Exercise prediction from generated training data using a suitable algorithm.

Bola Lawal

2022-05-06

Executive Summary

The goal of the project is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants as the data for creating a prediction algorithm. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. All the information is available from Data

Background

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.

Install the packages silently

Load the libraries

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, first, last

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

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

## Loading required package: lattice
```

Download all needed data

```
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv","training.csv",meth
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv","testing.csv",metho

training_set <- tibble::as_tibble(fread("training.csv",na.strings=c('#DIV/0!', '', 'NA')))
testing_set  <- tibble::as_tibble(fread("testing.csv",na.strings=c('#DIV/0!', '', 'NA')))
```

Analyze the Data

Testing Set

```
## Warning: Warning: fonts used in ‘flextable‘ are ignored because the ‘pdflatex‘
## engine is used and not ‘xelatex‘ or ‘lualatex‘. You can avoid this warning
## by using the ‘set_flextable_defaults(fonts_ignore=TRUE)‘ command or use a
## compatible engine by defining ‘latex_engine: xelatex‘ in the YAML header of the
## R Markdown document.
```

Observations	Variables
20	160

Training Set

```
## Warning: Warning: fonts used in ‘flextable‘ are ignored because the ‘pdflatex‘
## engine is used and not ‘xelatex‘ or ‘lualatex‘. You can avoid this warning
## by using the ‘set_flextable_defaults(fonts_ignore=TRUE)‘ command or use a
## compatible engine by defining ‘latex_engine: xelatex‘ in the YAML header of the
## R Markdown document.
```

Observations	Variables
19,622	160

Split the Data

```
set.seed(171)
training_sub_set <- createDataPartition( y = training_set$classe,p = 0.7,list = FALSE)
real_training <- training_set[training_sub_set,]
real_validation <- training_set[-training_sub_set,]
```

Pre-process the data

Remove variables with mostly N/A values

```
NA_vals <- sapply(real_training,function(x) mean(is.na(x))) > 0.95
real_training <- real_training[,NA_vals==FALSE]
real_validation <- real_validation[,NA_vals==FALSE]
```

Remove variables with low variance

```
nzv <- nearZeroVar(real_training) #Using the training across both datasets as there has to be conformit
real_training <- real_training[,-nzv]
real_validation <- real_validation[,-nzv]
```

Remove columns that will have no bearing on the results (V1, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp)

```
real_training_clean = select(real_training, -V1, -user_name, -raw_timestamp_part_1, -raw_timestamp_part
real_validation_clean = select(real_validation, -V1, -user_name, -raw_timestamp_part_1, -raw_timestamp_p
```

Testing Set

```
exit_values_3 <- tibble(
  "Observations" = nrow(real_validation_clean),
  "Variables" = ncol(real_validation_clean))
flextable(exit_values_3)
```

```
## Warning: Warning: fonts used in ‘flextable‘ are ignored because the ‘pdflatex‘
## engine is used and not ‘xelatex‘ or ‘lualatex‘. You can avoid this warning
## by using the ‘set_flextable_defaults(fonts_ignore=TRUE)‘ command or use a
## compatible engine by defining ‘latex_engine: xelatex‘ in the YAML header of the
## R Markdown document.
```

Observations	Variables
5,885	54

Training Set

```
exit_values_4 <- tibble(
  "Observations" = nrow(real_training_clean),
  "Variables" = ncol(real_training_clean))
flextable(exit_values_4)
```

```
## Warning: Warning: fonts used in ‘flextable‘ are ignored because the ‘pdflatex‘
## engine is used and not ‘xelatex‘ or ‘lualatex‘. You can avoid this warning
## by using the ‘set_flextable_defaults(fonts_ignore=TRUE)‘ command or use a
## compatible engine by defining ‘latex_engine: xelatex‘ in the YAML header of the
## R Markdown document.
```

Observations	Variables
13,737	54

Create validation partitions of sample observations from the Training set

```
control <- trainControl(method="cv",number = 10)
```

Algorithm Performance Check

Gradient Boosting Model

```
set.seed(171)
model_GBM <- train(classe ~.,
                   data = real_training_clean,
                   method = "gbm",
                   trControl = control,
                   verbose = FALSE)
```

Random Forest

```
set.seed(171)
model_RF  <- train(classe ~.,
                    data = real_training_clean,
                    method = "rf",
                    trControl = control)
```

k-Nearest Neighbors

```
set.seed(171)
model_KNN <- train(classe~.,
                    data=real_training_clean,
                    method="knn",
                    metric="Accuracy",
                    trControl=control)
```

Linear Discriminant Analysis

```
set.seed(171)
model_LDA <- train(classe~.,
                    data=real_training_clean,
                    method="lda",
                    metric="Accuracy",
                    trControl=control)
```

Checking performance for all the models

```
##
```

```
## Call:
## summary.resamples(object = model_results)
##
## Models: lda, knn, gbm, rf
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.6935953 0.7075528 0.7135049 0.7131097 0.7212518 0.7248908    0
## knn 0.8922853 0.9042940 0.9071710 0.9071840 0.9128768 0.9148472    0
## gbm 0.9796215 0.9858031 0.9872635 0.9868972 0.9883530 0.9927220    0
## rf  0.9949017 0.9963570 0.9981800 0.9975973 0.9990902 1.0000000    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.6119084 0.6301362 0.6376725 0.6370554 0.6470303 0.6520158    0
## knn 0.8637537 0.8789251 0.8826174 0.8825893 0.8897999 0.8922193    0
## gbm 0.9742337 0.9820388 0.9838906 0.9834265 0.9852670 0.9907950    0
## rf  0.9935500 0.9953916 0.9976981 0.9969606 0.9988492 1.0000000    0
```

Check performance using training dataset

Gradient Boosting Model

```
GBM_Pred <- predict(model_GBM, newdata=real_validation_clean)
conf_Mat_GBM <- confusionMatrix(GBM_Pred, as.factor(real_validation_clean$classe))
print(conf_Mat_GBM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1670   10    0    2    2
##          B    4 1112    9    2    6
##          C    0   16 1017   10    5
##          D    0    1    0  944   10
##          E    0    0    0    6 1059
##
## Overall Statistics
##
##                Accuracy : 0.9859
##                  95% CI : (0.9825, 0.9888)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9822
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
```

```
## Sensitivity            0.9976   0.9763   0.9912   0.9793   0.9787
## Specificity            0.9967   0.9956   0.9936   0.9978   0.9988
## Pos Pred Value         0.9917   0.9815   0.9704   0.9885   0.9944
## Neg Pred Value         0.9990   0.9943   0.9981   0.9959   0.9952
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1890   0.1728   0.1604   0.1799
## Detection Prevalence   0.2862   0.1925   0.1781   0.1623   0.1810
## Balanced Accuracy      0.9971   0.9859   0.9924   0.9885   0.9887
```

Random Forest

```
RF_Pred <- predict(model_RF, newdata=real_validation_clean)
conf_Mat_RF <- confusionMatrix(RF_Pred, as.factor(real_validation_clean$classe))
print(conf_Mat_RF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    2    0    0    0
##          B    0 1133    3    0    0
##          C    0    4 1023    5    0
##          D    0    0    0  959    5
##          E    0    0    0    0 1077
##
## Overall Statistics
##
##                Accuracy : 0.9968
##                  95% CI : (0.995, 0.9981)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9959
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9947   0.9971   0.9948   0.9954
## Specificity            0.9995   0.9994   0.9981   0.9990   1.0000
## Pos Pred Value         0.9988   0.9974   0.9913   0.9948   1.0000
## Neg Pred Value         1.0000   0.9987   0.9994   0.9990   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1925   0.1738   0.1630   0.1830
## Detection Prevalence   0.2848   0.1930   0.1754   0.1638   0.1830
## Balanced Accuracy      0.9998   0.9971   0.9976   0.9969   0.9977
```

k-Nearest Neighbors

```
LDA_Pred <- predict(model_LDA, newdata=real_validation_clean)
conf_Mat_LDA <- confusionMatrix(LDA_Pred, as.factor(real_validation_clean$classe))
print(conf_Mat_LDA)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1383  157   92   40   50
##          B   47  719   95   42  149
##          C  113  170  677  142   86
##          D  126   40  127  713  113
##          E    5   53   35   27  684
##
## Overall Statistics
##
##                Accuracy : 0.7096
##                  95% CI : (0.6978, 0.7212)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6327
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8262   0.6313   0.6598   0.7396   0.6322
## Specificity            0.9195   0.9298   0.8948   0.9175   0.9750
## Pos Pred Value         0.8031   0.6835   0.5699   0.6372   0.8507
## Neg Pred Value         0.9301   0.9131   0.9257   0.9473   0.9217
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2350   0.1222   0.1150   0.1212   0.1162
## Detection Prevalence   0.2926   0.1788   0.2019   0.1901   0.1366
## Balanced Accuracy      0.8728   0.7805   0.7773   0.8286   0.8036
```

Linear Discriminant Analysis

```
KNN_Pred <- predict(model_KNN, newdata=real_validation_clean)
conf_Mat_KNN <- confusionMatrix(KNN_Pred, as.factor(real_validation_clean$classe))
print(conf_Mat_KNN)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1599   54   11   11   14
##          B   25  991   30    5   24
##          C   18   39  959   75   15
##          D   25   25   19  854   36
##          E    7   30    7   19  993
##
## Overall Statistics
##
##                Accuracy : 0.9169
##                  95% CI : (0.9096, 0.9238)
```

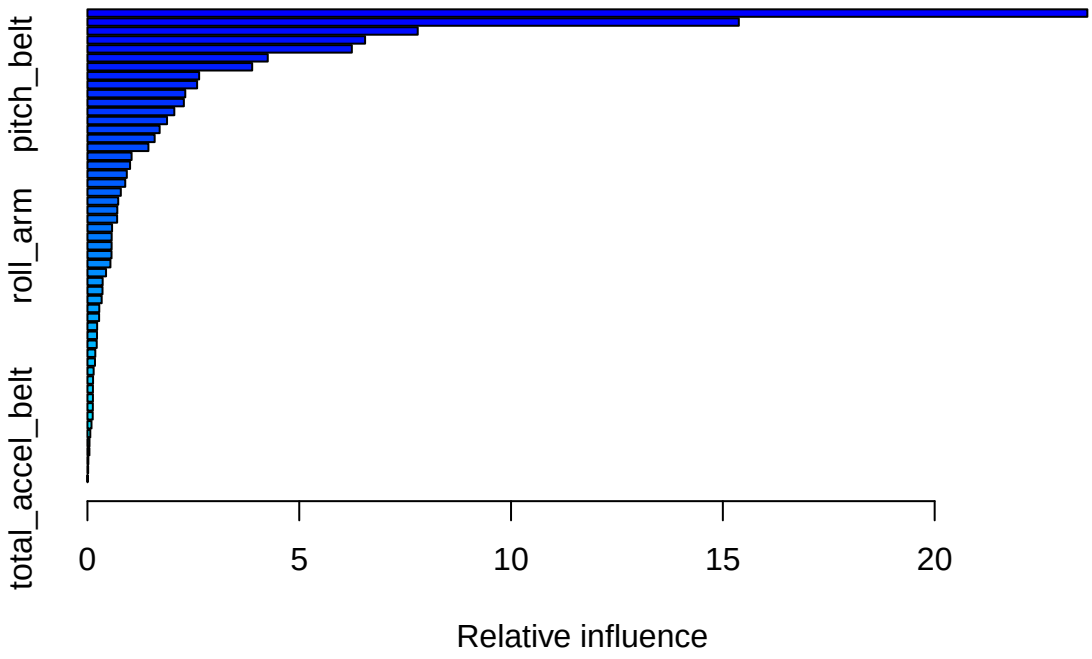
```
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8949
##
##  Mcnemar's Test P-Value : 2.11e-12
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9552   0.8701   0.9347   0.8859   0.9177
## Specificity            0.9786   0.9823   0.9697   0.9787   0.9869
## Pos Pred Value         0.9467   0.9219   0.8671   0.8905   0.9403
## Neg Pred Value         0.9821   0.9692   0.9860   0.9777   0.9816
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2717   0.1684   0.1630   0.1451   0.1687
## Detection Prevalence   0.2870   0.1827   0.1879   0.1630   0.1794
## Balanced Accuracy      0.9669   0.9262   0.9522   0.9323   0.9523
```

Summary of the performance

```
## a flextable object.
## col_keys: `Linear Discrimination Analysis`, `K- Nearest Neighbors`, `Gradient Boosting`, `Random Fore
## header has 1 row(s)
## body has 7 row(s)
## original dataset sample:
##   Linear Discrimination Analysis K- Nearest Neighbors Gradient Boosting
## 1                         0.9859               0.9968           0.7096
## 2                         0.9822               0.9959           0.6327
## 3                         0.9825               0.9950           0.6978
## 4                         0.9888               0.9981           0.7212
## 5                         0.2845               0.2845           0.2845
##   Random Forest
## 1        0.9169
## 2        0.8949
## 3        0.9096
## 4        0.9238
## 5        0.2845
```

Check the variables with the most influence

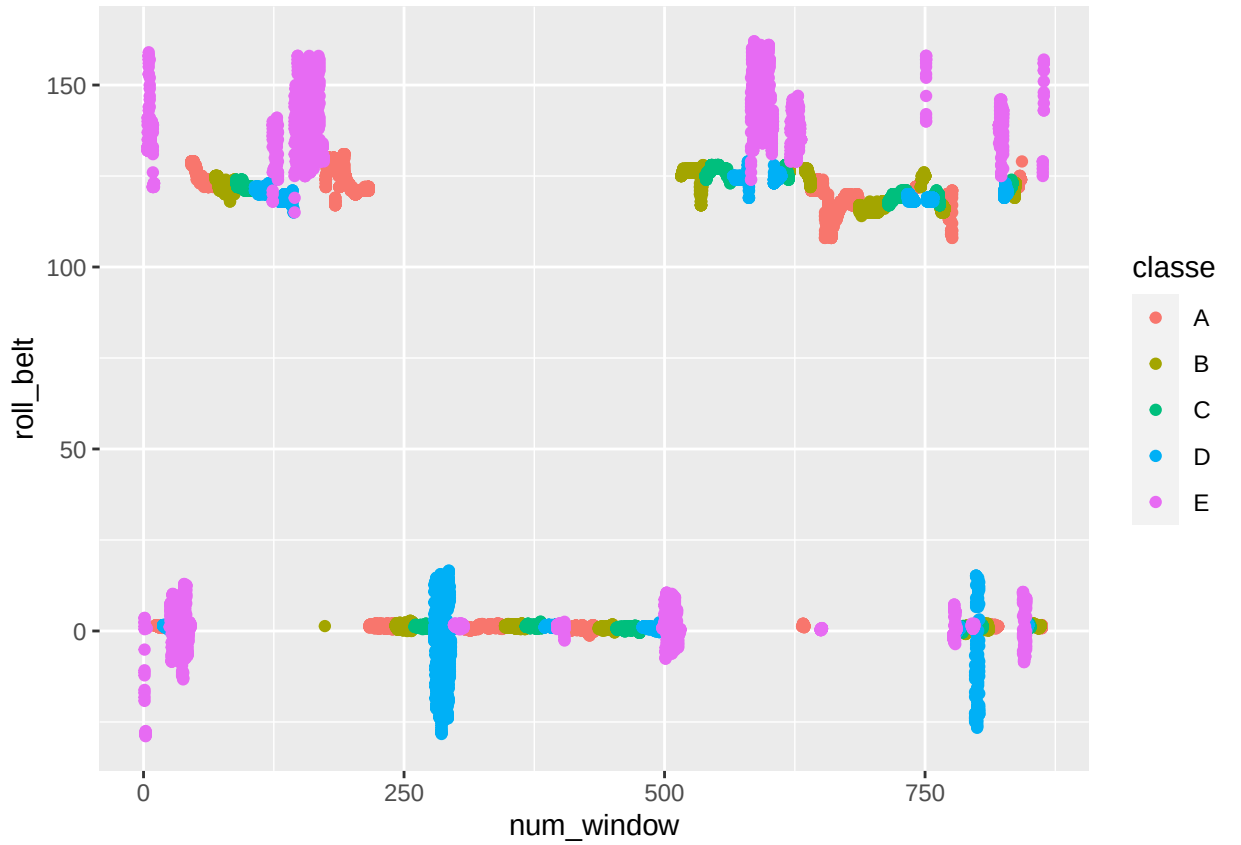
```
print(summary(model_GBM))
```



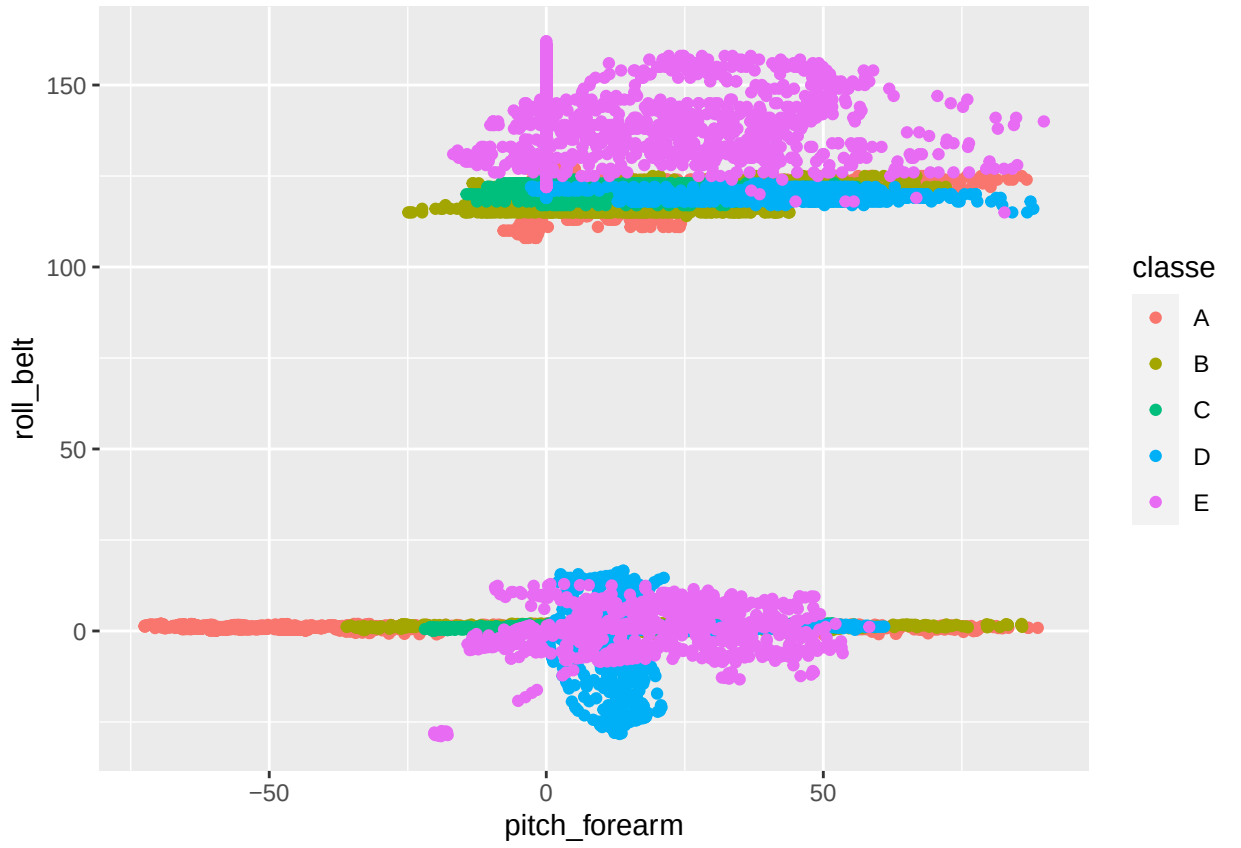

```
##                                   var     rel.inf
## num_window                 num_window 23.60457596
## roll_belt                   roll_belt 15.37667752
## pitch_forearm           pitch_forearm  7.79725822
## magnet_dumbbell_z     magnet_dumbbell_z  6.55453963
## yaw_belt                     yaw_belt  6.24130906
## magnet_dumbbell_y     magnet_dumbbell_y  4.25651142
## roll_forearm               roll_forearm  3.88933126
## magnet_belt_z             magnet_belt_z  2.63708037
## pitch_belt                   pitch_belt  2.58928797
## accel_dumbbell_z       accel_dumbbell_z  2.30972722
## accel_forearm_x         accel_forearm_x  2.27648673
## roll_dumbbell             roll_dumbbell  2.05114909
## gyros_belt_z               gyros_belt_z  1.87918311
## gyros_dumbbell_y       gyros_dumbbell_y  1.70475563
## magnet_forearm_z       magnet_forearm_z  1.58707056
## accel_dumbbell_y       accel_dumbbell_y  1.43961658
## accel_dumbbell_x       accel_dumbbell_x  1.04259949
## accel_forearm_z         accel_forearm_z  1.00636585
## magnet_belt_x             magnet_belt_x  0.92977184
## yaw_arm                         yaw_arm  0.89262990
## magnet_arm_z               magnet_arm_z  0.78811976
## accel_belt_z               accel_belt_z  0.72685571
## gyros_arm_y                 gyros_arm_y  0.70509937
## magnet_dumbbell_x     magnet_dumbbell_x  0.70151754
## magnet_arm_y               magnet_arm_y  0.58189044
```

```
## magnet_forearm_x         magnet_forearm_x  0.57222460
## roll_arm                       roll_arm  0.57097279
## total_accel_dumbbell total_accel_dumbbell  0.56832451
## magnet_belt_y               magnet_belt_y  0.54239728
## gyros_dumbbell_x         gyros_dumbbell_x  0.43967281
## magnet_arm_x                magnet_arm_x  0.35915268
## gyros_belt_y                gyros_belt_y  0.35558007
## accel_arm_x                  accel_arm_x  0.33631468
## accel_arm_z                  accel_arm_z  0.27932694
## gyros_arm_x                  gyros_arm_x  0.27549974
## pitch_dumbbell            pitch_dumbbell  0.22921203
## gyros_forearm_z           gyros_forearm_z  0.22621660
## gyros_dumbbell_z         gyros_dumbbell_z  0.21946912
## magnet_forearm_y         magnet_forearm_y  0.18564165
## accel_forearm_y           accel_forearm_y  0.17810277
## accel_arm_y                  accel_arm_y  0.14669319
## total_accel_arm           total_accel_arm  0.13301204
## yaw_dumbbell                 yaw_dumbbell  0.12993510
## total_accel_forearm   total_accel_forearm  0.12967444
## yaw_forearm                   yaw_forearm  0.12809036
## gyros_forearm_y           gyros_forearm_y  0.12440912
## pitch_arm                       pitch_arm  0.09547639
## gyros_belt_x                 gyros_belt_x  0.06752918
## accel_belt_y                 accel_belt_y  0.04842242
## accel_belt_x                 accel_belt_x  0.04651347
## gyros_forearm_x           gyros_forearm_x  0.02129621
## gyros_arm_z                  gyros_arm_z  0.01705104
## total_accel_belt         total_accel_belt  0.00437853
```

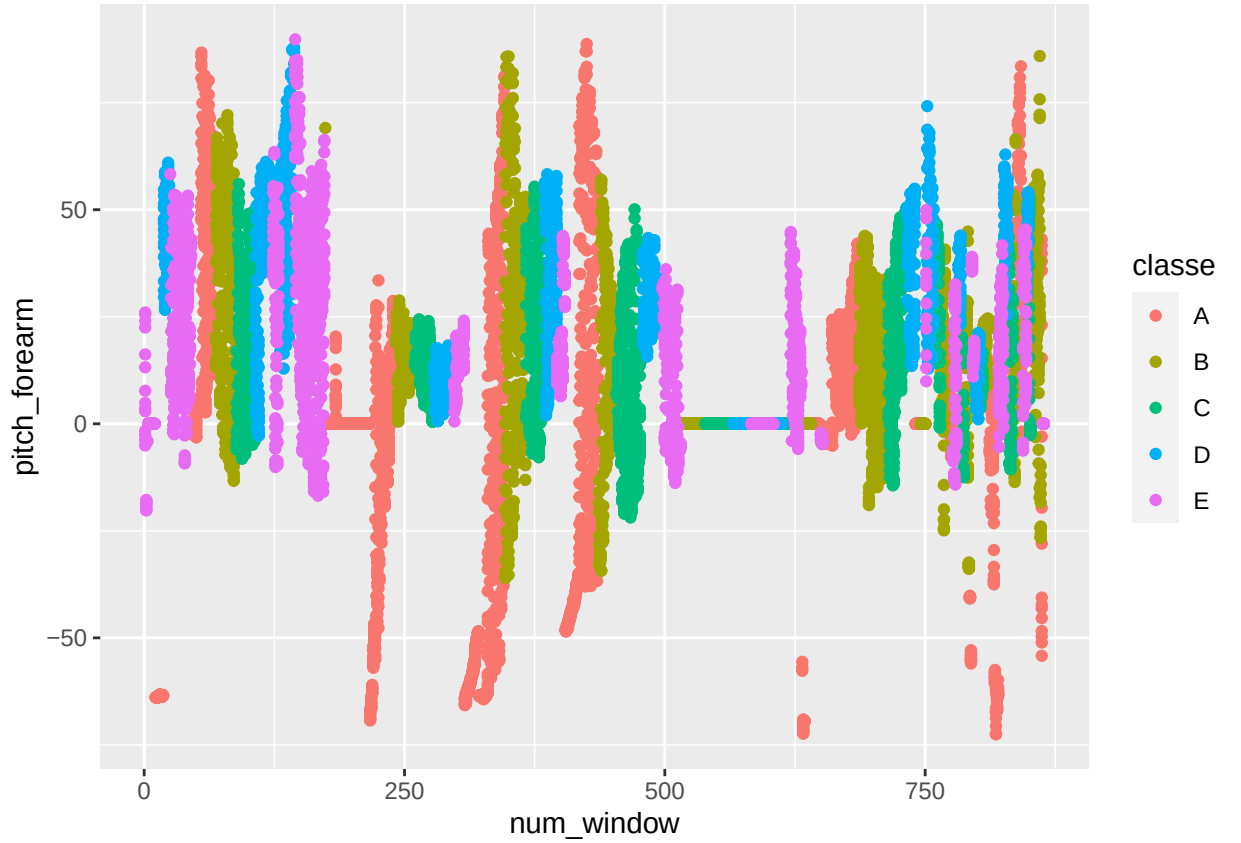
```
print(qplot(num_window, roll_belt, data = real_training, col = classe))
```




```
print(qplot(pitch_forearm, roll_belt, data = real_training, col = classe))
```




```
print(qplot(num_window, pitch_forearm, data = real_training, col = classe))
```



Prediction on the test dataset

```
model_prediction <- predict(model_RF, testing_set)
table(model_prediction,testing_set$problem_id)
```

```
##
## model_prediction 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
##                A 0 1 0 1 1 0 0 0 1  1  0  0  0  1  0  0  1  0  0  0
##                B 1 0 1 0 0 0 0 1 0  0  1  0  1  0  0  0  0  1  1  1
##                C 0 0 0 0 0 0 0 0 0  0  0  1  0  0  0  0  0  0  0  0
##                D 0 0 0 0 0 0 1 0 0  0  0  0  0  0  0  0  0  0  0  0
##                E 0 0 0 0 0 1 0 0 0  0  0  0  0  0  1  1  0  0  0  0
```

Summary

The best model based on the output in the model_results summary is the random forests model with an accuracy of 0.998. Due to the high number of trees, I believe that it is the best model to use.