# Practical Machine Learning project from coursera

*raquelhr*

*22 December 2015*

**Project directive** *In this project, your goal will be to use 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. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset). We acknowledge the data for this project came from this source: http://groupware.les.inf.puc-rio.br/har*

## Exploratory data analysis

Loading necessary packages and setting seed to meet reproducibility criteria

```
library(caret)
library(gbm)
library(AppliedPredictiveModeling)
library(randomForest)
library(rpart)
library(rattle)
library(dplyr)
library(plyr)
library(mlearning)
library(reshape2)
library(ggplot2)
library(gridExtra)
set.seed(3433)
```

## Getting data

```
url.training <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url.testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url.training, destfile="training.csv", method="curl")
download.file(url.testing, destfile="testing.csv", method="curl")
training.data <- read.csv("training.csv", na.strings=c("NA","#DIV/0!","", " "),
                          header=TRUE)
testing.data <- read.csv("testing.csv", na.strings=c("NA","#DIV/0!","", " "),
                         header=TRUE)
dim(training.data)
```

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

```
dim(testing.data)
```

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

## Cleaning data

There are many columns that only have NAs, or have majority NAs. We will deal with these so as to avoid overfitting.

```
#remove columns and rows with only NA
training.data <- training.data[ , !apply(is.na(training.data), 2, all) ]
testing.data <- testing.data[ , !apply(is.na(testing.data), 2, all)  ]
dim(training.data)
```

```
## [1] 19622   154
```

```
dim(testing.data)
```

```
## [1] 20 60
```

```
names(testing.data)
```

```
##  [1] "X"                    "user_name"            "raw_timestamp_part_1"
##  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
##  [7] "num_window"           "roll_belt"            "pitch_belt"
## [10] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
## [13] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [16] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
## [19] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
## [22] "pitch_arm"            "yaw_arm"              "total_accel_arm"
## [25] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
## [28] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [31] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
## [34] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [37] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
## [40] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
## [43] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
## [46] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
## [49] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [52] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
## [55] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
## [58] "magnet_forearm_y"     "magnet_forearm_z"     "problem_id"
```

```
#deleting columns in the training dataset with more than 60% of NAs
training.data <- training.data[, colSums(is.na(training.data)) < dim(training.data)[1]*0.6 ]
#only missing one is the classe, and this is the one we want to predict
names(training.data) == names(testing.data)
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [56]  TRUE  TRUE  TRUE  TRUE FALSE
```

```r
#remove first 2 columns, which should be irrelevant
#for the machine learning algorithms
training.data <- training.data[,-(1:2)]
testing.data <- testing.data[,-(1:2)]
dim(training.data)
```

```
## [1] 19622    58
```

```r
dim(testing.data)
```

```
## [1] 20 58
```

Now we have reduced the size of the data, we are ready to process it. #Processing the training dataset Training data is quite large, so we divide 75% into a training set and another 25% into a testing set to check for accuracy, before finally applying to the testing.data dataset.

```r
inTrain <- createDataPartition(training.data$classe, p = 3/4)[[1]]
inTrain.training <- training.data[ inTrain,]
inTrain.testing <- training.data[-inTrain,]
```

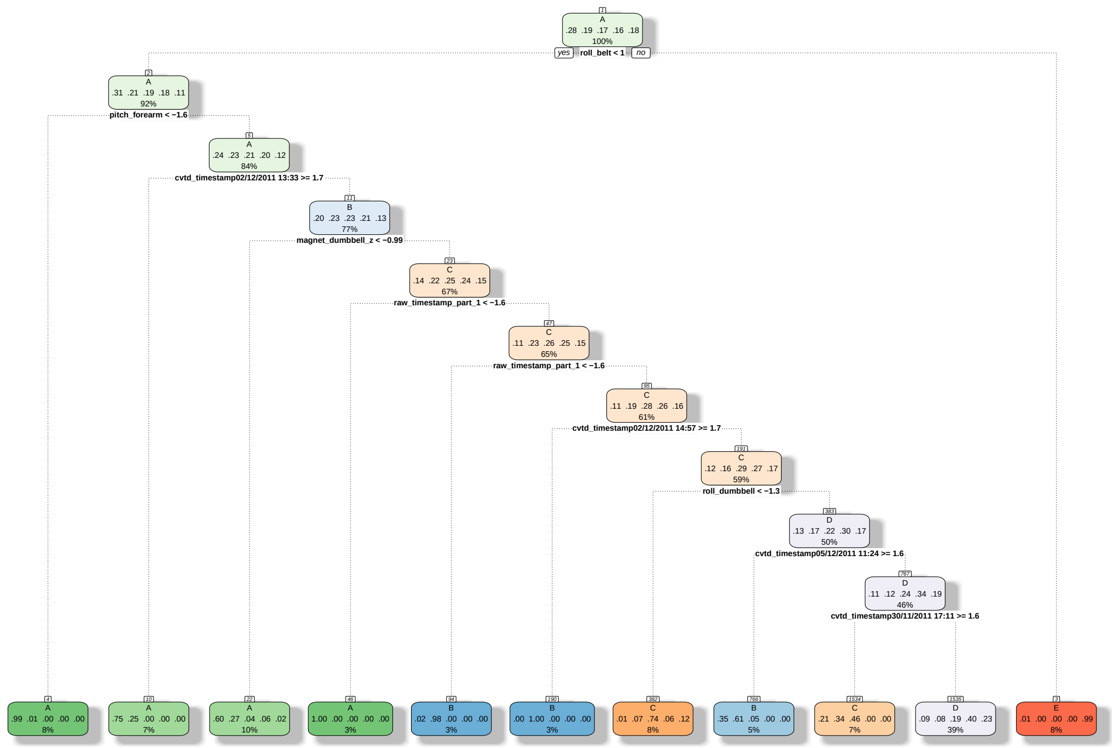
# Cross-validation and training

For estimating model accuracy we will perform repeated $k = 2$-fold Cross Validation. Ideally, we will use $k = 8$-fold, or so, but the computational time increases rapidly. As it turns out, this will be sufficient for our purposes.

```r
#for reproducibility
set.seed(62433)
train.control <- trainControl(method="repeatedcv", number=2, repeats=2)
```

Implementing different Machine Learning algorithms (this is only a representative list, and in no way exhaustive).

```r
rpart.inTrain.training <- train(classe ~ ., method='rpart', data=inTrain.training,
                                preProcess=c("center", "scale"),
                                trControl=train.control)
#depicting the regression tree
fancyRpartPlot(rpart.inTrain.training$finalModel,sub='rhr')
```




rhr

```
rf.inTrain.training <- train(classe ~ ., method='rf', data=inTrain.training,
                             preProcess=c("center", "scale"),
                             trControl=train.control)
boosted.inTrain.training <- train(classe ~ ., method='gbm', data=inTrain.training,
                                  verbose=FALSE,
                                  preProcess=c("center", "scale"),
                                  trControl=train.control)
lda.inTrain.training <- train(classe ~ ., method='lda', data=inTrain.training,
                              preProcess=c("center", "scale"),
                              trControl=train.control)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

Now, we test the trained algorithms and apply these into the inTrain testing set

```
predict.rpart.inTrain.training  <- predict(rpart.inTrain.training, inTrain.testing)
predict.rf.inTrain.training  <- predict(rf.inTrain.training, inTrain.testing)
predict.boosted.inTrain.training  <- predict(boosted.inTrain.training, inTrain.testing)
predict.lda.inTrain.training  <- predict(lda.inTrain.training, inTrain.testing)
```

# Checking accuracy of the methods within the training dataset

```
rpart.accuracy.inTrain.training <- confusionMatrix(predict.rpart.inTrain.training,
                                                   inTrain.testing$classe)
```

```r
rf.accuracy.inTrain.training <- confusionMatrix(predict.rf.inTrain.training,
                                                inTrain.testing$classe)
rf.accuracy.inTrain.training
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    1    0    0    0
##          B    0  948    1    0    0
##          C    0    0  852    0    0
##          D    0    0    2  804    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.9992
##                  95% CI : (0.9979, 0.9998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.999
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9989   0.9965   1.0000   1.0000
## Specificity            0.9997   0.9997   1.0000   0.9995   1.0000
## Pos Pred Value         0.9993   0.9989   1.0000   0.9975   1.0000
## Neg Pred Value         1.0000   0.9997   0.9993   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1933   0.1737   0.1639   0.1837
## Detection Prevalence   0.2847   0.1935   0.1737   0.1644   0.1837
## Balanced Accuracy      0.9999   0.9993   0.9982   0.9998   1.0000
```

```r
boosted.accuracy.inTrain.training <- confusionMatrix(predict.boosted.inTrain.training,
                                                     inTrain.testing$classe)
boosted.accuracy.inTrain.training
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    3    0    0    0
##          B    0  944    2    0    0
##          C    0    2  847    0    0
##          D    0    0    6  802    1
##          E    0    0    0    2  900
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9967
##                  95% CI : (0.9947, 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   0.9906   0.9975   0.9989
## Specificity            0.9991   0.9995   0.9995   0.9983   0.9995
## Pos Pred Value         0.9979   0.9979   0.9976   0.9913   0.9978
## Neg Pred Value         1.0000   0.9987   0.9980   0.9995   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1925   0.1727   0.1635   0.1835
## Detection Prevalence   0.2851   0.1929   0.1731   0.1650   0.1839
## Balanced Accuracy      0.9996   0.9971   0.9951   0.9979   0.9992
```

```
lda.accuracy.inTrain.training <- confusionMatrix(predict.lda.inTrain.training,
                                                  inTrain.testing$classe)
```

We observe that methods 'gradient boosting' and 'random forest' both offer better accuracy to predict the testing dataset within the training dataset.

## Predictions using the trained algorithms

Finally, let's now apply the methods we have experimented with to the true testing dataset, and provide the answers.

```
predict.rpart.testing <- predict(rpart.inTrain.training, testing.data)
predict.rf.testing  <- predict(rf.inTrain.training, testing.data)
predict.boosted.testing  <- predict(boosted.inTrain.training, testing.data)
predict.lda.testing  <- predict(lda.inTrain.training, testing.data)
#comparing predictions from methods employed
predict.rpart.testing
```

```
##  [1] D C C A A D D C A A B C B A D D D B D B
## Levels: A B C D E
```

```
predict.rf.testing
```

```
##  [1] 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
```

```
predict.boosted.testing
```

```
##  [1] 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
```

```
predict.lda.testing
```

```
##  [1] B B B A A E D C A A B C B A E E A B B B
## Levels: A B C D E
```

```
#we also note that
predict.rf.testing == predict.boosted.testing
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [15] TRUE TRUE TRUE TRUE TRUE TRUE
```

Thus 'gradient boosting' and 'random forest' offer the same predicted results, and also gave the highest accuracy in the training dataset. Therefore, we will use their results.

# Completion of the project

To complete the project, we publish the prediction of the testing

```
answers <- c('B', 'A', 'B', 'A', 'A', 'E', 'D', 'B', 'A', 'A', 'B',
             'C', 'B', 'A', 'E', 'E', 'A', 'B', 'B', 'B')
pml_write_files = function(x){
    n = length(x)
    for(i in 1:n){
        filename = paste0("problem_id_",i,".txt")
        write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
    }
}
pml_write_files(answers)
```

# Appendix

## Depicting confusion matrices with heatmaps