Fittness Regime Model

Bonolo Molopyane

2023-02-01

Introduction

This project aims to predict the class of fitbit used given the manner of readings imputed, i.e the possition a participent was in when performing a certain exercise. We will classify them for the purposes of this exercise as

A - Sitting B - Sitting down C - Standing D - Standing up E - Walking

The datasets for this project are available in the Github repository: https://github.com/Bonolo114/Fitbit-Project and can be downloaded through the following steps. Credit is given at this stage to the compilers of the dataset and more information can be obtained at http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

Well you"ll use your locataion right!

We will use the train data set to train and test the model and predict the classes on the test set

Before we go any further we then load all the libraries we will make use of. We will make use of the randomForest algorithm to train the data because one, its a powerful and second its very easy to implement and interpret

```
library(caret)
```

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

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

```
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
```

```
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:dplyr':
##
##     combine
```

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

A quick glance of the data via printing the data or running View(train) shows there are a lot of empty and NA values that will make running the algorithm difficult.

```
#You may view the the data by removing the # below
#str(train)
```

This is the critical phase in our assignment as the predictive power of the algorithm rests with the worth of the variables(predictors) used.

Lets first look at the columns that have NA's in them

Data Cleaning

```
entries <-train %>%
  summarise_all(funs(sum(as.numeric(!is.na(.)), na.rm = TRUE))) %>%
  collect()
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## i Please use a list of either functions or lambdas:
##
## # Simple named list: list(mean = mean, median = median)
##
## # Auto named with `tibble::lst()`: tibble::lst(mean, median)
##
## # Using lambdas list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

the results reveal an interesting observation that it is either there column has 19622 entries which means the rows of these columns are entirely populated and other columns have only 406 entries

taking a closer look at the columns shows this is true for all columns i.e

```
complete_entries <- grep("19622", entries)
incomplete_entries <- grep("406",entries)
length(sort(c(complete_entries,incomplete_entries)))
```

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

Thus it each row falls in one of these to categories. we checked the length of the columns just for extra precautions and indeed theres 160 entries.

Considering that we have a lot of variables and the data containing 406 values have just 2% of that column populated so we then will agree so as to approach parsimonity remove these columns

```
train_subset_1 <- train[,complete_entries]
```

Taking another look at this new data set we see that the empty spaces are still there.

There are probably better ways of removing them but I just took a fine tooth comb and craped through and found these columns then concartinated them

```
em_cols <- c(12:20,43:48,52:60,74:82)
```

We then create a dataset without these columns

```
train_subset_2 <- train_subset_1[,-em_cols]
```

Finally we realise that the first 5 rows cannot be used for our analysis as they are mainly identifiers and time stamps, we concartinate them and use them to build a data set that can be properly used for analysis

```
redundent<- c(1:5)

new_train <- train_subset_2[,-redundent]
```

Laslty we will we will just convert the new_window and classe variable into factor variables and we have our tidy data set ready for analysis

```
new_train$new_window <- as.factor(new_train$new_window)
new_train$classe     <- as.factor(new_train$classe)
```

The next step is to split up the dataset into two for training and testing the model. We will set a seed to prevent the indicators from changing every time we run the code lines and make sure the project is reproducible

```
set.seed(2242)
inTrain <- createDataPartition(y = new_train$classe, p= 0.85, list = FALSE)
training <- new_train[inTrain,]
testing  <- new_train[-inTrain,]
```

Seems like everything is ready for incorporation. Let us now fit in the model

The Model

```
model <- randomForest(classe~.,data = training, importance = TRUE)
```

Before we predict the values of unknown classes lets see how it performs for the the data we already have, i.e the training and testing set

```
trainingpredictions <- predict(model,training[,-55])
confusionMatrix(trainingpredictions,training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 4743    0    0    0    0
##          B    0 3228    0    0    0
##          C    0    0 2909    0    0
##          D    0    0    0 2734    0
##          E    0    0    0    0 3066
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

```
testingpredictions <- predict(model,testing[,-55])
confusionMatrix(testingpredictions,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 837   3   0   0   0
##          B   0 566   3   0   0
##          C   0   0 510   2   0
```

```
##          D   0   0   0 480   1
##          E   0   0   0   0 540
##
## Overall Statistics
##
##                Accuracy : 0.9969
##                  95% CI : (0.9942, 0.9986)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9961
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9947   0.9942   0.9959   0.9982
## Specificity            0.9986   0.9987   0.9992   0.9996   1.0000
## Pos Pred Value         0.9964   0.9947   0.9961   0.9979   1.0000
## Neg Pred Value         1.0000   0.9987   0.9988   0.9992   0.9996
## Prevalence             0.2845   0.1934   0.1744   0.1638   0.1839
## Detection Rate         0.2845   0.1924   0.1734   0.1632   0.1835
## Detection Prevalence   0.2855   0.1934   0.1740   0.1635   0.1835
## Balanced Accuracy      0.9993   0.9967   0.9967   0.9977   0.9991
```

We see that the predictors used deliver high accuracy thus we deem them sufficient for our purposes

A few more things. We could discuss whether the model as a whole is adequate but the results speak for themselves. What we can check further is how important or significant are the individual predictors we used in this model in improving the model

To do so we will just check the importance each predictor has on the model

```
importance<-data.frame(model$importance[,7])
importance <- arrange(importance,desc(model$importance[,7]))
sum <- sum(importance )
importance <- (importance/sum)*100
importance
```

```
##                      model.importance...7.
## num_window                      8.822337182
## roll_belt                       7.608760238
## yaw_belt                        5.110012519
## pitch_forearm                   4.546252221
## magnet_dumbbell_z               4.479026558
## magnet_dumbbell_y               4.304955364
## pitch_belt                      4.293086193
## roll_forearm                    3.502954591
## magnet_dumbbell_x               2.965731729
## accel_dumbbell_y                2.472696503
## roll_dumbbell                   2.397334971
## accel_belt_z                    2.371179033
## magnet_belt_z                   2.364711548
```

```
## magnet_belt_y                 2.171986307
## accel_dumbbell_z              2.151847912
## roll_arm                      1.918900731
## accel_forearm_x               1.896994423
## gyros_belt_z                  1.778055931
## total_accel_dumbbell          1.757281210
## magnet_forearm_z              1.635757948
## accel_dumbbell_x              1.577673883
## magnet_belt_x                 1.522407446
## total_accel_belt              1.482165030
## yaw_dumbbell                  1.474675694
## magnet_arm_x                  1.423014580
## accel_arm_x                   1.419811155
## accel_forearm_z               1.387431257
## gyros_dumbbell_y              1.347034478
## magnet_arm_y                  1.327245139
## yaw_arm                       1.320978170
## magnet_forearm_y              1.290105296
## magnet_forearm_x              1.252963446
## pitch_dumbbell                1.045675003
## yaw_forearm                   1.029591017
## pitch_arm                     0.992955929
## magnet_arm_z                  0.933945115
## accel_belt_y                  0.869643145
## accel_arm_y                   0.830034322
## accel_belt_x                  0.800732743
## accel_forearm_y               0.775481618
## accel_arm_z                   0.716438679
## gyros_dumbbell_x              0.703129144
## gyros_arm_y                   0.700070949
## gyros_forearm_y               0.679551162
## gyros_arm_x                   0.665175088
## gyros_belt_y                  0.659796438
## gyros_belt_x                  0.560478275
## total_accel_arm               0.553227164
## total_accel_forearm           0.547867937
## gyros_forearm_z               0.441816632
## gyros_dumbbell_z              0.408804601
## gyros_forearm_x               0.405148638
## gyros_arm_z                   0.302438311
## new_window                    0.002629403
```

```
sum(importance)
```

```
## [1] 100
```

So it seems that only the predictor new_window has the smallest predictive power and thus the rest of the predictors remain significant

Moving further towards finding a parsimonious model let us investigate what occurs when we consider the top 5 predictor variables

Analysis, Countering and challanging the Model

```
importance[1:5,]
```

```
## [1] 8.822337 7.608760 5.110013 4.546252 4.479027
```

```
model2 <- randomForest(classe~num_window+roll_belt+yaw_belt+magnet_dumbbell_z+pitch_forearm,
                       data = training, importance = TRUE)

testingpredictions2 <- predict(model2, testing[,-55])
confusionMatrix(testingpredictions2,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 837   1   0   0   0
##          B   0 568   3   0   1
##          C   0   0 510   0   0
##          D   0   0   0 482   1
##          E   0   0   0   0 539
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9956, 0.9993)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9982   0.9942   1.0000   0.9963
## Specificity            0.9995   0.9983   1.0000   0.9996   1.0000
## Pos Pred Value         0.9988   0.9930   1.0000   0.9979   1.0000
## Neg Pred Value         1.0000   0.9996   0.9988   1.0000   0.9992
## Prevalence             0.2845   0.1934   0.1744   0.1638   0.1839
## Detection Rate         0.2845   0.1931   0.1734   0.1638   0.1832
## Detection Prevalence   0.2848   0.1944   0.1734   0.1642   0.1832
## Balanced Accuracy      0.9998   0.9983   0.9971   0.9998   0.9982
```

When the testing set is placed under investigation and it is observed that the second model, i.e the one with only 5 predictors perform better than that of 54 predictors.

The accuracy of the first model is 0.9969 whilst that of the second is 0.998. though it may be a seemingly small difference of 0.0011 but considering the computational time and ease of use the second model is to be preferred.

It may be possible that the high performance attributed to the number of training windows implored by the participants. Let us extract this predictor and see how well it works on it own

It Suffices to say that the reduced predictors model outperforms the initial.

However it is possible that the training windows variable could have has a major impact on the model so if we look at the model predicted by using just this variable we have that:

```
model3 <- randomForest(classe~num_window, data = training, importance = TRUE)
testingpredictions3 <- predict(model3,testing[,-55])
confusionMatrix(testingpredictions3,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 837   0   0   0   0
##          B   0 568   2   0   0
##          C   0   0 511   0   0
##          D   0   0   0 482   0
##          E   0   1   0   0 541
##
## Overall Statistics
##
##                Accuracy : 0.999
##                  95% CI : (0.997, 0.9998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9987
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9982   0.9961   1.0000   1.0000
## Specificity            1.0000   0.9992   1.0000   1.0000   0.9996
## Pos Pred Value         1.0000   0.9965   1.0000   1.0000   0.9982
## Neg Pred Value         1.0000   0.9996   0.9992   1.0000   1.0000
## Prevalence             0.2845   0.1934   0.1744   0.1638   0.1839
## Detection Rate         0.2845   0.1931   0.1737   0.1638   0.1839
## Detection Prevalence   0.2845   0.1937   0.1737   0.1638   0.1842
## Balanced Accuracy      1.0000   0.9987   0.9981   1.0000   0.9998
```

This is astonishing, this model implies that the number of windows alone can predict which position the participant was in when excising. This is possibly the simplest model but somehow has and unsettling feel.

It beacons us to consider how the top 5 variables perform without the inclusion of the num_window variable

```
model4 <- randomForest(classe~roll_belt+yaw_belt+magnet_dumbbell_z+pitch_forearm, data = training, impo
testingpredictions4 <- predict(model4,testing[,-55])
confusionMatrix(testingpredictions4,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 805  12   3   2   4
##          B   7 517  15   1   7
##          C  13  25 466  20   3
##          D  11  15  28 459   7
##          E   1   0   1   0 520
##
## Overall Statistics
##
##                Accuracy : 0.9405
##                  95% CI : (0.9314, 0.9488)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9248
##
##  Mcnemar's Test P-Value : 1.08e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9618   0.9086   0.9084   0.9523   0.9612
## Specificity            0.9900   0.9874   0.9749   0.9752   0.9992
## Pos Pred Value         0.9746   0.9452   0.8843   0.8827   0.9962
## Neg Pred Value         0.9849   0.9783   0.9805   0.9905   0.9913
## Prevalence             0.2845   0.1934   0.1744   0.1638   0.1839
## Detection Rate         0.2736   0.1757   0.1584   0.1560   0.1768
## Detection Prevalence   0.2808   0.1859   0.1791   0.1768   0.1774
## Balanced Accuracy      0.9759   0.9480   0.9416   0.9637   0.9802
```

YET again high accuracy is obtained. Could it be that the training set is just too large given that it is over 5.6 times larger than the testing set?

Reducing the training data

What would happen if we reduced the training set to just 55% of the data.

Assuming that the accuracy patterns will follow that of the above models we will only run the first and forth models and code will be left for the reader to confirm if curious

```
set.seed(2242)
inTrain2 <- createDataPartition(y = new_train$classe, p= 0.55, list = FALSE)
training2 <- new_train[inTrain2,]
testing2  <- new_train[-inTrain2,]

model <- randomForest(classe~.,data = training2, importance = TRUE)
testingpredictions <- predict(model,testing2[,-55])
confusionMatrix(testingpredictions,testing2$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2510    8    0    0    0
##          B    0 1700    7    0    0
##          C    0    0 1529   14    0
##          D    0    0    3 1430    2
##          E    0    0    0    3 1621
##
## Overall Statistics
##
##                Accuracy : 0.9958
##                  95% CI : (0.9942, 0.997)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9947
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9953   0.9935   0.9883   0.9988
## Specificity            0.9987   0.9990   0.9981   0.9993   0.9996
## Pos Pred Value         0.9968   0.9959   0.9909   0.9965   0.9982
## Neg Pred Value         1.0000   0.9989   0.9986   0.9977   0.9997
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2844   0.1926   0.1732   0.1620   0.1836
## Detection Prevalence   0.2853   0.1934   0.1748   0.1626   0.1840
## Balanced Accuracy      0.9994   0.9972   0.9958   0.9938   0.9992
```

```
#model2b <- randomForest(classe~num_window+roll_belt+yaw_belt+magnet_dumbbell_z+pitch_forearm,
#                         data = training2, importance = TRUE)
#testingpredictions2b <- predict(model2b,testing2[,-55])
#confusionMatrix(testingpredictions2b,testing$classe)

#model3b <- randomForest(classe~num_window, data = training, importance = TRUE)
#testingpredictions3b <- predict(model3b,testing2[,-55])
#confusionMatrix(testingpredictions3b,testing2$classe)

model4b <- randomForest(classe~roll_belt+yaw_belt+magnet_dumbbell_z+pitch_forearm,
                        data = training, importance = TRUE)
testingpredictions4b <- predict(model4b,testing2[,-55])
confusionMatrix(testingpredictions4b,testing2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2465   12    1    5    2
##          B    7 1677   32    0    2
##          C   13   16 1453   11    1
##          D   24    3   52 1431    7
```

```
##          E    1    0    1    0 1611
##
## Overall Statistics
##
##                Accuracy : 0.9785
##                  95% CI : (0.9752, 0.9814)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9728
##
##  Mcnemar's Test P-Value : 9.025e-11
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9821   0.9819   0.9441   0.9889   0.9926
## Specificity            0.9968   0.9942   0.9944   0.9883   0.9997
## Pos Pred Value         0.9920   0.9761   0.9726   0.9433   0.9988
## Neg Pred Value         0.9929   0.9956   0.9883   0.9978   0.9983
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2793   0.1900   0.1646   0.1621   0.1825
## Detection Prevalence   0.2815   0.1946   0.1693   0.1719   0.1827
## Balanced Accuracy      0.9895   0.9880   0.9692   0.9886   0.9962
```

There is no denying the predictive power of the random forest algorithm even with a reduced dataset high accuracy is still maintained in the test dataset

Just for peace of mind, if we ran a glm how many of the variables will be deemed significant, (this might not a good barometer for our analysis especially at this stage) but would it reveal anything that would contradict our attained observations thus far?

Logistic Regression: Can they shed any light?

```
logistic <- glm(classe~.-1,data= training, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logistic)
```

```
##
## Call:
## glm(formula = classe ~ . - 1, family = "binomial", data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.9445  -0.1190   0.0744   0.2957   4.0973
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## new_windowno          5.249e+01  3.864e+00  13.585  < 2e-16 ***
```

```
## new_windowyes         5.244e+01  3.864e+00  13.571  < 2e-16 ***
## num_window            1.924e-03  1.525e-04  12.612  < 2e-16 ***
## roll_belt             2.373e-01  1.420e-02  16.715  < 2e-16 ***
## pitch_belt           -3.078e-01  2.196e-02 -14.017  < 2e-16 ***
## yaw_belt             -1.965e-01  8.519e-03 -23.062  < 2e-16 ***
## total_accel_belt     -5.292e-02  3.710e-02  -1.426 0.153731
## gyros_belt_x          1.911e+00  3.718e-01   5.140 2.75e-07 ***
## gyros_belt_y         -1.490e+00  1.095e+00  -1.360 0.173853
## gyros_belt_z          6.720e-01  2.708e-01   2.481 0.013093 *
## accel_belt_x         -1.223e-03  6.825e-03  -0.179 0.857769
## accel_belt_y         -3.558e-02  7.539e-03  -4.720 2.36e-06 ***
## accel_belt_z         -2.931e-02  6.130e-03  -4.782 1.74e-06 ***
## magnet_belt_x        -1.749e-02  3.018e-03  -5.795 6.82e-09 ***
## magnet_belt_y        -8.193e-02  5.092e-03 -16.089  < 2e-16 ***
## magnet_belt_z         6.559e-02  3.012e-03  21.778  < 2e-16 ***
## roll_arm              4.003e-03  6.200e-04   6.458 1.06e-10 ***
## pitch_arm            -7.484e-03  1.292e-03  -5.793 6.91e-09 ***
## yaw_arm               4.729e-03  5.212e-04   9.072  < 2e-16 ***
## total_accel_arm      -9.553e-03  4.216e-03  -2.266 0.023462 *
## gyros_arm_x           1.568e-01  5.080e-02   3.086 0.002026 **
## gyros_arm_y          -4.241e-02  1.279e-01  -0.332 0.740155
## gyros_arm_z          -1.013e-01  9.674e-02  -1.048 0.294866
## accel_arm_x          -1.154e-02  1.048e-03 -11.017  < 2e-16 ***
## accel_arm_y          -5.523e-03  2.228e-03  -2.479 0.013173 *
## accel_arm_z           2.414e-02  1.332e-03  18.120  < 2e-16 ***
## magnet_arm_x          4.849e-04  3.488e-04   1.390 0.164462
## magnet_arm_y         -1.199e-03  9.140e-04  -1.312 0.189652
## magnet_arm_z         -1.226e-02  5.645e-04 -21.721  < 2e-16 ***
## roll_dumbbell         4.762e-03  9.149e-04   5.205 1.94e-07 ***
## pitch_dumbbell       -1.712e-02  1.864e-03  -9.184  < 2e-16 ***
## yaw_dumbbell         -2.289e-02  9.449e-04 -24.227  < 2e-16 ***
## total_accel_dumbbell  2.047e-01  1.439e-02  14.225  < 2e-16 ***
## gyros_dumbbell_x      1.038e+00  1.318e-01   7.872 3.48e-15 ***
## gyros_dumbbell_y      7.128e-01  9.508e-02   7.496 6.56e-14 ***
## gyros_dumbbell_z      4.892e-01  9.166e-02   5.337 9.44e-08 ***
## accel_dumbbell_x      3.883e-02  2.572e-03  15.097  < 2e-16 ***
## accel_dumbbell_y      5.394e-03  2.004e-03   2.691 0.007127 **
## accel_dumbbell_z      4.593e-03  1.247e-03   3.683 0.000230 ***
## magnet_dumbbell_x    -8.605e-03  5.801e-04 -14.832  < 2e-16 ***
## magnet_dumbbell_y    -6.823e-03  4.351e-04 -15.679  < 2e-16 ***
## magnet_dumbbell_z     3.697e-02  9.210e-04  40.138  < 2e-16 ***
## roll_forearm          2.903e-03  3.870e-04   7.503 6.23e-14 ***
## pitch_forearm         2.465e-02  2.048e-03  12.039  < 2e-16 ***
## yaw_forearm          -1.916e-03  4.431e-04  -4.325 1.53e-05 ***
## total_accel_forearm   6.959e-02  5.231e-03  13.303  < 2e-16 ***
## gyros_forearm_x       3.245e-01  9.193e-02   3.530 0.000416 ***
## gyros_forearm_y       4.441e-02  2.793e-02   1.590 0.111780
## gyros_forearm_z       4.814e-02  7.997e-02   0.602 0.547200
## accel_forearm_x       4.159e-03  8.144e-04   5.107 3.27e-07 ***
## accel_forearm_y       2.088e-03  5.478e-04   3.812 0.000138 ***
## accel_forearm_z      -1.323e-02  6.343e-04 -20.865  < 2e-16 ***
## magnet_forearm_x     -3.441e-03  3.539e-04  -9.724  < 2e-16 ***
## magnet_forearm_y     -1.707e-03  2.333e-04  -7.317 2.54e-13 ***
## magnet_forearm_z      8.342e-05  2.027e-04   0.411 0.680758
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23123  on 16680  degrees of freedom
## Residual deviance:  7505  on 16625  degrees of freedom
## AIC: 7615
##
## Number of Fisher Scoring iterations: 8
```

```
logistic2 <- glm(classe~roll_belt+yaw_belt+magnet_dumbbell_z+pitch_forearm, data = training, family = "
summary(logistic2)
```

```
##
## Call:
## glm(formula = classe ~ roll_belt + yaw_belt + magnet_dumbbell_z +
##     pitch_forearm, family = "binomial", data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7918  -0.5190   0.4955   0.7780   1.9545
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -0.2174610  0.0545857  -3.984 6.78e-05 ***
## roll_belt          0.0096571  0.0006955  13.886  < 2e-16 ***
## yaw_belt          -0.0035745  0.0003811  -9.380  < 2e-16 ***
## magnet_dumbbell_z  0.0064995  0.0002128  30.539  < 2e-16 ***
## pitch_forearm      0.0377216  0.0008547  44.136  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 19916  on 16679  degrees of freedom
## Residual deviance: 15994  on 16675  degrees of freedom
## AIC: 16004
##
## Number of Fisher Scoring iterations: 5
```

Though some of the variables are seen as not significant according to the first logistic regression, these will likely align with the variables that have a lower importance ranking in the random forest model, but the second regression states that all the predictors are significant

The complete logistic regression shows that the following predictors are not significantly different from zero (magnet_forearm_z, gyros_forearm_y@, gyros_forearm_z@, magnet_arm_x, magnet_arm_y, gyros_arm_y@, gyros_arm_z@,accel_belt_x@, gyros_belt_y@, total_accel_belt ,new_window@)

of which 7 of the 11 variables are in the bottom 20 in the importance tables. Although a little can be deduced from this finding but we can be confident in the importance and significance of the predictors

Final Prediction: Predicting on new dataset

Skeptical as we have been comfort can be accepted when using the random forest, and this very skepticism move us towards using the forth model using 55% of the training data purely because it moves us a bit out of falling into a saturated (overfitted) model. Therefore

Now for the pudding, lets see how it performs for new data. Bus since we reduced the predictors for training we should do the same so that we use 54 predictors as opposed to 159. And that can be simply done as follows

```
chosen_columns <- names(training[,-55])

validation_set <- test[,chosen_columns]
validation_set$new_window<- as.factor(validation_set$new_window)
```

Now you would realize that if you try predict using the validation set as is you get an error stating that the predictors found in this data is different from the one's used in the model.

A variety of reasons why this is the case may be mentioned, I am still trying to figure it out myself.

However the work still needs to get done. And for that we implement a nifty trick of adding and removing the first row of the training set to this validation set

```
validation_set <- rbind(training[1,-55], validation_set)
validation_set <- validation_set[-1,]
```

Finally we can now predict on new data set after a few adjustments

```
validationpredictionfinal <- predict(model4b,validation_set)
validationpredictionfinal
```

```
##  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

Conclusion

The random forest is a powerful predictive algorithm used in many setting including for our purpose determining which position(class) a participant holds during their exercise regime.

Further studies can be incorporated to determine how much impact is each class in the over-all effectiveness of their exercise program but this is beyond our current scope. All that is needed is to find a way to have our accelerometer predict a participant's class and the random forest algorithm sufficiently provides that information.