**Final Project Report - Practical Machine Learning Course**

### Synopsis

Machine Learning is the process of using algorithms to learn from data. Perhaps the most important aspect of any machine learning problem is the rather human process of determing what we are trying to learn about.

In the study we will discuss in this paper (<http://groupware.les.inf.puc-rio.br/har>), they investigated the use of computing to evaluate “proper” exercise form (possibly allowing computers to replace personal trainers to help us become [better, faster, stronger](https://www.youtube.com/watch?v=HoLs0V8T5AA).

**Reproduceablity**

In order to reproduce the same results, you need a certain set of packages, as well as setting a pseudo random seed equal to the one I used. \*Note:To install, for instance, the caret package in R, run this command: install.packages(“caret”)

The following Libraries were used for this project, which you should install - if not done yet - and load on your working environment.

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(rpart)

library(rpart.plot)

library(RColorBrewer)

library(rattle)

## Rattle: A free graphical interface for data mining with R.

## Version 3.1.0 Copyright (c) 2006-2014 Togaware Pty Ltd.

## Type 'rattle()' to shake, rattle, and roll your data.

library(randomForest)

## randomForest 4.6-10

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

Finally, load the same seed with the following line of code:

set.seed(12345)

**Getting the data**

The training data set can be found on the following URL:

trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

The testing data set can be found on the following URL:

testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

Procedure 1) Procedure 1) assumes that you only want to store the data files in memory.

Load data to memory solely

training <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))

testing <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))

Procedure 2) Procedure 2) assumes that you want to store the data files in memory and on disk. (Thus, it involves downloading data directly to your hard drive.)

You can use following function to download the data:

#getDataFiles <- function(filesDirectory = "./") {

# if (!file.exists(filesDirectory)) {

# dir.create(filesDirectory)

# }

# testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

# trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

# trainFile <- "train.csv"

# testFile <- "test.csv"

# trainFilePath <- paste(filesDirectory, trainFile, sep = "/")

# testFilePath <- paste(filesDirectory, testFile, sep = "/")

# download.file(trainUrl, destfile = trainFilePath, method="curl")

# download.file(testUrl, destfile = testFilePath, method="curl")

# training <- read.csv(trainFilePath, na.strings=c("NA","#DIV/0!",""))

# testing <- read.csv(testFilePath, na.strings=c("NA","#DIV/0!",""))

#}

Run the function, for example, as follows:

#getDataFiles("/data")

Note that you can simply run the function without passing any argument, which means the file will be downloaded to your current working directory. Note: To view your current working directory run the following command:

getwd()

## [1] "/Users/DiogoAdmin/Documents/Developer/R/Practical Machine Learning/Course Project"

**Partioning the training set into two**

Partioning Training data set into two data sets, 60% for myTraining, 40% for myTesting:

inTrain <- createDataPartition(y=training$classe, p=0.6, list=FALSE)

myTraining <- training[inTrain, ]; myTesting <- training[-inTrain, ]

dim(myTraining); dim(myTesting)

## [1] 11776 160

## [1] 7846 160

**Cleaning the data**

The following transformations were used to clean the data:

Transformation 1: Cleaning NearZeroVariance Variables Run this code to view possible NZV Variables:

myDataNZV <- nearZeroVar(myTraining, saveMetrics=TRUE)

Run this code to create another subset without NZV variables:

myNZVvars <- names(myTraining) %in% c("new\_window", "kurtosis\_roll\_belt", "kurtosis\_picth\_belt",

"kurtosis\_yaw\_belt", "skewness\_roll\_belt", "skewness\_roll\_belt.1", "skewness\_yaw\_belt",

"max\_yaw\_belt", "min\_yaw\_belt", "amplitude\_yaw\_belt", "avg\_roll\_arm", "stddev\_roll\_arm",

"var\_roll\_arm", "avg\_pitch\_arm", "stddev\_pitch\_arm", "var\_pitch\_arm", "avg\_yaw\_arm",

"stddev\_yaw\_arm", "var\_yaw\_arm", "kurtosis\_roll\_arm", "kurtosis\_picth\_arm",

"kurtosis\_yaw\_arm", "skewness\_roll\_arm", "skewness\_pitch\_arm", "skewness\_yaw\_arm",

"max\_roll\_arm", "min\_roll\_arm", "min\_pitch\_arm", "amplitude\_roll\_arm", "amplitude\_pitch\_arm",

"kurtosis\_roll\_dumbbell", "kurtosis\_picth\_dumbbell", "kurtosis\_yaw\_dumbbell", "skewness\_roll\_dumbbell",

"skewness\_pitch\_dumbbell", "skewness\_yaw\_dumbbell", "max\_yaw\_dumbbell", "min\_yaw\_dumbbell",

"amplitude\_yaw\_dumbbell", "kurtosis\_roll\_forearm", "kurtosis\_picth\_forearm", "kurtosis\_yaw\_forearm",

"skewness\_roll\_forearm", "skewness\_pitch\_forearm", "skewness\_yaw\_forearm", "max\_roll\_forearm",

"max\_yaw\_forearm", "min\_roll\_forearm", "min\_yaw\_forearm", "amplitude\_roll\_forearm",

"amplitude\_yaw\_forearm", "avg\_roll\_forearm", "stddev\_roll\_forearm", "var\_roll\_forearm",

"avg\_pitch\_forearm", "stddev\_pitch\_forearm", "var\_pitch\_forearm", "avg\_yaw\_forearm",

"stddev\_yaw\_forearm", "var\_yaw\_forearm")

myTraining <- myTraining[!myNZVvars]

#To check the new N?? of observations

dim(myTraining)

## [1] 11776 100

Transformation 2: Killing first column of Dataset - ID Removing first ID variable so that it does not interfer with ML Algorithms:

myTraining <- myTraining[c(-1)]

Transformation 3: Cleaning Variables with too many NAs. For Variables that have more than a 60% threshold of NA’s I’m going to leave them out:

trainingV3 <- myTraining #creating another subset to iterate in loop

for(i in 1:length(myTraining)) { #for every column in the training dataset

if( sum( is.na( myTraining[, i] ) ) /nrow(myTraining) >= .6 ) { #if n?? NAs > 60% of total observations

for(j in 1:length(trainingV3)) {

if( length( grep(names(myTraining[i]), names(trainingV3)[j]) ) ==1) { #if the columns are the same:

trainingV3 <- trainingV3[ , -j] #Remove that column

}

}

}

}

#To check the new N?? of observations

dim(trainingV3)

## [1] 11776 58

#Seting back to our set:

myTraining <- trainingV3

rm(trainingV3)

Now let us do the exact same 3 transformations but for our myTesting and testing data sets.

clean1 <- colnames(myTraining)

clean2 <- colnames(myTraining[, -58]) #already with classe column removed

myTesting <- myTesting[clean1]

testing <- testing[clean2]

#To check the new N?? of observations

dim(myTesting)

## [1] 7846 58

#To check the new N?? of observations

dim(testing)

## [1] 20 57

#Note: The last column - problem\_id - which is not equal to training sets, was also "automagically" removed

#No need for this code:

#testing <- testing[-length(testing)]

In order to ensure proper functioning of Decision Trees and especially RandomForest Algorithm with the Test data set (data set provided), we need to coerce the data into the same type.

for (i in 1:length(testing) ) {

for(j in 1:length(myTraining)) {

if( length( grep(names(myTraining[i]), names(testing)[j]) ) ==1) {

class(testing[j]) <- class(myTraining[i])

}

}

}

#And to make sure Coertion really worked, simple smart ass technique:

testing <- rbind(myTraining[2, -58] , testing) #note row 2 does not mean anything, this will be removed right.. now:

testing <- testing[-1,]

**Using ML algorithms for prediction: Decision Tree**

modFitA1 <- rpart(classe ~ ., data=myTraining, method="class")

Note: to view the decision tree with fancy run this command:

fancyRpartPlot(modFitA1)

Predicting:

predictionsA1 <- predict(modFitA1, myTesting, type = "class")

(Moment of truth) Using confusion Matrix to test results:

confusionMatrix(predictionsA1, myTesting$classe)

## Confusion Matrix and Statistics

##

## Reference

## Prediction A B C D E

## A 2150 60 7 1 0

## B 61 1260 69 64 0

## C 21 188 1269 143 4

## D 0 10 14 857 78

## E 0 0 9 221 1360

##

## Overall Statistics

##

## Accuracy : 0.879

## 95% CI : (0.871, 0.886)

## No Information Rate : 0.284

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.847

## Mcnemar's Test P-Value : NA

##

## Statistics by Class:

##

## Class: A Class: B Class: C Class: D Class: E

## Sensitivity 0.963 0.830 0.928 0.666 0.943

## Specificity 0.988 0.969 0.945 0.984 0.964

## Pos Pred Value 0.969 0.867 0.781 0.894 0.855

## Neg Pred Value 0.985 0.960 0.984 0.938 0.987

## Prevalence 0.284 0.193 0.174 0.164 0.184

## Detection Rate 0.274 0.161 0.162 0.109 0.173

## Detection Prevalence 0.283 0.185 0.207 0.122 0.203

## Balanced Accuracy 0.976 0.900 0.936 0.825 0.954

#Overall Statistics

# Accuracy : 0.8683

# 95% CI : (0.8607, 0.8757)

# No Information Rate : 0.2845

# P-Value [Acc > NIR] : < 2.2e-16

# Kappa : 0.8335

**Using ML algorithms for prediction: Random Forests**

modFitB1 <- randomForest(classe ~. , data=myTraining)

Predicting in-sample error:

predictionsB1 <- predict(modFitB1, myTesting, type = "class")

(Moment of truth) Using confusion Matrix to test results:

confusionMatrix(predictionsB1, myTesting$classe)

## Confusion Matrix and Statistics

##

## Reference

## Prediction A B C D E

## A 2231 2 0 0 0

## B 1 1516 2 0 0

## C 0 0 1366 3 0

## D 0 0 0 1282 2

## E 0 0 0 1 1440

##

## Overall Statistics

##

## Accuracy : 0.999

## 95% CI : (0.997, 0.999)

## No Information Rate : 0.284

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.998

## Mcnemar's Test P-Value : NA

##

## Statistics by Class:

##

## Class: A Class: B Class: C Class: D Class: E

## Sensitivity 1.000 0.999 0.999 0.997 0.999

## Specificity 1.000 1.000 1.000 1.000 1.000

## Pos Pred Value 0.999 0.998 0.998 0.998 0.999

## Neg Pred Value 1.000 1.000 1.000 0.999 1.000

## Prevalence 0.284 0.193 0.174 0.164 0.184

## Detection Rate 0.284 0.193 0.174 0.163 0.184

## Detection Prevalence 0.285 0.194 0.174 0.164 0.184

## Balanced Accuracy 1.000 0.999 0.999 0.998 0.999

#Overall Statistics

# Accuracy : 0.999

# 95% CI : (0.998, 0.9996)

# No Information Rate : 0.2845

# P-Value [Acc > NIR] : < 2.2e-16

# Kappa : 0.9987

#Mcnemar's Test P-Value : NA

Random Forests yielded better Results, as expected!

**Generating Files to submit as answers for the Assignment:**

Finally, using the provided Test Set out-of-sample error.

For Random Forests we use the following formula, which yielded a much better prediction in in-sample:

predictionsB2 <- predict(modFitB1, testing, type = "class")

Function to generate files with predictions to submit for assignment

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(predictionsB2)