])
) * 100, digits = 3),
  cv.accuracy =round((1 - sum(model.all.features$test$confusion[,"class.error"]))
* 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Random Forest: all features with PCA

```
# Apply PCA
pre.pca <- preProcess(training.data[, -label.index], method = "pca", thresh = 0.99
)
training.data.pca <- predict(pre.pca, training.data[, -label.index])
cv.data.pca <- predict(pre.pca, cv.data[, -label.index])
testing.data.pca <- predict(pre.pca, testing.data[, -label.index])

# Random Forest: all feature with PCA
start.time <- Sys.time()
model.pca.features <- randomForest(
  x = training.data.pca,
  y = training.data$classe,
  xtest = cv.data.pca,
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.pca.features$ntree,
  features = ncol(training.data.pca),
  model = "PCA",
  training.accuracy = round((1 - sum(model.pca.features$confusion[,"class.error"])
) * 100, digits = 3),
  cv.accuracy = round((1 - sum(model.pca.features$test$confusion[,"class.error"]))
* 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Random Forest: the most correlated features

```
# Random Forest: the most correlated features
start.time <- Sys.time()
model.corr.features <- randomForest(
  x = training.data[, most.correlated.outcome.feature.indices],
  y = training.data$classe,
  xtest = cv.data[, most.correlated.outcome.feature.indices],
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.corr.features$ntree,
  features = ncol(training.data[, most.correlated.outcome.feature.indices]),
  model = "Correlated",
  training.accuracy = round((1 - sum(model.corr.features$confusion[,"class.error"]
)) * 100, digits = 3),
  cv.accuracy = round((1 - sum(model.corr.features$test$confusion[,"class.error"])
) * 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Random Forest: the most correlated features with PCA

```
# Apply PCA
pre.corr.pca <- preProcess(training.data[, most.correlated.outcome.feature.indices
], method = "pca", thresh = 0.99)
training.data.corr.pca <- predict(pre.corr.pca, training.data[, most.correlated.ou
tcome.feature.indices])
cv.data.corr.pca <- predict(pre.corr.pca, cv.data[, most.correlated.outcome.featur
e.indices])
testing.data.corr.pca <- predict(pre.corr.pca, testing.data[, most.correlated.outc
ome.feature.indices])

# Random Forest: the most correlated features with PCA
start.time <- Sys.time()
model.corr.pca.features <- randomForest(
  x = training.data.corr.pca,
  y = training.data$classe,
  xtest = cv.data.corr.pca,
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.corr.pca.features$ntree,
  features = ncol(training.data.corr.pca),
  model = "Correlated + PCA",
  training.accuracy = round((1 - sum(model.corr.pca.features$confusion[,"class.err
or"])) * 100, digits = 3),
  cv.accuracy = round((1 - sum(model.corr.pca.features$test$confusion[,"class.erro
r"])) * 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Random Forest: reduced features

```
# Random Forest: reduced features
start.time <- Sys.time()
model.reduced.features <- randomForest(
  x = training.data[, -c(duplicated.feature.indices, label.index)],
  y = training.data$classe,
  xtest = cv.data[, -c(duplicated.feature.indices, label.index)],
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.reduced.features$ntree,
  features = ncol(training.data[, -c(duplicated.feature.indices, label.index)]),
  model = "Reduced",
  training.accuracy = round((1 - sum(model.reduced.features$confusion[,"class.erro
r"])) * 100, digits = 3),
  cv.accuracy = round((1 - sum(model.reduced.features$test$confusion[,"class.error
"])) * 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Random Forest: reduced features with PCA

```
# Apply PCA
pre.reduced.pca <- preProcess(training.data[, -c(duplicated.feature.indices, label
.index)], method = "pca", thresh = 0.99)
training.data.reduced.pca <- predict(pre.reduced.pca, training.data[, -c(duplicate
d.feature.indices, label.index)])
cv.data.reduced.pca <- predict(pre.reduced.pca, cv.data[, -c(duplicated.feature.in
dices, label.index)])
testing.data.reduced.pca <- predict(pre.reduced.pca, testing.data[, -c(duplicated.
feature.indices, label.index)])

# Random Forest: reduced features with PCA
start.time <- Sys.time()
model.reduced.pca.features <- randomForest(
  x = training.data.reduced.pca,
  y = training.data$classe,
  xtest = cv.data.reduced.pca,
  ytest = cv.data$classe,
  ntree = number.trees,
  keep.forest = TRUE,
  proximity = TRUE,
  do.trace = FALSE)
end.time <- Sys.time()

model.summary <- rbind(model.summary, cbind(
  trees = model.reduced.pca.features$ntree,
  features = ncol(training.data.reduced.pca),
  model = "Reduced + PCA",
  training.accuracy = round((1 - sum(model.reduced.pca.features$confusion[,"class.
error"])) * 100, digits = 3),
  cv.accuracy = round((1 - sum(model.reduced.pca.features$test$confusion[,"class.e
rror"])) * 100, digits = 3),
  training.time = round(as.numeric(end.time - start.time, units = "secs"))
))
```

Summary of models

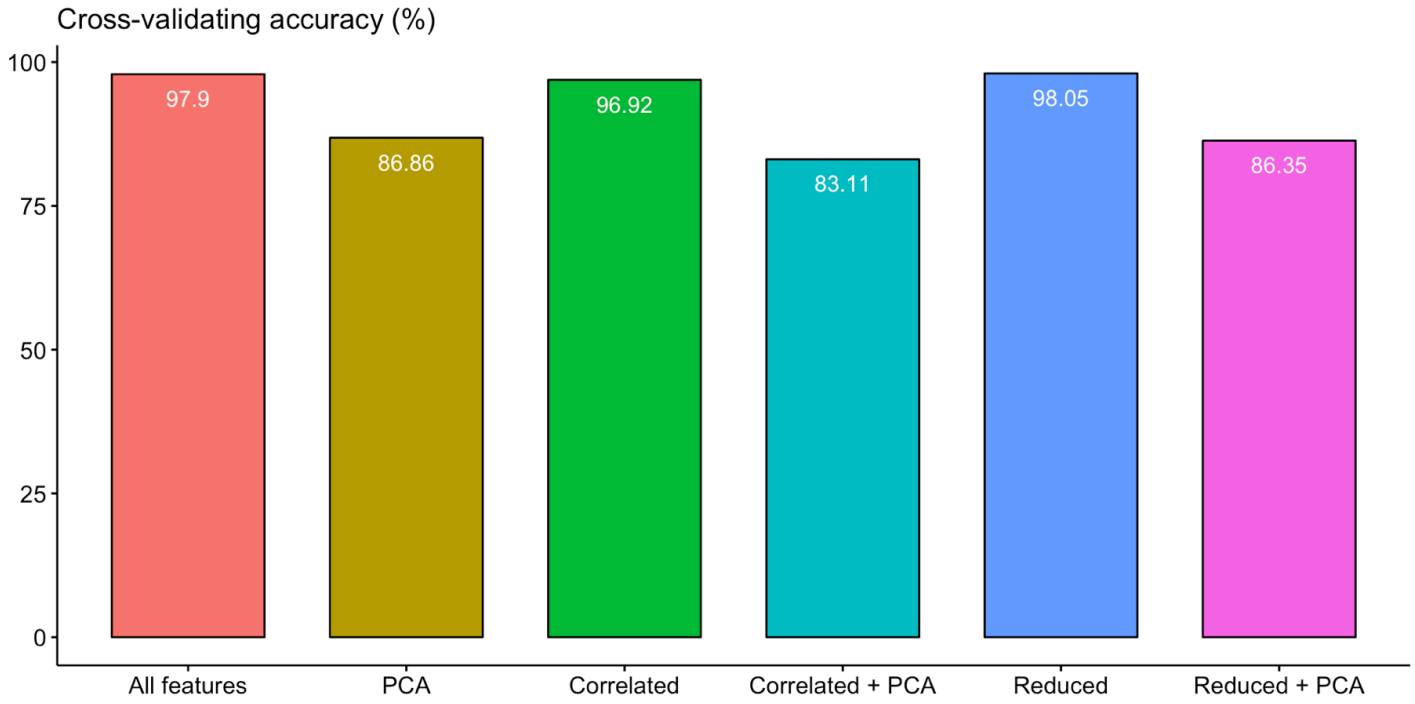
Summarizing the trained models.

```
model.summary$trees = as.numeric(as.character(model.summary$trees))
model.summary$features = as.numeric(as.character(model.summary$features))
model.summary$training.accuracy = as.numeric(as.character(model.summary$training.a
ccuracy))
model.summary$cv.accuracy = as.numeric(as.character(model.summary$cv.accuracy))
model.summary$training.time = as.numeric(as.character(model.summary$training.time)
)
model.summary
```

```
##   trees features            model training.accuracy cv.accuracy training.time
## 1    50       55     All features             97.499      97.904            41
## 2    50       38              PCA             83.505      86.858            39
## 3    50       33       Correlated             96.408      96.923            41
## 4    50       25 Correlated + PCA             80.398      83.113            39
## 5    50       48          Reduced             97.712      98.051            44
## 6    50       36    Reduced + PCA             80.878      86.352            47
```

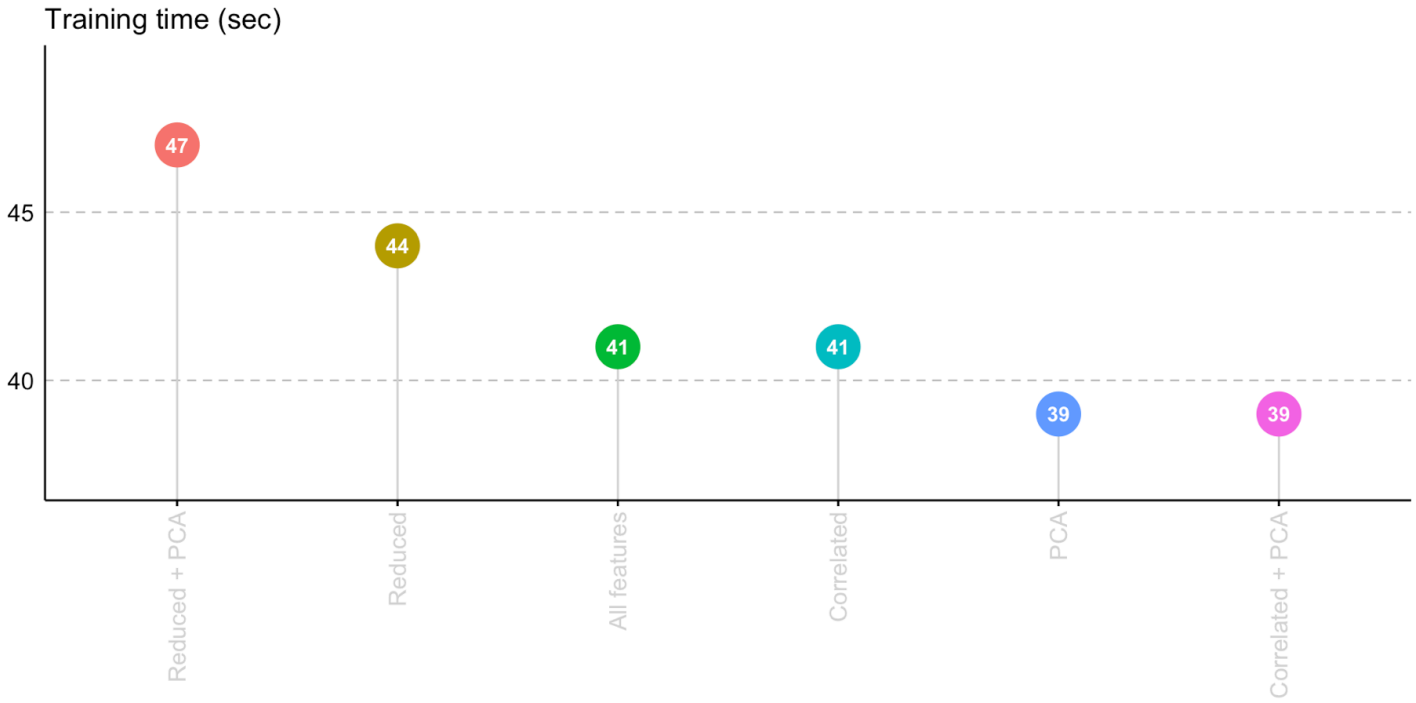
Plotting the cross-validating accuracy.

```
# Cross-validating accuracy chart
gg <- ggbarplot(
  model.summary, x = "model", y = "cv.accuracy",
  title = "Cross-validating accuracy (%)",      # Main title
  fill = "model",                               # Color by groups
  sort.val = "desc",                            # Sort value in descending order
  label = model.summary$cv.accuracy,            # Label values
  lab.pos = "in",                               # Position for labels
  lab.col = "white",                            # Color for labels
  lab.nb.digits = 2,                            # Number of decimal places (round)
  xlab = FALSE,                                 # X-axis labels
  ylab = FALSE,                                 # Y-axis labels
  rotate = FALSE,                               # Rotate vertically
  font.label = list(color = "white", size = 11, vjust = 1.5),
  ggtheme = theme_pubr()                        # ggplot2 theme
)
ggpar(gg, legend = "none", legend.title = "")   # Customize legend
```



Plotting the training times in seconds.

```
# Training time chart
gg <- ggdotchart(
  model.summary, x = "model", y = "training.time",
  title = "Training time (sec)",                 # Main title
  color = "model",                              # Color by groups
  sorting = "descending",                       # Sort value in descending order
  rotate = FALSE,                               # Rotate vertically
  label = model.summary$training.time,          # Label values
  add = "segments",                             # Add lolipop lines
  xlab = FALSE,                                 # X-axis labels
  ylab = FALSE,                                 # Y-axis labels
  dot.size = 10,                                # Large dot size
  y.text.col = TRUE,                            # Color y text by groups
  font.label = list(color = "white", size = 11, face = "bold", vjust = 0.5),
  ggtheme = theme_pubr(),                        # ggplot2 theme
)
ggpar(gg, legend = "none", legend.title = "",  ylim = c(min(model.summary$training
.time) * 0.95, max(model.summary$training.time) * 1.05)) + theme_cleveland()
```



Testing models

Using all of trained models to make predictions on the test dataset (20 observations). The ground truth dataset consist of the expected good values.

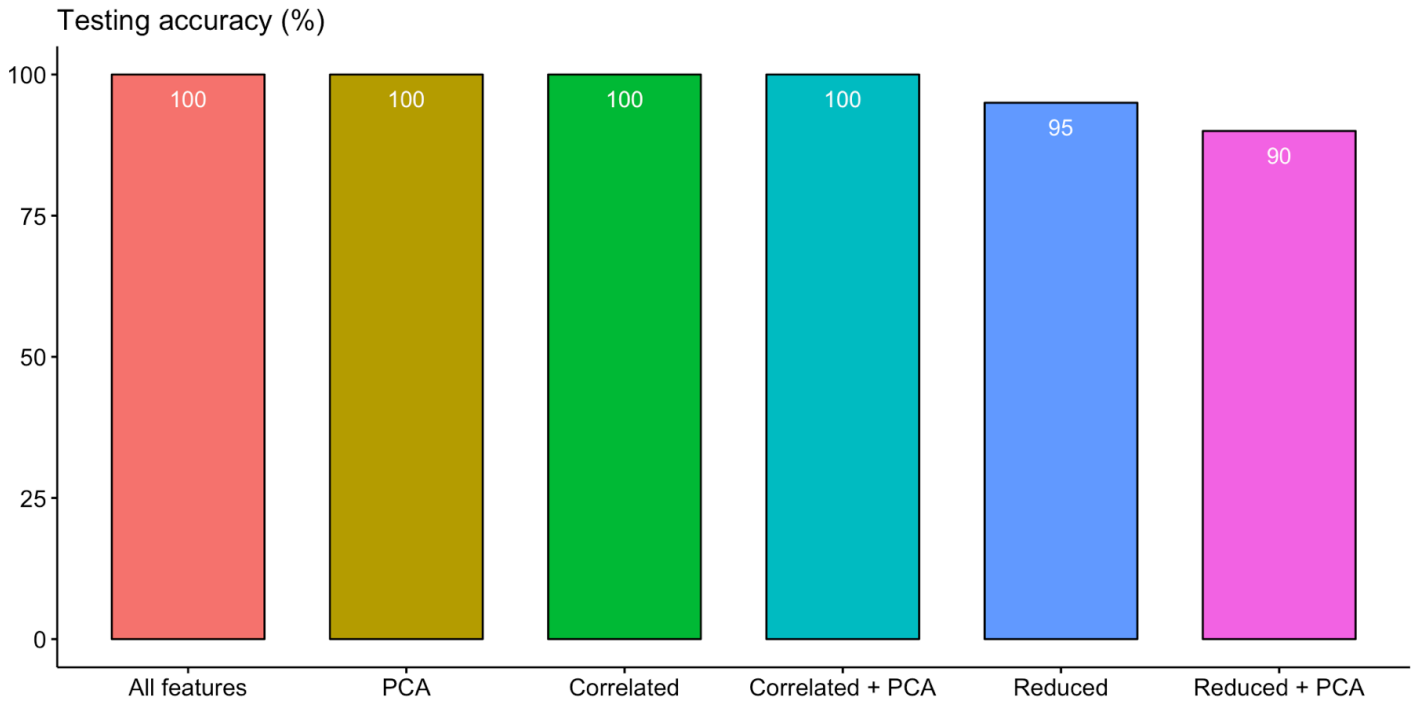
```
# Predictions of the models
test.predictions <- data.frame(row.names = 1:20, stringsAsFactors = TRUE)
test.predictions$ground.truth <- as.factor(c("B", "A", "B", "A", "A", "E", "D", "B
", "A", "A", "B", "C", "B", "A", "E", "E", "A", "B", "B", "B"))
test.predictions$all <- predict(model.all.features, testing.data[,-label.index])
test.predictions$all.with.pca <- predict(model.pca.features, testing.data.pca)
test.predictions$correlated <- predict(model.corr.features, testing.data[, most.co
rrelated.outcome.feature.indices])
test.predictions$corr.with.pca <- predict(model.corr.pca.features, testing.data.co
rr.pca)
test.predictions$reduced <- predict(model.reduced.features, testing.data[, -c(dupl
icated.feature.indices, label.index)])
test.predictions$reduced.with.pca <- predict(model.reduced.pca.features, testing.d
ata.reduced.pca)
test.predictions
```

```
##    ground.truth all all.with.pca correlated corr.with.pca reduced reduced.with.
pca
## 1             B   B            B          B             B       B
B
## 2             A   A            A          A             A       A
A
## 3             B   B            B          B             A       B
A
## 4             A   A            A          A             A       A
A
## 5             A   A            A          A             A       A
A
## 6             E   E            E          E             E       E
C
## 7             D   D            D          D             D       D
D
## 8             B   B            B          B             B       B
B
## 9             A   A            A          A             A       A
A
## 10            A   A            A          A             A       A
A
## 11            B   B            B          B             B       B
B
## 12            C   C            C          C             C       C
C
## 13            B   B            B          B             B       B
B
## 14            A   A            A          A             A       A
A
## 15            E   E            E          E             E       E
E
## 16            E   E            E          E             E       E
E
## 17            A   A            A          A             A       A
A
## 18            B   B            B          B             B       B
B
## 19            B   B            B          B             B       B
B
## 20            B   B            B          B             B       B
B
```

```
# Calculating testing accuracies
model.summary$testing.accuracy <- c(
  sum(equals(test.predictions$ground.truth, test.predictions$all)) / nrow(test.pre
dictions) * 100,
  sum(equals(test.predictions$ground.truth, test.predictions$all.with.pca)) / nrow
(test.predictions) * 100,
  sum(equals(test.predictions$ground.truth, test.predictions$all.with.pca)) / nrow
(test.predictions) * 100,
  sum(equals(test.predictions$ground.truth, test.predictions$correlated)) / nrow(t
est.predictions) * 100,
  sum(equals(test.predictions$ground.truth, test.predictions$corr.with.pca)) / nro
w(test.predictions) * 100,
  sum(equals(test.predictions$ground.truth, test.predictions$reduced.with.pca)) /
nrow(test.predictions) * 100
)
```

Plotting the final testing accuracy for 20 observations.

```
# Testing accuracy chart
gg <- ggbarplot(
  model.summary, x = "model", y = "testing.accuracy",
  title = "Testing accuracy (%)",               # Main title
  fill = "model",                               # Color by groups
  sort.val = "desc",                            # Sort value in descending order
  label = model.summary$testing.accuracy,       # Label values
  lab.pos = "in",                               # Position for labels
  lab.col = "white",                            # Color for labels
  lab.nb.digits = 2,                            # Number of decimal places (round
)
  xlab = FALSE,                                 # X-axis labels
  ylab = FALSE,                                 # Y-axis labels
  rotate = FALSE,                               # Rotate vertically
  font.label = list(color = "white", size = 11, vjust = 1.5),
  ggtheme = theme_pubr()                        # ggplot2 theme
)
ggpar(gg, legend = "none", legend.title = "")    # Customize legend
```

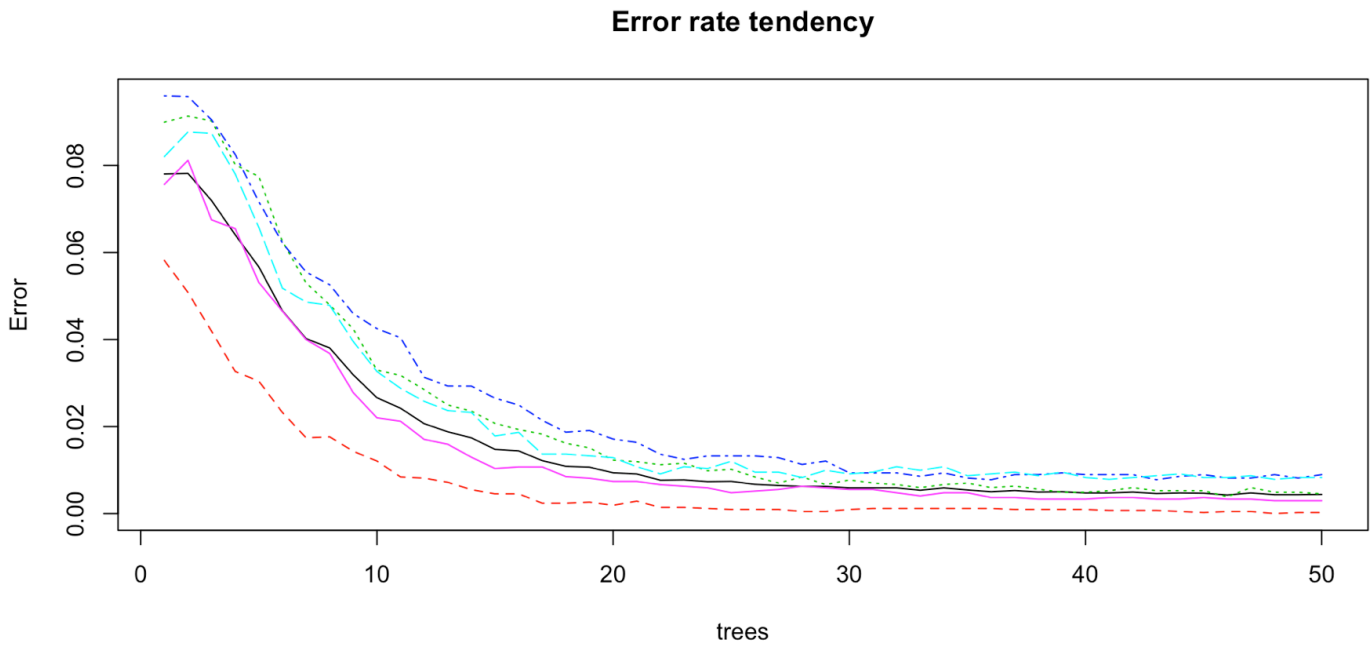


Conclusion

Using all features to train the model give us an appropriate cross-validating and testing accuracy. However the reduced features provide more validating accuracy.

Plotting the error rate tendency of the first trained model.

```
plot(model.all.features, main = "Error rate tendency")
```



The error rate doesn’t decline a lot after 50 trees.