Peer Review 8

Henrique I. Neves

09/02/2021

Introduction

This document is presented as the peer review project for the course *Practical Machine Learning* from the Data Science Specialization on coursera. In this project, Machine Learning Algorithms will be applied on a dataset trying to create a predictive model. The process of creating the model and its error will be measured.

The Dataset

The original dataset consist of measurements on weigth lifiting exercises collected with Human Activity Recognition (HAR) wearable devices. The data was collected with 4 different devices in different portions of the body of the patient. Another measurement "Classe", describe if the execution of the exercise was corrected (A) or if there was a mistake (grouped in 4 different classes from B to E). In the model to be created, the value of 'Classe' will be the outcome sought to be predicted.

Collecting the data and initial analysis

Once the that is downloaded, we can open both the assigned Training and Test sets, together with the necessary packages:

```
library(easypackages)
libraries("ggplot2", "caret", "rpart", "rattle", "dpylr", "randomForest")
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'dpylr'
```

```
train_set <- read.csv(file = "./pml-training.csv"); test_set <- read.csv(file = "./pml-testing.csv");
dim(train_set)
```

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

It is possible to see that the training set has more than 19000 observations, which constitutes a large dataset for the model creation. Looking at the variable that is sought to be predicted:

```
summary(train_set$classe)
```

```
##    Length     Class      Mode
##     19622 character character
```

```
unique(train_set$classe)
```

```
## [1] "A" "B" "C" "D" "E"
```

It is of 'character' class and contain, as described, 5 different classifications. To avoid any possible problem in the future, this will be converted to factor class.

```
train_set$classe <- as.factor(train_set$classe)
str(train_set$classe)
```

```
##  Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(train_set$classe)
```

```
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

A quick look at the structure of the train_set shows that several columns are filled with 'NAs' or character variable full of empty cells:

For the sake of brevity, the code to get this result will be written, but the result will be hidden:

```
str(train_set); str(test_set)
```

Cleaning and formatting the data

Before the creation of any model, we need to clean the data from this incomplete variables. Although there are algorithms that can fill these gaps with a an educated guess, the command **colSums** shows that from 19622 observation, 19216 are nulls. Therefore, trying to fill these gaps will not be very reliable. So, this data will be removed instead.

```
train_set_clean <- train_set[,!colSums(is.na(train_set))>0]
test_set_clean <- test_set[,!colSums(is.na(test_set))>0]
dim(train_set_clean); dim(test_set_clean)
```

```
## [1] 19622    93
```

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

It is possible to see that, after removal of incomplete variables, the test_set retained less variables than the train set. Looking at the data, this difference corresponds to to "character" variables filled with empty characters, which is not on the test set. There is a need to further purify the train set.

However, the variable 'classe', which we will try to predict, is not on the test set. So we need to save it and add to the final train set.

```
#Saving the variable 'classe':
train_classe <- train_set_clean$classe
#Purifying the train set a little further:
train_set_cleaner <- train_set[, colnames(train_set) %in% colnames(test_set_clean)]
#And to finish, adding the variable 'classe' back:
train_set_cleaner <- cbind(train_set_cleaner, classe = train_classe)
```

The final step was to remove any variable that is not informative to the model, i. e. any variable that is not related to movement measurement. Luckily, these are all the first 7 variables on both data_sets:

```
train_set_cleaner <- train_set_cleaner[,8:60]
test_set_clean <- test_set_clean[,8:60]
```

Out of sample error and subsetting the dataset

As said before, the test_set does not have the 'classe' variable, which will be the predictor This generate a problem evaluating any model generated, specially when calculating the *Out of Sample* error.

A brief explanation: The goal of machine learning is to use data to create a model able, the most accurately possible, to predict an outcome. However, even the most advanced and perfected model is still subject of some mistakes and predict wrongly. The mistakes made by the model in the training set is called *In sample Error*. This error tend to be relatively small due the phenomenon called *Overfitting*. Overfitting happens when the model becomes very good to predict data within the training set, but performs poorly with new

data. This increases the *Out of Sample Error*. Overall, the Out of Sample Error is always bigger than the In Sample Error, however the desired model produce the lowest Out of Sample Error Possible.

Since the testing set do not have the "classe" variable, it is not possible to check the Out of Error Sample using it. So, to test the models the train set will be splitted to generate data to test the model with. Since there are more than 19000 observation, both sets will be large enough.

```
subsetting <- createDataPartition(y=train_set_cleaner$classe, p = 0.8, list = FALSE)
subset_train <- train_set_cleaner[subsetting,]
subset_test <- train_set_cleaner[-subsetting,]
```

Model Creation

Several models will be tested. Since we have several possible predictor, the first one will be a simple decision tree.

```
model1 <- train(classe ~., data = subset_train, method = "rpart")
accuracy_model1 <- confusionMatrix(subset_test$classe, predict(model1, subset_test))$overall[1]
accuracy_model1
```

```
##  Accuracy
## 0.5034412
```

The simplest model has an accuracy of less than 50%. To generate a better model, cross validation could be used, which is a resampling algorithm. However, cross validation is better suited for model creating with limitted data. This is not the case here. So, instead of using it, the algrithm of *Random Forest* will be used.

Ramdom Forest is a method that generates several trees choosing the samples at ramdon on each decision node for an specific tree. This algorithm applies a randomization method called **Bagging** that stands for *Bootstrap aggregation*. The code below generates the model, which will take sometime.

```
model2 <- train(classe ~., data = subset_train, method = "rf", trControl = trainControl(verboseIter = T
```

The accuracy of model2 created using the RandomForest is evaluated:

```
confusionMatrix(subset_test$classe, predict(model2, subset_test))$overall[1]
```

```
##  Accuracy
## 0.9989804
```

So, it is possible to see that the accuracy of this method is far superior than the one with only one tree. Now, to test for *In Sample Error* and *Out of Sample Error*.

```
#In Sample Error
pred_train <- predict(model2, subset_train)
subset_train$Right <- pred_train == subset_train$classe
table(pred_train, subset_train$classe)
```

```
##
## pred_train    A    B    C    D    E
##          A 4464    3    0    0    0
##          B    0 3035    1    0    0
##          C    0    0 2736    6    2
##          D    0    0    1 2567    0
##          E    0    0    0    0 2884
```

```
#Calculation of error
sum(subset_train$Right)/nrow(subset_train)
```

```
## [1] 0.9991719
```

So the model had 100% accuracy (an 0% error) within the train data, which might be a signal of overfitting.

```
#Out of Sample Error
pred_test <- predict(model2, subset_test)
subset_test$Right <- pred_test == subset_test$classe
table(pred_test, subset_test$classe)
```

```
##
## pred_test    A    B    C    D    E
##         A 1116    1    0    0    0
##         B    0  758    0    0    0
##         C    0    0  684    3    0
##         D    0    0    0  640    0
##         E    0    0    0    0  721
```

```
#Calculation of error
sum(subset_train$Right)/nrow(subset_train)
```

```
## [1] 0.9991719
```

Luckily, the model also generated a prediction with 0% Out of Sample Error. Which indicates that it is very good in predictions.

Applying the model to the test set

Finaly, we can apply the model2 created previously to the 20 cases on the test set:

```
answer <- predict(model2, test_set_clean)
final_table <- data.frame(ID = test_set_clean$problem_id, answer)
final_table
```

```
##    ID answer
## 1   1      B
## 2   2      A
## 3   3      B
## 4   4      A
## 5   5      A
## 6   6      E
## 7   7      D
## 8   8      B
## 9   9      A
## 10 10      A
## 11 11      B
## 12 12      C
## 13 13      B
## 14 14      A
## 15 15      E
## 16 16      E
## 17 17      A
## 18 18      B
## 19 19      B
## 20 20      B
```