Course Project - Practical Machine Learning

Juan M Hernandez

27 de agosto de 2016

Synopsis

This human activity recognition research has traditionally focused on discriminating between different activities, i.e. to predict "which" activity was performed at a specific point in time.

The data is from six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five 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) and 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.

The data for this project come from this source: [http://groupware.les.inf.puc-rio.br/har].

Libraries Needed

For the purpose of this paper, we need to load the following libraries.

```
library(caret)
library(rpart)
library(MASS)
library(randomForest)
library(klaR)
```

Data Load

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]

When loading the data, we set the na.string to identify which data comes in that way.

```
#download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "pml-
#download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "pml-t

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

Data Cleaning

Knowing that data comes with a high number of NA values in some columns, we leave out those columns from our training data. After that, we check that our data have complete cases.

```
c1 <- as.data.frame(lapply(dataTraining, function(x) sum(is.na(x))/length(x) ))
dataTrainingNoNA <- dataTraining[,which(colMeans(c1) < 0.5)]
sum(complete.cases(dataTrainingNoNA)== FALSE)
```

```
## [1] 0
```

Getting data for training and testing

Having our data ready, we create from the variable #dataTraining# two partitions, one with the 60% of the data for training the model, and the 40% left for testing the models created.

```
inTrain <- createDataPartition(y = dataTrainingNoNA$classe, p = 0.6, list = F)
training <- dataTrainingNoNA[inTrain, ]
testing <- dataTrainingNoNA[-inTrain, ]
```

Preparing data for training

One step more before we start training the models. Due to the number of columns, there are some columns that are highly correlated, and that could affect our model process in two ways, time processing values that may not be necessary as predictors, and we are certain that the predictors left are the one we need to create our model.

```
colNumeric <- data.frame(x = names(dataTrainingNoNA), y = sapply(training, class))
colNumeric$x <- as.character(colNumeric$x)
colNumeric$y <- as.character(colNumeric$y)

colSelected <- colNumeric[colNumeric$y %in% c("integer", "numeric") & colNumeric$x != "X",]$x

dataTrainingNoNA.colSelected <- dataTrainingNoNA[, colSelected]

correlationMatrix <- cor(dataTrainingNoNA.colSelected)
highlyCorrelatedColumns <- findCorrelation(correlationMatrix, cutoff=0.75)

training.NoHighCorrelated <- dataTrainingNoNA.colSelected[, -highlyCorrelatedColumns]
training.NoHighCorrelated$classe <- dataTrainingNoNA$classe

str(training.NoHighCorrelated)
```

```
## 'data.frame':    19622 obs. of  34 variables:
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1323
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434
##  $ num_window          : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
```

```
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Model Selection

Now we are ready for creating some models. I have selected 4 models to process the data, some with preProcessing, and other with crossValidation.

*I recommend to use the library #beepr# when running this models, to get a sound when it's finished, now that for instance the model 4 took like 30 minutes to process.

```
set.seed(12345)
```

```
modFit <- train(classe ~ ., data = training.NoHighCorrelated, method="rpart"
                , preProcess = c("center", "scale"))
```

```
modFit2 <- train(classe ~ ., data = training.NoHighCorrelated, method="lda")
```

```
modFit3 <- train(classe ~ ., method = "rf", data = training.NoHighCorrelated
                 , trControl = trainControl(method = "cv"), number = 4)
```

Testing our models

Once we have our models done, we need to test those with our 40% (#variable 'testing'#) of our data stored in the variable dataTraining, to see how accurate is each model.

```
pr1 <- predict(modFit, newdata = testing)
pr2 <- predict(modFit2, newdata = testing)
pr3 <- predict(modFit3, newdata = testing)
```

Results

```
confusionMatrix(pr1, testing$classe)
```

Rpart Method

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1705  126    0   47    1
##          B  116  878  100  282  362
##          C  318  459 1203  483  550
##          D   93   54   65  356   60
##          E    0    1    0  118  469
##
## Overall Statistics
##
##                Accuracy : 0.5877
##                  95% CI : (0.5767, 0.5986)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4815
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7639   0.5784   0.8794  0.27683  0.32524
## Specificity            0.9690   0.8641   0.7206  0.95854  0.98142
## Pos Pred Value         0.9074   0.5052   0.3993  0.56688  0.79762
## Neg Pred Value         0.9117   0.8952   0.9659  0.87116  0.86594
## Prevalence             0.2845   0.1935   0.1744  0.16391  0.18379
## Detection Rate         0.2173   0.1119   0.1533  0.04537  0.05978
## Detection Prevalence   0.2395   0.2215   0.3840  0.08004  0.07494
## Balanced Accuracy      0.8664   0.7212   0.8000  0.61768  0.65333
```

```
confusionMatrix(pr2, testing$classe)
```

LDA Method

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1556  269  315  103  177
##          B  155  768  136  102  262
##          C  228  247  739  209  153
##          D  258  131  155  753  170
##          E   35  103   23  119  680
##
## Overall Statistics
##
##                Accuracy : 0.573
##                  95% CI : (0.562, 0.584)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4589
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6971  0.50593  0.54020  0.58554  0.47157
## Specificity            0.8461  0.89649  0.87079  0.89116  0.95628
## Pos Pred Value         0.6430  0.53970  0.46891  0.51329  0.70833
## Neg Pred Value         0.8754  0.88323  0.89968  0.91644  0.88934
## Prevalence             0.2845  0.19347  0.17436  0.16391  0.18379
## Detection Rate         0.1983  0.09788  0.09419  0.09597  0.08667
## Detection Prevalence   0.3084  0.18137  0.20087  0.18697  0.12236
## Balanced Accuracy      0.7716  0.70121  0.70550  0.73835  0.71392
```

```
confusionMatrix(pr3, testing$classe)
```

Random Forest Method

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    0 1518    0    0    0
##          C    0    0 1368    0    0
##          D    0    0    0 1286    0
##          E    0    0    0    0 1442
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9995, 1)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Conclusion

Comparing the accuracy of each model process, we see in the results that "rpart"" had a 58.7% of accuracy, where it's been the less accurate of the models, having "linear discriminant analysis" with 57.3% and "random forest" with 100%.

Final Testing Result

As a final test, I run the prediction with the random forest model on the dataTesting, and here are the results.

```
predict(modFit3, newdata = dataTesting)
```

```
##  [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
```