Practical Machine Learning - Prediction Assignment Writeup

1. SUMMARY

Devices such as Jawbone Up, Nike FuelBand, and Fitbit make possible to collect a large amount of data about personal activity relatively inexpensively.

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.

The provided dataset contained data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly, in 5 different ways.

- Training data source : https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
- Test data source : https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv
- Project source : http://groupware.les.inf.puc-rio.br/har

In this assignment, the goal was to build a model to accurately predict the manner in which people did their exercises. To accomplish this task, it was used random forests method and the `bigrf` package.

The overall prediction accuracy was approximately **99.3%**, which was a good and encouraging result. As reference, the accuracy achieved by in the original paper which used the proposed data set was **98.2%**. The out of sample error in the prediction was low, in the order of **0.726%**.

2. DOWNLOADING THE DATA AND LOADING INTO R

```
setwd("/Volumes/Documentos importantes/Coursera/8 - Practical Machine Learning/Quizzes and Project")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
    destfile = "pml-training.csv", method = "curl")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
    destfile = "pml-testing.csv", method = "curl")
train <- read.csv("pml-training.csv", header = T, na.strings = c("NA", "#DIV/0!",
    ""))
test <- read.csv("pml-testing.csv", header = T, na.strings = c("NA", "#DIV/0!",
    ""))
```

3. PARTITIONING THE TRAINING DATA SET

The training data set (object *train*) was partitioned in two data sets :

- 60% for algorithm development (*train2* object).
- 40% for algorithm testing before application to test dataset and out of sample error estimation(*test2* object).

```
if("caret" %in% rownames(installed.packages()) == FALSE) {install.packages("caret")}
library(caret)
```

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

```
set.seed(1)
split <- createDataPartition(y=train$classe, p=0.6, list=FALSE)
train2 <- train[split, ]; test2 <- train[-split, ]
dim(train2); dim(test2)
```

```
## [1] 11776   160
```

```
## [1] 7846  160
```

4. CLEANING AND CHECKING THE DATA SETS

The classification of the types of exercises should be performed based on the right variables, such as data from sensors on the belt, forearm, arm, and dumbell. Additionally, the result of the prediction should accurately classify the types of exercises as **correctly performed** (class A) or **performed incorrectly** (classes B, C, D, and E).

The mentioned variables follow the presented name patterns :

- gyros_xxx_x, gyros_xxx_y, gyros_xxx_z
- accel_xxx_x, accel_xxx_y, accel_xxx_z
- total_accel_xxx
- magnet_xxx_x, magnet_xxx_y, magnet_xxx_z
- roll_xxx
- pitch_xxx
- yaw_xxx

So, the predictiors were extrated using :

```
preds <- c(grep("^accel", names(train)), grep("^gyros", names(train)), grep("^magnet",
    names(train)), grep("^roll", names(train)), grep("^pitch", names(train)),
    grep("^yaw", names(train)), grep("^total", names(train)), grep("classe",
        names(train)))
train2_pred <- train2[, preds]
test2_pred <- test2[, preds]
```

It was possible to confirm that the remaining predictors were actually suitable, using the `nearZeroVar` function from Caret package. No remaining variable has variance close to zero :

```
nzv_train2<-nearZeroVar(train2_pred,saveMetrics=T)
nzv_test2<-nearZeroVar(test2_pred,saveMetrics=T)
nzv_train2; nzv_test2
```

```
##                      freqRatio percentUnique zeroVar   nzv
## accel_belt_x          1.000000    1.34171196   FALSE FALSE
## accel_belt_y          1.125140    1.16338315   FALSE FALSE
## accel_belt_z          1.088561    2.39470109   FALSE FALSE
## accel_arm_x           1.097087    6.43682065   FALSE FALSE
## accel_arm_y           1.185484    4.40726902   FALSE FALSE
## accel_arm_z           1.025974    6.41134511   FALSE FALSE
## accel_dumbbell_x      1.045918    3.48165761   FALSE FALSE
## accel_dumbbell_y      1.040541    3.87228261   FALSE FALSE
## accel_dumbbell_z      1.129252    3.30332880   FALSE FALSE
## accel_forearm_x       1.140351    6.60665761   FALSE FALSE
## accel_forearm_y       1.032258    8.23709239   FALSE FALSE
## accel_forearm_z       1.087912    4.68750000   FALSE FALSE
## gyros_belt_x          1.058824    1.05298913   FALSE FALSE
## gyros_belt_y          1.148268    0.55197011   FALSE FALSE
## gyros_belt_z          1.126091    1.32472826   FALSE FALSE
## gyros_arm_x           1.000000    5.27343750   FALSE FALSE
## gyros_arm_y           1.376147    3.04008152   FALSE FALSE
## gyros_arm_z           1.023739    1.98709239   FALSE FALSE
## gyros_dumbbell_x      1.014045    1.90217391   FALSE FALSE
## gyros_dumbbell_y      1.247887    2.22486413   FALSE FALSE
## gyros_dumbbell_z      1.099448    1.63043478   FALSE FALSE
## gyros_forearm_x       1.085443    2.35224185   FALSE FALSE
## gyros_forearm_y       1.109091    6.02921196   FALSE FALSE
## gyros_forearm_z       1.081633    2.44565217   FALSE FALSE
```

```
## magnet_belt_x         1.046512    2.50509511   FALSE FALSE
## magnet_belt_y         1.116711    2.36922554   FALSE FALSE
## magnet_belt_z         1.021505    3.60054348   FALSE FALSE
## magnet_arm_x          1.039216   11.02241848   FALSE FALSE
## magnet_arm_y          1.132075    7.19259511   FALSE FALSE
## magnet_arm_z          1.000000   10.59782609   FALSE FALSE
## magnet_dumbbell_x     1.065421    8.82302989   FALSE FALSE
## magnet_dumbbell_y     1.326531    6.85292120   FALSE FALSE
## magnet_dumbbell_z     1.096491    5.62160326   FALSE FALSE
## magnet_forearm_x      1.063830   11.99048913   FALSE FALSE
## magnet_forearm_y      1.075472   15.26834239   FALSE FALSE
## magnet_forearm_z      1.026316   13.40013587   FALSE FALSE
## roll_belt             1.043557    8.74660326   FALSE FALSE
## roll_arm             53.179487   19.24252717   FALSE FALSE
## roll_dumbbell         1.075000   87.48301630   FALSE FALSE
## roll_forearm         12.729730   14.97112772   FALSE FALSE
## pitch_belt            1.026786   13.79076087   FALSE FALSE
## pitch_arm            74.071429   22.46093750   FALSE FALSE
## pitch_dumbbell        2.425000   85.44497283   FALSE FALSE
## pitch_forearm        61.973684   20.88145380   FALSE FALSE
## yaw_belt              1.104027   14.55502717   FALSE FALSE
## yaw_arm              32.920635   21.22961957   FALSE FALSE
## yaw_dumbbell          1.052632   87.02445652   FALSE FALSE
## yaw_forearm          15.798658   14.17289402   FALSE FALSE
## total_accel_belt      1.044661    0.23777174   FALSE FALSE
## total_accel_arm       1.092453    0.55197011   FALSE FALSE
## total_accel_dumbbell  1.085185    0.35665761   FALSE FALSE
## total_accel_forearm   1.151194    0.58593750   FALSE FALSE
## classe                1.469065    0.04245924   FALSE FALSE

##                      freqRatio percentUnique zeroVar   nzv
## accel_belt_x          1.057508    1.93729289   FALSE FALSE
## accel_belt_y          1.097638    1.64414989   FALSE FALSE
## accel_belt_z          1.062874    3.46673464   FALSE FALSE
## accel_arm_x           1.083333    9.29135865   FALSE FALSE
## accel_arm_y           1.077778    6.44914606   FALSE FALSE
## accel_arm_z           1.265306    9.22763191   FALSE FALSE
## accel_dumbbell_x      1.014815    4.66479735   FALSE FALSE
## accel_dumbbell_y      1.072165    5.48049962   FALSE FALSE
## accel_dumbbell_z      1.139785    5.02166709   FALSE FALSE
## accel_forearm_x       1.025000    9.60999235   FALSE FALSE
## accel_forearm_y       1.097561   11.90415498   FALSE FALSE
## accel_forearm_z       1.093750    6.66581698   FALSE FALSE
## gyros_belt_x          1.058380    1.45296967   FALSE FALSE
## gyros_belt_y          1.137553    0.76472088   FALSE FALSE
## gyros_belt_z          1.017981    1.95003824   FALSE FALSE
## gyros_arm_x           1.040201    7.68544481   FALSE FALSE
## gyros_arm_y           1.590426    4.46087178   FALSE FALSE
## gyros_arm_z           1.267380    2.68926842   FALSE FALSE
## gyros_dumbbell_x      1.047809    2.80397655   FALSE FALSE
## gyros_dumbbell_y      1.232365    3.16084629   FALSE FALSE
## gyros_dumbbell_z      1.000000    2.30690798   FALSE FALSE
## gyros_forearm_x       1.019324    3.31379047   FALSE FALSE
## gyros_forearm_y       1.065359    8.66683660   FALSE FALSE
## gyros_forearm_z       1.188172    3.19908233   FALSE FALSE
## magnet_belt_x         1.260563    3.42849860   FALSE FALSE
## magnet_belt_y         1.075472    3.39026255   FALSE FALSE
## magnet_belt_z         1.102151    4.98343105   FALSE FALSE
## magnet_arm_x          1.076923   16.25031863   FALSE FALSE
```

```
## magnet_arm_y          1.111111   10.62962019   FALSE FALSE
## magnet_arm_z          1.024390   15.21794545   FALSE FALSE
## magnet_dumbbell_x     1.134328   12.09533520   FALSE FALSE
## magnet_dumbbell_y     1.037975    9.99235279   FALSE FALSE
## magnet_dumbbell_z     1.223881    8.00407851   FALSE FALSE
## magnet_forearm_x      1.062500   16.90033138   FALSE FALSE
## magnet_forearm_y      1.411765   21.34845781   FALSE FALSE
## magnet_forearm_z      1.035714   18.56997196   FALSE FALSE
## roll_belt             1.195906   10.33647719   FALSE FALSE
## roll_arm             45.793103   25.74560285   FALSE FALSE
## roll_dumbbell         1.096154   89.05174611   FALSE FALSE
## roll_forearm         10.192053   18.09839409   FALSE FALSE
## pitch_belt            1.048780   18.36604639   FALSE FALSE
## pitch_arm            73.833333   28.80448636   FALSE FALSE
## pitch_dumbbell        2.070175   87.45857762   FALSE FALSE
## pitch_forearm        61.520000   26.82895743   FALSE FALSE
## yaw_belt              1.004673   18.65918940   FALSE FALSE
## yaw_arm              29.511111   28.42212592   FALSE FALSE
## yaw_dumbbell          1.096154   88.45271476   FALSE FALSE
## yaw_forearm          14.647619   18.02192200   FALSE FALSE
## total_accel_belt      1.091532    0.34412439   FALSE FALSE
## total_accel_arm       1.082386    0.82844762   FALSE FALSE
## total_accel_dumbbell  1.054250    0.52255927   FALSE FALSE
## total_accel_forearm   1.094456    0.80295692   FALSE FALSE
## classe                1.470356    0.06372674   FALSE FALSE
```

Finally, it was checked if NAs remained in the data sets :

```
sum(is.na(train2_pred)); sum(is.na(test2_pred))
```

```
## [1] 0
```

```
## [1] 0
```

As both values were zeros, it was possible to conclude that no NAs remained in the data sets.

4. USING RANDOM FORRESTS

4.1. Using Caret package

The prediction model which was first tried used the Caret package and the **random forrests** method. However, due to the size of the `train2_pred` data frame (almost 12,000 entries), the processing time reached several minutes. According to the original paper, using random forrests, a weighted accuracy of 98.2% was achieved.

For this reason, an alternative method for running random forrests was tried, aiming to achieve a satisfactory accuracy.

4.2. Using 'bigrf' package

After a fast research at Google, the `bigrf` package was reverted as a faster option to traditional random forest in Caret package. the `bigrf` package is an implementation of Leo Breiman's and Adele Cutler's Random Forest algorithms for classification and regression, with optimizations for performance and for handling of data sets that are too large to be processed in memory.

- Package source : https://github.com/aloysius-lim/bigrf

Additionally, parallel processing using multicore features in `doParallel` package helped to enhance the overall computation speed.

The `bigrf` package could build the prediction model (using random forests method) and classification in few seconds.

4.3. Building the prediction model and applying to training set

The first step was the installation of the new package :

```
if("bigrf" %in% rownames(installed.packages()) == FALSE) {install.packages("bigrf")}
library(bigrf)
```

```
## Loading required package: bigmemory
## Loading required package: bigmemory.sri
## Loading required package: BH

## Warning: package 'BH' was built under R version 3.1.2

##
## bigmemory >= 4.0 is a major revision since 3.1.2; please see packages
## biganalytics and and bigtabulate and http://www.bigmemory.org for more information.
```

Later, the parallel/multicore processing was activated :

```
if ("doParallel" %in% rownames(installed.packages()) == FALSE) {
    install.packages("doParallel")
}
library(doParallel)
registerDoParallel(cores = detectCores(all.tests = TRUE))
```

So, the function `bigrfc` was used to build the classification model based in the random forests method :

```
set.seed(1)
fit<-bigrfc(train2_pred, train2_pred$classe,varselect = 1:52)
```

```
## OOB errors:
##  Tree  Overall error  Error by class
##                           A      B      C      D      E
##    10           5.96   3.97   8.16   7.74   7.20   3.93
##    20           2.28  0.657  3.774  2.532  3.679  1.709
##    30           1.44  0.329  2.326  1.753  2.591  0.924
##    40           1.16  0.209  1.843  1.509  2.228  0.647
##    50          0.968  0.119  1.667  1.120  2.021  0.462
```

Fifty (50) random forests were performed, in order to reduce the Out-Of-Bag (OOB) classification error to a minimum value.

The function `predict` was used to predict the classes of the same training set used to build the model :

```
pred<-predict(fit, train2_pred, train2_pred$classe)
```

```
## Test errors:
##  Tree  Overall error  Error by class
##                           A      B      C      D      E
##    10         0.0255  0.000  0.000  0.000  0.155  0.000
##    20           0.00   0.00   0.00   0.00   0.00   0.00
##    30           0.00   0.00   0.00   0.00   0.00   0.00
##    40           0.00   0.00   0.00   0.00   0.00   0.00
##    50           0.00   0.00   0.00   0.00   0.00   0.00
```

The error rates and confusion matrix can be verified :

```r
summary(pred)
```

```
## Predictions on 11776 examples using random forest with 50 trees.
##
## Test set labels:
##
##    A    B    C    D    E
## 3348 2279 2054 1930 2165
##
## Overall error rate: 0.00
##
## Test set confusion matrix (OOB):
##       Predicted
## Actual    A    B    C    D    E
##      A 3348    0    0    0    0
##      B    0 2279    0    0    0
##      C    0    0 2054    0    0
##      D    0    0    0 1930    0
##      E    0    0    0    0 2165
```

It was possible to verify that the built model scored **100%** accuracy of prediction in all classes, and **0.00%** overall error, in the training data set.

4.4. Testing prediction model using cross-validation

In despite of the built model showing good results in the prediction of training data set, an additional and important test was to apply the same model to a new data set. For this purpose, the `test2_pred` data set was used :

```r
pred2<-predict(fit, test2_pred, test2_pred$classe)
```

```
## Test errors:
##  Tree  Overall error  Error by class
##                           A      B      C      D      E
##    10           1.50  0.986  2.503  1.974  1.944  0.416
##    20          0.905  0.314  1.713  1.023  1.477  0.347
##    30          0.803  0.224  1.120  0.877  1.866  0.347
##    40          0.701  0.224  0.922  0.804  1.477  0.416
##    50          0.688  0.224  0.856  0.731  1.555  0.416
```

Finnaly, the error rates can be verified and a confusion matrix could be built, to compare the actual values to its predicted values :

```r
summary(pred2)
```

```
## Predictions on 7846 examples using random forest with 50 trees.
##
## Test set labels:
##
##    A    B    C    D    E
## 2232 1518 1368 1286 1442
##
## Overall error rate: 0.688
##
## Test set confusion matrix (OOB):
##       Predicted
```

```
## Actual    A    B    C    D    E
##      A 2227    3    1    0    1
##      B    6 1505    7    0    0
##      C    0    8 1358    2    0
##      D    0    0   20 1266    0
##      E    0    0    0    6 1436
```

The out of sample error rate in the classification was **0.726%**, which can be considered low.

The accuracy of the model applied to `test2_pred` was :

- Class A : 99.8%
- Class B : 99.1%
- Class C : 99.3%
- Class D : 98.4%
- Class E : 99.6%
- **Overall accuracy : 99.3%**

The overall accuracy of **99.3%** was considered to be suitable, and for this reason, the built model was accepted to the next step.

4.5. Applying the prediction model to test set

Finnaly, the built prediction model was applied to the `test` data set.

```
test_pred<-test[, preds]
pred3<-predict(fit, test_pred)
```

```
## Processing tree number:
##    10
##    20
##    30
##    40
##    50
```

```
pred3<-replace(pred3, pred3==1, "A")
pred3<-replace(pred3, pred3==2, "B")
pred3<-replace(pred3, pred3==3, "C")
pred3<-replace(pred3, pred3==4, "D")
pred3<-replace(pred3, pred3==5, "E")
```

The predicted classes were :

```
as.data.frame(pred3)
```

```
##    pred3
## 1      B
## 2      A
## 3      B
## 4      A
## 5      A
## 6      E
## 7      D
```

```
## 8       B
## 9       A
## 10      A
## 11      B
## 12      C
## 13      B
## 14      A
## 15      E
## 16      E
## 17      A
## 18      B
## 19      B
## 20      B
```

5. CONCLUSION

The random forrests method is, indeed, very accurate in classification procedures. The computing time may become an issue when the data set is large. Fortunatelly, there are options to make this process faster, keeping its accuracy in a high level.

The predicted classes for `test` data set were all correct according to project submission system, which confirms the high accuracy of the proposed process.

It was possible to achieve a better accuracy than obtained in the original paper. As seen in item 4.4, the overall accuracy was approximately **99.3%**, against **98.2%** of the original paper.

The out of sample error was low, only **0.726%**.

Finally, the parallel computation using multicore features of modern processors is a very relevant procedure for larger scale computations.