Prediction Assignment

Daniel J. Riesco

24 February 2016

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. In this project, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the way in which they did the exercise. ## Loading R Packages

```
library(lattice)
library(ggplot2)
library(caret)
library(rpart)
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

Downloading data source

```
## Getting the data from internet links and locate csv files in your working directory
URL_train <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
URL_test<- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
CSV_train <- "C:/R Programming/workspace/data/pml-training.csv"
CSV_test <- "C:/R Programming/workspace/data/pml.csv"
list.files("C:/R Programming/workspace/data")
```

```
## [1] "pml-testing.csv"  "pml-training.csv"
```

Reading the training and testing data

After downloading the data from the data source, we can read the two csv files into two data frames.

```
Raw_Train_Data<- read.csv("C:/R Programming/workspace/data/pml-training.csv")
Raw_Test_Data<- read.csv("C:/R Programming/workspace/data/pml-testing.csv")
```

The training dataset contains 19622 observations with 160 variables

```
dim(Raw_Train_Data)
```

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

The testing dataset contains 20 observations and 160 variables.

```
dim(Raw_Test_Data)
```

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

The testing dataset contains 20 observations and 160 variables. The "classe" variable in the training set is going to be our outcome to predict.

Preprocessing and Cleaning the data

After looking at the training data throgh "str()" we can see there are a lot of NA and undesirable variables. Ideally, we have to get rid of near zero values and missing values as well as some useless variables.

```
sum(complete.cases(Raw_Train_Data))
```

```
## [1] 406
```

Removing NA missing values from the raw dataset (training and testing)

```
Raw_Train_Data <- Raw_Train_Data[, colSums(is.na(Raw_Train_Data)) == 0]
Raw_Test_Data <- Raw_Test_Data[, colSums(is.na(Raw_Test_Data)) == 0]
classe <- Raw_Train_Data$classe
```

Removing undesirable and factor variables from no NA's training dataset

```
## Select non related variables through pattern matching. The function is applied to a data frame with r
Train_Match_Remove <- grepl("^X|timestamp|window", names(Raw_Train_Data))
## Subsetting non related variables to get rid of them
Raw_Train_Data <- Raw_Train_Data[, !Train_Match_Remove]
## Eliminate factor variables applying is.numeric fuction to the previous dataframe
Train_Clean <- Raw_Train_Data[, sapply(Raw_Train_Data, is.numeric)]
```

The clean training dataset has the following dimensions:

```
dim(Train_Clean)
```

```
## [1] 19622    52
```

Re-assign the outcome

```
Train_Clean$classe <- classe
```

Removing undesirable and factor variables from the no NA's testing dataset

```
## Select non related variables through pattern matching. The function is applied to a data frame with r
Test_Match_Remove<- grepl("^X|timestamp|window", names(Raw_Test_Data))
## Subsetting non related variables to get rid of them
Raw_Test_Data <- Raw_Test_Data[, !Test_Match_Remove]
## Eliminate factor variables applying is.numeric fuction to the previous dataframe
Test_Clean <- Raw_Test_Data[, sapply(Raw_Test_Data, is.numeric)]
```

The clean testing dataset has the following dimensions:

```
dim(Test_Clean)
```

```
## [1] 20 53
```

Creating a partition of training dataSet

The data is divided into an 80%/20% split for training/validating repectively following common standards

```
set.seed(24216)
inTrain <- createDataPartition(Train_Clean$classe, p=0.80, list=F)
training<- Train_Clean [inTrain, ]
testing <- Train_Clean [-inTrain, ]
```

Fitting the model

We are going to fit a predictive model with Random Forest algorithm. Random Forest is a decision tree method that make boostrapping of the samples and variables at each split. It is also very accurate and robust catching correlated predictors & outliers amd perfect to solve the question of this assigment.

```
modFit <-randomForest(classe ~ ., data=training)
modFit
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.45%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4461    2    0    0    1 0.000672043
## B    9 3024    5    0    0 0.004608295
## C    0   15 2719    4    0 0.006939372
## D    0    0   25 2546    2 0.010493587
## E    0    0    2    6 2878 0.002772003
```

Cross Validation

Random Forest algorithm has its own autovalidation process in the caret package but in order to make sure we are making our prediction properly, we are going to use the misclassification error with the validating test data to prove it.

```
missClass = function(values,prediction){sum(((prediction))!= values)/length(values)}
values <- testing$classe
predictions <- predict(modFit, testing)
missClass(values, predictions)
```

```
## [1] 0.001784349
```

The missclasification error is less than 0,2% what it means we are predicting with more than 99% accuraccy.

```
Predictors_importance <- varImp(modFit)
Predictors_importance
```

```
##                          Overall
## roll_belt              997.35513
## pitch_belt             556.14597
## yaw_belt               731.71276
## total_accel_belt       172.67141
## gyros_belt_x            78.75278
## gyros_belt_y            89.66832
## gyros_belt_z           256.75305
## accel_belt_x            92.71066
## accel_belt_y            94.77134
## accel_belt_z           340.03593
## magnet_belt_x          195.62377
## magnet_belt_y          304.22140
## magnet_belt_z          308.47138
## roll_arm               239.32012
## pitch_arm              145.07804
## yaw_arm                201.53622
## total_accel_arm         77.27077
## gyros_arm_x            109.31542
## gyros_arm_y            109.55956
## gyros_arm_z             48.13042
## accel_arm_x            192.94084
## accel_arm_y            123.99069
## accel_arm_z            108.34094
## magnet_arm_x           208.91480
## magnet_arm_y           164.21706
## magnet_arm_z           155.12127
## roll_dumbbell          343.35724
## pitch_dumbbell         143.69563
## yaw_dumbbell           197.64549
## total_accel_dumbbell   215.67305
## gyros_dumbbell_x       104.65636
## gyros_dumbbell_y       191.96354
## gyros_dumbbell_z        65.11106
## accel_dumbbell_x       203.67700
## accel_dumbbell_y       332.02557
## accel_dumbbell_z       286.31662
## magnet_dumbbell_x      397.14195
## magnet_dumbbell_y      503.17721
## magnet_dumbbell_z      611.80017
## roll_forearm           490.19654
## pitch_forearm          629.65381
## yaw_forearm            140.30513
## total_accel_forearm     87.29558
## gyros_forearm_x         61.39706
## gyros_forearm_y        104.41281
```

```
## gyros_forearm_z       67.14499
## accel_forearm_x      256.06166
## accel_forearm_y      112.58021
## accel_forearm_z      193.04875
## magnet_forearm_x     174.84295
## magnet_forearm_y     170.53668
## magnet_forearm_z     225.08108
```

```r
confusionM <- table(predictions, values)
confusionMatrix(confusionM)
```

```
## Confusion Matrix and Statistics
##
##            values
## predictions    A    B    C    D    E
##           A 1116    2    0    0    0
##           B    0  756    1    0    0
##           C    0    1  683    1    0
##           D    0    0    0  642    2
##           E    0    0    0    0  719
##
## Overall Statistics
##
##                Accuracy : 0.9982
##                  95% CI : (0.9963, 0.9993)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9977
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9960   0.9985   0.9984   0.9972
## Specificity            0.9993   0.9997   0.9994   0.9994   1.0000
## Pos Pred Value         0.9982   0.9987   0.9971   0.9969   1.0000
## Neg Pred Value         1.0000   0.9991   0.9997   0.9997   0.9994
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1927   0.1741   0.1637   0.1833
## Detection Prevalence   0.2850   0.1930   0.1746   0.1642   0.1833
## Balanced Accuracy      0.9996   0.9979   0.9990   0.9989   0.9986
```

Predicting the outcome for classe

It is time now to check the perfomanse of our fit model and apply for the test datase in order to predict the outcome

```r
result <- predict(modFit, Test_Clean[, -length(names(Test_Clean))])
result
```

```
##  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
```

Conclusion

The choosen model was highly accurate and has correctly predicted 20/20 from the test set. The random forest approach to machine learning worked very nicely and useful for this kind of problem. The response of prediction is very remarkable