Practical Machine Learning - Coursera Project

Gauthier le Courtois du Manoir

26 juillet 2016

Introduction

This project is for the Coursera MOOC "Practical Machine Learning" the Data Science specialization of the John Hopkins Bloomberg School of Public Health. With data from devices health tracker, we will create a model to predict the variable "classe" from the given data set :

- Class A : according to the specification
- Class B : Throwing the elbows to the front
- Class C : Lifting the dumbbell only halfway
- Class D : Lowering the bumbbel only halfway
- Class E : Throwing the hips to the front

Here is a brief explanation of the project :

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. 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. 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).

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Getting and Cleaning the data

Before fitting our models, we need to get and clean our data.

Getting the data

First, we get download the two CSV files in two variables "training" and "testing". We also set the seed to make the algorithm reproducible.

```r
library(caret)
library(randomForest)
library(rpart)
library(rpart.plot)
library(rattle)
library(e1071)
#Download the data
#Setting the working directory
setwd("C:/Users/SAMSUNG/Documents/R Working Directory/Machine Learning/")

if(!file.exists("./data")){dir.create("./data/")}
sourcefile_train<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
destfile_train<-"./data/pml-training.csv"
download.file(sourcefile_train, destfile_train)
sourcefile_test<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
destfile_test<-"./data/pml-testing.csv"
download.file(sourcefile_test, destfile_test)

#Load the data
training<-read.csv("./data/pml-training.csv", na.strings=c("NA",""),header=TRUE)
testing<-read.csv("./data/pml-testing.csv",na.strings=c("NA",""), header=TRUE)
#We set seed to make the results of the project reproducible
set.seed(8484)
#dim(training) 19622 160
#dim(testing) 20 160
```

The training data set has 19622 rows and 160 variables.

The testing dara set has 20 rows and 160 variables.

Cleaning the data

Now that we have the data, we will clean and prepare our data for training or model. First, we remove columns with near Zero Variance. Those variables won't have a lot of importance in helping our models to find the "classe"" parameter.

```r
########################################
#               STEP 1 : Clean 1
########################################
#Remove columns with not much variance
#View(nearZeroVar(training, saveMetrics=TRUE)) #To have a look in detail
nsv<-nearZeroVar(training) #
#length(nsv) #43
#We put away columns with not much variances : 43 columns
#and we create a new dataset
training.2<-training[,-nsv]
```

Once we have deleted 43 columns of our initial 160 columns data set, we now delete columns which have more that 50% NA as data

```r
########################################
#               STEP 2 : Clean 2
########################################
#Check NA Values and take only column with less than 0.5%
a<-0 #initializing variable
for (i in 1:dim(training.2)[2]){
  a[i]<-sum(is.na(training.2[,i]))
}
b<-dim(training.2)[1]
ratios_na<-a/b
#We take only column with number of NA <0.5
test_na<-ratios_na<0.5
training.3<-training.2[,test_na]
```

Now we also suppress the first column, the index so that it doesn't alter our models.

```
########################################
#               STEP 3 : Clean 3
########################################
#We remove the first column (index) so that it doesn't alter our models
trainingV2<-training.3[,-1]
#dim(training) #19622 Raw 160 Columns
#dim(trainingV2) #19622 Raw 58 Columns
#Our operation suppressed 102 dummy columns
```

At the end of the thirds step, we now have 58 columns. We succeded in suppressing 102 dummy variables with no or small value as a predictor. The final step in cleaning our data is splitting our training data in two independant set for cross-validation : one for real training (60%) and one for testing/validation (40%).

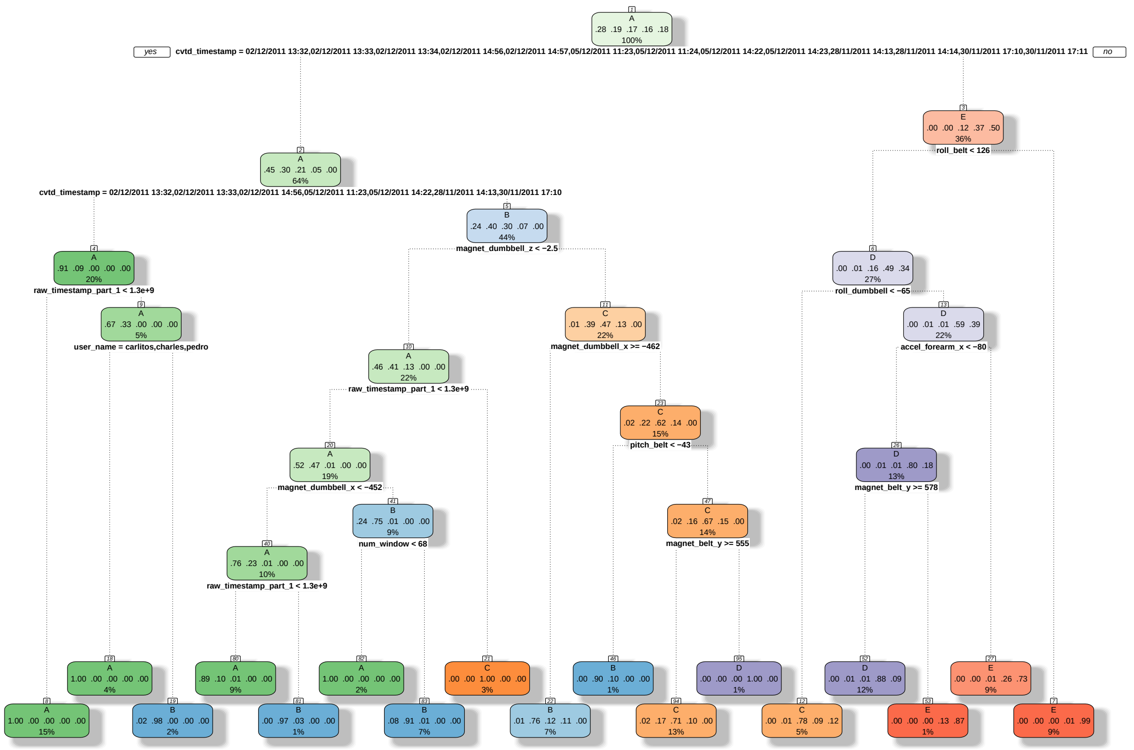
```
########################################
#               STEP 4 : New data set
########################################
#We split our training set in two sets, one four training and one for validation.
inTrain <- createDataPartition(y=trainingV2$classe, p=0.6, list=FALSE)
trainingV3 <- trainingV2[inTrain,]
validationV3 <- trainingV2[-inTrain,]

testingV3<-testing[colnames(trainingV3[,1:57])]
#We take for our final testing data set exactly the same columns as our training data set
levels(testingV3$cvtd_timestamp)<-levels(trainingV3$cvtd_timestamp)
```

Model creation : Model A - Rpart (Simple Tree)

Now that our data set is clean, we can create our first model : a simple classification tree.

the expected out of sample error is,

```
#We fit a model based on a simple tree and train it on our training set
#fit_rpart<-train(classe~., data=trainingV3, method="rpart")
#rpart gives better result than train
fit_rpart<-rpart(classe~., data=trainingV3, method="class")
#We plot the tree
par(mar=c(1,1,1,1))
fancyRpartPlot(fit_rpart)
```



```r
#We use our model to predict on our validation data set
predict_rpart<-predict(fit_rpart, newdata=validationV3,type='class')
confusionMatrix(predict_rpart,validationV3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2153   67    4    5    0
##          B   56 1256   79   64    0
##          C   23  185 1260  145   65
##          D    0   10   11  870   75
##          E    0    0   14  202 1302
##
## Overall Statistics
##
##                Accuracy : 0.8719
##                  95% CI : (0.8643, 0.8792)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8379
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9646   0.8274   0.9211   0.6765   0.9029
## Specificity            0.9865   0.9686   0.9355   0.9854   0.9663
## Pos Pred Value         0.9659   0.8632   0.7509   0.9006   0.8577
## Neg Pred Value         0.9859   0.9590   0.9825   0.9395   0.9779
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2744   0.1601   0.1606   0.1109   0.1659
## Detection Prevalence   0.2841   0.1854   0.2139   0.1231   0.1935
## Balanced Accuracy      0.9755   0.8980   0.9283   0.8309   0.9346
```

```
#Accuracy 0.8719% : good
```

To sum up the Overall Statitics for the model based on the simple classification tree :

- Accurracy : 0.8719
- 95 CI : (0.8643, 0.8792)
- P-Value < 2.2e-16
- Kappa : 0.8379

87% is good but maybe we can find another model with a higher accuracy rate.

Model creation : Model B - RandomForest (rf)

Our first model is based on a simple tree and have a score <95% in accuracy. As we learn in class, the two model oftenly used for machine learning, the two which gave the higher accuracy, are Boosting and Random Forest. Let's try now with the randomForest method and see if we can create a better model.

```
#We fit a model based on a simple tree and train it on our training set
fit_rf<-randomForest(classe~., data=trainingV3)
#randomForest faster than predict(fit_rf, newdata=validationV3)

#We use our model to predict on our testing set
predict_rf<-predict(fit_rf, newdata=validationV3, type='class')
confusionMatrix(predict_rf,validationV3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    1    0    0    0
##          B    0 1517    0    0    0
##          C    0    0 1367    1    0
##          D    0    0    1 1281    2
##          E    0    0    0    4 1440
##
## Overall Statistics
##
##                Accuracy : 0.9989
##                  95% CI : (0.9978, 0.9995)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9993   0.9993   0.9961   0.9986
## Specificity            0.9998   1.0000   0.9998   0.9995   0.9994
## Pos Pred Value         0.9996   1.0000   0.9993   0.9977   0.9972
## Neg Pred Value         1.0000   0.9998   0.9998   0.9992   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1933   0.1742   0.1633   0.1835
## Detection Prevalence   0.2846   0.1933   0.1744   0.1637   0.1840
## Balanced Accuracy      0.9999   0.9997   0.9996   0.9978   0.9990
```

```
#Accuracy  : 0.9986 Very good
```

To sum up the Overall Statitics for the model based on the random Forest algorithm :

- Accurracy : 0.9986
- 95 CI : (0.9978, 0.9995)
- P-Value < 2.2e-16
- Kappa : 0.9985

Conclusions

We can say that model based on random Forest gives better results that the one with unique classification tree.

```
predict_rf_real <-predict(fit_rf, newdata=testingV3, type='class')
predict_rf_real
```

```
##  1  2  3  4  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
```

```
#We create an output fil with the prediction pour the final testing data set.
write.csv(predict_rf_real,file='ouput_project.csv')
#1  2  3  4  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
```