Practical Machine Learning - Course Final Project

Alberto A. Caeiro Jr

February 18, 2015

Executive Summary & Background

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.

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. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

About the Data

The data for this project are available here: . Training dataset : "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)" . Testing dataset: "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)" The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (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. So let's have a look on the dataset and on the classe variable.

Data Processing

Obtaining and Cleaning the data

```
# Enabling Multi Core for modeling processing
library(doMC)
  registerDoMC(cores = 2)

#Loading used libraries
library(caret);library(klaR);  library(rpart)
library(randomForest); library(gbm)

#setting the seed for reproducible computation
set.seed(12345)

#setting the working directory folder
setwd("~/Developer/Data Science Specialization/Practical Machine Learning/Project")

# loading both testing and training dataset (considering both files were already downloaded)
trainFile <- "./pml-training.csv"
training <- read.csv(file=trainFile, header=TRUE, sep=",", na.strings=c("NA","#DIV/0!",""))
testFile <- "./pml-testing.csv"
testing <- read.csv(file=testFile, header=TRUE, sep=",", na.strings=c("NA","#DIV/0!",""))

# Summary for the training predictors and outcome
str(training)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1323084232 1323084232 13
23084232 1323084232 ...
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434 ...
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ 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 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_belt.1    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ 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 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
```

```
##  $ magnet_belt_z         : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ 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 ...
##  $ var_accel_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x           : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ 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 ...
##  $ accel_arm_x           : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y           : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z           : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x          : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y          : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z          : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_roll_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm           : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
```

```
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ 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 ...
##  $ kurtosis_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

As one can see in the “str(training)”, the dataset has a lot of NAs. Let’s first clean the data basically removing the NAs, the IDs and the zero variability columns.

```r
# Starting the Cleaning Process
nzvCol <- nearZeroVar(training)
training <- training[,-nzvCol]

# Since we have lots of variables, remove any with NA's or have empty strings, and the one's that are not predictors variables
filterData <- function(idf) {
    idx.keep <- !sapply(idf, function(x) any(is.na(x)))
    idf <- idf[, idx.keep]
    idx.keep <- !sapply(idf, function(x) any(x==""))
    idf <- idf[, idx.keep]

    # Remove the columns that aren't the predictor variables
    col.rm <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2",
                "cvtd_timestamp", "new_window", "num_window")
    idx.rm <- which(colnames(idf) %in% col.rm)
    idf <- idf[, -idx.rm]
    return(idf)
}

training <- filterData(training)
finalTrainingDS <- training
dim(finalTrainingDS)
```

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

```r
# Now let's perform the same cleaning process to the testing dataset as well
nzvCol <- nearZeroVar(testing)
testing <- testing[,-nzvCol]
testing <- filterData(testing)
finalTestingDS <- testing
dim(finalTestingDS)
```

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

Data Partitioning

Now we'll partition the data into training and testing datasets.

```
inTrain <- createDataPartition(y=finalTrainingDS$classe, p=0.85, list=FALSE)
newTraining <- finalTrainingDS[inTrain, ]
newTesting <- finalTrainingDS[-inTrain, ]
dim(newTraining); dim(newTesting)
```

```
## [1] 16680    53
```

```
## [1] 2942   53
```

Model Selection

We'll run some simulations with different machine learning algoritms. We'll use Random Forest, Decision Trees, Naive Bayes, Linear Discrimant Analysis and Generalized Boosted Regression Model. Besides this we'll check if using Principal Component Analysis also generates a good basis for prediction.

Note: from the referenced paper we know the AdaBoost algo yields something better than 99.5% accuracy. For this work we'll consider "good" anything higher than 98%.

Training Control Parameters

```
#Some fitting params - Repeated 5 Cross Validations
fitControl <- trainControl(method="repeatedcv", number=5, repeats=1, verboseIter=FALSE)
```

Predicting models with PCA pre-processing

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1156
##      2        1.5376             nan     0.1000    0.0801
##      3        1.4873             nan     0.1000    0.0828
##      4        1.4358             nan     0.1000    0.0559
##      5        1.3994             nan     0.1000    0.0562
##      6        1.3642             nan     0.1000    0.0432
##      7        1.3357             nan     0.1000    0.0422
##      8        1.3091             nan     0.1000    0.0360
##      9        1.2857             nan     0.1000    0.0321
##     10        1.2651             nan     0.1000    0.0361
##     20        1.1047             nan     0.1000    0.0178
##     40        0.9307             nan     0.1000    0.0092
##     60        0.8290             nan     0.1000    0.0057
##     80        0.7507             nan     0.1000    0.0043
##    100        0.6865             nan     0.1000    0.0034
##    120        0.6333             nan     0.1000    0.0020
##    140        0.5887             nan     0.1000    0.0032
##    150        0.5666             nan     0.1000    0.0022
```

Predicting Models without PCA

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2331
##      2        1.4621             nan     0.1000    0.1616
##      3        1.3604             nan     0.1000    0.1254
##      4        1.2810             nan     0.1000    0.1118
##      5        1.2118             nan     0.1000    0.0919
##      6        1.1536             nan     0.1000    0.0734
##      7        1.1071             nan     0.1000    0.0591
##      8        1.0689             nan     0.1000    0.0663
##      9        1.0283             nan     0.1000    0.0505
##     10        0.9964             nan     0.1000    0.0565
##     20        0.7627             nan     0.1000    0.0241
##     40        0.5382             nan     0.1000    0.0106
##     60        0.4136             nan     0.1000    0.0064
##     80        0.3320             nan     0.1000    0.0043
##    100        0.2707             nan     0.1000    0.0036
##    120        0.2275             nan     0.1000    0.0023
##    140        0.1923             nan     0.1000    0.0019
##    150        0.1781             nan     0.1000    0.0029
```

For the sake of comparison, we can see the overall indicadors for each prediction model in the table below:

```
# Analyzing the results
resultingDataTable <- rbind(cm_tree$overall, cm_tree_pca$overall, cm_lda$overall, cm_lda_pca$overall,
                      cm_nb$overall, cm_nb_pca$overall, cm_rf$overall, cm_rf_pca$overall,
                      cm_gbm$overall, cm_gbm_pca$overall)
rownames(resultingDataTable) <- c("Tree", "Tree w/ PCA", "LDA", "LDA w/ PCA",
                            "Naive Baeyes", "Naive Bayes w/ PCA", "Random Forest", "Random Forest w/ PCA",
                            "GBM", "GBM w/ PCA")

resultingDataTable
```

```
##                         Accuracy     Kappa AccuracyLower AccuracyUpper
## Tree                   0.5033990 0.3516115     0.4851663     0.5216250
## Tree w/ PCA            0.3793338 0.1669767     0.3617577     0.3971502
## LDA                    0.7046227 0.6257193     0.6877698     0.7210677
## LDA w/ PCA             0.5275323 0.4010752     0.5093025     0.5457073
## Naive Baeyes           0.7474507 0.6766211     0.7313398     0.7630681
## Naive Bayes w/ PCA     0.6451394 0.5507213     0.6275423     0.6624473
## Random Forest          0.9955812 0.9944106     0.9924556     0.9976452
## Random Forest w/ PCA 0.9816451 0.9767775     0.9761180     0.9861819
## GBM                    0.9629504 0.9531219     0.9554790     0.9694816
## GBM w/ PCA             0.8314072 0.7864595     0.8173808     0.8447711
##                      AccuracyNull AccuracyPValue McnemarPValue
## Tree                    0.5105370   7.861069e-01           NaN
## Tree w/ PCA             0.7396329   1.000000e+00           NaN
## LDA                     0.3031951   0.000000e+00  1.237497e-32
## LDA w/ PCA              0.3072740  1.260017e-135  3.489996e-38
## Naive Baeyes            0.3769545   0.000000e+00  4.565690e-49
## Naive Bayes w/ PCA      0.2970768   0.000000e+00  7.462410e-09
## Random Forest           0.2848402   0.000000e+00           NaN
## Random Forest w/ PCA    0.2865398   0.000000e+00           NaN
## GBM                     0.2872196   0.000000e+00  1.097292e-02
## GBM w/ PCA              0.2984364   0.000000e+00  9.194940e-14
```

Out-Of-Sample Error

Our out-of-sample error can be found using the (1 - Testing Accurary), as evaluated below (for Random Forest Algo).

```
oos_error <- 1 - cm_rf$overall[1]
print(paste("Out of Error Sample is: ", oos_error * 100, "%"))
```

```
## [1] "Out of Error Sample is:  0.441876274643105 %"
```

Conclusion

From the resulting table we can see the Random Forest algorithm yields a better result, and that using Principal Component Analysis, with a threshold of 99% of the variance would descrease of accuracy in about (aprox) 2% points. Besides this, comparing our results with the original paper results, we can see we have reach a very good prediction accuracy using Random Forest algorithm.

Appendix

Variable Importance

Just for sake of curiosity, lets check each variable importance, in the final model (random forest)

```
varImp(modelFitRF)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 52)
##
##                   Overall
## roll_belt          100.00
## yaw_belt            82.38
## magnet_dumbbell_z   70.10
## pitch_belt          66.28
## pitch_forearm       64.12
## magnet_dumbbell_y   61.37
## magnet_dumbbell_x   55.51
## roll_forearm        52.88
## accel_belt_z        47.39
## accel_dumbbell_y    44.42
## magnet_belt_z       43.88
## magnet_belt_y       43.05
## roll_dumbbell       40.15
## accel_dumbbell_z    36.47
## roll_arm            36.24
## accel_forearm_x     32.25
## accel_dumbbell_x    30.68
## yaw_dumbbell        30.04
## gyros_dumbbell_y    29.28
## magnet_forearm_z    28.86
```

Generating files to submitt

Now we'll use the original testing dataset and the best model for predicting the values.

```r
# This uses the code supplied by the class instructions
answers <- predict(modelFitRF, newdata=testing)
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)
    }
}
pml_write_files(answers)
```