Machine learning Project

Sabawoon Shafaq

11 September 2017

Introduction

Devices such as Jawbone Up, Nike FuelBand, and Fitbit makes it 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. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it.

In this project, the goal is to quantify how well the proposed activity is done. The data is collected from 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. Each participant performed one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: 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) and throwing the hips to the front (Class E). ONly class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes (http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)).

Loading and PreProcessing Data

The data collected from the devices is partially cleaned and can be downloaded from the link below.

```
## Warning: package 'caret' was built under R version 3.4.1
```

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

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

```
## Warning: package 'randomForest' was built under R version 3.4.1
```

```
## randomForest 4.6-12
```

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

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

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

A quick look at the dimension of the data can reveal on the number of variables and observations. Also, a quick look at the names of the variables and their column number will be useful in the analysis to come.

```
dim(training)
```

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

```
dim(testing)
```

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

```
names(training)
```

```
##   [1] "X"                       "user_name"
##   [3] "raw_timestamp_part_1"    "raw_timestamp_part_2"
##   [5] "cvtd_timestamp"          "new_window"
##   [7] "num_window"              "roll_belt"
##   [9] "pitch_belt"              "yaw_belt"
##  [11] "total_accel_belt"        "kurtosis_roll_belt"
##  [13] "kurtosis_picth_belt"     "kurtosis_yaw_belt"
##  [15] "skewness_roll_belt"      "skewness_roll_belt.1"
##  [17] "skewness_yaw_belt"       "max_roll_belt"
##  [19] "max_picth_belt"          "max_yaw_belt"
##  [21] "min_roll_belt"           "min_pitch_belt"
##  [23] "min_yaw_belt"            "amplitude_roll_belt"
##  [25] "amplitude_pitch_belt"    "amplitude_yaw_belt"
##  [27] "var_total_accel_belt"    "avg_roll_belt"
##  [29] "stddev_roll_belt"        "var_roll_belt"
##  [31] "avg_pitch_belt"          "stddev_pitch_belt"
##  [33] "var_pitch_belt"          "avg_yaw_belt"
##  [35] "stddev_yaw_belt"         "var_yaw_belt"
##  [37] "gyros_belt_x"            "gyros_belt_y"
##  [39] "gyros_belt_z"            "accel_belt_x"
##  [41] "accel_belt_y"            "accel_belt_z"
##  [43] "magnet_belt_x"           "magnet_belt_y"
##  [45] "magnet_belt_z"           "roll_arm"
##  [47] "pitch_arm"               "yaw_arm"
##  [49] "total_accel_arm"         "var_accel_arm"
##  [51] "avg_roll_arm"            "stddev_roll_arm"
##  [53] "var_roll_arm"            "avg_pitch_arm"
##  [55] "stddev_pitch_arm"        "var_pitch_arm"
##  [57] "avg_yaw_arm"             "stddev_yaw_arm"
##  [59] "var_yaw_arm"             "gyros_arm_x"
##  [61] "gyros_arm_y"             "gyros_arm_z"
##  [63] "accel_arm_x"             "accel_arm_y"
##  [65] "accel_arm_z"             "magnet_arm_x"
##  [67] "magnet_arm_y"            "magnet_arm_z"
##  [69] "kurtosis_roll_arm"       "kurtosis_picth_arm"
##  [71] "kurtosis_yaw_arm"        "skewness_roll_arm"
##  [73] "skewness_pitch_arm"      "skewness_yaw_arm"
##  [75] "max_roll_arm"            "max_picth_arm"
##  [77] "max_yaw_arm"             "min_roll_arm"
##  [79] "min_pitch_arm"           "min_yaw_arm"
##  [81] "amplitude_roll_arm"      "amplitude_pitch_arm"
##  [83] "amplitude_yaw_arm"       "roll_dumbbell"
##  [85] "pitch_dumbbell"          "yaw_dumbbell"
##  [87] "kurtosis_roll_dumbbell"  "kurtosis_picth_dumbbell"
##  [89] "kurtosis_yaw_dumbbell"   "skewness_roll_dumbbell"
##  [91] "skewness_pitch_dumbbell" "skewness_yaw_dumbbell"
##  [93] "max_roll_dumbbell"       "max_picth_dumbbell"
##  [95] "max_yaw_dumbbell"        "min_roll_dumbbell"
##  [97] "min_pitch_dumbbell"      "min_yaw_dumbbell"
##  [99] "amplitude_roll_dumbbell" "amplitude_pitch_dumbbell"
## [101] "amplitude_yaw_dumbbell"  "total_accel_dumbbell"
## [103] "var_accel_dumbbell"      "avg_roll_dumbbell"
## [105] "stddev_roll_dumbbell"    "var_roll_dumbbell"
## [107] "avg_pitch_dumbbell"      "stddev_pitch_dumbbell"
## [109] "var_pitch_dumbbell"      "avg_yaw_dumbbell"
## [111] "stddev_yaw_dumbbell"     "var_yaw_dumbbell"
## [113] "gyros_dumbbell_x"        "gyros_dumbbell_y"
```

```
## [115] "gyros_dumbbell_z"         "accel_dumbbell_x"
## [117] "accel_dumbbell_y"         "accel_dumbbell_z"
## [119] "magnet_dumbbell_x"        "magnet_dumbbell_y"
## [121] "magnet_dumbbell_z"        "roll_forearm"
## [123] "pitch_forearm"            "yaw_forearm"
## [125] "kurtosis_roll_forearm"    "kurtosis_picth_forearm"
## [127] "kurtosis_yaw_forearm"     "skewness_roll_forearm"
## [129] "skewness_pitch_forearm"   "skewness_yaw_forearm"
## [131] "max_roll_forearm"         "max_picth_forearm"
## [133] "max_yaw_forearm"          "min_roll_forearm"
## [135] "min_pitch_forearm"        "min_yaw_forearm"
## [137] "amplitude_roll_forearm"   "amplitude_pitch_forearm"
## [139] "amplitude_yaw_forearm"    "total_accel_forearm"
## [141] "var_accel_forearm"        "avg_roll_forearm"
## [143] "stddev_roll_forearm"      "var_roll_forearm"
## [145] "avg_pitch_forearm"        "stddev_pitch_forearm"
## [147] "var_pitch_forearm"        "avg_yaw_forearm"
## [149] "stddev_yaw_forearm"       "var_yaw_forearm"
## [151] "gyros_forearm_x"          "gyros_forearm_y"
## [153] "gyros_forearm_z"          "accel_forearm_x"
## [155] "accel_forearm_y"          "accel_forearm_z"
## [157] "magnet_forearm_x"         "magnet_forearm_y"
## [159] "magnet_forearm_z"         "classe"
```

The first variable “X” seems to be a count variable to examine this look at the summary of the variable. The varialbe has no relationship with the outcome class, which is in the last column. However, it seems to monotonically, increases with classe varialbe moves from A to E. This can potentially be a problem as this could result in a strong correlation between class and X, therefore this will have to be excluded.

```
summary(training$X)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1    4906    9812    9812   14717   19622
```

Ultimately, it is the variability of the predictors that can help with prediction. predictors that has no variability in them, does not help with prediction analysis and will only result in computational cost and loss of accuracy due to round-off errors. Therefore they need to be excluded. Therefore, variables that has near zero variability are excluded.

```
nzv <-nearZeroVar(training)
training<- training[ -nzv]
testing<- testing[ -nzv]
```

Data with significant number of NA values can cause problem for some of the prediction algorithms, therefore, preprocessing technique such as k nearest neighbors (knn) can help resolve this issue if significant amount of data is not missing. To see the number of NA values:

```
table(is.na(training))
```

```
##
##   FALSE    TRUE
## 1174344  787856
```

It is clear that there is significant number of NA values in the data. knn method will not be able to approximate accurately the missing values in this case, therefore these variable are excluded. Also, the first variable “X” and the last variable of the testing data “problem_id” are excluded.

```
a<- NULL
for(i in 1:dim(training)[2]){
        a[i]<- sum(is.na(training[, i]))
}
a<-which(a==0)
training<-training[a]
training<- training[-1]
testing<-testing[a]
testing<- testing[-c(1, 59)]
```

For this analysis Random Forest method is employed, and random forest method requires both the test and training set to have varialbes with the same class and the same levels for factors, otherwise, the method will result in an error. In this case, if the levels of the factor variables are not the same then the method will fail to predict the test classe values. The time varialbe “cvtd_timestamp” has a class of factor, however, different dates and times could exist in the test and train data set, therefore, the factor will have different levels. To fist this issue the variable is converted to a class of Date.

```
training$cvtd_timestamp<- as.Date(training$cvtd_timestamp)
testing$cvtd_timestamp<- as.Date(testing$cvtd_timestamp)
```

The training data is split into two parts called training_Train and training_Test. 60% of the data is assigned to training_Train while 40% is assigned to training_Test, this will be used for cross validation. Since the testing data given above does not have the classes values with it, the training data is split in this way so that the model can be tested.

```
training_Total <- createDataPartition(y=training$classe,p=0.6, list=FALSE)
training_Train <- training[training_Total,]
training_Test <-  training[-training_Total,]
```

Building Random Forest Model

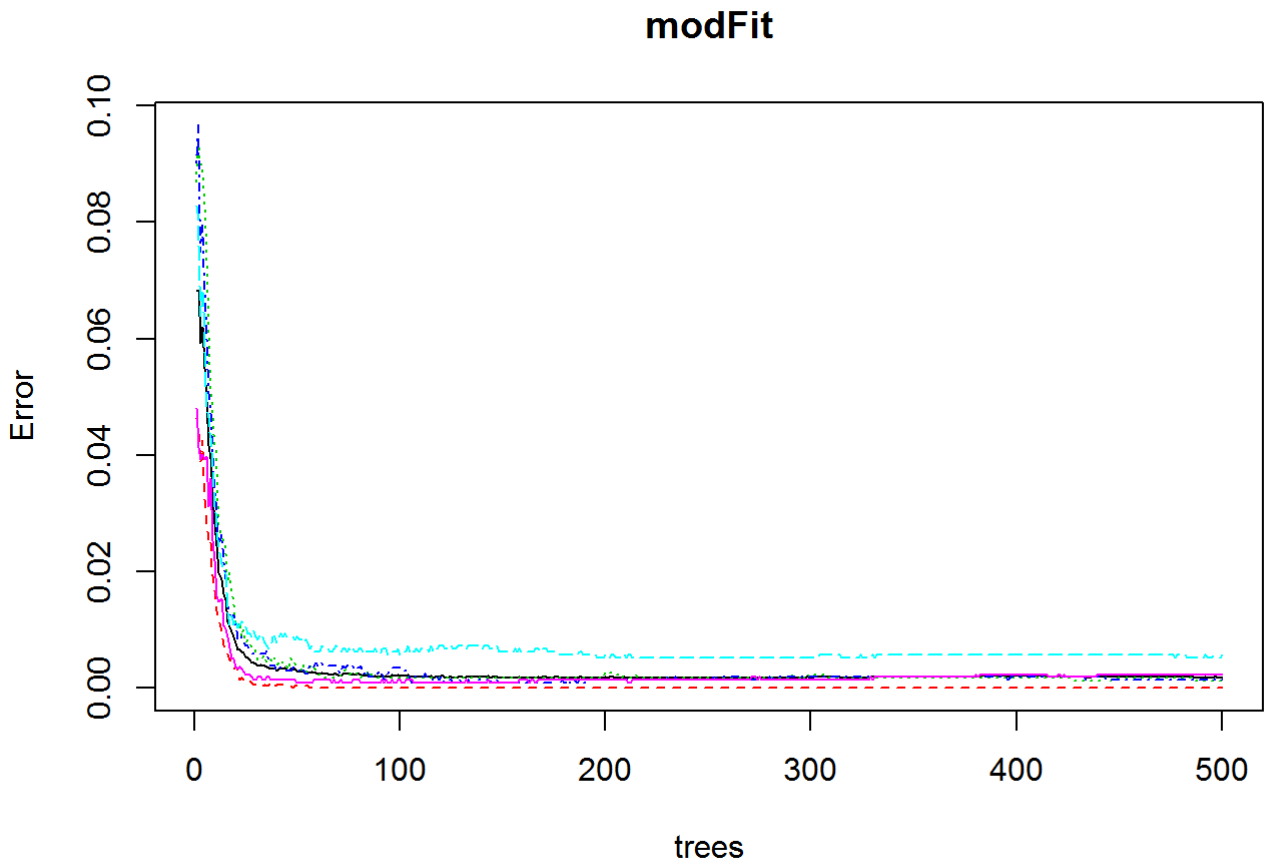
In general for accuracy many different models can be created using different methods and ensembled. However, the outcome of this problem consist of only 6 classes and large amount of data is available, therefore, a single method such as Random Forest can provide an accurate solution.

```
set.seed(112233)
modFit <- randomForest(classe ~ ., data=training_Train)
prediction <- predict(modFit, training_Test, type = "class")
confusionMatrix(prediction, training_Test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    1    0    0    0
##          B    0 1517    4    0    0
##          C    0    0 1364   10    0
##          D    0    0    0 1275    3
##          E    0    0    0    1 1439
##
## Overall Statistics
##
##                Accuracy : 0.9976
##                  95% CI : (0.9962, 0.9985)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9969
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9993   0.9971   0.9914   0.9979
## Specificity            0.9998   0.9994   0.9985   0.9995   0.9998
## Pos Pred Value         0.9996   0.9974   0.9927   0.9977   0.9993
## Neg Pred Value         1.0000   0.9998   0.9994   0.9983   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1933   0.1738   0.1625   0.1834
## Detection Prevalence   0.2846   0.1939   0.1751   0.1629   0.1835
## Balanced Accuracy      0.9999   0.9994   0.9978   0.9955   0.9989
```

By default, Random Forest method employed here uses 500 trees. If the data is significantly large, this could require significant amount of computational power. Oneway to examine how quickly has the error dropped would be to plot the error versus trees. In the following figure it can be seen that the error has reached zero for a very small number of trees.

```
plot(modFit)
```



Prediction

The following gives the prediction. The results were testing in the quiz provided with the project and 20 out 20 were predicted correctly.

```
predict(modFit, testing, type = "class")
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

Summary

As shown above the Random forest method provides an accuracy of 99.9% using only 60% of the data, therefore, the model predicted all 20 predictions correctly for the test case. Furthermore, for class A the sensitivity is given as 1, which means that all True positive cases can be identified 100% of the time if the data is collected in the manner provided. This is a significant level of accuracy considering that only 58 oof the 160 predictors were given in the original data.