Practical Machine Learning - Prediction Assignment

Carlos Barco

7/28/2017

Introduction

Data collection through self-monitoring and self-sensing combines wearable sensors (e.g.Electroencephalography, Electrocardiography) and wearable computing (smartwatchs, heart rate monitors, etc.) See Quantified self. Using wereable computers like known fitness accessories is now possible to collect a large amount of data about fitness personal activity in an inexpensively way. This trackers devices are part of a "life logging" movement, and a general characteristic of the information collected is that is known how much a particular activity was did it, but rarely quantified how well they did it.
In this project, the goal is 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: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Executive Resume

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.
The goal of this project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set.

Building the model

Reproducibility

Preparing the data and R packages

The following Libraries were used for this project, so you should install and load them in your own working environment.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
library(corrplot)
library(RColorBrewer)
set.seed(56789) #for reproducibility purposes
```

```
getRversion()
```

```
## [1] '3.4.0'
```

I choose randomForest (as a supervised learning algorithm), because of their known utility in classification problems, and gives an acceptable level of performance.

Getting Data

```
trainUrl <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
trainFile <- "/Users/carlosbarco/R/pml-training.csv"
testFile  <- "/Users/carlosbarco/R/pml-testing.csv"
if (!file.exists("./data")) {
  dir.create("./data")
}
if (!file.exists(trainFile)) {
  download.file(trainUrl, destfile=trainFile, method="curl")
}
if (!file.exists(testFile)) {
  download.file(testUrl, destfile=testFile, method="curl")
}
#Read the two data files into two independent data frames
trainRaw <- read.csv("/Users/carlosbarco/R/pml-training.csv", sep = ",")
testRaw <- read.csv("/Users/carlosbarco/R/pml-testing.csv", sep = ",")
dim(trainRaw)
```

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

```
dim(testRaw)
```

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

The raw training data has 19622 rows of observations and 160 features (predictors), inside the column X is an unusable row number. In raw test set exists 20 rows and the same 160 features. Inside this, column "classe" is the target outcome to predict.

Data Cleaning

Features require consistent data, and at the moment, there is so much useless values to be removed.
1) Reduce the number of predictors (activity monitors) by removing columns that have zero values, NA or are emptys (meaningless variables).

```
trainRaw <- trainRaw[, colSums(is.na(trainRaw)) == 0]
testRaw <- testRaw[, colSums(is.na(testRaw)) == 0]
```

2) Remove a few columns that are not useful for accelerometer function and contains NA´s: X (sequential number), user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, num_window.

```
classe <- trainRaw$classe
trainRemove <- grepl("^X|timestamp|window", names(trainRaw))
trainRaw <- trainRaw[, !trainRemove]
trainCleaned <- trainRaw[, sapply(trainRaw, is.numeric)]
trainCleaned$classe <- classe
testRemove <- grepl("^X|timestamp|window", names(testRaw))
testRaw <- testRaw[, !testRemove]
testCleaned <- testRaw[, sapply(testRaw, is.numeric)]
```

```
dim(trainCleaned)
```

```
## [1] 19622    53
```

```
dim(testCleaned)
```

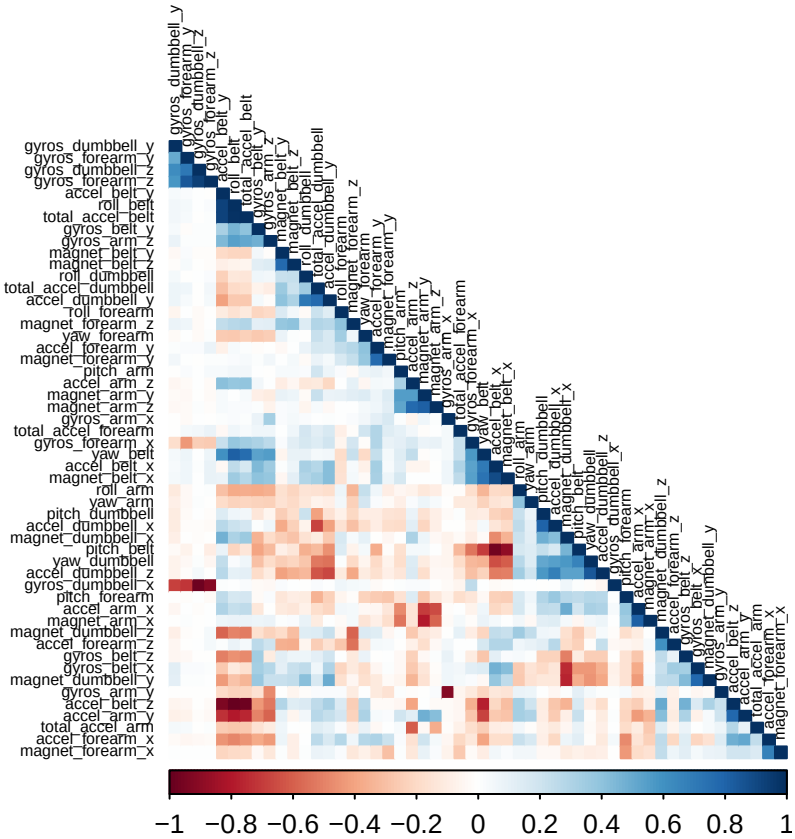
```
## [1] 20 53
```

With the cleaning process above, the number of variables for the analysis has been reduced to 53 only

Correlation Analysis in the Training Data Set

Plot a correlation matrix between features and outcomes, in order to see if they are orthogonal each others.

```
corMatrix <- cor(trainCleaned[, -53])
corrplot(corMatrix, order = "hclust", method = "color", type = "lower",
         tl.cex = 0.5, tl.col = rgb(0, 0, 0))
```



In darker are the most correlated variables. There is a not too high correlation between features.

Partitioning Training Set

Was achieved by splitting the training data into a training set (70%) and a validation set (30%) using the following:

```
set.seed(56789) # For reproducibile purpose
inTrain <- createDataPartition(trainCleaned$classe, p = 0.70, list = FALSE)
validation <- trainCleaned[-inTrain, ]
training <- trainCleaned[inTrain, ]
rm(inTrain)
```

The training data set consist of

```
dim(training)
```

```
## [1] 13737    53
```

The validation data set

```
dim(validation)
```

```
## [1] 5885   53
```

and the Testing Data of

```
dim(testCleaned)
```

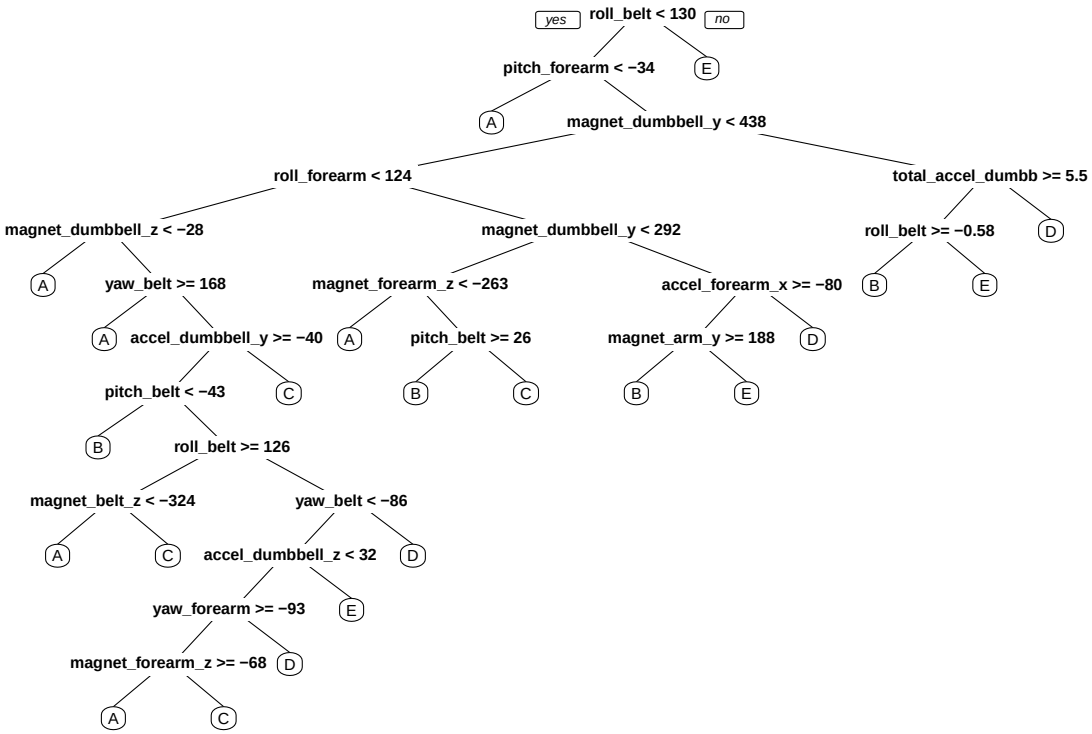
```
## [1] 20 53
```

Build Data Modeling

Decision Tree

Predictive model for activity recognition

```
modelTree <- rpart(classe ~ ., data = training, method = "class")
prp(modelTree)
```



Performance of the model on the validation data set

```
predictTree <- predict(modelTree, validation, type = "class")
confusionMatrix(validation$classe, predictTree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1490   38   47   67   32
##          B  240  592  127   84   96
##          C   20   61  853   65   27
##          D   85   26  143  639   71
##          E   43   62  136   65  776
##
## Overall Statistics
##
##                Accuracy : 0.7392
##                  95% CI : (0.7277, 0.7503)
##     No Information Rate : 0.3191
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.669
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7934   0.7599   0.6531   0.6946   0.7745
## Specificity            0.9541   0.8929   0.9622   0.9345   0.9373
## Pos Pred Value         0.8901   0.5198   0.8314   0.6629   0.7172
```

```
## Neg Pred Value         0.9079   0.9606   0.9068   0.9429   0.9529
## Prevalence             0.3191   0.1324   0.2219   0.1563   0.1703
## Detection Rate         0.2532   0.1006   0.1449   0.1086   0.1319
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.8737   0.8264   0.8077   0.8146   0.8559
```

```
accuracy <- postResample(predictTree, validation$classe)
ose <- 1 - as.numeric(confusionMatrix(validation$classe, predictTree)$overall[1])
rm(predictTree)
rm(modelTree)
```

Random Forest

Being the training size 70 % of total dataset, the bias can not be ignored and k=5 is reasonable [Choice of K in K-fold cross-validation]https://stats.stackexchange.com/questions/27730/choice-of-k-in-k-fold-cross-validation)

```
modelRF <- train(classe ~ ., data = training, method = "rf", trControl = trainControl(method = "cv", 5)
modelRF
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10988, 10991, 10990, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9922108  0.9901462
##   27    0.9906818  0.9882118
##   52    0.9863134  0.9826867
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

Now, we estimate the performance of the model on the validation data set.

```
predictRF <- predict(modelRF, validation)
confusionMatrix(validation$classe, predictRF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    1    0    0    0
##          B    6 1132    1    0    0
##          C    0    6 1020    0    0
##          D    0    0   18  943    3
##          E    0    0    0    4 1078
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9934
##                  95% CI : (0.991, 0.9953)
##     No Information Rate : 0.2853
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9916
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9939   0.9817   0.9958   0.9972
## Specificity            0.9998   0.9985   0.9988   0.9957   0.9992
## Pos Pred Value         0.9994   0.9939   0.9942   0.9782   0.9963
## Neg Pred Value         0.9986   0.9985   0.9961   0.9992   0.9994
## Prevalence             0.2853   0.1935   0.1766   0.1609   0.1837
## Detection Rate         0.2843   0.1924   0.1733   0.1602   0.1832
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9981   0.9962   0.9902   0.9958   0.9982
```

```
accuracy <- postResample(predictRF, validation$classe)
ose <- 1 - as.numeric(confusionMatrix(validation$classe, predictRF)$overall[1])
rm(predictRF)
```

Evaluate the model (out-of-Sample Error)

Now, we apply the Random Forest model to the original testing data set downloaded from the data source. We remove the problem_id column first.

```
rm(accuracy)
rm(ose)
predict(modelRF, testCleaned[, -length(names(testCleaned))])
```

```
##  [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
```

Creates submission files

```
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)
    }
}
```