Project: Prediction Assignment Writeup

Ali Attajer

16/06/2020

## How the model was built

Our outcome variable is classe, a factor variable with 5 levels. For this data set, “participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in 5 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) - throwing the hips to the front (Class E)

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes." [1] Prediction evaluations will be based on maximizing the accuracy and minimizing the out-of-sample error. All other available variables after cleaning will be used for prediction. Two models will be tested using decision tree and random forest algorithms. The model with the highest accuracy will be chosen as our final model.

## Cross-validation

Cross-validation will be performed by subsampling our training data set randomly without replacement into 2 subsamples: subTraining data (75% of the original Training data set) and subTesting data (25%). Our models will be fitted on the subTraining data set, and tested on the subTesting data. Once the most accurate model is choosen, it will be tested on the original Testing data set.

## Expected out-of-sample error

The expected out-of-sample error will correspond to the quantity: 1-accuracy in the cross-validation data. Accuracy is the proportion of correct classified observation over the total sample in the subTesting data set. Expected accuracy is the expected accuracy in the out-of-sample data set (i.e. original testing data set). Thus, the expected value of the out-of-sample error will correspond to the expected number of missclassified observations/total observations in the Test data set, which is the quantity: 1-accuracy found from the cross-validation data set.

Our outcome variable “classe” is an unordered factor variable. Thus, we can choose our error type as 1-accuracy. We have a large sample size with N= 19622 in the Training data set. This allow us to divide our Training sample into subTraining and subTesting to allow cross-validation. Features with all missing values will be discarded as well as features that are irrelevant. All other features will be kept as relevant variables. Decision tree and random forest algorithms are known for their ability of detecting the features that are important for classification [2]. ## Packages and libraries

Installing packages and loading libraries:

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(randomForest)

## randomForest 4.6-14

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

##   
## Attaching package: 'randomForest'

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

library(rpart)  
library(rpart.plot)  
library(rattle)

## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.  
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.  
## Entrez 'rattle()' pour secouer, faire vibrer, et faire défiler vos données.

##   
## Attaching package: 'rattle'

## The following object is masked from 'package:randomForest':  
##   
## importance

library(RColorBrewer)  
library(e1071)

## Seed for reproduceability

For reproduceability, An overall pseudo-random number generator seed was set at 1234 for all code:

set.seed(1234)

## Getting and cleaning data

Getting the training and testing data:

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

## Partioning the training set into two data sets

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 3 transformations were used to return tidy data:  
**Transformation 1:** Cleaning NearZeroVariance Variables Run this code to view possible NZV Variables:

myDataNZV <- nearZeroVar(myTraining, saveMetrics=TRUE)

Another subset of 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]  
  
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 we’re 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 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

#Setting back to our set:  
myTraining <- trainingV3  
rm(trainingV3)

Now let us do the exact same 3 transformations for 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 dimensions of observations  
dim(myTesting)

## [1] 7846 58

dim(testing)

## [1] 20 57

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 Corection really worked, simple smart technique:  
testing <- rbind(myTraining[2, -58] , testing) #note row 2 does not mean anything, this will be removed right.. now:  
testing <- testing[-1,]

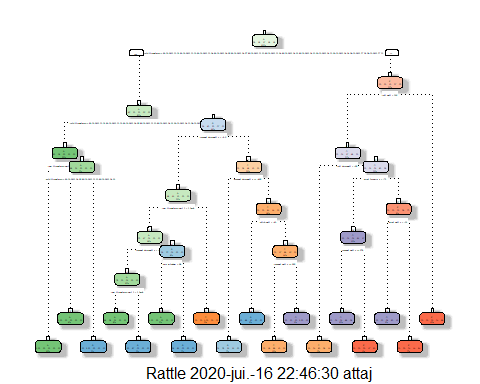
## Using ML algorithms for prediction: Decision Tree

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

To view the decision tree with fancy :

fancyRpartPlot(modFitA1)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting



## Predicting:

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

Using confusion Matrix to test results:

confusionMatrix(predictionsA1, myTesting$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2157 68 10 1 0  
## B 60 1265 73 67 0  
## C 15 177 1261 141 70  
## D 0 8 15 962 111  
## E 0 0 9 115 1261  
##   
## Overall Statistics  
##   
## Accuracy : 0.8802   
## 95% CI : (0.8728, 0.8873)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8484   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9664 0.8333 0.9218 0.7481 0.8745  
## Specificity 0.9859 0.9684 0.9378 0.9796 0.9806  
## Pos Pred Value 0.9647 0.8635 0.7578 0.8777 0.9105  
## Neg Pred Value 0.9866 0.9604 0.9827 0.9520 0.9720  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2749 0.1612 0.1607 0.1226 0.1607  
## Detection Prevalence 0.2850 0.1867 0.2121 0.1397 0.1765  
## Balanced Accuracy 0.9762 0.9009 0.9298 0.8638 0.9276

## Using ML algorithms for prediction: Random Forests

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

## Predicting:

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

Using confusion Matrix to test results:

confusionMatrix(predictionsB1, myTesting$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2232 5 0 0 0  
## B 0 1513 2 0 0  
## C 0 0 1361 6 0  
## D 0 0 5 1279 1  
## E 0 0 0 1 1441  
##   
## Overall Statistics  
##   
## Accuracy : 0.9975   
## 95% CI : (0.9961, 0.9984)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9968   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 1.0000 0.9967 0.9949 0.9946 0.9993  
## Specificity 0.9991 0.9997 0.9991 0.9991 0.9998  
## Pos Pred Value 0.9978 0.9987 0.9956 0.9953 0.9993  
## Neg Pred Value 1.0000 0.9992 0.9989 0.9989 0.9998  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2845 0.1928 0.1735 0.1630 0.1837  
## Detection Prevalence 0.2851 0.1931 0.1742 0.1638 0.1838  
## Balanced Accuracy 0.9996 0.9982 0.9970 0.9968 0.9996

**Random Forests yielded better Results.**

## 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")  
predictionsB2

## 1 2 3 41 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

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)