Human Activity Recognition

Weight Lifting Exercises Dataset

Eddy Delta

05/02/2019

Introduction

Context

Research on activity recognition has traditionally focused on discriminating between different activities, i.e. to predict *"which"* activity was performed at a specific point in time. The quality of executing an activity, the *"how (well)"*, has only received little attention so far, even though it potentially provides useful information for a large variety of applications. In this work we define quality of execution and investigate three aspects that pertain to qualitative activity recognition: specifying correct execution, detecting execution mistakes, providing feedback on the to the user.

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 data for this project come from this study: http://groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

The following links provides the informations about the authors and Documentation about the original study.

Data

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

The 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, Read more.

Exploratory data analysis

```
load_data <- function(file_url, data_dir = 'data', cache = TRUE){
    if(cache){
        if(!dir.exists(data_dir)) dir.create(data_dir)
        file_path <- file.path(data_dir, basename(file_url))
        if(!file.exists(file_path)) download.file(file_url, file_path, quiet = TRUE)
```

```
        if(!file.exists(file_path)) stop('Unable to download file ', file_url)
        # clean the dummy values like '', 'NA', '#DIV/0!'
        read.csv(file_path, na.strings = c('', 'NA', '#DIV/0!'))
        # load the data model from the url
    } else read.csv(url(file_url), na.strings = c('', 'NA', '#DIV/0!'))
    # NOTE : if new / transformed columns are needed
    # transform(
    #     read.csv(file_path, na.strings = c('', 'NA', '#DIV/0!')),
    #     raw_timestamp = as.POSIXct(as.integer(as.numeric(as.character(raw_timestamp_part_1))), origin
    #     cvtd_timestamp = strptime(cvtd_timestamp, format = '%d/%m/%Y %H:%M')
    # )
}

pml_training <- load_data('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv')
dim(pml_training)
```

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

A quick look at the data highlight few points.

```
set.seed(9999)

# hist(as.numeric(pml_training$classe), main = 'Histogram of classes', xlab = 'classe')
# aggregate(training$classe, list(classe = training$classe), length)
table(pml_training$classe)
```

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

The testing data doesn't contains a `classe` variable to be compared with. The `classe` elements seems pretty well balanced across the training data set although class `A` is over represented. Note that class `A` corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

Many missing values

```
# sapply(pml_training[, sapply(pml_training, function(x) sum(is.na(x)) < nrow(pml_training) * 0.95)], f
high_na <- sapply(pml_training, function(x) sum(is.na(x)) / nrow(pml_training))
highest_na <- high_na[high_na >= 0.05]
```

An exploratory of the variables values shows that many variables contains a high ratio of missing values. 62.5% of the variables filtered on a $rate > 0.05$ contains at least 97.93% of `NA`s values.

This rate is too high to impute data in the training data set with confidence.

Build tidy datas set

```
names(pml_training[, grep('num_window|belt|forearm|arm|dumbbell|classe', names(pml_training), invert = T
```

```
## [1] "X"                    "user_name"            "raw_timestamp_part_1"
## [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
```

```
library(caret)
tidy_data <- function(raw_data, treshold = 0.95){
```

```
    # NOTE : select predictors minus variables where the value is nearly the same
    tidy_predictor <- raw_data[, setdiff(grep('num_window|belt|forearm|arm|dumbbell|classe', names(raw_
    # NOTE : keep column if NA ratio < 0.95
    ceil <- nrow(tidy_predictor) * treshold
    # tidy_predictor[, sapply(tidy_predictor, function(x) sum(is.na(x)) < ceil)]
    tidy_predictor[, colSums(is.na(tidy_predictor)) < ceil]
}

tidy_training  <- tidy_data(pml_training)
classe_col <- which(colnames(tidy_training) == 'classe')
dim(tidy_training)
```

```
## [1] 19622    54
```

The predictors have been selected because of the naming related to the sensor types, plus the num_windows which after some digging help to group the records.

As this is a classification prediction problem, I decided to try 2 appoaches : 1. Random Forest 2. Regression Trees

The training dataset have been splited in half, one part for each training type. Then each subset is separated between a training data set and a validation dataset.

```
inBuild <- createFolds(y = tidy_training$classe, k = 2)
training <- tidy_training[inBuild[[1]], ]
validation <- tidy_training[-inBuild[[2]], ]

dim(training)
```

```
## [1] 9811   54
```

```
dim(validation)
```

```
## [1] 9811   54
```

Analysis

Random Forest

I'll first start with the random forest as it usually provides more accurate predictions with a cost of more computation power.

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

```
library(ggplot2)
library(randomForest)
# library(beepr)
modRf <- train(classe ~ ., data = training, method = 'rf', prox = TRUE, trControl = trainControl(method
# No k-mean cross validation, Bootstrap resampling method 1 resampling iteration
# Optimal setting, but overkill in the present case
# modRf <- train(classe ~ ., data = training, method = 'rf', prox = TRUE, trControl = trainControl(meth
# beep()
modRf
```

```
## Random Forest
##
## 9811 samples
##   53 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8830, 8830, 8829, 8828, 8830, 8830, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9913360  0.9890393
##   27    0.9953111  0.9940686
##   53    0.9932720  0.9914887
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
# modRf$resample
varImp(modRf)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 53)
##
##                      Overall
## num_window           100.000
## roll_belt             67.403
## pitch_forearm         43.461
## yaw_belt              36.739
## magnet_dumbbell_y     32.636
## magnet_dumbbell_z     32.409
## pitch_belt            30.479
## roll_forearm          23.247
## accel_dumbbell_y      13.844
## roll_dumbbell         12.237
## accel_forearm_x       11.594
## magnet_dumbbell_x     11.444
## accel_belt_z          10.673
## magnet_belt_y          9.847
## total_accel_dumbbell   9.666
## accel_dumbbell_z       9.133
## magnet_belt_z          8.442
## magnet_forearm_z       7.254
## magnet_belt_x          6.234
## accel_forearm_z        5.629
```

```
confusionMatrix.train(modRf)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 28.4  0.1  0.0  0.0  0.0
##          B  0.0 19.2  0.1  0.0  0.0
##          C  0.0  0.0 17.4  0.2  0.0
##          D  0.0  0.0  0.0 16.2  0.1
##          E  0.0  0.0  0.0  0.0 18.3
##
##  Accuracy (average) : 0.9953
```

```
predRf <- predict(modRf, validation)
cmRf <- confusionMatrix(predRf, validation$classe)
cmRf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2790    0    0    0    0
##          B    0 1898    0    0    0
##          C    0    0 1711    0    0
##          D    0    0    0 1608    0
##          E    0    0    0    0 1804
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9996, 1)
##     No Information Rate : 0.2844
##     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             0.2844   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2844   0.1935   0.1744   0.1639   0.1839
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1839
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

```
# Print out model summary
# print(modRf$finalModel, digits=3)
```

As expected the Random forest prediction accuracy (1) is very high, which makes the outcome quite reliable.

Decision tree

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).

The second prediction model used to predict the classes is a Regression tree. It worth noting that the default setting for the `caret` package is quite weak with this data set. The `rpart` function default parameters provide a better prediction. This prediction can be reach by tweaking the caret params though.

```
library(rpart)
# NOTE : this result can also be obtain using
# modRpart <- rpart(classe ~ ., data = training, method = 'class')
# predRpart <- predict(modRpart, validation, type = 'class')
# Here to get more accute outcome trainControl need to be disabled and cp set to 0.01
# It's a case where caret default optimisation are actually performing worst than rpart basic setup
modRpart <- train(classe ~ ., method = 'rpart', data = training, trControl = trainControl(method = 'none
modRpart
```

```
## CART
##
## 9811 samples
##   53 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: None
```

```
predRpart <- predict(modRpart, validation)
cmRpart <- confusionMatrix(predRpart, validation$classe)
cmRpart
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2745   28    6    6    9
##          B   18 1823   28   17   11
##          C   16   27 1650   30    9
##          D    5    8   25 1538   17
##          E    6   12    2   17 1758
##
## Overall Statistics
##
##                Accuracy : 0.9697
##                  95% CI : (0.9661, 0.973)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9617
##  Mcnemar's Test P-Value : 0.111
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9839   0.9605   0.9643   0.9565   0.9745
## Specificity            0.9930   0.9906   0.9899   0.9933   0.9954
## Pos Pred Value         0.9825   0.9610   0.9527   0.9655   0.9794
## Neg Pred Value         0.9936   0.9905   0.9924   0.9915   0.9943
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2798   0.1858   0.1682   0.1568   0.1792
```

```
## Detection Prevalence    0.2848    0.1934    0.1765    0.1624    0.1830
## Balanced Accuracy       0.9884    0.9756    0.9771    0.9749    0.9849
```

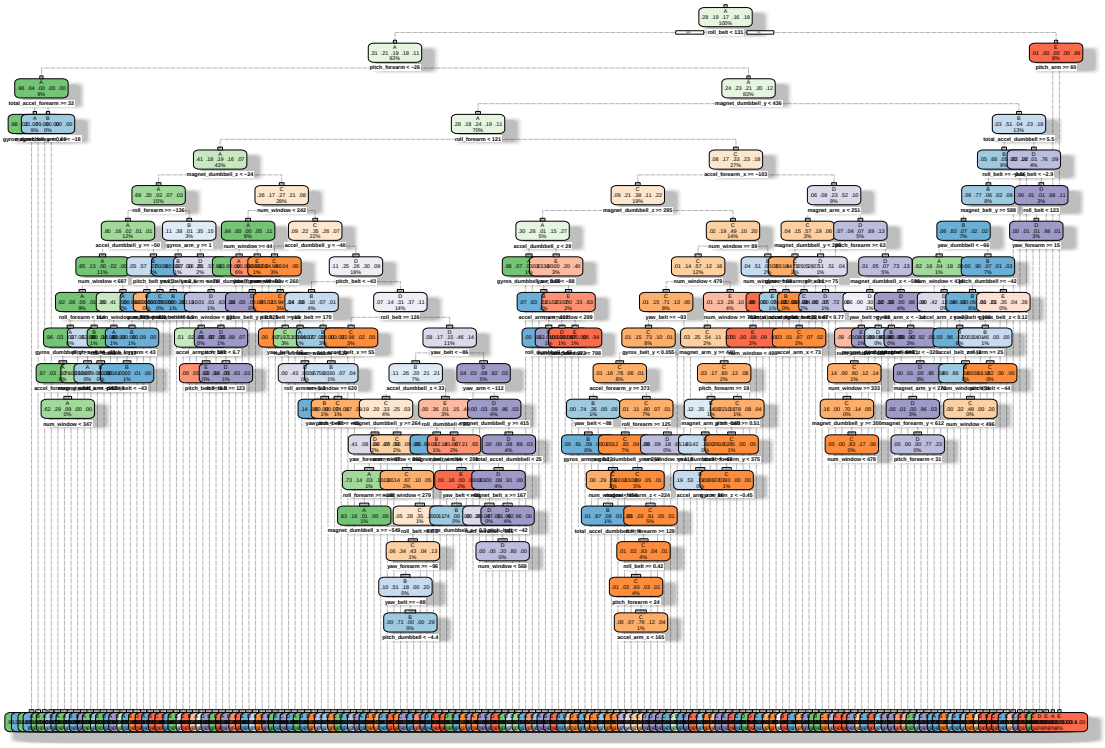
The Decision tree prediction accuracy (0.96973) is weaker than the Random forest prediction, even if it performs above the average.

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

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

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



Rattle 2019−Feb−13 11:48:18 edelta

Test cases predictions

The trained machine learning algorithm generated above are now used to predict 20 test cases.

```
pml_testing <- load_data('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv')
dim(pml_testing)
```

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

```
# variables in training not present in testing
setdiff(names(pml_training), names(pml_testing))
```

```
## [1] "classe"
```

```
# variables in testing not present in training
setdiff(names(pml_testing), names(pml_training))
```

```
## [1] "problem_id"
```

```
testing <- tidy_data(pml_testing)
testing <- testing[, intersect(names(testing), names(tidy_training))]
dim(testing)
```

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

```
testingRf <- predict(modRf, testing)
testingRf
```

```
##  [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
```

```
testingRpartPred <- predict(modRpart$finalModel, testing)
testingRpart <- factor(colnames(testingRpartPred)[apply(testingRpartPred, 1, function(x) which.max(x))])
testingRpart
```

```
##  [1] C A A A A E D A A A E C B A E E A B B B
## Levels: A B C D E
```

```
table(testingRpart, testingRf)
```

```
##             testingRf
## testingRpart A B C D E
##            A 7 2 0 0 0
##            B 0 4 0 0 0
##            C 0 1 1 0 0
##            D 0 0 0 1 0
##            E 0 1 0 0 3
```

There is a 80% of prediction matching between the 2 models.

Conclusion

The Random Forest prediction model is a clear winner to predict the excercice `classe` out of the available predictors.

The Random forest prediction is very sensitive to the settings parameters used to train the algorithm. It produce a decent prediction though, but not good enough to par with the Random forest predictions.